International Conference on Intelligent Robots and Systems (IROS), 2014

# Spherical Parabolic Blends for Robot Workspace Trajectories

Neil Dantam and Mike Stilman

***Abstract*— We present a new approach to generate workspace trajectories for multiple waypoints. To satisfy workspace constraints with constant-axis rotation, this method splines a given sequence of orientations, maintaining constant-axis within each segment. This improves on other approaches which are point-to-point or take indirect paths. We derive this approach by blending subsequent spherical linear interpolation phases, computing interpolation parameters so that rotational velocity is continuous. We show this method first on simulated manipulator and then perform a physical screwing task on a Schunk LWA4 robot arm. Finally, we provide permissively licensed software which implements this trajectory generation and tracking.**[1]

## I. Introduction

Many robot tasks require tracking workspace motions. Tasks such as screwing in a light bulb or turning a doorknob impose constraints on the motion: rotation must occur along a single fixed axis. For such tasks, we focus on moving with straight-line translation and constant-axis rotation. A common way to specify motions is to provide a sequence of $n$ workspace points for the robot to move through. While generating smooth trajectories from waypoints for both joint-space and the Euclidean-space translations is well studied, the task of continuously transitioning between constant-axis rotations is more challenging. We present a new method to transition through a sequence of constant-axis rotations based on parabolic blending of spherical linear interpolation (SLERP) segments.

The proposed method generates a robot trajectory through a sequence of waypoints such that the axis of rotation remains constant between waypoints and rotational velocity is continuous. Compared to typical approaches for interpolation of robot workspace orientations, this method provides a constant rotational axis between waypoints, is invariant to the local reference frame, and avoids gimbal lock. Compared to classic SLERP, this method transitions through multiple waypoints without stopping whereas SLERP is point-to-point. Compared to typical methods for quaternion splines, this method provides a constant axis of rotation between waypoints. We discuss the application of quaternion interpolation to robot inverse kinematics (see Sect. III). Then, we derive the equations for spherical parabolic blends to produce our desired trajectories (see Sect. IV), summarizing the trajectory generation algorithm (see Sect. V). Finally, we demonstrate this method on simulated and physical robot manipulators (see Sect. VI).

This work supported by a grant from Peugeot S.A.

Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA. ntd@gatech.edu, mstilman@cc.gatech.edu

[1]Software available at https://github.com/golems/reflex

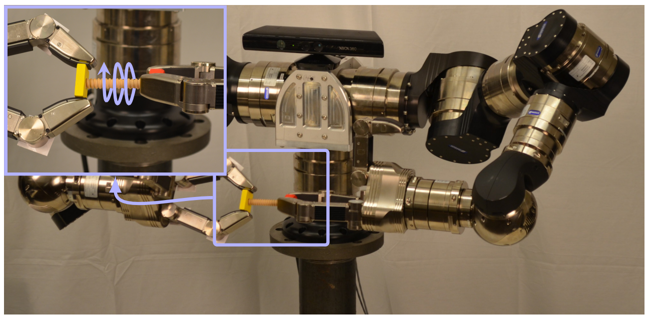

Fig. 1. Demonstrating spherical blending for a screwing task on a bimanual Schunk LWA4 manipulator with Schunk SDH hands.

## II. Related Work

Joint-space interpolation is a well studied research topic [5], [12], [11]. However, because the workspace orientations, $SO(3)$, are non-Euclidean, these joint-space methods are not directly applicable to orientation interpolation, particularly when we are concerned with the path taken between waypoints.

A related approach is to apply task constraints while planning a joint-space path [16], [17], [12], [8], [3]. We are considering a different problem: computing a workspace trajectory from a given sequence of waypoints. This enables correcting tracking errors directly in the workspace space (see Sect. V-B).

Typical methods for interpolating robot workspace orientations use Euler angles or *rotation vectors* (the rotation axis scaled by the rotation angle or equivalently the logarithm of the rotation). Interpolating the rotation vector representation [5, p217] varies the angle of rotation, which can produce undesirable paths, see Fig. 3. Euler angle approaches must contend with singularities (gimbal lock). Another approach is to vary the angle of a relative axis-angle orientation [15, p187], though this alone does not address continuity through waypoints. Instead, the quaternion representation is well suited for orientation interpolation as it avoids singularities with Euler angles and provides better paths than rotation vectors.

Spherical Linear Interpolation (SLERP) [14] interpolates between two quaternions along the unit, 4-dimensional hyper-sphere. SLERP has been applied to robot manipulation [4], [1], [2]. SLERP provides the desired constant axis of rotation, but is point-to-point, stopping at the beginning and end of each segment. We improve upon this by transitioning through a sequence of waypoints without stopping.

There is a large body of work on quaternion splines. The primary application domain for these approaches has been

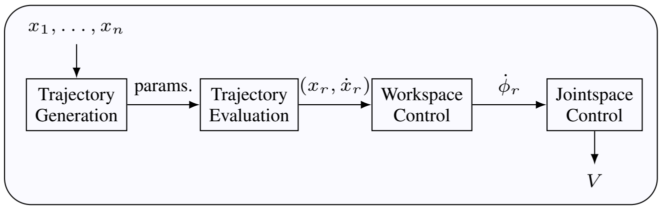

Fig. 2. Layers of abstraction going from a sequence of waypoints to inputs at each manipulator axis.

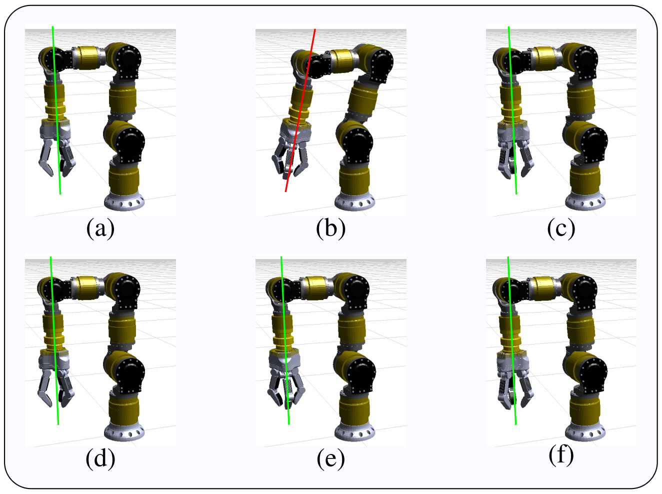

Fig. 3. Comparison of rotation vector [5, p217] and Spherical Linear Interpolation. In (a)-(c), interpolating the axis angle representation can give undesirable intermediate orientation (b). In (d)-(f), Spherical Linear Interpolation maintains a constant axis of rotation.

computer animation where the intermediate path may not be rigidly constrained compared to the end-effector of a physical robot. Consequently, methods such as quaternion Bezier curves [14], [10], SQUAD [6], and quaternion cubic splines [9] do not provide a constant axis of rotation. A key difference in our approach from these previous methods is that we explicitly differentiate between the interpolation parameter $u$ and time $t$. By considering $du/dt$, we can provide a smooth path with constant rotational axis for the bulk of the motion.

## III. SLERP for Inverse Kinematics

Spherical Linear Interpolation (SLERP) interpolates between an initial and final unit quaternion on the unit sphere [14]. SLERP can be computed as:

$$\texttt{slerp}\,(q_1, q_2; u) \triangleq q_1 \frac{\sin\left((1-u)\theta\right)}{\sin\theta} + q_2 \frac{\sin\left(u\theta\right)}{\sin\theta} \tag{1}$$

where $q_1$ and $q_2$ are the beginning and end points of the interpolation, interpolation parameter $u$ varies in $[0,1]$, and $\theta = \cos^{-1}(q_1 \cdot q_2)$.[2] To track this interpolation in real-time, we compute $u$ as a function of time.

We build upon SLERP to ensure smoothness of the path by considering the derivatives. Note that we must distinguish between the derivative with respect to interpolation parameter $u$ and with respect to time $t$.

$$\frac{dq}{dt} = \frac{dq}{du}\frac{du}{dt} \tag{2}$$

For a constant $q_1$ and $q_2$, the SLERP derivative is:

$$\frac{dq}{du}(u) = q_1 \frac{-\theta\cos\left((1-u)\theta\right)}{\sin\theta} + q_2 \frac{\theta\cos\left(u\theta\right)}{\sin\theta} \tag{3}$$

Given $dq/dt$, we can directly compute angular velocity as:

$$\omega = 2\frac{dq}{dt} \otimes q^* \tag{4}$$

where $q^*$ is the conjugate of $q$ and $\otimes$ is the quaternion multiplication operation.

This is then readily applied to robot workspace control via the Jacobian pseudo-inverse:

$$\dot{\phi}_r = (J^+)\begin{bmatrix}\dot{x} \\ \omega\end{bmatrix} \tag{5}$$

where $J$ is the manipulator Jacobian, $\dot{\phi}_r$ is the computed reference joint velocities and $\dot{x}$ is the desired translational velocity. For a robust, practical implementation, further considerations in (5) are also possible to correct position error and handle configurations near joint singularities [13].

However, we still need a method to compute $du/dt$ and ensure continuity of $dq/dt$

To simplify notation, we will sometimes write the time derivative $\frac{d\alpha}{dt}$ as $\dot{\alpha}$.

## IV. Derivation of Spherical Parabolic Blends

SLERP is useful for robots because it provides a constant axis of rotation during the motion. However, this constant axis of rotation introduces difficulties if we want to follow a path with waypoints. Consider the path from $q_i$ via $q_j$ to $q_k$. If we SLERP from $q_i$ to $q_j$ and $q_j$ to $q_k$, the path from $i$ to $j$ will have one axis of rotation and the path from $j$ to $k$ will have a different axis of rotation. This would produce a discontinuity in rotational velocity at point $j$, which could not be suitably followed by a physical robot. Thus, we must transition from the axis $ij$ to axis $jk$, maintaining $C^1$ continuity:

*Definition 1 (Differentiability class):* A function $f(t)$ is $C^k$ continuous if its derivatives $f', f'', \ldots, f^{(k)}$ exist and are continuous.

In the constant-axis, linear region from $q_i$ to $q_j$, we compute the orientation and its derivative via SLERP.

$$u = \frac{t - t_i}{t_j - t_i} \tag{6}$$

$$q(t) = \texttt{slerp}\,(q_i, q_j; u) \tag{7}$$

$$\frac{du}{dt} = \frac{1}{t_j - t_i} \tag{8}$$

$$\frac{dq}{dt} = \frac{dq}{du}(u)\frac{du}{dt} \tag{9}$$

where $t_i$ and $t_j$ are the times to reach orientations $q_i$ and $q_j$, respectively.

Around point $q_j$, we smoothly change the axis of rotation from that of $ij$ to that of $jk$. Over some blending interval

[2]A more accurate form is $\theta = 2\,\text{atan2}\left(|q_1 - q_2|, |q_1 + q_2|\right)$.

$t_b$, we "stretch" the $ij$ interpolation past $t_j$ and the $jk$ interpolation before $t_j$, ramping the interpolation parameters $u_{ij}$ and $u_{jk}$ over this time. To compute the actual $q$ in this region, we perform a third and final interpolation between the computed values for $q_{ij}$ and $q_{jk}$.

For this blend region around $q_j$, we compute the interpolation parameters by ramping down $\dot{u}_{ij}$, ramping up $\dot{u}_{jk}$.

$$t_{ij} = t_j - t_i \tag{10}$$
$$\Delta t = t - (t_j - t_b/2) \tag{11}$$
$$\ddot{u}_{ij} = -\frac{1}{t_{ij}t_b} \tag{12}$$
$$\dot{u}_{ij}(t) = \frac{1}{t_{ij}} + \Delta t \ddot{u}_{ij} \tag{13}$$
$$u_{ij}(t) = \frac{t_{ij} - t_b/2}{t_{ij}} + \frac{\Delta t}{t_{ij}} + \frac{\ddot{u}_{ij}}{2}(\Delta t)^2 \tag{14}$$
$$\ddot{u}_{jk} = \frac{1}{t_b\,(t_k - t_j)} \tag{15}$$
$$\dot{u}_{jk}(t) = \Delta t \ddot{u}_{jk} \tag{16}$$
$$u_{jk}(t) = \frac{\ddot{u}_{jk}}{2}(\Delta t)^2 \tag{17}$$

where $u_{ij}$ and $u_{jk}$ are the interpolation parameters from $q_i$ to $q_j$ and from $q_j$ to $q_k$, respectively, and $t_b$ is the blending period around $q_j$.

Then, we blend the two trajectories:

$$q_{ij}(t) = \texttt{slerp}\,(q_i, q_j; u_{ij}(t)) \tag{18}$$
$$q_{jk}(t) = \texttt{slerp}\,(q_j, q_k; u_{jk}(t)) \tag{19}$$
$$q(t) = \texttt{slerp}\,(q_{ij}(t), q_{jk}(t); u_j(t)) \tag{20}$$

Now, we return to computing the time derivative of SLERP, with the complication that the interpolation points in this case vary over time. This will let us derive the interpolation parameter for the blend region, $u_j(t)$, such that angular velocity is continuous. We can more precisely write the SLERP formula as:

$$q(t) = q_1(t)\frac{\sin((1-u(t))\theta(t))}{\sin\theta(t)} + q_2(t)\frac{\sin(u(t)\theta(t))}{\sin\theta(t)}$$
$$= q_1(t)a(t) + q_2(t)b(t) \tag{21}$$

where $u(t)$ and $\theta(t)$ are time varying functions, and $a(t)$ and $b(t)$ are substituted variables for the coefficients of $q_1(t)$ and $q_2(t)$.

Then, the time derivative is:

$$\frac{dq}{dt}(t) = \dot{q}_1(t)a(t) + q_1(t)\dot{a}(t) + \dot{q}_2(t)b(t) + q_2(t)\dot{b}(t) \tag{22}$$

The values of $\dot{q}_1$ and $\dot{q}_2$ can be computed from $dq/du$ in (3) and $\dot{u}$. We differentiate to find $\dot{a}(t)$ and $\dot{b}(t)$, dropping the parameter $t$ for brevity.

$$c_a = \cos(\theta(1-u)) \qquad s_a = \sin(\theta(1-u))$$
$$c_b = \cos(\theta u) \qquad s_b = \sin(\theta u)$$
$$\dot{a} = \frac{c_a\left(\dot{\theta}(1-u) - \dot{u}\theta\right)}{\sin\theta} - \frac{\dot{\theta}\cos(\theta)\,s_a}{\sin^2\theta} \tag{23}$$
$$\dot{b} = \frac{\left(\dot{\theta}u + \theta\dot{u}\right)c_b}{\sin\theta} - \frac{\dot{\theta}\cos(\theta)\,s_b}{\sin^2\theta} \tag{24}$$

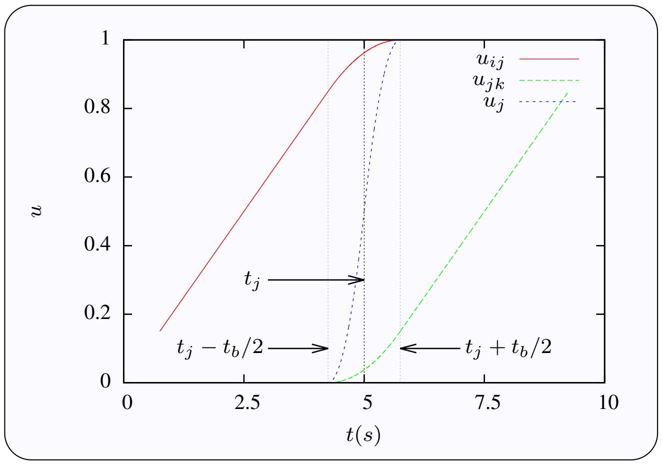

Fig. 4. SLERP $u$ values over linear and blend regions. $u_{ij}$ is the interpolation parameter between points $i$ and $j$, which ramps down during the blend. $u_j$ is the interpolation parameter for the blend region around $j$ when ramps up, then down during the blend. $u_{jk}$ is the interpolation parameter between points $j$ and $k$, which ramps up during the blend.

From (23) and (24) we can compute $u_j(t)$ to ensure that angular velocity is continuous. For this, we must ensure that the angular velocity at the beginning of the blend equals the angular velocity at the end the preceding linear segment and that angular velocity at the end of the blend equals that at the beginning of the following linear segment:

$$\dot{q}_j(t_j - t_b/2) = \dot{q}_{ij}(t_j - t_b/2) \tag{25}$$
$$\dot{q}_j(t_j + t_b/2) = \dot{q}_{jk}(t_j + t_b/2) \tag{26}$$

At the beginning of the blend segment around $j$ where $t = t_j - t_b/2$, we know $u_j = 0$, so we can simplify the coefficients of $\dot{q}_j$ in (22) as follows:

$$\dot{q}_j(t_j - t_b/2) = \dot{q}_{ij} + q_{ij}\dot{a} + q_{jk}\dot{b}_j \tag{27}$$
$$\dot{a}_j(t_j - t_b/2) = \dot{b}_j(t_j - t_b/2) = \frac{\dot{u}_j\theta}{\sin\theta} \tag{28}$$

Thus, if we have $\dot{u}_j(t_j - t_b/2) = 0$, then the coefficients $\dot{a}$ and $\dot{b}$ will be zero and (25) will be satisfied. A similar property holds at the end of the blend region, so we must have $\dot{u}_j(t_j - t_b/2) = \dot{u}_j(t_j + t_b/2) = 0$. We satisfy this property by computing $u_j$ based on a constant second derivative:

$$\ddot{u}_j = \frac{4}{t_b^2} \tag{29}$$

$$u_j(t) = \begin{cases} t \le t_j & 0.5\ddot{u}_j\,(\Delta t)^2 \\ t > t_j & 1 - 0.5\ddot{u}_j\,(t_j + t_b/2 - t)^2 \end{cases} \tag{30}$$

From (30), we compute $u_j(t)$ for (20) and $\dot{u}_j$ for (2). Fig 4 plots values of $u_{ij}$, $u_{jk}$ and $u_j$ over the linear and blend regions.

Now, we find $\dot{\theta}$. To simplify, we assume that $q_1(t)$ and $q_2(t)$ are unit quaternions.

$$\theta(t) = \cos^{-1}(q_1(t)\cdot q_2(t))$$
$$\dot{\theta}(t) = -\frac{q_1(t)\cdot\dot{q}_2(t) + \dot{q}_1(t)\cdot q_2(t)}{\sqrt{1 - (q_1(t)\cdot q_2(t))^2}} \tag{31}$$

Combining (22) - (31), we compute the quaternion derivative $dq/dt$, and with (4), we find the angular velocity $\omega$ in the blend region.

## V. Generating and Tracking Trajectories

Following the derivation in Sect. IV, we summarize generating trajectory parameters from a sequence of waypoints, computing reference workspace velocities, and finding corresponding reference joint velocities.

### A. Generation

Given a sequence of orientations $q_i$, waypoint times $t_i$, and blend times $t_{bi}$: $(q_0, t_0, t_{b0}), (q_1, t_1, t_{b1}) \ldots, (q_n, t_n, t_{bn})$,

1) Add virtual waypoints at $q = q_0$ at time $t_0 + t_b/2$ and $q = q_n$ at $t_n - t_b/2$ to the trajectory to provide blending for initial and final points.
2) For every triplet of orientations, $q_i$, $q_j$, and $q_k$, compute $\ddot{u}_{ij}$, $\ddot{u}_{jk}$, and $\ddot{u}_j$ according to (12), (15), and (29).

### B. Tracking

To track a generated trajectory, alternate between a sequence of blend and linear regions. Around each waypoint $q_j$ from $t_j - t_b/2$ to $t_j + t_b/2$, blend orientations. From $t_j + t_b/2$ to $t_k - t_b/2$, SLERP from $q_j$ to $q_k$.

Note that there is no linear region between $q_0$ and the virtual waypoints at $t_0 + t_b/2$, nor between the virtual waypoints at $t_n - t_b/2$ and $q_n$.

*Linear Regions:* For the linear region between $q_i$ and $q_j$:

1) $\dot{u}_{ij} = \frac{1}{t_j - t_i}$
2) Compute $u_{ij}(t) = (t - t_i)\,\dot{u}_{ij}$
3) Compute $q(u_{ij})$ according to (1)
4) Compute $\dot{q}(u_{ij})$ according to (3) and (2)
5) Compute $\omega(t)$ according to (4)

*Blend Regions:* For the blend region between $q_i$, $q_j$, and $q_k$:

1) Compute $\dot{u}_{ij}$, $u_{ij}$, $\dot{u}_{jk}$, $u_{jk}$, $\dot{u}_j$, and $u_j$ according to (13), (14), (16), and (17).
2) Compute $q_{ij} = \texttt{slerp}\,(q_i, q_j; u_{ij})$ and $q_{jk} = \texttt{slerp}\,(q_j, q_k; u_{jk})$
3) Compute $\dot{q}_{ij}$ $\dot{q}_{jk}$ according to (3) and (2).
4) Compute $q(u_j) = \texttt{slerp}\,(q_{ij}, q_{jk}; u_j)$
5) Compute $\dot{q}(u_j)$ from (31), (23), (24), and (22).
6) Compute $\omega(t)$ according to (4)

### C. Workspace Control

Now, we apply a singularity-robust Jacobian inverse kinematics to obtain joint velocities from the generated workspace trajectory [13]. To provide acceptable performance near joint singularities, we compute the damped pseudo-inverse of the Jacobian as follows:

$$J^+ = \sum_{i=0}^{\min(m,n)} \frac{s_i}{\max({s_i}^2, {s_{\min}}^2)} v_i u_i^T \tag{32}$$

where $J = USV^T$ is the singular value decomposition of $J$ and $s_{\min}$ is a selected constant for the minimum acceptable singular value.[3]

[3]On the Schunk LWA4 in Fig. 1, a reasonable value for $s_{\min}$ is .01.

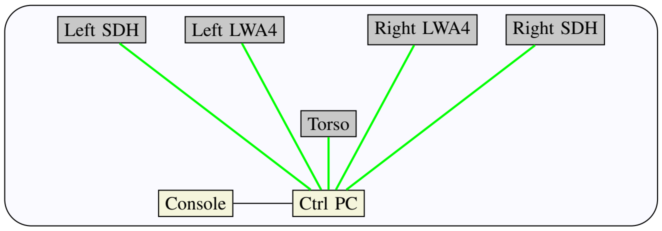

Fig. 6. Block Diagram of robot manipulator hardware components. The control PC communicates with the servo controllers over several CAN buses.

We compute the damped-least squares solution for feed-forward velocity and feedback-position control:

$$\begin{aligned}\dot{\phi}_r &= J^+ \left( \begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x e \right) \\ &= J^+ \left( \begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x \begin{bmatrix} x - x_r \\ \ln(q \otimes q_r^*) \end{bmatrix} \right)\end{aligned} \tag{33}$$

where $e$ is the position error.

In addition, for redundant manipulators with more than six degrees of freedom, we use the null-space projection to help avoid joint limits by directing the joint positions towards a nominal zero point.

$$\dot{\phi}_r = J^+ \left( \begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x \begin{bmatrix} x - x_r \\ \ln(q \otimes q_r^*) \end{bmatrix} \right) - k_\phi (J^+ J - I)(\phi - \phi_0) \tag{34}$$

where $k_x$ is the workspace position error gain, $k_\phi$ is the null-space projection gain, and $\phi_0$ is the nominal zero configuration.

We then use joint-level velocity control to track the reference joint velocities $\dot{\phi}_r$.

## VI. Experimental Results

### A. Simulation

We first demonstrate these trajectories on a kinematically simulated Schunk LWA3 robot with 7 Degrees of Freedom (DOF). Fig. 5 shows the workspace orientation and derivative through a sequence of orientations. From this, we can see that orientation is $C^1$ continuous. Between each of the waypoints, we have an accelerating segment, a constant-velocity segment, and a decelerating segment.

### B. Physical Implementation

We validate our trajectory generation approach on a physical Schunk LWA4 arm with Schunk SDH hand for a screwing task, Fig. 1. The LWA4 is a 7 DOF arm that offers a shorter distance between wrist point and end-effector than the LWA3. Fig. 6 gives an overview of the major physical system components. Our real-time software runs on Xeon E3-1270v2 PC under Linux 3.4.18-rt29 PREEMPT RT, and is implemented as multiple operating system processes communicating using the Ach interprocess communication library [7]. Fig. 7 summarizes the real-time software components.

We generate and execute a trajectory to screw together the wooden pieces. Because of the SDH's kinematic configuration, it grasps the screw such that the screw axis is offset

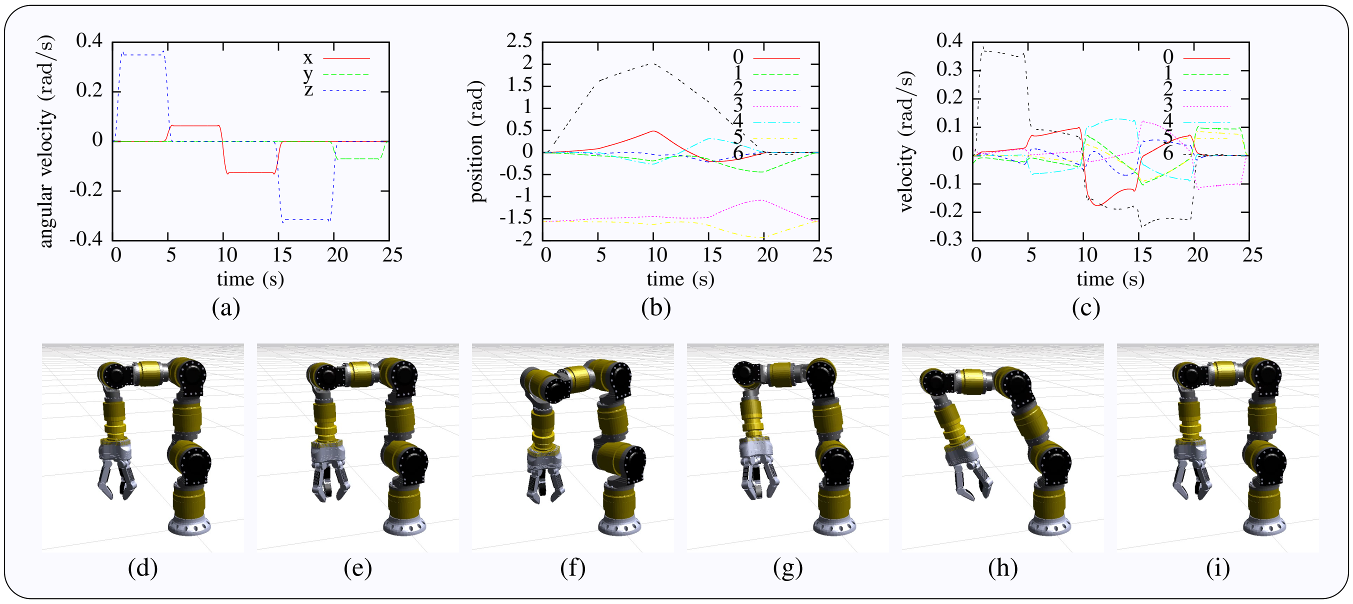

Fig. 5. Simulated trajectory through the following waypoints specified in XZY Euler Angles: $(-\pi/2, 0, \pi)$, $(-\pi/2, \pi/10, \pi)$, $(-\pi/2, -\pi/10, -\pi)$, $(\pi, -\pi/10, -\pi)$, $(\pi, 0, \pi)$. Interpolation is performed on the quaternion representation. (a) Workspace Angular Velocity. (b) Joint Position. (c) Joint Velocity. (d)-(i) Via Orientations.

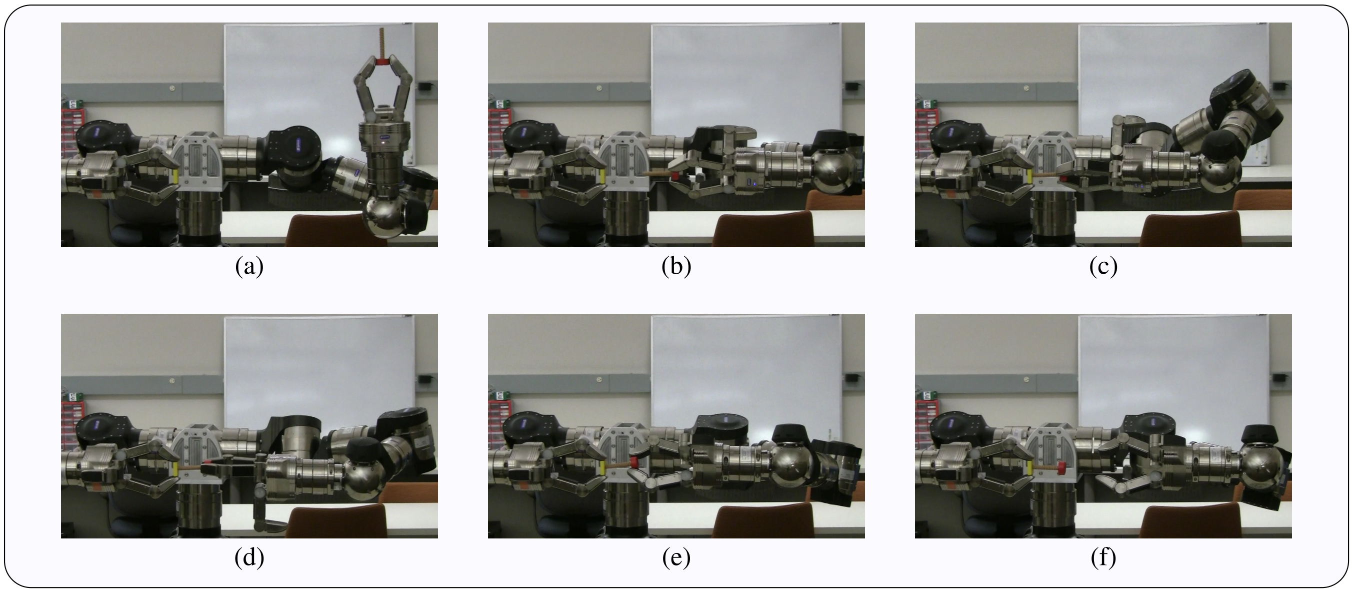

Fig. 8. Physical Screwing Task. (a)-(e), trajectory waypoints. (f), the inserted screw.

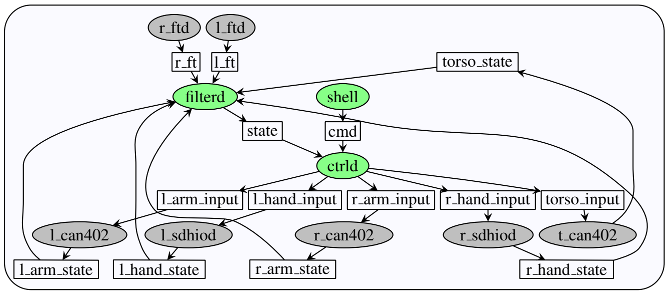

Fig. 7. Block diagram of real-time software components. Gray ovals are user-space driver processes, green ovals are controller processes, and rectangles are Ach channels.

from the last wrist axis. Thus, we cannot turn the screw by rotating only the last joint but must instead consider the entire arm. The provided waypoints to insert the screw are shown in Fig. 8, and the generated trajectory in workspace and joint space is plotted in Fig. 9. This trajectory aligns, inserts, and turns the screw in one continuous, non-stop motion.

### C. Computational Performance

This method is computationally efficient. The generation phase to pre-compute parameters requires $O(n)$ time, where $n$ is the number of waypoints. The tracking phase requires $O(1)$ time during each control cycle. Tracking does require

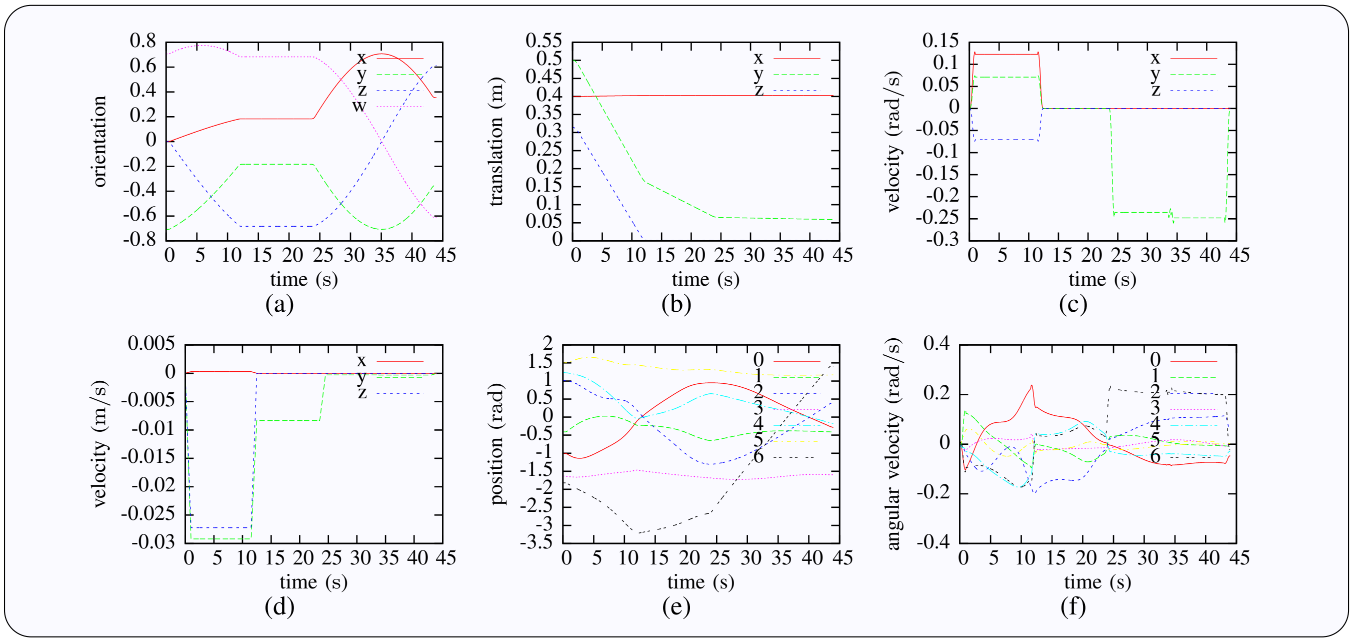

Fig. 9. Plots of positions and velocities for physical screwing task. (a) Workspace orientation. (b) Workspace translation. (c) Workspace rotational velocity. (d) Workspace translational velocity. (e) Joint positions. (f) Joint velocity

| | |
|---|---|
| Linear Region | .39 µs |
| Blend Region | 1.3 µs |

TABLE I

Time to compute reference parameters per control cycle. Average of 10 million evaluations on an Intel Xeon E5-1620.

evaluating a few transcendental functions during each control cycle; however, for modern CPUs at typical real-time control rates, e.g., one kilohertz, this is not a significant factor. Table I shows evaluation times on the order of one microsecond for a recent Intel CPU.

## VII. Conclusion

We have presented a method for generating workspace trajectories through a sequence of multiple waypoints, maintaining constant axis of rotation between each pair of points. This approach improves upon previous methods for workspace trajectories by avoiding singularities, continuously transitioning through multiple waypoints, and maintaining a constant axis of rotation between waypoints. This enables continuous motion without stopping for axis-constrained tasks such as screwing and rotating parts. We have demonstrated these trajectories in simulation and validated the approach on a physical manipulator by aligning and screwing together two parts in one continuous motion.

## References

[1] ABB. *Operating manual, RobotStudio*, 5.15k edition, 2013. Document ID: 3HAC032104-001.

[2] Jin-su Ahn, Won-jee Chung, and Chang-doo Jung. Realization of orientation interpolation of 6-axis articulated robot using quaternion. *Journal of Central South University*, 19(12):3407–3414, 2012.

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

[4] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

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

[6] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[7] N. Dantam and M. Stilman. Robustness and efficient communication for real-time multi-process robot software. In *Humanoids*. IEEE, 2012.

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

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

[10] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Computer Graphics and Interactive Techniques*, pages 369–376. ACM, 1995.

[11] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, January 2010.

[12] 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.

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

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

[15] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Verlag, 2009.

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

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