2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

High-Speed High-Accuracy Spatial Curve Tracking Using Motion Primitives in Industrial Robots

Honglu He, Chen-lung Lu, Yunshi Wen, Glenn Saunders, Pinghai Yang, Jeffrey Schoonover, John Wason, Agung Julius, John T. Wen

Abstract— **Industrial robots are increasingly deployed in applications requiring an end effector tool to closely track a specified path, such as in spraying and welding. Performance and productivity present possibly conflicting objectives: tracking accuracy, path speed, and motion uniformity. Industrial robots are programmed through motion primitives consisting of waypoints connected by pre-defined motion segments, with specified parameters such as path speed and blending zone. The actual executed robot motion depends on the robot joint servo controller and joint motion constraints (e.g., velocity, acceleration limits) which are largely unknown to the users. Programming a robot to achieve the desired performance today is time-consuming and mostly manual, requiring tuning a large number of coupled parameters in the motion primitives. The performance also depends on the choice of additional parameters: possible redundant degrees of freedom, location of the target curve, and the robot configuration. This paper presents a systematic approach to optimize robot motion parameters. The approach first selects the static parameters, then chooses the motion primitives, and finally iteratively updates the waypoints to minimize the tracking error. The ultimate performance objective is to maximize the path speed subject to the tracking accuracy and speed uniformity constraints over the entire path. We have demonstrated the effectiveness of this approach both in simulation and on physical systems for ABB and FANUC robots applied to two challenging example curves. Comparing with the baseline using the current industry practice, the optimized performance shows over 100% performance improvement.**

Keywords: Industrial Robot, Motion Primitive, Path Optimization, Redundancy Resolution, Trajectory Tracking

I. INTRODUCTION

Industrial robots are increasingly deployed in applications such as spray coating [1], arc welding [2], deep rolling [3], surface grinding [4], cold spraying [5], etc., where the tool center point (TCP) frame attached to the end effector needs to track complex geometric paths in both position and orientation. In most applications, the task performance is characterized by the motion speed (how long it takes to complete the task), motion uniformity (how much the speed varies along the path), and motion accuracy (the maximum position and orientation tracking errors along the path). There are generally two ways to program industrial robots for the motion tracking task: (1) The motion primitive method that uses vendor-specific proprietary robot programming languages (e.g., [6]–[11]) consisting of a sequence of predefined motion primitives, and (2) The streaming setpoint method which uses an external command mode to stream the desired joint position as setpoints to the low level joint servo controller (e.g., [12]–[15]). The motion primitive method is far more widely used in industries. It decomposes the desired motion with predefined motion primitives consisting of waypoints connected by motion segments. The streaming setpoint method is not limited to a small set of motion primitives, but it typically results in poorer tracking accuracy due to the lower sampling rate and the additional filtering and latency.

To program an industrial robot to follow a complex path using motion primitives, the most common approach is to use it like a machine tool [16], focusing only on the TCP motion. There are several advanced offline programming software (e.g., [17]–[19]) that convert a dense set of TCP waypoints to the robot program of a given robot vendor. The actual robot motion from the execution of the robot program will depend on the motion primitives and their parameters (waypoint locations, commanded path speed along the motion segment, size of the blending zone between motion segments) which are affected by the robot joint servo controller and the robot joint motion constraints (joint velocity and acceleration limits). Robot controller and robot joint constraints may not be known to the user. Therefore, programming a robot to achieve the desired performance is currently a time-consuming and largely manual exercise in tuning the motion primitive parameters and the result may be far from optimal. Compounding the challenges are the impact of additional parameters: a redundant roll degree of freedom in the tool frame, relative pose between the target curve and the robot base, and the robot configuration (out of the 8 possible choices in a 6 degrees of freedom (DoF) revolute robot).

The streaming setpoint method avoids motion blending, but the actual robot motion is difficult to predict, again because of the unknown low level controller and robot motion constraints, in addition to the latency and filtering present in the streaming operation. Motion planning tools such as Tesseract [20] (based on TrajOpt [21]), tries to optimize the robot joint path for path speed subject to tracking error and speed uniformity constraints. Such optimization involves a large number of variables and is plagued by computation time, presence of local minima, and the lack of accurate prediction of the actual robot motion.

A key motivation of this work is the *cold spray* application which impinges particles onto a surface at high speed to

Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, heh6@rpi.edu luc6@rpi.edu weny2@rpi.edu
Manufacturing Innovations Center, Rensselaer Polytechnic Institute, saundg@rpi.edu
GE Research, US, Pinghai.Yang@ge.com schoonov@ge.com
Wason Technology, LLC, wason@wasontech.com
Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, agung@ecse.rpi.edu wenj@rpi.edu

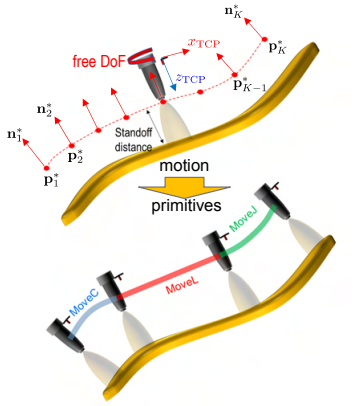


Fig. 1: Illustration of the motion primitive method for path following.

form protective coating [5]. For such applications, the speed of the TCP needs to be *high and uniform* to guarantee fast execution and uniform distribution of the deposited material. As shown in Fig. 1, the spraying path is given as a dense sequence of points $\{\mathbf{p}_i^*\}_{i=1}^K$ for the TCP position and the spraying direction in the negative tool z-axis is given by the surface normal $\{\mathbf{n}_i^*\}_{i=1}^K$. Note that these data specify 5-DoF of the TCP at each point, with the remaining free DoF being the rotation about the tool z-axis. Once the TCP frame is fixed, next choice is the robot joint configuration among the eight inverse kinematics solutions. We also have an additional 6 free DoF in the position and orientation of the curve, parameterized by the position $\mathbf{p}_{curve}$ and the angle-product representation of the orientation $\boldsymbol{\beta}_{curve}$. The motion primitive method first chooses all these free parameters, we call this step *redundancy resolution*. The spatial curve is then converted to motion primitives as shown in Fig. 1 for execution by the robot controller. This paper presents a new approach to motion optimization by decomposing the problem into a series of subproblems. We first use a combination of evolution algorithm and local optimization to resolve redundancy. Then we use a greedy algorithm for fitting the curve with the least number of motion primitives. Finally, an iterative gradient descent algorithm updates the waypoints to minimize the worst case tracking error. To evaluate this approach, the industry co-authors of this work suggested two test curves, Curve 1 contains a range of high curvature moves and Curve 2 is the leading edge of a sample turbine fan blade. A baseline approach is developed based on the current industry practice. The evaluation metric is the average path speed subject to the specified trajectory tracking and path speed uniformity constraints. We implemented the motion primitive optimization approach on an ABB IRB 6640 robot in simulation (RobotStudio) and physical testbed, and on an FANUC M10iA robot in simulation (RoboGuide) and physical testbed. In all cases, the performance vastly improves over the baseline, ranging over 149% to 489%.

Notation:

- $\mathbf{p} \in \mathbb{R}^3, \boldsymbol{\beta} \in \mathbb{R}^3$: Cartesian position and the angle-product representation of orientation.
- $\phi \in \mathbb{R}$: the rotation angle around tool-z axis, with $\phi = 0°$ at tool-x axis pointing toward curve tangent direction
- $\mathbf{p}^* \in \mathbb{R}^3$ and $\mathbf{n}^* \in \mathbb{R}^3$: the position and normal vector along the desired curve in the robot base frame.
- $\mathbf{q} \in \mathbb{R}^6$: robot joint position.
- $\mathbf{p}(\mathbf{q})$ and $\boldsymbol{\beta}(\mathbf{q})$: forward kinematics mapping from robot joint position.
- $e_z(\mathbf{q}) \in \mathbb{R}^3$: TCP z-axis unit vector in the robot base frame.
- $J(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} \omega \\ v \end{bmatrix}$: Jacobian mapping between joint velocity and TCP spatial velocity.

Contribution: We present a method for determining a set of motion primitives and their respective parameters (see Sec. II for more details) so that the TCP tracks the desired positions and orientations with sufficient accuracy and speed uniformity (see Sec. III for more details) as fast as possible.

II. Motion Primitives and Their Executions

A. Motion Primitives

Generally all robot programming languages support three types of motion primitives as described below.

- MoveL: The TCP moves linearly in Cartesian space and linearly in angle-product orientation from current pose to desired pose. This command is parameterized by the initial and final TCP positions and angle-product orientations.
- MoveC: The TCP moves on a circular arc in Cartesian space and linearly in angle-product orientation from current pose to desired pose. This command is parameterized by the initial and final TCP positions, an intermediate TCP position (to define the circular arc), and the initial and final angle-product orientations, an intermediate orientation (to define rotation orientation).
- MoveJ: The robot moves linearly in the *joint angles space*. Thus, the TCP path may not be linear nor circular. This command is parametrized by the initial and final joint angles.

B. Blending

Given a series of motion primitive segments, the desired tool path is blended together to avoid sharp turns [22]. The size of the blending zone is specified with the primitives. Larger blending zone means smoother transition to next segment, and hence higher and more uniform speed, but at the cost of large tracking error at the waypoint (intersection of motion segments). Small blending zone improves the tracking accuracy but could lead to sharp corner requiring slower speed with larger variation to meet the motion constraints. In current practice, expert human operators perform extensive manual adjustments of the waypoints, blending regions, and motion segment commanded speeds to ensure that tracking accuracy and cycle time requirements are both met. In our approach, we find the smallest uniform blending zone that avoids excessive speed variation around waypoints to balance between the two objectives.

III. Specifications and Proposed Approach

Based on the cold spray application, the trajectory tracking performance specification is given by:

1. Positional tracking accuracy $max(\mathbf{p}_{err})$**:** the maximum positional tracking error of the TCP is less than 0.5 mm.

2. Orientation tracking accuracy $max(\mathbf{n}_{err})$**:** the maximum orientation tracking error of tool z-axis is less than 3°.

3. Speed uniformity: the standard deviation of the TCP speed $\sigma(v)$ is less than 5% of the average TCP speed $\mu(v)$.

We calculate the TCP position and orientation using forward kinematics of the with joint reading. To calculate the position and orientation tracking errors, for each recorded joint position point $\mathbf{q}_k$ we compute

$$\begin{aligned} \mathbf{p}_{err,k} &:= \min_i \|\mathbf{p}(\mathbf{q}_k) - \mathbf{p}_i^*\|, \\ \mathbf{n}_{err,k} &:= \min_i \arccos\left(\mathbf{e}_z^T(\mathbf{q}_k)\mathbf{n}_i^*\right). \end{aligned} \tag{1}$$

where $\mathbf{e}_z$ is the tool z-axis. The maximum position and orientation tracking errors are found by maximizing both terms above over i. The general optimization could be formed as

$$\begin{aligned} &\max_u \mu(v) \qquad \text{subject to} \\ &\max(\mathbf{p}_{err}) \le 0.5, \max(\mathbf{n}_{err}) \le 3, \sigma(v) \le 5\%\,\mu(v) \end{aligned} \tag{2}$$

where u is a control variable vector including

- ($\mathbf{p}_{curve}$,$\boldsymbol{\beta}_{curve}$): 6 Dof curve pose in robot base frame
- ϕ: Tool-z rotation angle
- Robot Arm Configuration: One out of maximum eight joint configurations due to the non-uniqueness of the inverse kinematic map
- Waypoints in the trajectory
- Motion Primitive: Choice of motion primitives including segment speed and blending zone

Our approach decomposes the motion primitive optimization problem into three steps as shown in Fig. 2.

1. Redundancy Resolution: Optimize the redundant DoF of curve pose: ($\mathbf{p}_{curve}$,$\boldsymbol{\beta}_{curve}$), the redundant DoF along the path (ϕ), and the robot arm configuration. The output of this process is the full 6 DoF reference path for the TCP and the corresponding joint path.

2. Motion Primitive Planning: Generate a sequence of motion primitives (waypoints, blending zones, and path speed) to track the reference path.

3. Waypoints Adjustment: The motion sequence is executed in simulations or experiments. Tracking error from the executions is used to optimize the motion primitives parameters. These steps are described in greater details below.

A. Redundancy Resolution

In contrast to computer numerical control (CNC) machines, robot arm configuration significantly affects the TCP motion due to the robot joint constraints, including the joint position, velocity, and acceleration limits.

Our approach is to first resolve the tool redundant DoF by finding the complete robot joint path based on a given

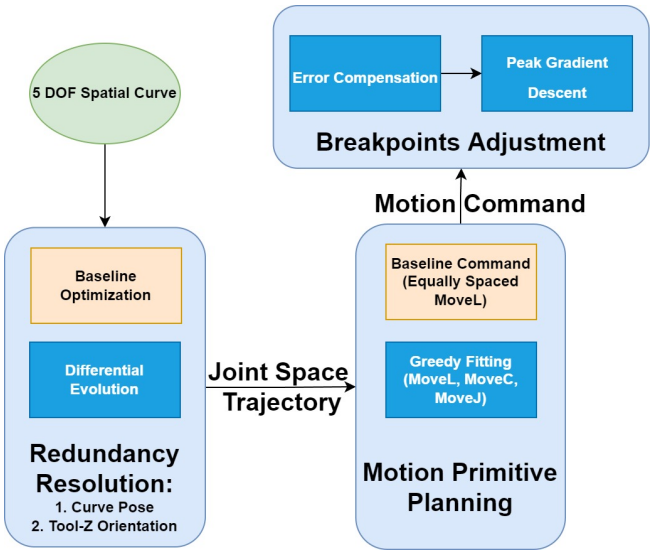


Fig. 2: Proposed Workflow for Spatial Curve Tracking with Motion Primitives.

curve pose and robot arm configuration. At each point on this joint path, we can find the path velocity limit based on the joint velocity and acceleration limits. We then choose the curve pose and robot arm configuration to maximize the lowest path velocity limit. The joint acceleration limit is due to the motor torque constraint. Since the arm inertia is configuration dependent, the joint acceleration limit is also configuration dependent (hence, an outstretched arm has a lower acceleration limit). We will show how to estimate this limit through simulation.

1) Tool Orientation Resolution: For a given curve pose, ($\mathbf{p}_{curve}, \boldsymbol{\beta}_{curve}$), we have $\{\mathbf{p}_k^*, \mathbf{n}_k^*\}_{k=1}^K$ in the robot base frame and the initial joint angle $\mathbf{q}_0$. We select the redundant DoF (ϕ) so that the TCP motion remains on the path with minimal incremental joint motion. Since joint motion and TCP motion are related through the arm Jacobain $J(\mathbf{q})$, this procedure tends to steer the robot away from singularities without compromising TCP tracking. We pose the incremental optimization as a quadratic program (QP): Given $\mathbf{q}_{k-1}$, $k = 1, \ldots, K$, find the incremental motion $\delta\mathbf{q}$ to move to the next point $(\mathbf{p}_k^*, \mathbf{n}_k^*)$ without excessive joint motion and within the joint limits:

$$\begin{aligned} &\min_{\delta\mathbf{q}} \|J_r(\mathbf{q}_{k-1})\delta\mathbf{q} - \nu^*\|^2 + \|\delta\mathbf{q}\|_{W_q}^2 \\ &\text{subject to } q_{\min} \preccurlyeq \mathbf{q}_{k-1} + \delta\mathbf{q} \preccurlyeq q_{\max} \end{aligned} \tag{3}$$

where (q_{min}, q_{max}) are joint limits, W_q is a weightfactor, ν^* is the TCP position and surface normal increment, and J_r is the reduced Jacobian mapping $\delta\mathbf{q}$ to the position and surface normal increment:

$$\nu^* := \begin{bmatrix} \mathbf{n}_k^* - \mathbf{e_z}(\mathbf{q}_{k-1}) \\ \mathbf{p}_k^* - \mathbf{p}(\mathbf{q}_{k-1}) \end{bmatrix}, \tag{4}$$

$$J_r(\mathbf{q}) := \begin{bmatrix} -\mathbf{e_z}(\mathbf{q})^\times & 0 \\ 0 & I \end{bmatrix} J(\mathbf{q}). \tag{5}$$

The next joint position is simply the current joint position incremented by $\delta\mathbf{q}$. Since the Jacobian mapping is only

approximate for finite increments, we optimize the increment size by writing $\mathbf{q}_k = \mathbf{q}_{k-1} + \alpha\delta\mathbf{q}$ and choose α from a line search:

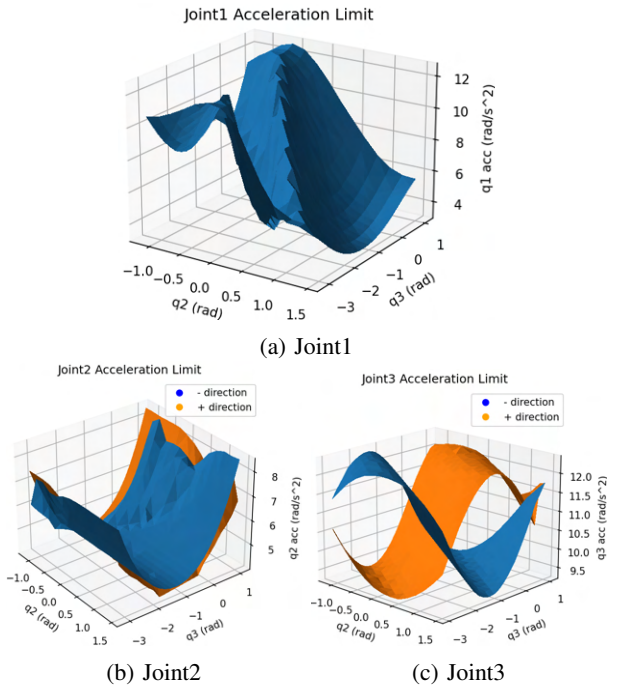
$$\min_{\alpha} \left\| \begin{bmatrix} \mathbf{e_z}(\mathbf{q}_k) - \mathbf{n}_k^* \\ \mathbf{p}(\mathbf{q}_k) - \mathbf{p}_k^* \end{bmatrix} \right\|.$$

The resulting sequence of joint angles $\{\mathbf{q}_k\}_{k=0}^K$ now completely specifies the TCP position and orientation.

2) Trajectory Traversal Speed Estimation: The robot TCP speed limit is determined by its joint velocity and acceleration limits. Industrial robots provide joint velocity limits in the data sheets [23], but the acceleration limit is due to the torque limit and is therefore affected by the robot dynamics, load, and robot configuration. Since the dynamical models of industrial robots are typically unknown, we establish a rough estimation by executing an sinusoidal trajectory both in simulation and on physical robots. Since the arm inertia is most heavily influenced by the shoulder and elbow joints (joints 2 and 3), we assume that the spherical wrist joints (joint 4, 5, and 6) have constant joint acceleration limits, while joints 1, 2 and 3 acceleration limits depend on (q_2, q_3), the shoulder and elbow joints, and joint 2 and 3 acceleration limits also depend on the direction of vertical motion due to gravity. The recorded result is shown in Table I and Fig. 3.

Joint	1	2	3	4	5	6
$\dot{q}_{max}$ (rad/s)	1.745	1.571	1.571	3.316	2.443	3.316
$\ddot{q}_{max}$ (rad/s^2)	*	*	*	42.5	36.8	50.5

TABLE I: ABB6640 joint velocity and acceleration limits. * = configuration dependent, see Fig. 3.



(a) Joint1

(b) Joint2 (c) Joint3

Fig. 3: Configuration Dependent Joint Acceleration Limit (ABB6640): Recorded maximum joint acceleration of joint 1,2,3 with respect to different joint 2 and 3 configurations.

The joint acceleration limits are stored as a look-up table dependent on (q_2, q_3): $\ddot{\mathbf{q}}_{\max}(q_2, q_3)$. For each robot joint angle point $\mathbf{q}_k$ along the path (from Sec. III-A.1), the maximum joint velocity limit is given by

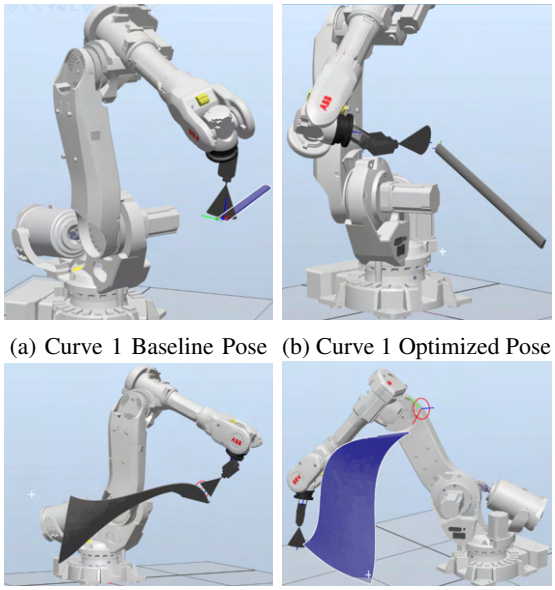
$$\dot{\mathbf{q}}_{\max_k} = \min\left(\dot{\mathbf{q}}_{\max}, \dot{\mathbf{q}}_{k-1} + \Delta t_k \ddot{\mathbf{q}}_{\max}(q_{2_{k-1}}, q_{3_{k-1}})\right). \tag{6}$$

For a specified (scalar) path speed v_d, $\mathbf{p}_k = \mathbf{p}_{k-1} + v_d \Delta t_k$. Therefore,

$$\Delta t_k = \frac{\|\mathbf{p}_k - \mathbf{p}_{k-1}\|}{v_d}. \tag{7}$$

Hence the path velocity limit for point k on the path is

$$v_k = \|J_v(\mathbf{q}_k)\dot{\mathbf{q}}_{\max_k}\|. \tag{8}$$



(a) Curve 1 Baseline Pose (b) Curve 1 Optimized Pose

(c) Curve 2 Baseline Pose (d) Curve 2 Optimized Pose

Fig. 4: Baseline curve pose (left) vs optimized curve pose (right) for Curve 1 and Curve 2 (described in Sec. IV-A.)

3) Differential Evolution with Optimizing Parameters: With the path speed limit parameterized by the curve pose and robot arm configuration, we can optimize them to maximize the lowest speed limit as

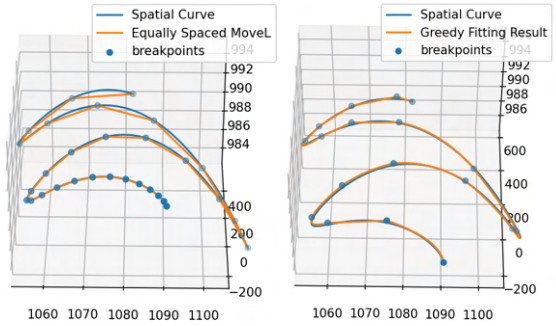
$$\max_{\mathbf{p}_{curve}, \boldsymbol{\beta}_{curve}, \mathbf{q}_0} min(\{v_k\}_{k=0}^K) \tag{9}$$

where v_k is given by (8). We use the differential evolution algorithm [24] to find the global optimizing solution for two example curves. The optimized poses are as shown in Fig. 4 versus the baseline (see Section IV-B).

B. Motion Primitive Planning with Greedy Fitting

Existing robot CAD/CAM software only use MoveL with a specified spatial curve. In order to take advantage of MoveC and MoveJ, we apply a spatial curve (represented in joint space and Cartesian space) greedy fitting algorithm. The fitting process is performed sequentially along the trajectory, starting from the initial point and bisection search for the longest possible primitive type within the error threshold. For

the initial segment, all regressions are unconstrained, while subsequent sequences are subject to continuity constraint (regression must pass through last segment endpoint). Fig. 5 shows a visualization of 30 equally spaced MoveL's and Greedy Fitting with all three primitives on Curve 1. While 30 linear segments will result in 1.47 mm max position tracking error, greedy fitting only uses 17 segments to achieve a max position tracking error under 0.3 mm. In general, too few waypoints lead to higher tracking error, and too many waypoints could result in speed variation due to the overlapping interpolation zones.



(a) Equally spaced MoveL's (b) Greedy fitting

Fig. 5: Curve Fitting Visualization.

C. Waypoints Adjustment

The execution of the robot motion program would likely result in a trajectory that does not meet the performance specification. We can iteratively modify the waypoints to reduce the trajectory tracking error using a descent algorithm. We apply the following two algorithms sequentially:

- Error Compensation: Push all waypoints in the error direction.
- Multi-peak Model Based Gradient Descent: Use an approximate trajectory to calculate the numerical gradient and adjust waypoints based on the gradient descent. To simplify computation, we only apply to points at the error peaks.

Error compensation takes place in the first few iterations, aiming to reduce the tracking error at each waypoint:

$$\begin{aligned} \mathbf{p}_k &\leftarrow \mathbf{p}_k + \gamma(\mathbf{p}_k^* - \mathbf{p}_k), \\ \mathbf{R}_k &\leftarrow rot(\hat{\mathbf{k}}, \gamma \mathbf{n}_{err,k})\mathbf{R}_k. \end{aligned} \tag{10}$$

where $\mathbf{p}_j$ and $\mathbf{R}_j$ are the position and orientation at waypoint k, $\hat{\mathbf{k}}$ is the normalized $\mathbf{e}_{z,k} \times \mathbf{e}_{z,k}^*$, rot is the rotation matrix for the given axis and angle, γ is the step size.

Error compensation step only focuses the error at the waypoints, while the tracking error between waypoints may be unacceptably large. Until maximum tracking error no longer decreases, we next apply a gradient descent method using an approximate numerical gradient to reduce the largest tracking errors. For a given set of waypoints, we can efficiently generate a predicted joint trajectory using cubic spline interpolation at each waypoint. This predicted trajectory may then be used to find the numerical gradient by perturbing the waypoints and record the corresponding tracking error change. As the trajectory error is predominately affected by its neighboring waypoints, we only generate a tri-diagonal gradient, and only for the peak trajectory errors.

IV. Implementation and Evaluation

A. Test curves

The performance is evaluation is based on two representative curves. Curve 1, shown in Fig. 5 and Fig. 6a, is a multi-frequency sine curve on a parabolic surface. This curve is representative of a high curvature spatial curve. Curve 2, shown in Fig. 6b, is extracted from the leading edge of a generic fan blade model, which is a typical case for cold spraying applications in industry.

B. Baseline

In consultation with industry experts, we establish a baseline performance based on the current practice. The curve is placed in the middle of the robot workspace, away from singularities. The tool redundancy is resolved by choosing the tool x-axis along the curve. The robot arm configuration is chosen based on the largest manipulability measure. The motion primitives are based on equally spaced moveL segments. We then search for the largest commanded path speed that satisfy both the path uniformity and trajectory tracking constraints as the baseline performance.

C. Simulation setup

RobotStudio [25] is an ABB robot dynamics simulator that provides close to real trajectory output. We use a Python interface [26] to the RobotStudio virtual controller to execute motion command directly. In order to achieve high constant speed, the commanded trajectory is extended at the first and last motion segments: MoveL and MoveJ are extended in Cartesian and joint space respectively, and MoveC is extend along the arc and linearly in angle-product orientation. For all motion commands, we start with a blending zone of 5 mm and gradually increase it until the speed profile converges such that the robot does not slow down around waypoints due to blending.

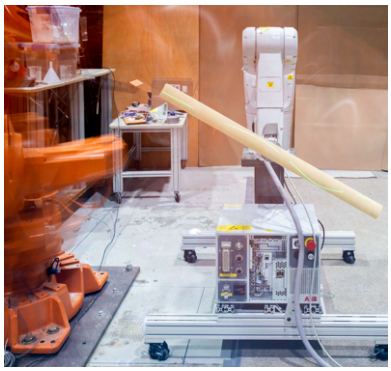
D. Experiment setup

Fig. 6 shows the experiment setup, with the ABB6640 robot holding a mock spray gun. Joint trajectory $\{\mathbf{q}_k\}_{k=0}^K$ is recorded through robot controller at 250 Hz. Robot forward kinematics is used to compute the TCP location

According to the manual of ABB 6640, the path repeatability is up to 1.06 mm [23]. To address this issue, for each set of primitive command, we run the robot 5 times and take the interpolated average of 5 recorded trajectories as the execution trajectory.

E. Results

For baseline execution, in order to satisfy tracking accuracy requirement, we apply bisection search on the commanded speed until the accuracy requirement is met. For waypoints adjustment, we run the waypoint iteration algorithm to reduce the tracking error until convergence. If



(a) Curve 1: Frequency-changing sinusoidal curve on a parabolic surface.

(b) Curve 2: Generic Fan Blade leading edge curve.

Fig. 6: Experiment setup with ABB1200 hold the part and ABB6640 holding a laser pointer with long exposure.

maximum tracking error does not meet the requirement, then further decrease the commanded speed (and repeat the waypoint iteration) until the tracking accuracy requirement is met.

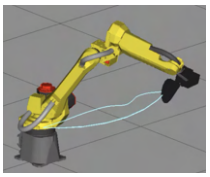
Curve 1	$max(\mathbf{p}_{err})$ mm	$max(\mathbf{n}_{err})$ deg	$\mu(v)$ mm/s	$\sigma(v)$ mm/s
Baseline (sim)	0.49	0.18	124.01	1.02
Baseline (real)	0.46	0.21	103.11	0.85
Wp Adjustment (sim)	0.16	0.57	251.29	4.77
Wp Adjustment (real)	0.38	0.56	248.77	4.33
Optimized (sim)	0.48	0.57	399.81	6.47
Optimized (real)	0.41	0.33	395.72	9.53
Curve 2	$max(\mathbf{p}_{err})$	$max(\mathbf{n}_{err})$	$\mu(v)$	$\sigma(v)$
Baseline (sim)	0.42	0.26	406.10	0.92
Baseline (real)	0.47	0.28	299.97	2.36
Wp Adjustment (sim)	0.24	0.33	1089.64	51.88
Wp Adjustment (real)	0.42	0.41	973.12	13.87
Optimized (sim)	0.42	1.09	1207.45	11.46
Optimized (real)	0.46	1.15	1197.43	13.33

TABLE II: Results for RobotStudio Simulation and Physical Robot Experiments. **Wp Adjustment**: Waypoints Adjustment from baseline results through iterative updates. **Optimized**: Waypoints adjustment with optimized curve pose and Tool-z orientation.

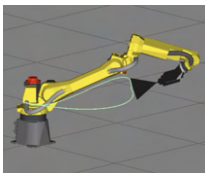
From the baseline results, we can see that without any optimization, it is necessary to slow down in order to achieve higher accuracy. Simply adjusting waypoints could help with increasing path speed and minimizing the tracking error. However, we can increase the path speed further by optimizing the all redundancies with robot joint velocity and acceleration constraints. The final optimized results show 222% (sim) 284% (real) speed increase for Curve 1 and 197% (sim) and 299% (real) speed increase for Curve 2 while keeping the tracking accuracy within the requirement. In the iterative update process, the algorithm prioritizes to compensate position error if the orientation constraint is satisfied, so the final result may result in an increase in orientation error.

F. Implementation for a FANUC Robot

The same algorithm has also been applied to FANUC M10iA robot using the FANUC robot simulation program RoboGuide and a physical robot testbed. Because of the robot motion limit, we used a 85% scaled down Curve 2 for the FANUC robot in simulation and a half of Curve 2 (scaled) for the physical robot. The result is showed in Table III. The speed increased by 489% for Curve 1, 371% for Curve 2 (scaled) and 248% (sim) 149% (physical) for Curve 2 (scaled-half) while keeping tracking accuracy within the requirement. This shows that the framework successfully generalized to another robot model without fine tuning of the parameters.



(a) Curve 1



(b) Curve 2

(c) Real Robot

Fig. 7: FANUC M10iA robot in RoboGuide and the physical robot testbed.

Curve 1	$max(\mathbf{p}_{err})$ mm	$max(\mathbf{n}_{err})$ deg	$\mu(v)$ mm/s	$\sigma(v)$ mm/s
Baseline (sim)	0.14	0.10	55.54	2.22
Wp Adjustment (sim)	0.46	0.28	174.20	8.27
Optimized (sim)	0.48	0.63	327.15	15.75
Curve 2 (scaled)	$max(\mathbf{p}_{err})$	$max(\mathbf{n}_{err})$	$\mu(v)$	$\sigma(v)$
Baseline (sim)	0.25	0.34	144.15	4.95
Wp Adjustment (sim)	0.47	0.46	239.74	8.65
Optimized (sim)	0.39	1.50	679.12	33.64
Curve 2 (half scale)	$max(\mathbf{p}_{err})$	$max(\mathbf{n}_{err})$	$\mu(v)$	$\sigma(v)$
Baseline (sim)	0.42	0.37	334.63	2.39
Baseline (physical)	0.45	0.38	390.05	1.19
Optimized (sim)	0.23	0.89	1167.18	9.31
Optimized (physical)	0.49	0.40	971.72	7.09

TABLE III: Result Comparison for FANUC robot

V. Conclusion and Future Work

This paper presents a complete procedure for optimal industrial robot tracking of high curvature spatial curves with high speed and high accuracy. Our current experiment setup determine TCP location based on the robot forward kinematics. We are working on implementing the approach for direct TCP measurement (e.g., using motion capture). We are also extending the approach to dual-arm systems where the performance is based on the relative TCP frames.

ACKNOWLEDGMENT

Research was sponsored by the Office of the Secretary of Defense and was accomplished under Agreement Number W911NF-17-3-0004. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of the Secretary of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The authors would also like to thank Santiago Paternain and Jonathan Fried for helpful discussion of the project.

REFERENCES

[1] H. Chen, T. Fuhlbrigge, and X. Li, "Automated industrial robot path planning for spray painting process: A review," in *2008 IEEE International Conference on Automation Science and Engineering*, 2008, pp. 522–527.

[2] X. Li, X. Li, S. S. Ge, M. O. Khyam, and C. Luo, "Automatic welding seam tracking and identification," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7261–7271, 2017.

[3] S. Chen, Z. Wang, A. Chakraborty, M. Klecka, G. Saunders, and J. Wen, "Robotic deep rolling with iterative learning motion and force control," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5581–5588, 2020.

[4] K. Ma, X. Wang, and D. Shen, "Design and experiment of robotic belt grinding system with constant grinding force," in *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, 2018, pp. 1–6.

[5] I. M. Nault, G. D. Ferguson, and A. T. Nardi, "Multi-axis tool path optimization and deposition modeling for cold spray additive manufacturing," *Additive Manufacturing*, vol. 38, p. 101779, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214860420311519

[6] Motoman, Incorporated, *Inform II User's Manual*.

[7] ABB Robotics, *Technical reference manual RAPID Instructions, Functions and Data types*.

[8] FANUC Robotics America, Inc., *FANUC Robotics SYSTEM R-J3iB Controller KAREL Reference Manual*.

[9] Adept Technology, Inc., *V+ Language User's Guide*.

[10] Staubli Faverges , *VAL3 REFERENCE MANUALV+ Language User's Guide*.

[11] KUKA Roboter GmbH, *SOFTWARE KR C1 / KR C2 / KR C3 Reference Guide*.

[12] Yaskawa America, Inc., *MotoPlus Software Development Kit*, 2022. [Online]. Available: https://www.motoman.com/en-us/products/software/development/motoplus-sdk

[13] ABB Robotics, *Application manual Externally Guided Motion*.

[14] Staubli Faverges , *Open interfaces to all control systems.* [Online]. Available: https://www.staubli.com/en-us/robotics/emo2-019/openinterface-to-all-controlsystems

[15] M. Schöpfer, F. Schmidt, M. Pardowitz, and H. Ritter, "Open source real-time control software for the kuka light weight robot," in *2010 8th World Congress on Intelligent Control and Automation*, 2010, pp. 444–449.

[16] RoboDK Inc. , *Simulate Robot Applications: Program any Industrial Robot with One Simulation Environment*. [Online]. Available: https://www.robodk.com

[17] ——, *Robot Machining with RoboDK*.

[18] Hypertherm, Inc., *RobotMaster:CAD/CAM for robots (Off-Line Programming)*. [Online]. Available: https://www.robotmaster.com,

[19] OCTOPUZ, *RobotMaster:CAD/CAM for robots (Off-Line Programming)*. [Online]. Available: https://www.octopuz.com/,

[20] ROS-Industrial, "Tesseract," 2022. [Online]. Available: https://github.com/tesseract-robotics/tesseract

[21] J. Schulman and the Robot Learning Lab, "trajopt: Trajectory optimization for motion planning," 2013. [Online]. Available: https://rll.berkeley.edu/trajopt/doc/sphinx_build/html/

[22] K. Min, C.-H. Lee, C. Yan, W. Fan, and P. Hu, "Six-dimensional b-spline fitting method for five-axis tool paths," *The International Journal of Advanced Manufacturing Technology*, vol. 107, no. 5, pp. 2041–2054, Mar 2020. [Online]. Available: https://doi.org/10.1007/s00170-020-05139-7

[23] ABB Robotics, *Product specification IRB 6640*.

[24] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec 1997. [Online]. Available: https://doi.org/10.1023/A:1008202821328

[25] ABB Robotics, *Operating Manual RobotStudio*.

[26] J. Wason, "abb_motion_program_exec," 2022. [Online]. Available: https://github.com/johnwason/abb_motion_program_exec