Pregrasp Manipulation as Trajectory Optimization

Jennifer King, Matthew Klingensmith, Christopher Dellin,
Mehmet Dogar, Prasanna Velagapudi, Nancy Pollard and Siddhartha Srinivasa
The Robotics Institute, Carnegie Mellon University
{jeking,mklingen,cdellin,mdogar,pkv,nsp,siddh}@cs.cmu.edu

***Abstract*—We explore the combined planning of pregrasp manipulation and transport tasks. We formulate this problem as a simultaneous optimization of pregrasp and transport trajectories to minimize overall cost. Next, we reduce this simultaneous optimization problem to an optimization of the transport trajectory with start-point costs and demonstrate how to use physically realistic planners to compute the cost of bringing the object to these start-points. We show how to solve this optimization problem by extending functional gradient-descent methods and demonstrate our planner on two bimanual manipulation platforms.**

I. Introduction

We address the problem of *pregrasp manipulation*, where an object must be manipulated to a feasible or convenient location before it can be grasped. Such situations occur often in the real world where objects are inaccessible or inconveniently placed: a thin book that must be slid to the edge of a table to be grasped and placed in a bookshelf, or a heavy drill that must be pulled close to be grasped and lifted (Fig.1). Pregrasp manipulation makes impossible manipulation tasks possible and hard tasks easier.

In addition to the complexity of motion planning in high-dimensional manipulator configuration spaces, pregrasp manipulation is further complicated by the tight coupling between the two phases of *reconfiguration*, where the object is moved to a grasp, and *transport*, where the grasped object is moved to its goal.

As a consequence, prior work ([27], detailed in Section II) has focused on producing *feasible* solutions to the pregrasp manipulation problem. Some work [4] does explore optimality, but only in a restricted setting of the single pose of the object just before it is being grasped.

Furthermore, these works all rely on the object being *rigidly grasped* at all times. But, as we saw in the above examples, pregrasp manipulation is often achieved not just by grasping but by pulling, pushing, sliding, or other nonprehensile physics-based actions, using the support surface to provide stability.

In this paper, we strive to overcome both of these limitations by developing algorithms to produce *optimal* pregrasp manipulation motion that allow *nonprehensile* reconfiguration motion.

Optimality. Our first contribution is a formulation of pregrasp manipulation as trajectory optimization (Section III). Through a series of reductions, we demonstrate how the coupled reconfiguration and transport problem can be reduced to a single constrained trajectory optimization problem.

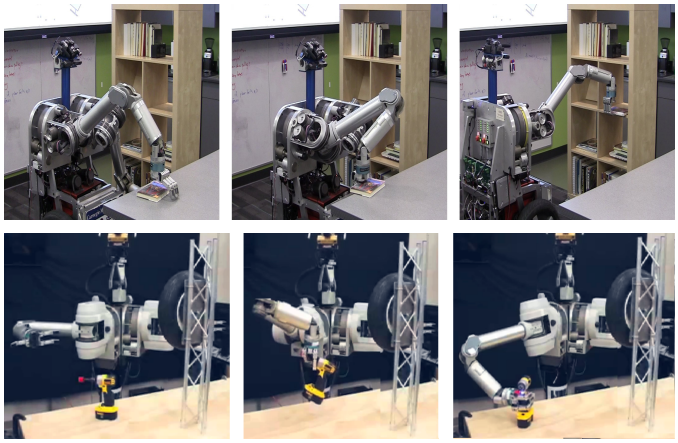

Fig. 1. The robot slides a book to the edge of a table, grasps the book spine, and places it into a bookshelf (top). The robot reconfigures a heavy drill to pick it up from a table (bottom).

Our second contribution is to solve this constrained trajectory optimization problem by extending functional gradient optimization techniques [8, 26] to address constraints and costs on starting configurations (Section IV).

Nonprehensile actions. Our third contribution is a formulation of reconfiguration with nonprehensile quasi-static pushing actions (Section V). This allows us to derive analytical bounds on the types of actions we can perform on the object. We exploit one such interesting bound: that the motion of a pushed or pulled object like a book by the fingertips can be modeled as a bounded-curvature Dubins car. This allows us to use extremely efficient lattice-style planners [18] used on ground vehicles for pushing.

We demonstrate an implementation of our algorithm on two bimanual manipulation platforms for the two different pregrasp manipulation tasks mentioned initially: pushing a book to the edge of a table, grasping it, and placing it to a bookshelf; and reconfiguring a heavy drill to pick it up from a table. Our experiments (Section VI) show that our algorithm has a significantly greater success rate and lower cost compared to a planner that only optimizes transport cost. This does come at the slight computational cost of solving the joint problem.

Our work takes a step towards incorporating physics-based reconfiguration actions into trajectory optimiza-

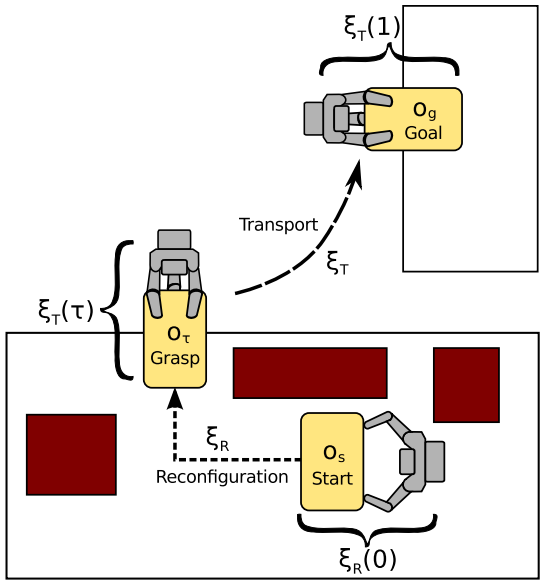

Fig. 2. An example manipulation problem involving a reconfiguration trajectory, a grasp, and a transport trajectory.

tion techniques. We are encouraged by our results.

II. Related Work

This work builds on trajectory-optimization approaches for arm motion planning, such as those involving elastic bands [25], or covariant Hamiltonian optimization [26]. General frameworks for trajectory optimization with contact constraints also exist [10, 24]. Our method takes advantage of the structure of pregrasp manipulation by identifying the points of regrasping and reducing the dimensionality of the search space. We use these points of regrasping as constraints during optimization [8].

Sampling-based approaches to arm planning, based on roadmaps [15], or trees [16] exist as well, for both free and constrained [3] spaces. Simeon [27] uses roadmaps to study the problem of manipulating objects constrained such that they require multiple pick and place operations to move to desired goal configurations. The proposed method produces long chains of pick and place operations, but does not consider optimality.

Several studies explore grasp metrics which consider both the grasp itself [5, 11] and the resulting transport trajectory [4, 14]. Such works attempt to solve the same class of problems as this work.

The general problem of nonprehensile manipulation has also been explored at length, whether by pushing in the plane [7, 19], tipping [22], pivoting [1], or caging [6]. These works focus on producing a desired change in the object of interest without considering potential subsequent prehensile manipulations.

III. Pregrasp Manipulation as Trajectory Optimization

Consider the problem presented in Fig.2, where a manipulator must move an object which cannot be grasped in its initial configuration. Instead, the manipulator performs some *reconfiguration* to get the object to a graspable location, grasps it, then *transports* the object to its final location. In this work we consider this general class of problem, in which manipulation is decomposed into *reconfiguration* and *transport* trajectories. Starting from a general formulation of trajectory optimization with pregrasp manipulation, we apply a series of reductions to find a form that is easily accessible to functional gradient optimization methods.

A. Functional gradient optimization

We define state $x = (q, o) \in \mathcal{X} = \mathcal{C} \times SE(3)$ as the configuration of the robot $q \in \mathcal{C}$ and the pose of the object $o \in SE(3)$. We control the motion of the robot, and the robot can move the object. It can do this by grasping, pulling, pushing, or loosely caging it. We define the mechanics of this manipulation in more detail in Section V.

Our objective is to get the object from some start o_s to some goal o_g. We seek to find the best trajectory $\xi : [0,1] \to \mathcal{X}$ that achieves this. To do this, we define a cost functional (a function of a function) $U : \Xi \to \mathbb{R}^+$ to minimize, where Ξ is a Hilbert space of trajectories:

$$\xi^* = \underset{\xi}{\operatorname{argmin}}\, U[\xi] \quad \textbf{s.t. } \xi(0) = (\cdot, o_s);\ \xi(1) = (\cdot, o_g) \tag{1}$$

Note that here, and henceforth, we use $\cdot$ to mean any feasible $q \in \mathcal{C}$, e.g. an element from the set of all inverse kinematic solutions of the arm that grasp o. Also, while our examples consider reconfiguration actions before the transport, the approach is equally applicable to such actions after transport.

B. Pregrasp manipulation

We consider the class of pregrasp manipulation problems where the object is manipulated *once* before it is grasped. Thus, motion decomposes into two phases: *reconfiguration* (R), and *transport* (T):

$$\xi = \xi_R[0, \tau] \oplus \xi_T[\tau, 1] \tag{2}$$

This allows us to rewrite (1) as:

$$\begin{aligned} \xi^* &= \underset{\xi}{\operatorname{argmin}}\, U[\xi_R \oplus \xi_T] = \underset{\xi_R, \xi_T}{\operatorname{argmin}} (U[\xi_R] + U[\xi_T]) \\ &\textbf{s.t. } \xi_R(0) = (\cdot, o_s);\ \xi_T(1) = (\cdot, o_g);\ \xi_R(\tau) = \xi_T(\tau) \end{aligned} \tag{3}$$

Unfortunately, the two optimizations are tightly coupled at $\xi(\tau)$, the state at which the two trajectories meet.

Our second reduction loosens this coupling by allowing the robot to release and grasp the object between the two phases for free. As a consequence, the only coupling between the two phases is the pose of the object, i.e.

$\xi_R(\tau) = (\cdot, o_\tau)$ and $\xi_T(\tau) = (\cdot, o_\tau)$, where o_τ is the pose of the object when it is grasped.

This restricts o_τ to the much smaller set $\mathbb{S} \subset SE(3)$ which are (1) a stable resting configuration for the object, (2) reachable by reconfiguration, and (3) accessible to grasping for transport.

This allows us to tease apart (3). Our next reduction is driven by computational efficiency. We restrict reconfiguration cost to functions only of object pose: $U[\xi_R] \triangleq U[o_R]$, where the trajectory $\xi_R = (q_R, o_R)$. This reduction enables us to compute reconfiguration cost extremely quickly. We then use standard graph-search techniques (explained in detail in Section V) to find the minimum reconfiguration cost of bringing the object to a particular o_τ:

$$u_R^*(o_\tau) = \min_{\xi_R} U[\xi_R] \quad \text{s.t.} \begin{cases} \xi_R(0) = (\cdot, o_s) \\ \xi_R(\tau) = (\cdot, o_\tau) \end{cases} \tag{4}$$

This allows us to rewrite (3) in its final form as an optimization of the transport trajectory which also takes into account the reconfiguration cost:

$$\xi_T^* = \arg\min_{\xi_T} \left(U[\xi_T] + u_R^*(o_\tau)\right) \quad \text{s.t. } \xi_T(\tau) = (\cdot, o_\tau \in \mathbb{S}) \tag{5}$$

We have finally reduced our problem to a solitary trajectory optimization problem with a constraint on the starting configuration of the trajectory. In the following section, we describe an extension of CHOMP, a functional gradient optimization technique used often in arm motion planning [26], to solve this problem.

IV. CHOMP with Start Costs

A. CHOMP

CHOMP [26] solves the unconstrained functional optimization problem (1). Given a norm A in the Hilbert space Ξ, CHOMP minimizes the regularized Taylor series approximation of U about the current trajectory ξ_i:

$$\xi_{i+1} = \operatorname{argmin}_{\xi} \{U[\xi_i] + \bar{\nabla} U^T (\xi - \xi_i) + \frac{\lambda}{2} ||\xi - \xi_i||_A^2\} \tag{6}$$

This can be minimized exactly to give the update rule:

$$\xi_{i+1} = \xi_i - \frac{1}{\lambda} A^{-1} \bar{\nabla} U \tag{7}$$

with the functional gradient operator:

$$\bar{\nabla} = \frac{\partial}{\partial \xi} - \frac{d}{dt} \frac{\partial}{\partial \xi'} \tag{8}$$

CHOMP models the cost function, $U[\xi]$, as a tradeoff between a "prior" smoothness cost, f_{prior}, and an obstacle cost, f_{obs}, bending the trajectory away from obstacles:

$$U[\xi] = f_{prior}[\xi] + f_{obs}[\xi] \tag{9}$$

A variant of CHOMP that addresses trajectory-wide constraints, called GSCHOMP [8] solves the problem:

$$\xi^* = \operatorname{argmin}_{\xi} U[\xi] \;\; \textbf{s.t. } h_\xi = 0 \tag{10}$$

By additionally linearizing the constraint about the current trajectory ξ_i as $h(\xi) = C(\xi - \xi_i) + b$, GSCHOMP obtains a simple Newton's method-style update:

$$\xi_{i+1} = \xi_i - \frac{1}{\lambda} A^{-1} \bar{\nabla} U \tag{11a}$$

$$+ \frac{1}{\lambda} A^{-1} C^T (C A^{-1} C^T)^{-1} C A^{-1} \bar{\nabla} U \tag{11b}$$

$$- A^{-1} C^T (C A^{-1} C^T) b \tag{11c}$$

where (11a) is the unconstrained update (7), (11b) is the projection onto the zero set $C(\xi - \xi_i) = 0$ and (11c) is the offset correction to further project onto $C(\xi - \xi_i) + b = 0$.

B. Start Costs

Our final reduction (5) posed the reconfiguration problem as optimizing a functional with a start constraint, which GSCHOMP addresses, but also a start cost, which we address now.

A key issue with start costs $u_R^*(o_\tau)$ is that they are only defined on the configuration space submanifold where the object is in $\mathbb{S}$. As a result, there is no gradient information available when the optimizer is not on the manifold, which happens very often due to linearization and numerical precision.

We address this by *lifting* the function and its gradient into $\mathcal{C}$ by projecting any configuration onto the submanifold and using that cost $u_R^*(q) = u_R^*(\text{proj}(q))$. Furthermore, we use the workspace projection for efficiency.

Once lifted, we can define the start cost functional as:

$$U_R^*[\xi] = u_R^*(\xi(\tau)) \tag{12}$$

This cost functional is then added to $U[\xi]$ (9).

A 'good' projection would ideally lift the cost onto the tangent space of the linearized constraint, and thereby result in no component along (11b). However, the computational cost of finding this projection overwhelms the simplicity of just eliminating the component with (11b).

C. Representing the Constraint

While the above formulation is general and will work for a wide variety of pregrasp manipulation actions, in this study we focus on the class of problems where reconfiguration is accomplished via nonprehensile actions such as pushing and rotating the object on a table. Therefore, we have the constraint, h_{ξ_T}, that the transport trajectory starts by grasping the object off the plane described by the table.

At the starting point, $\xi_T(\tau) = (q, o_\tau)$, the robot configuration and the object pose is related by:

$$o_\tau = \mathbf{FK}(q)\, T_g \tag{13}$$

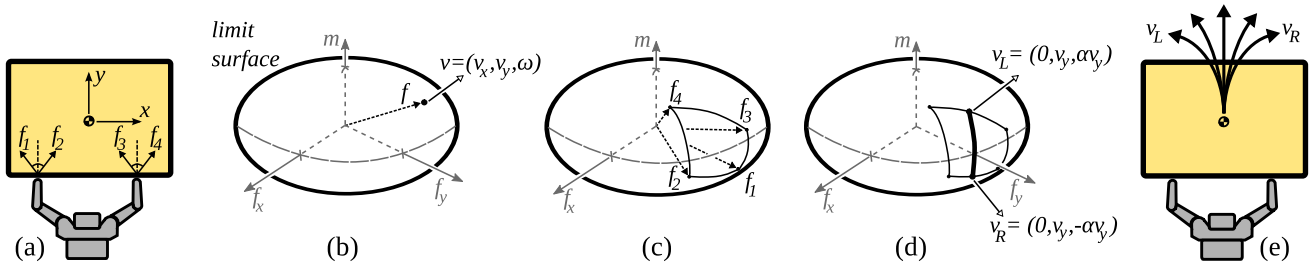

Fig. 3. Pushing model. (a) Friction cones at the contacts. (b) The limit surface. (c) The cone of all possible forces that can be applied by the hand. (d) A subset of the forces which create linear velocity along a single direction and a bounded range of angular velocities. (e) The Dubins car model.

where T_g is the grasp transform representing the relative pose of the object in the robot end-effector, and **FK** is the forward-kinematics of the robot giving the end-effector pose in the world.

We describe the table using the xy-plane of the coordinate frame T_t. Then, using (13), we can represent the constraint that the object must be on this plane:

$$h(\xi_T) = S\,T_t^{-1}\,\mathbf{FK}(q)T_g$$

where S is a selector matrix

$$S = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{14}$$

which selects the z-axis, the rotation around x, and the rotation around y. The result is a 3-dimensional vector which is **0** on the constraint.

We can find C through straightforward differentiation. Since S, T_t and T_g are constant, we have:

$$C = S\,T_t^{-1}\,J(q)\,T_g \tag{15}$$

where $J(q)$ is the Jacobian of the end effector motion evaluated at q.

V. The Mechanics of Pregrasp Manipulation

We build our reconfiguration planner based on the analysis of the quasi-static interactions between the robot hand and a pushed object. We illustrate the robot hand and an object in Fig.3-(a). Using the coefficient of friction between the hand and the object, μ, we can draw *friction cone* [23] edges at an angle $\tan^{-1}(\mu)$ with the contact normal at the contact points. Coulomb's law dictates that the possible pushing forces that the hand can apply are limited by the friction cones.

The *limit surface* [12, 13] relates the generalized forces applied on an object to the resulting generalized velocity. We assume that the object applies finite pressure to the underlying surface and the friction between the two is also finite. Then the limit surface is smooth and strictly convex, similar in shape to a three-dimensional ellipsoid (Fig.3-(b)) where the dimensions are f_x, f_y, and m, the force along x-direction, the force along y-direction, and the moment respectively. During quasi-static pushing the generalized forces applied to the object are exactly on the limit surface. Given such a generalized force we can find the object's generalized velocity using the normal to the limit surface at that point, and replacing the dimensions to be v_x, v_y, and ω, the linear velocity along x, the linear velocity along y, and the angular velocity, respectively.

In Fig.3-(c) we illustrate the three-dimensional cone which corresponds to all the combined forces that can be applied by the forces in the friction cones. The corresponding cone of generalized velocities gives the possible velocities the object can move with [9]. Several studies use this analysis to study the controllability and planning of pushing [2, 20, 21]. In this study we use a subset of these velocities, shown in Fig.3-(d), which correspond to a single linear velocity and a bounded range of angular velocities. This simplifies our pushing system to a Dubins car [17] for planning purposes (Fig.3-(e)). This model allows us to take advantage of a wide range of research done in motion planning for car-like vehicles. In particular, we choose to perform a graph search over a lattice state space [18].

Our action space includes the pushing actions and additionally *switching* actions where the robot changes the side of the object it is pushing on. In this sense, we model each object as a Dubins car that can move along the four primary directions. During planning, we check each action for collision between the pushed object and obstacles. We also check for collision between the end-effector and obstacles.

We define the cost of reconfiguration, $u_R^*(o_\tau)$, as the distance the object must travel from its starting pose, o_s, to the grasp pose, o_τ. Using such a planner in the low-dimensional state space enables us to generate the reconfiguration cost for an intermediate pose, o_τ, very quickly. This formulation associates zero cost with *switching* pushing sides, as the object does not move during this action.

While this planner is very fast and physically realistic, it produces only the trajectory of the end-effector and the object. We find the corresponding arm trajectory separately. This step can fail due to the arm's kinematic infeasibility, in which case we re-run our planner removing the offending action edges from our search graph.

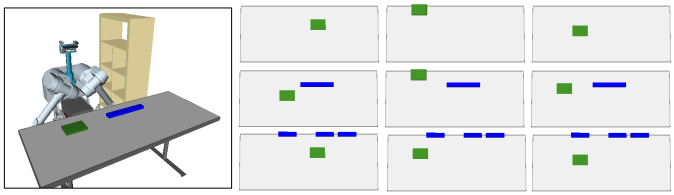

Fig. 4. The 9 scenes used to test the algorithm. In each scene, the green object represents the book to be reconfigured. The blue objects represent obstacles.

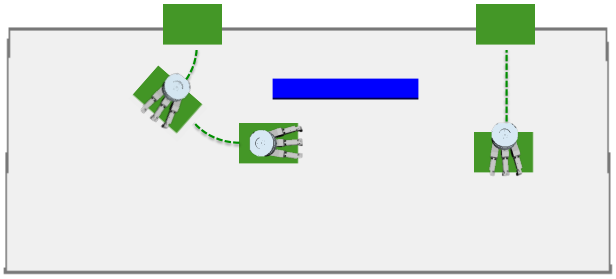

Fig. 5. Valid push paths for two initial configurations of the book. In the left configuration, the book is first pushed around the obstacle. The hand must then change sides of the book to continue the push to the edge of the table. In the right configuration, the book is pushed directly to the edge of the table.

VI. Experiments

A. Book Manipulation

We compare our algorithm Reconfiguration CHOMP with two alternatives, each of which optimizes the two costs *sequentially* instead of *jointly*: (A) Pre-Grasp CHOMP, which first computes the object pose that minimizes reconfiguration cost, and then fixes that pose and optimizes the transport cost via CHOMP; (B) Start-Set CHOMP, which optimizes transport cost by setting reconfiguration cost, $u_R^*(o_\tau)$ to zero in (5) but is seeded at the optimum of the reconfiguration cost.

We validate two key hypotheses:

(H1) Planning time vs. solution cost: Reconfiguration CHOMP significantly lowers path cost and increases success rate, but at a significant increase in planning time. However, the positive effect on path cost and success rate persists even when other planners are given this extra time to improve their solutions.

(H2) Reconfiguration vs. transport cost tradeoff: Reconfiguration CHOMP behavior is sensitive to the relative weighting between reconfiguration and transport costs.

Problem setup. We first consider a book manipulation scenario, where we task our robot to place a book lying in the middle of a table into a bookshelf (Fig.4). The book must first be reconfigured to the table edge, where it can then be stably grasped and transported to the bookshelf. The reconfiguration path of the book must avoid contact with any obstacles on the table.

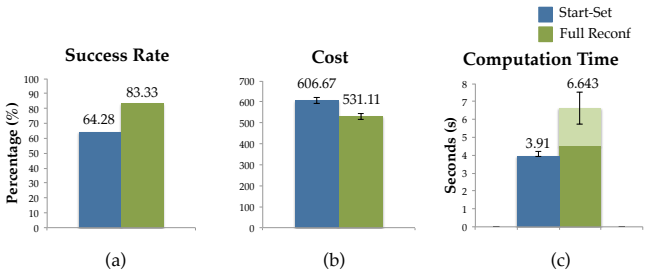

Fig. 6. A comparison between success rate, cost and computation time between Start-Set CHOMP and Full Reconfiguration CHOMP. In (c), the light green shows the extra cost required to generate the full reconfiguration cost function.

Fig.5 shows examples of reconfiguration plans automatically computed by our planner, with object paths as well as pushing directions. Our constraint surface is defined by reachable poses of the book which lie on edges of the table. To construct the lattice graph, we discretize the table's surface into 1 cm by 1 cm axis-aligned grid cells. We then use numerical differentiation to compute the reconfiguration cost gradient. During CHOMP iterations, we use linear interpolation to compute the continuous gradient.

We construct 9 test scenes, shown in Fig.4, which vary in number of obstacles and start location of the book. For each test scene, we define 14 initialization trajectories, resulting in 126 test cases for each of the 3 algorithms. The seed trajectory is a straight line trajectory in configuration space with start point uniformly sampled from our constraint surface. All 14 seed trajectories share a common goal.

Dependent measures. We compare end-to-end planning time, solution cost, and success rate. We define success as a feasible collision-free trajectory.

1) (H1) Planning time vs. solution cost: Our results are shown in Fig.6. In every case, we allow all algorithms to run multiple times (with random restarts) until the slowest algorithm has completed.

Our planner shows a 29% improvement in success rate. The most common cause of failure is selection of an intermediate pose that is infeasible to achieve because it is occupied by an obstacle.

We compare total path cost across all tests where both algorithms were successful. Our algorithm demonstrates a 12% improvement over Start-Set CHOMP. A t-test shows this difference is in fact statistically significant ($t(50) = 16.58, p < 0.001$). We note that the size of the cost improvement is sensitive to the scale of the reconfiguration cost relative to the transport cost. This will be further discussed in Section VI-A2.

Next, we consider single-run performance time. The increased difference between planning time is statistically significant ($t(50) = 36.03, p < 0.001$). Much of this increase can be attributed to calculating the reconfiguration cost function.

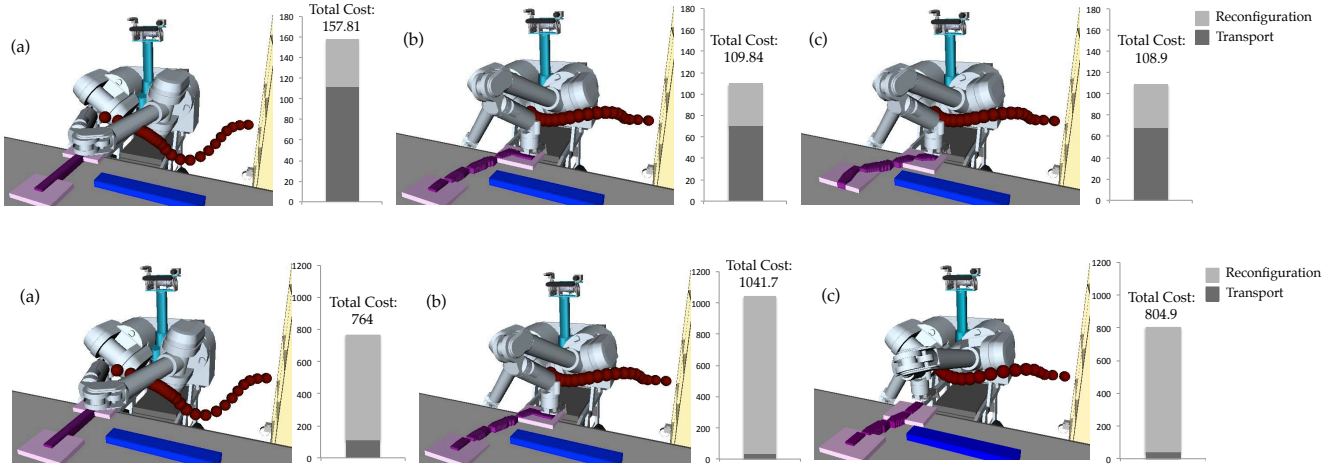

Fig. 7. Comparison of reconfiguration and transport trajectories for a single scene between three algorithms: (a) Pre-Grasp CHOMP (b) Start-Set CHOMP (c) Reconfiguration CHOMP. The top images show the results when transport cost is weighted high. The bottom images show results when reconfiguration cost is weighted high.

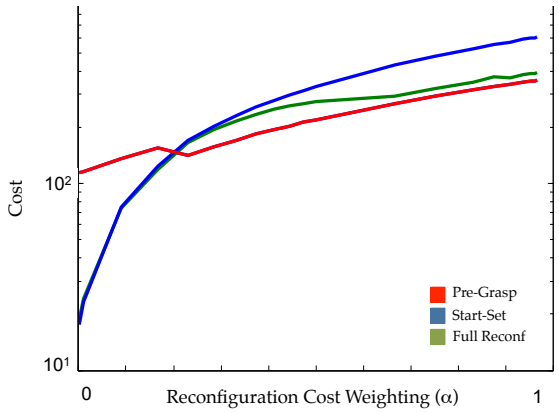

Fig. 8. Comparative performance of algorithms as cost is weighted between pure reconfiguration ($\alpha = 1$) and pure transport cost ($\alpha = 0$).

2) (H2) Reconfiguration vs. transport cost tradeoff: To test this hypothesis we use a subset of a our 126 test cases. In particular, we examine only the test cases where the seed trajectory starts at a configuration corresponding to the optimal reconfiguration pose. Using this subset allows us to create a reconfiguration-informed Start-Set CHOMP algorithm. It now starts from a point of optimal reconfiguration, but still optimizes only over transport cost.

Fig.7 shows the selected intermediate location and associated object path and transport trajectory for each of the three algorithms for a single scenario. The top three images show a scenario where transport cost is large relative to reconfiguration cost. The bottom row of images show a scenario with reconfiguration cost increased. As can be seen, Reconfiguration CHOMP selects a shorter reconfiguration trajectory when reconfiguration cost has large weight. Conversly, the trajectories selected by the other two algorithms are unaffected.

We further explore the performance of the three algorithms as we trade importance between reconfiguration and transport cost in Fig.8. Here we show the change in total path cost as the relative weight of the reconfiguration cost is increased. When relative weight is very small, our algorithm reduces to Start-Set CHOMP, optimizing only transport cost. As reconfiguration cost becomes more important, our algorithm tends toward performance of Pre-Grasp CHOMP, optimizing only reconfiguration cost.

The best performing algorithm varies with α. Predicting which algorithm will perform best for a given α is difficult, as this varies by task. One obvious alternative to Reconfiguration CHOMP is to run Pre-Grasp CHOMP and Start-Set CHOMP for a given α and use the minimum cost trajectory. We note that while in some cases this technique will provide a lower cost solution, it comes with the time penalty incurred by running two planners.

B. Lifting a Drill

Problem Setup We next examine a scenario where the robot is tasked with using a small impact drill (Fig.1-bottom) to remove lugnuts from a wheel hub. In order to use the drill, the robot must first lift it off the table. Often the pose and weight of the drill make it impossible to find a valid grasp due to joint torque limits. Instead, the robot must first *reconfigure* the drill by sliding it to a pose where a feasible lift trajectory can be found.

In these experiments, our constraint surface is defined by reachable poses of the drill on the table. We assume the drill can slide and rotate in the plane of the table. Like with the book task described previously, we wish to define a reconfiguration cost function that penalizes

a trajectory based on the length of the reconfiguration. In addition, we wish to penalize poses which requires large joint torques when lifting the drill. Thus we define a two part reconfiguration cost function:

$$u_R^*(o_\tau) = \beta_f u_f(o_\tau) + \beta_d u_d(o_\tau) \tag{16}$$

where $u_f(o_\tau)$ represents the joint torque penalty, $u_d(o_\tau)$ represents the cost based on the length of the reconfiguration trajectory and $\beta_f, \beta_d \in \mathbb{R}$ are weighting constants.

We define $u_d(o_\tau)$ as the translational distance between o_τ (the pose of the object at the start of the lift) and the initial pose, o_s. The gradient of this cost function is straightforwardly computed as the vector from o_τ to o_s passed through the pseudo-inverse of the kinematic Jacobian of the robot.

For the second part of our reconfiguration cost function, we consider joint torques. If we assume that the drill, when grasped by the end-effector, produces a force F due to gravity, the joint torque can be calculated by $J(q)^T F$, where $J(q)$ is the kinematic translational Jacobian at configuration q. Using this information, we can define a cost function which represents the sum squared joint torques:

$$u_f(o_\tau) = F^T J(q_\tau)^T J(q_\tau) F \tag{17}$$

where q_τ is a valid configuration of the manipulator when grasping the drill at pose o_τ. Here poses which require high torque for the drill to be lifted are penalized with higher reconfiguration cost. The gradient of this function contains the kinematic Hessian, and is difficult to compute directly. Instead, we use finite differencing to approximate the gradient.

For each experiment, we randomly select the initial pose for the drill from the set of reachable poses. The robot is given a goal of lifting the object and then extending its arm outward. Other than the table, no other obstacles exist in the environment. Fig.9 shows an example solution. As can be seen, the optimal reconfiguration pose, o_τ is closer to the robot than the initial pose, o_s.

Also shown in the figure is the effect of varying the parameters β_f (the weight on the torque cost function), and β_d (the weight on the reconfiguration cost function) given some initial starting pose (green). The resulting steps of optimization (orange) push the object toward a local minimum of the feasibility cost function given the weights on each component. As β_f increases relative to β_d, the drill is pushed closer to the robot so that the joints use the minimal energy to lift the drill. In contrast, as β_d increases, the drill's configuration is pushed closer and closer to its initial configuration. At very high values of β_f, the resulting reconfiguration motion is a pure rotation.

In each of these experiments, the performance of CHOMP was not very adversely limited by incorporating constraints and additional loss functions. Average per-iteration time increased from 30 ms to 38 ms from single-goal CHOMP [26], and total planning time increased from 3.1 seconds to 3.9 seconds (for 100 iterations).

VII. Discussion and Future Work

Trajectory optimization is a powerful tool to generate intelligent behavior from simple costs. Pregrasp manipulation, however, requires planning for multiple modes of prehensile and nonprehensile interaction, e.g. pick-and-place, pushing, pulling, toppling. These modes often have very different system dynamics, meaning optimization over the complete manipulation trajectory requires a very high-dimensional state space and a complicated joint description of the system dynamics. Instead, we observe that the complete manipulation trajectory can be divided into steps at the regrasp points. This enables us to run specialized, lower-dimensional planners for each different mode of interaction, while still sharing the cost of each mode with neighboring modes at the regrasp points.

In this paper we present results on tasks which require only two modes of interactions: pushing and pick-and-place. However, our general formulation can easily be extended to work with other modes of interactions, e.g. toppling. Similarly, it can be extended for tasks which require more than two steps of interaction.

Planning multiple steps of open-loop manipulation actions makes them vulnerable to growing uncertainty during the execution of these actions. For example, after multiple pushing actions, an object might end up in a different spot than the original goal, breaking the connection to subsequent transport trajectories which have already been planned. An interesting extension would be to integrate a cost for uncertainty-inducing actions into the optimization process to increase the robustness of the resulting plans.

Our optimization process uses the gradient of the reconfiguration costs computed for possible object grasp poses at a high resolution. This computation creates an overhead in computation time. In future work we aim for a tighter integration between the transport trajectory optimization and the reconfiguration planning, such that reconfiguration planning can be dynamically invoked as needed by the optimizer.

Acknowledgments

This material is based upon work supported by NSF Grant No. 1208388, NSF-EEC-0540865, DARPA-BAA-10-28 and the Toyota Motor Corporation. We thank the members of the Personal Robotics Lab for very helpful discussion and advice.

References

[1] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *IEEE/RSJ IROS*, 1993.

[2] S. Akella and M. T. Mason. Posing polygonal objects in the plane by pushing. *International Journal of Robotics Research*, 17(1):70–88, January 1998.

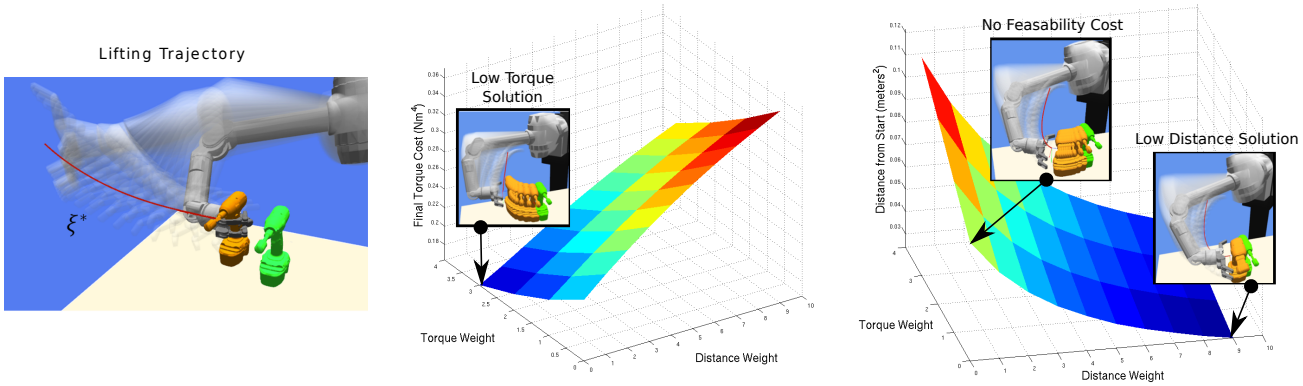

Fig. 9. Comparison of solutions as torque and distance cost weighting is varied.

[3] D. Berenson, S. Srinivasa, and J. Kuffner. Task Space Regions: A framework for pose-constrained manipulation planning. *IJRR*, 30(12):1435–1460, 2011.

[4] L. Chang, S. Srinivasa, and N. Pollard. Planning pre-grasp manipulation for transport tasks. In *IEEE ICRA*, 2010.

[5] L. Y. Chang and N. Pollard. Posture optimization for pre-grasp interaction planning. In *IEEE ICRA, Workshop on Manipulation Under Uncertainty*, 2011.

[6] R. Diankov, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning with caging grasps. In *IEEE-RAS Humanoids*, 2008.

[7] M. Dogar and S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 33(3):217–236, 2012.

[8] A. Dragan, N. Ratliff, and S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *IEEE ICRA*, 2011.

[9] M. Erdmann. On a representation of friction in configuration space. *International Journal of Robotics Research*, 13(3), 1994.

[10] T. Erez and E. Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *IEEE/RSJ IROS*, pages 4914–4919. IEEE, 2012.

[11] M. Gienger, M. Toussaint, and C. Goerick. Task maps in humanoid robot manipulation. In *IEEE/RSJ IROS*, 2008.

[12] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. Part 1. Limit surface and moment function. *Wear*, (143):307–330, 1991.

[13] R. D. Howe and M. R. Cutkosky. Practical Force-Motion Models for Sliding Manipulation. *IJRR*, 15(6):557–572, 1996.

[14] D. Kappler, L. Y. Chang, M. Przybylski, N. Pollard, T. Asfour, and R. Dillmann. Representation of pre-grasp strategies for object manipulation. In *IEEE-RAS Humanoids*, December 2010.

[15] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE ICRA*, 1996.

[16] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *IJRR*, 20(5):378–400, 2001.

[17] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.

[18] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *RSS*, 2008.

[19] K. Lynch and M. T. Mason. Controllability of pushing. In *IEEE ICRA*, 1995.

[20] K. M. Lynch. Locally controllable manipulation by stable pushing. 15(2):318 – 327, April 1999.

[21] K. M. Lynch and M. T. Mason. Stable Pushing: Mechanics, Controllability, and Planning. 15(6):533–556, 1996.

[22] Y. Maeda, H. Kijimoto, Y. Aiyama, and T. Arai. Planning of graspless manipulation by multiple robot fingers. In *IEEE ICRA*, 2001.

[23] M. T. Mason. Mechanics and Planning of Manipulator Pushing Operations. *IJRR*, 5(3):53–71, 1986.

[24] M. Posa and R. Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *WAFR*, pages 527–542. Springer, 2013.

[25] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *IEEE ICRA*, 1993.

[26] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE ICRA*, 2009.

[27] T. Simeon, J.-P. Laumond, J. Cortes, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *IJRR*, 23(7–8):729–746, 2004.