Multigrid CHOMP with Local Smoothing

Keliang He Elizabeth Martin Matt Zucker

Department of Engineering
Swarthmore College, Swarthmore PA, USA
Email: mzucker1@swarthmore.edu

***Abstract*— The Covariant Hamiltonian Optimization and Motion Planning (CHOMP) algorithm has found many recent applications in robotics research, such as legged locomotion and mobile manipulation. Although integrating kinematic constraints into CHOMP has been investigated, prior work in this area has proven to be slow for trajectories with a large number of constraints. In this paper, we present Multigrid CHOMP with Local Smoothing, an algorithm which improves the runtime of CHOMP under constraints, without significantly reducing optimality. The effectiveness of this algorithm is demonstrated on two simulated problems, and on a physical HUBO+ humanoid robot, in the context of door opening. We demonstrate order-of-magnitude or higher speedups over the original constrained CHOMP algorithm, while achieving within 2% of the performance of the original algorithm on the underlying objective function.**

I. Introduction

In recent years, the Covariant Hamiltonian Optimization and Motion Planning (CHOMP) algorithm has proven to be a successful method for quickly generating locally optimal trajectories while avoiding obstacles [1]. In the original CHOMP algorithm, equality constraints were not taken into consideration. However, such constraints are abundant in robotic applications [2], [3], due to their ability to specify end effector orientation and closed kinematic chains, to name just two examples. Research has been done on incorporating constraints into CHOMP, such as using constraints as goal sets [4] as opposed to a fixed goal point. However, it is known that constrained CHOMP has a long runtime when many constraints are used (see subsection II-B for details).

In this paper, we present Multigrid CHOMP, a method incorporating the multigrid method into constrained CHOMP to drastically decrease the runtime while maintaining the high trajectory quality of the original algorithm. Furthermore, we present the addition of Local Smoothing to Multigrid CHOMP to improve its optimality further without increasing the runtime significantly.

One key motivation for this work is the DARPA Robotics Challenge (DRC), a U.S. government-sponsored competition aimed at promoting innovation in robotic technology for disaster response operations. The DRC involves eight tasks, including driving a utility vehicle, walking over rough terrain, and climbing a ladder, among others. Although the algorithm presented here was developed to address the door opening task of the DRC, many the tasks share in common a need for an efficient planner capable of generating high quality robot motion which obeys kinematic constraints.

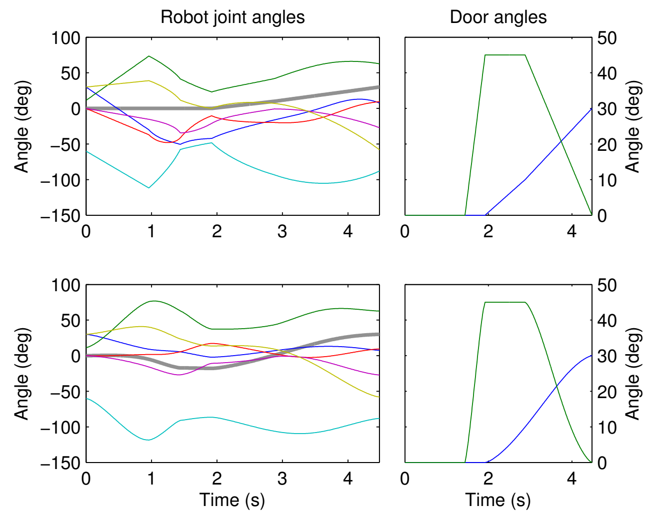

Fig. 1: Joint angles for the HUBO+ humanoid robot (left) and door (right) before (top) and after (bottom) minimizing accelerations while preserving closed kinematic chain constraints on a door opening task. The traces plotted on the left are the six DOF for the arm (fine colored lines), as well as waist yaw (bold gray line). Note that the C0-continuous original trajectory, generated via IK, has been transformed into one with continuous higher-order derivatives throughout.

The rest of this document is organized as follows: section II provides the mathematical background underlying constrained CHOMP, as well as the algorithms used. The software implementation is outlined in section III, and section IV details experimental evaluations on two simulated platforms and one real robotic platform. Finally, section V concludes the work.

II. Theory

The goal of CHOMP is to produce high-quality robot motion which is smooth in the sense of minimizing some sum of squared derivatives along the trajectory. Let a robot configuration for a robot with m degrees of freedom (DOF) be represented by a vector $q \in \mathbb{R}^m$. Then a trajectory $\xi = (q_1^T, \ldots, q_n^T)^T$ is represented in CHOMP as a sequence of n waypoints equally spaced in time, with two fixed endpoints q_0 and q_{n+1}.

Using the finite difference approximation of a derivative, we can always represent any sum of squared derivatives along the trajectory as a function of the form

$$f(\xi) = \frac{1}{2}\|K\xi + e\|^2 \tag{1}$$

where K is a finite differencing matrix, and e is a vector holding information about boundary conditions. In this work, we minimize squared acceleration, leading to a finite differencing matrix of size $m(n+2) \times mn$:

$$K = \begin{bmatrix} 1 & & & & & \\ -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & & \ddots & & & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \\ & & & & 1 \end{bmatrix} \otimes I_{mxm}$$

We also set e to encode zero velocity at the endpoints by inserting appropriately scaled copies of q_0 and q_{n+1} at each end, with zeros in between. The expression for $f(\xi)$ can be simplified into the following quadratic equation:

$$\begin{aligned} f(\xi) = \frac{1}{2}\|K\xi + e\|^2 &= \frac{1}{2}(K\xi + e)^T(K\xi + e) \\ &= \frac{1}{2}\xi^T A\xi + \xi^T b + c \end{aligned} \tag{2}$$

where $A = K^T K, b = \frac{1}{2}K^T e, c = \frac{1}{2}e^T e$. In particular, we note that the matrix A of size $mn \times mn$ is positive definite, band-diagonal, and quite sparse.

Although the original CHOMP algorithm was intended to generate collision free motion, we do not explicitly consider collision checking in this work. Instead, we assume that the trajectories are validated for collisions before and after optimization. However, we note that all of the efficient collision handling routines from the original CHOMP work can be folded into the objective function f, and the effects of obstacle avoidance would arise straightforwardly in the objective gradient ∇f without significantly altering the discussion below.

A. Handling constraints via Lagrange multipliers

In solving a trajectory optimization problem, we are often faced with constraints such as keeping a manipulated object upright, or preserving a closed kinematic chain. Such constraints (and many others) can be phrased as a set of equality constraints on the trajectory of the form $h(\xi) = 0$, with $h : \mathbb{R}^{mn} \to \mathbb{R}^k$.

Constrained CHOMP, originally proposed in [4], uses the method of Lagrange multipliers to set up a gradient descent problem to optimize trajectories under constraints. The overall Lagrangian in the vicinity of the current trajectory ξ is given by

$$L(\delta, \lambda) = f(\xi + \delta) + \lambda^T h(\xi + \delta) \tag{3}$$

Our goal is to take a small step δ at each iteration to move toward the constraint manifold while moving along the manifold to smaller objective values. By linearizing the Lagrangian and adding a penalty on step size, we obtain

$$L(\delta, \lambda) = \underbrace{f(\xi) + \nabla f(\xi)^T \delta}_{\substack{\text{Linearization of} \\ f(\xi+\delta)}} + \overbrace{\frac{1}{2\alpha}\delta^T A\delta}^{\substack{\text{Step} \\ \text{size penalty}}} + \lambda^T \underbrace{[h(\xi) + H\delta]}_{\substack{\text{Linearization of} \\ h(\xi+\delta)}} \tag{4}$$

Here, H denotes the constraint Jacobian, which is typically quite sparse since each of the k constraints is usually only active during a single timestep of the trajectory (as shown in Figure 12). Also, $\delta^T A\delta = \|\delta\|_A^2$ is the norm of δ under the metric A, as opposed to the usual Euclidean metric. The intent is to produce incremental changes to the trajectory which are themselves smooth (i.e. contributing little additional acceleration). By using A as the metric, the incremental change δ is projected onto a space of trajectories with low acceleration. Accordingly, if the gradient and constraint suggest that a particular point should be moved, the change will cause its neighbors to also move in the same direction. Therefore, changes take less time to propagate down the trajectory, speeding the overall convergence of the algorithm. Finally, the variable α is a parameter specifying the tradeoff between objective function reduction and stepsize, and is required due to the fact that the first order Taylor series expansion is only valid for a small neighborhood around ξ.

The gradient of L can be set to 0 to obtain δ and λ.

$$\begin{aligned} \nabla L = \begin{bmatrix} \frac{\partial L}{\partial \delta} \\ \frac{\partial L}{\partial \lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} &= \begin{bmatrix} \nabla f(\xi) + \frac{1}{\alpha}A\delta + H^T\lambda \\ h(\xi) + H\delta \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{\alpha}A & H^T \\ H & 0 \end{bmatrix} \begin{bmatrix} \delta \\ \lambda \end{bmatrix} + \begin{bmatrix} \nabla f(\xi) \\ h(\xi) \end{bmatrix} \end{aligned} \tag{5}$$

Solving for δ gives

$$\begin{aligned} \delta = &-\alpha(A^{-1} - A^{-1}H^T Q^{-1} H A^{-1})\nabla f(\xi) \\ &- A^{-1}H^T Q^{-1} h(\xi) \end{aligned} \tag{6}$$

where $Q = HA^{-1}H^T$. This is the update rule for gradient descent originally derived in [4].

B. Constrained CHOMP runtime

Most previous implementations of CHOMP have focused on minimizing velocity, as opposed to higher-order derivatives. This is largely due to the choice of algorithm for solving equations of the form

$$Ax = y$$

using the A matrix defined in the previous section. In previous CHOMP implementations, Thomas' backsubstitution algorithm [5] was used due to both its $O(mn)$ speed and existing implementations in a number of math tools and libraries. However, the algorithm is restricted to tridiagonal matrices. For CHOMP, A is a tridiagonal matrix only if K has only 2 entries per row. Thus if Thomas' algorithm is used for applying A^{-1}, minimizing velocity is the only choice.

In our implementation, the skyline Cholesky decomposition [6] is used instead. Because A is band-diagonal, its

Cholesky decomposition is also band-diagonal, allowing us to solve this equation in both $O(mn)$ space and time for any particular derivative used. With a suitable choice of K and e, our algorithm could minimize jerk just as easily as acceleration without affecting the asymptotic runtime.

Now, we consider the runtime for a single update of Equation 6 above. The constraint Jacobian H is of size $k \times mn$. Since typically there is a separate set of constraints active at each timestep, and since the number of constraints per timestep is on the order of the number of degrees of freedom of the robot, we also assume $k = O(mn)$. Using an optimized solver for A^{-1}, we assume the matrix $A^{-1}H^T$ can therefore be computed in $O(m^2n^2)$ time as opposed to the $O(m^3n^3)$ runtime implied by naïve matrix inversion. Unfortunately, the matrix Q in Equation 6 is dense, and inverting it takes time $O(k^3) = O(m^3n^3)$. This step is the dominant one in the algorithm, leading to an overall runtime of $O(m^3n^3)$ for the constrained CHOMP update.

C. Multigrid CHOMP

In order to reduce the runtime of constrained CHOMP, we adopt a multiresolution approach to produce Multigrid CHOMP. Multigrid is a group of algorithms used for quickly solving numerical problems [7], which has been successfully used in recent graphics applications to approximate solutions to complex diffusion problems [8], [9]. The core motivation for multigrid in our domain is that gradient descent will converge more quickly when initialized using an already near-optimal trajectory. Hence, we start with a coarsely discretized trajectory with a small number of waypoints n, and alternate stages of optimizing the trajectory with upsampling by inserting new points between each pair of previous trajectory elements, doubling n at each iteration.

When optimizing an upsampled trajectory, we fix the points retained from the downsampled version, and only optimize the new points added during the upsampling step. Although this is by no means globally optimal, it does result in an approximate 8x speedup by reducing the size of the matrix Q in Equation 6 by a factor of two. This is equivalent to computing A, H, and Q as defined in the previous subsections, and subsequently discarding the rows/columns corresponding to the odd-numbered timesteps (i.e. the fixed points); however, our implementation avoids computing the discarded elements.

In the next section, we introduce the concept of local smoothing, which attempts to alleviate the loss in optimality due to fixing alternate waypoints, while still maintaining low runtimes. See the bottom row of Figure 3 for an illustration of the algorithm in action.

D. Local Smoothing

As previously stated, Multigrid CHOMP only performs optimization on the newly added points at each temporal resolution. Although the fixed points may have been optimal in the coarser representation, their optimal location changes once the new points are considered. In order to rectify this effect, local smoothing is applied. The motivation for performing local smoothing is to apply slight "tweaks" to the previously established positions of waypoints without incurring the runtime penalty associated with the constrained CHOMP update from Equation 6. The underlying assumption is that the overall shape of the trajectory is most likely correct, but that the addition of new waypoints creates small opportunities for shortcutting motions that depend only on the new points added.

To perform local smoothing, the Lagrange Multiplier method is used to minimize f_t subject to $h_t = 0$, where f_t is the contribution to the objective function (squared acceleration) at time t, which considers only q_t and just enough neighboring waypoints to evaluate the finite difference approximation to the derivative. Similarly, h_t is the vector of constraints associated with timestep t, and H_t is the Jacobian of h_t with respect to q_t.

Just as in subsection II-A, we set δ_t be the incremental update to the trajectory in the vicinity of q_t, and perform gradient descent using Lagrange multipliers. Linearizing the Lagrangian produces

$$L_t(\delta_t, \lambda_t) = f_t + \nabla f_t^T \delta_t + \frac{1}{2\alpha}\delta_t^T \delta_t + \lambda_t^T [h_t + H_t \delta_t] \quad (7)$$

Similarly to Equations 4 through 6, we obtain the update rule

$$\delta_t = -\alpha \left(I - H_t^T (H_t H_t^T)^{-1} H_t \right) \nabla f_t - H_t^T (H_t H_t^T)^{-1} h_t \quad (8)$$

The δ_t terms are summed up and applied to ξ, producing a new trajectory that is smoother, but which also satisfies the constraints. Although the overall process is similar to that of subsection II-A, local smoothing is much faster due to the fact that it only needs to consider a single trajectory element (and a small, constant number of neighbors) at a time, and hence the runtime for a local smoothing update across the trajectory is $O(mn)$.

Our overall method, outlined in Algorithm 1 iterates constrained CHOMP, local smoothing, and upsampling. To determine whether either constrained CHOMP or local smoothing has converged, we look at the approximate relative error of the objective function, computed by

$$\varepsilon_{\text{approx}} = \frac{f(\xi) - f(\xi + \delta)}{f(\xi + \delta)}$$

Considering solely $f(\xi)$ and not $h(\xi)$ to determine convergence could, in theory, lead to problems if the trajectory is far from the constraint manifold, but we observed no such issues in practice. All constraints in our experiments were successfully met to nearly 10 decimal places by the time optimization had converged.

III. Implementation

For this project, all code was implemented in C++, with most matrix operations performed in OpenCV [10]. Our custom skyline Cholesky solver is built on top of the OpenCV matrix math library, and was found to be much faster than using the library's own general matrix solvers to invert A. Robot specifications were obtained from the OpenHUBO project [11].

Algorithm 1 Multigrid CHOMP with Local Smoothing

function MULTIGRIDCHOMP(ξ)
  while !CHOMPHasConverged(ξ) **do**
    $\xi \leftarrow \xi +$ CalculateCHOMPStep(ξ);
  end while
  while !LocalSmoothingHasConverged(ξ) **do**
    $\xi \leftarrow \xi +$ CalculateLocalSmoothingStep(ξ);
  end while
  if AtDesiredResolution() **then**
    return ξ;
  else
    $\xi_{up} \leftarrow$ UpSample(ξ);
    MultigridCHOMP(ξ_{up});
  end if
end function

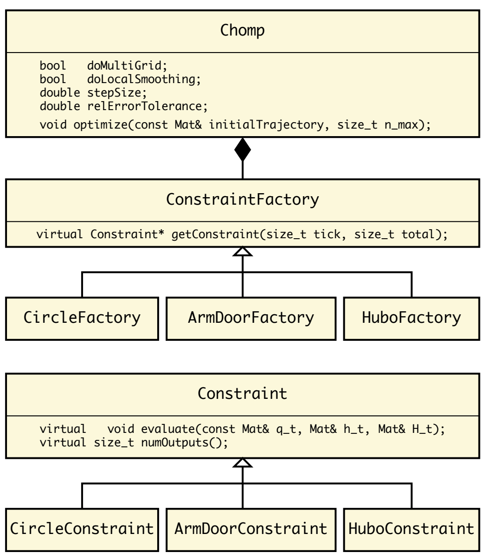

Fig. 2: UML class diagram indicating organization of our C++ code. The **`Chomp`** solver itself is generic and re-usable across a variety of problem instances, specified by subclassing the **`Constraint`** and **`ConstraintFactory`** classes.

To maximize reusability, we use the Abstract Factory design pattern [12] to encapsulate the problem-specific code and separate it from the underlying solver. The core of Multigrid CHOMP with Local Smoothing is implemented in the **`Chomp`** class, which is initialized with parameters of the optimization, as well as a reference to a **`ConstraintFactory`** object. The **`ConstraintFactory`** is responsible for generating individual **`Constraint`** objects for any given timestep of the trajectory, specific to the particular application. Each one evaluates the constraint function value $h_t(q_t)$, and the constraint Jacobian $H_t(q_t)$ for a particular timestep. When a low-resolution initial trajectory is given to the **`Chomp`** object, Multigrid CHOMP with Local Smoothing is run to find the optimal trajectory at a specified higher resolution. The architecture is diagrammed in Figure 2.

IV. EXPERIMENTS

We demonstrate the proposed method in two simple simulated scenarios, as well as on a physical robotic platform. The first problem, referred to as the "Circle problem" concerns a translating planar point that must snap to a circular path during some portion of the trajectory. Next, in the "Arm Door problem", we consider a planar 2D arm opening a door. Finally, we tackle the "Hubo problem" – opening a door with a physical HUBO+ robot.

Throughout this section, we normalize all objective function values as a fraction of the initial trajectory's objective function value. That is, given an initial trajectory ξ_{init} and a final trajectory ξ_{final}, we compute the quantity

$$\rho = \frac{f(\xi_{\text{final}})}{f(\xi_{\text{init}})}$$

In general, ρ will be less than one as long as the initial trajectory is feasible (obeys constraints); however, an infeasible initial trajectory may in fact be smoother than the optimal feasible one, especially at coarse resolutions. In presenting the results for each problem, we will also refer to the performance metric, defined as follows:

$$\text{Performance Metric} = (\rho_{\text{MCLS}} - \rho_{\text{CC}}) \cdot 100\%$$

where ρ_{MCLS} represents the normalized objective for Multigrid CHOMP with Local Smoothing and ρ_{CC} represents the normalized objective for constrained CHOMP. The metric can be considered as the percentage difference in objective function values between the two algorithms, relative to the initial trajectory (which is identical for both algorithms).

Data from our experiments are summarized in Table I, as well as Figures 5, 9, and 13. In the table and plots, "CC" refers to the original constrained CHOMP algorithm, "MC" refers to Multigrid CHOMP, and "MCLS" refers to Multigrid CHOMP with Local Smoothing.

A. Circle Problem

Our first example is a simple constraint problem for a translating point ($m = 2$), illustrated in Figure 3. The goal is to move from some start point in the plane, represented by the red dot at the upper left, to some end point in the plane, represented by the final blue dot dot. The point is constrained to be located on the circle for the middle half of the trajectory (see Figure 4 for a diagram).

For the finest resolution tested ($n = 511$), we found that that computation time was decreased by a factor of 8, with a performance metric of 6.6% (see Figure 5 for plots). For all of the values of n, it is clear that using Local Smoothing achieves a more optimal trajectory in terms of objective value function than Multigrid CHOMP alone, while adding only minimal extra computation time.

B. Arm Door Problem

The Arm Door Problem is a simplified version of the DRC scenario, using the planar arm and door illustrated in Figure 6. In both problems, we treat the combined robot and door DOF's as a single kinematic system. Matching the

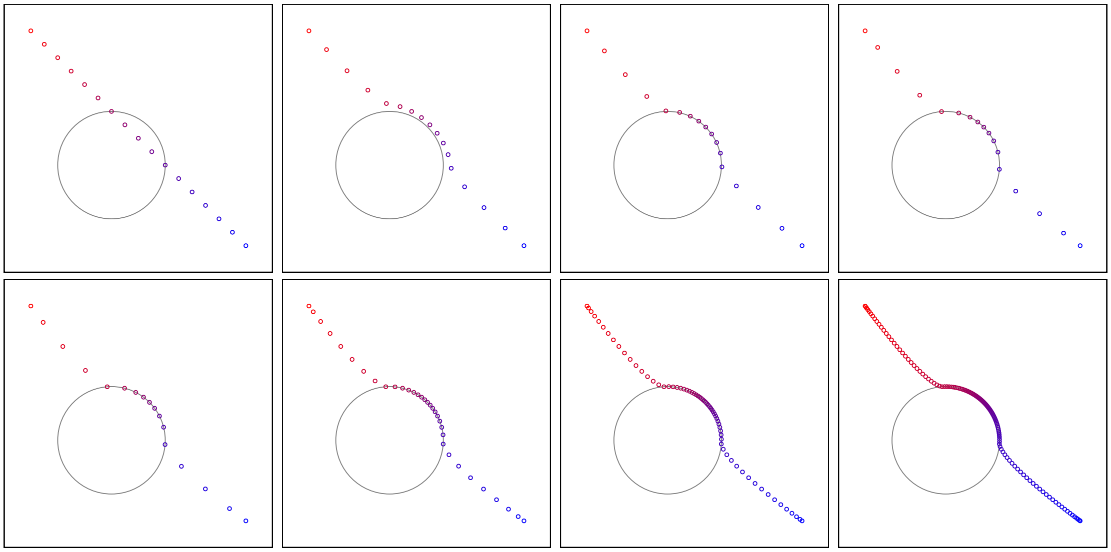

Fig. 3: Illustration of Circle problem. The translating point "robot" must transition from the top left to the bottom right of the area, while spending the middle 50% of the time on the circle. *Top row:* constrained CHOMP successfully warps the trajectory to minimize acceleration while obeying the constraints. *Bottom row:* Multigrid CHOMP with Local Smoothing upsamples to add intermediate points while continuing to refine the trajectory.

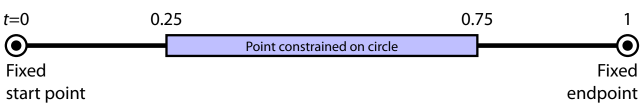

Fig. 4: Circle problem constraint diagram. The point is constrained to be on the circle during the middle 50% of the trajectory.

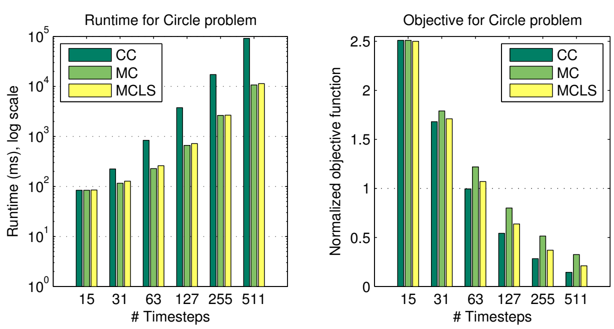

Fig. 5: Circle problem timing and objective function results.

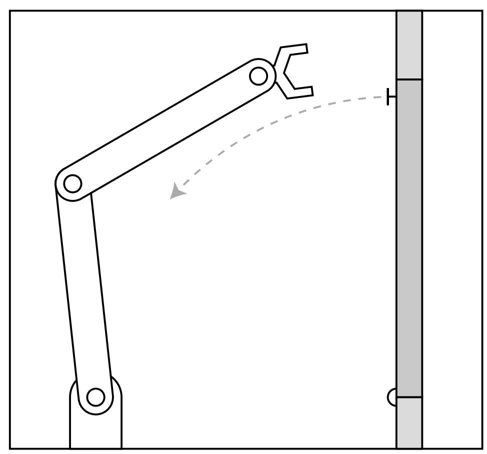

Fig. 6: Arm Door problem set-up. The planar arm has three degrees of freedom, with a shoulder joint, elbow joint, and wrist joint. The end effector and door handle will form a closed kinematic chain during opening.

end effector pose with the door handle therefore represents a closed kinematic chain constraint. An important effect of this problem formulation is that the trajectory for the door is modified, as well as that of the robot. For the Arm Door problem, the robot configuration is specified by three joint angles (shoulder, elbow and wrist), and the door configuration is specified by a single hinge angle ($m = 4$).

During first part of the trajectory, there is only one constraint; that the door is closed (hinge angle = 0°). Throughout the second half, the closed kinematic chain constraint is active. At the midpoint of the trajectory, the pose of the system is fully specified (door closed, hand on handle). See Figure 8 for an illustration.

The initial trajectory is generated in two phases: for the first half of the trajectory, the joint angles of the arm are linearly interpolated to the initial grasping position; in the second half, the door angle is linearly interpolated from closed to open, while IK is used to fix the end effector to the handle. Since the closed chain constraint leaves a single remaining degree of freedom, minimizing acceleration essentially amounts to re-timing the second half of the trajectory to match up the derivatives at the midpoint, improving the original C0 continuity. See Figure 7 for joint angles before and after optimization.

For the finest discretization ($n = 511$) of the Arm Door problem, we decrease computation time by a factor of 22.6, while achieving a performance metric 2.4% (see Figure 9). In this case, Local Smoothing boosts performance by roughly 50% versus Multigrid CHOMP alone.

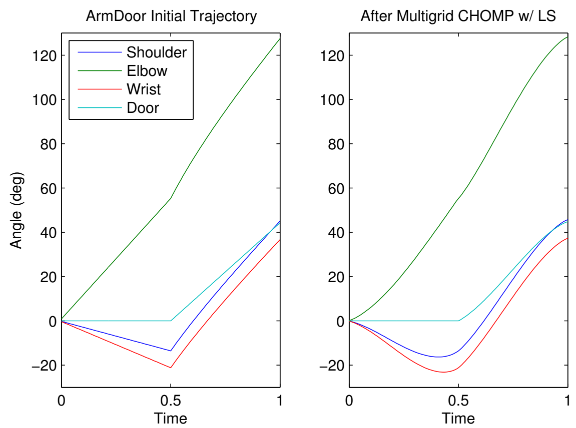

Fig. 7: Joint angles for Arm Door problem before (left) and after (right) optimization. Since the constraints on the second half of the trajectory leave only a single degree of freedom, minimizing accelerations largely corresponds to re-timing this portion of the trajectory.

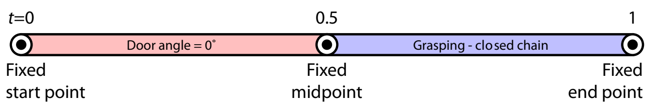

Fig. 8: Arm Door problem constraint diagram. During the first half of the trajectory, the door/handle is untouched, and during the second half of the trajectory the end effector pose must agree with the door handle. The configuration of the entire Arm Door system is fully specified at the midpoint.

C. Hubo Problem

In our final example, we apply Multigrid CHOMP with Local Smoothing to the DRC door opening problem. In this case, we use the six DOF of the HUBO+ arm, as well as the waist yaw joint. Additionally, we consider both the door hinge and handle angle, for a total of $m = 9$ DOF.

Figure 11 shows the constraint diagram. As shown in the figure, during the first part of the trajectory, the door is closed and nothing is touching the doorknob. Before the HUBO+ robot grabs the doorknob, the end effector must pass through a "pregrasp" pose. Although no constraints on the end effector are enforced immediately before or after pregrasp (as seen in Figures 11 and 12), this stage causes the end effector to approach the door from slightly above

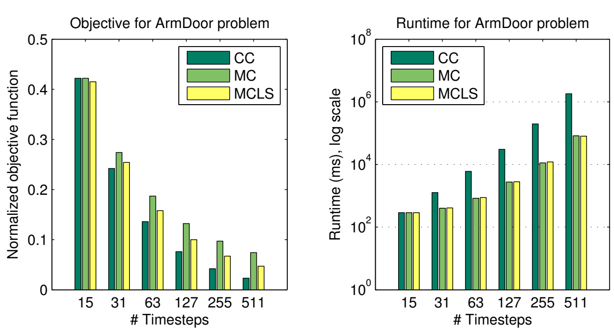

Fig. 9: Arm Door problem timing and objective function results.

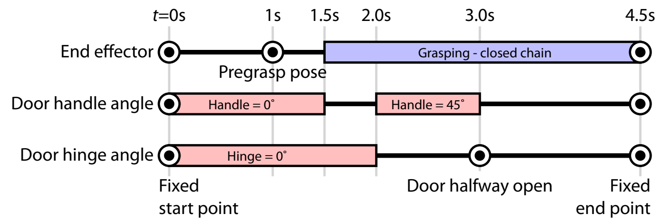

Fig. 11: Hubo problem constraint diagram. In this problem, we optimize jointly over seven degrees of freedom of the robot (six arm DOF plus waist rotation), as well as the door hinge and door handle, for a total of $m = 9$ DOF.

and away from the doorknob. This prevents the end effector from approaching an angle that could potentially cause the fingers to get caught on the doorknob and damage the robot.

The initial trajectory for the Hubo problem is generated by interpolating the arm joint angles to reach the pregrasp and grasp poses. Thereafter, IK is used to manipulate the door using linear ramps for door angle and handle angle, and the waist joint is ramped linearly at the end of the trajectory to augment the workspace of the arm when the doorway is fully opened. Hence, the initial trajectory illustrated in Figure 1 is only C0 continuous, with discontinuities in the derivative where various ramps are applied and trajectory generation methods are switched.

Unlike the previous two examples, the Hubo problem is executed on a physical robot (as shown in Figure 10), which necessitates consideration of actual execution time. The durations of the various trajectory phases depicted in Figure 11 are selected by hand to allow sufficient time for the door opening to take place without saturating torque or velocity limits. We target a control frequency of 100 Hz, resulting in a maximum of $n = 447$ trajectory elements for the roughly 4.5 second trajectory.

For the finest discretization, Multigrid CHOMP with Local Smoothing achieves a speedup of nearly 25x over constrained CHOMP, while reaching a performance metric of 2.1% (see Figure 13). As the graphs in Figure 1 show, there is a visible improvement in trajectory smoothness after optimization. Once again, Local Smoothing accounts for a 50% improvement over Multigrid CHOMP alone.

Due to constraints on planning time, we are currently targeting the 25 Hz ($n = 111$) resolution for our DRC efforts (pending further improvements to the algorithm, see section V). The results are still significant in this range, with a speedup of about 12.5x and performance metric of 1.1%.

D. Discussion

Our experiments demonstrate that Multigrid CHOMP with Local Smoothing retains nearly all of the optimality of the original constrained CHOMP method, while providing significantly faster performance. Runtime is reduced by a factor of 8.0, 22.6, and 24.8 on the Circle, Arm Door, and Hubo problems respectively.

The significant decrease in runtime can be attributed to two main factors. First, Multigrid CHOMP spends the bulk

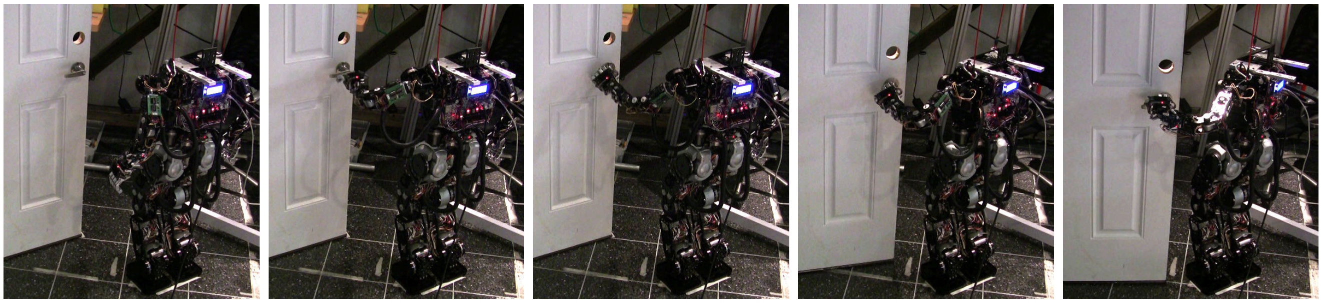

Fig. 10: Door opening trajectory successfully running on HUBO+ humanoid robot. Left to right: initial pose, pregrasp, grasping, door half open with handle turned, final position with handle restored to zero angle.

Problem		Circle ($m = 2$)						Arm Door ($m = 4$)						Hubo ($m = 9$)				
	n	15	31	63	127	255	511	15	31	63	127	255	511	27	55	111	223	447
Time (s)	CC	0.08	0.22	0.84	3.76	17.32	91.0	0.29	1.26	6.02	30.4	196.8	1,812	2.4	14.1	113.1	1,144	10,160
	MC	0.08	0.12	0.23	0.66	2.64	10.7	0.29	0.40	0.83	2.7	11.1	82.8	2.4	3.5	8.6	45.8	387
	MCLS	0.09	0.13	0.26	0.72	2.66	11.4	0.29	0.41	0.88	2.8	12.0	80.2	2.4	3.5	9.1	51.0	409
ρ	CC	2.51	1.68	1.00	0.54	0.28	0.15	0.42	0.24	0.14	0.08	0.04	0.02	0.35	0.22	0.12	0.07	0.04
	MC	2.51	1.79	1.22	0.80	0.51	0.33	0.42	0.27	0.19	0.13	0.10	0.07	0.35	0.24	0.17	0.12	0.09
	MCLS	2.50	1.71	1.07	0.64	0.37	0.21	0.42	0.25	0.16	0.10	0.07	0.05	0.34	0.22	0.14	0.09	0.06
Metric (%)		-1.0	3.0	7.5	9.5	8.6	6.6	-0.7	1.2	2.2	2.4	2.5	2.4	-0.6	0.3	1.4	1.9	2.1

TABLE I: Summary of results from all experiments. Note that MCLS performance metric is negative for base resolution of all cases because Local Smoothing is run after the initial optimization has nearly converged, resulting in a slight further improvement to the objective function.

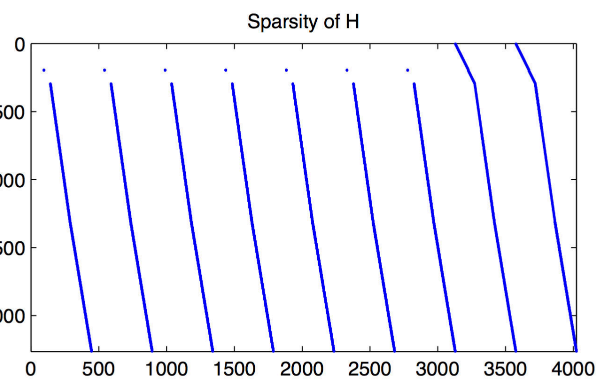

Fig. 12: Sparsity of Hubo problem constraint Jacobian H. The x-axis indexes both the $m = 9$ degrees of freedom (visible as distinct slanting traces) as well as the $n = 447$ timesteps, for a total of 4023 elements. The y-axis indexes the $k = 2264$ constraints (see Figure 11). For example, the dot-like "islands" in the upper left of the plot correspond to the pregrasp pose of the robot early in the trajectory, a constraint which is active for only a single timestep.

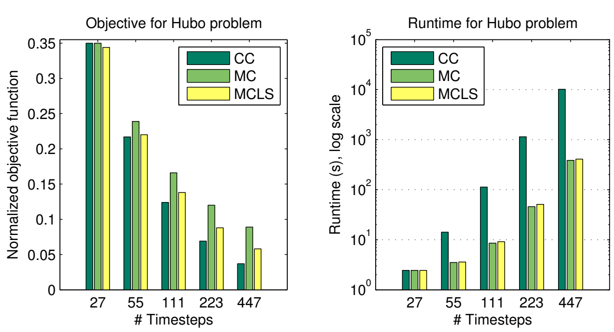

Fig. 13: Hubo problem timing and objective function results.

of its time optimizing only newly added points in upsampled trajectories, resulting in an 8x speedup over a full trajectory optimization. Second, after upsampling, the initial trajectory at the new resolution is much closer to optimal than the initial trajectory, resulting in fewer iterations needed at higher resolutions.

With Local Smoothing, for all three of the problems, the objective function is further reduced by roughly 50%. The runtime increase, on the other hand, is never over 10%. In fact, in one case (Arm Door problem, $n = 511$), we observe that Local Smoothing actually *improves* runtime slightly. We conjecture that this is a direct result of the second factor above.

As evident in Table I, for the base resolution in each problem, Multigrid CHOMP with Local Smoothing outperforms CHOMP in objective function minimization. This is a consequence of the implementation described in Algorithm 1. At each resolution, constrained CHOMP is performed until (approximate) convergence, after which Local Smoothing is performed. Since Local Smoothing is always applied at least once, this accounts for the slight reduction in objective function value.

When testing on the physical HUBO+ robotic platform, we were able to run the generated trajectories very repeatably to confirm their smoothness and effectiveness. In the initial trajectory, the waist joint (bold trace in Figure 1) is only used to open the door, but not used to reach the door handle. In the optimized trajectory, however, the waist joint is used to assist in reaching for the door handle in the first part of the trajectory. We believe this human-like, emergent behavior to be a valuable product of our algorithm.

V. Conclusions and Future Work

We have introduced Multigrid CHOMP with Local Smoothing, a multiresolution approach to speeding up the constrained CHOMP algorithm while preserving its strengths in trajectory optimization. We believe that this algorithm will prove useful in a number of real world behavior generation tasks, including the upcoming DARPA Robotics Challenge.

In future work, we plan to answer a number of questions about our approach. The choice of base condition in recursion awaits more study. A recursion deeper than needed could introduce extra runtime as well as more error. On the other hand, if recursion is not performed enough, runtime will not be significantly improved. Furthermore, our current implementation assumes a fixed start point and end point to each trajectory. Using a goal set [4] would allow Multigrid CHOMP with Local Smoothing to be applied to problems with flexible targets.

Although we are encouraged by the speedups on the Hubo problem, we would like even faster performance for the DRC. We suspect that more careful parameter tuning could contribute in that regard. Aside from the base resolutions mentioned above, we picked *ad-hoc* values for both the stepsize α, and the relative error tolerance for convergence of gradient descent. We believe better runtime and performance could be achieved with more careful parameter search, especially investigating whether these parameters should be changed after each upsample.

An efficient parallelized implementation of Equation 6 could also help speed up our approach. In recent years, multiple promising GPU-based linear algebra systems have appeared which could assist in solving these types of large-scale, dense linear systems [13], [14].

An open question, especially for the DRC, is how to integrate force feedback and/or compliant control to execute the trajectories produced by our algorithm. So far, we have applied the algorithm to a free-hanging door with no closer on it; however, in the future, we expect to need to make more careful consideration of forces and balancing while executing trajectories.

Finally, as mentioned in section II, none of our example problems consider collisions between the robot and its environment. However, prior to this work, the chief obstacle to doing large-scale constrained CHOMP was the prohibitive runtime, which scales at $O(k^3)$ with respect to the number of constraints k. In contrast, the obstacle avoidance aspects should scale linearly in the number of timesteps n, assuming an efficient collision response scheme like the signed distance field approach of [1].

Acknowledgments

The authors wish to thank the members of the Drexel Autonomous Systems Lab for their assistance in modeling and operating the HUBO+ robot, particularly Robert Ellenberg and Daniel Lofaro.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) award #N65236-12-1-1005 for the DARPA Robotics Challenge.

References

[1] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2009.

[2] M. Stilman, "Task constrained motion planning in robot joint space," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. IEEE, 2007.

[3] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. IEEE Int'l Conf. on Robotics and Automation*. IEEE, 2009.

[4] A. Dragan, N. Ratliff, and S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2011.

[5] S. Conte and C. de Boor, *Elementary Numerical Analysis*. McGraw-Hill, New York, 1972.

[6] A. George and J. W. H. Liu, *Computer solution of large sparse positive definite systems*. Prentice-Hall Inc., 1981.

[7] W. L. Briggs, S. F. McCormick *et al.*, *A multigrid tutorial*. Siam, 2000, vol. 72.

[8] J. McCann and N. S. Pollard, "Real-time gradient-domain painting," in *ACM Transactions on Graphics (SIGGRAPH)*, August 2008.

[9] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: a vector representation for smooth-shaded images," in *ACM Transactions on Graphics (TOG)*. ACM, 2008.

[10] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[11] Robert Ellenberg et al., "OpenHUBO," GitHub repository. [Online]. Available: https://github.com/daslrobotics/openHubo

[12] G. Erich, H. Richard, J. Ralph, and V. John, "Design patterns: elements of reusable object-oriented software," *Reading: Addison Wesley Publishing Company*, 1995.

[13] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *Proc. of IPDPS'10*. IEEE, 2010.

[14] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: hybrid GPU accelerated linear algebra routines," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2010.