ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

Robot Learning from Demonstration: A Task-level Planning Approach

Staffan Ekvall and Danica Kragic
Computational Vision and Active Perception Lab
Centre for Autonomous Systems
School of Computer Science and Communication
Royal Institute of Technology, Stockholm, Sweden
E-mail: dani@kth.se

Abstract: *In this paper, we deal with the problem of learning by demonstration, task level learning and planning for robotic applications that involve object manipulation. Preprogramming robots for execution of complex domestic tasks such as setting a dinner table is of little use, since the same order of subtasks may not be conceivable in the run time due to the changed state of the world. In our approach, we aim to learn the goal of the task and use a task planner to reach the goal given different initial states of the world. For some tasks, there are underlying constraints that must be fulfille, and knowing just the final goal is not sufficient. We propose two techniques for constraint identification. In the first case, the teacher can directly instruct the system about the underlying constraints. In the second case, the constraints are identified by the robot itself based on multiple observations. The constraints are then considered in the planning phase, allowing the task to be executed without violating any of them. We evaluate our work on a real robot performing pick-and-place tasks.*
Keywords: *Programming by demosntration, task learning, task planning, object manipulation*

1. Introduction

Robot task learning has during the past years received significant attention [1]–[10] and it has been recognized that more natural programming interfaces are necessary to allow ordinary users to teach robots new tasks. Motivated by the fact that imitation enables humans to easily learn new skills, the robotics community has taken upon this idea and used it in the design of some of the recent task learning systems for robots. From the task learning in humans it is known that such a strategy where a teacher's demonstration is used as a starting point significantly speeds up the learning process. In robotics, this approach has been demonstrated in frameworks of Learning by Imitation and Programming by Demonstration (PbD), [11], [12] where sensory based task representation and task analysis have been recognized to represent the basis for development of robust and flexible learning systems. For the work presented here, we consider a service robot scenario in which we want to enable a robot to learn and refine representations and understanding of complex domestic tasks. The robot should be able to learn either by a direct learning process with a human teacher, or through observation by extracting new information and learn new skills just by looking at human. A PbD interface can be used to interpret teacher's demonstration and generate control commands required by the robots for completing the task. Such a programming interface is natural for humans since it does not require any programming skills and can potentially be used to program very complex tasks. Naturally, an important issue to deal with is that the 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 than originally demonstrated. Hence, it is not sufficient to just replicate the human movements but the robot must have the ability to recognize what parts of the whole task can be segmented and considered as subtasks so to perform online planning for task execution given the current state of the environment.

The main contribution of this work is the use of a task planning approach in combination with robot learning from demonstration strategies. The important problem considered 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, in a table setting scenario, 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, additional contributions of this paper are the state generation and constraints identification methodologies based on multiple human demonstrations. The proposed methodologies are evaluated in the framework of robotic object manipulation tasks. Learning such tasks is

considered a hard problem since robots have 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 which is not the 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. Here, we have chosen the latter alternative since it allows the robot to learn from multiple observations and improve the task model over time.

The paper is organized as follows. In Section 2 we motivate our work and briefly present some alternative solutions given in the literature. Section 3 describes the pose estimation method, and Section 4 presents the task planning method. Section 5 describes how the system is able to generalize and learn from multiple observations. The methods are evaluated in Section 6. Finally, Section 7 concludes the work.

2. Motivation and related work

For humans, one of the fundamentals of social behavior is the understanding of each others intentions, skill transfer and learning through interaction and observation. Skill learning in humans have been well studied and most common forms of teaching are observation, physical guidance, demonstration and verbal instruction, [13]. When learning a new task, the viewer usually does not know which details are important for the task outcome and that the appropriate level of detail should be provided for successful learning, [13]. In a similar way, the robot task learning have to be made easy and flexible. Some of the open questions in robot task learning are:

- How should the robot be instructed complex tasks when the temporal order for some of the subtasks is important but unimportant for others?
- How should objects and actions that can be performed on them be represented?
- Should the task goal be represented by the final goal state or should it be learned by considering temporal dependencies between the subtasks?
- How does the representation and number of demonstrations facilitate the generalization of tasks?

In general, we would like to teach the robot some useful tasks such as 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, [14]. For this task, the robot is required to recognize objects, grasp them and put them on the table in specific 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 as, for example mail delivery [15] include constraints such as that the mail has to be collected before it can be delivered but the order of delivery may be irrelevant. Hence, for some of the tasks a specific order of subtasks is required and for some it is not. In the work presented here, the problem of learning tasks that include object manipulation is solved by identifying the goal state and the spatio-temporal constraints of the task. Many of the current robot instruction systems that deal with programming by demonstration are 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 multiple times. In other words, we need a task level learning and planning system that builds constraints automatically identified from multiple demonstrations. This problem has previously been considered in regard to suboptimality in demonstration [12], [16] where different sources of sub-optimality have been recognized: where the human demonstrates unnecessary, incorrect or unmotivated actions; where there is a choice of scenario regarding when to apply an action; where the user does not know enough about the task and thus the actions are demonstrated wrongly. Some of the solutions have been studied in [12] where the sub-optimality on the task-level in considered as noise and removed before any programming of the robot takes place. Differently from the work presented in [12], [17] that generates plans autonomously using geometric properties of objects or instructions provided by the demonstrator, we deal with learning and refinement of high-level tasks based on a set of underlying capabilities already available to the robot. In particular, we are interested in evaluating scenarios where the robot is able to reproduce a task based only on a desired outcome or final goal of the task, preferably also generalizing from multiple demonstration trials. Similar problem has been studied in a robot navigation scenario, [18] 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 the work presented here, 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 [5], generation of task models based on multiple human demonstrations is presented. *Essential interactions* that represent the important hand movements during a manipulation task are identified. Then the relative trajectories corresponding to each essential interaction are generalized by calculating their mean and variance. High variance means arbitrary motion is

allowed, while low variance means strictly precise motion is required. The learned trajectories are stored in the task model, which is used to reproduce a skilled behavior. It is important to point out that trajectories are related to the essential interactions with the manipulated objects meaning that many different trajectories corresponding to the same object manipulation are represented. This makes the method 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.

3. System description

The general outline of the system is shown in Fig. 1. The teacher demonstrates the task and and the robot makes observation based on visual input. After a learning trial, explained in more detail later on, the robot first plans and then executes the task using visual input and grasp planning. In the current system, task learning can be performed in three different ways:

- **Imitation Learning**

The term *imitation learning* is commonly used to represent task learning at a low level by considering reproduction of trajectories and/or robot joint configurations, [19]–[21]. In our work, with the imitation learning we denote the task reproduction process where the robot is given only the task goal and it tries to achieve the same result.

- **Learning in Dialogue with Teacher**

In human teaching, it is common that the teacher demonstrates the task once, while continuously explaining each step. We investigate a similar approach by allowing the teacher to add some constraints to the task while performing it. Hence, the robot is explicitly instructed what not to do and it is thus able to avoid solving the task in a wrong way.

- **Generalizing from Multiple Observations**

We also believe that the robots should be able to improve or learn new tasks not only through a direct teaching process but also by observing humans performing tasks in everyday settings. Multiple observations of the same task can be utilized to form a more general and thus flexible model of the task by autonomously identifying the spatio-temporal constraints of the subtasks or detect the irrelevant subtasks.

In this paper, we model and evaluate all of the above approaches. The experimental platform is a PowerBot from MobileRobots Inc, see Fig. 5. The robot is equipped with a 6DOF robotic manipulator on the top that is here used to demonstrate the task learning and planning processes. At the last joint of the arm, there is a firewire camera used for automatic pose estimation of the objects. In this paper, we work with polyhedral objects but the presented methodology can be applied to a large set of shapes as long an accurate pose estimate is available. The next section shortly describes the pose estimation process.

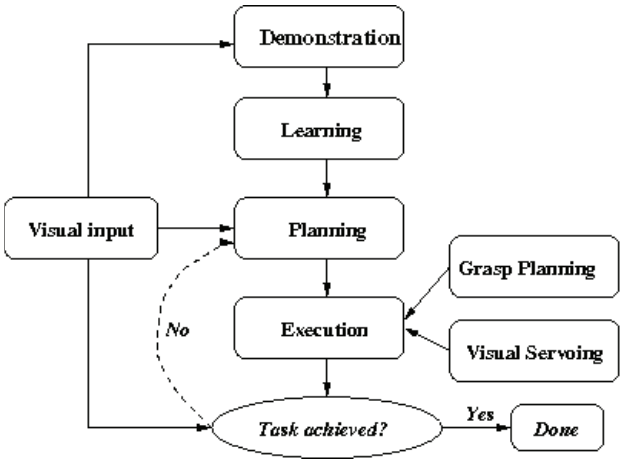

Fig. 1. Integration of task learning from demonstration and task level planning.

3.1 Pose estimation

As stated, pose estimation of objects is performed automatically. The objects are first modeled using a set of geometric primitives as shown in Fig. 2 where only the size has to be known in advance. In the simplest case, the primitives are the apparent object edges modeled using points, lines and polygons defined both in the camera (3D) and image (2D) space. Given the current pose of the object, hidden primitive removal is performed using back face culling [22].

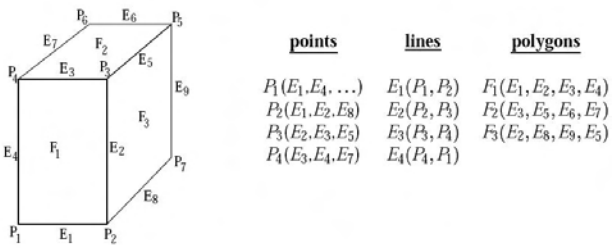

Fig. 2. An object is represented with points, lines and polygons.

Due to the rich textural properties of the object, pose estimation cannot be performed by using solely the outer contours of the object. This is why in an off-line learning stage, we store a single file representing a set of SIFT points originally presented in [23]. With the stored image, we also store the pose of the object corresponding to that particular view of the object. At run time, SIFT feature detector is applied to the whole image. The detected features are then matched to the stored set of points defined for each of the objects. For planar objects, a homography based matching with robust outlier rejection (RANSAC) is used for pose estimation, as presented in [24]. Here, for each point on the surface, it is enough to know which facet it belongs to thus the exact 3-D position of each individual point is not required. Since the pose of the object for the stored view is known, the problem of scale ambiguity related to homography decomposition is easily solved. Examples of the pose estimation process are shown in Fig. 3 and Fig. 4.

4. Task level planning

Task planning is the problem of finding a sequence of actions to reach a desired goal state, [25]. This is a classical AI problem that is commonly formalized using a suitable language to represent task relevant actions, states and constraints.

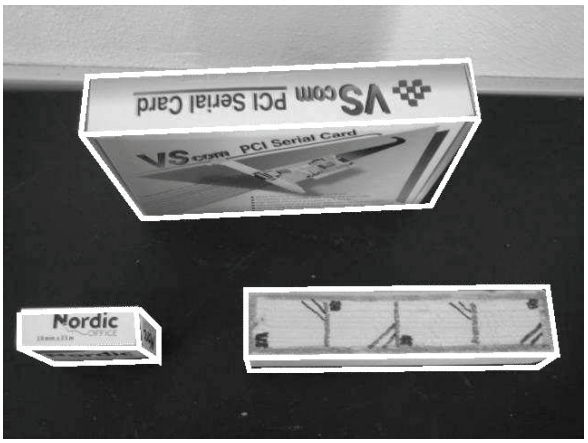

Fig. 3. Examples of estimated poses overlaid in white.

Fig. 4. Examples of estimated poses overlaid in white.

The robot has to be able to plan the demonstrated task before executing it if the state of the environment has changed after the demonstration took place. The objects to be manipulated are not necessarily at the same positions as during the demonstration, and thus the robot may be facing a particular starting configuration it has never seen before. Our task planner is inspired by the STRIPS planner, [26] and is based on operations which contain several preconditions and effects. These describe the changes of the world state once an action has been executed. The second part of the planner is the problem file, which is designed to reflect the current world state. The file contains all objects in the current scene, their locations and task defined destinations. It is automatically generated at run-time. The domain and problem is provided to the planner in an XML format. The domain is formalized using first order logic, which allows us to model the changing state of the world. Currently, there are only two types of operations available to the planner, defined below.

Definition 1. Domain Operations

- *pickUp(o, l, g)* - Grasp the object *o* at location *l* using grasp type *g*.
- *putDown(o, l, g)* - Put down the object *o* at location *l* using grasp type *g*.

The world state is described with a list of predicates:

Definition 2. Domain Predicates

- *handEmpty* - indicates whether the hand is empty or not.
- *holding(o)* - indicates if the hand is currently holding object *o*.
- *objAtLoc*(o, l) - indicates if location *l* is occupied by the object *o*.
- *graspAble*(l, g) - indicates if the object at location *l* is graspable with grasp type *g*.
- *collision(l1, l2, g)* - indicates if a collision would occur if a grasp *g* is applied to location *l1*, and both locatio *l1* and *l2* are occupied.
- *mustOccurBefore(o1, l1, o2, l2)* - A special predicate to handle temporal constraints. Indicates if object *o1* must be placed at location *l1* before object *o2* can be placed at location *l2*.

Below we define the preconditions and effects for the operators.

Operation *pickUp(o, l, g)*

Precondition:

handEmpty^objAtLoc(o, l)^graspable(l, g) ^ ¬ ∃ (obj, loc) (collision(l, loc, g)^objAtLoc(obj, loc))

Effect:

holding(o) ^ ¬ handEmpty ^ ¬ objAtLoc(o, l)

Operation *putDown(o, l, g)*

Precondition:

holding(o)^graspable(l, g) ^ ¬ ∃(obj)^objAtLoc(obj, l) ^ ¬ ∃ (obj, loc) (collision(l, loc, g)^objAtLoc(obj, loc)) ^ ¬ ∃ (obj, loc) mustOccurBefore(obj, loc,o, l)

Effect:

handEmpty ^ ¬ holding(o) ^ objAtLoc(obj, loc) ^ ∀ (obj, loc) ¬ mustOccurBefore(o, l, obj, loc)

The grasp type for an object is selected automatically at run-time, based on the predicates provided to the planner by the vision system. We operate with a limited number of predefined grasps. For planning, a grasp must be chosen so that it does not cause collisions with other objects, and also so that it is within the robot's reach. A grasp *g* at location *a* is not possible if there is a nearby location *b* occupied by another 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 for collisions and reachability. This approach allows the robot to select the best grasp depending on the target pose of the object.

The locations are limited to the source and target locations of each object, plus a number of additional free locations that can be used for freeing up the workspace. As an example, the source location for a box is labeled *loc box s*, and similarly the target location is labeled *loc box t*. Section 6.1 provides an example of how the planner operates. We note here that more advanced logical languages will be considered once more complex tasks are to be modeled. One example is the Linear Dynamic Event Calculus proposed in [27], a logical language that combines aspects of situation calculus with linear and dynamic logics.

4.1 Detecting object collisions

The work here relates also to the path planning problem, [28]. Compared to the task level planning, a path planner searches for a path in robot's configuration space to reach a desired configuration while avoiding obstacles, self-collisions, etc. In contrast, a task planner performs high level operations and relies on an existing low-level robot controller to carry out the necessary operations. Using a path planner for planning the entire task is not feasible as the complexity of the task would make the planner infeasible. Also, it is hard to accurately incorporate the dynamics of the actual grasping into the path planner. For task planning, it is however important to also consider the reachability of the robot. While a path planner only explores locations within the robot's workspace, a task planner operates in the task space and must at each step check that the specific world location is reachable. The robot used in this work has a very limited workspace, as shown in Fig. 5

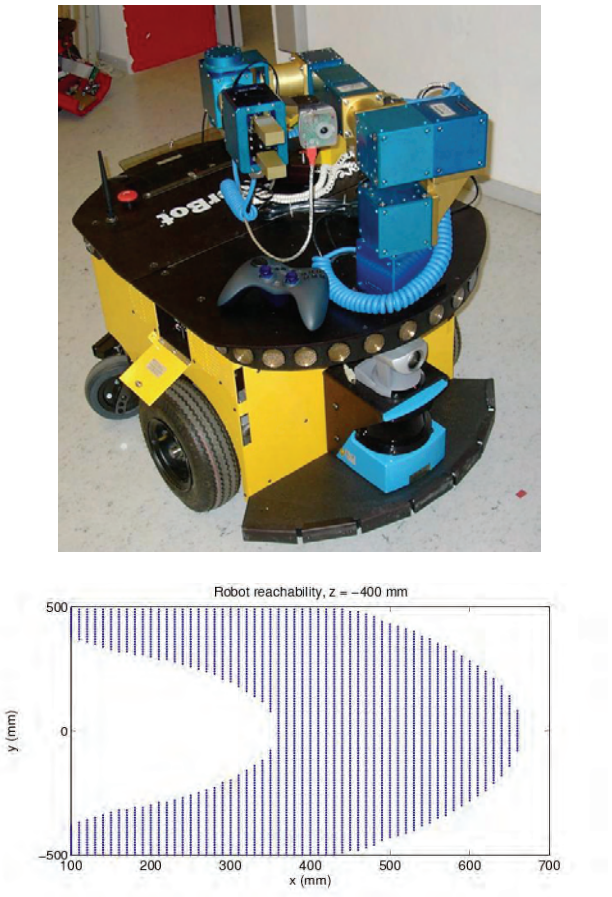

Fig. 5. Top: The robot used in the experiments. It has a 6-DOF arm with a parallel-jaw gripper. Bottom: The workspace of the robot, visualized for the plane 40 cm below the base cube, where the experiments were conducted.

Before the planner is initiated, all possible object collisions that can occur during task execution are evaluated. This is a fast process for settings with a few objects. If many objects are involved in a task, techniques that limit the number of collision checks may have to be utilized, e.g., only checking nearby locations for collision. Fig. 7 visualizes the data available to the robot before performing collision checks. For each object, there are a number of grasping configurations that can be applied to each object and a suitable one is chosen based on the initial and destination position of the object and the current position of other objects in the environment, see Fig. 6. A collision check results in a list of grasps that are applicable to the object in the initial location, if its destination location is currently occupied. The collision check is performed in three steps. First, it is tested whether the two objects would collide with each other. In that case, no grasp is possible. If not, each grasp for the object in the first location is checked for collision with the object in the second location. If the grasp still does not cause a collision, it is checked whether the robot can actually perform this grasp considering its kinematic constraints and limited reachability.

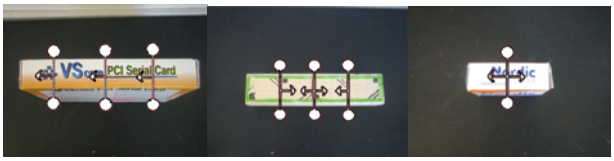

Fig. 6. The predefined grasp types for the different objects. The box has three grasps, the wooden block has four, and the tape only two.

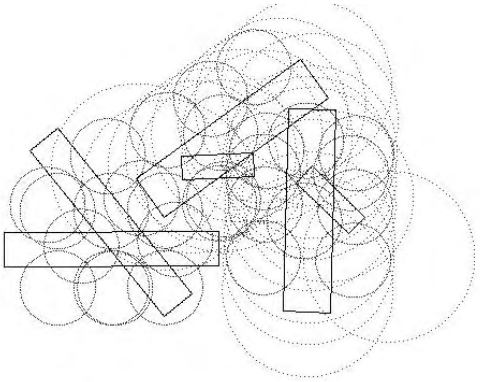

Fig. 7. A visualization of what the robot sees and knows. The thick lines are the object positions including both their current position and their target position from the images seen in Fig. 4. The dotted circles are the collision spheres from the different grasps. The small circles origin from the gripper fingers, while the large circles arise from the gripper itself and the camera mounted on the end effector, see Fig. 8. These larger spheres are on a higher altitude and only cause collisions if a small object is being grasped next to a tall object.

The collision checks are currently performed as following: two objects collide if their bodies occupy the same part of the environment. This check is done in 2D since all objects are assumed to be placed on a table in a vertical position. A grasp collides with an object if any of its collision spheres intersect with the object's box. Since the robot

Fig. 8. The choice of grasp is very important. Here, the camera has not been modeled and thus the robot tries to grasp the wooden block. However, this causes a collision between the tall box and the camera.

knows the height of each object and each sphere has an altitude, this collision check is done in 3D. A grasp may have several collision spheres. For the parallel-jaw gripper we are using, there are one small sphere for each of the fingers, one large sphere on a higher altitude for the hand, and another large sphere to model the camera mounted on the hand. Fig. 8 shows a typical error that occurs if the camera is not modeled - there is a collision between the camera and one of the objects. Thus, the robot will have to plan grasps such that the camera does not collide. Due to the large camera and the limited workspace, the planning task is not trivial.

4.2 Finding free space

The initial and destination locations of objects are usually not enough for the planner to solve the task. The workspace is narrow and for many tasks the robot needs free space to unload objects temporarily. These locations are found by searching for a location that has the fewest number of collisions as visualized in Fig. 7. This location is then treated as any other location and any possible collision that may still occur is provided to the planner. Since searching every possible location would be too time consuming, the search is limited to every 5 cm and every 60 degrees rotation. This yields 210 possible locations, but this can easily be increased at the expense of increased search time.

4.3 Taking profit from human advice

We have also modeled tasks in which the human has the possibility to instruct the robot that one of the objects (*tape*) has to be manipulated (*placed on the table*) before any other object. We note here that there is currently no verbal communication - instead the expected result of such a system was encoded in the planner. The instruction "The tape should be placed first" adds two constraints: the tape should be at its target location before the box, and also before the block. These are fed to the planner as predicates, *mustOccurBefore(tape, loc_ tape_t, box, loc_ box_ t)*.

5. Automatic generalization from multiple examples

If the teacher does not instruct the robot about the constraints of the task directly, the robot should still be able to detect these by observing the task performance several times. Beside this, in the current system the robot can calculate the average location of each object and does not need to place them at the exact locations from a particular demonstration. The ultimate goal is to have a robot autonomously moving around in the environment, observing humans performing their everyday tasks and so learn new or update models of the existing tasks. Then, without further instruction, the robot can ideally acquire the knowledge to perform the tasks itself. With the work presented here, we aim to automatically identify the underlying constraints of the task. Generalizing from multiple examples requires more components than when simply imitating a human task. As Fig. 1 shows, the necessary steps are:

Segmentation - The segmentation of the task into isolated operations or *primitive tasks* is a research issue that has been studied before, [1], [2], [4], [11]. 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 that 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 that are represented by the same state. The similarity is measured in terms of effects on the world state.

Task Generalization - This process 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.

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)
...	...	...	....

Table 1. Task *cut cucumber* as modeled in our system (z-axis anti-parallel to gravity).

5.1 Example task

Let us study a specific task we would like to teach a robot, *cutting-a-cucumber*. The following objects are

considered in the task: a cutting board, a cucumber and a knife. Given that the objects' poses are estimated, this task can be learned incrementally as shown in Table 1. Here, object positions can be represented given either absolute world coordinates or relative to other objects already manipulated. 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. The task segmentation process is performed manually.

5.2 State generation

In this section we provide more information of how continuous measurements are quantized from the operations. For the tasks considered in our work, the placement of certain objects can be defined relative to other objects (*Place glass to the left of the main plate*) but some objects are to be placed to a specific position defined in absolute coordinates, i.e. robot centered coordinate system or some world coordinate system. To decide if the position should be regarded absolute or relative, we compute the minimum variance with respect to already placed objects:

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

where **x** represents the position of an object. If *relobj* = *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 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 in general the case in PbD systems. In detail, each position can be considered as a point in N-dimensional space, N being the number of DoFs for the object. If the same object is placed at approximately the same location in several demonstrations, the corresponding points will lie close to each other. K-means clustering automatically finds groups of points, which can then be labeled as the same state. The optimal number of subgroups is the one that 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:

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

The best value is sought over all objects and cluster possibilities. Here, **x** denotes subset **k** when object **i** is clustered into *c* clusters. With this approach, we are able to identify multiple suitable 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.

5.3 Task generalization

After the demonstrations have been abstracted to state sequences, the robot can analyze all sequences to build a general task model. Fig. 9 illustrates how this is done.

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. We then utilize our planner to calculate a sequence of actions to achieve the goal under the constraints. Note that as the sequence is calculated at run-time, the robot does not have to follow any of the human examples.

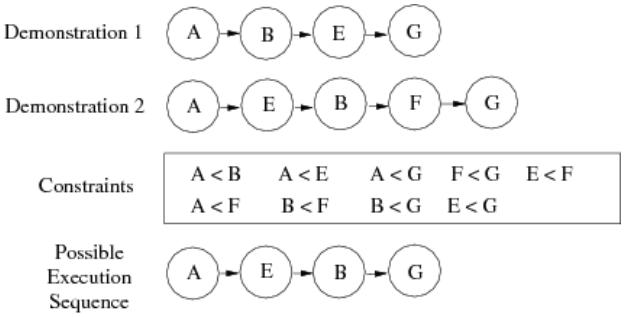

Fig. 9. Top: Two demonstrations given to the robot. Center: Nine constraints are identified. Bottom: One of the possible sequences to follow at execution time.

Another method for task generalization is presented in [18]. The method is based on the longest common subsequence (LCS) of the state sequences, and the LCS of several demonstrations constitute the generalized task model, in which the other actions appear as alternative 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.

6. Experimental evaluation

Throughout the experiments presented here, we have operated in three dimensions. Each object is placed on a

table and its height is known. The robot has a limited workspace, about 40x20 cm as indicated in Fig. 5, so the choice of grasp type and move order sequence is crucial for the task success.

6.1 Planning example

Here we give an example of how the planner operates. We ran the planner on the two test images seen in Fig. 4. The list of predicates generated consists of three parts. First, the collisions are listed:

collision(loc box s, loc box t, box left)
collision(loc box s, loc box t, box center)
collision(loc box s, loc box t, box right)
collision(loc block s, loc tape s, block center) ….

Then, the reachability of each grasp is listed:

graspable(loc_ box_ s, box_ left)
graspable(loc_ box_ s, box_center)
graspable(loc_ box_ s, box_ right)
graspable(loc_ block_ s, block_ center) ….

Finally, the location of each object is listed:

objAtLoc(box, loc_ box_ s)
objAtLoc(block, loc _block_ s)
objAtLoc(tape, loc_ tape_ s)

The goal state is listed separately as:

objAtLoc(box, loc_ box_ t)
objAtLoc(block, loc_ block_ t)
objAtLoc(tape, loc_ tape_ t)

The planned solution is then:

pickUp(box, loc_ box_ s, box_ center)
putDown(box, loc_ box_ t, box_ center)
pickUp(tape, loc_ tape_ s, tape_ center_ 180)
putDown(tape, free1, tape_center_ 180)
pickUp(block, loc_ block_ s, block_ center)
putDown(block, loc_ block_ t, block_ center)
pickUp(tape, free1, tape_ center)
putDown(tape, loc_ tape_ t, tape_ center)

Thus, the system identifies a free location and uses it to store the tape at while the other objects are moved into place. The planner chooses grasps and a sequence of movements such that the task can be completed without collisions.

6.2 Imitation learning

To evaluate the imitation learning and the overall performance of the system, we placed the three objects in six different configurations according to Fig. 10. For each configuration, the robot was asked to reconfigure the objects to one of the other configurations. In total, the robot had to plan and execute 30 tasks using its 6-DOF arm and parallel-jaw gripper. In the imitation setting, a "demonstration" is simply an image of the target configuration. However, the starting configuration varied slightly for each experiment since it is not possible to place the objects exactly according to the image. Table 2 shows the results.

The table should be interpreted as follows.

- S - success, all objects were successfully moved to their new position.

Source Conf. \ Target Conf.	1	2	3	4	5	6
1	-	F2	F3	F3	F1	S
2	S	-	S	S	F1	F1
3	F3	F3	-	S	F1	F3
4	S	F2	S	-	F1	S
5	F1	F1	F1	F1	-	F1
6	S	F2	S	S	F1	-

Table 2. The outcome of each of the tasks in the experiments.

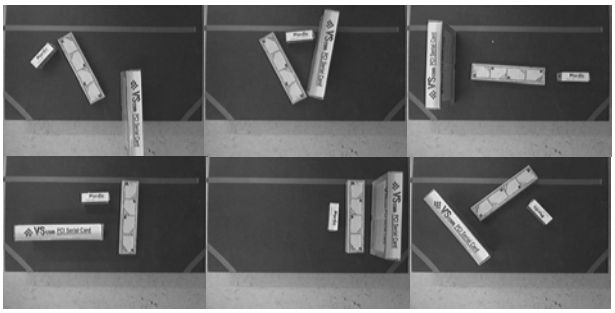

Fig. 10. The six different object configurations used in the experiments, seen from the robot's point of view.

- F1 - failure type 1. One of the objects in the source or target configurations was not found.
- F2 - failure type 2. The grasping of an object was not as expected, which caused collision when it was put down. For example, the wood block is heavy and may slip, or the box may tilt when being grasped.
- F3 - failure type 3. Imprecise pose estimation caused the gripper to collide with the object when grasping.

In this experiment, 11 of 30 tasks were successfully completed. Of the 19 failures, 16 occurred due to pose estimation errors, that is, due to the imperfect visual input. Since we only use one image for pose estimation, we believe that these errors can be avoided by using several images for pose estimation. For example, 10 of the unsuccessful task executions are related to the image shown in the center of the bottom row in Fig. 10. Apart from pose estimation errors, type F2 failures are not easily detectable and hard to predict. One solution is that the robot visually verifies the final configuration of the manipulated object. Then, if the result is not as expected, the robot replans.

Fig. 11. Left: The robot gripper as it about to put down the box and finish the task. Right: The final configuration of the objects, arranged by the robot. When compared with the demonstrated task, Fig. 10 (top-left), the configurations seem identical.

6.3 Learning from human advice

This experiment is similar to the one presented in Section 6.2. This time, the human instructs the robot that the tape is to be in place before the other objects, which adds two constraints to the planner. This results in a bit longer plan:

pickUp(box, loc_ box_ s, box_ center)
putDown(box, free1, box_ center)
pickUp(block, loc_ block_ s, block_ left)
putDown(block, free2, block_ left)
pickUp(tape, loc_ tape_ s, tape_ center_ 180)
putDown(tape, loc_ tape_ t, tape_ center_ 180)
pickUp(block, free2, block_ left)
putDown(block, loc_ block_ t, block_ left)
pickUp(box, free1, box_ center)
putDown(box, loc_ box_ t, box _center)

As seen, the robot fulfills the constraint of placing the tape at the target location before the other objects are placed at their target locations.

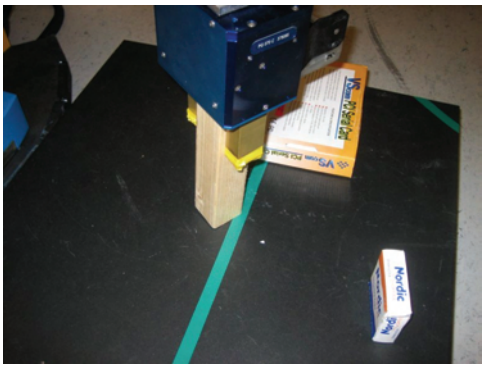

Fig. 12. An unexpected error which the system cannot detect. The wooden block is heavy and sometimes slips from the gripper when grasped on the side. The gripper still holds on to the block but due to its weight it rotates, and when being put down it causes a collision.

State	Object	Relative Position	(x,z)
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 3. Experiment in virtual environment: the states are generated from the demonstrations.

6.4 Generalizig from multiple examples

We have also performed experiments to evaluate our approach of generalizing from multiple examples. The first experiment is performed in a virtual environment and evaluates the state generation module with several objects. The second experiment shows how the task generalization module operates by automatically identifying a hidden constraint from several examples in the real world.

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

Table 4. The state sequences found in the demonstrations (virtual environment).

1) Generalizing in a Virtual Environment: In this experiment, the *set table task* described in Section 5.1 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 its position that makes this experiment easy to analyze. Fig. 13 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 3 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 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 4 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.

2) *Generalizing From Real Examples:* The method has also been evaluated in a real scenario where the robot observed the human performing the task three times. The start configuration was different each time, which led to three different observations, shown in Table 5. The task was to organize the objects according to test image 6, and the starting configurations were according to test image 1, 2 and 3 from Fig. 10. The user had an underlying constraint when performing the task, that the tape must be placed at the target location before the other objects reach their locations. From the observations, five states were automatically generated, as shown in Table 6. It is interesting to see that the system has correctly identified that the block should be placed relative to the tape, and that the box should be placed relative to the block. The observations are then remapped to the identified states. This results in state sequences, as shown in Table VII. From these, three constraints, $C < D$, $C < E$ and $D < E$, and

one common goal, *C*, *D*, *E* are identified. The goal is the final position of each object. Considering that this is a pick-andplace task, only the final positions are important. For other types of tasks the robot may have to perform either A or B as well. Both underlying constraints as well as an additional constraint *D* < *E*, have been identified. Although not intentionally demonstrated, that constraint is necessary in order to be able to align the box next to the block. The planned solution is the same as in Section 6.3, although the locations are slightly different since they are calculated from several observations.

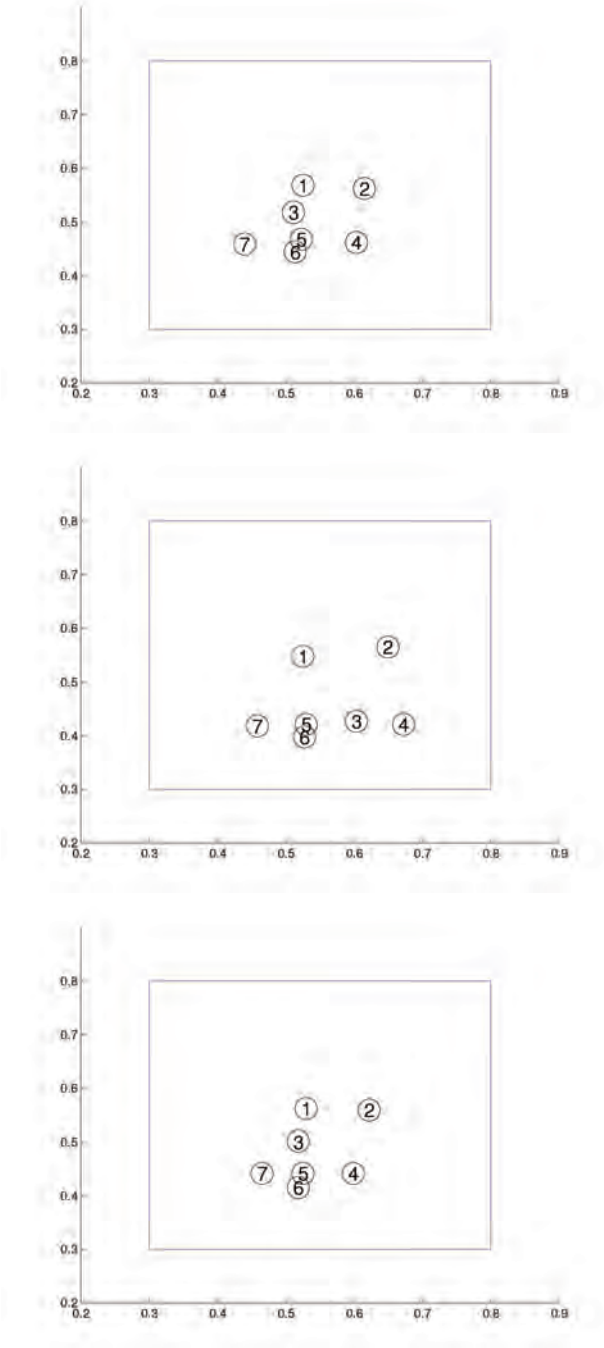

Fig. 13. Example demonstrations in the virtual environment. Numbers represent objects as: 1 (Glass), 2 (Food), 3 (Spoon), 4 (Knife), 5 (Plate), 6 (Napkin), 7 (Fork).

Observation 1	move box to (214, -40, -10) move tape to (12, -76, 57) move block to (-120,-39,-35) move box to (-174, 49, -121)
Observation 2	move box to (220, -60, 4) move tape to (13, -55, 52) move block to (-109, -25, -36) move box to (-106, 64, -122)
Observation 3	move block to (268, -82, -138) move tape to (23, -59, 52) move block to (-94, -16, -41) move box to (-93, 72, -126)

Table 5. The three observations perceived by the system (real environment).

State	Object	Rel. Position	Rel. Orientation	(x,z,θ)
A	box			(217, -50, -3)
B	block			(268, -82, -138)
C	tape			(16, -63, 54)
D	block	tape	tape	(-124, 37, -91)
E	box	block	block	(-17, 88, -86)

Table 6. Experiment in real environment: the states generated from the observations.

Observation 1	A-C-D-E
Observation 2	A-C-D-E
Observation 3	B-C-D-E

Table 7. The state sequences found in the observations.

7. Conclusions

In this work, we have presented a task learning system where a robot learning by demonstration scenario is integrated with a task level planning system. Three learning techniques have been considered: task learning from imitation, learning from human advice and learning from multiple observations where a task is represented by its goal configuration and task constraints. A task planner for manipulation tasks and reaching the goal state given different initial world states has been demonstrated. We have also addressed the issue of task constraints. If there are some underlying constraints that must be fulfilled the knowledge of just the final goal is not sufficient for task execution. We have proposed two techniques for constraint identification. In the first case, the teacher can instruct the system and, in the second case, the constraints are identified through the merging of multiple observations. The constraints are then considered in the planning phase, allowing the task to be executed without violating any of them. The experimental evaluation has been performed both in a virtual environment and with a robot manipulator. It has been demonstrated that the system is able to perform

tasks with real objects. We believe that the proposed framework is easily extendable to tasks involving more complex objects. The current system requires that all objects are visible at the planning stage. Our future work will consider settings where new objects and obstacles will be discovered during execution and thus online task replanning will be required.

8. References

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.

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

S. Schaal, "Is imitation learning route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, pp. 232–242, 1999.

M. Mataric, "Getting humanoids to move and imitate", in *IEEE Intelligent Systems*, pp. 18–24, July 2000.

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.

C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 481–487, 2002.

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.

M. Ehrenmann, O. Rogalla, R. Zöllner, 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.

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.

S. Ekvall and D. Kragic, "Grasp recognition for programming by demonstration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005.

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

J. Chen and A. Zelinsky, "Programming by demonstration: coping with suboptimal teaching actions," *International Journal of Robotics Research*, vol. 22, pp. 299–319, May 2003.

R. Schmidt and T. Lee, *Motor Control and learning: a behavioral emphasis*. Human Kinetics, 3rd edition, 1999.

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

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.

H. Friedrich and R. Dillmann, "Obtaining good performance from a bad teacher," in *Workshop: Programming by Demonstration vs Learnin from Examples; International Conference on Machine Learning*, 1995.

T. Lefebvre, H. Bruyninckx, and J. D. Schutter, "Task planning with active sensing for autonomous compliant motion," *International Journal of Robotics Research*, vol. 24, no. 1, pp. 61–81, 2005.

M. N. Nicolescu and M. J. Mataric, "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.

A. Ude, "Robust estimation of human body kinematics from video," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1489–1494, 1999.

M. Riley, A. Ude, and C. G. Atkeson, "Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching," in *AAAI and CMU Workshop on Interactive Robotics and Entertainment*, pp. 35–42, April 2000.

M. Ruchanurucks, S. Nakaoka, S. Kudo, and K. Ikeuchi, "Humanoid robot motion generation with sequential physical constraints," in *IEEE International Conference on Robotics and Automation*, pp.2649–2654, 2006.

J. Foley, A. van Dam, S. Feiner, and J. Hughes, eds., *Computer graphics - principles and practice*. Addison-Wesley Publishing Company, 1990.

D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

V. Kyrki and D. Kragic, "Integration of model-based and model-free cues for visual object tracking in 3d," in *IEEE International Conference on Robotics and Automation, ICRA'05*, pp. 1566–1572, 2005.

S. Russel and P. Norvig, *Artificial intelligence: A modern approach*. Second edition, Prentice Hall, 2003.

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.

M. Steedman, "Temporality," in *Handbook of Logic and Language* (J. van Benthem and A. ter Meulen, eds.), pp. 895–938, Elsevier, 1997.

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