2013 IEEE RO-MAN: The 22nd IEEE International Symposium on
Robot and Human Interactive Communication
Gyeongju, Korea, August 26-29, 2013

Robot Task Learning from Demonstration Using Petri Nets*

Guoting (Jane) Chang and Dana Kulić

Abstract— **The ability to learn is essential for robots if they are to function within human environments. Learning requires an understanding of the underlying structure of what has been observed. This paper proposes a learning method that automatically creates Petri nets from observation of human demonstrations to model the underlying structure of tasks. The Petri net can be learned via a single or multiple demonstrations. The learned Petri nets are capable of generating action sequences to allow a robot to imitate the task. The proposed model also allows for generalization and variations in performing the task. The proposed method is tested on demonstrations of block stacking tasks and verified through robot imitation of the tasks in simulation and in physical experiments.**

I. INTRODUCTION

In order for robots to perform different services in society, they need the ability to carry out new tasks and adapt to changing environments. This requires robots to have the ability to learn. Existing implementations of robot learning tend to focus on specific tasks that are not easily generalized [1]. However, learning should allow a robot to generalize knowledge by grasping the *underlying structure* of what has been observed. This requires appropriate mechanisms of *abstraction* and *representation*.

The learning by demonstration approach [2], [3], [4] allows robots to learn tasks from human demonstration, allowing a human teacher to demonstrate the task in a natural manner. This paper proposes using Petri nets to represent abstracted knowledge from observed task demonstrations. Petri nets are compact descriptions of processes that can model parallel, non-deterministic and asynchronous systems [5], [6]. Furthermore, Petri nets come with pre-existing analysis tools, including the ability to determine a sequence of actions needed to go from one state in the system to another [5], [6]. The main contribution of this paper is the development of a novel algorithm for automatically creating Petri nets based on observed demonstrations that can be used for robot learning by demonstration. The Petri nets can be learned from both a single or multiple demonstrations. Once learned, the Petri nets can be used to generate action sequences for task imitation. Lastly, the Petri nets are flexible enough to allow generalization and variations in performing the task both during demonstration and generation. The proposed approach is validated through experiments teaching block manipulation tasks to a humanoid robot.

*This work was supported by the Ontario Graduate Scholarship Program and the Natural Science and Engineering Research Council of Canada.

G. J. Chang and D. Kulić are with the Department of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada {g5chang, dkulic}@uwaterloo.ca

II. RELATED WORK

Previous work related to robot learning can be divided into approaches that are mainly focused on task recognition [7], [8], [9] and approaches that enable both task recognition and action generation [10], [2], [3], [4]. Among these approaches, some focus on the objects and their relations [7], [8], [10] some focus on the actions, i.e. motions, involved in the task [9], [2], while some use both objects and actions to model the task [3], [4].

Summers-Stay et al. [7] extract events where objects touch each other or the demonstrator's hand into a tree structure called an activity tree. This method of modeling the task is capable of separating two tasks that are interleaved, but no actions are modeled and thus, actions for task imitation cannot be generated.

Aksoy et al. [8] use changes in the spatial relation of objects captured in videos to construct a transition matrix called the semantic event chain (SEC) to model the human demonstrated task. The SEC is a condensed sequence of spatial relations between objects, which encode whether objects are touching or overlapping. The original SEC model does not incorporate actions such that motions could be reproduced. This work was extended in [10] to execute simple straight-line pushing actions for manipulation tasks. The extension does not yet take trajectory shapes into account.

An example where motions are investigated to analyze a task is [9]. In this approach, a combination of support vector machines (SVMs) and hidden Markov models (HMMs) are used to model and identify actions either as simple primitives or as a composite of primitives. The composite of primitives model forms a directed graph that models the sequence of actions for performing a task. However, this work focuses on task recognition rather than task imitation.

Konidaris et al. [2] propose an approach where demonstrated trajectories are segmented into skill chains using reinforcement learning (RL) and change point detection. The skill chains represent the sequence of actions required to perform a task. In the case of multiple demonstrations, multiple skill chains may be created and then merged into skill trees. More general graphs than trees are not supported.

An early example of teaching by demonstration involving both actions and objects is the work in [3], where the robot learns from a human demonstrator via video recordings. Vision processing is applied to extract objects, actions and the overall goal of the task. The task is modelled only minimally using a stack of actions and environmental states.

A more recent example of teaching by demonstration using both actions and objects is [4]. Their proposed learning

978-1-4799-0509-6/13/$31.00 ©2013 IEEE

system has a similar structure to ours, where both objects and actions are extracted separately from video image frames and the abstracted information is then combined to model the overall task. However, the task modeling is simply a sequential ordering of the actions and objects detected that does not allow for the learned task to be generalized to different situations.

Petri nets have been applied in the robotics domain to model human-robot interaction [11], multi-robot systems [12], [13] and tasks with qualitative performance evaluation [14]. In all these applications the Petri nets were created manually and not automatically generated. Ziparo et al. [13] use a modularized approach to make dynamic adjustments to the Petri net structure based on new observations, i.e. a large Petri net is composed of known elementary Petri nets that are arranged in a sequence. However, the elementary Petri nets are still defined by humans rather than learned from observation. In this paper, we propose a novel approach to automatically create Petri nets from observation.

III. PETRI NET OVERVIEW

Petri nets (PNs) are a tool for modeling processes including any type of information or part flow. Originally developed by Carl Adam Petri [15], Petri nets have been applied in a variety of fields, including distributed software systems, manufacturing control systems, local-area networks, etc. [5]. Petri nets are usually generated manually and are used to *describe* and *analyze* systems [6].

A Petri net consists of two main components: places (P) and transitions (T). Places represent states or conditions in the system being modeled, while transitions represent different actions or events that may occur. The arcs connecting places and transitions are indicated by defining input places (I) for each transition and output places (O) for each transition, effectively defining the states or pre-conditions necessary for an action or event to take place and the states or post-conditions resulting from a particular action or event.

To describe the dynamics of a Petri net, places in the Petri net are marked with $tokens$ to indicate the current state(s) or condition(s) of the system. The tokens indicate which transitions are $enabled$, i.e. which action(s) or event(s), if any, may take place. A transition is considered enabled if all of its input places have at least one token in them. The $firing$ of a transition transfers tokens from the input places of the transition to the output places.

A marked Petri net consists of a 5-tuple [5], [6]:

$$C = \{P, T, I, O, \mu_0\} \tag{1}$$

where

P is the finite set of all places p_i (circles in graph)
T is the finite set of all transitions t_i (bars in graph)
$I(t_i)$ is the set of all input places of transition t_i
$O(t_i)$ is the set of all output places of transition t_i
μ_0 is the initial marking (initial location of tokens)

A marking μ consists of a vector with an element for each place p_i to indicate the number of tokens in that place.

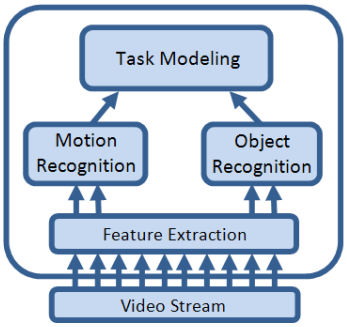


Fig. 1. Overview of proposed learning system

The reachability problem in Petri nets pertains to whether a particular marking of the Petri net can be attained given an initial marking [5], [6]. Reachability in Petri nets can be determined via the construction of a reachability tree [5], where the root node of the reachability tree is the initial marking. Every child node in the reachability tree represents a marking reachable from its parent node marking via one transition firing. The tree branch connecting each parent and child node is labeled with the corresponding transition that "fired". Thus, by finding the transitions that are enabled and computing the resulting marking by removing a tokens from input places and adding tokens to output places, the reachability tree branches and nodes can be constructed starting from the root node. The leaf nodes of the tree consist of the markings after which there are no more enabled transitions (terminal nodes) and markings which are the same as the marking of a previous node in the tree (repeated node). A repeated node just needs to link back to the previous node with the same marking, as the necessary child nodes are already in place for that previous node.

IV. PROPOSED APPROACH

The proposed approach develops an algorithm for automatically creating Petri nets from observations of human demonstration(s) of tasks. The Petri nets can then be used both to recognize known object states and motions when they are repeated by the human, and to generate sequences of motions that generalize the task and allow the robot to imitate it. Figure 1 shows an overview of how information is abstracted from video stream data capturing the human demonstration to form the Petri net model of the task.

To model a particular task, the places in the Petri net are used to represent states of the objects in the system related to the task, while the transitions in the Petri net are used to represent motions, i.e. actions taken by the human demonstrator. The sequence of object states and motions observed dictates the sequence of places and transitions of the Petri net. In order to automatically create such a Petri net, object and object state recognition, as well as motion recognition must be performed and the sequence of object states and motions determined.

Object and object state recognition are performed using two self-organizing maps (SOMs). Given a video stream

input, simple vision techniques are used to segment primary colored objects from each video frame, which allows consistent tracking of the objects in the environment. Object features, such as color, location and object size are also retrieved from each video frame. Using the first SOM, the color of an object is used as input to obtain an output label identifying the object. Using the second SOM, the object locations and relationships are used to assign a state label to each object. The object relationship is determined using the object locations and object sizes and indicates whether there is an object on top or below the current object. Vision and object recognition is not the main focus of this report, so simple techniques were used, but could be replaced later with more sophisticated methods.

Action recognition is performed using dynamic movement primitives (DMPs) [16] and affinity propagation (AP) [17] at the end of each observed motion, i.e. when an object moves, the motion is recorded and once an object stops moving, the motion is modeled and clustered [18]. DMPs model motions as dynamic nonlinear systems and were chosen as they can be used to generate motions to arbitrary goal positions during task imitation. The AP algorithm is used to cluster the weight parameters of the DMPs to identify similar motions. Furthermore, the AP algorithm selects one motion from each cluster to represent all the motions of the cluster. This representative motion is used to generate movement during robot task imitation.

The Petri net creation algorithm proceeds as follows:

1) When all objects are stationary, the state of each object is represented using a place in the Petri net. If the place representing a particular state does not already exist, a new place is added to the Petri net to represent the state of the object.
2) When one or more of the objects are moving, the object positions are recorded in order to obtain the trajectories of the motion.
3) When a motion has just finished, i.e. objects are no longer moving, the motion is first modeled via a DMP and then identified using AP. If the motion that has just finished is a new, previously unseen motion, it is added as a new transition to the Petri net. Currently, the proposed algorithm only considers the case when one object is moving at a time. Smaller motions of other objects are assumed to be due to occlusions when one object is moving.
4) The state (i.e. place in the Petri net) right before the motion and the state right after the motion are compared for each object. If the states are different, the state right before the motion is added to the input place(s) and the state right after the motion is added to the output place(s) of that transition. Additionally, the object that was moved during the motion, i.e. the object undergoing the most displacement, will also have its state prior to the motion and its state right after the motion connected via the transition, regardless of whether these states are different.
5) Return to step 1) or stop if all stationary and motion sequences in the observation have been processed.

Multiple demonstrations of the same task are supported, as the same sequence of object states and motions would result in places and transitions already present in the Petri net and would not be added again. In the case that there are variations in how the same task is performed during the demonstration, the created Petri net will either already reflect the variation or the Petri net will be extended to include new ways of performing the task.

For imitating the observed task, the created Petri net can be used to generate a sequence of motions as follows:

1) Use an image of the initial and desired final states of the task to be imitated as input.
2) Extract the objects and object states from the input images and convert the extracted states to a marking of the Petri net.
3) Using the initial marking as the root node, generate the reachability tree for the Petri net.
4) Using depth first tree traversal, go through the reachability tree to find a path between the root node (initial marking) and a tree node with the final marking. The path generated via depth first tree traversal contains the necessary sequence of transitions to reach the final marking from the initial marking. Always traverse the entire tree and store all potential paths found.
5) Select the shortest path found. If multiple shortest paths exist, pick the first one found. If additional criteria for task imitation are available, a more complex path selection system could be used.

The sequence of transitions in the shortest path found via depth first tree traversal of the reachability tree is the sequence of motions required to imitate the task. The DMPs will then be used to generate the actual motion trajectories for the robot to perform the task.

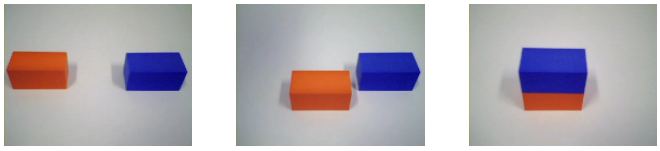
V. EXPERIMENTS

To demonstrate the proposed approach, videos of various block stacking tasks were captured using the NAO humanoid robot [19]. As only one camera is used, stereo images are not available and the distances to the blocks is fixed in the experiments. If necessary, depth information of objects could be extracted from the captured video using knowledge of the actual object size. To simplify the object tracking problem, we assume that the orientation of the objects do not change during the task demonstration. Two test settings are considered during the experiments: 2 block stacking and 3 block stacking.

A. 2 Block Setting

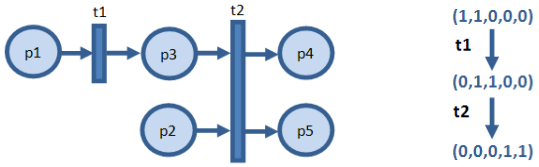
In this test setting two primary colored blocks were stacked by a human demonstrator as shown in Figure 2. For this test setting the Petri net has to be able to model a sequential task, where the order of actions matters.

A number of test cases were considered for this setting: exact imitation, subset imitation, multiple demonstrations and minor generalization. The result of each test case was used to let the NAO robot imitate the task in simulation



(a) Start (b) Intermediate step (c) Finish

Fig. 2. Frames of captured video depicting the 2 block stacking task



(a) Petri net graph (b) Reachability tree

Fig. 3. 2 block task, exact imitation case

(for all the test cases) and on actual robot hardware (for the exact imitation case). The video attachment of this paper illustrates the exact imitation case. During imitation, magnets were used to allow the robot to "grasp" and release the blocks. It is assumed that the robot already knows how to let the magnets connect and disconnect for "grasping" and releasing blocks, as the only motion learned is how to move the blocks themselves. The "grasping" motion simply consists of position interpolation from the robot's current hand position to the position of the object to be "grasped". The position of the object is extracted via vision feedback and the "grasping" motion is adjusted accordingly. The releasing motion proceeds similarly from the final object position to a pre-determined home position of the robot's hand.

1) Exact Imitation: The purpose of this test case is to see if the Petri net can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed. The input video depicts a red block and blue block being stacked. The red block starts off on the left side of the workspace as seen by the robot, while the blue block starts off on the right side of the workspace. The red block is moved towards the middle of the workspace first, before the blue block is moved on top of the red block. The resulting Petri net generated is shown in Figure 3a.

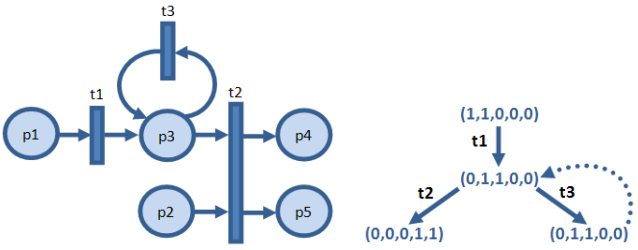
The places and their meaning, i.e. the object states that they represent, are summarized in Table I. The meaning of the transitions is as follows: t_1 refers to the first motion observed, which is the red block moving towards the center and t_2 refers to the second motion observed, which is the blue block on top of the red block. The created Petri net has captured the sequential nature of the task, as the second block can only be moved if the first block is already in place at the center of the workspace, i.e. place $p3$ is marked.

The initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The reachability tree constructed from the Petri net is shown in Figure 3b and the path between $\mu_{initial}$ and μ_{final} found

TABLE I

2 BLOCK TASK PLACE MEANINGS (L = LEFT, R = RIGHT, C = CENTER)

P	Object Location			Other Object Above or Below		
	L	R	C	No	Above	Below
p1	✓			✓		
p2		✓		✓		
p3			✓	✓		
p4			✓		✓	
p5			✓			✓



(a) Petri net graph (b) Reachability tree

Fig. 4. 2 block task, multiple demonstrations case

through the reachability tree is $t_1 t_2$ which exactly matches the task to be imitated. This illustrates that the Petri net created from observation can indeed be used by the robot to determine the correct sequence of actions to imitate the demonstrated task.

2) Subset Imitation: The purpose of this test case is to show that the Petri net can generate a sequence of actions corresponding to only part of the task that was observed. The same Petri net as for the exact imitation case is used. However, the initial and final images used as input were changed. For the first subset imitation, the initial and intermediate step image (as shown in Figure 2) was used as input, resulting in $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 1, 1, 0, 0)$ being extracted. The path found through the reachability tree is t_1, which is exactly the transition required to perform this part of the task. The second subset imitation shows the intermediate step and final image (as shown in Figure 2) as input, resulting in $\mu_{initial} = (0, 1, 1, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$ being extracted. The path found through the reachability tree is t_2, which is exactly the transition required to perform this part of the task. Hence, the Petri net can be used to generate sub-parts of a task, having been shown only the full task.

3) Multiple Demonstrations: The purpose of this test case is to demonstrate that the Petri net can summarize multiple demonstrations of the same task in the same model. A second video is used as input to the code. This video depicts a green block and a red block being stacked in the same fashion as in the first video. The green block starts off on the left side of the workspace as seen by the robot, while the red block starts off on the right side of the workspace. The only difference is that after moving the green block, the green block is gently tapped to straighten out its position. This tapping motion was not planned initially, but was added by the human demonstrator to better position the green block. The resulting Petri net generated from observing both the first and the second video is shown in Figure 4a.

The resulting Petri net is the same as the Petri net for the exact imitation case except for the tapping motion that is added as a third transition t_3 in the form of a self-loop. The self-loop reflects that the tapping motion was just a minor adjustment and did not change the position of the green block sufficiently in order to cause a change in the state detected. It is clear that the two demonstrations of the task were summarized successfully into the same Petri net which reflects the underlying structure of the task being modeled while taking minor variations, such as the extra tapping motion, into consideration.

The initial and final images used as input show the initial and final state of the task in the second video. The resulting initial and final markings are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The reachability tree constructed from the Petri net is shown in Figure 4b and the path between $\mu_{initial}$ and μ_{final} found through the reachability tree is t_1t_2 which exactly matches the task to be imitated. Even with multiple demonstrations, the Petri net can be used to find the correct sequence of actions for the robot to imitate the task. The tapping motion does not contribute to stacking the blocks and did not affect the path found through the reachability tree.

4) Minor Generalization: The purpose of this demonstration is to see if the Petri net can be used to generate the sequence of actions needed to perform the task with a slightly different set-up than what was observed. In this particular case, a new combination of colored blocks is the difference in the set-up. The same Petri net that was generated for the multiple demonstrations case and is shown in Figure 4a is used. However, the initial and final images used as input show a combination of a green block on the left and a blue block on the right. Using the initial and final images of the green and blue block combination, the initial and final markings that were extracted are $\mu_{initial} = (1, 1, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 1, 1)$. The path found via the Petri net generation algorithm is t_1t_2 which is the expected sequence of transitions to perform the desired task. Thus, the Petri net is able to generalize to new object combinations.

B. 3 Block Setting

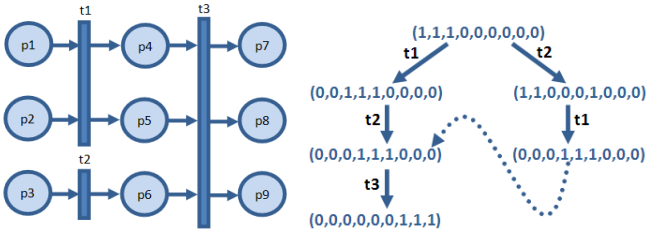
In this test setting, three primary colored blocks were stacked by a human demonstrator who placed two of the blocks side by side in the middle of the workspace first, before placing the third block on top of the first two blocks as shown in Figure 5. For this test setting the Petri net has to be able to model a combination of a parallel (concurrent) and a sequential task, i.e. the order of actions for placing the first two blocks can be interchanged, but the action of placing the third block must come last.

The two test cases considered for this setting are exact imitation and multiple demonstrations. Subset imitation and minor generalization capabilities were also verified.

As before, the results of each test case (including the subset imitation and minor generalization) were used to let the NAO robot imitate the task in simulation. The same assumptions as for the 2 block setting apply. The exact

(a) Start (b) Intermediate step (c) Finish

Fig. 5. Frames of captured video depicting the 3 block stacking task



(a) Petri net graph (b) Reachability tree

Fig. 6. 3 block task, exact imitation case

imitation case is shown in the video attachment of this paper. In the video, the NAO robot does not perfectly align the blocks due to joint limitations. This can be corrected in the future by treating the resulting block positions as an error case and letting the robot perform appropriate adjustments.

1) Exact Imitation: The purpose of this test case is to see if the Petri net can be used to model the stacking task and generate the sequence of actions needed to perform the task exactly as observed. The input video depicts a blue, red and green block being stacked. The blue block is initially on the left side of the workspace, while the red block is placed on top of the green block at the right side of the workspace. The red block is moved first in the video towards the middle of the workspace, then the blue block is moved towards the middle as well and placed to the left of the red block. Lastly, the green block is stacked on top of the blue and red blocks. The resulting Petri net generated is shown in Figure 6a.

The places and their meanings, i.e. the object states that they represent, are summarized in Table II. The meaning of the transitions is as follows: t_1 refers to the first motion observed, which is the red block moving from the right side of the workspace towards the center, t_2 refers to the second motion observed, which is the blue block moving from the left side next to the red block and t_3 refers to the third motion observed, which is the green block moving on top of the blue and red blocks. As can be seen, the created Petri net correctly reflects that the moving of the red and blue blocks are parallel actions and their order of occurrence can be interchanged without affecting the end result.

The initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1, 1, 1, 0, 0, 0, 0, 0, 0)$ and $\mu_{final} = (0, 0, 0, 0, 0, 0, 1, 1, 1)$. The reachability tree constructed from the Petri net is shown in Figure 6b and two paths are found through the reachability tree for the given $\mu_{initial}$ and μ_{final} markings: $t_1t_2t_3$ which exactly matches the demonstrated task and $t_2t_1t_3$ which matches a possible

TABLE II

3 BLOCK TASK PLACE MEANINGS (L = LEFT, R = RIGHT, C = CENTER)

P	Object Location					Other Object Above or Below		
	L	R	Around C			No	Above	Below
			L	R	C			
p1		✓						✓
p2		✓					✓	
p3	✓					✓		
p4				✓		✓		
p5		✓				✓		
p6			✓			✓		
p7				✓			✓	
p8					✓			✓
p9			✓				✓	

alternative sequence of performing the task. From this result, it can be seen that the Petri net not only correctly reflects parallel actions, but it is also possible to find variations in the sequence of actions for performing a task, even though only one of the possible sequences was observed in demonstration. Currently, the code is set up to select the first shortest path found, i.e. $t_1t_2t_3$, but if additional information about the desirability of one sequence versus another were available, it would be possible to take the additional information into consideration.

2) Multiple Demonstrations: The purpose of this demonstration is to see if the code can summarize multiple demonstrations of the same task in the same Petri net. A second video is inputted into the code depicting the alternate sequence of stacking the blocks, i.e. the initial and final block positions are the same, but the blue block is moved towards the middle of the workspace first, before the red block is moved. The resulting Petri net after showing both videos is the same as for the exact imitation case shown in Figure 6a. The created Petri net captures the fact that the underlying structure of the demonstrated task is the same despite variations in the sequence of the parallel components over multiple demonstrations.

Again, the initial and final images used as input show the initial and final state of the task. The resulting initial and final markings are $\mu_{initial} = (1,1,1,0,0,0,0,0,0)$ and $\mu_{final} = (0,0,0,0,0,0,1,1,1)$. Two paths are found through the reachability tree for the given $\mu_{initial}$ and μ_{final} markings: $t_1t_2t_3$ and $t_2t_1t_3$ which match the two observed variations of performing the task. This result illustrates that even with multiple demonstrations of a task with parallel and sequential components, the Petri net can be used to determine the sequence of actions needed for the robot to perform the task.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes an approach to automatically build Petri nets from videos of human demonstrations of tasks. The created Petri net may be used to determine a path between an initial and final (desired) marking and can be used both for action sequence recognition and generation. The created Petri net can successfully capture sequential and parallel tasks and allows the detection of sub paths and alternate paths. Furthermore, multiple demonstrations depicting the same underlying task lead to the same Petri net being generated with variations successfully incorporated. The Petri net path generation can also be applied to situations slightly different from the demonstration. Future work will involve creating a hierarchy of Petri nets to model more complex task sequences to avoid single, large reachability trees which can be computationally expensive. Furthermore, comparisons between our proposed approach and other robot learning approaches may be made. Also, error correction via dynamical reconstruction of the reachability tree will be considered when failed actions are encountered. Lastly, vision techniques for object segmentation, tracking and feature extraction can be improved in the future and image depth information and object pose estimation can be added.

REFERENCES

[1] A. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Third IEEE Conf. Devel. Learn.*, 2004.

[2] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *IJRR*, vol. 31, no. 3, pp. 360–375, 2012.

[3] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Learning by watching: extracting reusable task knowledge from visual observation of human performance," *IEEE Trans. Robot. Autom.*, vol. 10, no. 6, pp. 799–822, 1994.

[4] Y. Nagai, "From bottom-up visual attention to robot action learning," in *IEEE 8th Int. Conf. Devel. Learn.*, 2009.

[5] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541 – 580, April 1989.

[6] J. L. Peterson, "Petri nets," *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252, September 1977.

[7] D. Summers-Stay, C. L. Teo, Y. Yang, C. Fermüller, and Y. Aloimonos, "Using a minimal action grammar for activity understanding in the real world," in *IROS*, 2012, pp. 4104–4111.

[8] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of object-action relations by observation," *IJRR*, vol. 30, no. 10, pp. 1229–1249, 2011.

[9] I. S. Vicente, V. Kyrki, and D. Kragic, "Action recognition and understanding through motor primitives," *Adv. Robotics*, vol. 21, no. 15, pp. 1687–1707, 2007.

[10] E. E. Aksoy, B. Dellen, M. Tamosiunaite, and F. Wörgötter, "Execution of a dual-object (pushing) action with semantic event chains," in *Humanoids*, 2011, pp. 576–583.

[11] C. Chao and A. L. Thomaz, "Timing in multimodal turn-taking interactions: Control and analysis using timed petri nets," *JHRI*, vol. 1, no. 1, pp. 4–25, 2012.

[12] D. Coman, A. Petrisor, A. Ionescu, and M. Florescu, "Role selection mechanism for the soccer robot system using petri net," in *EUROCON*, 2007, pp. 662–667.

[13] V. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara, "Petri net plans," *Auton. Agent Multi-Agent Syst.*, vol. 23, pp. 344 – 383, 2011.

[14] P. Lima, H. Gracio, V. Veiga, and A. Karlsson, "Petri nets for modeling and coordination of robotic tasks," in *IEEE Conf. on Sys., Man, Cybern.*, vol. 1, 1998, pp. 190–195.

[15] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Darmstadt University of Technology, Germany, 1962.

[16] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *ICRA*, 2002, pp. 1398–1403.

[17] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, pp. 972–976, 2007.

[18] G. J. Chang and D. Kulić, "Motion learning from observation using affinity propagation clustering," in *Ro-Man*, 2013. To be published.

[19] (2012) Home - corporate - aldebaran robotics. [Online]. Available: http://www.aldebaran-robotics.com/