STOMP: Stochastic Trajectory Optimization for Motion Planning

Mrinal Kalakrishnan[1] Sachin Chitta[2] Evangelos Theodorou[1] Peter Pastor[1] Stefan Schaal[1]

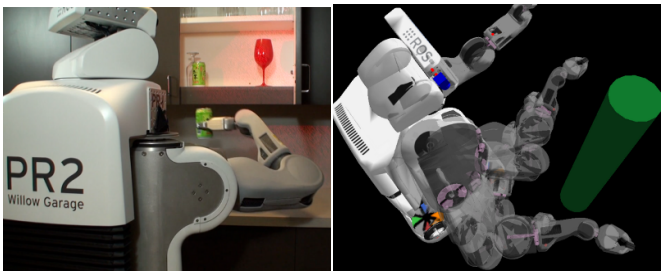
***Abstract*— We present a new approach to motion planning using a stochastic trajectory optimization framework. The approach relies on generating noisy trajectories to explore the space around an initial (possibly infeasible) trajectory, which are then combined to produced an updated trajectory with lower cost. A cost function based on a combination of obstacle and smoothness cost is optimized in each iteration. No gradient information is required for the particular optimization algorithm that we use and so general costs for which derivatives may not be available (e.g. costs corresponding to constraints and motor torques) can be included in the cost function. We demonstrate the approach both in simulation and on a mobile manipulation system for unconstrained and constrained tasks. We experimentally show that the stochastic nature of STOMP allows it to overcome local minima that gradient-based methods like CHOMP can get stuck in.**

I. Introduction

Motion planning for manipulation and mobile systems has received extensive interest in recent years. Motion planning for avoiding collision has been the most common goal, but there are other objectives like constraint handling, torque or energy minimization and achieving smooth paths that may be important in certain scenarios as well. Domestic and retail scenarios, in particular, will have lots of cases where constraint satisfaction may be a prime goal, e.g. carrying a glass of water. Mobile manipulation systems may need to minimize energy to conserve power and stay active for a longer period of time. Non-smooth jerky trajectories can cause actuator damage. There is definitely a need for a motion planner that can address such scenarios.

In this paper, we present a new approach to motion planning that can deal with general constraints. Our approach involves stochastic trajectory optimization using a series of noisy trajectories. In each iteration, a series of such trajectories is generated. The generated trajectories are simulated to determine their costs which are then used to update the candidate solution. Since no gradient information is required in this process, general constraints and additional non-smooth costs can be optimized.

We demonstrate our approach through both simulation and experimental results with the PR2 mobile manipulation robot. We consider end-effector orientation constraints, e.g. constraints that require the end-effector to keep a grasped object horizontal through the entire trajectory. We attempt to minimize collision costs using a signed distance field to measure proximity to obstacles. We further impose a smoothness cost that results in trajectories that can be directly executed on the robot without further smoothing. We also demonstrate the optimization of motor torques used to perform a movement.



(a) (b)

Fig. 1. (a) The Willow Garage PR2 robot manipulating objects in a household environment. (b) Simulation of the PR2 robot avoiding a pole in a torque-optimal fashion.

II. Related Work

Sampling-based motion planning algorithms have proved extremely successful in addressing manipulation problems [1], [2], [3], [4], [5]. They have been used to address task space constraints and to impose torque constraints while lifting heavy objects [6]. While sampling based planners result in feasible plans, they are often lacking in the *quality* of the paths produced. A secondary shortcutting step is often required to smooth and improve the plans output by these planners [7].

Another class of motion planners that have been applied to manipulation systems are optimization based planners. Motion planning for cooperating mobile manipulators was addressed in [8] where an optimal control approach was used to formulate the motion planning problem with obstacles and constraints. The problem was solved using a numerical approach by discretizing and solving for the desired plan. In [9], a covariant gradient descent technique was used for motion planning for a 7 DOF manipulator. This algorithm (called CHOMP) minimized a combination of smoothness and obstacle costs and required gradients for both. It used a signed distance field representation of the environment to derive the gradients for obstacles in the environment. We adopt a similar cost function for our approach, but in contrast to CHOMP, our optimization approach can handle general cost functions for which gradients are not available.

[1] Mrinal Kalakrishnan, Evangelos Theodorou, Peter Pastor, and Stefan Schaal are with the CLMC Laboratory, University of Southern California, Los Angeles, USA {kalakris,etheodor,pastorsa,sschaal}@usc.edu

[2] Sachin Chitta is with Willow Garage Inc., Menlo Park, CA 94025, USA sachinc@willowgarage.com

III. The STOMP Algorithm

Traditionally, motion planning is defined as the problem of finding a collision-free trajectory from the start configuration to the goal configuration. We treat motion planning as an optimization problem, to search for a smooth trajectory that minimizes costs corresponding to collisions and constraints. Specifically, we consider trajectories of a fixed duration T, discretized into N waypoints, equally spaced in time. In order to keep the notation simple, we first derive the algorithm for a 1-dimensional trajectory; this naturally extends later to multiple dimensions. This 1-dimensional discretized trajectory is represented as a vector $\boldsymbol{\theta} \in \mathbb{R}^N$. We assume that the start and goal of the trajectory are given, and are kept fixed during the optimization process.

We now present an algorithm that iteratively optimizes this discretized trajectory, subject to arbitrary state-dependent costs. While we keep the cost function general in this section, Section IV discusses our formulation for obstacle, constraint, energy, and smoothness costs. We start with the following optimization problem:

$$\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i) + \frac{1}{2}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\mathbf{R}\tilde{\boldsymbol{\theta}}\right] \tag{1}$$

where $\tilde{\boldsymbol{\theta}} = \mathcal{N}(\boldsymbol{\theta}, \boldsymbol{\Sigma})$ is a noisy parameter vector with mean $\boldsymbol{\theta}$ and variance $\boldsymbol{\Sigma}$. $q(\tilde{\boldsymbol{\theta}}_i)$ is an arbitrary state-dependent cost function, which can include obstacle costs, constraints and torques. $\mathbf{R}$ is a positive semi-definite matrix representing *control* costs. We choose $\mathbf{R}$ such that $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}$ represents the sum of squared accelerations along the trajectory. Let $\mathbf{A}$ be a finite differencing matrix that when multiplied by the position vector $\boldsymbol{\theta}$, produces accelerations $\ddot{\boldsymbol{\theta}}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\ddot{\boldsymbol{\theta}} = \mathbf{A}\boldsymbol{\theta} \tag{3}$$

$$\ddot{\boldsymbol{\theta}}^{\mathrm{T}}\ddot{\boldsymbol{\theta}} = \boldsymbol{\theta}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\mathbf{A})\boldsymbol{\theta} \tag{4}$$

Thus, the selection of $\mathbf{R} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$ ensures that $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}$ represents the sum of squared accelerations along the trajectory.

Previous work [9] has demonstrated the optimization of the non-stochastic version of Eqn. 1 using covariant functional gradient descent techniques. In this work, we instead optimize it using a derivative-free stochastic optimization method. This allows us to optimize arbitrary costs $q(\tilde{\boldsymbol{\theta}})$ for which derivatives are not available, or are non-differentiable or non-smooth.

Taking the gradient of the expectation in Eqn. 1 with respect to $\tilde{\boldsymbol{\theta}}$, we get:

$$\nabla_{\tilde{\boldsymbol{\theta}}}\left(\mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i) + \frac{1}{2}\tilde{\boldsymbol{\theta}}^{\mathrm{T}}\mathbf{R}\tilde{\boldsymbol{\theta}}\right]\right) = 0 \tag{5}$$

which leads to:

$$\mathbb{E}\left(\tilde{\boldsymbol{\theta}}\right) = -\mathbf{R}^{-1}\nabla_{\tilde{\boldsymbol{\theta}}}\left(\mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i)\right]\right) \tag{6}$$

Further analysis results in:

$$\mathbb{E}\left(\tilde{\boldsymbol{\theta}}\right) = -\mathbf{R}^{-1}\mathbb{E}\left(\nabla_{\tilde{\boldsymbol{\theta}}}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i)\right]\right) \tag{7}$$

The expression above can be written in the form $\mathbb{E}\left(\tilde{\boldsymbol{\theta}}\right) = -\mathbf{R}^{-1}\hat{\delta\boldsymbol{\theta}}_G$ where $\hat{\delta\boldsymbol{\theta}}_G$ is now the gradient estimate defined as follows:

$$\delta\hat{\boldsymbol{\theta}}_G = \mathbb{E}\left(\nabla_{\tilde{\boldsymbol{\theta}}}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i)\right]\right) \tag{8}$$

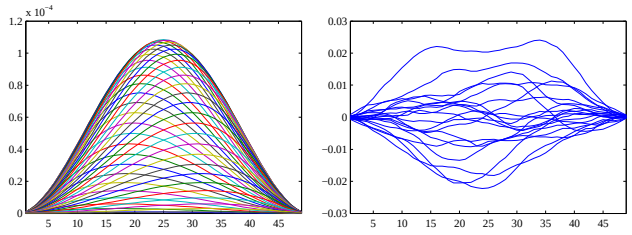
Previous approaches [9] have used the analytical functional gradient to derive an iterative gradient descent update rule. While this may be efficient, it requires a cost function that is smooth and differentiable. Moreover, even though this has not been proposed [9], for a given cost function $J(\boldsymbol{\theta})$ the positive definiteness condition of the hessian $\nabla_{\boldsymbol{\theta}\boldsymbol{\theta}}J(\boldsymbol{\theta}) > 0$ is required to guarantee convergence. Our proposed gradient estimation is motivated by the limitations of gradient based optimization when it comes to non-differentiable or non-smooth cost functions. Inspired by previous work in the probability matching literature [10] as well as recent work in the areas of path integral reinforcement learning [11], we propose an estimated gradient formulated as follows:

$$\delta\hat{\boldsymbol{\theta}}_G = \int \delta\boldsymbol{\theta}\ d\mathbf{P} \tag{9}$$

Essentially, the equation above is the expectation of $\delta\boldsymbol{\theta}$ (the noise in the parameter vector $\tilde{\boldsymbol{\theta}}$) under the probability metric $\mathbf{P} = \exp\left(-\frac{1}{\lambda}\mathbf{S}(\tilde{\boldsymbol{\theta}})\right)$ where $\mathbf{S}(\tilde{\boldsymbol{\theta}})$ is the state dependent cost defined on the trajectory and it is designed as $\mathbf{S}(\tilde{\boldsymbol{\theta}}) = \left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i)\right]$. Thus the stochastic gradient is now formulated as follows:

$$\delta\hat{\boldsymbol{\theta}}_G = \int \exp\left(-\frac{1}{\lambda}\mathbf{S}(\boldsymbol{\theta})\right)\ \delta\boldsymbol{\theta}\ d(\delta\boldsymbol{\theta}) \tag{10}$$

Even though our optimization procedure is static in the sense that it does not require the execution of a trajectory rollout, our gradient estimation process has strong connection to how the gradient of the value function is computed in the path integral stochastic optimal control framework [11]. More precisely the goal in the framework of stochastic optimal control is to find optimal controls that minimize a performance criterion. In the case of the path integral stochastic optimal control formalism, these controls are computed for every state $\mathbf{x}_{ti}$ as $\delta\hat{\mathbf{u}} = \int p(\mathbf{x})\delta\mathbf{u}$ where $\delta\mathbf{u}$ are the sampled controls and $p(\mathbf{x})$ corresponds to the



(a) (b)

Fig. 2. (a) Each curve depicts a column/row of the symmetric matrix $\mathbf{R}^{-1}$. (b) 20 random samples of $\boldsymbol{\epsilon}$, drawn from a zero mean normal distribution with covariance $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} = \mathbf{R}^{-1}$.

TABLE I

THE STOMP ALGORITHM

- **Given:**
  - Start and goal positions x_0 and x_N
  - An initial 1-D discretized trajectory vector $\boldsymbol{\theta}$
  - An state-dependent cost function $q(\boldsymbol{\theta}_i)$
- **Precompute:**
  - $\mathbf{A}$ = finite difference matrix (Eqn 2)
  - $\mathbf{R}^{-1} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}$
  - $\mathbf{M} = \mathbf{R}^{-1}$, with each column scaled such that the maximum element is $1/N$
- **Repeat until convergence of trajectory cost Q($\boldsymbol{\theta}$):**
  1) Create K noisy trajectories, $\tilde{\boldsymbol{\theta}}_1 \ldots \tilde{\boldsymbol{\theta}}_K$ with parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k = \mathcal{N}(0, \mathbf{R}^{-1})$
  2) For $k = 1 \ldots K$, compute:
     a) $S(\tilde{\boldsymbol{\theta}}_{k,i}) = q(\tilde{\boldsymbol{\theta}}_{k,i})$
     b) $P\left(\tilde{\boldsymbol{\theta}}_{k,i}\right) = \frac{e^{-\frac{1}{\lambda}S(\tilde{\boldsymbol{\theta}}_{k,i})}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\tilde{\boldsymbol{\theta}}_{l,i})}]}$
  3) For $i = 1 \ldots (N-1)$, compute: $[\delta\tilde{\boldsymbol{\theta}}]_i = \sum_{k=1}^{K} P(\tilde{\boldsymbol{\theta}}_{k,i})[\boldsymbol{\epsilon}_k]_i$
  4) Compute $\delta\boldsymbol{\theta} = \mathbf{M}\delta\tilde{\boldsymbol{\theta}}$
  5) Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$
  6) Compute trajectory cost $Q(\boldsymbol{\theta}) = \sum_{i=1}^{N} q(\boldsymbol{\theta}_i) + \frac{1}{2}\boldsymbol{\theta}^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}$

probability of every trajectory $\boldsymbol{\tau}_i$ starting from $\mathbf{x}_{ti}$ and ending in the terminal state $\mathbf{x}_{tN}$. This probability is defined as $p(\mathbf{x}) = \exp\left(-S(\boldsymbol{\tau}_i)\right)$ with $S(\boldsymbol{\tau}_i)$ being the cost of the path $\boldsymbol{\tau}_i = (\mathbf{x}_{ti}, ..., \mathbf{x}_{tN})$. Thus, $p(\mathbf{x})$ is inversely proportional to the cost $S(\boldsymbol{\tau}_i)$ and therefore paths with higher cost will have a lower contribution to the optimal controls than paths with lower cost. The process above is repeated for every state $\mathbf{x}_{ti}$ until the terminal $\mathbf{x}_{tN}$ (multistage optimization). Then the controls are updated according to $\mathbf{u} = \mathbf{u} + \delta\hat{\mathbf{u}}$ and new trajectories are generated. In our case, we assume that each state cost $q(\boldsymbol{\theta}_i)$ is purely dependent only on the parameter $\boldsymbol{\theta}_i$, and we do not blame future or past costs on the current state. Hence, we simplify the problem by defining a local trajectory cost $S(\boldsymbol{\theta}_i) = q(\boldsymbol{\theta}_i)$, i.e., we remove cumulation of costs. We find that this simplification significantly accelerates convergence in the experiments presented in Sec. V.

The final update equations for STOMP are presented in Table I. There are a few points which warrant further discussion:

A. Exploration

In order to keep control costs of noisy trajectories low, we sample the noise $\boldsymbol{\epsilon}$ from a zero mean normal distribution, with $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} = \mathbf{R}^{-1}$ as the covariance matrix, as shown in Figure 2(b). This is preferable to sampling with $\boldsymbol{\Sigma} = \mathbf{I}$ for several reasons: (1) samples $\boldsymbol{\epsilon}$ have a low control cost $\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{R}\boldsymbol{\epsilon}$, and thus allow exploration of the state space without significantly impacting the trajectory control cost; (2) these noisy trajectories may be executed without trouble on a physical system; (3) the samples do not cause the trajectory to diverge from the start or goal. Goal-convergent exploration is highly desirable in trajectory-based reinforcement learning of point to point movements, where dynamical systems have been designed that satisfy this property [12].

B. Trajectory updates

After generating K noisy trajectories, we compute their costs per time-step $S(\boldsymbol{\theta}_k, i)$ (Table I, Step 2(a)). In Step 2(b), we compute the probabilties $P(\boldsymbol{\theta}_k, i)$ of each noisy trajectory, per time-step. The parameter λ regulates the sensitivity of the exponentiated cost, and can be automatically optimized per time-step to maximally discriminate between the experienced costs. We compute the exponential term in Step 2(b) as:

$$e^{-\frac{1}{\lambda}S(\boldsymbol{\theta}_{k,i})} = e^{-h\frac{S(\boldsymbol{\theta}_{k,i}) - \min S(\boldsymbol{\theta}_{k,i})}{\max S(\boldsymbol{\theta}_{k,i}) - \min S(\boldsymbol{\theta}_{k,i})}}, \qquad (11)$$

with h set to a constant, which we chose to be $h = 10$ in all our evaluations. The max and min operators are over all noisy trajectories k. The *noisy* update for each time-step is then computed in Step 3 as the probability-weighted convex combination of the noisy parameters for that time-step.

Finally, in Step 4, we smooth the noisy update using the $\mathbf{M}$ matrix, before updating the trajectory parameters in Step 5. The $\mathbf{M}$ matrix is formed by scaling each column of $\mathbf{R}^{-1}$ (shown in Figure 2(a)) such that the highest element in the column has a magnitude $1/N$. This scaling ensures that no updated parameter exceeds the range that was explored in the noisy trajectories. Multiplication with $\mathbf{M}$ ensures that the updated trajectory remains smooth, since it is essentially a projection onto the basis vectors of $\mathbf{R}^{-1}$ shown in Figure 2(a).

These trajectory updates can be considered *safer* than a standard gradient descent update rule. The new trajectory is essentially a convex combination of the noisy trajectories which have already been evaluated, i.e. there are no unexpected jumps to unexplored parts of the state space due to a noisy gradient evaluation. Our iterative update rule is analogous to an expectation-maximization (EM) algorithm, in which we update the mean of our trajectory sampling distribution to match the distribution of costs obtained from sampling in the previous iteration. This procedure guarantees that the average cost is non-increasing, if the sampling is assumed to be dense [10], [13]. An additional advantage is that no gradient step-size parameter is required; the only open parameter in this algorithm is the magnitude of the exploration noise.

IV. MOTION PLANNING FOR A ROBOT ARM

In this section, we discuss the application of the stochastic trajectory optimization algorithm in Table I to the problem of motion planning of a high-dimensional robot manipulator.

We address the design of a cost function that allows planning for obstacle avoidance, optimization of task constraints, and minimization of joint torques.

A. Setup

The algorithm in Table I was derived for a one-dimensional discretized trajectory. Scaling this to multiple dimensions simply involves performing the sampling and update steps for each dimension independently in each iteration. The computational complexity of the algorithm thus scales *linearly* with the dimensionality of the problem. For the application to robot motion planning, we represent the trajectory in joint space, with a fixed duration and discretization. We assume that the start and goal configurations are provided in joint space.

B. Cost Function

The cost function we use is comprised of obstacle costs q_o, constraint costs q_c, and torque costs q_t.

$$q(\boldsymbol{\theta}) = \sum_{t=0}^{T} q_o(\boldsymbol{\theta}_t) + q_c(\boldsymbol{\theta}_t) + q_t(\boldsymbol{\theta}_t) \tag{12}$$

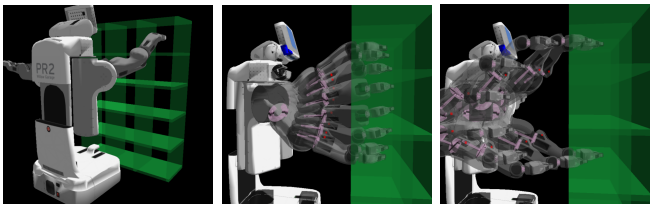
The additional smoothness cost $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}$ is already incorporated in the optimization problem in Equation 1.

1) Obstacle costs: We use an obstacle cost function similar to that used in previous work on optimization-based motion planning [9]. We start with a boolean voxel representation of the environment, obtained either from a laser scanner or from a triangle mesh model. Although our algorithm can optimize such non-smooth boolean-valued cost functions, faster convergence can be achieved by using a function for which local gradients are available (or can be estimated by sampling). We compute the signed Euclidean Distance Transform (EDT) [14] of this voxel map. The signed EDT $d(x)$, computed throughout the voxel grid, provides information about the distance to the boundary of the closest obstacle, both inside and outside obstacles. Values of the EDT are negative inside obstacles, zero at the boundary, and positive outside obstacles. Thus, the EDT provides discretized information about penetration depth, contact and proximity.

We approximate the robot body $\mathcal{B}$ as a set of overlapping spheres $b \in \mathcal{B}$. We require all points in each sphere to be a distance at least ϵ away from the closest obstacle. This constraint can be simplified as the center of the sphere being at least $\epsilon + r$ away from obstacles, where r is the sphere radius. Thus, our obstacle cost function is as follows:

$$q_o(\boldsymbol{\theta}_t) = \sum_{b \in \mathcal{B}} \max(\epsilon + r_b - d(x_b), 0)\|\dot{x}_b\|, \tag{13}$$

where r_b is the radius of sphere b, x_b is the 3-D workspace position of sphere b at time t as computed from the kinematic model of the robot. A straightforward addition of cost magnitudes over time would allow the robot to move through a high-cost area very quickly in an attempt to lower the cost.



(a) (b) (c)

Fig. 3. (a) Simulation setup used to evaluate STOMP as a robot arm motion planner. (b) Initial straight-line trajectory between two shelves. (c) Trajectory optimized by STOMP to avoid collision with the shelf, constrained to maintain the upright orientation of the gripper.

Multiplication of the cost by the magnitude of the workspace velocity of the sphere ($\|\dot{x}_b\|$) avoids this effect [9].

2) Constraint costs: We optimize constraints on the end-effector position and/or orientation by adding the magnitude of constraint violations to the cost function.

$$q_c(\boldsymbol{\theta}_t) = \sum_{c \in C} |v_c(\boldsymbol{\theta}_t)|, \tag{14}$$

where C is the set of all constraints, v_c is a function that computes the magnitude of constraint violation for constraint $c \in C$.

3) Torque costs: Given a suitable dynamics model of the robot, we can compute the feed-forward torque required at each joint to track the desired trajectory using inverse dynamics algorithms [15]. The motor torques at every instant of time are a function of the joint states and their derivatives:

$$\boldsymbol{\tau}_t = f(\mathbf{x}_t, \dot{\mathbf{x}}_t, \ddot{\mathbf{x}}_t), \tag{15}$$

where $\mathbf{x}_t = \boldsymbol{\theta}_t$ represents the joint states at time t, and $\dot{\mathbf{x}}_t, \ddot{\mathbf{x}}_t$ are the joint velocities and accelerations respectively, obtained by finite differentiation of $\boldsymbol{\theta}$.

Minimization of these torques can be achieved by adding their magnitudes to the cost function:

$$q_t(\boldsymbol{\theta}_t) = \sum_{t=0}^{T} |\boldsymbol{\tau}_t| dt \tag{16}$$

C. Joint Limits

Joint limits can potentially be dealt with by adding a term to the cost function that penalizes joint limit violations. However, we prefer to eliminate them during the exploration phase by clipping noisy trajectories $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ at the limits, i.e. an L_1 projection on to the set of feasible values. Since the noisy trajectory stays within the limits, the updated trajectory at each iteration must also respect the limits, since it is a convex combination of the noisy trajectories. Additionally, since the convex combination of noise is smoothed through the $\mathbf{M}$ matrix, the resulting updated trajectory smoothly touches the joint limit as opposed to bumping into it at high speed.

V. Experiments

STOMP is an algorithm that performs local optimization, i.e. it finds a locally optimum trajectory rather than a global one. Hence, performance will vary depending on the initial

TABLE II

RESULTS OBTAINED FROM PLANNING ARM TRAJECTORIES IN THE SIMULATED SHELF ENVIRONMENT SHOWN IN FIGURE 3(A).

Scenario	STOMP Unconstrained	CHOMP Unconstrained	STOMP Constrained
Number of successful plans	210 / 210	149 / 210	196 / 210
Avg. planning time to success (sec)	0.88 ± 0.40	0.71 ± 0.25	1.86 ± 1.25
Avg. iterations to success	52.1 ± 26.6	167.1 ± 113.8	110.1 ± 78.0

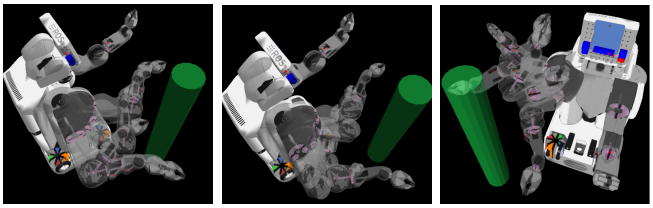
trajectory used for optimization. STOMP cannot be expected to solve typical motion planning benchmark problems like the alpha puzzle [16] in a reasonable amount of time. In this section, we evaluate the possibility of using STOMP as a motion planner for a robot arm in typical day-to-day tasks that might be expected of a manipulator robot. We conduct experiments on a simulation of the Willow Garage PR2 robot in a simulated world, followed by a demonstration of performance on the real robot.

A. Simulation

We created a simulation world containing a shelf with 15 cabinets, as shown in Figure 3(a). Seven of these cabinets were reachable by the 7 degree-of-freedom right arm of the PR2. We tested planning of arm trajectories between all pairs of the reachable cabinets in both directions, resulting in 42 different planning problems. Since the algorithm is stochastic in nature, we repeated these experiments 5 times. In each case, the initial trajectory used was a straight-line through configuration space of the robot. Each trajectory was 5 seconds long, discretized into 100 time-steps.

These planning problems were repeated in two scenarios with two different cost functions. The *unconstrained* scenario used a cost function which consisted of only obstacle costs and smoothness costs. Success in this scenario implies the generation of a collision-free trajectory. In the *constrained* scenario, we added a constraint cost term to the cost function. The task constraint was to keep the gripper upright at all times, i.e. as though it were holding a glass of water. Specifically, the roll and pitch of the gripper was constrained to be within ± 0.2 radians. Success in this scenario involves generation of a collision-free trajectory that satisfies the task constraint within the required tolerance. We also evaluated the performance of CHOMP, a gradient-based optimizer [9] on the unconstrained scenario using the same cost function. The exploration noise magnitude for STOMP, and the gradient descent step size for CHOMP were both tuned to achieve good performance without instability. Both algorithms were capped at 500 iterations. For STOMP, $K = 5$ noisy trajectory samples were generated at each iteration, and an additional 5 best samples from previous iterations used in the update step.

Table II shows the results obtained from these experiments. STOMP produced a collision-free trajectory in all trials in the unconstrained scenario. In contrast, CHOMP fails on nearly 30% of them, presumably due to the gradient descent getting stuck in local minima of the cost function[1]. The execution times are comparable, even though CHOMP usually requires more iterations to achieve success. This is because each iteration of STOMP requires multiple trajectory cost evaluations, but can make larger steps in a more stable fashion than the CHOMP gradient update.



(a) (b) (c)

Fig. 4. Planning problem used to evaluate torque minimization. (a) Plan obtained without torque minimization: arm is stretched. (b,c) Two different plans obtained with torque minimization. In (b), the entire arm is pulled down, while in (c) the elbow is folded in: both solutions require lower gravity compensation torques. Figure 5(b) shows the torques required for these movements.

In the constrained scenario, 93.3% of trials resulted in plans that were both collision-free and satisfied the task constraints. Figure 5(a) shows the iterative evolution of trajectory costs for one of the constrained planning problems, averaged over ten trials.

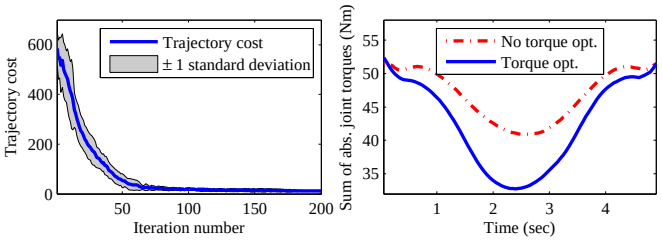
These results are obtained when initializing the algorithm with a naïve straight-line trajectory through configuration space, usually infeasible due to collisions with the environment (see Fig. 3(b)). The algorithm is able to push the trajectory out of collision with the environment, as well as optimize secondary costs like constraints and torques. The trajectory after optimization is ready for execution on a robot, i.e., no smoothing is necessary as is commonly used in conjunction with sampling-based planners [7].

In order to test the part of the cost function that deals with minimization of torques, we ran 10 trials on the planning problem shown in Figure 4, with and without the torque term in the cost function. The resulting torques of the optimized movements in both cases are shown in Figure 5(b). Since the movements are rather slow, most of the torques are used in gravity compensation. Hence the torques at the beginning and end of the movement stay the same in both cases. The planner finds parts of the state space towards the middle of the movement which require lower torques to support the weight of the arm. Interestingly, the planner usually finds one of two solutions as shown in Figures 4(b) and 4(c): (1) the entire arm is pulled down, or (2) the elbow is folded in on itself, both of which require lower torque than the stretched out configuration.

B. Real Robot

The attached video shows demonstrations of trajectories planned using STOMP in a household environment, executed

[1]This result was obtained using the standard CHOMP gradient update rule, not the Hamiltonian Monte Carlo variant. Please refer Sec. VI for details.



(a) (b)

Fig. 5. (a) Iterative evolution of trajectory costs for 10 trials of STOMP on a constrained planning task. (b) Feed-forward torques used in the planning problem shown in Figure 4, with and without torque optimization, averaged over 10 trials.

on the Willow Garage PR2 robot.

C. Code, Replication of Results

All of the software written for this work has been published under the BSD open source license, and makes use of the Robot Operating System (ROS) [17]. Further instructions on installing the software and replicating the results in this paper can be found at [18].

VI. Discussion

A Hamiltonian Monte Carlo variant of CHOMP is discussed in [9], [19], as a principled way of introducing stochasticity into the CHOMP gradient update rule. While theoretically sound, we found this method difficult to work with in practice. It introduces additional parameters which need to be tuned, and requires multiple random restarts to obtain a successful solution. In contrast, our algorithm requires minimal parameter tuning, does not need cost function gradients, and uses a stable update rule which under certain assumptions guarantees that the average cost is non-increasing.

VII. Conclusions

We have presented an algorithm for planning smooth trajectories for high-dimensional robotic manipulators in the presence of obstacles. The planner uses a derivative-free stochastic optimization method to iteratively optimize cost functions that may be non-differentiable and non-smooth. We have demonstrated the algorithm both in simulation and on a mobile manipulator, for obstacle avoidance, optimization of task constraints and minimization of motor torques used to execute the trajectory.

A possibility for future work is to augment this local trajectory optimizer with a trajectory library approach, which can recall previous trajectories used in similar situations, and use them as a starting point for futher optimization [20]. The STOMP algorithm could also be applied to problems in trajectory-based reinforcement learning, where costs can only be measured by execution on a real system; we intend to explore these avenues in future work.

Acknowledgements

This research was conducted while Mrinal Kalakrishnan and Peter Pastor were interns at Willow Garage. This research was additionally supported in part by National Science Foundation grants ECS-0326095, IIS-0535282, IIS-1017134, CNS-0619937, IIS-0917318, CBET-0922784, EECS-0926052, CNS-0960061, the DARPA program on Advanced Robotic Manipulation, the Army Research Office, the Okawa Foundation, and the ATR Computational Neuroscience Laboratories. Evangelos Theodorou was supported by a Myronis Fellowship.

References

[1] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *IEEE International Conference on Robotics and Automation*, May 2009.

[2] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. J. Kuffner., "Bispace planning: Concurrent multi-space exploration," in *Robotics: Science and Systems*, Zurich, Switzerland 2008.

[3] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *In Proc. 20th Int'l Symp. Robotics Research (ISRR'03)*, 2003.

[4] R. B. Rusu, I. A. Şucan, B. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time perception guided motion planning for a personal robot," in *International Conference on Intelligent Robots and Systems*, St. Louis, USA, October 2009.

[5] I. A. Şucan, M. Kalakrishnan, and S. Chitta, "Combining planning techniques for manipulation using realtime perception," in *IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 2010.

[6] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE International Conference on Robotics and Automation*, May 2009.

[7] K. Hauser and V. Ng-Thow-Hing, "Fast Smoothing of Manipulator Trajectories using Optimal Bounded-Acceleration Shortcuts," in *IEEE International Conference on Robotics and Automation*, 2010.

[8] J. P. Desai and V. Kumar, "Motion planning for cooperating mobile manipulators," *Journal of Robotic Systems*, vol. 16(10), pp. 557–579, 1999.

[9] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 12–17.

[10] P. Dayan and G. E. Hinton, "Using em for reinforcement learning," *Neural Computation*, vol. 9, 1997.

[11] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: a path integral approach," in *IEEE International Conference on Robotics and Automation*, 2010.

[12] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 15*. MIT Press, 2002, pp. 1547–1554.

[13] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Fitness expectation maximization," *Parallel Problem Solving from Nature–PPSN X*, pp. 337–346, 2008.

[14] Q. Ye, "The signed Euclidean distance transform and its applications," in *9th International Conference on Pattern Recognition, 1988.*, 1988, pp. 495–499.

[15] R. Featherstone, *Rigid body dynamics algorithms*. Springer-Verlag New York Inc, 2008.

[16] L. Zhang, X. Huang, Y. Kim, and D. Manocha, "D-plan: Efficient collision-free path computation for part removal and disassembly," *Journal of Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 774–786, 2008.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *International Conference on Robotics and Automation*, ser. Open-Source Software workshop, 2009.

[18] "STOMP: Stochastic trajectory optimization for motion planning," http://www.ros.org/wiki/Papers/ICRA2011_Kalakrishnan.

[19] N. D. Ratliff, "Learning to search: structured prediction techniques for imitation learning," Ph.D. dissertation, Carnegie Mellon University, 2009.

[20] N. Jetchev and M. Toussaint, "Trajectory Prediction in Cluttered Voxel Environments," in *IEEE International Conference on Robotics and Automation*, 2010.