2010 IEEE-RAS International Conference on Humanoid Robots
Nashville, TN, USA, December 6-8, 2010

Combining Motion Planning and Optimization for Flexible Robot Manipulation

Jonathan Scholz and Mike Stilman

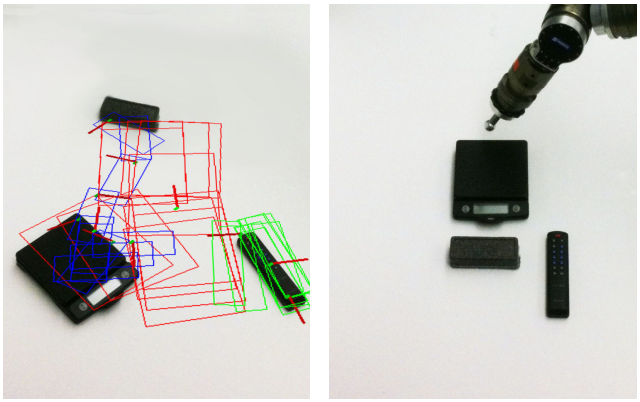
Abstract— **Robots that operate in natural human environments must be capable of handling uncertain dynamics and underspecified goals. Current solutions for robot motion planning are split between graph-search methods, such as RRT and PRM which offer solutions to high-dimensional problems, and Reinforcement Learning methods, which relieve the need to specify explicit goals and action dynamics. This paper addresses the gap between these methods by presenting a task-space probabilistic planner which solves general manipulation tasks posed as optimization criteria. Our approach is validated in simulation and on a 7-DOF robot arm that executes several tabletop manipulation tasks. First, this paper formalizes the problem of planning in underspecified domains. It then describes the algorithms necessary for applying this approach to planar manipulation tasks. Finally it validates the algorithms on a series of sample tasks that have distinct objectives, multiple objects with different shapes/dynamics, and even obstacles that interfere with object motion.**

I. Introduction

Service manipulation is a critical direction for future robotics research that will make it possible for robots to assist humans in problems ranging from house cleaning to collaborative factory automation. In contrast to traditional factory manipulators, service robots will be faced with two challenges: (1) unfamiliar objects and (2) abstract goals. Uncertainty about the types of objects and their response to robot actions is unavoidable when working in natural environments. Robots must learn about objects in addition to planning interactions with them. Robots must also accept abstract goals from a user or programmer through intuitive mechanisms for a broad range of domains such as *setting a table*. We propose an approach that handles both of these challenges by expressing service manipulation as task-space optimization and bridging two forms of optimization through sampling-based planning.

Our approach combines the classical reinforcement learning (RL) formulation for abstract problems with efficient model learning and model-based planning. In reinforcement learning [21], robot goals are given as optimization criteria that can be specified with intuitive rewards. Furthermore, actions are allowed to have uncertain effects. However, as we describe in Section II, standard methods for solving RL problems do not scale to high dimensional tasks such as service manipulation. In contrast, research in model-based planning [18], [19] explicitly addresses high-dimensional tasks, but typically focuses on user-specified goal configurations and deterministic actions. Our solution combines the strengths of both in order to accommodate uncertain models and abstract goals in high-dimensional spaces.

Interactive Computing, Georgia Institute of Technology, 801 Atlantic Drive,Atlanta, GA 30332 jkscholz@gatech.edu mstilman@cc.gatech.edu



(a) Initial State, Plan (b) Final State

Fig. 1. Table configurations before and after executing a plan computed to optimize the table-cleaning objective function. Colored outlines depict the sequence of states visited during execution for each object.

In Sections II and III, we pose the service task problem identically to model-free RL. However, instead of seeking an optimal policy we focus on finding a feasible plan that achieves the abstract task objectives. This trade-off offers greater scalability to high-dimensional manipulation problems than current methods in reinforcement learning. Section II places our research in the context of existing methods in planning and RL. Section III discusses the basic requirements for planning in task-space, and present an algorithm that meets these requirements. We conclude in Section IV by showing an implementation of our method on a robot arm which is required to set and clean a table.

II. Related Work

Standard motion planning algorithms such as Probabilistic Roadmaps (PRM) [13] and Rapidly-exploring Random Trees (RRT) [14] rely on goals that are specified as exact states and actions that have precisely known consequences. Both of these limitations make it difficult to apply such planners directly to service manipulation problems where goals and dynamics are under-specified.

Model-free Reinforcement Learning provides a more suitable representation for service manipulation, because it offers a mechanism for learning transition dynamics and reward functions online [21]. However, these methods are difficult to apply to manipulation problems due to the high cost of exploring with a real robot [1]. Thus, even state-of-the-art methods in RL are restricted to problems with much lower

dimensionality (2-3D) than ones that are commonly found in service tasks.[17]

A common method that addresses the challenges of model-free methods is to pre-specify a model or motion primitive for robot actions [11]. Model-based learners [2], [3] can find entire policies for high-dimensional spaces based on local optimization over action primitives and their parameters. However, these methods typically take the form of *learning from demonstration* [7], [22], which imitates existing plans generated by humans. This makes it difficult to generalize primitive learning and autonomous planning to new types of objects or novel tasks.

An approach to solving high-dimensional problems that does not require human examples relaxes the demands on global optimality. Kaelbling et al show that planning can be accelerated in MDP's by narrowing the state-space according to the time the planner has available [6]. Katz uses a *relational* RL approach for compactly representing states in a planar manipulation task [12]. However, the basic challenge with these approaches is that *optimality* is fundamentally harder to achieve than *feasibility* when computing a plan. In contrast, model-based planners learn and use dynamic models to identify collision-free paths between start and goal states [4], [5], [19]. The drawback to these methods is that in giving up the pursuit of an optimal plan they also remove the ability to easily specify abstract goals.

Evidence supporting the scalability of optimization over a space action primitives can also be found in the domain of motor neuroscience. Wolpert [24] shows that humans use primitives and forward models to plan motor tasks for highly redundant systems. Motor basis functions underly much of biological motion [9] and tool use [10] in the human brain. Furthermore, there exists strong evidence that humans solve task-level and motor-level challenges though optimization processes. A good overview is given by Todorov[23].

Our algorithm is inspired by the modern understanding of reinforcement learning, and analogous processes in the human brain. We combine model-based planning with optimization. Our method accepts the original formulation of reinforcement learning where both the goal states and the consequences of actions are under-specified. We achieve task-space planning by learning the dynamics of a set of motion primitives, which we refer to as *forward models* following [24]. We then utilize the efficient search properties of RRT to quickly find plans that lead to optimal states.

III. Planning as Optimization

In the context of service manipulation, the two key advantages of reinforcement learning over classical path planning are that abstract goals can be specified through intuitive rewards and that actions can have uncertain outcomes. Because tasks like cleaning a table require the robot to displace multiple objects, the configuration space for planning has exponential dimension in the number of objects [20]. This makes it infeasible to apply standard RL strategies. Yet, we observe that reaching an optimal world configuration is more important than finding the optimal way to reach it. We use this insight to decouple the RL problem into three tasks: (1) determining the goal, or the optimal configuration, (2) finding forward models for robot actions and (3) planning to use the actions to reach the goal.

In this paper, Sect. III-A and III-B describe optimization procedures. Sect. III-C describes a planning algorithm which seeks a feasible, but not necessarily optimal plan for robot actions. We consider a common service task: cleaning a table. Cleanliness is naturally expressed as an objective function over object poses. We focus on cases in which the objective function is provided by a human programmer. However, as with reward-function learning in RL, these criteria can also be extracted from interactions with humans or the environment. First, we present a method for formulating the objective function. Second, we present a method for learning forward models of object motion. Finally, we combine these elements with sampling-based planning.

A. Objective Function Specification

Table-setting for an arbitrary number of guests is an abstract goal. This goal is fundamentally distinct from positioning plates at desired locations since it is the spacing between the dinnerware that matters to the guests rather than their precise locations. Using our method, the programmer can specify the goal as an abstract optimization metric.

Without loss of generality, consider a dinner where n guests must be given n plates and m platters must be placed at the center of the table. The programmer should be able to state the following objectives:

1) The plates should be located far from each other.
2) The platters should be at the center of the table.
3) The platters should be aligned parallel to the table.

The same plate positions will not satisfy these criteria for different numbers of guests. However, the optimization criteria indicated by the objectives are easily formulated with Eq. 1-3. We define two sets of objects: plates, P, and platters, Q. Each object location is parameterized by position and orientation $\{x, y, \theta\}$. For a rectangular table, parallel to the frame of reference its center is (x_G, y_G).

$$c_{dist} = -\sum_{p1\in P}\sum_{p2\in P}\sqrt{(x_{p1}-x_{p2})^2+(y_{p1}-y_{p2})^2} \quad (1)$$

$$c_{pos} = \sum_{q\in Q}\sqrt{(x_q-x_G)^2+(y_q-y_G)^2} \quad (2)$$

$$c_{ortho} = \sum_{q\in Q}|(\text{mod}(\theta_q, \frac{\pi}{2})| \quad (3)$$

The programmer or high-level planner should also specify environment constraints. For example, Eq. 4 limits all objects to the table dimensions:

$$x_{min} \le x \le x_{max}, y_{min} \le y \le y_{max}, \quad (4)$$

More generally applicable optimization criteria can also be specified. For instance, the table center (x_G, y_G) can be inferred from the table dimensions as shown in Eq. 5.

$$x_G = \text{argmax}_x(\min(x-x_{min}), (x_{max}-x))$$
$$y_G = \text{argmax}_y(\min(y-y_{min}), (y_{max}-y)) \quad (5)$$

The overall objective for cleaning a table is simply the sum of our intuitive criteria as given in Eq. 6. The weights α, β, γ must be specified with regard to the relative importance of the subtasks.

$$C_{table} = \alpha\, c_{dist} + \beta c_{pos} + \gamma c_{ortho} \tag{6}$$

B. Action Model Learning

The second optimization problem we consider is determining the relationship between robot actions and their effects on objects. Here we consider only collision free motions, and handle collision-avoidance in the planner. This approach requires significantly less data than when modeling all possible contacts between an unbounded number of objects. We now describe the action model learning procedure which allows the planner to reason about the outcomes of its actions in various states.

In our implementation, the robot was equipped with a ball-shaped end-effector that could only push objects. In other domains, the robot might grasp objects. In either case, a service robot will encounter new objects. Different pushes or grasps will have distinct effects on object displacement. For simplicity, consider the pushing domain. Our actions change the pose of an object, O, placed on the table at $\{x_O, y_O, \theta_O\}$.

In order to encode uncertainty in action outcomes we follow the definition of Markov Decision Processes (MDP). Our action model is a function that maps state and action to a probability distribution over states. Given some state-space S and actions A, a planner must know the probability of a given outcome of any action in any state:

$$P(s'|a, s)\ \forall\ s \in S,\ a \in A \tag{7}$$

Our approach automatically generates $P(s'|a, s)$ from data obtained by exploration. In our domain, object displacement has a direct relationship to the motion of the end-effector relative to the object rather than relative to the world. We defined six actions relative to the object as shown in Figure 5. We then compute *probability models of displacement*:

$$P(\Delta s|a, o)\ \forall o \in O,\ a \in A \tag{8}$$

To estimate $P(\Delta s|a, o)$ for every action applied to every object, we implemented the optimization algorithm shown in Figure 2. LEARN_MODEL incrementally completes a tabular probability distribution that relates each action, $a \in \mathbf{A}$, to the displacement, Δs, for the object, $o \in O$. On each iteration, the robot applies an action to the object, observes the result using an overhead camera, and uses UPDATEDISTR to update the probability distribution, $P(\Delta s|a, o)$, as shown in Eqs. 9-10. Our model represents the probability of displacement with a normal distribution. This requires UPDATEDISTR to update the distribution mean and variance. For each variable in state $\mathbf{s}$, $\{v = x, y \text{ or } \theta\}$, the n^{th} update is given as follows:

$$\mu_v^n = \frac{n\mu_v^{n-1} + \Delta v}{n} \tag{9}$$

$$\sigma_v^n = \sqrt{\frac{(n-1)\,\sigma_v^{2(n-1)} + (\Delta v - \mu_v^n)\,(\Delta v - \mu_v^{n-1})}{n}} \tag{10}$$

```
LEARN_MODEL(o, A, s_init)
1  s = s_init
2  for i ← 1 to |A|
3  do while σ² > σ²_ref
4      do (Δs, s) = APPLY_ACTION(a_i, O)
5          P(Δs|a_i, o) = UPDATEDISTR(P(Δs|a_i, o), a_i, Δs)
6          σ² ← VARIANCE(P(Δs|a_i, o))
```

Fig. 2. Pseudo-code for learning forward models. Note that Δs encodes state transitions in object coordinates.

Iteration terminates when the variance of the distribution reaches a desired threshold, σ_{ref}^2. This criterion identifies that the model is sufficiently accurate for planning. In our experiments, we empirically determined that $\sigma_{ref}^2 = 0.4$ yielded a reasonable compromise between execution times for learning and plan execution for our table-top domain.

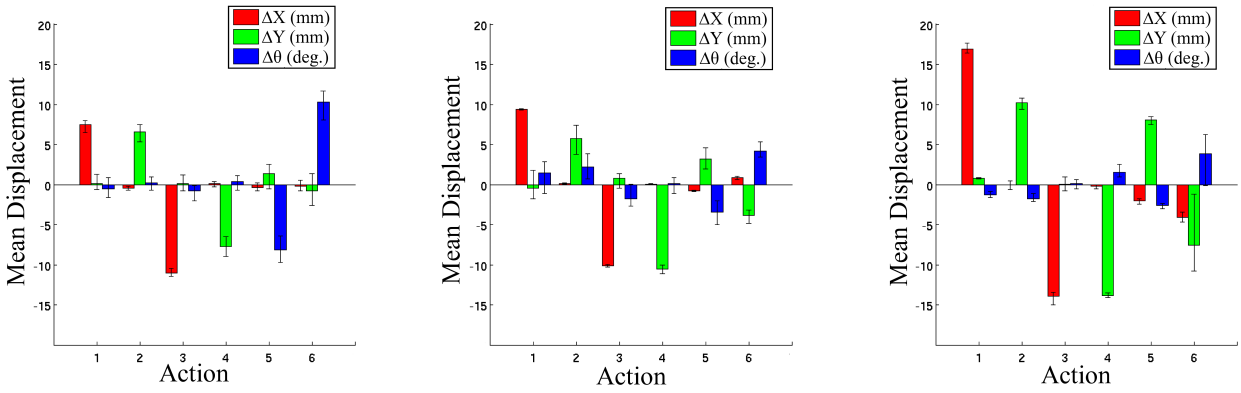
The model achieved by the algorithm for three objects is illustrated in Figure 3. The figure shows mean displacements and confidence intervals when the robot applies our six actions to each object. Notice the significant variation in these parameters. This is precisely the reason that learning object models is essential to the construction of valid plans.

C. Task-Space Planning

Given a task-level goal and a forward model that relates robot actions to world effects, the remaining challenge is to produce a planner that efficiently finds a sequence of actions that achieves an optimal configuration. In order for this optimization process to be useful it must respect the physical limitations of the environment/robot and be reachable from the initial state of the problem. The first condition is satisfied by imposing collision and boundary constraints (Eq. 4). To satisfy the second, the optimizer must also respect the motion constraints specified by our models.

We present a motion planning algorithm that directly searches the optimization landscape using models of the robot's actions. By combining optimization and motion planning into a single search, we restrict the search to states that are relevant given the robot's available actions. Our algorithm prevents the planner from aiming towards optimal configurations that are unreachable. Furthermore, since the planner always knows the best state in its search graph, it offers more reliable anytime characteristics than approaches that separate goal optimization from planning. The remainder of this section describes the basic functionality of the algorithm, as well as an extension which makes it more efficient when searching large spaces.

Our algorithm, given in Figure 4, extends RRT [15], [14], a probabilistically complete method that is commonly applied to motion planning in high-dimensional spaces. Basic RRTs produce a tree of valid, collision-free configurations for n-dimensional spaces through incremental expansions of the tree towards random configurations. Application of the basic RRT to problems such as manipulator control relies on some form of inverse model for the system to determine how to reach a child node from its parent. When working in task



(a) Eraser (b) Remote (c) Scale

Fig. 3. Model learning results with three objects: a chalkboard eraser, a TV remote, and a digital scale. Bars show the mean and 95% confidence intervals extracted from each object for all degrees of freedom. Actions are those illustrated in Figure 5.

```
TASK_SPACE_RRT(s_init, A)
 1  for i ← 1 to |O|
 2  do Model ← LEARN_MODEL(O_i, A, s_init)
 3  T.init(s_init)
 4  for i ← 1 to max_nodes
 5  do s_GD ← DIRECTGD(T)
 6     if RAND() > ε
 7        then s_samp ← s_GD
 8        else s_samp ← RANDOM_CONFIG()
 9     s_near ← NEAREST_NEIGHBOR(s_samp)
10     a* ← ARGMIN_a(ρ (MODEL(s_near, a), s_samp))
11     s_new ← MODEL(s_near, a*)
12     if not IN_COLLISION(s_new)
13        then ADD_VERTEX(s_new)
14             ADD_EDGE(s_near --a*--> s_new)
```

Fig. 4. Pseudo-code for RRT-based task space planner using forward models. T is the search tree, and ρ is the distance function defined in Eq. 11.

space, however, we have no clear model for how to transition between nodes. Instead we modify our approach to perform a forward search over the robot's possible actions. The core idea is that by confining the search to a set of pre-defined actions for which we have controllers, we can both restrict the search to reachable states, and relieve the need for an inverse model. A similar approach is given by Frazzoli [8].

Our method preserves the *rapidly exploring* characteristics standard RRT search with two modifications. First, rather than growing the tree directly towards sampled states, we select the state-action pair which results in a node closest to a sample point according to a weighted distance metric. If this state is valid, we add a directed edge labeled with the appropriate action to the tree. Since the best action can fail the subsequent collision check, we maintain state-action pairs in a queue ordered by distance. The planner iterates through this list and adds the first collision-free pair that is closer than the parent node. This modification yields efficient extensions to new states using only forward models.

The second unique modification is the addition of a conventional gradient descent optimization routine, DIRECTGD, during the planning process. DIRECTGD is a heuristic that searches the objective function directly, without branching over robot actions. Periodically executing this search from leaf nodes in the tree provides a way to quickly discover the nearby local minima of the objective function. Empirically, we found that this heuristic significantly speeds up search towards the global minimum while the random action of the RRT prevents the overall planner from getting stuck in any local minimum. Our technique is inspired by existing RRT heuristics like RRT-CONNECT [14] which try to extend the tree towards a goal or a goal tree. However, in our case the "goals" are the modes of the optimization landscape, and DIRECTGD allows us to explore this space more efficiently. If DIRECTGD returns a more optimal state than one that currently exists in the planner's search tree, it attempts to apply RRT-CONNECT using the robot's actions to find a path to this state.

As with a conventional RRT, our planner contains an ϵ parameter that trades exploration and exploitation. The planner takes greedy steps towards the lowest-cost state identified by DIRECTGD with probability $1 - \epsilon$. In our domain we empirically determined that $\epsilon = 0.8$ gave the fastest convergence to a global optimum.

In addition to the two RRT algorithm modifications we must design a distance metric that accurately reflects the similarity between states:

$$\rho : f(s_1, s_2) \rightarrow \mathbb{R} \tag{11}$$

This distance function is used to query nearest neighbors in TASK_SPACE_RRT (line 9). In task space, the parameters used to describe state are specified by the programmer. They are not required to share a coordinate frame or scale. For our table-top manipulation experiments we empirically determined that weighing orientation parameters $40 : 1$ over position produced the most consistent alignment of objects.

D. Executing Motion Primitives

We designed a set of six simple motion primitives for 2D manipulation with a single-point contact end effector (Figure 5). In order to minimize the branching factor of the

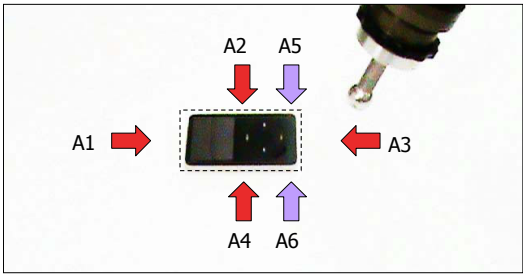
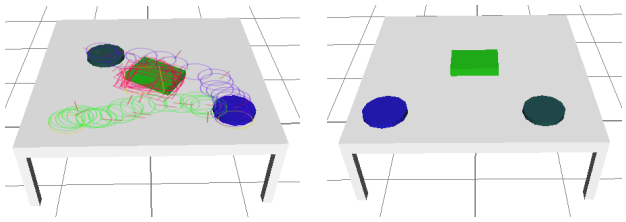


Fig. 5. Six motion primitives defined with respect to a bounding box around a sample object. Arrows indicate motion vectors for the end-effector as defined in the object frame.



(a) Initial Plan (b) Result

Fig. 6. Execution of the table-setting planner in the srLib simulator with two plates and one platter. Colored outlines depict the sequence of states visited during execution for each object.

planner, we defined actions based on relative motions with respect to simplified object geometry.

The robot identified objects in its workspace using a simple threshold-based segmentation scheme with an overhead camera. Objects were extracted by finding closed contours in a mask image. Each object was represented by a bounding box aligned with its long axis and then classified based on the size and aspect ratio of the bounding box. The position and orientation parameters were taken as the centroid and the angle of the box from horizontal, respectively. We defined six primitives to control the three degrees of freedom as independently as possible, one for increasing and one for decreasing each parameter.

Each of the six primitives was defined by a workspace vector in the object frame (Figure 5). During action execution, the robot lifted and lowered its end-effector to the point on the bounding box along the vector direction. It then pushed the object by tracking the vector in workspace for a distance of 5cm. The robot transformed the nominal workspace trajectory into the arm's joint-space with analytical inverse kinematics and tracked the reference with PD control. In order to produce mostly linear translations of the object, actions 1-4 were set as vectors normal to the bounding box at the midpoint of each face. Actions 5-6 were set as the two opposing vectors at 90% of the box length along the long-axis box faces, targeting rotation.

The effect of primitive actions on each of three classes of objects is given by Figure 3. During learning, we found that each action's influence was largely confined to the state parameters it was designed to influence. However, the extent of influence varied greatly due to differences in physical properties. For the scale, rotation actions also produced significant changes in position terms. This was due to the rubber feet on the corners of the scale, which tended to break loose and skip all at once as the robot pushed from the end. One advantage of our approach is that observable dynamics which are difficult to model explicitly are captured through learning and accounted for at plan time.

IV. Experiments

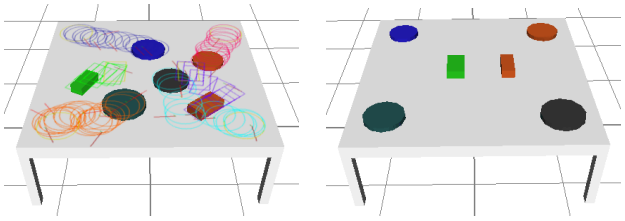
We conducted four experiments using the TASK_SPACE_RRT planner. The first two experiments were simulated table-setting tasks. These were performed in SRLib, which replicated a real world task domain with dynamics and visually perceived object states [16]. We used the SRLib block and cylinder primitive shapes to represent plates, platters, and a table. In both tasks the problem was to optimize the table-setting function (Eq. 6). Figure 6 and 7 present two manipulation plans from our algorithm that generate neat tabletop configurations from random configurations of three and six objects. Notice that the planner gracefully adapts to object quantity and geometry resulting in different final configurations under different circumstances. Figure 10 gives the planning statistics and shows the performance improvement from the GD heuristic. The greatest challenge in larger problems was collision detection when the free-space became increasingly taken up by objects. Most RRT steps, both greedy and random, fail when the planner becomes deadlocked, reducing the algorithm to a slow random search.

For the first demonstration on the physical robot we designed a simple task of positioning an object in the presence of a stationary obstacle. We reused the distance (Eq. 2) and orthogonality (Eq. 3) terms from the previous experiment, and added an additional constraint in the form of a fixed obstacle. The eraser was used for the mobile object, and the hole-punch was labeled as an obstacle (Figure 8). After learning a motion model for the eraser, the robot found a plan that involved pushing the object up and around the obstacle to a goal point on the left side of the table (Figure 8).

To demonstrate a more practical application, the goal our final experiment was to make the robot straighten up a desk. The scale, the remote, and the eraser were distributed randomly around the workspace at the start of the experiment, and it was the robot's job to find a sequence of actions which minimized an objective function for "table-cleaning" with the same form as (Eq. 6). The planner execution for this task can be seen in Figure 1. After learning models for the three objects, the robot proceeded to push the three objects into orthonormal alignment with a center of mass less than two inches from the center of the table. Execution took 13 minutes, and required re-planning 4 times (Figure 9).

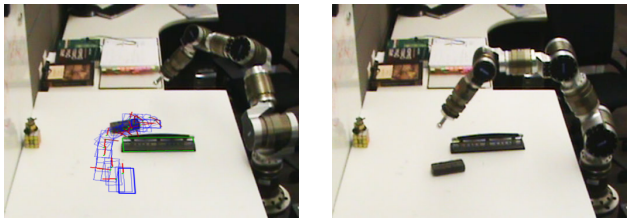
V. Conclusion

We have presented an algorithm that addresses two outstanding challenges for object manipulation in human environments. By defining tasks as constrained optimization, we gave an alternative mechanism to reinforcement learning for handling abstract tasks. We also introduced an approach to learning motion models on-line for efficient planning in arbitrary task domains with a focus on planar manipulation.



(a) Initial Plan (b) Result

Fig. 7. Execution of the table-setting planner in the srLib simulator with four plates and two platters. Colored outlines depict the sequence of states visited during execution for each object.



(a) Initial Plan (b) Correcting from error

Fig. 8. Manipulation in the presence of obstacles

These methods allow for flexible task-space planning independent of the MDP formalism. Our experiments show that the proposed algorithms are able to generate plans for robots that perform abstract tasks over multiple unknown objects.

Future work will explore broader tools and domains that increase the generality of task-space planning by combining planning, learning and optimization. Navigation tasks will globally minimize the distance to tables, power outlets, or maximize distance to obstacles. Manipulation, such as object clustering by similarity, will be expressed compactly as a minimization of differences over a set of object features. The common property between these examples is the need to satisfy a task objective posed as a function rather than a state. Our proposed solutions to problems that are typically addressed through reinforcement learning will broaden the range of intuitive high degree-of-freedom tasks that can be solved by autonomous robots.

	TSet 6	TSet 3	Obst	Clean
n-nodes w/ GD	2161	1438	541	981
n-nodes w/o GD	1898	1399	549	954
time (m:s) w/ GD	0:32	0:18	7:22	13:40
time (m:s) w/o GD	1:34	0:33		
n-steps w/ GD	114	122	44	89

Fig. 9. Selected statistics on planning for 6 object table-setting, 3 object table-setting, 1 object obstacle, and 3 object cleaning.

Task	TSet 6	TSet 3	Obst	Clean
DOF	18	9	3	9
# replans	1	1	2	4

Fig. 10. Degrees of freedom (DOF) and the number of required re-plans for each problem in Figure 9

REFERENCES

[1] B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] C.G. Atkeson and S. Schaal. Robot learning from demonstration. In *Machine Learning-International Workshop then conference*, pages 12–20. Citeseer, 1997.

[3] D.C. Bentivegna and C.G. Atkeson. Learning from observation using primitives. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1988–1993. Citeseer, 2001.

[4] S. Brown and C. Sammut. An architecture for tool use and learning in robots. In *Australian Conference on Robotics and Automation*. Citeseer, 2007.

[5] P. Cheng, J. Fink, and V. Kumar. Abstractions and Algorithms for Cooperative Multiple Robot Planar Manipulation. *Robotics: Science and Systems IV*, page 143, 2009.

[6] T. Dean, L.P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579. Citeseer, 1993.

[7] B. Dillman and P. Steinhaus. ARMAR II–a learning and cooperative multimodal humanoid robot system. *International Journal of Humanoid Robotics*, 1(1):143–155, 2004.

[8] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Proceedings of the 39th IEEE Conference on Decision and Control, 2000*, volume 1, 2000.

[9] SF Giszter, FA Mussa-Ivaldi, and E. Bizzi. Convergent force fields organized in the frog's spinal cord. *Journal of Neuroscience*, 13(2):467–491, 1993.

[10] H. Imamizu, S. Miyauchi, T. Tamada, Y. Sasaki, R. Takino, B. Putz, T. Yoshioka, and M. Kawato. Human cerebellar activity reflecting an acquired internal model of a new tool. *Nature*, 403(6766):192–195, 2000.

[11] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence*, 4(1):237–285, 1996.

[12] D. Katz, Y. Pyuro, and O. Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and Systems*. Citeseer, 2008.

[13] L.E. Kavraki and J.C. Latombe. Probabilistic roadmaps for robot path planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, 53, 1998.

[14] JJ Kuffner Jr and SM LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA'00*, volume 2, 2000.

[15] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*, page 293. AK Peters, Ltd., 2001.

[16] Frank C. Park and Jaeyoung Haan. srlib - snu robot dynamics library. http://r-station.co.kr/srlib/.

[17] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proc. 3rd IEEE-RAS Intl Conf. on Humanoid Robots*, pages 29–30. Citeseer, 2003.

[18] A. Safonova, N.S. Pollard, and J.K. Hodgins. Optimizing human motion for the control of a humanoid robot. In *2nd International Symposium on Adaptive Motion of Animals and Machines (AMAM2003)*. Citeseer, 2003.

[19] A. Shkolnik, M. Levashov, S. Itani, and R. Tedrake. Motion planning for bounding on rough terrain with the littledog robot. In *Submitted to the International Conference on Robotics and Automation, Anchorage, Alaska. IEEE/RAS*, 2010.

[20] M. Stilman and J.J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proc. IEEE Int. Conf. on Humanoid Robotics (Humanoids'04)*, 2004.

[21] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2004.

[22] A.L. Thomaz and C. Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1000. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[23] E. Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7(9):907–915, 2004.

[24] D.M. Wolpert and Z. Ghahramani. Computational principles of movement neuroscience. *nature neuroscience*, 3:1212–1217, 2000.