Teaching Sequential Tasks with Repetition through Demonstration

(Short Paper)

Harini Veeraraghavan
Computer Science Department
Carnegie Mellon University
harini@cs.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
veloso@cs.cmu.edu

ABSTRACT

For robots to become prevalent in human environments, the robots need to be able to perform complex tasks often involving sequential repetition of actions. In this work, we present a demonstration-based approach to teach a robot generalized plans for performing sequential tasks with repetitions. We introduce action definitions through perception. Using the action definitions and the demonstration, the robot learns a task specific plan for tasks containing repetition of sub-sequences.

1. INTRODUCTION

A majority of tasks in human environments involve repetitions, be it assembling furniture using actions such as "HammerNail", "TightenScrew". For a robot to automatically generate a plan using the actions alone for a complex task such as assembling a furniture or performing some elaborate sequence of motions is very challenging. On the other hand, given an example demonstration, the robot can easily learn a task specific plan for performing the same task on different problems.

In this work, we contribute a demonstration-based approach to teach a robot task specific plans. We focus on real world domain and present an approach that learns task specific plans with repeating sub-tasks. Concretely, in our approach, both the human and robot actively participate in the learning task. Through demonstration the human instantiates the task specific actions. The robot learns the appropriate action definitions and then using the sequence of executed actions, learns a task specific plan with repetitions.

This paper is organized as follows. We first present the related work in Section 2 followed by the experimental domain and the basics of the teaching approach in Section 3. We present the learning approach in Section 4, an illustrative result in Section 5 and finally conclude the paper in Section 6.

2. RELATED WORK

Examples of works that actually implement a planning algorithm on a robot for learning to execute a task include the works in [3, 8]. Demonstration based learning approaches such as in [4, 5, 7] learn generalized plans for sequences of actions with little or no repetitions. The work in [1] uses demonstration-based learning for a single action. Another interesting approach to teaching sequential plans is in [9] where a robot learns a specific plan without any generalization or repetition from a user through simple spoken language dialogues. Our work to learning looping plans is most closely related to the work of Winner and Veloso [11] which learns domain specific plans from example demonstrations for completely defined domain specific actions in simulated domains. Another interesting approach to loop learning from demonstration with annotations is in [6]. Our approach differs from the afore-mentioned work in that the action definitions are themselves obtained through perception in a real world problem domain and the robot successfully learns a plan containing repetitions on sub-tasks.

Cite as: Teaching Sequential Tasks with Repetition through Demonstration (Short Paper), H. Veeraraghavan and M. Veloso, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril,Portugal,pp.1357-1360.

3. ELEMENTS OF LEARNING TASK AND EXPERIMENTAL DOMAIN

Figure 1: The robot used for the clear table task.

The reference task domain used in this work for learning plans with loops consists of clearing a table. Fig. 1 depicts the experimental platform and the robot used in this work. The task consists of applying a sequence of actions repeatedly to move all the objects from a table into a destination box. The individual actions such as *pick*, *drop*, *search*, etc are the set of skills that are pre-programmed into the robot. However, the robot does not know in what sequence the various actions need to be carried out to achieve a particular task, such as clearing the table. Similarly, the action definitions are general so that the robot does not know what objects are associated with each task. We call such actions robot behaviors. During demonstration, it learns the association of the robot behaviors to objects relevant to the particular task. We call such actions task-specific actions. The objects are identified by their color using color thresholding. We now define the different terms used in the paper.

- **Robot behavior** is a non-instantaneous sequence of physical actions performed by the robot that changes the state of the world. A behavior is composed of a number of primitive actions that are executed in a predefined sequence. The behaviors are pre-programmed into the robot. Example of a behavior is *PickObject(object)* which can be executed on any object.

- **Task-specific action** is an instantiation of the robot behavior but associated with a specific object type. For example, *PickObjectYellowBall(object - yellowballType)* is an instantiation of behavior *PickObject* but always executed only on objects that are yellow colored balls. The task specific action is also composed of a completely defined precondition and effects corresponding to the specific object types in its argument list. The task specific action with parameterized objects is referred to as task-specific operator or simply operator. The task specific action instantiated with specific objects such as *ball1* is referred to as a grounded action.

- **Predicate or Proposition** is a representation of the sensed measurements as facts or relations between different objects. Each proposition is associated with a visual measurment. For example, in order to verify the truth of a proposition such as *holdingObject(yellowBall)* , the robot first moves its hands to the level of its head and then checks if it can color segment the ball object.

- **State** is a set of observed predicates.

Robot Behavior A robot behavior is activated by the human demonstrator using an appropriate vision-based cue such as a colored card and can be executed on any object. Once associated with an object, a task-specific action is generated which is used for the task specific plan. The task specific actions on the other hand can only be executed on the specific object types associated with the input arguments. The object types in our case correspond to specific object color. The task specific actions are also generated during the demonstration phase.

- *searchObject_objecttype* : This action is performed by executing the following physical actions in order. (a) If object is not detected, follow table for a few time steps, else stop, (b) Move close to table and look for object, (c) If object is detected, stop, else go back to (a). The task specific action is renamed with the appropriate object type such as *searchObject_type1* for an object of type *type1* .

- *pickObject_objecttype* : This action is performed by executing (a) While object is not in center of image, move to align with object, (b) Grasp object with both hands.

- *carryObjectToBasket_objecttypeList* : This action is performed by executing the following physical actions: (a) If basket is not detected, hold object and follow table for a few time steps, else stop, (b) Move close to table and look for basket, (c) If basket is detected, stop, else go back to (a).

- *dropObjectIntoBasket_objecttypeList* : This action is performed as follows: (a) Release hands to drop the object.

Representing World Observations The effect of executing a behavior results in a change in the state of the world. In order to obtain the state of the world, the robot needs to have a knowledge of the relevant observations and know what measurements are required for each observation. These observations are represented in the plan as propositions. Thus, in addition to the behaviors, task specific actions and objects, the robot also contains a list of propositions with appropriate visual measurements. The set of propositions and the associated vision-based measurements used by the robot are as follows:

- *closeToObject_objecttype* Vision-based measurement checks whether the robot is standing sufficiently close to the object by measuring the objects relative distance from the robot. The proposition is set to true when the object is detected within a fixed pre-defined threshold τ from the robot.

- *holdingObject_objectype* To detect whether the robot is holding an object, it moves its arms to the level of its head and checks whether it can segment the appropriate object.

- *in_objectypeList* This is a binary predicate where the vision-based measurement checks which pair of objects in the argument list of action satisfy the condition for *in* . One object *obj1* is said to be inside the other object *obj2* when the bounding rectangle of the *obj1* is enclosed by the bounding rectangle of *obj2* .

4. TEACHING TASKS WITH REPETITIONS

The teaching approach consists of a demonstration phase and a learning phase. In the demonstration phase, the robot executes the sequence of robot behaviors on specific objects as indicated by the human. It then instantiates task specific actions from the behaviors, fills in the appropriate preconditions for those actions, and then learns a task specific plan for the executed action sequence.

4.1 Demonstration Phase

Demonstration is the first step in the teaching approach. To demonstrate the action sequence, the demonstrator indicates the appropriate actions and the objects relevant to the same action. Every action is associated with a specific color that can be identified by color thresholding upon viewing a correspondingly colored *action card*. The human indicates the objects by moving a laser pointer across the object. The robot tracks the laser spots and computes the region of interest as the bounding box enclosing all the tracked spots. It then identifies the appropriate object by matching the target model such as a color histogram or by applying an average (RGB) color threshold on the region of interest. As the goal of problem is not robust sensing, we eliminate perceptual ambiguities such as occlusions.

4.2 Learning Phase: Filling Action Preconditions and Effects

The first step to recognizing the plan for the demonstrated task is to extract the task specific action preconditions and effects. The algorithm for filling the action preconditions and effects is depicted in Algorithm 1. By operator we mean the task specific action associated with an object type. A grounded action on the other hand is a task specific action associated with a particular object corresponding to an object type. For task specific action definition extraction, we make use of the multiple occurrences of the same action instantiated on different objects but of the same type. In our case, the types correspond to the color. Given that the demonstrator is assumed to guide the robot through the actions in the correct order, in general the states preceding an action will contain all the predicates necessary to execute the same action. This in turn simplifies the extraction of the preconditions and effects. However, in the absence of multiple instantiations, a different learning algorithm such as [2]

Algorithm 1 Precondition and Effect Filling

Input: Grounded actions $\langle a_1,\ldots,a_N\rangle$ from demonstration
Input: Preceding and Succeeding states $\langle\{^{-}S_{a_1},^{+}S_{a_1}\}\ldots\{^{-}S_{a_N},^{+}S_{a_N}\}\rangle$ for each action
Output: Grounded actions $\langle a_1,\ldots,a_N\rangle$ with filled preconditions and effects

1: GROUP grounded actions into Operators $O_1,\ldots,O_k$, s.t. $\forall O_{op,op=1\ldots k}, \nexists a_j,a_k\{a_j,a_k \in op\}$, SUBSTITUTE$(a_j,a_k)$ is invalid
2: **For all** Operators op **do**
3: Collect the action states $\{^{-}S_{a_j},^{+}S_{a_j}\}\forall_j a_j \in op$.
4: Remove inconsistent action states.
5: **For all** Operators op **do**
6: Get $Preconditions^{op} \longleftarrow {}^{-}S_{a_1}\wedge\ldots\wedge{}^{-}S_{a_k}, a_1,\ldots,a_k \in op$
7: Effects:
8: **If** exists effect $e^x_{a_j}, a_j \in op \wedge \exists a_k \in op$ where $\forall_y e^y_{a_k}$ SUBSTITUTE$(e^x_{a_j}, e^y_{a_k})$ is invalid **then**
9: **If** exists predicates $c^{W=\{w_1,\ldots,w_m\}} \in {}^{-}S_{a_j}$ where $\arg(e^x_{a_j}) \cap \arg(c^W) \neq \emptyset \wedge \forall_{i,i\neq j}$, SUBSTITUTE$(e^x_{a_j}, e^y a_i)$ is invalid $\wedge\ c^W \ni {}^{-}S_{a_i}$, where $a_i, a_j \in op$ **then**
10: Add conditional Effect $condEffect^{op} \longleftarrow \{c^W, e^x_{a_j}\}$
11: **Else**
12: Add disjunctive Effect $Effects^{op} \longleftarrow e^x_{a_j} \vee Effects^{op}$
13: **Else If** $\exists$ effect $e^x_{a_j} \in \forall_i\{\triangle\langle{}^{-}S_{a_i},{}^{+}S_{a_i}\rangle\}a_i, a_j \in op$ **then**
14: Add conjunctive Effect $Effects^{op} \longleftarrow e^x_{a_j} \wedge Effects^{op}$
15: Fill in Preconditions and Effects for each action $a_j \in op$

will be more appropriate than the one presented in this work.

As shown in Algorithm 1, the inputs to the algorithm consists of the sequence of grounded actions obtained from the demonstration, and the corresponding states associated with each action. A state with negative '$-$' superscript such as ${}^{-}S_{a_j}$ corresponds to the state prior to executing the action a_j, while the state with positive superscript '$+$', such as ${}^{+}S_{a_j}$ corresponds to the state following the same action.

The first step in the Algorithm 1 is to group the grounded actions into their corresponding operators. The **SUBSTITUTE** procedure as shown in Line 1 of Algorithm 1 checks for the equality of two actions. In other words, substitute corresponds to replacing the parameters of one action for the other. So two grounded actions a_i, a_j correspond to the same operator when the result of their substitution is identical. Note that, here we are just comparing the action name and the arguments.

The next step of the algorithm is to collect the set of states corresponding to every action in each operator, following which, any inconsistent states are removed as depicted in Lines[2-4]. An inconsistent state is one where the intersection of the same (preceding) state corresponding to a specific grounded action with at least T (preceding) states corresponding to T grounded actions for the same operator is an empty set. In other words, for a pair of states ${}^{-}S_{a_k}, {}^{+}S_{a_k}$ abbreviated as ${}^{-}S, {}^{+}S$,

$$Inconsistent({}^{-}S,{}^{+}S) = \begin{cases} \text{If} \quad \sum_{i=1}^{m}\{{}^{-}S_{a_k} \cap {}^{-}S_{a_j}\} \rightarrow \{\emptyset\} > T, & \text{true} \\ \text{Else,} & \text{false} \end{cases}$$

Only the states preceding the action are checked for inconsistency. However, both of $\{{}^{-}S_{a_j}{}^{+}S_{a_j}\}$ from an action a_j will be removed when ${}^{-}S_{a_j}$ is found to be inconsistent. The inconsistency check is performed to ensure that the precondition list of an operator is never an empty set. It is assumed that all preconditions for an action are conjunctive.

Next, the preconditions for the operator is obtained as the intersection of all the consistent preceding states for the associated actions as shown in Line 6 of the Algorithm 1. The procedure for obtaining the effects is depicted in Lines [8-14] of the Algorithm 1. The effects of a grounded action a_j corresponds to the set of predicates in the succeeding state ${}^{+}S_{a_j}$ occurring mutually exclusively from the preceding state ${}^{-}S_{a_j}$. This is represented as $\triangle\{{}^{-}S_{a_i},{}^{+}S_{a_i}\}$ on Line 13 of Algorithm 1. An effect $e^x_{a_j}$ of a grounded action a_j for operator op is added as a conditional effect when,

- there does not exist a substitution for the same effect $e^x_{a_j}$ in at least one other grounded action a_k in the same operator,
- there exists one or more predicates $c^W \in {}^{-}S_{a_j}$ where the intersection of the argument list of c^W with $e^x_{a_j}$ is not an empty set and there exists no substitution for the same predicates in any state ${}^{-}S_{a_k}$ where substitution for the effect $e^x_{a_j}$ is invalid.

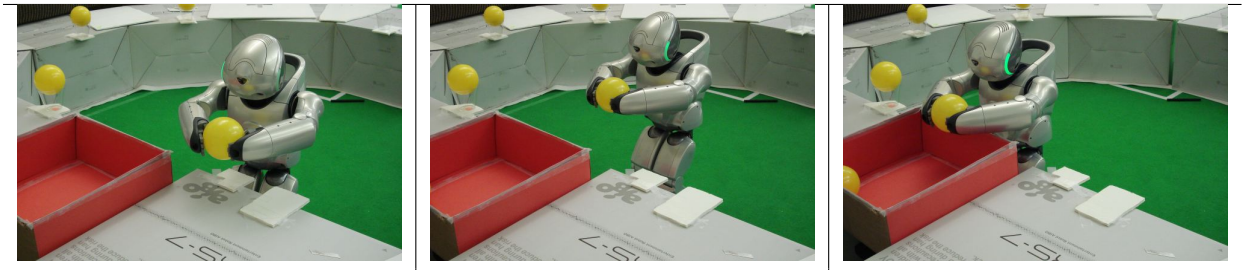
However, when no condition can be found, the effect $e^x_{a_j}$ is added as a disjunctive effect to the existing set of effects. Finally, the set of effects that occur in all the consistent action instantiations are added as conjuctive effects. At the end of this stage in learning, each action is represented in the classical PDDL format with preconditions and effects. As an example, the dropObject is represented as,

```
(:action dropObject_type1
 :parameters (?obj1 - type1 ?obj2 - type2)
 :preconditions (and (holdingObject_type1 obj1)
                     (closeToBasket_type2 obj2))
 :effects (and (in obj2 obj1)))
```

Figure 2: Representation of task specific dropObject action

4.3 Extracting Plans With Repetitions

Using the action definitions and the demonstration sequence, we then extract a partial ordering graph of the individual actions in the plan using [10] which links two steps in the demonstration that satisfy a producer-consumer relation where the producer step has an effect which is a precondition for the consumer step. The precondition is the rationale for the link. The last phase in learning from demonstration is to extract an executable planner from the demonstrated action sequence. In this work, we follow the similar definition as in programming languages for the *loops*. A sequence of actions forms a *loop* if and only if the same sequence of actions are repeated over different instances of the loop variable and there are no ordering constraints between actions occurring at different loop variable instances. The algorithm for learning the generalized plans from demonstration is depicted in Algorithm. 2. The first step transitively reduces the partial order graph for the action steps in the demonstration sequence to simplify computation for loop detection. In the next step, the actions are parameterized such that the grounded actions are replaced by actions with variables. Finally, the different steps and loops are arranged in the dependency order as obtained from the partial orderings. A set of actions forming a loop is merged with another loop when the actions in one loop are connected to the actions in the other loop through the producer-consumer orderings. Note that this merging still maintains the parallel execution of the steps along different branches of the merged loop.



Pick Object | Carry Object | Drop Object

Figure 3: Example sequence showing the execution of the task during demonstration.

Algorithm 2 Learning Looping Plans from Example

Input: Partial Order (PO) Graph
Output: Generalized Looping Plan
1: Transitively reduce PO Graph
2: Parameterize trace step actions
3: Detect LOOPS(Actions $a_1, \ldots, a_N$)
4: Order Steps by links.

5. ILLUSTRATIVE EXPERIMENT

Experiments were performed in indoor setting with the experimental setup as shown in Fig. 1. The objective of the experiment was to test whether the robot could learn a correct plan from the demonstrated sequence of actions. The domain is controlled such that no perceptual ambiguity such as occlusions or illumination variations occur. While restrictive for real world applications, robust sensing and planning with robust sensing is not the focus of this paper.

An example demonstration sequence consisted of the steps *searchObject, pickObject, carryObjectToBasket, dropObject* for two different balls repeated one after the other in the same order. An example of the task executed by the robot during the demonstration is depicted in Fig. 3. The plan learned from this demonstration sequence is depicted in Fig. 4. As shown, the robot is correctly able to learn an executable plan for the demonstrated sequence.

```
 while(?loopvar : type1)
if(haveObjectToSearch_type1 loopvar) then
   (searchObject_type1  loopvar)
if(closeToObject_type1 loopvar) then
   (pickObject_type1 loopvar)
if(holdingObject_type1 loopvar) then
  (carryObjectToBasket_type1_type2 loopvar obj2)
if(and (holdingObject_type1 loopvar)
       (closeToObject_type2 obj2))
   (dropObjectToBasket_type1_type2 loopvar obj2)
```

Figure 4: Example task specific plan.

6. CONCLUSIONS

In this work, we present an approach for teaching complex sequential tasks with repetitions through demonstration. We present a contribution where using the set of generic behaviors or skills and a demonstration, the robot can extract the task specific action definitions and finally a plan with repetitive execution of a sequence of actions. Additionally, the system has been implemented on a real world domain where the robot successfully transforms its executed actions and vision-based sensed measurements into an instantiated plan that can be used for learning an task specific planner.

7. ACKNOWLEDGEMENTS

The authors would like to SONY for making the QRIOs available to us for this project. The authors would also like to thank SONY for also making available the robot specific software libraries.

8. REFERENCES

[1] C. Breazeal, G. Hoffman, and A. Lockerd. Teaching and working with robots as a collaboration. In *Proc. Autonomous Agents and Multiagent Systems*, pages 1028–1035, 2004.

[2] Y. Gil. Learning by experimentation: incremental refinement of incomplete planning domains. In *Proc. Intl. Conf. on Machine Learning*, 1994.

[3] K. Haigh and M. Veloso. Interleaving planning and execution for asynchrnous user requests. *Autonomous Robots*, 5(1):79–95, 1998.

[4] N. Koenig and M. Mataric. Demonstration-based behavior and task learning. In *Working Notes AAAI Symposium To Boldly Go Where No Human-Robot Team Has Gone Before*, 2006.

[5] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Trans. on Robotics and Automation*, 10(6):799–822, 1994.

[6] T. Lau, S. Wolfman, P. Domingos, and D. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.

[7] M. Nicolescu and M. Mataric. *Models and Mechanisms of Immitation and Social Learning in Robots, Humans, and Animals: Behavioral, Social and Communicative Dimensions*, chapter Task learning through immitation and human-robot interaction, pages 407–424. 2006.

[8] N. Nilsson. Shakey the robot. Technical Report 323, SRI International, AI Center, SRI International, Menlo Park, CA, 1984.

[9] P.E.Rybski, K. Yoon, J. Stolarz, and M. Veloso. Interactive robot task training through dialog and demonstration. In *Proc. Human Robot Interaction Conf.*, 2007.

[10] E. Winner and M. Veloso. Analyzing plans with conditional effects. In *Proc. Intl. Conf. Artificial Intelligence and Planning Systems*, pages 23–33, 2002.

[11] E. Winner and M. Veloso. Loopdistill: Learning domain-specific planners from example plans. In *In ICAPS Workshop on Planning and Scheduling*, 2007.