# Graph-based Trajectory Planning through Programming by Demonstration

Nik A. Melchior and Reid Simmons[1]

***Abstract*— As robots are utilized in a growing number of applications, the ability to teach them to perform tasks safely and accurately becomes ever more critical. *Programming by demonstration* offers an expressive means for teaching while being accessible to domain experts who may be novices in robotics. This work investigates a programming by demonstration approach to learning motion trajectories for robotic manipulator tasks. Using a graph constructed to determine correspondences between multiple imperfect demonstrations, the robot learner plans novel trajectories that safely and smoothly generalize the teacher's behavior, while attenuating those imperfections. The learner also actively detects instances of diverging strategy between examples, requesting advice for resolving these ambiguities. We demonstrate our approach in example domains with a 7 degree-of-freedom manipulator.**

## I. Introduction

Robots are becoming more common in many domains: They are used as tools for manufacturing, instruments for surgery, and toys for consumers. As their areas of application expand, it becomes increasingly critical to reliably transfer task knowledge to the robot. While domain experts often can be found who understand the task, they may know nothing about programming robots. *Programming by demonstration* (PbD) is an approach that facilitates knowledge transfer from a domain expert to an autonomous system. It provides an intuitive approach for someone skilled in performing a task to teach a robot to perform that task without having to learn to program the robot. Conversely, it does not require a robotics expert to become skilled in the task.

A primary difficulty in PbD approaches to robotic manipulation is representing and understanding the constraints imposed by the physical world. *Geometric* constraints, including the locations of physical obstacles, are the most obvious issues, since detecting the locations of objects can be difficult for current sensor technologies. Vision or LIDAR sensors must be located to observe the entire environment, without suffering from occlusion due to the objects or the robot itself. Alternately, the robot may be provided with an *a priori* model of the objects of interest, but constructing this model may require the skills of a robotics specialist.

Even if a model could be constructed to provide the robot with knowledge of obstacles, *non-geometric* task constraints must also be considered. For example, a robot carrying a cup of liquid must maintain its end-effector orientation to avoid spilling. Similarly, a robot routing cable around complex objects may need to follow a particular path to avoid snagging the cable. While a detailed physical simulation may be able to detect violations of such constraints, it would be challenging for a novice robot user to communicate them to a planner. However, if a user is able to demonstrate successful strategies for completing the task, he does not need to articulate the criteria that he is optimizing in a manner comprehensible to the robot.

Thus, PbD enables domain experts to teach robots about geometric and non-geometric task constraints without explicitly formalizing those constraints. The robot then create novel plans respecting the constraints, generalizing the actions performed by the teacher. In this work, we present a method for planning novel motion trajectories for robot manipulators based on demonstrations. Our approach learns to perform fixed, repetitive tasks in the presence of static obstacles, with minor variations due to uncertainty in both initial conditions and the ability to follow a trajectory precisely. In particular, we do not assume that example demonstrations are flawless, but rather that they contain jitter, non-optimal movements, and inconsistencies between examples. The robot must therefore discern the essential motions common to multiple examples and incorporate them in novel plans. In addition, we assume that the examples may differ in essential characteristics, making it difficult for the robot to combine the trajectories into a single, learned strategy. In such cases, the robot must refine its knowledge by actively requesting additional information from the teacher, in order to differentiate the manipulation strategies.

Our approach relies on a *neighbor graph* over the demonstrations, a representation that relates discretely sampled points from example trajectories. Our previous work [1], summarized in Section III, details a method to build such a data structure. The planning algorithm contributed in the present work uses this neighbor graph to create new plans that are guaranteed to be safe (avoiding obstacles) as long as the demonstration trajectories are also safe. Each point in a new plan is constructed using a set of neighbors from the demonstration set, refined to reduce jitter and minor inconsistencies, while maintaining the essential (common) characteristics of the demonstration trajectories. The planning algorithm also detects demonstrations that cluster into qualitatively distinct trajectories (for instance, circumventing an obstacle by two entirely different routes). In such cases, the robot requests additional advice from the operator to determine which strategy should be preferred. Experimental results in Section V demonstrate our approach in two different domains: The robot learner is able to safely trace a smooth path through a maze (with a single solution) and across a planar surface with multiple obstacles and multiple

[1]Robotics Institute of Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, USA; {nmelchio,reids}@cs.cmu.edu

demonstrated routes between them. We also present results from a user study that indicates how well novices can use our approach to train the robot.

## II. Related Work

Approaches to programming by demonstration differ widely in their use of prior knowledge and the models used to create new plans. Simplified motion or world models [2], [3], [4] and even traditional motion planning [5] can be used to plan between demonstrated states if an environment model is available. Use of the model may alternately be limited to collision testing plans created by other methods [6], [7].

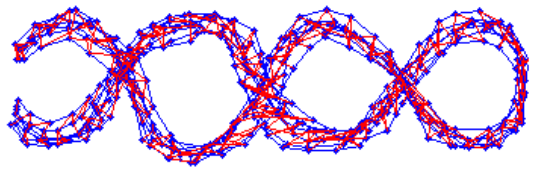
Many approaches collect similar actions into libraries, which can be applied to create motion plans for new tasks or situations. [8], [9]. Others attempt to learn dynamical models of the system using GMMs [10], Neural Networks [11], or other statistical feedback techniques [12]. These models may then be used to create new plans. Feature-based policy-learning approaches such as that of Argall [13], Chernova [14], and Ratliff [15] also create results that are applicable to new tasks and domains.

Our approach is most similar in spirit to data-driven approaches such as the interpolation-based methods of Ude [16], Lee [17] and Aleotti [6] or flow-field techniques developed by Mayer [18] and Drumwright [19]. These approaches, like ours, focus on learning entire tasks (or portions of tasks) at a time, rather than computing an action based solely on local features. Among similar approaches, only Delson's work [20], though, explicitly considers safety during the generation of novel trajectories. This approach is most completely developed in two dimensions, but an extension to three dimensions is presented in [21].

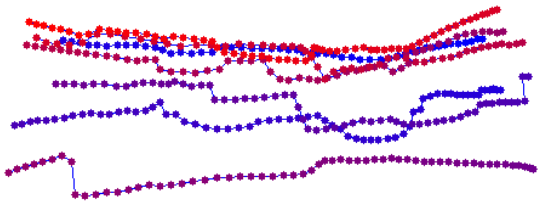
Dimensionality reduction approaches attempt to discover a lower dimensional representation of the demonstrated motions that retains the crucial features of the examples. While some such approaches use techniques, such as PCA, that retain as much variance as possible [22], others attempt to directly estimate correspondences between portions of example trajectories [23]. Rather than optimize for variance across the entire dataset, this alternative, which builds upon Isomap [24], optimizes for distances between points in the dataset. This approach requires a neighbor graph connecting points that are considered qualitatively similar. Distances between neighboring points are calculated directly using the metric of choice (e.g. Euclidean), but all other pairwise distances are calculated as the sum of shortest-path link distances through the neighbor graph. These *geodesic* distances indicate the nearness of points in terms of task execution, and provide a basis for determining which points can reasonably be clustered and interpolated.

## III. Background

Our previously reported work [1] developed a neighbor-finding technique that used a set of heuristics to produce a neighbor graph suitable for reduced-dimensionality planning. Planning in a low-dimensional *latent* space is attractive because the arrangement of trajectory points in this space



(a) Demonstration *slalom* trajectories (blue) with neighbor links (red).



(b) Slalom trajectories in the latent space.

Fig. 1. Slalom trajectories.

is expected to correlate with the semantics of the task. For example, the trajectories of Figure 1(a) cross over themselves in work space, but are "unrolled" in the latent space (Figure 1(b)). The start and goal points, previously close to one another, are now at opposite ends of the horizontal axis. Thus, this dimension represents progress through the task, while the vertical dimension separates distinct demonstrations. Planning through this space would seem to be intuitive since movement in each direction has a semantic explanation.

Unfortunately, we have found that this and similar approaches to dimensionality reduction do not lead to robust planning. For one, distances in the neighbor graph are particularly susceptible to spurious neighbor links. While these algorithms are robust to a large number of false negatives (missing neighbor links), even a single false positive link can have disastrous effects on the embedding [25]. This is because spurious neighbor links create "short circuit" connections between unrelated portions of trajectories, thus lowering the geodesic distances between points in semantically distant regions. This forces the embedding to retain the proximity of these regions in the low-dimensional latent space, introducing false semantic relevance between points.

While our neighbor-finding approach mostly avoids the problem of spurious neighbor links [1], another fundamental problem is that *lifting* novel trajectories from the latent space back to the work (Euclidean or configuration) space often produces second-order discontinuities. Lifting is typically performed by projecting discrete points sampled from the trajectory in the latent space into the work space. Individual query points are lifted by considering nearby sample points whose corresponding high-dimensional points are known. The high-dimensional point corresponding to the query is calculated by interpolating between the high-dimensional neighbor points, weighted by their distance from the query in the low-dimensional space. Unfortunately, non-linear em-

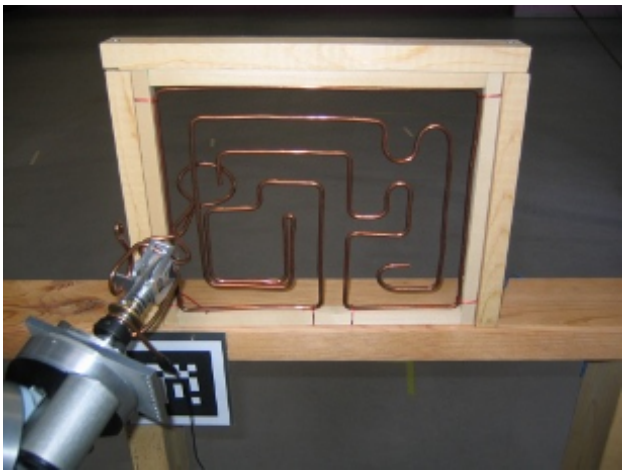


Fig. 2. The WAM manipulator traversing a wire maze. The fiducial in the lower-left is used to detect the location of the rig relative to the robot.

beddings, such as Isomap [24], do not guarantee smoothness of lifted trajectories, even when points are densely sampled from a smooth latent-space trajectory. Moreover, because dimensionality reduction is not *injective* (multiple regions of the work space can map to the same regions of the latent space), there may be ambiguities in lifting points back to the work space. These problems are actually exacerbated as additional example trajectories are provided to the learner. More examples produce a more cluttered latent space with more interconnections between neighbor points. This creates additional opportunities for discontinuities in lifted trajectories. Clearly, decreasing performance with an increasing amount of information is not a desired attribute of a planner.

These problems are clearly illustrated in our wire maze domain (Figure 2). Here, participants were asked to guide a 7-DOF Barrett WAM arm through a wire maze, with the arm in passive gravity-compensation mode. Figure 3 displays workspace traces of six demonstration trajectories (in purple) beginning at the green points near the bottom of the image and ending at the red points. The latent space embedding created by our algorithm is shown on the right. As expected, this embedding essentially "unrolls" the trajectories, stretching them out so that task time, or progression through the maze, maps from left to right on the horizontal axis. The vertical axis provides a dimension for variation between examples. The blue line represents a candidate plan, created as a series of line segments stretching from the start to the goal in the latent space. Discontinuities result, however, when this plan is lifted back to the original workspace in the left image. Although a post-processing step could be applied to smooth the planned trajectory, it is not clear whether a smoothed version would follow the nuances of the demonstration trajectories, or even maintain safety.

## IV. Approach

The approach described in this paper addresses these issues by planning directly in the work space. We still use the neighbor graph to determine correspondences between trajectory points, but rather than planning in the latent space, interpolation and distance calculations are performed in the original (higher-dimensional) work space. Our planner interpolates between the multiple demonstration trajectories, eliding inconsistencies (errors and suboptimal motions) while maintaining the essential geometric characteristics of the demonstrations.

As the first step in neighbor finding, we uniformly sample each trajectory at fixed distances in the work space. The sampling is done to compensate for temporal differences in the demonstrations (this assumes that dynamics are not fundamental to task achievement). Then, heuristics (described in detail in [1]) are used to find correspondences between neighboring points. The heuristics attempt to find local coherence between trajectories; for instance, encoding the idea that corresponding points are more likely to have nearby predecessors that also correspond. The result is a graph where connections between trajectories (usually) indicate semantically meaningful matches, while points that remain unmatched typically result from imperfections in the demonstrations, such as jitter or small detours.

The planning algorithm proceeds iteratively by calculating an action to take based on a set of neighbor points and then choosing a new set of neighbor points based on that action, until a goal state is reached. The initial set of neighbor points is chosen simply as those nearest the start state of the robot. The action is computed as a locally weighted average of the actions associated with each point in the neighbor set (Fig. 4(b)). Since the points in the neighbor graph are sampled from the demonstration trajectories, the actions recorded during demonstration do not necessarily correspond exactly to those points. Instead, actions are computed as the vector from the current neighbor point to its subsequent point in the same trajectory. As a matter of notation, we represent an example trajectory point as $t_i$ and its successor as $t_i^+$. Thus, given $N$ neighbors of a plan point $p$, their weights $w_n$, normalization factor $W$, and the subsequent plan point $p^+$ are calculated as:

$$w_n = |t_n - p|$$
$$W = \sum_{n=0}^{N} w_n$$
$$p^+ = p + \sum_{n=0}^{N} \frac{w_n}{W} \left(t_n^+ - t_n\right)$$

The next step is to select a new set of neighbors. These are not merely the closest points to $p^+$, but must also represent portions of the demonstration trajectories at semantically equivalent states during execution of the task. Since distinct portions of the trajectories may appear in close proximity in non-Markovian regions (c.f. Figure 1) we need to rely on points that are close geodesically in the neighbor graph. However, because the demonstration trajectories may have small imperfections, rather than simply advancing to the set of points subsequent to the previous neighbors, we search

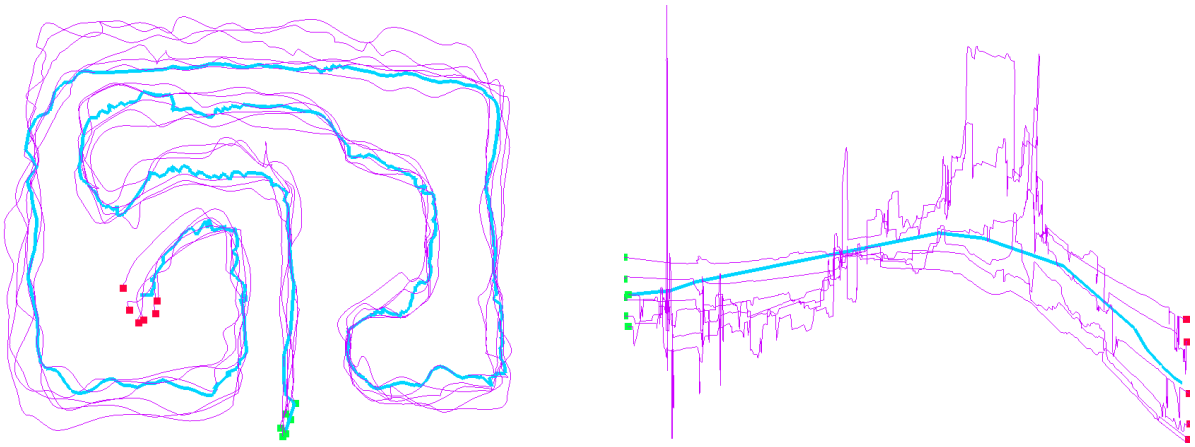


Fig. 3. Traces of the end-effector position while traversing the wire maze, shown in the workspace (left) and in the learned reduced dimensionality space (right). The light blue trace is a naïve plan created in the reduced dimensionality space, then lifted to the original workspace.

near $p^+$, removing neighbors that are too far away and adding new neighbors that are nearby in both metric and graph distance (Fig. 4(c)). In our experiments, we found that searching for points within $d_u$ (the trajectory sampling distance) of $p^+$ and three graph links from the previous neighbor set gave good results.

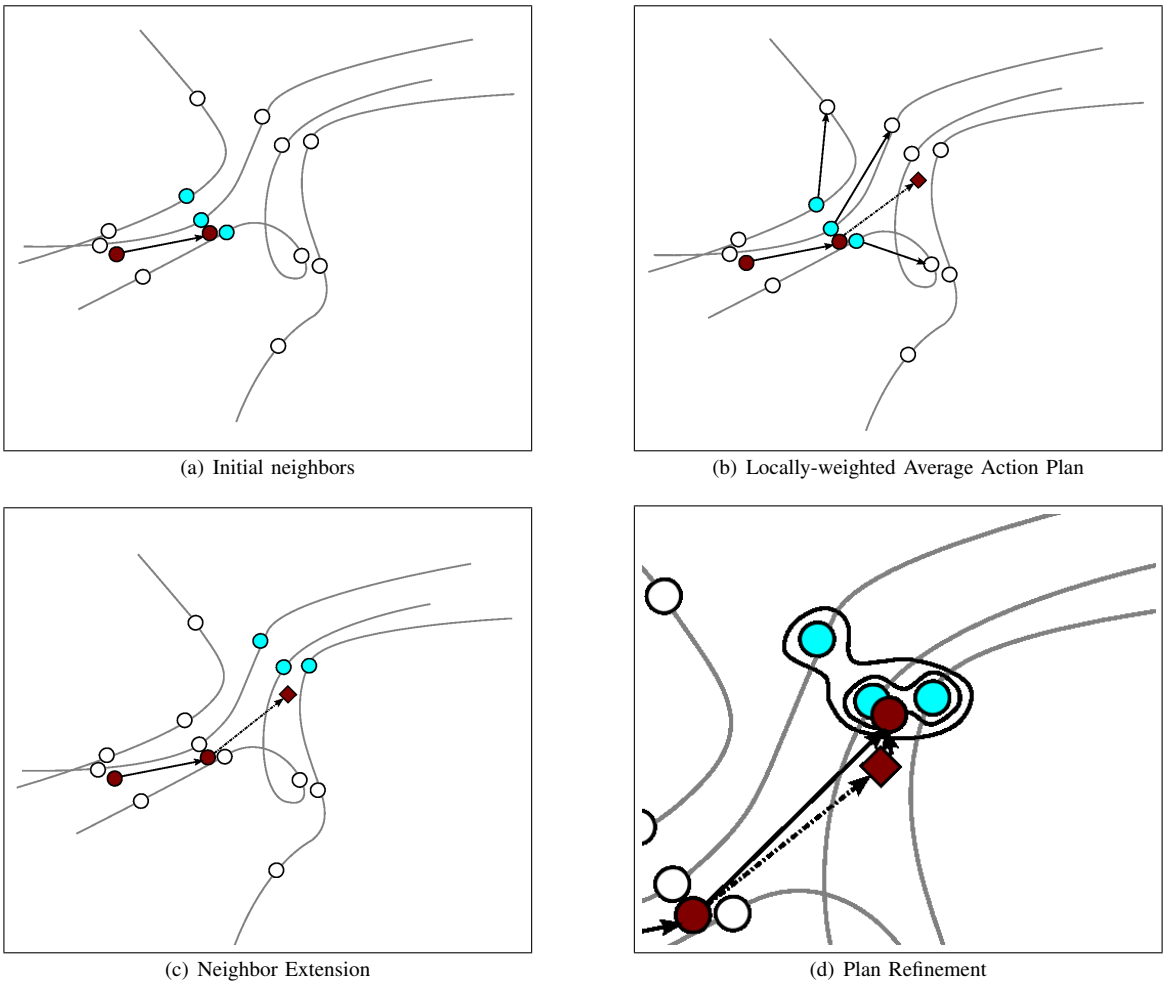
Finally, we refine our choice of $p^+$. While the action computed from the neighbor set faithfully reproduces the actions demonstrated by the teacher at that point in the task, undesired actions are represented as well. We thus adjust $p^+$ toward a position that is both representative of the average action performed by the teacher and close to the new neighbor set. To do this, we place a Gaussian over each new neighbor and use gradient ascent to refine the position of $p^+$. Figure 4(d) illustrates level sets of this reward function around the new neighbors. The new plan point is refined to lie at a local peak of this function.

This approach to planning ensures that interpolation occurs only between demonstrated points that are similar in pose and graph distance: precisely where interpolation is expected to safely respect both geometric and non-geometric constraints. In addition, we use a volumetric model of the robot to ensure safe operation. A CAD model of the robot is used to calculate the volume of space that the robot occupies as it moves through the demonstration trajectories. These swaths of space are known to be free of physical obstacles. Then, during planning, we check whether the robot stays within the union of the swaths all along the path. If not, we use a signed distance field (SDF) [26] to further adjust the position of the planned point $p^+$. An SDF is an occupancy grid that, in addition to a bit indicating the safety of a given grid cell or voxel, provides a vector pointing to the nearest safe cell. This provides a computationally efficient means for adjusting trajectory points that stray into unsafe regions.

An important concern when planning is *bifurcations* in the neighbor graph: areas where demonstration trajectories diverge into two (or more) qualitatively distinct strategies. A common cause of bifurcations is physical obstacles that the teacher must circumvent. In a two-dimensional workspace, the potential for such obstacles may be reliably detected by examining the swaths of workspace occupied during training. More simply, the teacher may be instructed to demonstrate only trajectories that follow the same path around and between obstacles. If this instruction is observed, we can be assured that interpolating between any demonstrations must be safe (at least with respect to geometric constraints). All such demonstrations are *homotopic* because there exists a safe, continuous deformation between any pair of paths while keeping start and end points fixed [27]. If two demonstrations take different paths around an obstacle, say one to the left and the other to the right, a continuous deformation between them would pass through the obstacle, which is clearly not desirable.

This strategy is demonstrated in the plane by [20], but does not extend to higher dimensions. For instance, if the previous example were extended to three dimensions, there may be a safe path above the obstacle. Thus, the paths to the left and the right could safely be continuously deformed through this third path. The paths to the left and the right are thus homotopic, but a simple linear interpolation is not safe. In fact, the interpolation between them may be arbitrarily complex. A robot learner should ideally reach the same conclusion suggested by our intuition: these paths follow separate strategies, and should be treated separately. Moreover, the learner should take advantage of the teacher's knowledge in order to determine whether one strategy should be preferred over the other.

Using the SDF to detect bifurcations is a tempting, but incomplete, approach. We might produce trial robot poses by interpolating between two trajectories, then test whether these poses are safe. However, this will detect only those bifurcations caused by physical obstacles. For a more general approach, we once again rely up the neighbor graph. A bifurcation in demonstration strategies should appear as a localized partition in the graph. Example trajectories that are linked in the graph at some point in time will no longer be



(a) Initial neighbors

(b) Locally-weighted Average Action Plan

(c) Neighbor Extension

(d) Plan Refinement

Fig. 4. (a) The graph-based planning algorithm begins with a partial plan (red points) and a set of neighbors (blue points) selected from among demonstration trajectories (blue and white). (b) The initial estimate for the next plan point (red diamond) is computed using the locally-weighted average of neighbor actions (black arrows). (c) Neighbors (blue) are selected for the new plan point. (d) The pose of the new plan point $p^+$ (red circle) is refined using the new set of neighbors.

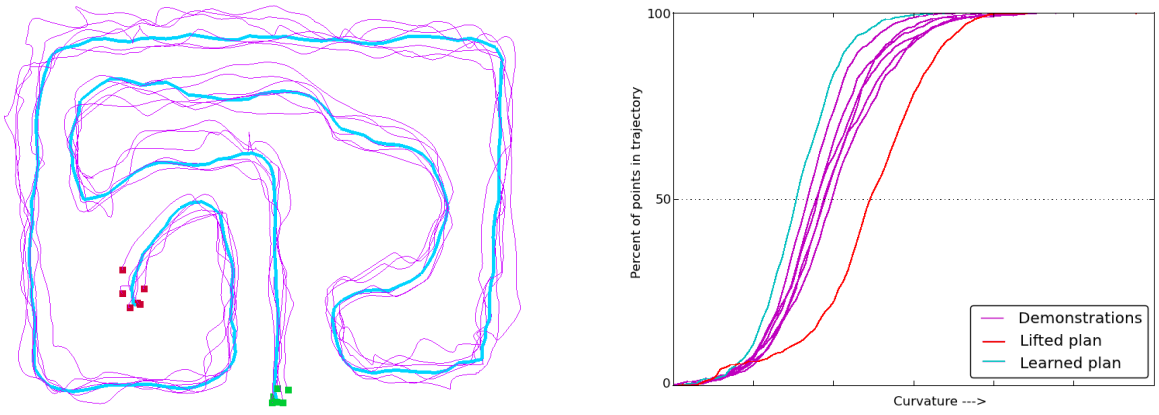



Fig. 5. The example trajectories (purple) from figure 3 with a new plan in blue. The graph on the right shows a cumulative curvature plot, illustrating discontinuities of the lifted plan (in red), while the new plan (in blue) is smoother than the examples.

linked after the bifurcation. We use the normalized cuts graph partitioning algorithm [28] to detect these locations. This algorithm compares the number of links within a partition, or cluster, to the number of links between clusters. The links between clusters are *cut* to form partitions. A good partition should cut relatively few links, with a large number of links

remaining within clusters.

To apply this algorithm to trajectories, we iterate over the points of each example trajectory to find clusters of points that are densely linked to other examples relative to some point in the near future. In our experiments, we compared the interconnections at each query point to the interconnections that exist between ten and 20 steps in the future. Since cuts must occur between trajectories, all the points along a single trajectory are coalesced into a single graph node and the best normalized cut partition is calculated for each such node. In most cases, when no bifurcation exists, the normalized cuts algorithm will successfully assign all trajectories to the same cluster. Otherwise, a high score will indicate that a partition may exist. When many high scores occur within a few graph links of one another, this collection of points is considered to be a *cut neighborhood*, a region in which a bifurcation is likely to exist.

To handle bifurcations, we use *active learning*, where the robot learner asks the human teacher which branch of the bifurcation is to be preferred. Rather than present graphical representations of the trajectories on a computer screen, which may be hard for novices to interpret, the robot learner instead executes partial *trial trajectories* to demonstrate the branches of the bifurcation. In this way, the user is given a clear conception of what the robot plans to do. The user may indicate that one plan or the other is to be preferred, and future plans will follow that branch of the bifurcation. That is, when a future plan encounters points from the *cut neighborhood*, points from the preferred branch are used as neighbors, but points from the other branch are not. The user may also indicate "no preference," in which case the planner will choose a branch randomly, weighted by the number of demonstrations provided in each branch.

A final possibility is that the user discerns no distinction between the two branches. This may occur if the provided trajectories are widely separated, causing the robot to discern a bifurcation where none actually exists. In such cases, the robot requests a new example demonstrating the possibility of planning in the (currently) unknown region. The learner solicits this additional demonstration by creating a plan that approaches the bifurcation and proceeds through it, averaging the trial trajectories, until it reaches a point no longer known to be safe. At this point, the user is asked to continue the task, essentially demonstrating the equivalence of the two strategies by providing r̂ajectory that bridges the gap. If the learner continues to detect bifurcations, additional examples are requested.

## V. Experimental Results

This planning algorithm has been tested in two experimental domains. The wire maze domain (Figure 2) provides a testbed for comparison with plans created in the reduced-dimensionality latent space. Figure 5 illustrates a plan in the maze domain created by our current approach. This plan compares favorably to the jagged lifted plan of Figure 3, produced by our previous approach [1]. To quantify our claim that the current planning approach produces smooth plans,

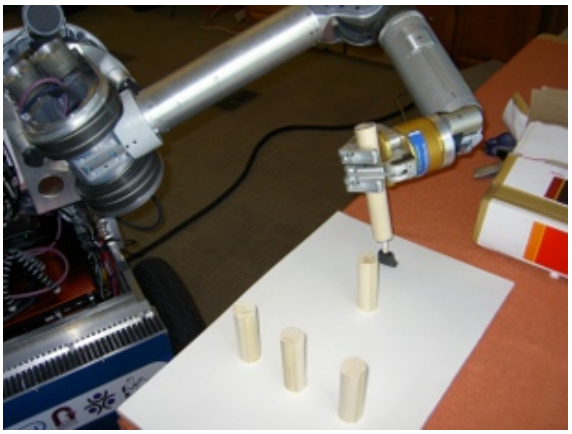


Fig. 6. The artist domain, in which the task is to sweep a paint brush across the board, while avoiding obstacles.

Figure 5, right, is a cumulative curvature plot that shows the percentage of points with curvature less than or equal to the value on the horizontal axis. The red curve, representing the original plan lifted from the latent space, is shifted to the right of the example trajectories, indicating that more points in this trajectory have higher curvature. The blue line, representing the new approach, is to the left of the example trajectories, indicating that it is smoother than all of the examples. This is mainly because our new approach averages out the imperfections in the demonstration trajectories. While not an explicit objective of the planner, this smoothing also decrease trajectory length. On average, in this domain, the new approach produces plans that are 1.5% shorter than the demonstrations.

We also investigated the effects of adding more demonstrations. An untrained user provided six demonstrations of the wire maze task (mean curvature 3.86). With just two examples, the planner produced a plan with mean curvature of 3.86. Adding in one extra trajectory at a time, the mean curvature decreased monotonically (and approximately linearly), reaching 3.2 using all six demonstration trajectories.

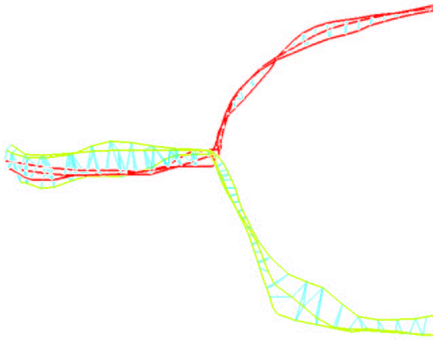


Fig. 7. A detailed view of the neighbor graph at a bifurcation.

In the second domain, which we call the *artist* domain (Figure 6), the user is asked to paint a line across a board with

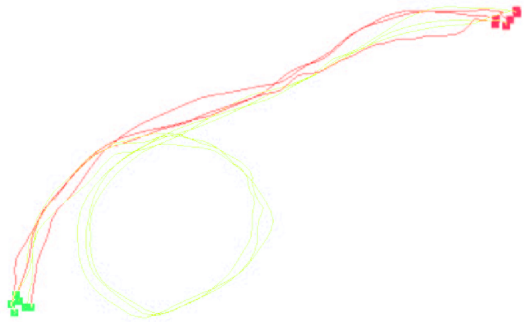


Fig. 8. Example trajectories in the artist domain. Half the examples loop around an obstacle.

vertical dowel rods protruding. This domain was developed to investigate the detection of bifurcations and the strategy for planning through them. In particular, multiple strategies are possible for traversing between and around the dowels. Two sets of example trajectories are shown in Figures 7 and 8. Figure 7 shows six demonstrations that split in half. The trajectories are shown in yellow and red, indicating their partitioning into two groups at the bifurcation, and their neighbor links are shown in blue. The six examples in Figure 8 follow similar paths across the space, but half of them take a detour in the form of a loop around one of the dowel rods. This optional detour is correctly detected as a bifurcation. Depending on the branch preferred by the user, the learner's future plans may, or may not, include this loop.

A user study was run to evaluate the ease and efficiency of training, and to test the bifurcation detection technique. 18 participants, with varying degrees of experience with robots, were recruited from Carnegie Mellon. 11 had no prior experience with robots, 4 had general experience with robots, and 3 considered themselves experienced with robotic manipulation. They were requested to provide three demonstrations each of two different strategies for achieving the task. Most complied, but one provided only a single strategy and one demonstrated four (Figure 9). The system successfully detected the bifurcations, except in the case where there was only a single trajectory for the strategy. At the end, the users filled out a survey on their experience, consisting of 13 questions on a 5 point Likert scale. Chronbach's alpha was used to cluster responses that were internally consistent, and we ended up with four general categories: ease of programming, quality of plans, training questions (the robot's active learning) and effectiveness of learning. With respect to ease of programming, all users felt that the approach was effective (mean 1.22) but inexperienced users rated the ease of use significantly higher than more experienced users. In terms of quality of plans, the users rated the plans as reasonably good (mean 1.861), although the 3 users with manipulator experience rated the plans worse (2.5, but that did not reach significance). The responses to the adequacy of the robot's training questions tended towards neutral (2.5), but with large variation, and the effectiveness

| Dataset | Path Length Mean | Path Length Std. Dev. | Curvature Mean | Curvature Median |
|---|---|---|---|---|
| **maze:** | | | | |
| demonstrations | 1.52 | 0.053 | 3.83 | 3.75 |
| planned | 1.50 | 0.060 | 3.20 | 3.16 |
| **artist:** | | | | |
| demonstrations | 0.702 | 0.211 | 3.64 | 2.83 |
| planned | 0.608 | 0.154 | 1.82 | 1.43 |

TABLE I

USER TRIAL STATISTICS

of learning question had a mean of 1.611, indicating general agreement that the robot's plan followed the strategy the teacher chose to convey. The data show a trend towards more positive assessment as experience decreases, but not to statistical significance.

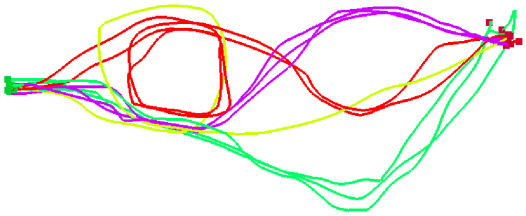


Fig. 9. A set of demonstrations from a single user. Four distinct strategies are shown in different colors. Our system detected two bifurcations, but did not detect the singleton strategy shown in yellow.

In addition, we quantified the improvements in path length and smoothness achieved by the planner in both domains. Table I presents the mean and standard deviation of path lengths across all demonstration and planned trajectories. For curvature, we instead report mean and median; standard deviation is less informative because the trajectories consist of curved and straight segments. In both cases, the planned trajectories are significantly smoother. The improvement is most pronounced in the artist domain, where the task and workspace were far less constrained, and the mean curvature is reduced by 50%. This helps support our claim that the approach is applicable to users who are not robotics experts. Similarly, the planned artist trajectories are, on average, 13% shorter than the average of the demonstrations, although there is wide variance amongst the different strategies (ranging from under 1% to 28% shorter).

## VI. CONCLUSIONS AND FUTURE WORK

The planning by demonstration manipulator motion planning algorithm described here is able to successfully create safe, smooth, novel plans using only demonstrations provided by a domain expert and a volumetric model of the robot. This represents a feasible method for novices in robotics to train a robot to perform sophisticated motion tasks without programming or modeling the environment. In particular, the approach enables the teacher to indicate both geometric and non-geometric constraints to the robot learner. While the approach smooths out jitter and inconsistencies, it maintains

the essential characteristics of the demonstrations, including small perturbations that are consistent across the various demonstration trajectories.

One extension is to consider additional objectives, such as path smoothness constraints or non-uniform costs over regions of the work space. The current implementation preserves high-frequency noise only in areas where that noise is correlated between multiple examples. Otherwise, it is eliminated as unintended jitter when, in fact, small random motions may be necessary for some sorts of tasks. Another extension is to apply our approach to other types of robotic platforms. Although our focus has been on redundant robot manipulators, the approach is equally applicable to mobile robot platforms. Planning on a non-holonomic base is likely to present a unique set of challenges, however, particularly if dynamics are to be considered. A significant extension is to learn safe trajectories in the presence of moving obstacles. While this would necessitate detecting obstacles, we still would not have to model them explicitly within the planner. Finally, we would like to analyze our approach more formally – the approach uses a number of heuristics and parameters, and it would be useful to investigate how these choices affect performance and the guarantees associated with the approach.

In summary, we believe that our approach will enable domain experts to teach robots effectively and efficiently, with minimal training in robotics. As a result of this, and similar efforts, we anticipate that autonomous robots will become applicable to a much wider variety of domains.

## VII. Acknowledgments

This work was supported by AFOSR grant #FA2386-10-1-4138 and NASA award NNX06AD23G under subcontract Z627402.

## References

[1] N. A. Melchior and R. Simmons, "Dimensionality reduction for trajectory learning from demonstration," in *Proceedings International Conference on Robotics and Automation*, May 2010.

[2] H. Asada and H. Izumi, "Automatic program generation from teaching data for the hybrid control of robots," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 166–173, 1989.

[3] H. Asada, "Teaching and learning of compliance using neural nets: representation and generation of nonlinear compliance," in *Proceedings International Conference on Robotics and Automation*, vol. 2, 1990, pp. 1237–1244.

[4] J. Chen and A. Zelinsky, "Programing by demonstration: Coping with suboptimal teaching actions," *International Journal of Robotics Research*, vol. 22, no. 5, pp. 299–319, 2003.

[5] M. Stolle and C. Atkeson, "Policies based on trajectory libraries," in *Proceedings International Conference on Robotics and Automation*, 2006, pp. 3344–3349.

[6] J. Aleotti, S. Caselli, and G. Maccherozzi, "Trajectory reconstruction with nurbs curves for robot programming by demonstration," in *Proceedings International Symposium on Computational Intelligence in Robotics and Automation*, 2005, pp. 73–78.

[7] H. Friedrich, J. Holle, and R. Dillmann, "Interactive generation of flexible robot programs," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1998, pp. 538–543.

[8] D. Bentivegna and C. Atkeson, "Learning from observation using primitives," in *Proceedings International Conference on Robotics and Automation*, vol. 2, 2001, pp. 1988– 1993 vol.2.

[9] G. Hovland, P. Sikka, and B. McCarragher, "Skill acquisition from human demonstration using a hidden markov model," in *Proceedings International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 2706–2711.

[10] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Transactions on Robotics*, 2008.

[11] J. D. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings International Conference on Robotics and Automation*, May 2001.

[12] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proceedings National Conference on Artificial Intelligence*, 1990, pp. 796–802.

[13] B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *Proceedings International Conference on Human-Robot Interaction*. Arlington, Virginia, USA: ACM Press, 2007, pp. 57–64.

[14] S. Chernova and M. Veloso, "Tree-based policy learning in continuous domains through teaching by demonstration," in *Modeling Others from Observations: Papers from the AAAI Workshop*, G. Kaminka, D. Pynadath, and C. Geib, Eds. American Association for Artificial Intelligence, 2006, pp. 24–31.

[15] N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19*. Cambridge, MA: MIT Press, 2007.

[16] A. Ude, C. Atkeson, and M. Riley, "Planning of joint trajectories for humanoid robots using b-spline wavelets," in *Proceedings International Conference on Robotics and Automation*, vol. 3, 2000, pp. 2223–2228 vol.3.

[17] C. Lee, "A phase space spline smoother for fitting trajectories," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, pp. 346–356, 2004.

[18] H. Mayer, I. Nagy, A. Knoll, E. Braun, R. Lange, and R. Bauernschmitt, "Adaptive control for human-robot skilltransfer: Trajectory planning based on fluid dynamics," in *Proceedings International Conference on Robotics and Automation*, Rome, Italy, 2007.

[19] E. Drumwright, O. Jenkins, and M. Matarić, "Exemplar-based primitives for humanoid movement classification and control," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 2004, pp. 140–145.

[20] N. Delson and H. West, "Robot programming by human demonstration: adaptation and inconsistency in constrained motion," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1996, pp. 30–36.

[21] ——, "Robot programming by human demonstration: the use of human variation in identifying obstacle free trajectories," in *Proceedings International Conference on Robotics and Automation*, vol. 1, 1994, pp. 564–571.

[22] S. Calinon and A. Billard, "Recognition and reproduction of gestures using a probabilistic framework combining PCA, ICA and HMM," in *Proceedings International Conference on Machine Learning*, 2005, pp. 105–112.

[23] O. C. Jenkins and M. J. Matarić, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *Proceedings International Conference on Machine Learning*. Banff, Alberta, Canada: ACM Press, 2004, p. 56.

[24] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, Dec. 2000.

[25] A. Tsoli and O. C. Jenkins, "Neighborhood denoising for learning high-dimensional grasping manifolds," in *International Conference on Intelligent Robots and Systems*, Nice, France, Sep 2008, pp. 3680–3685.

[26] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[27] J. R. Munkres, *Topology*. Upper Saddle River, NJ: Prentice Hall, 2000.

[28] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Proceedings Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington, DC, USA: IEEE Computer Society, 1997.