2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

Manipulation Planning with Goal Sets Using Constrained Trajectory Optimization

Anca D. Dragan, Nathan D. Ratliff, and Siddhartha S. Srinivasa

***Abstract*—Goal sets are omnipresent in manipulation: picking up objects, placing them on counters or in bins, handing them off — all of these tasks encompass continuous sets of goals. This paper describes how to design optimal trajectories that exploit goal sets. We extend CHOMP (Covariant Hamiltonian Optimization for Motion Planning), a recent trajectory optimizer that has proven effective on high-dimensional problems, to handle trajectory-wide constraints, and relate the solution to the intuition of taking unconstrained steps and subsequently projecting them onto the constraints. We then show how this projection simplifies for goal sets (i.e. constraints that affect only the end-point). Finally, we present experiments on a personal robotics platform that show the importance of exploiting goal sets in trajectory optimization for day-to-day manipulation tasks.**

I. Introduction

This paper is the result of two complementary goals. Our first goal is very practical: to enable personal robots ([1]–[3]) to perform useful manipulation tasks in close proximity to humans. Our second goal is very theoretical: to understand the underpinnings of constrained trajectory optimization, emphasizing the need for mathematically sound solutions, and seeking the right insight from these solutions.

While these goals might sound contrasting, the practice motivates the theory. Since we want humans to be comfortable in a robot's workspace, and to be able to predict its motion, we want to look beyond planners that produce feasible but random trajectories ([4], [5]). Natural, legible and predictable movements require optimization. The trajectory optimization problem has a very rich history, seeded in the work of Euler, Lagrange and Hamilton [6] with the principle of least action, and continuing with the variational calculus view of optimal control [7]. Zefran [8] was one of the first to use variational calculus to do iterative functional gradient optimization. Later, Quinlan [9] and Brock [10] extended this to what they called Elastic Band trajectory smoothing, which used the concept of collision bubbles to ensure feasibility. Building on this, we introduced Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [11], a functional gradient descent algorithm that uses a smoothness metric in trajectory space that propagates local changes to the entire trajectory, moving it out of collision faster while preserving smoothness.

This work partially supported by Intel Labs Pittsburgh and the National Science Foundation under Grant No. EEC-0540865. Special thanks to Chris Atkeson and members of the Personal Robotics project at Intel Labs Pittsburgh for insightful comments and discussions.

A. Dragan is with The Robotics Institute, Carnegie Mellon University. adragan@cs.cmu.edu. N. Ratliff and S. Srinivasa are with Intel Labs Pittsburgh. {nathan.ratliff, siddhartha.srinivasa}@intel.com.

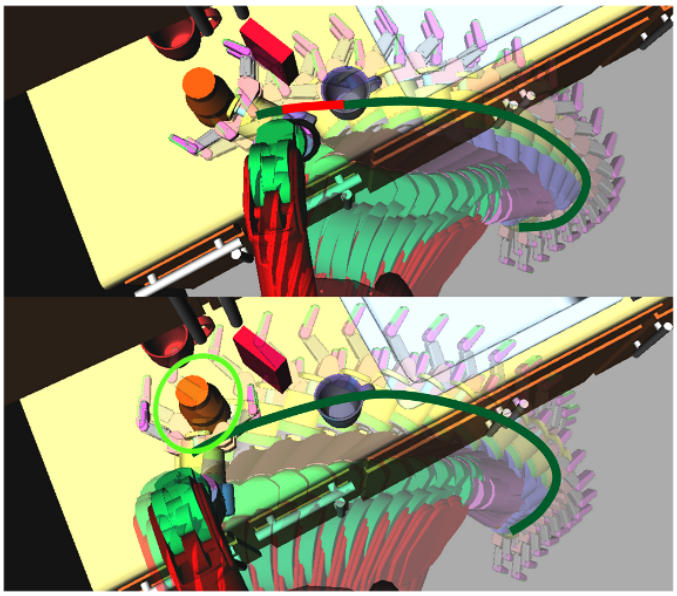

Fig. 1. Top: The trajectory found by an optimizer that uses a specified single goal. The optimizer cannot avoid collision with the red box. Bottom: A feasible trajectory found by an optimizer that can take advantage of an entire goal set.

CHOMP does well in practice: it converges rapidly for many high-dimensional motion planning problems. However, it makes a classic simplifying assumption: CHOMP can only plan to a single goal configuration. This single goal assumption constrains the capabilities of the planner for many robotic applications. For example, choose any object around, from a laptop to a salt shaker, and think of all the ways your hand could grasp it — there are usually entire regions of viable goals. Now for each grasp, think of the different positions your elbow could take: upwards to avoid the table, downwards to avoid a wall. With a redundant manipulator, like the human arm, there are multiple configurations for each grasp. This multiplicity of goals forms what we refer to as a *goal set*. A planner that exploits goal sets has greater chances of succeeding compared to a single goal planner (Fig.1). Even when both planners succeed, shifting to an easier goal often increases the quality of the trajectory. Overall, planners can produce better solutions if they take advantage of goal sets.

So how can trajectory optimizers like CHOMP exploit goal sets? One simple approach would be a penalty method [12]: add a term to the objective function that penalizes deviations from the goal set. Unfortunately, we have found that such a term tends to fight with the objective, particularly the obstacle

978-1-61284-385-8/11/$26.00 ©2011 IEEE

cost, making it difficult to tune for fast (or even sufficient) convergence. Instead, we propose a constrained optimization solution. We put the problem in context by solving it for constraints that affect the entire trajectory, and then show how the solution simplifies when these constraints are restricted to the last configuration (goal set constraints). At each iteration, CHOMP performs functional gradient descent by linearizing the objective and locally minimizing it within a trust region. In order to extend CHOMP to trajectory constraints, we simply subject this linearized objective to the constraints. We then relate this to the very natural problem of projecting the unconstrained gradient onto the constraints. Doing so helps us identify the correct metric for this projection. It also plays a key role in interpreting the concrete solution, which we derive by linearizing the constraints: we show that this constrained update rule implements a two step projection with respect to CHOMP's smoothness metric. For the case of goal constraints, this update rule simplifies to a very intuitive algorithm: project the unconstrained end-configuration in Euclidean space, and propagate the change to the rest of the trajectory.

In our experiments with HERB [1], our personal robotics platform, we empirically demonstrate the improvement trajectory optimization with goal sets achieves over the original CHOMP algorithm. We use day-to-day manipulation tasks, like picking up objects, handing them off or recycling them. The results not only show the importance of goal sets in trajectory optimization, but also reinforce the need for theoretically sound solutions.

II. Functional Gradient Trajectory Optimization: CHOMP

This section outlines CHOMP (Covariant Hamiltonian Optimization for Motion Planning): a functional trajectory optimization algorithm that minimizes a collision cost while maintaining certain dynamical properties, usually pertaining to the notion of trajectory smoothness. While our goal is to make this recap self-contained, additional details are available in Ratliff [11], [13].

A. The Objective Functional

CHOMP models the trajectory optimization objective as a trade-off between a "prior" and an obstacle cost:

$$\mathcal{U}[\xi] = \lambda f_{prior}[\xi] + f_{obs}[\xi], \tag{1}$$

Both terms are *functionals*: functions of a function $\xi : [0,1] \to \mathcal{Q}$, mapping time to robot configurations $q \in \mathcal{Q} \subset \mathbb{R}^d$. The first one, f_{prior}, typically measures dynamical quantities across the trajectory, such as velocities and accelerations, and relates to the notion of smoothness of the trajectory. In this work, we choose the integral over squared velocity norms:

$$f_{prior}[\xi] = \frac{1}{2}\int_0^1 \|\xi'(t)\|^2 dt \tag{2}$$

While the first term keeps the trajectory smooth, the second term, f_{obs}, bends the trajectory away from obstacles by penalizing parts of the robot that are close to or already in collision. Let $\mathcal{B} \subset \mathbb{R}^3$ be the set of body points on the robot and $x : \mathcal{Q} \times \mathcal{B} \to \mathbb{R}^3$ the forward kinematics mapping from configuration space to workspace at a particular body point. Furthermore, let $c : \mathbb{R}^3 \to \mathbb{R}$ be a workspace cost function that penalizes the points inside and around the obstacles. Since we want to drive the body points away from collision, we can write the obstacle cost as an integral that collects the cost encountered by each workspace body point on the robot as it sweeps along the trajectory:

$$f_{obs}[\xi] = \int_0^1 \int_{\mathcal{B}} c(\ x(\xi(t),u)\ ) \left\| \frac{d}{dt} x(\xi(t),u) \right\| du dt \tag{3}$$

In the next section, we derive the iterative update that minimizes the linear approximation of the objective within a trust region. To do so, we need the gradient of this functional:

$$\bar{\nabla} f_{obs}[\xi] = \int_{\mathcal{B}} J^T \|x'\| \left[(I - \hat{x}'\hat{x}'^T)\nabla c - c\kappa \right] du, \tag{4}$$

where $\hat{x}'$ is the normalized velocity vector, $\kappa = \|x'\|^{-2}(I - \hat{x}'\hat{x}'^T)x''$ is the curvature vector along the workspace trajectory traced by a particular body point, and J is the kinematic Jacobian at that body point. We also use the functional gradient of the prior term in (2): $\bar{\nabla} f_{prior}[\xi](t) = -\frac{d^2}{dt^2}\xi(t)$. In order to simplify the notation, we suppress dependencies on time t and body point u in these expressions.

B. The Iterative Update Rule

Although so far we remained unencumbered by a particular parametrization of the trajectory ξ, performing gradient descent with the objective in (1) on a real robot mandates a parametrization choice. In this work, we will use a straightforward discretization of the trajectory function over time: $\xi \approx (q_1^T, q_2^T, \ldots, q_n^T)^T \in \mathbb{R}^{n \times d}$. Under this parametrization, we can write the prior term as:

$$f_{prior} = \frac{1}{2}||K\xi + e||^2 = \frac{1}{2}\xi^T A \xi + \xi^T b + c \tag{5}$$

Here, K is a finite differencing matrix and e is a vector that accounts for the contributions of the configurations that remain constant in the trajectory – originally both the start *and* the goal configurations. If we are to extend CHOMP to goal sets, a first requirement is for the end-configuration to be able to move, which demands a slight change in K and e: they must treat q_n as a variable. Fig.2 shows the impact this change has on the inverse of the dynamics matrix, $A^{-1} = (K^T K)^{-1}$, which plays an important role in the gradient updates: as the remainder of this section will reveal, A^{-1} acts as a "smoothing" operator that propagates the Euclidean gradient of the objective (1) along the trajectory in the way depicted by Fig.2. Since the last configuration is no longer constant, the update rule achieves a smoother trajectory by propagating the change without any damping.

Given this parametrization, CHOMP is simply a variant on gradient descent with a metric that naturally fits the problem. At each iteration, it is minimizing the linear approximation of

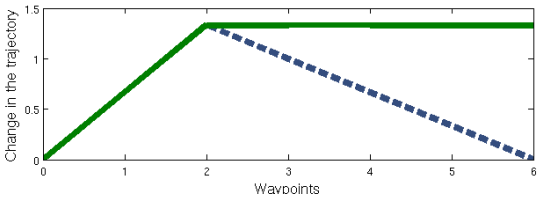

Fig. 2. The ith row of A^{-1} consists of two linear segments joining at entry i. Superimposed on this image (dotted line) is the corresponding second linear segment of the original (unique goal) CHOMP algorithm, which descends to zero as a result of its fixed end-configuration.

$\mathcal{U}$ about ξ_t within an ellipsoid trust region ([12]). The distance metric that shapes this ellipsoid is defined by A:

$$\xi_{t+1} = \min_{\xi \in \Xi} \mathcal{U}(\xi_t) + g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}\|\xi - \xi_t\|_A^2, \quad (6)$$

where $\|\xi\|_A^2 = \xi^T A \xi$, g_t is the discretized functional gradient, and $\{\eta_t\}_{t=1}^T$ is a problem specific sequence of weights, correlated to the step sizes. Since this objective is quadratic in ξ, we can solve for the update by setting the gradient to zero:

$$\xi_{t+1} = \xi_t - \frac{1}{\eta_t} A^{-1} g_t \quad (7)$$

We changed the A matrix so that this update rule can alter the trajectory end-point, as in (Fig.2). Providing this ability is necessary for incorporating goal sets, since the goal is no longer constant. On the other hand, there is nothing yet to constrain the end-point. For example, in the absence of any obstacles, a trajectory from start to a feasible goal will eventually become a point trajectory at the start, since the stand-still trajectory is the global minima of the prior cost. Therefore, the next section will focus on constraining this update rule to ensure reaching a feasible goal.

III. Incorporating General Trajectory Constraints

In this section, we derive the update rule for the constrained version of CHOMP, which minimizes $\mathcal{U}$ subject to generic trajectory constraints. These could simply constrain the goal configuration alone, but they could also affect the whole trajectory (e.g. "move the objects without tilting it"). We assume that we can describe a constraint on the space of all trajectories in the form of a nonlinear differentiable vector valued function $h : \Xi \to \mathbb{R}^k$ that is reasonably approximated linearly, and for which $h(\xi) = 0$ when the trajectory ξ satisfies the required constraints.

In order to constrain the updates, we need to optimize the regularized linear approximation from (6), subject to the nonlinear constraints $h(\xi) = 0$:

$$\xi_{t+1} = \arg\min_{\xi \in \Xi} \mathcal{U}(\xi_t) + g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}\|\xi - \xi_t\|_A^2 \quad (8)$$
$$s.t. \;\; h(\xi) = 0$$

This is in fact equivalent to the very natural problem of taking the unconstrained solution in (7) and projecting it onto the constraints, where this projection happens with respect to the smoothness metric given by A. To see this, we can simply rewrite the objective: $\min \mathcal{U}(\xi_t) + g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}\|\xi - \xi_t\|_A^2 \Leftrightarrow \min g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}(\xi - \xi_t)^T A(\xi - \xi_t) \Leftrightarrow \min(\xi_t - \frac{1}{\eta_t}A^{-1}g_t - \xi)^T A(\xi_t - \frac{1}{\eta_t}A^{-1}g_t - \xi)$. The problem can thus be written as:

$$\xi_{t+1} = \arg\min_{\xi \in \Xi} \|\xi \overbrace{- \frac{1}{\eta_t}A^{-1}g_t}^{\textbf{unconstr. (7)}} - \xi\|_A^2 \quad (9)$$
$$s.t. \;\; h(\xi) = 0$$

This interpretation will become very relevant in the next section, which uncovers the insight behind the new update rule we obtain by solving (8).

To derive a concrete update rule for (8), we linearize the h around ξ_t: $h(\xi) \approx h(\xi_t) + \frac{\partial}{\partial \xi}h(\xi_t)(\xi - \xi_t) = C(\xi - \xi_t) + b$ where $C = \frac{\partial}{\partial \xi}h(\xi_t)$ is the Jacobian of the constraint function evaluated at ξ_t and $b = h(\xi_t)$. The Lagrangian of the constrained gradient optimization problem in (8), now with linearized constraints, is $\mathcal{L}_g(\xi, \lambda) = \mathcal{U}(\xi_t) + g_t^T(\xi - \xi_t) + \frac{\eta_t}{2}\|\xi - \xi_t\|_A^2 + \lambda^T(C(\xi - \xi_t) + b)$, and the corresponding first-order optimality conditions are:

$$\begin{cases} \nabla_\xi \mathcal{L}_g = g_t + \eta_t A(\xi - \xi_t) + C^T\lambda = 0 \\ \nabla_\lambda \mathcal{L}_g = C(\xi - \xi_t) + b = 0 \end{cases} \quad (10)$$

Since the linearization is convex, the first order conditions completely describe the solution. We can therefore derive a new update rule in closed form. If we denote $\frac{\lambda}{\eta_t} = \gamma$, from the first equation we get $\xi = \xi_t - \frac{1}{\eta_t}A^{-1}g_t - A^{-1}C^T\gamma$. Substituting in the second equation, we get $\gamma = (CA^{-1}C^T)^{-1}(b - \frac{1}{\eta_t}CA^{-1}g_t)$. Using γ in the first equation, we can solve for ξ:

$$\xi = \xi_t \overbrace{- \frac{1}{\eta_t}A^{-1}g_t}^{\textbf{unconstr. (7)}} \overbrace{+ \frac{1}{\eta_t}A^{-1}C^T(CA^{-1}C^T)^{-1}CA^{-1}g_t}^{\textbf{zero set projection}} \quad (11)$$
$$\underbrace{-A^{-1}C^T(CA^{-1}C^T)^{-1}b}_{\textbf{offset}}$$

Our experiments in Section VI explicitly implement this update. The labels on the terms above hint at the goal of the next section, which answers the following question: Can we use this equation to go beyond the explicit analytical implementation and understand how this result relates to the projection problem in (9)?

IV. The Update Rule — Geometrical Intuition

Looking back at the constrained update rule in (11), we can explain its effect by analyzing each of its terms individually. Gaining this insight not only allows us to understand the algorithm we are implementing, but it also relates it to intuition for handling constraints in general. By the end of this section, we will have mapped the algorithm indicated by (11) to the projection problem in (9): take an unconstrained step, and then project it back onto the feasible region.

We can split the update rule in three parts, depicted in Fig.3: taking the unconstrained step, projecting it onto a hyperplane through the current trajectory that is parallel to the constraint surface approximation, and then correcting the offset between these hyperplanes:

1) The first term computes the **unconstrained** step: smooth the unconstrained euclidean gradient g_t through A^{-1} and scale it, as in (7). Intuitively, the other terms will need to adjust this step, such that the trajectory obtained at the end of the iteration, ξ_{t+1} is feasible. Therefore, these terms must implement the projection with respect to A, as shown in (9).
2) Linearizing h provides us with an approximation of the constraint surface, given by $C(\xi - \xi_t) + b = 0$. The current trajectory, ξ_t, lies on a parallel hyperplane, $C(\xi - \xi_t) = 0$. When ξ_t is feasible, $b = 0$ and the two are identical, intersecting the constraint surface at ξ_t. What the second term in the update rule does is to **project** the unconstrained increment onto the **zero set** of $C(\xi - \xi_t)$ with respect to our metric A, as depicted in Fig.3. Formally, the term is the solution to the problem that minimizes the adjustment to the new unconstrained trajectory (w.r.t. A) needed to satisfy $C(\xi - \xi_t) = 0$:

$$\min_{\Delta\xi} \ \frac{1}{2}\|\Delta\xi\|_A^2 \qquad (12)$$
$$s.t. \ \ C(\ (\xi_t - \frac{1}{\eta_t}A^{-1}g_t + \Delta\xi) - \xi_t\ ) = 0$$

   Therefore, the second term projects the unconstrained step onto the zero set of $C(\xi - \xi_t)$. If $b \neq 0$, the trajectory is still not on the approximation we have to the constraint surface, and the third step makes this correction.
3) After the first two steps, we obtain a trajectory on $C(\xi - \xi_t) = 0$, at an **offset** from the hyperplane that approximates the feasible region, $C(\xi - \xi_t) + b = 0$. Even if the Euclidean gradient g_t is 0 and the previous two terms had no effect, the trajectory ξ_t might have not been feasible, leading to $b \neq 0$. The third term subtracts this offset, resulting in a trajectory that lies on the approximate constraint surface. It is the solution to the problem that minimizes the adjustment to ξ_t (again, w.r.t. our metric) such that the trajectory gets back onto the target hyperplane:

$$\min_{\Delta\xi} \ \frac{1}{2}\|\Delta\xi\|_A^2 \qquad (13)$$
$$s.t. \ \ C(\ (\xi_t + \Delta\xi) - \xi_t\ ) + b = 0$$

   As Fig.3 shows, adding the third term to the result of the previous two steps (ξ_t when the unconstrained step is zero, somewhere else along $C(\xi - \xi_t) = 0$ otherwise) brings the trajectory onto the approximate constraint surface.

Overall, the algorithm can be thought of as first taking an unconstrained step in the direction dictated solely by the cost function, and then projecting it onto its guess of the feasible region in two steps, the last of which aims at correcting previous errors. For the special case of goal set constraints, which the next section addresses, the projection further simplifies to a purely Euclidean operator, which is then smoothed through the dynamics matrix.

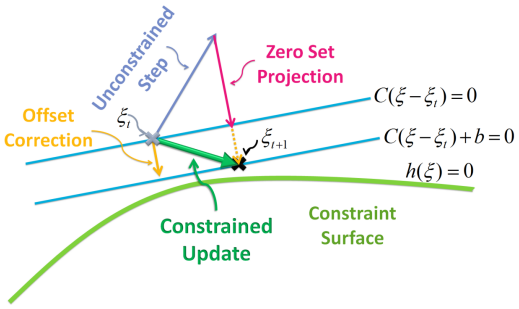

Fig. 3. The constrained update rule takes the unconstrained step and projects it under A onto the hyperplane through ξ_t parallel to the approximated constraint surface (given by the linearization $C(\xi - \xi_t) + b = 0$). Finally, it corrects the offset between the two hyperplanes, bringing ξ_{t+1} close to $h(\xi) = 0$.

V. Goal Sets as a Special Case

Extending CHOMP to handle full goal sets instead of a single goal can be seen as a two step procedure: First, the algorithm needs to be able to move the end-configuration, and we saw in Section II-B that this is a matter of simply re-deriving the dynamics matrix. Second, the algorithm has to constrain that movement to a limited set of feasible goals. Such sets can be described, for example, as simple shapes such as circles or rectangles, or as more complex Task Space Regions [14]. This section explains how the terms in the update rule for general constraints simplify in the special case of goal sets. The resulting update is intuitive and easy to implement.

Constraints that affect only the goal are a special case of trajectory constraints, for which $h(\xi) = h_n(q_n)$ (the constraint is a function of only the final configuration of the trajectory). Therefore, a large portion of the update rule will focus on the last configuration. Since $C = [0, \ldots, 0, \widetilde{C}]$, in this case C only affects the last block-row of A^{-1}, which we denote by $B_n \in \mathbb{R}^{d \times nd}$. Also note that the last $d \times d$ block in A^{-1}, B_{nn}, is in fact equal to βI_d, since there are no cross-coupling terms between the joints. Therefore, the update rule simplifies to:

$$\xi_{t+1} = \xi_t - \frac{1}{\eta_t}A^{-1}g_t + \frac{1}{\eta_t\beta}B_n^T\widetilde{C}^T(\widetilde{C}\widetilde{C}^T)^{-1}\widetilde{C}B_n g_t \qquad (14)$$
$$+ \frac{1}{\beta}B_n\widetilde{C}^T(\widetilde{C}\widetilde{C}^T)^{-1}b$$

The goal set CHOMP algorithm, as depicted in Fig.4, follows the classic "take an unconstrained step and project it" rule, only this time the projection is much simpler: it happens in *Euclidean configuration* space rather than in the A *metric trajectory* space. In other words, the same projection from Fig.3 now happens in Euclidean space and applies only to the end-configuration of the trajectory. To see this, note that $\frac{1}{\eta_t}B_n g_t$ gets the unconstrained step for the end configuration from $\frac{1}{\eta_t}A^{-1}g_t$, and $\widetilde{C}^T(\widetilde{C}\widetilde{C}^T)^{-1}\widetilde{C}$ projects it onto the row

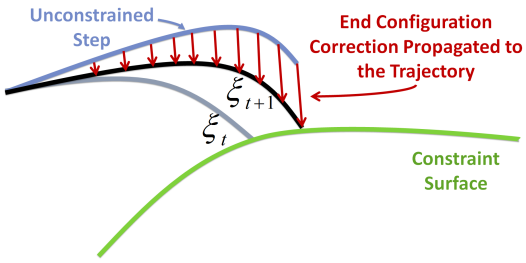

Fig. 4. One iteration of the goal set version of CHOMP: take an unconstrained step, project the final configuration onto the constraint surface, and propagate that change to the rest of the trajectory.

space of $\widetilde{C}$. This correction is then propagated to the rest of the trajectory, as illustrated by Fig.4, through $\frac{1}{\beta}B_n^T$. Looking back at Fig.2, imagine moving i from 2 (shown in the figure) to n: B_n, on each dimension, looks like a line from 0 to β. Therefore, $\frac{1}{\beta}B_n^T$ linearly interpolates from a zero change at the start configuration to the correction at the end point. Since B_n^T multiplies the last configuration by β, $\frac{1}{\beta}$ scales everything down such that the end-point projection applies exactly.

So far in this section, we showed that the projection onto a linearized version of the goal set constraints simplifies to a two step procedure. We first project the final configuration of the trajectory onto the linearized goal set constraint with respect to the Euclidean metric in the configuration space, which gives us a desired perturbation Δq_n of that final configuration. We then smooth that desired perturbation linearly back across the trajectory so that each configuration along the trajectory is perturbed by a fraction Δq_n; in particular for our prior, the perturbation at configuration i is $\Delta q_i = i/n\Delta q_n$. One can show that the linearization of the constraints is not required in general. For any goal set, the projection of a trajectory onto the set may be decomposed as prescribed above.[1] This algorithm works well in practice, although we caution that despite the procedure's intuitive appeal, one should be careful when implementing the procedure to follow the dictums of the derivation. In particular, while it might seem natural to project a configuration onto the goal set by minimizing the distance of the end-effector to the closest goal in task space, doing so effectively projects with respect to the wrong metric. Below, we demonstrate experimentally that such ad-hoc modifications may lead to worse results when coupled with the rest of the optimization procedure.

VI. Goal Sets in Practice: Experiments and Results

We demonstrate the utility of our method on four different tasks: grasping in cluttered environments with both an easy and a difficult starting pose, recycling, and handoffs. We start off our experiments by motivating the idea of goal sets – we assess how beneficial it is to think about an entire region of goals, and why it further makes sense to permit changes at the goal during the optimization. We then move on to the main experiments, which show that the goal set algorithm substantially outperforms the single goal algorithm on day-to-day manipulation tasks.

We see better performance in most of the scenarios, but this is not universally true. We address a couple of issues we encountered, the main one being the possibility that the goal set method converges to a trajectory that does not satisfy the goal constraint. A natural follow-up experiment assesses an algorithm that does a naive "projection" in task space, but that guarantees remaining on the goal manifold. Our comparison indicates that this algorithm does not perform well, indicating that projecting as prescribed by our derivation is indeed important.

A. Setup

Our experimental platform is a 7DOF BarrettWAM arm mounted on a Segway mobile base in a kitchen environment performing common manipulation tasks. Throughout our experiments, we use the same parameter setup for all algorithms: we use the deterministic variant presented here, with a decreasing step size at every iteration. The obstacle cost is weighted heavily relative to the smoothness cost. We compute the obstacle cost using body points along the skeleton, together with an obstacle padding of ϵ=0.2m within which the cost increases quadratically (we use the same workspace cost function as the one described in Ratliff [11]).

B. A motivating example

To exemplify the importance of goal sets and of allowing flexibility at the goal during optimization, we ran an experiment on a grasping scenario similar to the ones depicted in Fig.1 and Fig.5. The task was to obtain a trajectory that brings the arm to a pre-grasp pose. We defined a goal set to be comprised of a circle of transforms around a target bottle.

Next, we ran both the single goal and the goal set algorithms, initialized with a straight line trajectory from the start pose to each of 27 samples from the goal region. We obtained these initial goals by discretizing the goal set into 40 equidistant samples. We searched for Inverse Kinematics (IK) solutions for the arm, discarded 13 samples that were unreachable by the robot, and selected an IK solution for each of the 27 remaining samples, out of which 7 were in collision with the clutter. For the goal set variant, we chose the constraint function as the squared distance to the nearest feasible goal transform. Fig.6 shows the cost obtained by each method and how it positively correlates with the starting goal – dark regions represent infeasible starting goals that are in collision, while light regions represent feasible starting goals.

As expected, the cost profile in Fig.6 shows that the choice of a goal heavily influences the quality of the single goal CHOMP's solutions. Not only do the collision-free goals yield the lower cost trajectories, but also the cost tends to

[1]Specifically, one can show that for any single point in the goal set, the magnitude of the optimal projection is proportional to the Euclidean distance between the final configurations of the original trajectory and the projected trajectory. Therefore, the optimal projection is dictated by that Euclidean distance. Moreover, the optimal projection onto each of those given point in the goal set is given by the smoothing procedure described above.

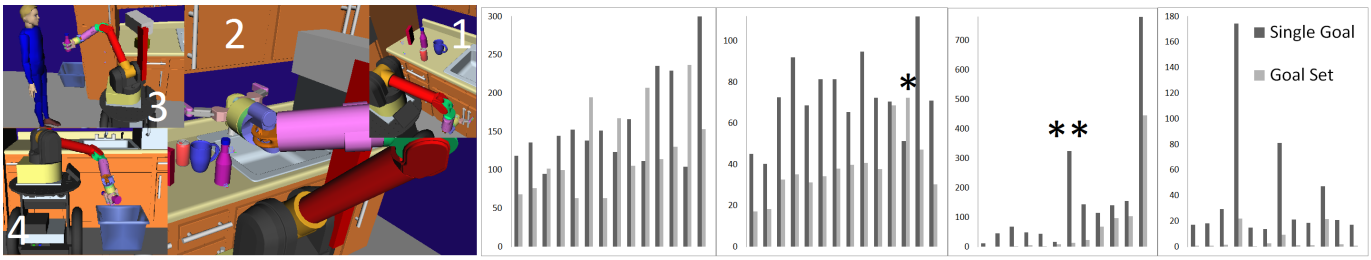

Fig. 5. The average cost increases over the estimated minimum for the scenarios in each of the four tasks: a difficult grasp, an easier grasp with a better starting configuration, a hand-off, and recycling or "drop" task. Each instance is an average across approx. 25 runs, one for each starting goal sample.

decrease towards the middle of the goals situated in free space. Therefore, a good goal configuration for one scenario can be very disadvantageous when the clutter pattern changes.

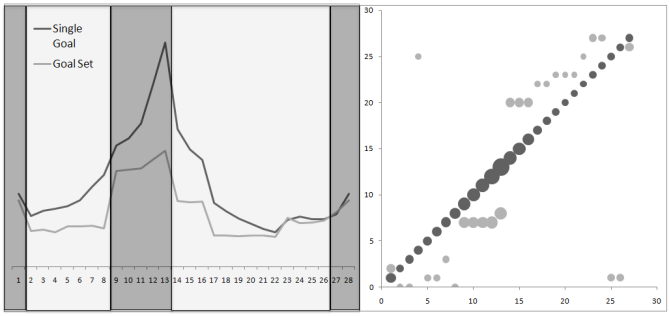

Fig. 6. Left: The costs (Y axis) that the single goal(black) and goal set(grey) algorithms obtain on a grasping scenario, when initialized with the straight-line trajectory to each of a number of goals around the bottle (X axis). The white regions represent collision-free goals. Right: For each algorithm, and for every starting goal (X axis), we plot what the final goal is (Y axis). The area of every bubble is proportional to the cost of the solution for that run.

But do we need to allow the optimizer to change the goal, as our method does? Or can we simply commit to one in the beginning of the optimization? Although there might be heuristics for selecting an initial goal (and future work addresses learning to select a goal based on the situation), it is likely that the goal set algorithm will outperform the single goal one when seeded at that initial goal, as indicated in Fig.6 by the downward shift in the cost profile. The same figure (right) shows a plot of the final goals reached, and illustrates (through the size of each bubble) what the cost of the solutions is. The goal set algorithm tends to move the end point away from the clutter, and the decrease in cost indicates that integrating the goal set into the optimization process usually increases the quality of the solutions, even when starting with a promising goal. The main experiments to follow strengthen this observation, showing both average cost and best cost improvement on the vast majority of the scenarios we tested.

C. Main experiments

Our experiments can be grouped into four main tasks, illustrated in Fig.5: two pre-grasping tasks differing in the starting position of the robot (an easy vs. a hard starting configuration), a hand-off task, and a recycling task. Each task consists of various scenarios that represent different obstacle configurations and, in the case of the last hand-offs and recycling, different initial poses of the robot.

For each scenario, we ran the same collection of tests depicted in Fig.6, resulting in approximately 1300 runs of each algorithm. We report the average and minimum cost, as well as the success rate (the fraction of times the solution was feasible). Fig.5 shows, for each scenario, the amount by which the algorithms deviated, on average, from the minimum cost achieved by either the single goal or goal set optimizers (we treat this as an estimate of the best achievable performance, thus removing the bias). In 88% of the scenarios, the goal set algorithm averages a lower cost. Furthermore, the best cost across all starting goals improves in 94% percent of the tests. Across the tasks, the success rate improves by 30%. Fig.7 shows an example from the scenario on (**) in Fig.5, on which the goal set version performed better by a high margin.

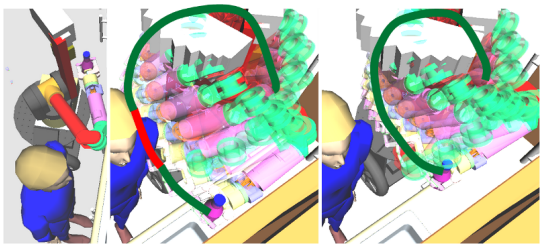

Fig. 7. An example from the scenario (**) in Fig.5. From left to right: the initial pose of the robot, the trajectory obtained by the single goal algorithm, and the one obtained by the goal set algorithm. The latter is able to take advantage of the goal set and find a goal that is further away from the human, yet still at a comfortable distance. Although the bottle offered was excluded from the trajectory visualization, it was treated as an extension to the robot for the feasibility check.

A rather surprising observation is that the goal set algorithm is faster on average (by a small amount) than single goal CHOMP , even though it needs to do the extra projection step. This is because CHOMP's computational bottleneck is computing Jacobians at the body points across the trajectory, and this can be avoided for body points that have cost zero, which means that the more points on the trajectory stay further from obstacles than a given ϵ=0.2m, the less time it will take for an iteration to conclude. For the recycling task, the improvement was the most apparent, reducing the single goal 5.15s to 4.24s, on a 2.4GHz Intel Core i5.

D. Issues

An issue we noticed in our experiments was that the CHOMP obstacle cost is less correlated to the feasibility of a trajectory than expected. A feasible trajectory that stays close to obstacles will accumulate more cost than an infeasible trajectory that collides slightly at one point, but on average stays further away from obstacles. We plan to address this issue in our future work.

Additionally, there are some scenarios, such as scenario (*) in Fig.5, that show a higher average cost for the goal set algorithm. This can in theory be explained by the fact that both these algorithms are local methods, and the goal set one could make a locally optimal decision which converges to a shallower local minima. but at the same time, we do expect that the average performance improves by allowing goal sets. A further analysis of these scenarios suggested a different explanation: although on most cases the goal set version was better, there were a few runs when it did not converge to a "goal-feasible" trajectory (and therefore reported a very high cost of the last feasible trajectory, which was close to the initial one). We noticed that this is mainly related to the projection being impeded by joint limits. Formalizing joint limits as trajectory constraints and projecting onto both constraint sets at the same time would avoid this problem.

E. The importance of a correct update rule

To overcome the problem of goal-infeasible solutions, a naive alternative to the algorithm we propose is to use a simple "projection" that does not linearize and is guaranteed to stay on the goal manifold at every step: project in task space rather than configuration space first, and find the closest IK solution at that point to the desired unconstrained step.

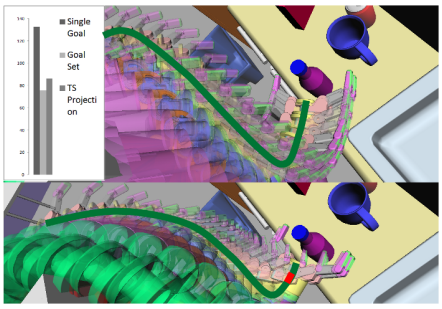

Fig. 8. Top: The goal set algorithm we derived finds a feasible soltution. Bottom: Doing an incorrect projection finds a trajectory that collides with the bottle. The graph shows the average cost across the entire scenario – doing the incorrect projection yields worse performance.

Our experiments with this naive update illustrate the importance of having a correct projection: Going back to scenario (*), where the correct projection was impeded by joint limits and produced goal-infeasible solutions, we found that although the solutions generated by this naive projection do satisfy the goal constraint, they still collide with the clutter. Furthermore, this method also fails in cases where the correct projection method succeeds, such as the one in Fig.8, and obtains a worse cost across an entire scenario. Moreover, the search for the closest IK solution increases the running time to approx. 40s in our implementation.

VII. Conclusion

In this work, we extended a trajectory optimization algorithm to handle constraints and mapped the solution the intuitive method of projecting unconstrained steps back onto the feasible region. We showed how this projection simplifies when analyzing goal sets as a special type of constraint and demonstrated that the idea works well in practice. We also stress the importance of correctly interpreting the solution; intuitively appealing alternatives often perform worse. The algorithm is by no means perfect – it is by nature local and cannot find globally optimal solutions. Introducing exploration is an active area of future work.

We are currently exploring the natural extensions of this work to other constraints, such as start sets for finding waypoints (for which the algorithm is identical, but the trajectory start rather than the end is allowed to change) or trajectory-wide end effector constraints, by developing an over-arching framework for understanding trajectory optimization, that integrates obstacle constraints, joint limits, etc. into a single prescriptive algorithm.

References

[1] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, "HERB: a home exploring robotic butler," *Autonomous Robots*, vol. 28, no. 1, pp. 5–20, 2010.

[2] WillowGarage, "The Personal Robot Project," 2008. [Online]. Available: http://www.willowgarage.com

[3] H. Nguyen, C. Anderson, A. Trevor, A. Jain, Z. Xu, and C. Kemp, "El-E: An Assistive Robot that Fetches Objects from Flat Surfaces," in *IEEE Proceedings of Human Robot Interaction, The Robotics Helpers Workshop*. IEEE, 2008.

[4] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, San Francisco, CA, Apr. 2000, pp. 995–1001.

[5] L. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration space," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] R. Weinstock, *Calculus of variations*. Dover publications, 1974.

[7] L. Pontryagin, *The mathematical theory of optimal processes*. Interscience New York, 1962.

[8] M. Zefran and V. Kumar, "A variational calculus framework for motion planning," in *IEEE Conference on Advanced Robotics*, Monterey, CA, 1997, pp. 415–420.

[9] S. Quinlan, "The real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1994.

[10] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, p. 1031, 2002.

[11] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[12] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2000.

[13] N. Ratliff, "Learning to search: Structured prediction techniques for imitation learning," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2009.

[14] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ser. ICRA'09. IEEE Press, 2009, pp. 1397–1403.