Learning Task Models from Multiple Human Demonstrations

Staffan Ekvall and Danica Kragic*
Computational Vision and Active Perception and Centre for Autonomous Systems
Royal Institute of Technology, Stockholm, Sweden
ekvall,kragic@nada.kth.se

***Abstract*— In this paper, we present a novel method for learning robot tasks from multiple demonstrations. Each demonstrated task is decomposed into subtasks that allow for segmentation and classification of the input data. The demonstrated tasks are then merged into a flexible task model, describing the task goal and its constraints. The two main contributions of the paper are the state generation and contraints identification methods. We also present a task level planner, that is used to assemble a task plan at run-time, allowing the robot to choose the best strategy depending on the current world state.**

I. Introduction

Robot task teaching has during the past years received significant attention [1]–[8] and it has been recognized that more natural teaching methods are necessary so to allow ordinary users to teach robots new tasks by simply demonstrating them. From the viewpoint of task learning in humans it is known that such a strategy where a teacher's demonstration is used as a starting point of learning significantly speeds up the process and reduces the amount of trial-and-error steps. In robotics, such an approach to learning has been considered in frameworks of Learning by Imitation or Programming by Demonstration (PbD). In our previous work, we have considered the integration of different sensory modalities for task model generation in a PbD framework, [8].

An important issue to consider is that the initial task setting will change between the demonstration and execution time. A robot that has to set-up a dinner table may have to plan the order of handling plates, cutlery and glasses in a different way that previously demonstrated by a human teacher. Hence, it is not sufficient to just replicate the human movements but the robot i) must have the ability to recognize what parts of the whole task can be segmented and considered as subtasks so to ii) perform online planning for task execution given the current state of the environment. The important problem here is how to instruct or teach the robot the essential order of the subtasks for which the execution order may or may not be crucial. As an example, the main dish plate should always be under the appetizer or a soup plate and the order in which these are placed on the table is important. One way of addressing this problem is to demonstrate a task to the robot multiple times and let the robot learn which order of the subtasks is essential. In relation, the two main contributions in this paper are the state generation and contraints identification methodologies based on multiple human demonstrations.

This work has been supported by EU through the project PACO-PLUS, FP6-2004-IST-4-27657 and Swedish Research Council.

In this work, the proposed methodologies are evaluated in the framework of robotic object manipulation tasks. Learning such tasks is considered a hard problem since robots have a very limited world knowledge to start with and are mainly constrained by the type of available sensory modalities. For humans, much of the background knowledge is innate and one demonstration is often sufficient. This is not a case when considering a robot. There are two possible directions here: either we let the robot assume that the actions can be executed in any order, or that the actions have to be executed in the same order as the demonstration. The first alternative requires that the human instructs the robot of the possible task constraints during the demonstration. In this paper, we have chosen the latter alternative since it allows the robot to learn from multiple observations and improve the task model over time.

II. Motivation and Related Work

In our work we would like to teach the robot of how to set-up a dinner table, slice a cucumber or put in dishes into a dishwasher. Setting up a dinner table task can be viewed as a sequence of pick-and-place object manipulation subtasks, [9]. For this task, the robot is required to recognize objects, grasp them and put them on the table in specific geometric relation to each other. The relationship between objects can be represented relative to one object, e.g. main plate. Cutting a cucumber is more difficult since the robot has to learn that a knife should be held in a specific way related to the cucumber. Different from the first example, the relative relationship between objects changes during task execution. Mobile manipulation tasks such as mail delivery [10] include constraints, for example that the mail has to be collected before it can be delivered but the order of delivery may be irrelevant. In summary, for some of the tasks a specific order of subtasks is required and for some it is not. Here, the problem of learning object or mobile manipulation tasks is solved by identifying the goal state and the spatio-temporal constraints of the task.

Many of the current robot instruction systems concentrate on learning by imitation or PbD based on a single demonstration. However, the robot should be able to update the initial task model by observing humans or another robot performing the task. In other words, we need a task level learning system that builds constraints automatically identified from multiple demonstrations.

This problem has been studied for robot navigation and mobile pick-and-place tasks, [11] where a task is represented by the alternate paths shown during the teaching phase. Compared to our work, the robot is still required to follow one of the human demonstrations unless the task is refined. In this paper, we focus on object manipulation tasks which require that objects are represented in relation to each other. Thus, our work differs both in the state representation and the task generalization.

In [4], generation of task models based on multiple human is presented. *Essential interactions* that represent the important hand movements during a manipulation task are identified. Compared to our approach, the above system represents the task using generalized trajectories. This makes the method easier to adapt to different situations, but also less flexible, as it requires the world state to be roughly the same as during the demonstration. In our work, we do not store the hand trajectories, but instead what has been done. The robot can then reproduce the results of the human demonstration at execution time by planning a sequence of actions to reach the goal state.

III. System Overview

In a PbD framework, the robot learns the task by observing a human or another robot. In most systems, the user only demonstrates the task a single time. A more complete task model can be made if recordings from several user demonstrations are available. The system is designed so that several independent building blocks work together to produce the desired results. Figure 1 illustrates the program flow from demonstration to final execution.

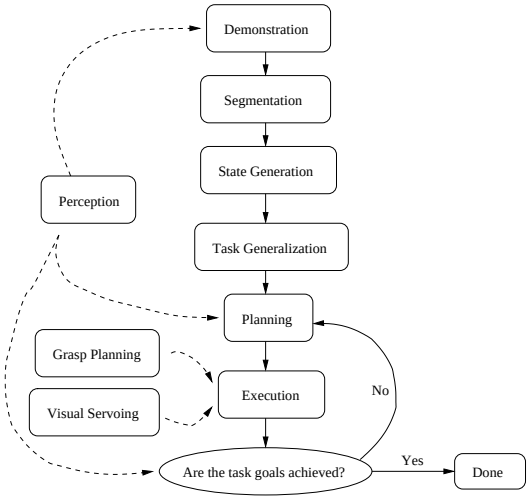

Fig. 1. The building blocks of our system.

We shortly summarize each of the building blocks of the system.

Segmentation - The segmentation of the task into isolated operations is an important research issue, [1]–[3], [12]. The task as a whole is unlikely to be observed again because of the minor variations that occur from demonstration to demonstration. We view the task as a composite of specific actions, *primitives*, which can be easily recognized.

State Generation - To enable generalization over multiple demonstrations, the subtasks are modeled as states, describing the impact of a certain action to the current world state, e.g., *"Knife moved 10 cm to the right of the plate"*. The *state generation* block takes all demonstrations into account and searches for similar subtasks which are represented by the same state. The similarity is measured in terms of effects on the world state.

Task Generalization - This block is used to identify which states must occur before others and possibly which states that are irrelevant for the task goal. From a single demonstration, the task is carried out in the exact same order unless some prior knowledge is available. From multiple demonstrations, the robot acquires more knowledge about the task and achieves the goal by assembling its own action sequence from a combination of all demonstrations. An example of a constraint is that the plate has to be moved first, before food can be served on it.

Planning - The robot has to be able to plan task execution and reaching the goal state given the current state of the environment. Planning is also needed when a failure is detected and the task execution has to be replanned. In the task space, the robot first plans and then executes *which* objects must be moved *where* to achieve the goal given the task constraints.

Execution - Once a plan of how to reach the goal state has been generated, the execution of the task commences. Here, grasp planning and robust visual servoing play an important role for the task success. Currently there is no robot even near having the human capabilities of grasping objects, much because our superior sensory feedback. In our current work, we consider grasping of simple objects since the main focus is on planning and task model generation.

Perception - The level of task complexity that the robot is able to perform is strongly dependent on the sensory perception and modeling and it is usually a bottleneck of any PbD system. Pose estimation in general is a difficult problem, and in this work we settled for a simple vision system able to estimate the pose of cubic blocks.

IV. Implementation

For easier understanding of the system implementation details, let us study a specific task we want to teach a robot, *cutting-a-cucumber*. The follwing objects are considered in the task: a cutting board, a cucumber and a knife. Given that the objects' poses estimated, this tasks can be learned incrementaly as shown in Table. I. Here, object positions can be represented given either absolute coordinates or relatively to other objects already moved. This allows tasks to be executed with greater precision compared to a method based only on absolute coordinates. The demonstrated tasks are segmented, and each subtask is quantized to a state. A demonstration is then represented as a state sequence. Another example task used later on in the paper is *setting up a dinner table*. This task consists of placing plate, knife, fork, spoon, glass, food and napkin on a dinner table.

TABLE I

TASK *cut cucumber* AS MODELED IN OUR SYSTEM (z-AXIS ANTIPARALLEL TO GRAVITY).

Object	Relative Position	Relative Orientation	(x,y,z,θ,ϕ,ψ) Pose [cm, degrees]
Cutting board	None	None	(393, 123, 0, 0, 0, 0)
Cucumber	Cutting board	CuttingBoard	(10, 15, 1, 90, 0, 0)
Knife	Cucumber	Cucumber	(25, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(25, 0, 0, 90, 90, 0)
Knife	Cucumber	Cucumber	(25, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(24, 0, 6, 90, 90, 0)
Knife	Cucumber	Cucumber	(24, 0, 0, 90, 90, 0)
...	...	...	....

To decompose the task into subtasks, a method for automatic detection of a subtasks beginning and end is required. This is a largely open reasech problem and it is not considered in this work.

A. State Generation

Here, the continuous measurements are quantized from the operations. For the tasks considered in our work, the placement of certain objects can be defined relatively to other objects (*Place glass to the left of the main plate*) but some objects are to be placed to a specific point defined in absolute coordinates. To decide if the position should be regarded absolute or relative, we compute the minimum variance with respect to already placed objects:

$$relobj_i = \underset{\forall j \ moved}{\operatorname{argmin}} \ |cov(\mathbf{x_i} - \mathbf{x_j})| \quad (1)$$

where $\mathbf{x_i}$ is the position of object i. If $relobj_i = i$, then the position should be regarded as absolute. The same procedure is done for the orientation, meaning that an object can have a relative position to one object and a relative orientation to another object. For some tasks, there may be several positions that are valid for a certain object. A difficult problem is how to automatically decide when a position should be regarded as a new state, and when it should be regarded as a variation of an existing state. We use K-means clustering, [13], to quantize the position and orientation for a specific object into a number of subgroups. This quantization method is good even though the amount of data is low which is general the case in PbD systems. The optimal number of subgroups are the one which yields the lowest maximum variance. However, the clusters are not allowed to lie closer than a certain threshold to each other, to prevent the scenario of a single cluster for each measurement. The improved algorithm becomes:

$$relobj_i = \underset{\forall j \ moved, \ c \in [1,N_{demo}]}{\operatorname{argmin}} \ \max_{k=1}^{c} |cov(\mathbf{x_i^k} - \mathbf{x_j^k})| \quad (2)$$

The minimum maximum covariance is sought over all objects and cluster possibilities. Here, x_i^k denotes subset k when object i is clustered into c clusters. With this approach, we are able to identify multiple possible positions and orientations for a single object, e.g., for a *set table* task, the spoon can be either above or to the right of the plate.

B. Task Generalization

After the demonstrations have been abstracted to state sequences, the robot can analyze all sequences to build a general task model. Fig. 2 illustrates how this is done.

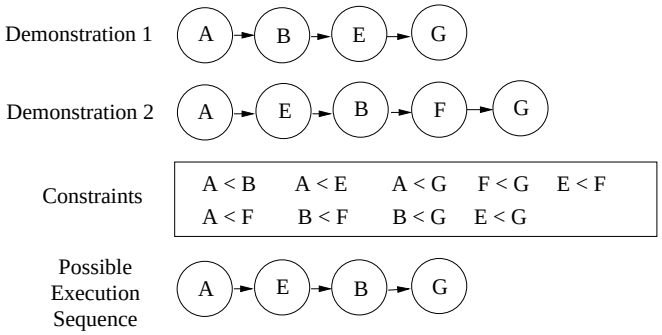

Fig. 2. Top: Two demonstrations given to the robot. Center: Nine constraints are identified. Note that state B and E are not constrained. Bottom: One of the possible sequences to follow at execution time

In this example, there are two demonstrations. From these, nine *constraints* are identified. Initially, all actions are constrained to the order they were demonstrated. When two or more constraints contradict each other, they are removed. Thus, in the example above the constraints $B < E$ and $E < B$ have been removed. The robot is then free to reach any of the goal states demonstrated, as long as it does not violate any of the constraints. As more demonstrations are added, the list is modified. It is important to note that this approach requires a *planner*, that calculates a sequence to achieve the goal under the constraints. However, this is not a drawback as the planner is needed anyway if an operation cannot be performed directly (target position may be blocked). Also note that as the sequence is calculated at run-time, the robot does not have to follow any of the human examples.

Another method for task generalization is presented in [11]. The method is based on the longest common subsequence (LCS) of the state sequences, and the LCS of several demonstrations constitute the generalize task model, in which the other actions appear as alternate paths. However, this approach is not suitable for manipulation tasks. Many pick-and-place tasks can be performed in arbitrary order, so the LCS for those tasks may be as short as a single state. Instead, we propose to build up a list of constraints that describes which states must occur before others.

Among the constraints generated in the example above, some are unnecessary, e.g., $A < G$, when the constraints $A < B$ and $B < G$ are present. These types of constraints can be removed, but they actually serve a purpose: they make the planning go faster. The planner does not have to try $G - A - B$, which is a dead end.

C. Planning

At execution time, the robot must be able to plan a sequence of actions to reach the demonstrated goal state. The objects to be manipulated are not necessarily at the same positions as during the demonstrations. Inspired by the STRIPS

planner, [14], our planner is based on operations which contain several preconditions and effects. These describe the changes on the world state if the operator is executed. The second part of a planner is the problem file, which is designed to reflect the current world state. The file contains all objects in the current scene and their locations and destinations. The grasp type for an object is selected automatically at run-time. We let each object have a couple of predefined grasps. For planning, a grasp must be chosen so that is does not cause collisions with the other objects. A grasp *g* at location *a* is not possible if there is a nearby location *b* occupied with an object, that would cause a collision when grasp *g* is applied to *a*. For all location pairs, the robot tests all grasps against all objects using a *path planner*. A path planner works at a lower level compared to a task planner, see [15] for more information. If the path planner fails to find a solution, then the location pair is marked with the grasp type that is not possible. The other objects are provided to the path planner as obstacles, with a slightly bigger size to account for motor inaccuracies in the execution phase. As the isolated gripping task is relatively simple, the planner quickly returns the solution if there is any. Using a path planner for planning the entire task is not feasible as the complexity of the task would make the planner search a very long time. Also, it is hard to accurately incorporate the dynamics of the actual grasping into the path planner.

This approach allows the robot to select the best grasp depending on the target pose of the object. The robot reasons much like a human being in this sense - if the object would be at the target location, would it then be graspable with the current grasp? If the workspace is cluttered with other objects, it may happen that to reach the goal state only one grasp position is possible. Below is an example of a problem described to our planner in XML. Each problem file starts with declaring all variables. The variable types are objects, locations and grasps. The objects are declared first:

```
<object name="fork" type="Object"/>
<object name="food" type="Object"/>
<object name="cup" type="Object"/>
<object name="box" type="Object"/>
```

Then several locations are declared. These correspond to both the starting and end poses of each object found during the demonstration. Also, a couple of free-space locations are given.

```
<object name="loc264" type="Location"/>
<object name="loc293" type="Location"/>
<object name="freespace1" type="Location"/>
...
```

The possible grasps are generated depending on which objects were found present in the scene:

```
<object name="grasp1" type="Grasp"/>
<object name="grasp2" type="Grasp"/>
...
```

After the variable declarations, the current state at run-time is provided. The state consists of the location and grasp types of each object, and the list of location pairs that may be in collision for certain grasp types.

```
<state>
  <arg name="fork loc10"/>
  <arg name="food loc15"/>
  <arg name="cup loc22"/>
  <arg name="box loc264"/>

  <arg name="fork grasp1"/>
  <arg name="food grasp2"/>
  <arg name="food grasp3"/>
  <arg name="cup grasp4"/>
  <arg name="cup grasp5"/>
  <arg name="box grasp6"/>

  <arg name="loc10 loc15 grasp1"/>
  <arg name="loc15 loc15 grasp4"/>
  <arg name="loc15 loc22 grasp5"/>
  <arg name="loc264 loc293 grasp1"/>
  <arg name="loc264 loc293 grasp2"/>
  <arg name="loc264 loc293 grasp4"/>
  <arg name="loc264 loc293 grasp6"/>

  <arg name="hand-empty"/>
</state>
```

The last part of the problem file describes the desired goal state:

```
<goal>
  <arg name="fork loc35"/>
  <arg name="food loc264"/>
  <arg name="cup loc293"/>
</goal>
```

To solve the problem, the planner uses two operators: GRASP and PUT_DOWN. Using a breadth-first search, the shortest solution that does not cause any collisions is found:

```
GRASP box loc264 grasp6
PUT_DOWN box freespace1 grasp6
GRASP food loc15 grasp2
PUT_DOWN food loc264 grasp2
GRASP fork loc10 grasp1
PUT_DOWN fork loc35 grasp1
GRASP cup loc22 grasp4
PUT_DOWN cup loc293 grasp4
```

V. Experimental Results

We evaluate the performance of the system in two different experimental settings. The first experiment is in a virtual environment and shows how the state generation block operates on multiple demonstrations to generate state representations. The second experiment is shown in a real *micro-world*, consisting of cubic blocks. The experiment shows the task generalization over multiple demonstrations, and also the robot execution and planning capabilities.

A. Experiment 1: Virtual Environment

In this experiment, the *set table task* described in Section IV is considered. The task was demonstrated by the user three times in a virtual environment where each object was only allowed to be moved, not rotated. The state of each object can then be represented using only two degrees of freedom (table plane is assumed known) which makes this experiment easy to analyze.

Fig. 3 shows the result of each demonstration. Demonstration 1 and 3 were similar but the objects were not moved in the same order. Demonstration 2 was different because of the spoon being put to the right of the plate, instead of behind it. Table II shows the states generated by the system. As expected, the knife, fork and napkin are specified relative to the plate. Because the glass position varied too much relative to the plate, its position is specified in absolute coordinates. The system correctly identifies the two possible placements

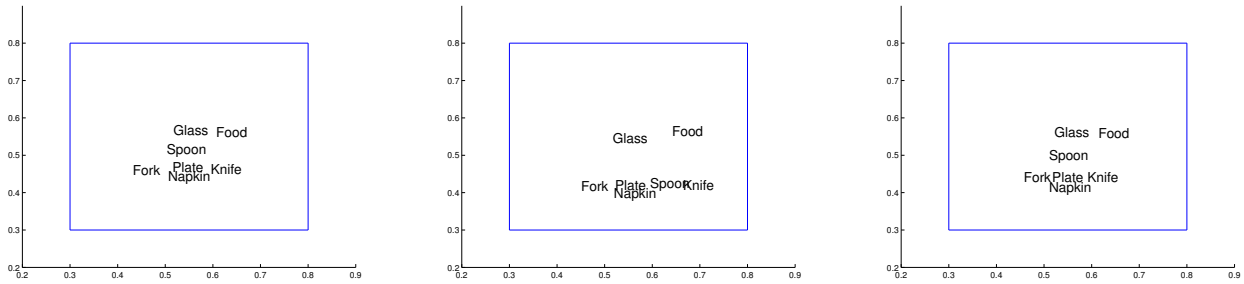

Fig. 3. The three demonstrations in Experiment 1.

of the spoon (state D and H). State J arises since the fork actually has lower variance towards the glass compared to the plate. In the first demonstration the fork was put down before the glass and positions can only be specified towards already placed objects. Table III shows the generated state sequences for the demonstrations. From these, a total of 32 constraints were identified. In this example, the constraints make sure that when an object is to be placed, its 'relative' object is already in desired position.

TABLE II

EXPERIMENT 1: THE STATES GENERATED FROM THE DEMONSTRATIONS.

State	Object	Relative Position	(x,y)
A	Plate		(0.52, 0.44)
B	Knife	Plate	(0.1, 0)
C	Fork	Plate	(-0.07, 0)
D	Spoon	Knife	(-0.08, 0.04)
E	Glass		(0.52, 0.56)
F	Food		(0.62, 0.56)
G	Napkin	Plate	(-0.01, -0.02)
H	Spoon	Plate	(0.02, 0.04)
I	Knife	Spoon	(0.08, -0.04)
J	Fork	Glass	(-0.07, -0.12)

TABLE III

THE STATE SEQUENCES FOUND IN THE DEMONSTRATIONS.

Demonstration 1	A-B-C-D-E-F-G
Demonstration 2	A-E-H-I-J-F-G
Demonstration 3	A-C-B-F-E-D-G

B. Experiment 2: Blocks World

In the second experiment, the operator demonstrated a task in an environment consisting of colored blocks. There were three equal-sized blocks of different colors available, which we labeled 'red', 'green' and 'wood'. The task was to stack a pyramid of blocks according to Fig. 4, with the red block on top. The purpose of the experiment is to show that the robot can acquire knowledge about the world from human demonstrations, in this case that the red block has to be placed last. Thus, such facts does not have to be

Fig. 4. Left: The task demonstrated by the human operator. Center: The state of the environment at run-time. Note that the parallel-jaw gripper can only grasp one of the blocks. Right: The robot is placing the final block on top of the others.

incorporated into the planner. The experiment will also show the capabilities of our proposed task planner. At execution time, the blocks were organized so that they block each other, and only one block is graspable to begin with.

In the experiment, we use an ActiveMedia PowerBot robot platform with an 6DOF arm on the top and a single camera mounted on the robot gripper. The robot first moved its arm to a top view of the scene to estimate the pose of each block. For this experiment it is sufficient to estimate the (x,y,θ) parameters, i.e., the position and orientation in the plane. The height of the table z was known. A four-dimensional Hough transform was used to estimate (x,y,θ,s) of each block, s being the length of a side of a block. As we do not use stereo vision, we cannot measure the z-position of blocks stacked on each other. Instead, we estimate the position of each block every time a block is moved. If a block that was visible change to fully or partly occluded, it is concluded that the last block moved was placed upon that block.

For this experiment, a path planner was not necessary to check for collisions. Instead, we used a simple approach: Location a is colliding with location b for grasp type g if $|\mathbf{b_p} - (\mathbf{a_p} \pm \hat{\mathbf{g}_p})| < r + g_r$, where r is the radius of a block, g_r is the space required for the robot gripper, also set to r, $\mathbf{p}$ denotes the corresponding translation vector, and $\hat{\mathbf{g}_p}$ is $\mathbf{g_p}$ rotated with the same rotation vector as a. For all blocks, there are two valid grasps.

Two demonstrations were conducted. First in the order wood - green - red, then in the order green - wood - red. Table IV shows the states generated from these demonstrations, and Table V shows the state sequences. Note that although it would be nice if the second state sequence was B-A-D, this is not possible as state B needs the coordinates of the wood block, which is not moved yet in demonstration 2.

TABLE IV

EXPERIMENT 2: THE STATES GENERATED FROM THE DEMONSTRATIONS.

State	Object	Relative Position	Relative Orientation	(x,y,z,θ)
A	Wood			(0.34, 0.04, 0.00, 0)
B	Green	Wood	Wood	(0.06, 0.00, 0, 0)
C	Red	Wood	Wood	(0.03, 0.00, 0.04, 0)
D	Green			(0.40, 0.04, 0.00, 0)

TABLE V

EXPERIMENT 2: THE STATE SEQUENCES FOUND IN THE DEMONSTRATIONS.

Demonstration 1	A-B-C
Demonstration 2	D-A-C

From the two demonstrations, four constraints were generated. The only real constraint is that *C* should occur last, but due to the few demonstrations, some other constraints follow as well. The center of Fig. 4 shows the status of the blocks at execution time. As seen, only the red block is graspable. The robot generates two plans, one for each demonstration. Of these, the plan for the first demonstration is five steps, while the plan for second demonstration is only four steps: *Move red away, move green to position, move wood to position* and finally *move red on top of green and wood*. The right part of Fig. 4 shows the robot executing the final subtask. Several other experiments were conducted to test the task level planning with different setups and up to five blocks. Most of the time, the robot executed planned and executed the task without any problems, although some specific setups caused problems. Two types of faults occured. First, the simple vision system sometimes had trouble identifying the number of blocks and their pose if their initial positions were too close to each other. It is because the voting maxima in the Hough space interfere with each other. Second, the planner had no model for the camera mounted on the hand. Normally the camera, mounted above the gripper, is in no risk of colliding with the blocks, but when stacking several blocks on top of each other a new high obstacle is created that may collide with the camera.

VI. CONCLUSION

In this paper, we dealt with the problem of learning robot tasks from multiple demonstrations in a Programming by Demonstration setting. Classically, this type of robot programming has been based on individual task demonstrations.

A generalized task model is represented by the goal states of the demonstrations, and the task constraints. At run-time, the task model is used to build a task sequence using a planner. Experiments are used to show the generated task model in two different settings. In one of these, a robot is used to reproduce the result of the human demonstrations.

We also presented a method for performing task level manipulation planning. This allows the robot to execute the task although the current setup is not the same as the one in which the training was performed. The task planner was tested in a simple environment with equal sized objects. The next challenge will be to use several objects with different size and shapes, which require a more powerful perception system.

The system scales up well, although with more objects present, more demonstrations are needed to give the robot the desired flexibility. If too few demonstrations are provided, many unnecessary constraints will be encoded in the task description.

In the current system, it is assumed that there is only one object of each type. If there are two identical blocks present in the second experiment, the system gets confused since this knowledge is not explicitly defined. Our current research is devoted to solving this problem.

REFERENCES

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

[2] C. Atkeson and S. Schaal, "Robot learning from demonstration," in *In Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97) (ed. D. H. Fisher Jr.)*, pp. 12–20, July 1997.

[3] M. J. Matarić, "Getting humanoids to move and imitate," in *IEEE Intelligen Systems*, pp. 18–24, jul 2000.

[4] K.Ogawara, J.Takamatsu, K.Kimura, and K.Ikeuchi, "Generation of a task model by intergrating multiple observations of human demonstrations," in *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, pp. 1545–1550, May 2002.

[5] H. Friedrich, R. Dillmann, and O. Rogalla, "Interactive Robot Programming Based on Human Demonstration and Advice," in *Sensor Based Intelligent Robots*, pp. 96–119, 1998.

[6] M. Ehrenmann, O. Rogalla, R. Zllner, and R. Dillmann, "Teaching service robots complex tasks: Programming by demonstration for workshop and household environments," in *Proceedings of the 2001 International Conference on Field and Service Robots(FSR)*, pp. 397–402, 2001.

[7] J. Aleotti, S. Caselli, and M. Reggiani, "Leveraging on a virtual environment for robot programming by demonstration," in *Robotics and Autonomous Systems, Special issue: Robot Learning from Demonstration*, vol. 47, pp. 153–161, 2004.

[8] S. Ekvall and D. Kragic, "Grasp recognition for programming by demonstration," in *IEEE/RSJ International Conference on Robotics and Automation, ICRA'05*, 2005.

[9] S. Ekvall and D. Kragic, "Integrating object and grasp recognition for dynamic scene interpretation," in *IEEE International Conference on Advanced Robotics, ICAR'05*, 2005.

[10] P. Jensfelt, S. Ekvall, D. Kragic, and D. Aarno, "Integrating slam and object detection for service robot tasks," in *IEEE International Conference on Intelligent Robots and Systems, IROS'04, Workshop on Mobile Manipulators: Basic Techniques, New Trends and Applications*, 2005.

[11] M. N. Nicolescu and M. J. Matarić, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the Second International Joint Conference on Autonomous Agens and Multi Agent Systems*, 2003.

[12] H. Friedrich, R. Dillmann, and O. Rogalla, "Interactive robot programming based on human demonstration and advice," in *Christensen at al (eds.):Sensor Based Intelligent Robots, LNAI1724*, pp. 96–119, 1999.

[13] J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 1:281–297, University of California Press, 1967.

[14] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence 2*, pp. 189–205, 1971.

[15] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *Proceedings of the International Conference on Robotics and Automation*, pp. 521–528, 2000.