Jerk-bounded Position Controller with Real-Time Task Modification for Interactive Industrial Robots

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC0

SUBMISSION DATE / POSTED DATE

23-01-2022 / 28-01-2022

CITATION

Liu, Ruixuan; Chen, Rui; Sun, Yifan; Zhao, Yu; Liu, Changliu (2022): Jerk-bounded Position Controller with Real-Time Task Modification for Interactive Industrial Robots. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.18866423.v1>

DOI

[10.36227/techrxiv.18866423.v1](https://doi.org/10.36227/techrxiv.18866423.v1)

Jerk-bounded Position Controller with Real-Time Task Modification for Interactive Industrial Robots

Ruixuan Liu, Rui Chen, Yifan Sun, Yu Zhao and Changliu Liu

Abstract—Industrial robots are widely used in many applications with structured and deterministic environments. However, the contemporary need requires industrial robots to intelligently operate in dynamic environments. It is challenging to design a safe and efficient robotic system with industrial robots in a dynamic environment for several reasons. First, most industrial robots require the input to have specific formats, which takes additional efforts to convert from task-level user commands. Second, existing robot drivers do not support overwriting ongoing tasks in real-time, which hinders the robot from responding to the dynamic environment. Third, most industrial robots only expose motion-level control, making it challenging to enforce dynamic constraints during trajectory tracking. To resolve the above challenges, this paper presents a jerk-bounded position control driver (JPC) for industrial robots. JPC provides a unified interface for tracking complex trajectories and is able to enforce dynamic constraints using motion-level control, without accessing servo-level control. Most importantly, JPC enables real-time trajectory modification. Users can overwrite the ongoing task with a new one without violating dynamic constraints. The proposed JPC is implemented and tested on the FANUC LR Mate 200id/7L robot with both artificially generated data and an interactive robot handover task. Experiments show that the proposed JPC can track complex trajectories accurately within jerk limits and seamlessly switch to new trajectory references before the ongoing task ends.

Index Terms—Industrial robot control, Jerk-bounded position control, Online trajectory modification.

I. INTRODUCTION

Industrial robots are widely used in many applications with structured and deterministic environments, *i.e.*, automotive assembly, pick and place, material removal, furniture fabrication, 3D printing [4, 12]. However, industrial robots are rarely applied in unstructured and dynamic environments, such as human-robot interaction (HRI) [3]. This is mainly caused by the difficulty of designing and implementing a safe and efficient robotic system in such scenarios. In this paper, we aim to fill the gap between industrial robots and dynamic task environments by proposing a unified robot control driver. The difficulty of this task mainly lies in three aspects: fixed robot-specific input formats, online task over-writing, and motion-level control with dynamic constraints. Next, we elaborate

Fig. 1: Illustration of the real-time task modification. The first user command P^{t_i} starts at t_i . A new command $P^{t_{i+1}}$ arrives at t_{i+1} . The JPC modifies the planned joint position trajectory from $t_{i+1} + \phi$ to track $P^{t_{i+1}}$, and the rest of P^{t_i} is aborted. The transition delay ϕ upper bounds the trajectory computation time of JPC. In this way, JPC plans the new trajectory from a valid future robot state and enforces dynamic constraints during task transition.

on each of these aspects and motivate our unique controller design.

First, the control interfaces of most industrial robots only accept input in certain formats (*e.g.*, specific sampling frequency and length). In robotic applications, however, user commands are often at task level and can have flexible formats. Thus, researchers usually need to spend non-trivial efforts to convert user commands to robot-specific formats for execution. In this paper, we aim to lift such efforts by developing an interface that translates task-level commands to low-level commands directly compatible with the hardware.

Second, existing robot drivers do not support overwriting ongoing tasks in real-time, which hinders the robot from responding to the dynamic environment. With current control interfaces, most industrial robots cannot execute a new task until the ongoing task is finished or force stopped. However, the overwriting feature is critical for the robot to operate safely in a dynamic environment since the robot should be able to respond to surrounding changes (*e.g.*, human movements) and adjust its behaviors accordingly. To ensure safety, most existing industrial robots either only work in highly-controlled spaces or simply halt upon unexpected environmental changes. For example, some robots are only used within designated work cells with fences to separate from workers

This work is in part supported by Siemens. The robot arm is donated by FANUC Corporation.

R. Liu, R. Chen, and C. Liu are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA. (412)-268-3818. ruixuanl, ruic3, cliu6@andrew.cmu.edu

Y. Sun is with the Carnegie Institute of Technology, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA. yifansu2@andrew.cmu.edu

Y. Zhao is with FANUC America Corporation. 29700 Kohoutek way, Union City, CA, 94587. yu.zhao@fanucamerica.com

Fig. 2: Robot control loops in task, motion, and servo levels. The only available robot interface is at motion-level, while the servo control interface is hidden. Hence, the robot hardware, as well as the integrated servo-level controller, can be jointly considered as a black box that accepts joint position trajectories with bounded jerk and returns robot states.

[9]. Another common approach is to stop when detecting potential collisions [11]. This is highly inefficient, especially when more interactions are anticipated to happen between humans and industrial robots in the factories [6]. Ideally, the robot should simply perturb its originally planned motions to avoid collisions. Therefore, in order to operate safely and efficiently, it is critical that industrial robots can overwrite their tasks in real-time while maintaining tracking performance (see Fig. 1). To the best of our knowledge, ROS control [2] is the only publicly available control framework that supports task overwriting. However, the trajectory modification is executed at a low priority in the ROS control hierarchy and cannot achieve real-time task overwriting. In this paper, we propose an industrial-compatible robot control driver that supports real-time trajectory modification with minimal transition time and enforced dynamic constraints. Relatedly, [5] proposes a method to generate motion trajectories for mechanical systems during runtime. There is a major difference between the objectives of their work and ours. [5] focuses on *online trajectory generation* which concerns about instantaneous response to sensor events, while this paper focuses on *real-time trajectory modification* which concerns about satisfying dynamic constraints during task transitions.

Third, most industrial robots only expose motion-level control, making it challenging to enforce dynamic constraints when executing user commands and overwriting current user commands with new ones. To illustrate this challenge, let us consider the different robot control loops in a typical interactive task as shown in Fig. 2. In an interactive task (e.g., robot handover), some task-level controller first generates user commands, which can be waypoints with arbitrary sparsity and frequency. Then, a motion-level controller converts user commands to high-frequency joint position trajectories. Finally, a servo-level controller computes proper joint torques and applies to the robot. Meanwhile, the robot can only operate within some dynamic constraints and will come to a hard stop when the constraints are violated. In this paper, we consider jerk-bounded robots. Such constraints can be easily satisfied if one can directly send torque commands via the servo control

interface (e.g., by saturating the control input). However, the servo-level control for industrial robots is often hidden. What is generally available instead is the motion-level position control. Hence, our robot driver needs to operate on motion level while satisfying the dynamic constraints, treating the robot hardware as well as the embedded servo-level controller jointly as a black box (see Fig. 2). In other words, we desire a motion-level controller that converts user commands to jerk-bounded joint position trajectories. In literature, jerk-bounded planning has been widely studied [1, 7, 8, 10]. In this paper, we base our work on a naive approach and focus on the problem of real-time trajectory modification without violating jerk constraints. As future work, we will adapt our trajectory modification design to other jerk-bounded planners.

Combining the above motivations, this paper proposes a jerk-bounded position control driver (JPC) for industrial robots. First, it provides a unified programming interface for the users to control the robot with complex trajectory commands. The input to JPC is a high-level trajectory containing position waypoints for the robot to track. Then, JPC calculates an appropriate joint position trajectory with bounded jerk. Second, JPC enables online trajectory modification, which allows the users to overwrite the ongoing task with minimal transition time to new tasks while satisfying the jerk bound. Thus, the robot can seamlessly adapt its motions to dynamic environments and operate safely. The proposed JPC is implemented and tested on the FANUC LR Mate 200id/7L, which supports high-bandwidth position control. We test JPC both with artificially generated tracking references and in an interactive robot handover task. The experiment results demonstrate that the proposed JPC can achieve accurate trajectory tracking as well as smooth trajectory switching on the FANUC robot. The JPC can also be easily transferred to other industrial robots with a similar high-bandwidth position control interface.

II. PROBLEM FORMULATION

The motion-level controller (in Fig. 2) takes in the user commands at different time steps $(t_0, t_1, t_2, \dots)$, which may not be sent at the same frequency, i.e., $t_{i+1} - t_i$ is not a constant. At every t_i , the user command P^{t_i} composes of a set of joint trajectory points where $P^{t_i} = [P_1^{t_i}; P_2^{t_i}; \dots; P_{N^{t_i}}^{t_i}]$ and N^{t_i} is the number of waypoints. These waypoints have sampling frequency $1/T^{t_i}$. The commands P^{t_i} and $P^{t_{i+1}}$ may have overlaps, in the sense that $t_i + N^{t_i}T^{t_i} > t_{i+1}$. In this case, the new user command will overwrite the old user command at the time when overlap happens, as shown in Fig. 1. Command overwriting is essential in many interactive robot tasks. For instance, when the robot is handing a tool over to a human and the human suddenly moves his/her hand, the task-level controller will compute and send a new user command. Then, the robot should immediately overwrite the ongoing command with the new one in order to finish the handover.

The goal of the motion-level controller is to translate the user command P^{t_i} to acceptable jerk-bounded trajectories (see Fig. 3), $u^{t_i} = JPC(P^{t_i})$, that is executable by the

Fig. 3: Jerk-bounded Trajectory Tracking Illustration. Given a task P^{t_i} at t_i and $M = 5$, the bounded control jerk J^{t_i} is shown in green. The position control u^{t_i} is shown in black.

robot through a motion-level interface (shown in Fig. 2). The motion-level interface takes in joint position signals with sampling time dt , which is usually much smaller than T^{t_i} . The tracking of the position command in the servo-control level (though which is unknown to us) is assumed to be perfect as long as the jerk is bounded.

To realize the goal, we need to solve the following two problems: 1) turn P^{t_i} into a jerk-bounded trajectory with sampling time dt such that the waypoints in P^{t_i} are perfectly tracked; and 2) modify an existing jerk-bounded trajectory as soon as a new $P^{t_{i+1}}$ is received and still ensure boundness on jerk during the transition. The challenge is limited computation time due to short sampling time dt (the robot will enter a hard stop if no position command is received in the next sampling time). In the following discussion, we introduce the jerk-bounded position controller which is able to achieve the goal under the aforementioned challenge. For clarity, we will use superscript to denote the global time, i.e., P^{t_i} denotes the task received at time t_i . We use subscript to denote the task time step, i.e., $P_1^{t_i}$ denotes the first time step of the task received at t_i .

III. JERK-BOUNDED POSITION CONTROL DRIVER

A. Jerk-bounded Trajectory Tracking

Given a user command P^{t_i} with sampling time T^{t_i} for an n -DOF robot, we compute the position control sequence by up-sampling P^{t_i} into jerk-bounded u^{t_i} with sampling time dt . For the rest of this section, we omit the superscript t_i since the method applies to user command P received at any time. We model the dynamics of robot joints using a discrete-time jerk control system in order to enforce the jerk bound. In addition, we apply the same trajectory up-sampling procedure to each robot joint independently. As such, we omit the joint index in the following discussions and focus on an arbitrary robot joint of interest.

The state of a robot joint is denoted as $x = [p; v; a]$, where p , v , and a are the joint position, velocity, and acceleration.

We have the robot dynamics as

$$x_{k+1} = \underbrace{\begin{bmatrix} 1 & \tau & \frac{1}{2}\tau^2 \\ 0 & 1 & \tau \\ 0 & 0 & 1 \end{bmatrix}}_{A(\tau)} x_k + \underbrace{\begin{bmatrix} \frac{3}{2}\tau^3 \\ \frac{1}{2}\tau^2 \\ \tau \end{bmatrix}}_{B(\tau)} j_k, \quad (1)$$

where τ is the time increment and j_k denotes the jerk control input at timestep k . It is likely to be infeasible to travel between consecutive waypoints using a constant jerk. Therefore, we up-sample P by a factor M and allow at maximum M different jerk commands in each time interval T . For trajectory tracking, we want the robot to track the positions of the waypoints in P , with unknown intermediate velocity and acceleration. The velocity and acceleration at the goal point are set to 0. Thus, by setting the robot initial state to be x_0 and $\tau = dt$, we aim to find a sequence of jerk commands J such that the following equality holds:

$$\underbrace{\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_N \\ 0 \\ 0 \end{bmatrix}}_{\hat{P}} = \hat{A}x_0 + \hat{B} \underbrace{\begin{bmatrix} j_0 \\ j_1 \\ \vdots \\ j_{M_1-1} \\ j_{M_1} \\ \vdots \\ j_{N \cdot M \cdot M_1 - 1} \end{bmatrix}}_J, \quad (2)$$

where $J \in \mathbf{R}^{N \cdot M \cdot M_1 \times 1}$ and $M_1 = \frac{T}{dt \cdot M} \in \mathbf{Z}^+$. Since we allow M different jerk commands in each interval T , we apply each jerk command for M_1 times. Namely, in (2), $j_{bM_1} = j_{bM_1+1} = \dots = j_{(b+1)M_1-1}$ for $b \in \{0, 1, \dots, NM - 1\}$. $\hat{A} \in \mathbf{R}^{(N+2) \times 3}$ and $\hat{B} \in \mathbf{R}^{(N+2) \times (N \cdot M \cdot M_1)}$ are the matrices for the up-sampled system, which can be computed by lifting $A(\tau)$ and $B(\tau)$ as

$$\hat{A} = \begin{bmatrix} CA(\tau)^M \\ CA(\tau)^{2M} \\ \vdots \\ CA(\tau)^{(N-1)M} \\ A(\tau)^{NM} \end{bmatrix}, \quad \hat{B} = \begin{bmatrix} CA(\tau)^{MM_1-1}B(\tau) & CA(\tau)^{MM_1-2}B(\tau) & \dots & 0 \\ CA(\tau)^{2MM_1-1}B(\tau) & CA(\tau)^{2MM_1-2}B(\tau) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ A(\tau)^{NMM_1-1}B(\tau) & A(\tau)^{NMM_1-2}B(\tau) & \dots & B(\tau) \end{bmatrix}, \quad (3)$$

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}.$$

We obtain a least square solution of (2) by solving the following optimization:

$$J = \underset{J}{\operatorname{argmin}} \|\hat{B}J - \hat{P} + \hat{A}x_0\|. \quad (4)$$

The jerk-bounded state sequence for tracking the high-level trajectory P is calculated by propagating J as shown in (1).

Finally, the JPC outputs the position control sequence by extracting the positions from the state sequence as

$$u = JPC(P) = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{N \cdot M \cdot M_1} \end{bmatrix}. \quad (5)$$

Figure 3 illustrates an example of trajectory tracking by JPC with $M = 5$. Given a user command P^{t_i} at time t_i , we can calculate a bounded jerk profile J^{t_i} (green) for trajectory tracking by solving (4). In addition, we can calculate the jerk-bounded trajectory u^{t_i} (black), velocity profile (blue), and acceleration profile (purple) by propagating J^{t_i} using (1). Since the example has $M = 5$, we have 5 different jerk values for completing the tracking.

Next, we discuss the computational complexity and the effect of different user commands. The runtime of JPC is dominated by the computation of jerk control sequence J using (4), which is a least-square solution to (2). (4) can be solved in $\mathcal{O}(N^3 M^3)$ if one re-writes the linear system and only preserves the $N \cdot M$ unique jerk commands in J . The JPC computation time should be less than dt in order for JPC to run at maximum frequency. Since the JPC runtime scales with respect to both the length N and granularity factor M , users should try to compose short commands in order to gain higher control flexibility (with a large M). In general, a large M leads to better tracking performance but longer computation time. Finally, given the waypoint sampling time T (if not tunable), user commands should also select M such that $M_1 = \frac{T}{dt \cdot M} \in \mathbf{Z}^+$, and each different jerk command is executed for some positive integer times. An appropriate combination of T , N and M is discussed in section IV based on our hardware system.

B. Real-time Trajectory Modification

The real-time trajectory modification is illustrated in Fig. 1. Suppose at time t_{i+1} , the robot is executing an ongoing task u^{t_i} (blue). The user sends a new task $P^{t_{i+1}}$ (red points) to the robot. In order to track the new task, we need to calculate a jerk-bounded control $u^{t_{i+1}} = JPC(P^{t_{i+1}})$ that enables the robot to seamlessly transition to the new tracking task (red). Note that in order to calculate $u^{t_{i+1}}$, we need to obtain the robot initial state x_0 as shown in (4). An inappropriate x_0 would generate unsmooth task transition and violate the jerk bound. For a regular tracking task, x_0 is the robot current state. However, when doing real-time trajectory modification, since the robot is in motion during the trajectory computation, the actual initial state of the robot when the new trajectory is computed and ready to be executed would be different from the current robot state when the computation is initiated. As a result, the jerk limit would be violated if the new trajectory is computed using the current robot state as the initial state. Therefore, the major challenge becomes selecting an appropriate initial robot state x_0 for calculating $u^{t_{i+1}}$.

This paper presents a prediction-based real-time trajectory modification method. Assume the JPC receives the new user

Fig. 4: Jerk-bounded Position Controller Implementation Diagram. The JPC is deployed to the Speedgoat real-time target machine, which is connected in between of the host and the robot. Right: FANUC LR Mate 200id/7L. Black-bounded modules: 1kHz. Red-bounded modules: 125Hz.

command at t_{i+1} , we set a prediction horizon ϕ and estimate the future robot state $x^{t_{i+1}+\phi}$. The tracking control u' for the new task is then calculated with $x_0 = x^{t_{i+1}+\phi}$ in (4). The new control sequence, $u^{t_{i+1}}$, is constructed by concatenating the residual control from t_{i+1} to $t_{i+1} + \phi$ (blue curve between the red and black dashed lines in Fig. 1) and u' (red curve).

However, it is difficult, if not impossible, to accurately estimate the robot future state. Figure 4 demonstrates the controller design, in which a decision buffer module (yellow) implements the real-time trajectory modification. Note that (5) implements an open-loop control, in which the robot state sequence is known. Suppose a task comes at time t_i , once the state sequence x^{t_i} is calculated (solve (4) and propagate J^{t_i} using (1)), we store x^{t_i} in the buffer and send the corresponding control signal $u_j^{t_i}$ to the robot sequentially. Suppose the new task comes at t_{i+1} , the next control to be sent is $u_j^{t_i}$. Due to the buffer design, we can easily obtain $x_0 = x^{t_{i+1}+\phi} = x_{j+\phi/dt}^{t_i}$ by looking forward into the buffer. The decision buffer then discards the executed sequence (Fig. 1 blue curve before the vertical red dashed line) and the aborted sequence (Fig. 1 blue dashed line). The new buffer is constructed as $x^{t_{i+1}} = [x_j^{t_i}; \dots; x_{j+\phi/dt}^{t_i}; x']$, where x' is computed by propagating u' . The control for executing the new task, $u^{t_{i+1}}$, is extracted from the buffer sequentially.

Note that the prediction horizon ϕ is tunable. Intuitively, ϕ determines how fast the robot transitions from the old task to the new task, and is regarded as the *transition delay* of JPC. With a larger ϕ , the robot would have a larger delay in transitioning to the new task. Ideally, we want to choose the smallest ϕ in order to achieve fast transition. However, it is required that ϕ is large enough so that $u_{j+\phi/dt}^{t_i}$ (the connection point) will be executed after $u^{t_{i+1}}$ is calculated and the new buffer $x^{t_{i+1}}$ is constructed. An analysis of ϕ in our system implementation will be discussed in section IV.

IV. HARDWARE IMPLEMENTATION

This paper implements the JPC on the FANUC LR Mate 200id/7L. The FANUC robot provides a high-bandwidth position control interface via Ethernet (called *stream motion*),

which requires a 125Hz position control sequence, $dt = 0.008s$, with bounded jerk. In addition, according to our practical experience, the position control interface requires a 1kHz communication for stable performance. It is difficult, if not impossible, for a general-purposed computer to maintain such a high communication frequency. Therefore, instead of directly connecting the host computer to the robot, we have a Speedgoat baseline real-time target machine (SG) in between as shown in Fig. 4. The host computer and the SG has query-based Ethernet communication. The SG and the robot has a stable Ethernet communication at 1kHz. The JPC in section III is deployed to the SG. Note that this paper uses the FANUC robot, but the JPC is also applicable to other industrial robots with similar control interfaces after minor adaptation to the actual control frequency and dynamic constraints.

Fig. 4 shows the implementation diagram of the JPC on the SG. The controller is implemented using MATLAB Simulink. Since the algorithms discussed in section III need non-trivial computation time while the robot platform requires 1kHz communication frequency, we implement a parallel design of JPC on SG. The modules with black borderlines run at 1kHz, and the modules with red borderlines run at 125Hz.

- 1) *User Command*: The user command module, shown in pink, is the communication interface between the controller running on SG and the host.
- 2) *Send/Receive Command*: The send and receive command modules, shown in green, are the communication interface between the controller and the robot.
- 3) *State Machine*: The state machine module, shown in blue, tracks the controller and robot states and sends the feedback to the host computer.
- 4) *Trajectory Computation*: The trajectory computation module, shown in purple, computes the position control sequence discussed in section III-A.
- 5) *Decision Buffer*: The decision buffer module, shown in yellow, implements the online trajectory modification discussed in section III-B.

In our application scenarios, the computation time is usually less than $8ms$. Therefore, we have the trajectory computation module running at 125Hz. The JPC can solve user tasks with $T = 1s$, $M = 25$, $N = 8$ without timeout violation. As we increase N to 10, the computation would occasionally violate the time limit. If we decrease M to 5 and maintain $N = 10$, the module runs without time violation. However, a smaller M would lead to worse tracking performance. Therefore, it is important to select appropriate parameters based on the hardware and the application. For applications that require longer computation time, we need to either tune the hyperparameters in the trajectory computation module, or decrease the module frequency (increase the allowed computation time). In addition, recall that we should select ϕ to be no less than the computation time of JPC to ensure the validity of initial state of the new trajectory. As mentioned previously, our trajectory computation usually takes less than $8ms$. Hence technically, ϕ can be as small as $8ms$. In our implementation, we set

$\phi = 40ms$ to be conservative. For applications requiring longer computation time, ϕ should increase accordingly.

V. EXPERIMENT RESULTS

In this section, we benchmark the tracking performance of JPC on FANUC robot under various scenarios to show that JPC can (a) convert user-provided trajectories into valid position control sequences and (b) overwrite ongoing tasks with new ones in real time. To illustrate (a), we program the robot to track both step reference and various sinusoidal references and examine the tracking error. To illustrate (b), we send new tracking references to the robot while another task is still running. In addition, we perform an HRI handover task, where the human might move the receiving hand at any time, to further illustrate (b). We show how the JPC adapts future position control sequences and seamlessly transition to tracking new references.

A. Trajectory Tracking

We benchmark the tracking performance of JPC by synthesizing two types of input tracking references: step and sinusoidal signals. All references assume zero initial state and are sampled at 5 Hz for each robot joint. The step reference (Case step) has increments of 3 degrees with periods of 2 seconds. For sinusoidal signals, we create four test cases with varying amplitudes A and frequencies ω , as listed below.

- 1) Case sine #1: $A = 5^\circ$, $\omega = 2\pi/10$.
- 2) Case sine #2: $A = 5^\circ$, $\omega = 2\pi/6$.
- 3) Case sine #3: A linearly increases from 4° to 6° halfway and then decreases to 5° , ω increases from $2\pi/15$ to $2\pi/5$ halfway and then decreases to $2\pi/10$.
- 4) Case sine #4: A linearly increases from 4° to 6° halfway and then decreases to 5° , ω increases from $2\pi/11$ to 2π halfway and then decreases to $2\pi/6$.

We set $M = 25$ and $T = 0.2s$ (determined by the sampling frequency). N is determined by the length of the task and the value of T . We visualize the trajectory tracking performance of all test cases in Fig. 5. The blue points are the user-specified waypoints for the robot to track. The red plot shows the control reference calculated by JPC. For each task, all the joints have the same task. Therefore, we only plot one joint for each task in Fig. 5. We can see from the figures that all the red lines (JPC-calculated control reference) align with the blue waypoints. This indicates that JPC tracks the user-specified task accurately.

To quantitatively evaluate the tracking performance, we report the tracking error as shown in table I. The error is reported using the absolute error between the user-specified waypoints and the JPC-calculated control reference at the timesteps of the waypoints, *i.e.*, $0.2s$, $0.4s$, etc. Table I indicates that when having $M = 25$, JPC can accurately track the user-specified trajectory. The tracking errors for all test cases are smaller than 10^{-11} degrees, and thus, are negligible in table I. To study the influence of the hyperparameter M on JPC tracking, we decrease M to 5 and compare the JPC performance. From table I, the tracking errors are significantly larger than the

Fig. 5: Tracking performance of JPC. Blue points: User-specified input waypoints for tracking. Red: JPC-calculated control references.

Fig. 6: Real-time task overwriting performance of JPC. Left: Case overwrite #1. Middle: Case overwrite #2. Right: Case overwrite #3. Blue points: User-specified input waypoints for the first task. Red points: User-specified input waypoints for the second task. Blue solid line: Executed JPC-calculated control reference for the first task. Blue dashed line: aborted JPC-calculated control reference for the first task. Red solid line: Executed JPC-calculated control reference for the second task. Black: JPC-controlled robot trajectory.

errors when $M = 25$, though the errors are still smaller than 1° . The error in case Sine #4 is the largest since the task is the most complicated among all. However, the computation time ($0.04 \sim 0.05s$) is around 4 times shorter than the time when $M = 25$ ($\sim 0.15s$). Therefore, there exists a clear tradeoff between the tracking accuracy and the computation time when selecting M . It is important to adjust M based on the need of the application scenario.

B. Real-time Task Overwriting

In the previous section, we send the entire trajectory to the robot for execution. In this section, we demonstrate the real-time task overwriting feature of JPC. Specifically, we select three test case pairs from the previous section but with lower sampling rate (1Hz). For each test pair, we sequentially send the first test case followed by the second test case. The first test cases in all the test pairs have 6 sampled waypoints, and thus, the regular execution time is 6 seconds. In order to demonstrate the real-time task overwriting feature, we send the second test case 2.5 seconds after the first test is sent. Thus, the first

Test Case	Min.	$M = 5$		Min.	$M = 25$	
		Max.	Avg.		Max.	Avg.
Step	0	0.3394	0.0931	0	0	0
Sine #1	0	0.0698	0.0072	0	0	0
Sine #2	0	0.1258	0.0231	0	0	0
Sine #3	0	0.1329	0.3777	0	0	0
Sine #4	0	3.0442	0.0931	0	0	0

TABLE I: Absolute tracking error (deg) with different parameter M . In each column, the value is the absolute error between the user-specified waypoints and the JPC-calculated control reference. Errors are negligible when $M = 25$. The actual values are within $10^{-12} \sim 10^{-11}$ degrees range.

tracking task is not yet finished and will be overwritten by the second task. Ideally, the robot should immediately start to track the new task without coming to a halt. The three test cases are composed as follows.

- 1) Case overwrite #1: First test: Case sine #2. Second test: last 4 waypoints in Case sine #3.
- 2) Case overwrite #2: First test: Case sine #1. Second test:

Fig. 7: Performance of JPC in HRI for each joint. Points: User-specified task waypoints for robot handover. Blue: Delivering with human hand pose in Fig. 8(a). Red: Delivering with human hand pose in Fig. 8(b). Green: Delivering with human hand pose in Fig. 8(c). Magenta: JPC-calculated control references. Black: JPC-controlled robot trajectories. Dashed lines: aborted trajectories.

Fig. 8: The human hand poses in the HRI handover task.

last 4 waypoints in Case sine #4.

- 3) Case overwrite #3: First test: Case step. Second test: last 4 waypoints in Case sine #4.

Note that we only use the last 4 waypoints of the second test because we want to maintain the number of waypoints tracked to be 6.

We visualize all tests in Fig. 6. Note that in each plot, two of the test cases used in Fig. 5 are sent sequentially with overlap. The blue curve indicates the control reference generated from the first task (blue points). The red curve indicates the control reference generated by the second task (red points). Due to the task overlap, the first reference is aborted (where the blue line turns from solid to dashed) when the new task is received. A new control reference is then calculated based on the second task and the robot state. It is evident that even with some task already ongoing, the JPC is able to replace future position control sequences according to newly received tasks. This enables the robot (trajectories shown in black) to seamlessly transition to new tasks on the fly and react to surrounding changes in real time.

C. Real-time Human Robot Interaction

To further demonstrate JPC, we conduct a real-time HRI task: robot handover¹. In the task, the robot is supposed to pass a workpiece to the right hand of a human. As the robot is delivering, the human moves his hand, and thus, the robot needs to overwrite its ongoing tracking reference with newly received tasks in real time. We track the human pose using a Kinect camera and compose tracking reference P by linearly interpolating between current robot state and the real-time right hand position. Figure 8 shows the human's right hand poses at different time. Initially, the hand is close to the robot. As the robot delivers, the human moves his hand away from the robot. We send a new P to JPC every 4 seconds.

Figure 7 demonstrates the performance of JPC in the HRI (robot handover) task. The blue area shows the robot execution with the human hand pose in Fig. 8(a). Before the robot delivers the workpiece, the human moves his hand further as shown in Fig. 8(b). Thus, the robot aborts the first task (blue dashed lines) and start the new task (red). However, the human again moves his hand away as shown in Fig. 8(c). Therefore, the robot aborts the second task (red dashed lines) and delivers to the new position (green). Eventually, the robot successfully delivers the workpiece to the human. From Fig. 7, we can see that JPC can achieve fast and smooth real-time trajectory modification while maintain the tracking performance.

VI. CONCLUSION

This paper proposes a jerk-bounded position control driver (JPC) for industrial robots with several advantages. First,

¹This human-robot interaction study only involves one human performing hand motions. The harm or discomfort anticipated are no greater than those ordinarily encountered in daily life.

it provides a unified offline programming interface for the users to control the robot with complex trajectory commands. The user can directly input the high-level planned trajectory, without any further modification, to the JPC to control the robot. Second, the proposed JPC enables online trajectory modification, which allows the users to overwrite the ongoing task with new tasks without halting the robot. Therefore, the robot can adapt its motions and operate safely in dynamic environments, *i.e.*, during human-robot interactions. Third, the JPC is able to enforce robot-specific dynamic constraints (*i.e.*, bounded jerk) at any time via motion-level control, without access to servo-level control. The experiment results demonstrate that the proposed JPC can achieve accurate trajectory tracking as well as smooth trajectory switching on the FANUC robot.

JPC empowers industrial robots (whose genuine control interfaces are generally inflexible) to perform interactive tasks (where user commands may appear at unpredictable times and frequencies). For future work, we plan to apply JPC to more complex interactive tasks where user commands change in larger scales. For example, in collaborative sorting, the human might choose to pick up an object that is also targeted by the robot. In such case, the task-level command should be updated for the robot to pick up another object, which can cause large discrepancy between the old trajectory and the new. It is worth studying whether the JPC can still perform the online trajectory modifications within dynamic constraints in such cases. Another promising direction is to study the interaction between JPC and task-level controller. For example, when calculating new commands, the high-level decision maker can take into account the JPC transition delay. Finally, it is worth integrating real-time safety measures into JPC. For instance, one can apply safe control algorithms to the jerk commands before computing the position trajectory, to avoid collision with obstacles (*e.g.*, human users interacting with the robot).

REFERENCES

- [1] P. Boscariol and A. Gasparetto. Model-based trajectory planning for flexible-link mechanisms with bounded jerk. *Robotics and Computer-Integrated Manufacturing*, 29(4):90–99, 2013. ISSN 0736-5845. doi: <https://doi.org/10.1016/j.rcim.2012.11.003>.
- [2] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Genaro Raiola, Mathias Lüdtké, and Enrique Fernandez Perdomo. *ros_control*: A generic and simple control framework for ROS. *The Journal of Open Source Software*, 2(20):456 – 456, December 2017.
- [3] Henrik I. Christensen, Nancy M. Amato, Holly A. Yanco, Maja J. Matarić, Howie Choset, Ann W. Drobni, Ken Goldberg, Jessy W. Grizzle, Gregory Hager, John M. Hollerbach, Seth Hutchinson, Venkat N. Krovi, Daniel Lee, Billy Smart, Jeffrey C. Trinkle, and Gaurav S. Sukhatme. A roadmap for us robotics - from internet to robotics 2020 edition. *Found. Trends Robotics*, 8:307–424, 2021.
- [4] Varun Gopinath, Fredrick Ore, and Kerstin Johansen. Safe assembly cell layout through risk assessment – an application with hand guided industrial robot. *Procedia CIRP*, 63:430–435, 2017.
- [5] Torsten Kröger and Friedrich M. Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2010. doi: 10.1109/TRO.2009.2035744.
- [6] Changliu Liu and Masayoshi Tomizuka. Algorithmic safety measures for intelligent industrial co-robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3102, 2016.
- [7] Huashan Liu, Xiaobo Lai, and Wenxiang Wu. Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints. *Robotics and Computer-Integrated Manufacturing*, 29(2):309–317, 2013. ISSN 0736-5845. doi: <https://doi.org/10.1016/j.rcim.2012.08.002>.
- [8] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 19(1): 42–52, 2003. doi: 10.1109/TRA.2002.807548.
- [9] George Michalos, Panagiotis Karagiannis, Nikos Dimitropoulos, Dionisis Andronas, and Sotiris Makris. *Human Robot Collaboration in Industrial Environments*, pages 17–39. Springer International Publishing, 2022.
- [10] K. Petrínek and Z. Kovacic. Trajectory planning algorithm based on the continuity of jerk. In *2007 Mediterranean Conference on Control Automation*, pages 1–5, 2007. doi: 10.1109/MED.2007.4433880.
- [11] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018. ISSN 0957-4158.
- [12] George Q. Zhang, Xiongzi Li, Remus Boca, Jeremy Newkirk, Biao Zhang, Thomas A. Fuhlbrigge, Helen K. Feng, and Nick J. Hunt. Use of industrial robots in additive manufacturing - a survey and feasibility study. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–6, 2014.