

# An Approach for Efficient Planning of Robotic Manipulation Tasks

**Richard Dearden and Chris Burbridge**

University of Birmingham

Edgbaston, Birmingham, B15 2TT, U.K.

*richard.dearden@gmail.com, C.J.C.Burbridge@cs.bham.ac.uk*

## Abstract

Robot manipulation is a challenging task for planning as it involves a mixture of symbolic planning and geometric planning. We would like to express goals and many action effects symbolically, for example specifying a goal such as for all  $x$ , if  $x$  is a cup, then  $x$  should be on the tray, but to accomplish this we may need to plan the geometry of fitting all the cups on the tray and how to grasp, move and release the cups to achieve that geometry. In the ideal case, this could be accomplished by a fully hybrid planner that alternates between geometric and symbolic reasoning to generate a solution. However, in practice this is very complex, and the full power of this approach may only be required for a small subset of problems. Instead, we plan completely symbolically, and then attempt to generate a geometric plan by translating the symbolic predicates into geometric relationships. We then execute this plan in simulation, and if it fails, we backtrack, first in geometric space, and then if necessary in symbolic. We show that this approach, while not complete, solves a number of challenging manipulation problems, and demonstrate it running on a robotic platform.

## 1 Introduction

Planning for robotic manipulation tasks such as cleaning and tidying a house is a challenging problem because it contains a mixture of symbolic and geometric constraints. For example, we can plan to tidy a table by placing cups and plates on a tray and then moving them to the kitchen to be cleaned, but to accomplish the task we need to not only reason about the symbolic problem of having all the objects on the tray, but also the geometric problem of fitting all the objects on the tray and potentially stacking them if there is insufficient space. This paper considers this problem, and in particular the problem of translating back and forth between symbolic and geometric states during planning. We assume the robot's actions are reliable and that the world is perfectly known, and concentrate on the challenge of efficiently generating plans for these kinds of tasks.

Many manipulation planning approaches (see Chapter 7 of (LaValle 2006) for an overview) assume that the task can be treated entirely as a geometric problem, with the challenge being to place all the objects in their desired positions.

However, it is clearly desirable to specify goals and reason at a symbolic level. Specifying the exact geometric location of every object essentially requires knowing the plan in advance since, for example, if two of the cups need to be stacked to get them all on the tray, this must be specified in the goal. For this reason, we adopt a combination of task-level symbolic planning using Fast Downward (Helmert 2006), geometric planning to decide the desired positions of objects, and path planning (Kuffner and LaValle 2000; Siméon, Laumond, and Nissoux 2000) to generate trajectories for the robot to reach the object positions.

Due to the interactions between symbolic and geometric planning, the ideal approach would be to interleave the two, with geometric planning potentially causing symbolic backtracking when it fails, and vice versa. Unfortunately, because generating paths and geometric states is typically very time-consuming, this *hybrid* planning approach can be unacceptably slow due to the many geometric states and robot path plans generated for symbolic states that are not used in the final plan. To overcome this, we take a different approach, preferring to generate a complete plan at the symbolic level, then translate that plan into a geometric one and generate paths to achieve the geometric configurations. If this process fails, we allow a limited amount of purely geometric backtracking—proposing new positions for the objects moved by the robot—before giving up and trying backtracking at the symbolic level to generate a different plan.

Our approach requires us to be able to generate the initial symbolic state from the geometric state, and for any symbolic state to generate a corresponding geometric state. To do this we learn a bi-directional mapping between geometric and symbolic states. The mapping is learned from a large set of labelled training data of configurations and the symbolic predicates that are true in them. One significant advantage of this approach is that if the training data comes from “natural” scenes (in our case, it is partly generated by hand, and partly from existing robot programs), then the mapping will naturally reflect any implicit constraints in the data, for example ensuring that no objects are too close to the edge of a table. From the training data we learn a kernel density estimate (Scott 2008; Morariu et al. 2008) for each predicate. This allows us not only to label unseen geometric states (the forward direction for the mapping), but also, via hill-climbing search in

the probability density function, to find geometric states that match a symbolic predicates with high probability (the backwards direction). We extend this to find geometric states for conjunctions of symbolic predicates and also—for backtracking—to find multiple, significantly different geometric states for a predicate.

In the next section we present the planning approach in detail. In Section 3 we present the approach to learning the mapping, and in Section 4 we examine previous approaches to the problem. We present experiments to demonstrate the effectiveness of the approach in Section 5, and our conclusions and future work in Section 6.

## 2 Task Planning

The planning task is made challenging by the mixture of geometric and symbolic state. To plan optimally for both of these we would need a hybrid planner that interleaved symbolic and geometric planning, where a geometric problem (e.g. a collision between objects) leads to backtracking in the symbolic plan and vice versa. However, this approach is slow because geometric states and arm motion paths will be generated for states that are reached during search but never used in the final plan. Since the translation from symbolic to geometric states (described in Section 3) is quite slow, this amount of backtracking is unacceptable. Instead our approach is to plan symbolically, then generate a geometric plan from the symbolic plan and use an RRT planner to generate motion paths between the geometric states. If the motion plan fails we attempt to generate different geometric solutions by backtracking over the translation from symbolic to geometric states, and if that fails we then backtrack in the symbolic planner.

The high-level description of the algorithm is given in Algorithm 1. We describe each step in more detail below, using as a running example the task of placing two stackable cups on a saucer. In the example, the goal is `(above cup1 saucer) (above cup2 saucer)  $\forall ?x$  (not (holds ?x))` and the initial state is `(above cup1 table) (touching cup1 table) (above cup2 table) (touching cup2 table) (above saucer table) (touching saucer table)`. We assume there are three actions available to the robot: `(grasp ?x)` moves the robot’s hand to object `?x` and grips it, `(release ?x)` releases object `?x` if it has been grasped, and `(move ?x ?y)` moves grasped object `?x` to be above and touching object `?y`.

The symbolic planner we use is Fast Downward (Helmert 2006), using the FF heuristic. This typically generates initial plans for these simple manipulation domains in under a second—significantly faster than the translation to geometric states or the generation of paths for the robot. For our example, we might expect the initial plan generated to be `((grasp cup1) (move cup1 saucer) (release cup1) (grasp cup2) (move cup2 saucer) (release cup2))`.

To produce a geometric plan from a symbolic one, we extract the symbolic state after each action from the plan, and starting with the last state, translate it to a geometric state using the method described in Section 3. We work

---

### Algorithm 1 High-level description of planning.

---

```

Start-state  $\leftarrow$  geometric state mapped to symbolic predicates
Plan  $\leftarrow$  symbolic planner applied to domain, goal, start-state
states  $\leftarrow$  the sequence of states visited by the plan
for  $s \in$  states, starting with the final state do
    Find a geometric state that matches all geometric predicates
    in  $s$  with high probability
end for
Generate RRT paths between the geometric states
if the RRT planner fails to generate a path or the geometric plan
fails in simulation then
    Backtrack, first geometrically and then symbolically
else
    Pass the plan to the robot for execution
end if

```

---

our way back through the states generating new geometric states wherever the robot places an object in a location for which a geometric state hasn’t already been generated. For our example, this means finding a geometric state corresponding to the symbolic state `((above cup1 saucer) (touching cup1 saucer) (above cup2 saucer) (touching cup2 saucer))`. Since the saucer is not moved during the plan, its position is fixed in the geometric search and only the positions of the two cups are optimised. It is possible that the geometric translation fails immediately at this point if no position with the two cups on the saucer exists. In our example this is the only geometric translation which needs to take place as we have now fixed the position of every moved object, but in a more complex example we may need to regress through the plan and optimise the locations of other objects if they are moved more than once.

If the translation from a symbolic to a geometric plan succeeds, we then generate paths for the robot motions to grasp and move the objects using the OpenRAVE RRT path planner. If this step fails, this indicates that it is impossible to move an object without some kind of collision (either with an obstacle, or a self-collision of the robot arm) occurring, and backtracking will be required. Otherwise, we now have a usable plan in the geometric space, and the robot can go ahead and execute it. In our example, if the translation succeeds in placing the cups (very close together) on the saucer, this is then simulated, and will likely fail because the robot can’t place the second cup on the saucer without its gripper colliding with the first cup.

If the simulation of the geometric plan fails, backtracking is used to try to find a plan that will succeed. The first backtracking is in the geometric translation as described in Section 3. This generates alternative geometric states that are consistent with the symbolic state. We try this only a small number of times (a parameter that can be varied by the user) as there are generally infinitely many geometric states for a given symbolic one. If we fail to generate a geometric state we can plan for in the small number of trials, we then backtrack the symbolic planner to search for a different plan to accomplish the goal. In our example this might produce the new plan `((grasp cup1) (move cup1 saucer) (release cup1) (grasp cup2) (move cup2 cup1)`

(release cup2)), which nests cup2 inside cup1 on the saucer. This is again translated to a geometric plan, the robot motions planned by the RRT planner, the geometric plan is tested in the simulator, and then finally passed to the robot for execution.

### 3 Symbolic to Geometric Mapping

To use a symbolic planner for these kinds of problems we need to translate the geometric initial state into a symbolic state, and translate symbolic states generated in the plan into equivalent geometric states. As we will discuss in Section 4, most previous work has assumed a hand-built translation between geometric and symbolic states. Here we consider a learning approach: Given training data in the form of geometric states labelled with the predicates true in them we wish to learn a mapping between them that can be used in both directions. Since the symbolic to geometric mapping is clearly one-to-many and we wish to support backtracking in the planner, we also require a representation from which we can generate significantly different geometric states.

The approach we take is to represent the mapping using a kernel density estimate (KDE) (Scott 2008; Morariu et al. 2008) which provides a probability density function  $Pr(C|G)$  where  $C$  is a symbolic predicate and  $G$  is a geometric state. This allows us to efficiently compute the probability of a predicate holding as a sum of Gaussians, and to generate a geometric state where a predicate holds with high probability by gradient ascent. Given a symbolic state  $S = C_1 \wedge C_2 \wedge \dots \wedge C_n$ , assuming that the probability of each predicate is independent, the probability that the symbolic state holds in a geometric state  $G$  is:

$$\Pr(S|G) = \prod_{i=1}^n \Pr(C_i|G). \quad (1)$$

#### Geometric state representation

For a predicate over two objects  $A$  and  $B$ , we assume each object is represented by its bounding box, and position and rotation in world coordinates. We define the geometric state representation as:

$$G = [ S_A \quad S_B \quad T_B^A \quad R_A \quad R_B ] \quad (2)$$

where  $S_o, R_o$  are the size and rotation of object  $o$  respectively, and  $T_B^A$  is the relative translation between objects  $A$  and  $B$ . Using the difference in position between the objects rather than absolute positions provides better generalisation from the examples. However, the same can not be done with the rotation as many predicates require specific orientations (for example, containers need to be upright). We represent the rotations using quaternions (although this adds an extra dimension per rotation, it avoids discontinuities in the probability distributions) taking the geometric state vector to 17 dimensions. We can similarly define the geometric state representation for a predicate of arity one as a ten dimensional vector:

$$G = [ S \quad P \quad R ] \quad (3)$$

where  $S$  and  $R$  are the size and rotation of the object as before, and  $P$  is its absolute position.

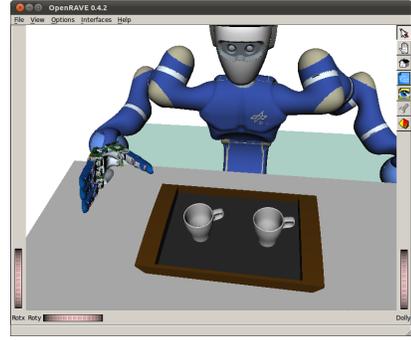


Figure 1: Example geometric state. The predicate labels for this state are ((above cup1 tray) (touching cup1 tray) (above cup2 tray) (touching cup2 tray) (above tray table) (touching tray table)).

The training data is derived from a large set of object configurations; an example is shown in Figure 1. Each geometric configuration is labelled with all the predicates true in the scene, so for example the figure is a positive example for (above cup1 tray) and a negative example for (above cup1 cup2). We can then derive the 17-dimensional relationship between the objects for each predicate and use these to train a KDE for each predicate.

#### Probability model

Given a set of  $n^+$  example geometric states  $G^+$  that a predicate holds in, and a set of  $n^-$  geometric states  $G^-$  in which the same predicate does not hold, we model the probability as:

$$\Pr(C|G) = \frac{\Pr(G|C)}{\Pr(G|C) + \Pr(G|\neg C)}, \quad (4)$$

where  $\Pr(G|C)$  and  $\Pr(G|\neg C)$  are the multivariate KDEs of the positive and negative datasets for  $C$  respectively, evaluated at  $G$ :

$$\Pr(G|C) = \frac{1}{n^+} \sum_{i=1}^{n^+} e^{-\|G_i^+ - x\|^2/h^2} \quad (5)$$

$$\Pr(G|\neg C) = \frac{1}{n^-} \sum_{i=1}^{n^-} e^{-\|G_i^- - x\|^2/h^2}, \quad (6)$$

where  $h$  is the kernel bandwidth. We use the FigTree Fast Gaussian Transform library (Morariu et al. 2008) for efficient KDE implementation.

Figure 2 shows an example of the probability model for a one dimensional case. In areas of negative examples the function is near zero and in areas of positive examples it is near 1. In the boundary between positive and negative examples the function varies smoothly with a smoothness dependent upon the width of the Gaussian example points either side. This allows numerical hill-climbing techniques as described below. Figure 3 shows a geometric state and the learned distributions for a variety of predicates and symbolic states.

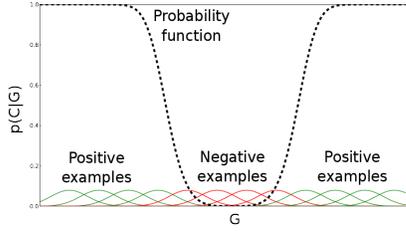


Figure 2: 1D example of a probability distribution modelled using equation 4.

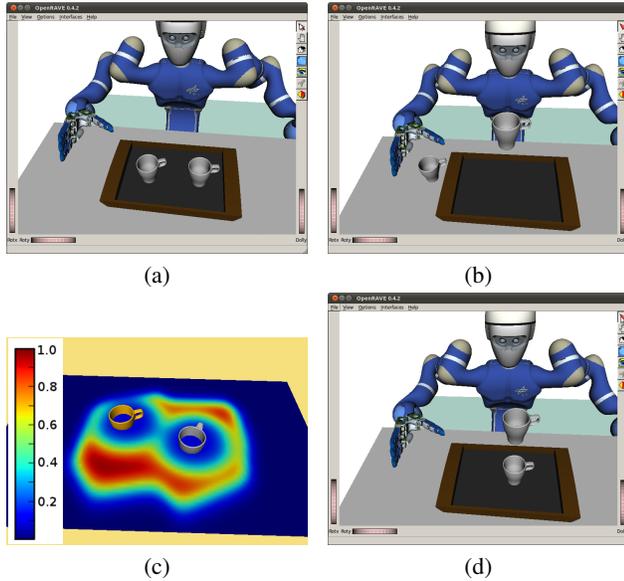


Figure 3: Probability distributions superimposed on a geometric state. (a) Tray and two cups. (b) Probability of a cup above the table. (c) Probability of a cup above the table and not touching other cups. (d) Probability of a cup above and touching other cup.

### Using the learned mapping

The learned probability distribution can now be used to generate the initial state for the symbolic planner by classify a geometric state into a symbolic state, and to generate a geometric plan from a symbolic plan by translating the symbolic states encountered in the plan into geometric states.

Mapping from geometric states to symbolic is straightforward: we simply threshold the class probability according to the KDE at some sensible value. In practice we use values close to one.

The more interesting case is generating geometric states matching desired symbolic configurations. For a single predicate goal we achieve this by maximising the 17D probability function, keeping the 6 elements corresponding to object size constant. The smooth and continuous nature of the function allows maximisation to be carried out numerically using hill climbing from an initial estimate selected from  $G^P$ . We found the Broyden-Fletcher-Goldfarb-Shanno optimisation method performs suitably for this purpose.

When considering more than one predicate using Equation 1, the dimensionality of the space increases, but not in a simple way. If an object appears in more than one predicate, the size and rotation of the object should only appear in the space once. Similarly, if the state contains a translation from object A to B, and from B to C, the translation from A to C is determined, so should not be separately optimised. We avoid these problems by optimising over the absolute poses of all non-fixed objects in a single step, rather than optimising each object’s relative pose.

As an example consider using the approach to find a geometric state for the 5 clause symbolic state (and (above cup1 tray) (touching cup1 tray) (above cup2 tray) (touching cup2 tray) (not (touching cup1 cup2))). If each predicate was treated independently then the search space would contain 85 dimensions. Removing the repeated dimensions for the orientations of objects appearing in the state multiple times, and setting the sizes of all the objects that the state concerns constant leaves a search space of 27D. However, optimising directly in this space could lead to an impossible geometric state as the space contains a loop of transformations: if a transformation between cup1 and the tray is found, and a transformation between cup2 and the tray is found, then the transformation between cup1 and cup2 has to be fixed and can not be found independently. Hence we *optimise* over the rotation and absolute position of each object, so for the three objects in the example we have 21 dimensions. We then *evaluate* a solution by generating the 17D vector for each predicate and computing the product of the KDEs as in Equation 1.

### Backtracking

When the planner backtracks we need to ensure that the geometric states it generates are sufficiently different from those already generated. The use of a KDE representation lends itself to a straight forward solution: At each backtrack  $j$  we generate a masking function for each predicate. The masking function consists of a Gaussian centred on the geometric state found by hill-climbing for that predicate. For each point in the optimisation space we then subtract the value of the masking Gaussian that is largest at that point for each specific predicate. Let  $\mu_i$  be the geometric state found for predicate  $C_i$ . To mask the predicate for backtracking we create a Gaussian with mean  $\mu_i$  and covariance matrix  $Q_i$ :

$$K_j^{C_i}(G) = e^{-\frac{1}{2}(G-\mu_i)^T Q_i^{-1}(G-\mu_i)}. \quad (7)$$

We then modify Equation 1 as follows:

$$\Pr(S|G) = \prod_{i=1}^n \Pr(C_i|G) - \max_j K_j^{C_i}(G) \quad (8)$$

This forces the hill climbing to avoid masked geometric states. The set of masked states  $G^{mask}$  (one set per predicate) expands as candidate states are generated, so if we backtrack multiple times, we end up with multiple masks for each predicate. Varying the covariance matrix  $Q_i$  varies the masked area around the predicate. In the experiments reported here we are using a covariance selected to mask an area about the size of the object being moved.

## 4 Related Work

Typical manipulation planning systems (see Chapter Seven of (LaValle 2006) for an overview) treat planning as a completely geometric problem of searching in the set of reachable geometric configurations to reach a goal. However, this requires the goal to be completely specified geometrically, which means that the kinds of alternative ways of achieving a symbolic goal that we can generate are impossible. A recent alternative (Mösenlechner and Beetz 2009) is to specify goals symbolically but evaluate the plan geometrically. The idea is to use a high-fidelity physics simulation to predict the effects of actions and a hand-built mapping from geometric to symbolic states. However, this is likely to be very expensive for complex plans.

More closely related to our approach is aSyMov (Gravot, Cambon, and Alami 2003), which solves tasks very similar to ours using a planner that combines a symbolic planner with a probabilistic roadmap (Siméon, Laumond, and Nissoux 2000) for geometric planning. The planning algorithm is broadly similar to ours, using a mixture of symbolic and geometric planning, although in a fully hybrid approach. However, the approach appears to rely on the fact that the mapping between symbolic and geometric states is much simpler: There is only a single predicate “on” that corresponds to a geometric state, and they restrict the objects to certain fixed world positions so the translation from symbolic to geometric states is trivial.

Other recent fully hybrid approaches include (Dornhege et al. 2012) and (Gregory et al. 2012). In both approaches the symbolic planning language includes calls to specialised solvers that check preconditions of actions or calculate their effects. For a manipulation domain, as described in (Dornhege et al. 2009) the RRT planner would be used in this way. In both cases it’s unclear if it would be possible in a general way to use the specialised solvers with our learned geometric predicates.

Another example is (Kaelbling and Lozano-Pérez 2011), which does full hybrid planning, but to reduce the complexity of the problem, uses a hierarchical planner and interleaves planning with execution. As with the above approach, it again only seems to use a single symbolic predicate for a geometric concept. Their mapping from symbolic to geometric states is handled by *geometric suggesters* which are hand-built functions for generating geometric states. Similar approaches are found in (Wolfe, Marthi, and Russell 2010) and (Karlsson et al. 2012) which both use a hierarchical task network planner to constrain the search space.

The idea of learning the mapping from geometric to symbolic state has been considered before in the literature. However, what sets our approach apart from others is the idea of using the learned mapping in reverse, generating geometry from symbolic description to enable symbolic planning operators to be utilised to produce geometric effects. The closest work in this area is (Sjöö and Jensfelt 2011), which learns to classify object relationships by generating samples in simulation. While classification of several relations is successful, they are only interested in learning the mapping from geometric to symbolic states; using the mapping in reverse is not considered. Furthermore, the geometric state

is described by a 93 dimensional feature vector comprising of contact points, surface normals and object poses, and the symbolic object relationships are learned from this feature vector. The high dimensionality means that far more training data is required to learn a good mapping and makes adapting their approach to compute the reverse mapping impractical. A similar approach is presented in (Rosman and Ramamoorthy 2011). Again, contact points are used, this time between segmented point clouds. However, the use of K-Nearest neighbours for classification again prevents using the classifier in reverse.

## 5 Experiments and Results

To test our system, example geometric states of two cups and a tray were used for the geometric-symbolic mapping. Each state was labelled to indicate whether one cup was ‘above’ and/or ‘touching’ the tray or the other cup. The data set for the ‘above’ predicate comprised of 328 positive and 1672 negative examples, and the data set for the ‘touching’ predicate comprised of 384 positive and 1872 negative examples.

The samples were then used to create the 17 dimensional probability distribution detailed in Section 3. Examples of the  $x$  and  $y$  dimensions of the distribution with  $z$  3cm above the tray are shown in Figure 3. Each point on the plane is coloured according to the probability that a cup placed at that  $(X,Y,Z)$  position meets the specific symbolic state: sub image (b) shows the probability that the position is above the tray; (c) above the tray and not touching either other cup; (c) above and touching the cup on the right. The bandwidth parameter (variance of the Gaussians) to the Kernel density estimation was selected manually. Larger values make the boundary transitions slower, while low values lead to abrupt transitions that are more difficult for the optimisation.

### Learned Mappings

**The forward mapping** To evaluate the predicate classifiers in the forward direction we carried out leave-one-out cross-validation for each predicate. Carrying out the cross-validation for several classification thresholds suggested an optimum value of about 0.8. We found that with significantly less training samples than the already small number that we demonstrate with here, the classification results in large numbers of false negatives. The resulting confusion matrices when using a threshold of 0.8 were as follows:

		Predicted	
		above	not above
Actual	above	328	0
	not above	24	1648

		Predicted	
		touching	not touching
Actual	touching	376	8
	not touching	24	1848

Figure 4 illustrates the forward mapping by showing four simulated geometric configurations. For each of them, we compute the probability of a number of symbolic predicates, with the results in Table 1. From the table it can be seen that

Table 1: Classification probabilities for the states shown in Figure 4

State	(a)	(b)	(c)	(d)
(above cup tray)	0.999	1.000	1.000	0.999
(touching cup tray)	0.971	0.000	0.001	0.643
(above cup2 tray)	0.999	0.016	0.999	0.999
(touching cup2 tray)	0.974	0.000	0.962	0.962
(touching cup cup2)	0.038	0.000	0.105	0.925
(above cup cup2)	0.009	0.000	0.983	0.970

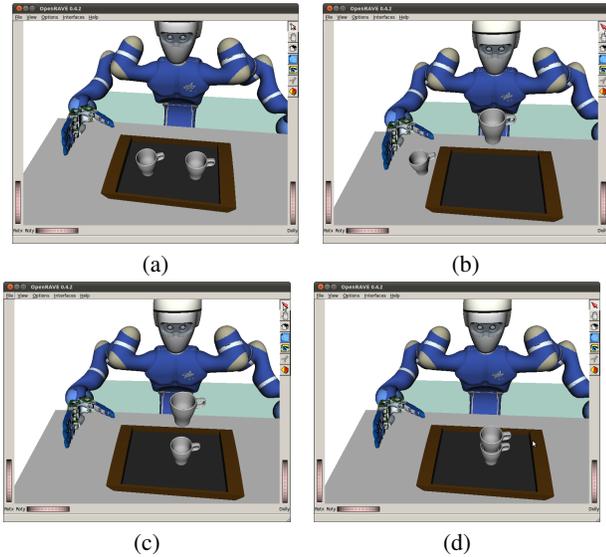


Figure 4: The geometric states tested in Table 1

even with a higher threshold in the region of 0.9 these states are classified correctly. Although the geometric configurations differ from the examples that the system was trained with, the choice of representation for the probability distribution means that states falling between two positive examples are awarded near 1 probability. The smooth nature of the probability estimate can be further seen in example (d) (one cup stacked inside another) when deciding if the upper cup is touching the tray or not. A probability of 0.643 indicates that it is close to the tray, but not as close as the cup below which is touching with probability greater than 0.9.

**The reverse mapping** Figure 5 shows four geometric configurations automatically generated to match the compound states listed in Table 2. States (b) and (c) demonstrate the result of combining the individual predicate distributions. Trying to find a geometric state that has multiple cups on the tray, but not touching each other results in a process somewhat like a spring-and-mass system during hill climbing where the cups are pushing each other apart while the tray is pulling them towards it. The end configuration is the cups on the tray and as far apart from each other as possible.

Table 2: Classification probabilities for the states shown in Figure 5

	Goal state supplied	Prob.
(a)	(and (above cup tray) (touching cup tray))	0.989
(b)	(and (above cup tray) (touching cup tray) (above cup2 tray) (touching cup2 tray) (not touching cup cup2))	0.977
(c)	(and (above cup tray) (touching cup tray) (above cup2 tray) (touching cup2 tray) (above cup3 tray) (touching cup3 tray) (not touching cup cup2)) (not touching cup cup3)) (not touching cup2 cup3))	0.941
(d)	(and (above cup tray) (touching cup tray) (above cup2 tray) (touching cup2 tray) (above cup3 cup2) (touching cup3 cup2) (not touching cup cup2))	0.887

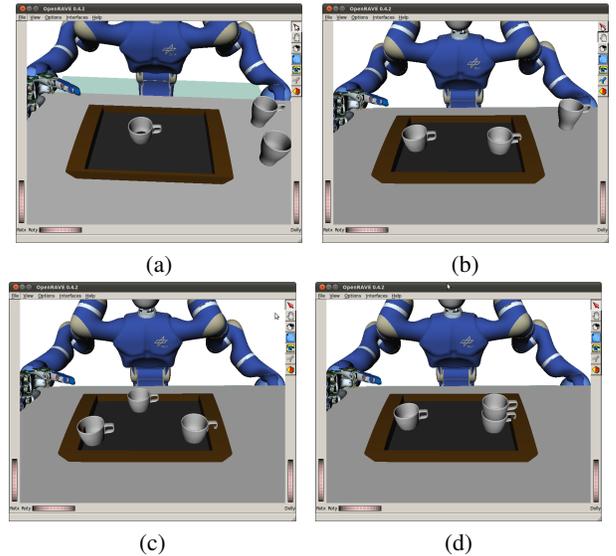


Figure 5: Geometric states generated for the symbolic states give in Table 2.

## Robotic Manipulation Planning

To demonstrate the applicability our planning approach in a real world robotics context we used the learned predicates and planning system to generate plans for execution by a robot manipulator. We constructed a planning domain description in PDDL (McDermott et al. 1998) comprising actions to pick objects and place them on other objects. For each symbolic action we created a chunk of robot code to perform the action, with additional parameters for the geometric positions. The actions make calls to the OpenRAVE RRT path-planner to move the 6 DOF arm between poses.

We tested our approach using the *Fast Downward* (Helmert 2006) symbolic planner with the initial state being four cups and a tray on a table, and the goal of having no cups on the table or held. The planner generated the sequence of actions necessary to arrange the cups and turned the symbolic plan into a full geometric plan ready for executing on the robot. The plan was executed on the robot

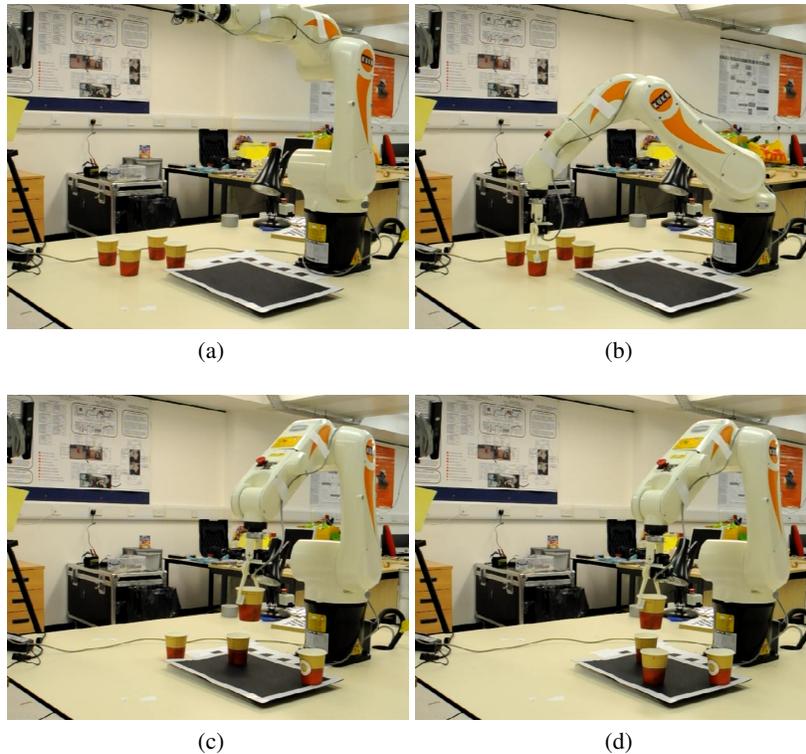


Figure 6: The test planning scenario on the Kuka arm. (a) The initial configuration. (b) Ready to grasp a cup. (c) Cup above table. (d) Cup above another cup.

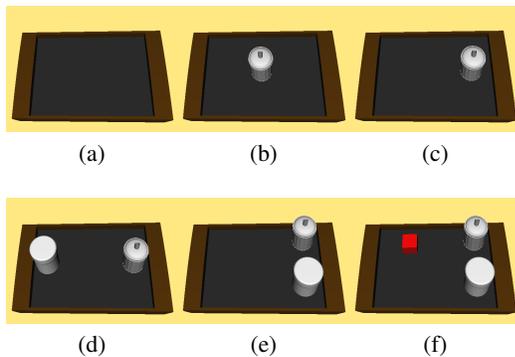


Figure 7: The tray states for evaluating the planner. The objects on the tray are not moveable by the robot.

as shown in Fig. 6<sup>1</sup>. The robot was an industrial arm (Kuka KR5-Sixx R850) with a two finger gripper (RoadNarrows Graboid). Object localisation used a Kinect RGBD camera. The point cloud was segmented into clusters above the table plane, with each cluster centroid giving the location of an object. The transformation from Kinect frame to robot frame was calculated using a calibration board at a known location in the robot frame.

<sup>1</sup>Video of execution available at: <http://goo.gl/Qrm6B>

We then tested our planning system in a simulation of the Justin robot (Fuchs et al. 2009), again using OpenRAVE as the RRT planner, to evaluate both the symbolic and the geometric backtracking. Twenty-four scenarios were evaluated: six configurations of the tray with unmoveable objects on it (see Figure 7), each with between one and four cups next to the tray on the table. The goal each time was that none of the cups be touching the table. The results are shown in Table 3.

The results show that typically only one or two geometric backtracks are necessary even on a cluttered tray. This is because the masking of the predicates quickly pushes the hill climbing away from occupied areas. Masking each predicate individually rather than simply using one mask for the complete state helps too. If it is found that one cup does not fit in a given location on the tray then this information is stored at the predicate level and prevents the system from trying to put any other cups in that location.

Table 4 shows how the total time is split between symbolic planning, geometry generation and path planning, with each time being cumulative over all the times that procedure was run (so in four cups on tray (d), the symbolic planner and geometry generation ran twice each, with the sum of the times reported). The efficiency of the approach hinges on the number of geometric backtracks, which is why a fully hybrid approach that may generate geometric states that aren't on the way to the goal appears unlikely to be efficient. As the table shows, time to generate a geometric state varies from

Table 3: Resulting states and backtrack counts when creating plans to place different numbers of cups onto the tray states shown in Figure 7. In all cases except placing four cups in configurations (d) and (f) all the cups were placed individually on the tray. For four cups in (d) one cup was nested in another; in (f) the three reported results correspond to three different outcomes: In  $F_1$  the planner failed to return a plan, in  $F_2$  it returned a plan with all four cups individually on the tray, while in  $F_3$  it nested one cup.

Tray State	No. of Cups	Geometric Backtracks	Symbolic Backtracks	Total Time (s)
A	1	0	0	6.01
A	2	0	0	16.33
A	3	0	0	27.51
A	4	0	0	46.03
B	1	1	0	11.17
B	2	0	0	15.27
B	3	0	0	28.46
B	4	0	0	49.84
C	1	0	0	5.95
C	2	1	0	30.67
C	3	1	0	55.29
C	4	1	0	91.41
D	1	0	0	5.87
D	2	2	0	54.39
D	3	1	0	70.35
D	4	6	1	246.26
E	1	0	0	6.27
E	2	1	0	36.35
E	3	1	0	61.47
E	4	1	0	124.73
F	1	0	0	6.40
F	2	2	0	44.37
F	3	1	0	59.13
$F_1$	4	6	4	528.60
$F_2$	4	1	0	110.85
$F_3$	4	1	1	270.08

3.5 seconds for a simple scenario to over 100 seconds for placing four cups on a cluttered tray. However, in the latter case this time consists of multiple gradient ascents since many attempts do not lead to a maximum of sufficiently high probability, so gradient ascent is randomly restarted.

Symbolic backtracking occurred when trying to place four cups on tray (d) and (f). For tray (d), geometric backtracking six times failed to find a configuration for the first symbolic plan. Symbolic backtracking generated a new plan to place three cups on the tray and stack the fourth. A geometry was then found without further backtracking. For tray (f), different runs produced different outcomes, with 40% ( $F_1$  in Table 3) failing to find a plan despite generating five different symbolic plans (four of them variations on stacking one of the cups in another), 10% finding a solution with all four cups squeezed onto the tray ( $F_2$ ), and 50% of runs symbolically backtracking to find a plan with a stacked cup ( $F_3$ ).

Table 4: Times in seconds for each stage in plan generation for selected problems from Table 3.

Tray State	No. of Cups	Sym. Plan	Geom. Gen.	Path Plan	Total Time
A	1	0.02	3.50	2.49	6.01
B	1	0.02	6.20	4.56	11.17
D	1	0.02	3.49	2.36	5.87
D	2	0.04	40.09	13.46	54.39
D	4	0.92	210.60	34.74	246.26

## 6 Discussion and Future Work

The combination of geometric and symbolic planning makes manipulation planning a difficult task. In this paper we have attempted to show that by planning purely symbolically, then translating the plan to a geometric one we can generate plans more efficiently than by using a complete hybrid planner. Since the computation time is far larger for generating geometric states and robot motions, minimising the number of times this needs to be done is critical to keeping the overall computation time down. Performing the geometric translation backwards through the symbolic plan makes this even more efficient by optimising the positions of as many objects as possible simultaneously, thus greatly reducing the number of optimisations that must be performed—even though the optimisation is in a higher dimensional space, this is less significant than doing repeated searches with backtracking.

The approach we describe is not complete—there are solvable problems for which it will not find a plan. The simplest example of these is if there is a movable object that completely blocks the tray in the initial state. In this case the planner will generate a plan to place the cups on the tray, that plan will fail because no geometric state can be found, but symbolic backtracking will never generate a plan to move the object because from the symbolic planner’s perspective the object is irrelevant. While we are choosing to sacrifice completeness for efficiency, we are nevertheless considering ways to solve this problem. One possibility is to allow the geometric path generation to change the symbolic problem to report obstacles before triggering backtracking.

The learned mapping between the symbolic and geometric states has a number of important properties from a planning perspective. Firstly, it means the planner generates “natural” geometric positions for the objects, which helps capture constraints that may not be explicit in the domain. Also, it can be easily extended to additional geometric predicates in more complex domains. In fact, we are currently investigating whether we can automatically detect properties of human-built robot programs that might be useful predicates for planning. An example of this might be the relationship between two cups when stacking one cup inside the other—often the bounding boxes should intersect to nest the cups, rather than releasing the top cup when it is above and touching the other. Similarly, unsupervised learning from arm trajectories might yield constraints such as that cups should be kept open-end-up when moving them.

## Acknowledgements

This work was supported by the European Community's Seventh Framework Programme as part of the GeRT project, grant agreement number ICT-248273.

## References

- Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, 1–6. IEEE.
- Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2012. Semantic attachments for domain-independent planning systems. *Towards Service Robots for Everyday Environments* 99–115.
- Fuchs, M.; Borst, C.; Giordano, P. R.; Baumann, A.; E. Kramer, J. L.; Gruber, R.; Seitz, N.; Plank, G.; Kunze, K.; R. Burger, F. S.; Wimboeck, T.; and Hirzinger, G. 2009. Rollin' justin - design considerations and realization of a mobile platform for a humanoid upper body. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.
- Gravot, F.; Cambon, S.; and Alami, R. 2003. aSyMov: A planner that deals with intricate symbolic and geometric problems. In *ISRR*, volume 15 of *Springer Tracts in Advanced Robotics*, 100–110. Springer.
- Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. *Proceedings of the International Conference on Automated Planning and Scheduling*.
- Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.
- Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Proceedings of Robotics and Automation (ICRA)*, 1470–1477.
- Karlsson, L.; Bidot, J.; Lagriffoul, F.; Saffiotti, A.; Hillenbrand, U.; and Schmidt, F. 2012. Combining task and path planning for a humanoid two-arm robotic system. In *Proceedings of TAMPPRA: Combining Task and Motion Planning for Real-World Applications (ICAPS 2012 workshop)*.
- Kuffner, Jr., J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-00)* 995–1001.
- LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.
- McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL—The planning domain definition language. Technical Report DCS TR-1165, Yale University, New Haven, Connecticut.
- Morariu, V. I.; Srinivasan, B. V.; Raykar, V. C.; Duraiswami, R.; and Davis, L. S. 2008. Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems (NIPS)*.
- Mösenlechner, L., and Beetz, M. 2009. Using physics-and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior. In *19th International Conference on Automated Planning and Scheduling (ICAPS)*.
- Rosman, B., and Ramamoorthy, S. 2011. Learning spatial relationships between objects. *The International Journal of Robotics Research* 30(11):1328–1342.
- Scott, D. W. 2008. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Hoboken, NJ, USA: John Wiley and Sons, Inc.
- Siméon, T.; Laumond, J.; and Nissoux, C. 2000. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*.
- Sjöö, K., and Jensfelt, P. 2011. Learning spatial relations from functional simulation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 1513–1519. IEEE.
- Wolfe, J.; Marthi, B.; and Russell, S. 2010. Combined task and motion planning for mobile manipulation. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*.