International Conference on Humanoid Robotics (HUMANOIDS), 2015

Kinematically Constrained Workspace Control via Linear Optimization*

Zachary K. Kingston[1], Neil T. Dantam[1], and Lydia E. Kavraki[1]

***Abstract*— We present a method for Cartesian workspace control of a robot manipulator that enforces joint-level acceleration, velocity, and position constraints using linear optimization. This method is robust to kinematic singularities. On redundant manipulators, we avoid poor configurations near joint limits by including a maximum permissible velocity term to center each joint within its limits. Compared to the baseline Jacobian damped least-squares method of workspace control, this new approach honors kinematic limits, ensuring physically realizable control inputs and providing smoother motion of the robot. We demonstrate our method on simulated redundant and non-redundant manipulators and implement it on the physical 7-degree-of-freedom Baxter manipulator. We provide our control software under a permissive license.**

I. Introduction

Many manipulation tasks require direct motion in the robot's Cartesian workspace. Examples of such tasks include servoing an end-effector towards a target object, maintaining a workspace constraint such as the rotation axis of a valve, or tracking a trajectory though workspace waypoints. The conventional solution for direct workspace motion uses the manipulator Jacobian to map from desired workspace motions to joint-level commands for the robot. Jacobian inverse kinematics (IK) is efficient, effective, and widely implemented. However, Jacobian IK does not directly account for joint-level constraints on the kinematics of the arm, such as position limits, maximum achievable velocities, or desired acceleration limits for smooth motion. Additionally, it is common to execute lower-priority tasks by projecting the desired task velocity into the nullspace of the higher-priority manipulator Jacobian; if not treated carefully this can cause large accelerations at the start of motion. To address these challenges of classic Jacobian IK, we present a new workspace control method that uses linear programming to find an optimal, achievable solution to tracking a workspace reference while respecting the kinematic constraints of the robot.

The proposed *linearly-constrained Cartesian control* (LC^3) method uses linear programming to compute locally optimal, achievable workspace motions that satisfy joint-level constraints. The linear program computes the achievable, instantaneous workspace acceleration that will best track the desired velocity for the current time step (see section II). The linear constraints on the system are found in terms of the change in joint velocity, incorporating the Jacobian pseudo-inverse into the constraint matrix to map from workspace to joint-space values (see subsection II-B). In addition, a scaled nullspace velocity is also included. The solution to the linear program is a workspace acceleration which respects the kinematic constraints on the system and is singularity robust by way of using the damped pseudo-inverse of the Jacobian. We implement LC^3 and demonstrate results in simulation and on physical hardware (see section III). We provide our software under a permissive license. [2]

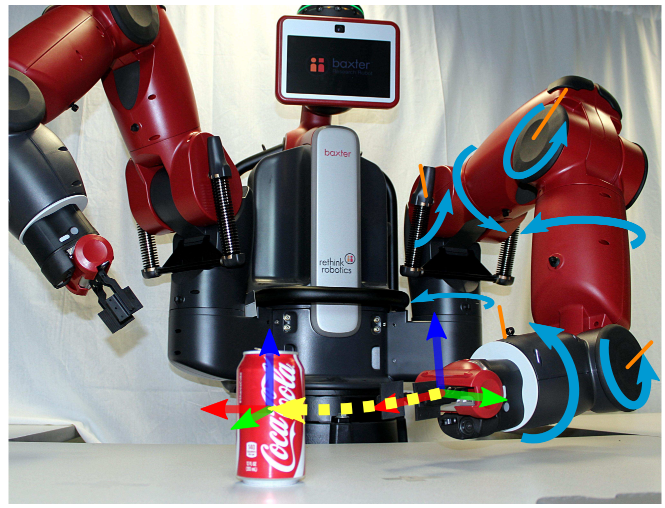


Fig. 1. Baxter executing workspace control via LC^3. The end-effector servos to a target frame in workspace (the yellow dotted line), with computed joint-velocities (blue arrows around joints) subject to kinematic constraints, e.g., position, velocity, and acceleration limits (orange bars over joints).

A. Related Work

Jacobian IK for robotics is a well studied approach [6], [21]. Jacobian least-squares methods compute to numerical precision the joint velocities to match a workspace reference. Singularity-robust Jacobian damped least squares methods operate reliably near singular configurations of the manipulator [18], [5]. Jacobian-based methods are also used to execute hierarchies of workspaces tasks [19]. Jacobian methods, however, do not directly account for joint-level constraints of the manipulator, e.g., acceleration limits, thus requiring additional considerations – or assumptions on initial state – to produce feasible motion of the manipulator. The LC^3 method proposed in this paper builds on Jacobian inverse

* This work has been supported in part by NSF IIS 1317849 and funds by Rice University.

[1] Z. K. Kingston, N. T. Dantam, and L. E. Kavraki are with the Department of Computer Science at Rice University, Houston TX 77005, USA. Email: {zkk1, ntd, kavraki}@rice.edu.

[2] Software available at http://github.com/golems/reflex

kinematics to directly address joint-level constraints through linear programming.

Position-based IK methods are distinct from Jacobian-based derivative methods, though one can solve position IK via gradient descent using the Jacobian. Analytical inverse kinematic solutions have been found for certain manipulator topologies, e.g., for anthropomorphic limbs [24]. Cyclic coordinate descent (CCD) effectively finds position IK solutions for arbitrary manipulators [25]. The FABRIK method by Aristidou and Lasenby uses a heuristic approach to find efficient IK solutions for chains of rotational joints [1]. Position IK methods find a joint-space *position* that achieves a desired workspace position. Respecting velocity and acceleration constraints while servoing to the computed joint-space position requires additional considerations or assumptions. In contrast, LC^3 finds a feasible joint-space *velocity* that best achieves a desired workspace velocity, directly accounting for joint-level constraints.

Quadratic optimization has also been applied to methods of robot control. Hierarchical quadratic programs can be used to satisfy multiple workspace tasks [9], [11]. Kim and Oh present a quadratic programming framework for position control that respects position, velocity, and acceleration limits [16]. Team MIT used sequential quadratic programs to find constrained IK solutions for an Atlas humanoid robot [10]. [12] uses quadratic programming to find optimal control of a teleoperated robot that respects kinematic constraints, workspace constraints, and task objectives. In contrast to these approaches which require solving quadratic programs, the presented LC^3 method requires solving only a linear program. The key difference between quadratic and linear approaches is the objective function for optimization. The quadratic program's objective function allows for minimization of Euclidean distance towards a task objective, while a linear program only permits minimization of the Manhattan distance. Our experimental results (see section III) show that the linear objective function minimizing Manhattan distance performs well in practice. Therefore, we use a linear program as it is computationally more efficient. The objective function for LC^3 is discussed in subsection II-A.

A related approach is to apply task constraints while planning a joint-space path [22], [23], [17], [14], [2]. We are considering a different problem: directly tracking a workspace reference. By tracking a workspace reference, tracking errors can be corrected directly in the workspace of the robot.

There are numerous approaches for computing robot workspace trajectories [6], [20], [7], [15], [8]. All of these workspace trajectory methods depend on some underlying inverse kinematics approach to find the joint-level robot inputs. The LC^3 method presented here is one such approach.

B. Model and Notation

The robot is modeled as an n-degree-of-freedom (DOF) manipulator operating in a workspace of dimension m. We represent a manipulator joint configuration with variable q. Workspace positions are designated with variable X. Time derivatives of some variable y are represented as $\dot{y}$, and double time derivatives as $\ddot{y}$.

We denote actual, reference, and constraint variables with the following subscripts. Actual joint configurations observed on the robot are designated with subscript a. We assume that that actual joint position q_a and actual joint velocity $\dot{q}_a$ are observable. Desired reference values of a trajectory are given by subscript r. For a workspace motion, we have the desired workspace position X_r and desired workspace velocity $\dot{X}_r$. Control commands are designated by subscript u, e.g., the controlled joint velocity $\dot{q}_u$. Constraints on the system are designated by bounding constants with subscript $\min$ and $\max$ for the minimum and maximum values on the system respectively.

The manipulator Jacobian matrix J and its pseudo-inverse J^* relate joint-space and workspace derivatives:

$$\dot{X} = J\dot{q}$$
$$\dot{q} = J^*\dot{X}$$

The damped Jacobian pseudo-inverse J^+ produces acceptable motion near kinematic singularities, avoiding large velocities produced by the undamped pseudo-inverse J^*. The damped pseudo-inverse can be computed as follows [18].

$$J^+ = J^T(JJ^T + \lambda^2 I)^{-1}$$

where λ is the damping constant and I is the identity matrix.

We can also use the singular value decomposition (SVD) of the Jacobian J to damp the solution only near singularities, avoiding error from damping when far from singularities:

$$J = USV^T$$
$$J^+ = \sum_{i=1}^{\min(m,n)} \frac{s_i}{\max\left({s_i}^2, {s_\epsilon}^2\right)} v_i {u_i}^T$$

where s_ϵ is a threshold on the singular values below which we introduce damping.

The nullspace of the manipulator Jacobian represents joint velocities which will not change the end-effector velocities. We project a desired joint space velocity into the Jacobian nullspace using projection matrix N:

$$N = I - J^+J$$
$$\dot{q}_n = N\dot{q}_r$$

where $\dot{q}_r$ is a desired joint velocity and $\dot{q}_n$ is the projection into the Jacobian nullspace.

In our case, we project velocities to move joints away from position limits. This velocity is designated by $\dot{q}_{rn}$ (see subsection II-B).

The standard form of a Linear Program (LP) is:

$$\begin{aligned} \textbf{Maximize} \quad & c^T x \\ \textbf{Subject to} \quad & Ax \leq b \\ \textbf{and} \quad & x > 0 \end{aligned} \tag{1}$$

where x is the vector of optimization variables, A and b encode the linear constraints, and c encodes the linear objective function. LC^3 in standard LP form is discussed in subsection II-D.

II. Method

Our proposed *linearly-constrained Cartesian control* (LC[3]) uses linear optimization to compute the joint-space commands that best track a workspace reference, subject to position, acceleration, and velocity constrains. LC[3] is given actual configuration q_a and velocity $\dot{q}_a$, manipulator Jacobian J, desired workspace velocity $\dot{X}_r$, and desired joint velocity $\dot{q}_{rn}$ (for nullspace projection). The robot motion is constrained based on joint-space limits in position $q_{\min}$ and $q_{\max}$, velocity $\dot{q}_{\min}$ and $\dot{q}_{\max}$, and acceleration $\ddot{q}_{\min}$ and $\ddot{q}_{max}$. The output of LC[3] is a feasible joint-space velocity command $\dot{q}_u$ that is within the specified constraints and that optimally matches $\dot{X}_r$ and $\dot{q}_{rn}$. We assume that the robot provides joint-level velocity control.

A. Objective Function

The optimization variables for LC[3] are achievable workspace accelerations $\ddot{X}_u$ and a nullspace projection gain k. We optimize over workspace accelerations to enable feedback correction of position and velocity errors directly in workspace. By optimizing workspace acceleration $\ddot{X}_u$, we can define a linear objective function that most closely matches $\ddot{X}_u$ with the necessary acceleration to achieve the desired velocity $\dot{X}_r$. The computed optimization variable $\ddot{X}_u$ will accelerate as closely as possible to the desired velocity $\dot{X}_r$ for the current timestep. We address the signs of optimization variables in subsection II-C via a sign transformation.

We compute a desired workspace acceleration $\ddot{X}_r$ based on the desired velocity vector in workspace $\dot{X}_r$, and the robot's actual velocity $\dot{q}_a$:

$$\dot{X}_a = J\dot{q}_a \qquad\qquad \ddot{X}_r = \frac{\dot{X}_a - \dot{X}_r}{\Delta t}$$

where $\dot{X}_a$ is the actual workspace velocity and Δt is the control time step.

The linear objective function to find the optimal acceleration $\ddot{X}_u$ is:

$$\ddot{X}_r \cdot \ddot{X}_u + C_u k \tag{2}$$

We use the dot product between the desired and controlled accelerations, $\ddot{X}_r \cdot \ddot{X}_u$, because it relates to the directions of these vectors. If the acceleration vectors are orthogonal, then their dot product is 0. As the two vectors point more closely in the same direction, their dot product increases. The difference between the quadratic cost and linear reward functions in QP and LP is illustrated in Figure 2. The dot product between the desired and computed vectors closely matches the result of quadratic cost objective function.

The term $C_u k$ relates to the velocity $\dot{q}_{rn}$ that will be projected into the nullspace of the Jacobian, e.g., in order to move the arm away from joint limits. The objective function coefficient C_u is a parameter to LC[3] indicating the relative importance of matching the desired acceleration $\ddot{X}_r$ compared to applying the nullspace projection of $\ddot{q}_{rn}$. The optimization variable k is the actual gain for the nullspace projection of $\dot{q}_{rn}$. We additionally limit k to a maximum value with the constraint $k \leq k_{\max}$.

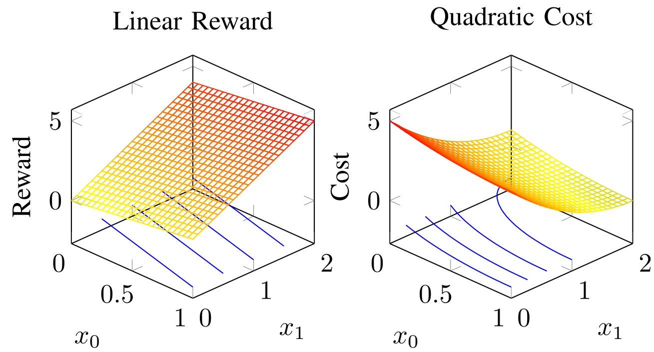


Fig. 2. Comparison of linear and quadratic objective functions. The linear reward function is the dot product of the desired and computed vectors while the quadratic cost is the sum of squared error between the two vectors. The blue contour lines on the bottom plane show equal reward / cost regions of the functions. When the solution is unconstrained, both objective functions are return the optimal value, the reference. When the solution is constrained, the quadratic cost function may produce a vector that better matches the direction of the optimal solution; however, solutions for both cases are close.

For some robot states $(q, \dot{q})$ and desired workspace velocities $\dot{X}_r$, the desired workspace acceleration $\ddot{X}_r$ may not be possible to achieve under the given kinematic constraints. For these cases, the result of the LP will include an optimal workspace acceleration $\ddot{X}_u$ that respects the constraints of the robot.

B. Derivation of Constraints

We derive linear constraints for position, velocity, and acceleration limits of the manipulator. The constraints, as with any LP, are a set of inequalities (1) based on the optimization variables, which for our problem are commanded acceleration $\ddot{X}_u$ and nullspace projection gain k.

We can express all kinematic limits in terms of commanded *change* in velocity at each time step, $\Delta\dot{q}_u$. We compute $\Delta\dot{q}_u$ from commanded acceleration $\ddot{X}_u$ and nullspace projection gain k:

$$\Delta\dot{q}_u = J^{+}\Delta t\ddot{X}_u + kN\Delta\dot{q}_{rn} \tag{3}$$

Equation (3) uses the Jacobian damped pseudoinverse J^{+} to transform a workspace velocity into a jointspace velocity vector so that we can apply joint-space constraints. The controlled acceleration, $\ddot{X}_u$, is multiplied by the current change in time Δt to get an instantaneous velocity change, i.e., an Euler integration step, resulting in $\Delta\dot{q}_u$. The resultant velocity change is singularity robust because it uses the Jacobian damped pseudoinverse J^{+}.

The last term in (3) is the projection of the change of the desired joint velocity $\Delta\dot{q}_{rn}$ into the nullspace of the manipulator Jacobian N. It is scaled by the optimization variable k. We calculate the change in desired joint velocity $\Delta\dot{q}_{rn}$ as the difference between the actual and desired joint velocities:

$$\Delta\dot{q}_{rn} = \dot{q}_{rn} - \dot{q}_a$$

Now, we derive the corresponding commanded joint velocities $\dot{q}_u$ and accelerations $\ddot{q}_u$.

From (3), the commanded joint velocity $\dot{q}_u$ is:

$$\dot{q}_u = \dot{q}_a + \Delta\dot{q}_u \tag{4}$$

Then, the corresponding commanded acceleration computed via finite difference is:

$$\ddot{q}_u = \frac{\dot{q}_u - \dot{q}_a}{\Delta t} = \frac{\Delta\dot{q}_u}{\Delta t} \tag{5}$$

Next, we write the constraints in terms of these commanded joint-space values.

1) Acceleration Constraints: Joint accelerations are limited by minimum $\ddot{q}_{\min}$ and maximum $\ddot{q}_{\max}$:

$$\ddot{q}_{\min} \leq \ddot{q}_u \leq \ddot{q}_{\max}$$

Replacing $\ddot{q}_u$ with the result from (5) and rearranging terms, we write the inequality in terms of $\Delta\dot{q}_u$ from (3):

$$\ddot{q}_{\min}\Delta t \leq \Delta\dot{q}_u \leq \ddot{q}_{\max}\Delta t \tag{6}$$

2) Velocity Constraints: Joint velocities are limited by minimum $\dot{q}_{\min}$ and maximum $\dot{q}_{\max}$:

$$\dot{q}_{\min} \leq \dot{q}_u \leq \dot{q}_{\max}$$

Again, we rearrange terms to write the inequality in terms of $\Delta\dot{q}_u$:

$$\dot{q}_{\min} - \dot{q}_a \leq \Delta\dot{q}_u \leq \dot{q}_{\max} - \dot{q}_a \tag{7}$$

3) Position Constraints: We derive the position constraint based on distance traveled, x, for an initial velocity $\dot{x}_0$ and constant acceleration $\ddot{x}$ during time step Δt:

$$x = x_0 + \dot{x}_0\Delta t + \frac{1}{2}\ddot{x}\Delta t^2 \tag{8}$$

The manipulator's joint positions are limited by minimum $q_{\min}$ and maximum $q_{\max}$. Given the current joint position q_a, the controlled joint velocity $\dot{q}_u$, and acceleration limits $\ddot{q}_{\{\max,\min\}}$, we use (8) to find minimum and maximum achievable positions, $q_{n_{\min}}$ and $q_{n_{\max}}$, of the joint for the current control cycle:

$$q_{n_{\max}} = q_a + \dot{q}_a\Delta t + \frac{1}{2}\ddot{q}_{\max}\Delta t^2$$

$$q_{n_{\min}} = q_a + \dot{q}_a\Delta t + \frac{1}{2}\ddot{q}_{\min}\Delta t^2$$

If these exceed the respective bounding limits $q_{\min}$ or $q_{\max}$, then the controlled velocity $\dot{q}_u$ is invalid. This gives the bounds on the position:

$$q_{\min} \leq q_{n_{\max}} \qquad q_{n_{\min}} \leq q_{\max}$$

Rearranging terms, we state the position limit in terms of $\Delta\dot{q}_u$:

$$\frac{q_{\min} - q_a}{\Delta t} - \frac{\ddot{q}_{\max}\Delta t}{2} - \dot{q}_a \leq \Delta\dot{q}_u \leq \frac{q_{\max} - q_a}{\Delta t} - \frac{\ddot{q}_{\min}\Delta t}{2} - \dot{q}_a \tag{9}$$

4) Simplified Combined Constraints: By stating all constraints in terms of $\Delta\dot{q}_u$, we can combine these into a single inequality, reducing the number of constraints by a factor of three. We find LP bounds from the most-constrained values in inequalities (6), (7), and (9).

$$c_{\min} = \mathbf{max}\begin{cases} \dfrac{q_{\min} - q_a}{\Delta t} - \dfrac{\ddot{q}_{\max}\Delta t}{2} - \dot{q}_a \\ \dot{q}_{\min} - \dot{q}_a \\ \ddot{q}_{\min}\Delta t \end{cases} \tag{10}$$

$$c_{\max} = \mathbf{min}\begin{cases} \dfrac{q_{\max} - q_a}{\Delta t} - \dfrac{\ddot{q}_{\min}\Delta t}{2} - \dot{q}_a \\ \dot{q}_{\max} - \dot{q}_a \\ \ddot{q}_{\max}\Delta t \end{cases} \tag{11}$$

Then, the resulting LP constraint is:

$$c_{\min} \leq \Delta\dot{q}_u \leq c_{\max} \tag{12}$$

C. Sign Transformation of Optimization Variables

The optimization variables in standard LP form must be positive. However, cases where the robot's controlled velocity cannot match the sign of the desired velocity, e.g., due to a large initial velocity in the opposing direction, must also be handled. This restriction is relaxed as we are operating over accelerations in workspace, and the sign of the desired acceleration must be matched. To account for this, we create sign transformation matrix M to convert the optimization variables to positive values. M is a diagonal matrix whose entries are the sign of the desired workspace acceleration $\ddot{X}_r$:

$$M_{ij} = \begin{cases} 0 & i \neq j \\ \text{sign}(\ddot{X}_r) & i = j \end{cases}$$

In the case of the workspace reference acceleration $\ddot{X}_r$ having 0 for an entry, the 0 is treated as a positive variable so M is always invertible.

We apply M to matrices J^+ and $\ddot{X}_u$ to ensure positive optimization variables:

$$\widetilde{J}^+ = J^+M^{-1} \qquad \widetilde{\ddot{X}}_u = M\ddot{X}_u \qquad \widetilde{\ddot{X}}_r = M\ddot{X}_r$$

As $M^{-1}M$ equals the identity matrix, the result of multiplication is preserved:

$$\widetilde{J}^+\widetilde{\ddot{X}}_u = J^+M^{-1}M\ddot{X}_u = J^+\ddot{X}_u$$

Consequently, the LP objective function from Equation 2 is:

$$\widetilde{\ddot{X}}_r \cdot \widetilde{\ddot{X}}_u + C_u k \tag{13}$$

Then, Equation 3 becomes:

$$\Delta\dot{q}_u = \widetilde{J}^+\Delta t\widetilde{\ddot{X}}_u + N\Delta\dot{q}_{rn}k \tag{14}$$

Using the sign transformation matrix M ensures that optimization variable vector $[\widetilde{\ddot{X}}_u, k]$ will be positive. We can then recover the actual value for commanded acceleration via M^{-1}:

$$\ddot{X}_u = M^{-1}\widetilde{\ddot{X}}_u \tag{15}$$

D. Standard LP Form

We now represent LC^3 in the standard LP form given in (1). Writing our problem in this canonical form enables use of efficient algorithms [4] and solvers [3].

The objective function for optimization is:

$$\widetilde{\ddot{X}}_r \cdot \widetilde{\ddot{X}}_u + C_u k$$

The constraints are:

$$c_{\min} \leq \widetilde{J}^{+} \Delta t \widetilde{\ddot{X}}_u + N \Delta \dot{q}_{rn} k \leq c_{\max}$$

$$k \leq k_{\max} \qquad \widetilde{\ddot{X}}_u \leq \widetilde{\ddot{X}}_r$$

The manipulator and LP values are of the following dimensions:

- Joint-space vector $q \in \mathbb{R}^n$
- Workspace vector $X \in \mathbb{R}^m$
- LP optimization variable vector $x \in \mathbb{R}^{m+1}$
- LP objective function vector $c \in \mathbb{R}^{m+1}$
- LP bounding vector $b \in \mathbb{R}^{2n+m+1}$
- LP constraint matrix $A \in \mathbb{R}^{(2n+m+1)\times(m+1)}$

The LP vectors x, b, and c in Equation 1 are:

$$x = \begin{bmatrix} \widetilde{\ddot{X}}_u \\ k \end{bmatrix} \qquad c = \begin{bmatrix} \widetilde{\ddot{X}}_r \\ C_u \end{bmatrix} \qquad b = \begin{bmatrix} c_{\max} \\ -c_{\min} \\ k_{\max} \\ \widetilde{\ddot{X}}_r \end{bmatrix}$$

The LP constraint matrix A is:

$$A = \begin{bmatrix} \widetilde{J}^{+}\Delta t & N\Delta\dot{q}_{rn} \\ -\widetilde{J}^{+}\Delta t & -N\Delta\dot{q}_{rn} \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

All together, the LP is as follows:

$$c^T x = \begin{bmatrix} \widetilde{\ddot{X}}_r & C_u \end{bmatrix} \begin{bmatrix} \widetilde{\ddot{X}}_u \\ k \end{bmatrix} = \widetilde{\ddot{X}}_r \cdot \widetilde{\ddot{X}}_u + C_u k$$

$$Ax \leq b \rightarrow \begin{bmatrix} \widetilde{J}^{+}\Delta t & N\Delta\dot{q}_{rn} \\ -\widetilde{J}^{+}\Delta t & -N\Delta\dot{q}_{rn} \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \widetilde{\ddot{X}}_u \\ k \end{bmatrix} \leq \begin{bmatrix} c_{\max} \\ -c_{\min} \\ k_{\max} \\ \widetilde{\ddot{X}}_r \end{bmatrix}$$

III. Experimental Results

We implement and demonstrate LC^3 on simulated and physical robot manipulators. Our implementation of LC^3 is written in C using *openBLAS* [13], [26] for linear algebra and *lp_solve* [3] for linear programming. We test our software on an Intel® i7-4790 CPU under Linux 3.18.16-rt13+ PREEMPT RT. Figure 3 summarizes the computational performance of LC^3 compared to the baseline Jacobian damped least-squares methods (DLS).

We compare a baseline Jacobian DLS with LC^3 by servoing the manipulator to a desired workspace position. The desired workspace velocity is the logarithm of error between actual pose ${}^{B}\mathcal{S}_e$ and desired pose ${}^{B}\mathcal{S}_r$:

$$\dot{X}_r = \begin{bmatrix} \omega_r \\ \dot{v}_r \end{bmatrix} = \ln\left({}^{B}\mathcal{S}_e{}^{*} \otimes {}^{B}\mathcal{S}_r\right) \tag{16}$$

Method	Average Time	Normalized Value
Jacobian DLS LU	0.0094 ms	1.0000
Jacobian DLS SVD	0.0302 ms	3.1972
LC^3 LU	0.1785 ms	18.8980
LC^3 SVD	0.2034 ms	21.5287

Fig. 3. Computation time results for LC^3 compared to baseline Jacobian Damped Least Squares (DLS) for a 7 DOF arm. We compared the Jacobian DLS and LC^3 using both the LU decomposition and the SVD to compute the Jacobian damped pseudo-inverse. The CPU was an Intel i7-4790. All methods are fast enough to operate at typical control rates of 1 kHz with LC^3 taking about 20 times as long as the baseline due to solving the linear program program in addition to computing the damped pseudo-inverse.

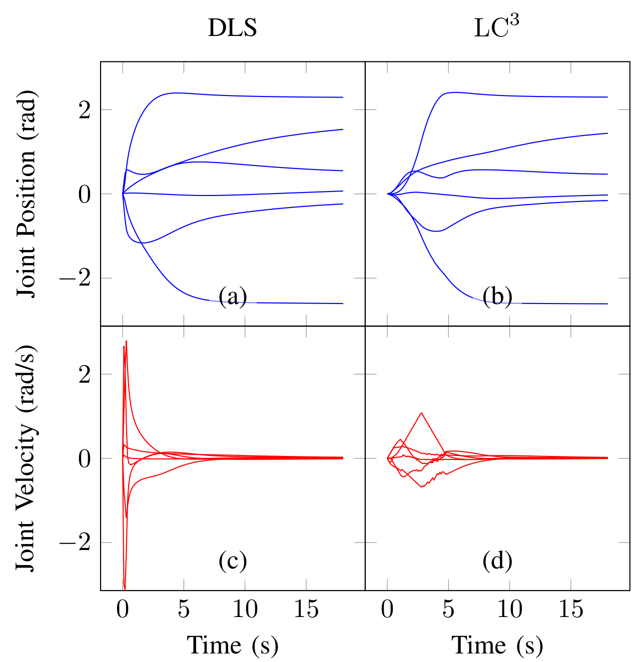
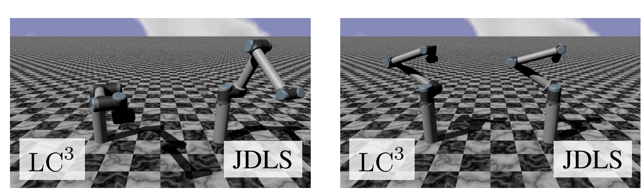


Fig. 4. Simulation on the UR10 robot, maneuvering its end-effector to a point. The beginning joint configuration has all of the robot's joints at their zeroed position. The top row shows the joint positions q_a, and the bottom row shows the controlled / actual joint velocities $\dot{q}_u$. Each line represents one joint on the manipulator. The baseline DLS has a large velocity spike at the beginning of execution (c), while LC^{3}has a gradual increase, respecting the acceleration limits on the system (d). Both converge to the desired position in approximately the same time.



(a) (b)

Fig. 5. Images of simulated UR10 robot. The arm on the left is using LC^3, and the arm on the right is using Jacobian DLS (JDLS). (a) shows the larger initial displacement of the right manipulator due to the large initial velocity spike of the Jacobian DLS. (b) shows how both LC^3 and Jacobian DLS converge to approximately the same configuration at the same time.

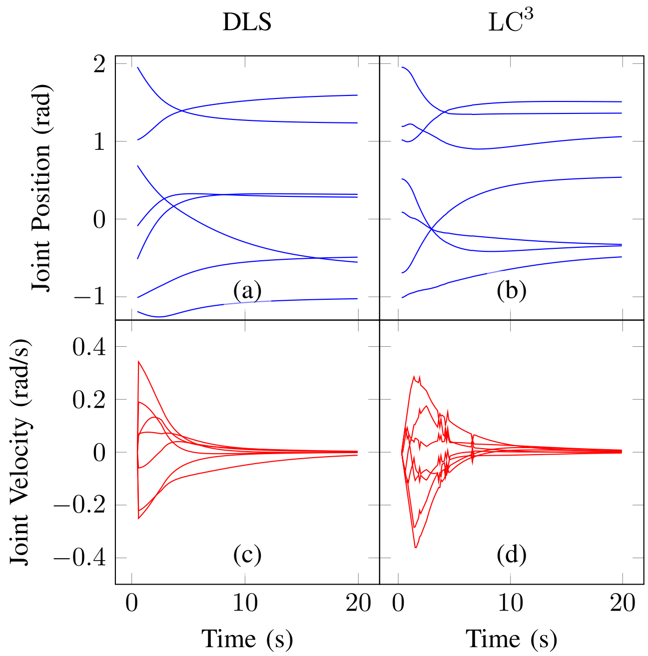


Fig. 6. Simulation on the Baxter robot, servoing its right and left end-effectors to mirrored points. The beginning joint configuration is the joint positions of the Baxter robot when its arms have been untucked. The format of the plots is the same as Figure 4. Similar profiles as the other simulation can be seen. As the Baxter's manipulators are redundant, the velocity $\dot{q}_{rn}$ is projected into the nullspace of the Jacobian.

(a) (b)

Fig. 7. Images from the experiment on the Baxter robot. The robot's right arm is using LC^3, and its left arm is using Jacobian DLS (JDLS). (a) is from the beginning of the experiment, showing the larger initial displacement of the robot's left manipulator due to the large initial velocity spike of the Jacobian DLS on the right-hand robot. (b) shows how both LC^3 and Jacobian DLS converge to approximately the same configuration at the same time.

where ω_r is reference rotational velocity, $\dot{v}_r$ is reference translational velocity, ${}^{B}\mathcal{S}_e$ is the actual end-effector pose dual quaternion, ${}^{B}\mathcal{S}_r$ is the reference pose dual quaternion, and $\otimes$ is the quaternion multiplication operator.

We use kinematic redundancy to center the joints within the middle of their position ranges, thereby avoiding poor manipulator configurations which may prevent movement. We compute the nullspace projection velocity $\dot{q}_{rn}$ to center each joint as.

$$(\dot{q}_{rn})_i = \frac{(q_c)_i - (q_a)_i}{(q_{\max})_i - (q_{\min})_i} \tag{17}$$

where $(\dot{q}_{rn})_i$ is the i^{th} element of the nullspace projection velocity, $(q_c)_i$ is the i^{th} joint center position, $(q_a)_i$ is the i^{th} actual velocity, and $(q_{\max})_i$ and $(q_{\min})_i$ are the i^{th} joint position limits.

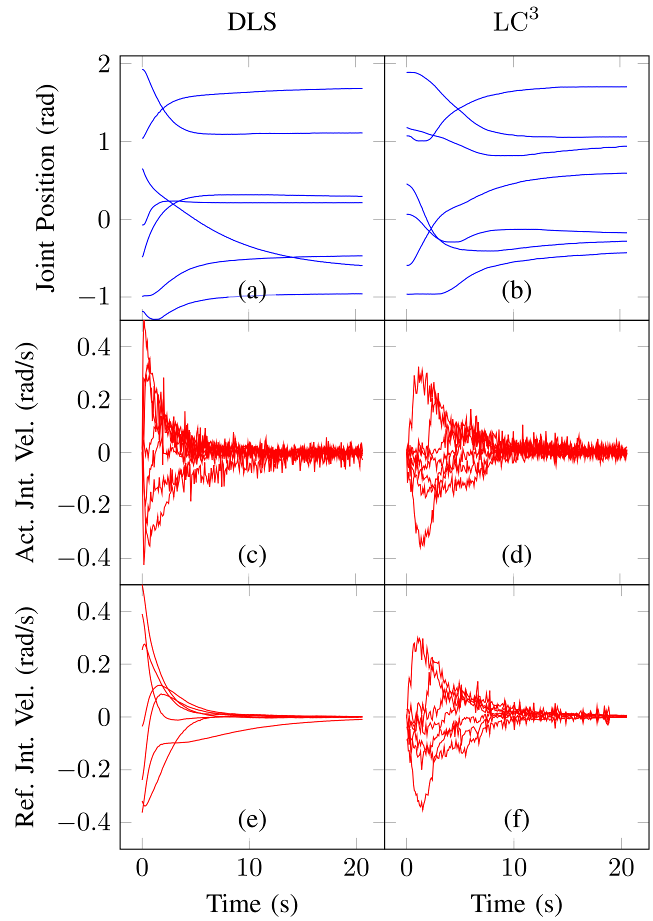


Fig. 8. Experimental results on the physical Baxter robot, performing the same experiment as in Figure 6. Top row shows actual joint positions q_a, middle row shows actual joint velocities $\dot{q}_a$, and bottom row shows controlled velocity $\dot{q}_u$. As in the simulation, the baseline DLS produces a large spike in velocity while LC^3 provides a smooth ramp. Compared to simulation, the physical results have significant noise due to joint compliance and backlash of the Baxter. Note that commanded velocity $\dot{q}_u$ for LC^{3}also includes this noise since actual velocity is included in its feedback whereas the baseline method uses only feedforward velocity.

A. Simulation Results

We present kinematic simulation results on the Universal Robotics *UR10* and Rethink Robotics *Baxter* robots. The UR10 is a 6-DOF non-redundant manipulator, while the arms of the Baxter are 7-DOF redundant, anthropomorphic limbs. We simulate servoing to a point in workspace from a nominal starting configuration, comparing LC^3 with the baseline Jacobian damped least-squares approach. The positions and velocities of the UR10 robot are plotted in Figure 4 and those of the Baxter in Figure 6. There is a dramatic difference in the starting velocity ramp of each joint, with the baseline Jacobian DLS generating a large initial velocity spike, and LC^3 providing a limited velocity ramp bounded by the maximum possible acceleration limits provided. Despite the difference in initial velocity, both LC^3 and the baseline converge to the final joint positions at approximately the same time. These results show that LC^3 provides a smooth velocity ramp for redundant and non-redundant manipulators.

B. Physical Robot Results

We demonstrate LC^3 on a physical Baxter robot. As in the simulation, we compare LC^3 and the baseline Jacobian DLS for servoing to a workspace position. The actual joint positions and velocities q_a and $\dot{q}_a$ along with the commanded velocities $\dot{q}_u$ are shown in Figure 8. Note the difference between the commanded and actual velocities, arising due to physical limits of the manipulator. Just as in the simulation case, LC^3 provides reduced acceleration and velocity requirements while converging in similar time compared to the baseline.

IV. Discussion and Conclusion

We presented a new Cartesian workspace controller, linearly-constrained Cartesian control (LC^3). This new method respects the position, velocity, and acceleration constraints in the manipulator's joint-space. LC^3 is singularity-robust, provides smoother and more gradual motions of the manipulator, and in the case of redundant manipulators, maneuvers the arm away from poor configurations near joint limits that reduce maneuverability. We demonstrated this controller in simulation and on a physical manipulator.

There are several tuneable parameters within LC^3. These include the damping constant, λ or s_ϵ, for the Jacobian damped pseudo-inverse J^+, the weighting constant of the nullspace projection in optimization C_u, the maximum possible value of the nullspace projection gain $k_{\max}$, and the constraints upon the joints themselves. These all can affect the performance of the controller in terms of achievability and accuracy. Furthermore, it is possible to modify the weighting of joints in the nullspace projection velocity to change the resulting motion. For example, by more-heavily weighting joints with smaller range of motion, the projection favors centering these joints instead of those with greater range of motion. Generally this helps the manipulator remain in configurations with greater reachability.

A key advantage of LC^3 is increased robustness to initial state compared to the baseline Jacobian damped least-squares (DLS). The DLS may produce large accelerations from some initial configurations or require additional velocity ramping to produce smooth motion, and it does not consider initial velocity. In contrast, LC^3 will produce acceleration-limited motion regardless of initial configurations or velocities.

References

[1] Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243 – 260, 2011.

[2] Dmitry Berenson, Siddhartha Srinivasa, Dave Ferguson, and James Kuffner. Manipulation planning on constraint manifolds. In *Intl. Conf. on Robotics and Automation*, pages 625–632. IEEE, 2009.

[3] Michel Berkelaar, Kjell Eikland, Peter Notebaert, et al. lpsolve: Open source (mixed-integer) linear programming system, 2004. http://lpsolve.sourceforge.net/5.5/.

[4] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[5] Samuel Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics, GPU, and Game Tools*, 10(3):37–49, 2005.

[6] John Craig. *Introduction to Robotics: Mechanics and Control*. Pearson, 3rd edition, 2005.

[7] Erik Dam, Martin Koch, and Martin Lillholm. *Quaternions, Interpolation and Animation*. Datalogisk Institut, Københavns Universitet, 1998.

[8] Neil Dantam and Mike Stilman. Spherical parabolic blends for robot workspace trajectories (presented). In *International Conference on Intelligent Robots and Systems*. IEEE, 2014.

[9] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, June 2014.

[10] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai, Claudia Pérez D'Arpino, Robin Deits, Matt DiCicco, Dehann Fourie, et al. An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254, 2015.

[11] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher Atkeson. Optimization-based full body control for the darpa robotics challenge. *Journal of Field Robotics*, 32(2):293–312, 2015.

[12] Janez Funda, Russell Taylor, Benjamin Eldridge, Stephen Gomory, and Kreg Gruben. Constrained cartesian motion control for teleoperated surgical robots. *Robotics and Automation, IEEE Transactions on*, 12(3):453–465, 1996.

[13] Kazushige Goto and Robert Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Transactions on Mathematical Software (TOMS)*, 35(1):4, 2008.

[14] Kris Hauser. Fast interpolation and time-optimization on implicit contact submanifolds. In *Robotics: Science and Systems*, Berlin, Germany, June 2013.

[15] Ingyu Kang and Frank Park. Cubic spline algorithms for orientation interpolation. *International journal for numerical methods in engineering*, 46(1):45–64, 1999.

[16] Inhyeok Kim and Jun-Ho Oh. Inverse kinematic control of humanoids under joint constraints. *International Journal of Advanced Robotic Systems*, January 2013.

[17] Tobias Kunz and Mike Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Robotics: Science and Systems*, pages 09–13, July 2012.

[18] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematics solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement, and Control*, (108):163–171, 1986.

[19] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518, 2005.

[20] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254, 1985.

[21] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Verlag, 2009.

[22] Mike Stilman. Task constrained motion planning in robot joint space. In *Intl. Conf. on Intelligent Robots and Systems*, pages 3074–3081, October 2007.

[23] Mike Stilman. Global manipulation planing in robot joint space with task constraints. *Trans. on Robotics*, 26(3):576–584, 2010.

[24] Deepak Tolani, Ambarish Goswami, and Norman Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

[25] Li-Chun Tommy Wang and Chih Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulator. *Robotics and Automation, IEEE Transactions on*, 7(4):489–499, 1991.

[26] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. AUGEM: Automatically generate high performance dense linear algebra kernels on x86 CPUs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2013.