Extracting general task structures to accelerate the learning of new tasks*

Ilya Dianov, Karinne Ramirez-Amaro, Pablo Lanillos, Emmanuel Dean-Leon,
Florian Bergner and Gordon Cheng

***Abstract*— Teaching a robot new tasks through kinesthetic demonstrations can be a long and complicated process. For example, a human has to demonstrate a new "pick and place" task each time the object or the target location has changed. However, obtaining the abstract representation of such task can significantly reduce the learning time as the human only has to teach the necessary parameters required for the successful execution, e.g. the location of an object. In this work, we present a framework which allows to extract general task structures which together with the obtained knowledge can improve and accelerate the teaching of new tasks. Additionally, our framework exploits the semantic similarities between task parameters in order to infer the possible structure of unknown tasks. Our proposed method utilises symbolic representations of tasks combined with an ontology which makes it applicable to different environments in various domains. We analysed our framework in an orange sorting scenario and a cleaning scenario to demonstrate that it allows reducing the time required for teaching from 136.3 to 53 seconds (61.12%) and from 48.7 to 21 seconds (56.87%) respectively compared to learning only by kinesthetic demonstrations.**

I. INTRODUCTION

There are several mechanisms which allow robot to execute different tasks: each step of the task can be manually programmed by the human [1]; a planning algorithm can be used to generate tasks from a library of actions with known preconditions and effects [2]; or a learning from demonstrations can be used to acquire optimal task execution models [3]. However, with increasing complexity of the robot environment the acquisition of the new tasks can be a very time consuming process as a robot has to obtain a new task every time its environment changes. In order to improve and accelerate task acquisition, the robot should be able to extract abstract representations of tasks which are applicable in different scenarios and adaptable to changes in the environment.

For example, assume that a human is teaching a robot the task of picking an orange and placing it into the box. Such task consists of the following steps: *Reach* an orange, *Take* the orange, *Put* the orange into a box and *Release* the orange. Assume that the person wants the robot to execute the *pick and place* task, but instead of placing the orange in the box, he needs to place it in the trash, which is a different location than the box. Therefore, the human is required to teach the new steps: *Reach* an orange, *Take* the orange, *Put* the orange into the trash and *Release* the orange. However, during the second demonstration the human repeats the same steps from the previous task, which makes it very redundant. Then, to improve the teaching process, the robot should extract the general structure of the *pick and place* task from the first demonstration (Fig. 1) and use it to learn the seconds task, where the human has only to specify a new location for the *Put* activity. Additionally, the obtained structure can be reused to accelerate the learning process of other tasks such as *plate cleaning* which consists of *picking* a sponge, *placing* it on top of the plate and then *Wiping* the plate. The human should only demonstrate the new task of *Wiping* the plate, and the robot can utilise its reasoning to infer the rest of the steps using knowledge from the previous *pick and place* tasks.

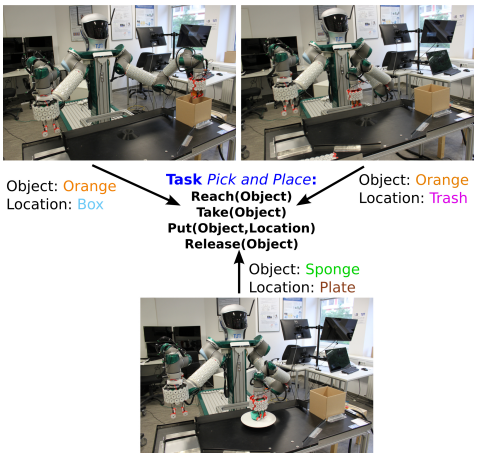


Fig. 1: Example of a common structure in different tasks.

The main contribution of this paper is a new method to teach robots new tasks in a fast and efficient manner by extracting key structures from the demonstrated tasks and utilising contextual knowledge[1] from previous experience. The extracted structure is represented as a directed graph, which allows to capture important relationships between task components and simplifies the search of known tasks. Additionally, we connect the obtained task graph with an ontology to enhance the generalization of our method to make it applicable across different domains.

*This work has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement no. 609206

Faculty of Electrical Engineering, Institute for Cognitive Systems, Technical University of Munich, Germany {ilya.dianov, karinne.ramirez, dean, p.lanillos, florian.bergner, gordon}@tum.de

[1]In this work, an ontology representation encodes contextual knowledge.

II. RELATED WORK

One set of approaches to learn a task structure from demonstrations is based on the analysis of the low-level trajectory information [4]. For example, Lee et.al. [5] represent a task as a set of motion primitives and utilise the Gaussian Mixture Models (GMM) to extract task models which can be later combined with Dynamic Motion Primitives (DMP) to improve model generalisation [6]. Other methods manually segment the motion sequences and model them as a finite state machine [7] or DMP [6]. On the other hand, several approaches are using Hidden Markov Models (HMM) to recognise repeated structures at several levels of abstraction in the demonstration data which are segmented into separate tasks [8]. While such methods are robust to small changes in the task environment (for example, the position of the target for the *pick and place* task), they do not capture task semantics which makes it complicated to reuse the obtained models in different domains (for example, reuse *pick and place* task in *plate cleaning* task). Additionally, bootstrapping can be used to extract both semantic and sensory-motor information from the demonstrations which can significantly improve the acquisition of the task models [9], though the high complexity of the task structure do not allow the reusability of the obtained knowledge to accelerate task learning.

On the other hand, the symbolic representation can provide better flexibility in building complex task structures applicable to different domains. Hierarchical Task Networks (HTN) is a common example of such abstract task representation which utilises symbolic actions with environmental preconditions and effects [10]. HTN can be manually specified [11], generated using predefined heuristic rules [12], or autonomously constructed through causality analysis and subgoal discovery [13]. Another way of obtaining symbolic specifications is through natural language interaction with a demonstrator where the task structure can be encoded as a Petri net [14] or a graph [15]. The obtained specifications can be then used to perform autocompletion of the new learned tasks and reduce the number of steps required for demonstrations [16]. However, the presented methods do not provide any mechanism to modify and relearn a single component of known tasks, or reuse the obtained task structures to speed up learning.

Similar to the above approaches, we propose a technique capable of extracting abstract symbolic task specifications from human demonstrations. The main advantage of our method is that it allows to accelerate teaching process by using knowledge about learned tasks and utilising similarity between different task parameters. Additionally, our technique allows to extract general representations of the task structures which are applicable to different domains.

III. FRAMEWORK OVERVIEW

In this section, we will define the main components of our system, describe connections between them and provide an overview of the proposed learning algorithm.

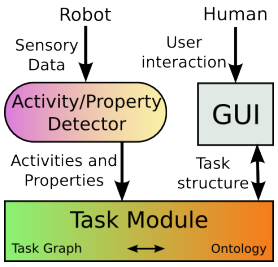


Fig. 2: Diagram of the main framework components.

A. Framework design

Our framework is shown in Fig. 2, it consists of activity/property detector module, task module and GUI. First, activities and properties are detected from the robot sensory data using the Inferring Semantic Human Activities Reasoner ISHAR method [17]. This method extracts semantics of the human activities using decision tree, and discretize continuous environment into hand motion and object properties. The activities are semantic descriptions of high-level human actions and have symbolic representations (e.g. *Reach*, *Take*, *Release*, *Wipe* etc.)[2]. The properties can either represent relations between an abstract object and the environment (for example, the property *IsInHand(Fruit)* defines that any object which belongs to the class *Fruit* is located in the robot hand) or describe a specific state of the object using a data value (for example, the property *HasPosition(Sponge, [1 1 1])* defines that the object *Sponge* has a Cartesian position [1,1,1]). The detected activities and properties are utilised to generate a task graph, which represents the key structure of the learned tasks, using the method described in Section III-B. To extract an abstract and domain independent structure of the tasks we enhance our framework with an ontology. The ontology contains the contextual information of extracted task structures and allows to reuse it for teaching similar tasks using backtracking and siblings analysis techniques. The example of the ontology is shown in Fig. 3b. Our ontology is represented through the Web Ontology Language (OWL) [18] and generated as a knowledge base in Knowrob [19]. Additionally, the demonstrator can use a graphical interface (GUI) in order to request the system to infer the structure of the desired task, and also select specific values for this task (for example, selecting the location for the *pick and place* task).

B. Problem formalization

We introduce a *task graph* which is a directed graph $G(V,E,P_g)$. Vertices V represent *activities* $\{a_1,...,a_n\} \in A$, where $V \subset A$, as described in the previous section. Edges E represent *transitions* between the activities and $P_g = \{p_1,...,p_k\}$ represents the graph properties which is a list of properties for all the activities in V. An example of the generated task graph is shown in Fig. 3a, where $A = \{a_1,a_2,a_3,a_4,a_5\}$ is the list of activities in the graph,

[2]In this work, we do not consider parametrization of the low-level motor primitives or motion trajectories.

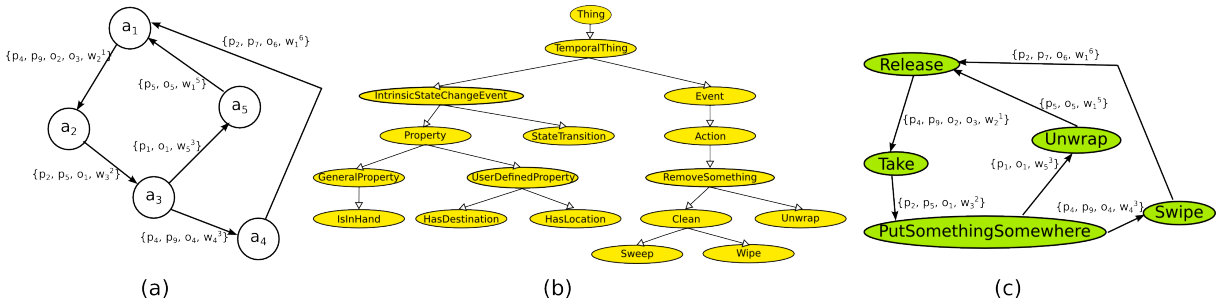


Fig. 3: Example of the task graph learning, where (a) represents an abstract graph, (b) represents the ontology and (c) represents graph (a) where $\{a_1, a_2, a_3, a_4, a_5\}$ were replaced by actual detected activities.

$P_g = \{p_1, p_2, p_4, p_5, p_9\}$ is a list of task graph properties. A *transition* is identified by a set of objects O_t involved in this transition, the properties P_t which are a subset of the graph properties $P_t \subset P_g$ and a weight $w_i^{i-1}, i = 1..n$ which indicates a number of transition occurrences between activities a_{i-1} and a_i from the previous demonstrations, and the weight will later be used for the task inference. For example, in Fig. 3a we observe that the nodes a_1 and a_2 are connected, creating a transition (a_1, a_2) with weight w_2^1, objects $O_t = \{o_2, o_3\}$ and properties $P_t = \{p_4, p_9\}$.

Algorithm 1 Task graph learning algorithm.

1: **Input:** $\{a_1, ..., a_n\}$
2: **Output:** $G(V, E, P_g)$
3: **for** $i \leftarrow 2, n$ **do**
4: **if** $a_i \notin G$ **then** $G.append(a_i)$
5: $P^{i-1} \leftarrow get\_properties\_from\_sensors(a_{i-1})$
6: $P^i \leftarrow get\_properties\_from\_sensors(a_i)$
7: $O_t \leftarrow get\_objects\_from\_sensors$
8: **for** $j \leftarrow 1, m$ **do** ▷ m=sizeof(P^i)
9: **if** $p_j^i \notin P_g$ **then** ▷ $p_j^i \in P^i$ is a property of a_i
10: $P_g.append(p_j^i)$
11: **for** $j \leftarrow 1, k$ **do** ▷ k=sizeof(P_g)
12: **if** $p_j^{i-1} \neq p_j^i$ **then**
13: $P_t.append(p_j^{i-1}, p_j^i)$
14: **for** $j \leftarrow 1, k$ **do**
15: **if** subclass(p_j^i, $UserDefinedProperty$) **then**
16: $p_j^i \leftarrow Var$
17: **if** $(a_{i-1}, a_i) \notin E$ **then** ▷ E are edges of current G
18: $w_i^{i-1} \leftarrow 1$
19: $E.append((a_{i-1}, a_i), w_i^{i-1}, P_t, O_t)$
20: **else** ▷ update edges
21: $w_i^{i-1} \leftarrow w_i^{i-1} + 1$
22: $P_t' \leftarrow$ current edge properties
23: $O_t' \leftarrow$ current edge objects
24: **for** $j \leftarrow 1, k$ **do**
25: **if** $p_j^i \notin P_t'$ **then**
26: $P_t'.append(p_j^i)$
27: **for** $j \leftarrow 1, l$ **do**
28: **if** $o_j \neq o_j'$ **then** $o_j' \leftarrow common\_class(o_j, o_j')$

C. Task graph learning

Using activities and properties detected from the robot sensory data (Fig. 2) the framework records the activity sequence $\{a_1, ..., a_n\}$ corresponding to the demonstrated task. Next, this sequence is used as an input for Algorithm 1. For example, in order to generate the graph shown in Fig. 3a we use two sequences of activities $\{a_1, a_2, a_3, a_5, a_1\}$ $\{a_3, a_4, a_1\}$. The algorithm first initialise the graph $G(V, E, P_g)$ as an empty graph and selects the first pair (a_1, a_2) from the input sequence. Then, the algorithm generates two vertices a_1, a_2 in the graph, detects properties (P^1, P^2), and the objects O_t involved in the transition (a_1, a_2) are obtained from the property detector module (Fig. 2). Next, the algorithm detects that the only properties which are changing during the transition are $\{p_4, p_9\}$. After that, the algorithm detects that the transition (a_1, a_2) does not exist in the set of graph edges and adds it to E using the following format: $\{p_4, p_9, o_2, o_3, w_2^1\}$, where w_2^1 is set to 1 (first transition). Next, the algorithm select a new pair from the input set of activities and repeats the process until the graph shown in Fig. 3a is obtained.

Additionally, the algorithm checks using the ontology whether an activity property belongs to the class of *UserDefinedProperty* and replaces its value with a variable (see Algorithm 1 line 20) in order to indicate that during task inference the user has to specify the value of this property. For example, activities $\{a_1, a_2, a_3, a_4, a_5\}$ from the Fig. 3a are replaced by the actual detected activities (see Fig. 3c). Additionally, p_4 is a *HasDestination* property which defines the destination of the *PutSomethingSomewhere* activity. The current value of p_4 is the *Box*. Using our ontology the framework will infer (Fig. 3b) that *HasDestination* is a subclass of *UserDefinedProperty* and replace its value *Box* with the variable. When the system infers the generic task structure (e.g. *pick and place*) the value of this variable should be defined by the user (for example, the user can select value *Trash* in order to generate a specific task *pick Fruit and place in Trash*).

D. Task inference

In the previous section, we described how to extract knowledge about different tasks. In order to reuse this knowledge to accelerate the learning process, the human

requests the framework about the structure of the task he wants to teach the robot. The framework will infer the most similar structure of the requested task and output it to the human using a GUI (Fig. 2).

To infer a task T represented as a directed subgraph of G, the demonstrator has to formulate a request using the GUI (Fig. 2). This request is defined as a pair (a_{key}, a_{stop}), where a_{key} indicates the key activity of the desired task, and a_{stop} defines the final state of the task. The framework uses Algorithm 2 to find the most similar structure of the requested task.

Algorithm 2 Task inference algorithm.

Input: $\{a_{key}, a_{stop}\}$
Output: T ▷ inferred task
$init(T)$
$P_g \leftarrow get\_current\_properties\_from\_sensors$
$a_{start} \leftarrow get\_activity\_from\_properties(P_g)$
if $a_{start} \notin V$ **then**
  return T
$a_{key} \leftarrow \arg\max_{a \in V} Sim(a, a_{key})$
$a_{stop} \leftarrow \arg\max_{a \in V} Sim(a, a_{stop})$
$G_{key}^{start} \leftarrow find\_shortest\_path\_with\_max\_weight(a_{start}, a_{key})$
$G_{stop}^{key} \leftarrow find\_shortest\_path\_with\_max\_weight(a_{key}, a_{stop})$
$T \leftarrow G_{key}^{start} \cup G_{stop}^{key}$

For example, the demonstrator wants to infer the cleaning task, in which a robot should wipe a plate, using the learned graph shown in Fig. 3c. The demonstrator selects *Wipe* as a key activity of the task and *Release* as a final state, and the framework (Fig. 2) executes Algorithm 2 to infer the desired task. The algorithm uses sensory data to infer the current values from the task graph properties P_g, identify the activity corresponding to these properties and mark it as starting activity a_{start}. In our example, *Take* is the starting activity. Next, the algorithm finds in the task graph the semantically closest activities for a_{key} and a_{stop} using Wu and Plamer concept similarity measure [20]. We define a_{ij} as a common activity class in the ontology for activities a_i and a_j and evaluate the similarity measure as follows:

$$Sim(a_i, a_j) = \frac{2 \times D_{ij}(a_{ij})}{D_i(a_i) + D_j(a_j) + 2 \times D_{ij}(a_{ij})} \qquad (1)$$

where $D_i(a_i)$ is a distance from a_i to a_{ij} (which is a number of nodes on the path in the ontology tree), $D_j(a_j)$ is a distance from a_j to a_{ij} and $D_{ij}(a_{ij})$ is a distance from the a_{ij} to the root class.

For example, we evaluate *Wipe* against the activities *Sweep* and *Unwrap* from the graph in Fig. 3c. Using the ontology shown in Fig. 3b the algorithm identifies that *Clean* is a common class for *Wipe* and *Sweep*, and *RemoveSomething* is a common class for *Wipe* and *Unwrap*. Next, the algorithm evaluates similarity measures using Eq. (1):

$$Sim(Sweep, Wipe) = 0.67$$

$$Sim(Unwrap, Wipe) = 0.4$$

The algorithm repeats the process for the remaining activities in the graph and detects that the most similar activity for *Wipe* is the activity *Sweep* with a similarity measure of 0.67 and the most similar activity for *Take* is *Take* with a similarity measure of 1. Those activities will be selected as a new pair (a_{key}, a_{stop}).

Next, the algorithm finds the shortest path from a_{start} to a_{key} and from a_{key} to a_{stop} using Dijkstra algorithm [21]. After that, the system uses the maximum weight to select the best path. Then, the algorithm saves the obtained subgraphs as $G_{key}^{start} = \{a_1, a_2\}$ and $G_{stop}^{key} = \{a_2, a_3, a_5\}$ respectively. Next, the algorithm merges the obtained subgraphs using the key activity a_{key} as a link in order to obtain the requested task $T = \{a_1, a_2, a_3, a_5\}$.

IV. RESULTS

In order to test the generalization of our method, we first trained our framework described in Section III using human demonstrations from a sandwich making scenario. Then, we applied the obtained graph to a different problem, where we were teaching a mobile robot an orange sorting scenario. The results will demonstrate that our method can extract the abstract structures of the tasks for the trained domain, and then successfully reuse them to accelerate the teaching of new tasks in a different domain.

A. Task graph learning

In order to show the advantages of having symbolic representation, we use the proposed framework to learn the task graph (Fig. 2) from the image domain and then applied it to the real robot. For training, we select sandwich making datasets [3] which contain 10335 image recordings obtained from 5 cameras synchronised at 60Hz. We select 3 random participants, where each participant has 3 trials, which results in total 9 training datasets [22]. We manually label the data and identify 14 different activities (e.g. *Reach*, *Take*, *Unwrap*, etc.). To obtain the task graph we use the Algorithm 1 described in Section III-B and the resulting graph is shown following the video example: `http://www.youtube.com/watch?v=g0P6annwiTg`.

B. Experimental Setup

Next, we validate our obtained task graph on the different scenario of orange sorting. The experimental setup is shown in Fig. 4. We use a robotic platform called Tactile Omni-directional Mobile Manipulator (TOMM) which consists of two industrial robot arms (UR-5) covered with artificial skin [23], two Lacquey grippers also covered with our artificial skin and one Asus Xtion Pro, which is a color image and depth sensor. We implemented the framework described in Section III (Fig. 2) using ROS[4] middleware and a visual attention system [24].

[3] http://www.ics.ei.tum.de/ics-data-sets/cooking-data-set/
[4] http://www.ros.org/

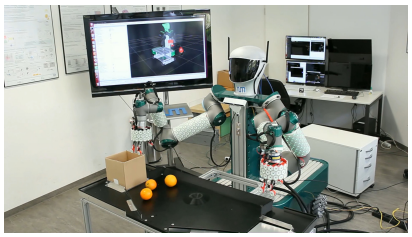


Fig. 4: Experimental setup for orange scenario teaching.

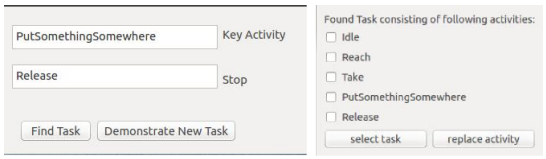
C. Testing scenarios

In order to demonstrate that our method can successfully accelerate the task learning process, we select two different scenarios. In the first one, we evaluate the teaching acceleration when the structures of the learned tasks are known to the framework. In the second scenario, we evaluate how our framework can improve the teaching of unknown tasks using similarity of task structures.

We select the orange sorting as a first scenario, which was originally implemented in [25]. In this scenario, the demonstrator shows the robot how to identify good oranges (the one with rigid texture) and bad oranges (the one with soft texture). Then, the demonstrator shows that good oranges should be put into the box and bad oranges should be thrown into the trash. The following tasks are defined by the demonstrator:

1) Put orange into the squeezing area (the special area on the table to check the orange quality).
2) Put good orange into the box.
3) Put bad orange into the trash.
4) Identify good orange.
5) Identify bad orange.

However, the tasks 1-3 are just variants of the *pick and place* task with different destinations and tasks 4-5 are different by a parameter which identifies good/bad orange. Instead of conducting redundant demonstrations the robot can utilise its knowledge to infer the known structure of the desired tasks.



(a) (b)

Fig. 5: GUI for robot task teaching.

Moreover, the demonstrator can interact with the robot through a GUI (see Fig. 5a). In order to request the knowledge about the *pick and place* task structure, the demonstrator selects *PutSomethingSomewhere* as a key activity and *Release* as a final state (see Section III-C). The framework will infer the task structure shown in Fig. 5b. Then, the user selects the obtained task structure, and the framework notifies him that he has to specify the desired destination of *PutSomethingSomewhere* activity. The framework assists humans by providing a list of possible destinations (*Box*, *Trash* and *Squeeze Area*) for the *PutSomethingSomewhere* activity using the data from the visual sensors. The human can select a destination from the list suggested by the framework, or use the kinesthetic teaching to demonstrate it. Similar to this, the human teaches the robot tasks 2 and 3.

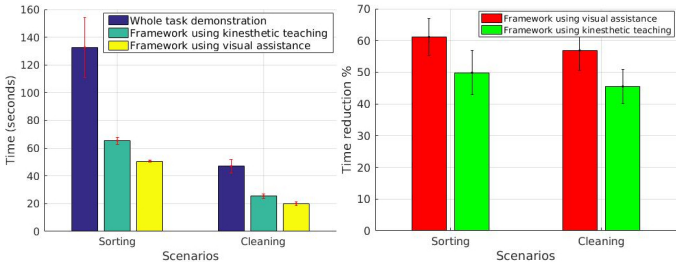
The task 4 is unknown to the system, and the human has to use kinesthetic teaching to demonstrate this task. The framework will insert a new branch in the task graph, which represents the structure of task 4. Then the demonstrator can reuse the obtained knowledge to teach task 5, where he need to demonstrate only values of a bad orange.

We define the cleaning task as a second scenario, to evaluate the robustness of our system against tasks with the similar structure. In this scenario, the demonstrator teaches the robot to clean a plate with a sponge. The demonstration consist of the following steps: reach a sponge; take the sponge; move the sponge above the plate; wipe the plate with the sponge; move the sponge to the original location; release the sponge.

The demonstrator generates a request to the knowledge base using *Wipe* as a key activity. The task graph does not contain any information about tasks involving this activity, however, it will find a similar task of unwrapping cheese with similarity measure 0.35. The retrieved sequence of steps is: reach the wrapped cheese; take the wrapped cheese; move the wrapped cheese to the cutting board; unwrap the cheese; move the wrap to the trash, release the wrap. If the demonstrator wants to reuse this structure, he selects *UnWrap* activity in the GUI and selects “replace” button (Fig. 5b). Then, through the GUI and kinesthetic teaching, the human demonstrates the new activity *Wipe* to the framework, which will use it to modify the task structure shown Fig. 5b and update the existing task graph.

In order to evaluate our framework, we measure the time required to teach each scenario using our method and compare the results with the time required to teach the same scenarios using only kinesthetic demonstrations. We conducted 5 experimental trials, and the results are shown in Fig. 6a. The blue bar indicates time measurements for the whole task demonstration, the green bar shows time measurements for our framework, where kinesthetic teaching was used to demonstrate the missing information, and the yellow bar indicates time measurements when the data from the visual sensors was used to specify locations of *pick and place* tasks.

The second scenario was evaluated similarly to the previous one. We measured the time required to teach the entire cleaning task using the whole task demonstration and compared it with our system, where the similarity between structures of different tasks was used to accelerate the teaching process. The results are shown in Fig. 6a. The advantage of our method compared to manual task specification [26] is that the user is required to demonstrate only missing information, instead of generating a complete description every time he teaches a new task. The time



(a) (b)

Fig. 6: Experiment results: (a) is time measurements for task teaching using the whole demonstration and our framework, (b) is time reduction for task teaching provided by our framework compared to the whole demonstration.

reduction provided by our method compared to the whole task demonstration is shown in Fig. 6b. Our framework allows to reduce the time required for teaching the orange sorting scenario from 136.3 to 53 seconds (61.12%) and time required for teaching the cleaning scenario from 48.7 to 21 seconds (56.87%).

V. CONCLUSIONS

In this work we introduce a technique which allows to extract key structures of different tasks from human demonstrations and reuse the obtained knowledge and past experience to accelerate the teaching process. First, we generate the task knowledge from a dataset of the sandwich making scenario. Then, we validate the obtained task graph in our robot TOMM by teaching it two different scenarios of orange sorting and plate cleaning. The results show that by reusing the learned task structure, our framework can accelerate the teaching process time from 136.3 to 53 seconds (61.12%) for sorting oranges scenario and from 48.7 to 21 seconds (56.87%) for cleaning scenario. Additionally, our framework was trained in one domain (cooking scenario) and then reused the obtained knowledge in different domains (sorting and cleaning).

Our future work will evaluate the framework robustness towards complex tasks demonstrated by multiple non-expert users. Additionally, we will explore in more detail the similarity between different task structures to find suitable methods for autonomous extraction of new properties and activities.

REFERENCES

[1] M. Stenmark, J. Malec, and A. Stolt, "From high-level task descriptions to executable robot code," in *Intelligent Systems' 2014*. Springer, 2015, pp. 189–202.

[2] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek *et al.*, "Robot task planning and explanation in open and uncertain worlds," *Artificial Intelligence*, 2015.

[3] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *2015 IEEE ICRA*. IEEE, 2015, pp. 1503–1510.

[4] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[5] S. H. Lee, I. H. Suh, S. Calinon, and R. Johansson, "Autonomous framework for segmenting robot trajectories of manipulation task," *Autonomous Robots*, vol. 38, no. 2, pp. 107–141, 2015.

[6] A. M. Ghalamzan, C. Paxton, G. D. Hager, and L. Bascetta, "An incremental approach to learning generalizable robot tasks from human demonstration," in *2015 IEEE ICRA*, May 2015, pp. 5616–5621.

[7] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg, "Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms," in *2015 IEEE ICRA*, May 2015, pp. 1202–1209.

[8] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[9] F. Wörgötter, C. Geib, M. Tamosiunaite, E. E. Aksoy, J. Piater, H. Xiong, A. Ude, B. Nemec, D. Kraft, N. Krüger *et al.*, "Structural bootstrappinga novel, generative mechanism for faster and more efficient acquisition of action-knowledge," *IEEE Transactions on Autonomous Mental Development*, vol. 7, no. 2, pp. 140–154, 2015.

[10] A. Tate, "Generating project networks," in *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2*. Morgan Kaufmann Publishers Inc., 1977, pp. 888–893.

[11] N. Nejati, P. Langley, and T. Konik, "Learning hierarchical task networks by observation," in *The 23rd ICML*. ACM, 2006, pp. 665–672.

[12] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *The Tenth ACM/IEEE HRI*. ACM, 2015, pp. 205–212.

[13] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in *2016 IEEE ICRA*. IEEE, 2016, pp. 5469–5476.

[14] G. Gemignani, E. Bastianelli, and D. Nardi, "Teaching robots parametrized executable plans through spoken interaction," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 2015, pp. 851–859.

[15] C. Meriçli, S. D. Klee, J. Paparian, and M. Veloso, "An interactive approach for situated task specification through verbal instructions," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. IFAAMAS, 2014, pp. 1069–1076.

[16] G. Gemignani, S. D. Klee, M. Veloso, and D. Nardi, "On task recognition and generalization in long-term robot teaching," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 2015, pp. 1879–1880.

[17] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities," *Artificial Intelligence*, 2015.

[18] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "Owl web ontology language reference," *W3C Recommendation February*, vol. 10, 2004.

[19] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013.

[20] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, 1994, pp. 133–138.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[22] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Understanding the intention of human activities through semantic perception: observation, understanding and execution on a humanoid robot," *Advanced Robotics*, vol. 29, no. 5, pp. 345–362, 2015.

[23] P. Mittendorfer and G. Cheng, "Humanoid multimodal tactile-sensing modules," *IEEE Transactions on robotics*, vol. 27, no. 3, pp. 401–410, 2011.

[24] P. Lanillos, J. F. Ferreira, and J. Dias, "Designing an artificial attention system for social robots," in *2015 IEEE/RSJ IROS*. IEEE, 2015, pp. 4171–4178.

[25] E. Dean, K. Ramirez-Amaro, F. Bergner, I. Dianov, P. Lanillos, and G. Cheng, "Robotic technologies for fast deployment of industrial robot systems," in *42nd IEEE IECON*. IEEE, October 2016.

[26] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *2015 IEEE ICRA*. IEEE, 2015, pp. 6167–6174.