2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

UTTG: A Universal Teleoperation Framework via Online Trajectory Generation

Shengjian Fang[1,#], Yixuan Zhou[1,#], Yu Zheng[1], Pengyu Jiang[1], Siyuan Liu[1], and Hesheng Wang[1,*]

***Abstract*— Teleoperation is crucial for hazardous environment operations and serves as a key tool for collecting expert demonstrations in robot learning. However, existing methods face robotic hardware dependency and control frequency mismatches between teleoperation devices and robotic platforms. Our approach introduces a unified interface that automatically extracts kinematic parameters from Unified Robot Description Format (URDF) files, enabling plug-and-play deployment across diverse robotic systems. The proposed interpolation algorithm bridges the frequency gap between low-rate human inputs and high-frequency robotic control commands through online continuous trajectory generation, without requiring access to the closed, bottom-level control loop. To further reduce latency, a joint prediction module is incorporated to anticipate operator intent and compensate for delays. Moreover, we introduce a minimum-stretch spline to optimize motion smoothness and quality. The system supports both precision and rapid operation modes for different task requirements. Experiments on three robotic platforms, including dual-arm setups, demonstrate our framework's generality, smoothness, and responsiveness. Teleoperation latency remains below 50ms at 30Hz input and approaches 15ms at 200Hz input. The code is developed in C++ with a Python interface, and available at https://github.com/IRMV-Manipulation-Group/UTTG.**

I. INTRODUCTION

Teleoperated robotic systems have become indispensable in hazardous environments, including construction sites and radioactive contamination zones, as these systems enhance human operational capacity while ensuring worker safety [1]. Through remote task execution capabilities, such systems effectively mitigate occupational hazards and enhance operational efficiency. Furthermore, the recent advancement of data-driven approaches in robotics has amplified the importance of teleoperation, particularly for imitation learning (IL) frameworks [2]. IL enables robots to acquire dexterous manipulation skills through large-scale human demonstration datasets [3], where teleoperation systems serve as critical tools for curating expert level demonstrations and transmitting human knowledge to robotic platforms [4].

*This work was supported by National Key R&D Program of China (Grant No. 2024YFB4708900). It was also supported in part by the Natural Science Foundation of China under Grant Nos. 62225309, U24A20278, 62361166632. It was sponsored by The Explorers Program of Shanghai (Basic Research Funding) (Grant No. 25TS1415100). (Corresponding Author: Hesheng Wang)

[1] Authors are with the School of Automation and Intelligent Sensing, Shanghai Jiao Tong University and Shanghai Key Laboratory of Navigation and Location Based Services, Shanghai 200240, China. Emails: {fang20021005, yixuanzhou, zhengyu0730, jiangpengyu, siyuan_l, wanghesheng}@sjtu.edu.cn

#These authors contributed to the work equally and should be regarded as co-first authors.

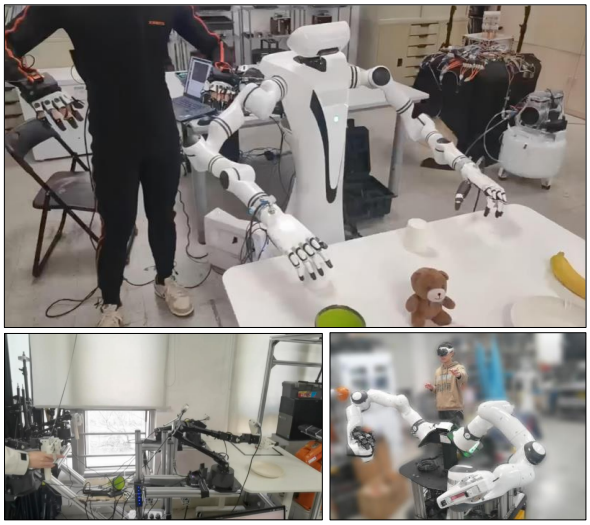


Fig. 1: Demonstration of the proposed framework. Experimental validation was conducted across three robotic platforms.

Recent advancements in teleoperation have primarily focused on two distinct research directions. One develops modular accessory kits [5] to augment commercial robotic platforms, yet such solutions are often hardware-specific, complicating their adaptation to novel robotic platforms and increasing the deployment overhead. The other employs learning-based agents that map human input trajectories to robot motion commands [6], [7]. However, these agents must be painstakingly retrained or fine-tuned for each new robot, as they are inherently coupled to the kinematic and dynamic parameters of the specific hardware they were trained on. These limitations restrict the general applicability and scalability of current methods as robotic platforms diversify. To resolve these challenges, we propose a teleoperation framework featuring two capabilities. First, our method resolves the control frequency mismatch between low-frequency human-input interfaces (e.g. 10-50Hz vision-based skeleton recognition) and high-frequency robotic platforms like Franka (typically 200-1000Hz) thereby supporting cross-platform deployment. Second, the framework provides a set of uniform hardware-agnostic interfaces, enabling pluggable teleoperation across heterogeneous robotic platforms, eliminating the need for platform-specific agent training or simulation. The key contributions of our framework are as follows:

1) An online trajectory generation algorithm is proposed, which utilizes a minimum-stretch cubic spline with assigned two-ordered boundary conditions, to reconcile

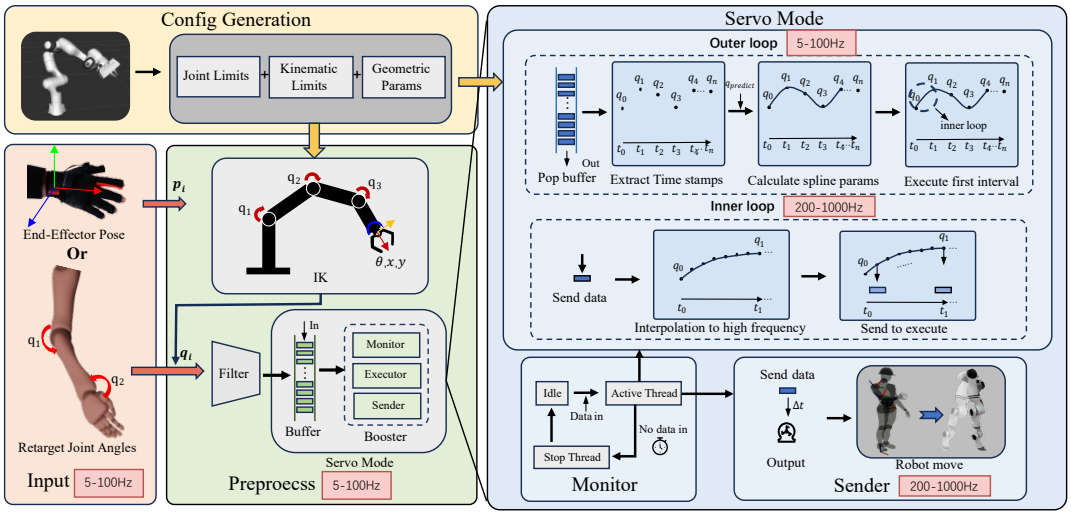


Fig. 2: UTTG framework: We utilize URDF of robot to automatically generate the required parameter files, decoupling human input from specific robot configurations. Our input can be either the end-effector pose or the joint angles. Using the generated files, we perform IK to compute the corresponding joint angles, and the output consists of high-frequency joint position commands for the robot.

heterogeneous frequency interfaces.

2) A unified teleoperation framework is established, incorporating standardized interfaces, URDF-based configuration, kinematic resolution, and predictive modules to ensure platform-agnostic generality.(see Fig. 1)
3) Two adaptive operation modes (rapid and precise) are introduced, validated extensively on real hardware, demonstrating robust performance across diverse tasks, with latency under two time-steps (e.g., 50ms at 30Hz).

II. RELATED WORK

A. Teleoperation

The teleoperation task involves converting human motion to robot joint commands, which introduces significant dependencies on specific devices and robot models. Several control strategies have been proposed to address this issue.

Joint replication directly maps operator movement to robot joint angles, which establishes a one-to-one correspondence between human motion and robot actuation [8], [9]. Two main approaches exist: a master-slave structure and a robot-like exoskeleton design. Both require the teleoperation device and the robot to share similar structures, as the mapping depends on kinematic congruence.

Motion retargeting adjusts operator input to accommodate robot limitations such as joint constraints and workspace boundaries. It typically uses exoskeletons [10] or motion capture systems [11], [12], with data processed by optimization-based algorithms [13]. However, these methods typically operate at low frequencies due to computational overhead from solving constraint optimization problems or processing high-dimensional human motion data.

End effector control and **vision-based control** [14] focus on human hand pose estimation, eliminating need for complex external devices but introducing challenges like computational latency or reduced precision.

A fundamental challenge common to most approaches is that their lower operating frequencies struggle to meet the high-frequency command requirements of standard robot execution layers. Joint replication methods allow high frequency sampling, whereas motion retargeting and vision-based control tend to operate at lower frequencies and often struggle to maintain continuous control.

To overcome these challenges, we develop a simplified and generalized teleoperation framework. The framework maintains the essential continuity and precision required for effective teleoperation, seamlessly handling both low and high-frequency inputs to bridge the gap between human operation and robotic execution, while reducing reliance on particular device and robot configurations.

B. Online Trajectory Generation

A direct method for teleoperation is to design a PD (Proportional-Differential) controller requiring access to the bottom-level actuators. However, it is typically disabled by manufacturers and provided as a specific hardware-dependent feature. Fortunately, the position-level real-time servo interface is a standard capability for modern robotic platforms. Thus, with the help of online trajectory generation, we can achieve teleoperation system evolving time-varying targets.

Traditional trajectory generation methods [15], [16] employ offline computation paradigms that precompute entire trajectories before execution. These approaches fundamentally lack the capacity to handle real-time target updates or inherent system state disturbances in teleoperation scenarios. While existing online trajectory generation solutions partially address this limitation, the approach in [17] exhibits performance degradation in low-frequency operations and is not open-sourced, and functional implementations [18] are often commercial and complex.

Consequently, we propose an open-source trajectory generation method that incrementally constructs motion paths

from the current state to each new target point, continuously updating trajectories to satisfy teleoperation requirements, demonstrating consistent performance even under low-frequency inputs.

III. METHOD

As illustrated in Fig. 2, the framework adopts a modular architecture comprising four components. The **Config Generation** module automatically extracts essential configuration information required for teleoperation from URDF. The **Input** module provides standardized interfaces for two input types: joint angles and end-effector poses, the latter being converted to joint space via an integrated IK solver. Both input streams will undergo filtering to remove unintended human tremors or noise in the **Preprocess** module. The filtered joint targets and their synchronized timestamps are stored in a buffer queue for subsequent trajectory planning.

The **Servo Mode** module performs trajectory generation and execution by retrieving points from the buffer and generating smooth trajectories in precise or rapid modes based on the task requirements. The generated trajectories are dispatched to robotic actuators through standardized position-level interfaces for execution. To maintain stable and continuous trajectory execution, three threads operate in parallel: a monitor thread monitors system status, an executor thread generates trajectories, and a sender thread sends the resulting commands to the motors.

A. Trajectory Generation

To interpolate discrete points, cubic splines are commonly adopted for trajectory interpolation due to their inherent smoothness and continuity. And its low computational cost is also suitable for online trajectory generation.

As shown in [19], by connecting $n+1$ samples with n segments of cubic splines and assigning each interval as T_i for $i \in [0, n-1]$, the expression of the cubic spline on the i-th segment can be written as

$$\mathbf{s}_i(t) = \mathbf{a}_i(\mathbf{m}_i) + \mathbf{b}_i(\mathbf{m}_i)t + \mathbf{c}_i(\mathbf{m}_i)t^2 + \mathbf{d}_i(\mathbf{m}_i)t^3, \qquad (1)$$

where $t \in [0, T_i]$. Following [20], the spline fitting problem can be solved by computing the supplementary variables $\mathbf{m}_i$.

Following the formulation in [19], we consider a specific scenario where the initial and final velocities and accelerations are pre-assigned to ensure boundary continuity. In this case, two assistant points must be added, and time instants become

$$\mathbf{T} = [\overline{T}_0, \overline{T}_1, T_1, \cdots, T_{n-2}, \overline{T}_{n-2}, \overline{T}_{n-1}],$$

where $\overline{T}_0 = \beta T_0, \overline{T}_1 = (1-\beta)T_0, \overline{T}_{n-1} = \beta T_{n-1}, \overline{T}_{n-2} = (1-\beta)T_{n-1}$, $\beta \in (0,1)$ is a predefined partitioning factor. The band matrix $\mathbf{A}$ and coefficient matrix $\mathbf{C}$ are given by:

$$\mathbf{A} = \frac{1}{6}\begin{bmatrix} \frac{2-\beta}{1-\beta}T_0 & (1-\beta)T_0 & & & & \\ \frac{1-2\beta}{1-\beta}T_0 & 2((1-\beta)T_0+T_1) & T_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & (1-\beta)T_{n-1} & \frac{2-\beta}{1-\beta}T_{n-1} \end{bmatrix} \qquad (2)$$

$$\mathbf{C} = \begin{bmatrix} -\frac{1}{1-\beta}T_0^{-1} & \frac{1}{1-\beta}T_0^{-1} & & & & \\ \frac{1}{1-\beta}T_0^{-1} & -\frac{1}{1-\beta}T_0^{-1} - T_1^{-1} & T_1^{-1} & & & \\ & \ddots & \ddots & \ddots & & \\ & & T_{n-2}^{-1} & -T_{n-2}^{-1} - \frac{1}{1-\beta}T_{n-1}^{-1} & \frac{1}{1-\beta}T_{n-1}^{-1} & \\ & & & \frac{1}{1-\beta}T_{n-1}^{-1} & -\frac{1}{1-\beta}T_{n-1}^{-1} \end{bmatrix}. \qquad (3)$$

Then the second-derivative coefficient vector $\mathbf{m}$, obtained by solving the banded linear system, $\mathbf{m} = \mathbf{A}^{-1}(\mathbf{CS} - \mathbf{D})$, uniquely determines the analytic expressions of all cubic spline segments through backward substitution of boundary conditions under boundary condition $\mathbf{D}$. The standard cubic splines excel at interpolating issued waypoints $\mathbf{s}_i = \mathbf{q}_i$, where $\mathbf{s}_i$ represents the actual interpolated point, while $\mathbf{q}_i$ expresses the waypoint issued.

Practical teleoperation requires an intentional trade-off between path accuracy and motion quality. We introduce the min stretch cubic spline with assigned initial and end status, formulated to minimize:

$$\mathbf{S}^* = \underset{\mathbf{s}_i}{\operatorname{argmin}} \left(\mu \sum_{i=0}^{n} w_i (\mathbf{s}_i - \mathbf{q}_i)^2 + (1-\mu) \sum_{i=0}^{n-1} \int_0^{T_i} \|\ddot{\mathbf{p}}_i(t)\|^2 dt \right), \qquad (4)$$

where $\mu \in [0,1]$ controls the importance between positional accuracy and smoothness, w_i modifies the weight of the i-th fitting error. $\ddot{\mathbf{p}}_i(t)$ represents the acceleration vector of the trajectory segment at time t, ensuring the objective function penalizes abrupt changes in motion. Here, $\mathbf{S}^*$ is the optimal set of points that minimizes the joint objective of accuracy and smoothness. The second term of (4) can be simplified by substituting (1) to $\operatorname{tr}\left(\mathbf{m}^T \overline{\mathbf{A}} \mathbf{m}\right)$, where $\overline{\mathbf{A}}$ is formulated as:

$$\overline{\mathbf{A}} = \frac{1}{6}\begin{bmatrix} 2T_0 & T_0 & & & & \\ T_0 & 2(T_0+T_1) & T_1 & & & \\ & T_1 & 2(T_1+T_2) & T_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & T_{n-1} & 2T_{n-1} \end{bmatrix}, \qquad (5)$$

where $\operatorname{tr}(\cdot)$ denotes the trace of a matrix. Then (4) can be reformulated into a compact form as

$$\begin{aligned} \mathbf{S}^* = \underset{\mathbf{s}_i}{\operatorname{argmin}} & \left(\mu \operatorname{tr}((\mathbf{Q}-\mathbf{S})^T \mathbf{W} (\mathbf{Q}-\mathbf{S})) + (1-\mu) \operatorname{tr}\left(\mathbf{m}^T \overline{\mathbf{A}} \mathbf{m}\right)\right) \\ & s.t. \mathbf{m} = \mathbf{A}^{-1}(\mathbf{CS} - \mathbf{D}), \end{aligned} \qquad (6)$$

where $\mathbf{W}$ is the compact matrix form of w_i and $\mathbf{S}$ represents the optimized trajectory waypoints that balance these objectives. It should be noted that the current formulation focuses on achieving deterministic computation cycles, prioritizing real-time guarantees over Kinematic inequality constraints typically requiring iterative resolution [21].

To ensure feasibility while maintaining computational efficiency, kinematic constraints (e.g., velocity, acceleration, and jerk limits) are enforced prior to the optimization by dynamically adjusting the time steps T_i between waypoints. This approach avoids introducing inequality constraints into the QP, thus enabling fast, deterministic solution times.

The problem (6) can be solved efficiently with its convex property and modern QP-solver such as piqp [22]. Significantly, when requiring motion between static configurations

(with zero initial/final velocities and accelerations), the spline generation permits a simplified closed-form solution through boundary condition analysis, as we shall derive in subsequent sections.

1) Analytical solution: Setting $\mathbf{D} \equiv \mathbf{0}$ (zero initial/final velocities and accelerations) and substituting the constraint $\mathbf{m} = \mathbf{A}^{-1}\mathbf{C}\mathbf{S}$ into the objective function, we solve the resulting equality-constrained quadratic program by enforcing first-order optimality conditions:

$$\mathbf{S}^* = \mathbf{Q} - \lambda \mathbf{W}^{-1}\mathbf{C}^T(\mathbf{G}^{-1} + \lambda \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^T)^{-1}\mathbf{C}\mathbf{Q}, \quad (7)$$

with $\lambda = \frac{1-\mu}{\mu}$ and $\mathbf{G} = \mathbf{A}^{-1}\overline{\mathbf{A}}\mathbf{A}^{-1}$. The direct solution for $\mathbf{S}^*$ involves computationally expensive matrix inversions, prompting us to reformulate the optimization in terms of $\mathbf{m}$ through algebraic manipulation. Suppose $(\mathbf{G}^{-1} + \lambda \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^T)\mathbf{g} = \mathbf{C}\mathbf{Q}$, we substitute $\mathbf{S}^*$ from (7) into the constraint $\mathbf{m} = \mathbf{A}^{-1}\mathbf{C}\mathbf{S}$ and derive the simplified expression:

$$\mathbf{m} = \mathbf{A}^{-1}(\mathbf{C}\mathbf{Q} - \lambda \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^T\mathbf{g}) = \overline{\mathbf{A}}^{-1}\mathbf{A}\mathbf{g}. \quad (8)$$

For efficient computation, we suppose $\overline{\mathbf{A}} = \mathbf{A}$, $\mathbf{W} = \mathbf{I}$, and the system reduces to:

$$\mathbf{m} = \mathbf{g} = (\mathbf{A} + \lambda \mathbf{C}\mathbf{C}^T)^{-1}\mathbf{C}\mathbf{Q}, \quad (9)$$

where $\mathbf{A} + \lambda \mathbf{C}\mathbf{C}^T$ is a band matrix. This sparsity structure enables efficient numerical methods for matrix inversion, improving computational speed and scalability of the proposed approach.

Additionally, the proposed methodology intentionally excludes collision constraints, instead leveraging situational awareness of human operators for real-time obstacle avoidance through the teleoperation interface.

B. Framework for Precise Manipulation Task

This subsection presents the framework design for high-precision manipulation tasks, such as fine assembly, surgical operations, and precision grasping. In these applications, it is essential for the generated trajectory to pass through all issued waypoints accurately. Accordingly, the optimization weight μ in (6) is set close to unity ($\mu = 0.999$) to prioritize positional accuracy over trajectory smoothness.

As shown in Algorithm 1, when the system receives inputs from the Preprocess module, the StartServo flag initiates trajectory generation and execution(Line 1). The process has two phases based on execution state. In the initial phase(Line 5-7), the first target point is handled specially due to potentially large deviation from the current robot position. The system uses a point-to-point method (SolvePTP, employing the simplified closed-form solution from Section III-A), which takes the robot's kinematic configuration $\mathscr{C}_{\text{robot}}$(including the velocity and acceleration limits) as input, to smoothly and physically-consistently guide joints to the initial target (Line 6). We define $\sigma(\Xi, \tau)$ as a cubic spline trajectory parametrized by time τ, where Ξ represents the spline coefficients. This trajectory is then executed in the next loop, with $q_{\text{SendServo}}$ calculated at Line 9.

In subsequent iterations (Line 9-14), joint angles are extracted and time steps calculated using buffer timestamps.

Algorithm 1: Precise Mode Trajectory Planning

Input: $buffer$, $\mathscr{C}_{\text{robot}}$, Δt_{output}
Output: $q_{\text{SendServo}}$
1 **Initialization:** $StartServo \leftarrow$ True, $FirstTime \leftarrow$ True
2 **while** $StartServo =$ *True or* $buffer$ *is not empty* **do**
3 $q_{\text{current}} \leftarrow$ GetCurrentPositions;
4 **if** $FirstTime$ **then**
5 $q_{\text{first}} \leftarrow$ pop buffer;
6 $\sigma(\Xi, \tau) \leftarrow$ SolvePTP$(q_{\text{current}}, q_{\text{next}}, \mathscr{C}_{\text{robot}})$;
7 $FirstTime \leftarrow$ False;
8 **else**
9 $q_{\text{SendServo}} \leftarrow$ ExecuteTrajectory$(\sigma(\Xi_{\text{current}}, \tau))$;
10 $q_i, T_i \leftarrow$ extract all points from buffer;
11 $q_{\text{predict}} \leftarrow$MotionPredictor;
12 $\dot{q}_{\text{current}}, \ddot{q}_{\text{current}} \leftarrow$ calculate by $\sigma(\Xi_{\text{current}}, \tau)$;
13 $\sigma(\Xi_{\text{next}}, \tau) \leftarrow$ MinStretchSpline$(q_i, \dot{q}_{\text{current}}, \ddot{q}_{\text{current}}, T_i, \mathscr{C}_{\text{robot}})$;
14 $\sigma(\Xi_{\text{current}}, \tau) \leftarrow \sigma(\Xi_{\text{next}}, \tau)$;
15 **end**
16 **end**
17 $StartServo \leftarrow$ **Monitor Thread**;
Execute Trajectory Divide the duration into Δt_{output} intervals and compute the corresponding joint values for each.
Monitor Thread: Set $StartServo \leftarrow$ False if teleoperation is stopped;
Sender Thread: Send $q_{\text{SendServo}}$ to motor controller every Δt_{output};

To compensate for teleoperation latency, a motion prediction module(**MotionPredictor**) is optionally incorporated to estimate future states q_{predict} based on the historical trajectory points and their timestamps. The predicted state q_{predict} is then incorporated into the waypoint set q_i.

The **MinStretchSpline** (implementing in (6)) generates trajectories by integrating current states with buffered target sequences(up to 50), while dynamically adapting time steps T_i to satisfy kinematic limits(velocity, acceleration, and jerk) extracted from URDF models. If the planned motion exceeds any limit, T_i is automatically extended to ensure feasibility. Only the first trajectory segment is executed to support incremental planning. Typically, **ExecuteTrajectory** and **MinStretchSpline** run asynchronously—executing current and calculating next segments simultaneously—improving efficiency.

Throughout all the process, the **Monitor Thread** dynamically updates the StartServo flag to be paused or resumed servoing as needed. Concurrently, the **Sender Thread** transmits the generated joint commands $q_{\text{SendServo}}$ to the motor controllers at a stable frequency Δt_{output}, ensuring consistent, high-fidelity performance.

C. Framework for Rapid Motion Task

Time delay significantly impacts the stability and transparency of teleoperation systems [23], [24]. In the precise mode, delays accumulate as the robot must sequentially execute all commanded points from its start position. New commands arriving during this motion further exacerbate the latency. In this mode, the optimization weight μ is set to 0.5.

Algorithm 2: Rapid Mode Trajectory Planning

Input: $buffer$, $\mathscr{C}_{\text{robot}}$, Δt_{servo}, Δt_{output}
Output: $q_{\text{SendServo}}$
1 **Initialization:** $StartServo \leftarrow$ True, $\sigma(\Xi, \tau) \leftarrow$ empty
2 **while** $StartServo = True$ *or buffer is not empty* **do**
3 $q_{\text{current}} \leftarrow$ GetCurrentPositions;
4 **if** $\sigma(\Xi, \tau)$ *is empty* **then**
5 $q_{\text{first}} \leftarrow$ pop buffer;
6 $\sigma(\Xi, \tau) \leftarrow \text{SolvePTP}(q_{\text{current}}, q_{\text{first}}, \mathscr{C}_{\text{robot}})$;
7 **else**
8 $q_{\text{SendServo}} \leftarrow$ ExecuteTrajectory$(\sigma(\Xi_{\text{current}}, \tau), \Delta t_{\text{servo}})$;
9 $q_{\text{new}} \leftarrow$ pop buffer;
10 $q_{\text{predict}} \leftarrow$MotionPredictor;
11 $q_{\text{current}}, \dot{q}_{\text{current}}, \ddot{q}_{\text{current}}, q_{\text{end}} \leftarrow \sigma(\Xi_{\text{current}}, \tau)$;
12 $q_i \leftarrow \text{PlanJointPath}(q_{\text{current}}, q_{\text{end}}, q_{\text{new}})$;
13 $\sigma(\Xi_{\text{next}}, \tau) \leftarrow$ MinStretchSpline$(q_i, \dot{q}_{\text{current}}, \ddot{q}_{\text{current}}, T_i, \mathscr{C}_{\text{robot}})$;
14 $\sigma(\Xi_{\text{current}}, \tau) \leftarrow \sigma(\Xi_{\text{next}}, \tau)$;
15 **end**
16 **end**
17 $StartServo \leftarrow$ **Monitor Thread**;

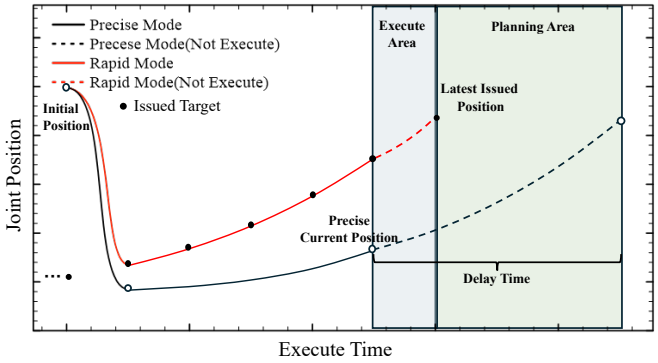


Fig. 3: Illustrations of rapid mode and precise mode.

To mitigate latency, as shown in Fig. 3, the rapid mode prioritizes the latest waypoint over any previously commanded but unreached points. Here, Δt_{servo} denotes the replanning interval: in each cycle, only the first Δt_{servo} segment of the current trajectory is executed before replanning with the latest waypoint. As illustrated in Algorithm 2, the initial trajectory to the first waypoint is fully planned(Line 5-6), but only a segment corresponding to time Δt_{servo} is executed(Line 8). The trajectory is then continuously updated to incorporate the latest waypoint(Line 8-14).

However, generating trajectories directly from the current position to the latest waypoint can induce jerky motion due to abrupt acceleration changes. To ensure smoothness, each new trajectory is constructed using three key points: current desired position(q_{current}), the end point of the previous trajectory(q_{end}), and the newly received waypoint(q_{new}) (Line 9-12). This strategy promotes gradual acceleration changes, preventing abrupt movements while maintaining responsiveness and operational stability.

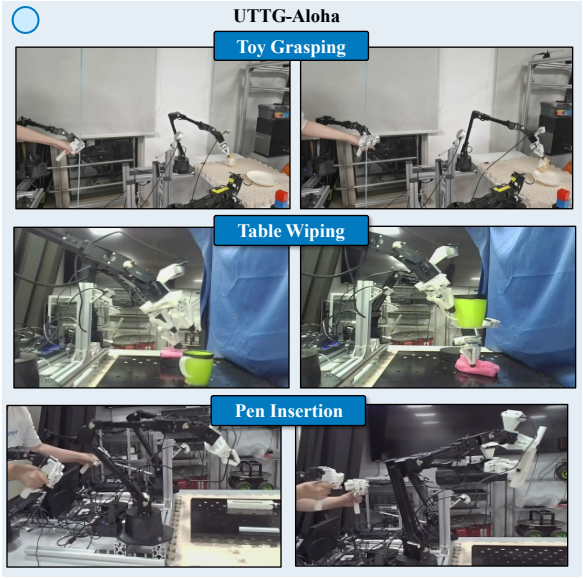


Fig. 4: Snapshots in aloha platform.

IV. EXPERIMENTS

This section presents experimental evaluations of the UTTG framework across three distinct robotic platforms. The experiments demonstrate the general applicability of the framework and evaluate its performance regarding tracking accuracy, motion smoothness, and time delay. Furthermore, the effectiveness of the incorporated human motion prediction module is validated under varying delay conditions to assess its capability in compensating for teleoperation latency.

A. Experiment Setup

We conducted three experiments to evaluate different aspects of UTTG framework:

Trajectory Tracking: Realman[1] executed predefined motions for trajectory accuracy evaluation. (Input: 30Hz from a motion capture suit in end-effector mapping mode, Output: 200Hz after interpolation)

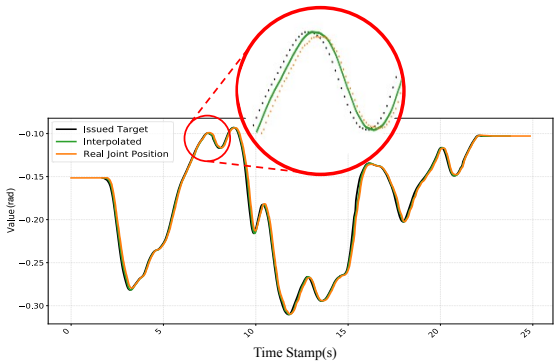
Precision Tasks: Aloha performed three manipulation tasks(shown in Fig. 4):

- *Object Manipulation:* Grasping and placement of a soft toy and a cylindrical can.
- *Pen Insertion:* Put the pen into the box.
- *Table Wiping:* Dual-arm coordination: (i) cup displacement, (ii) table wiping, (iii) environment reset.

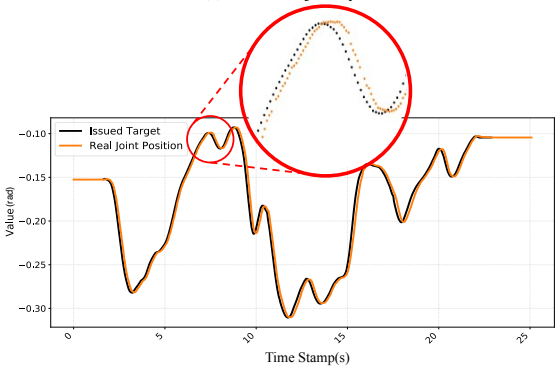
(Input: 20Hz from master arms in master-slave mapping mode, Output: 200Hz to slave arm after interpolation)

Predictor Performance Evaluation: The Franka robot was employed to evaluate the effectiveness of our prediction module in compensating for teleoperation latency. (Input: 30Hz from a Vision Pro[2] in end-effector mapping mode, Output: 1000Hz after interpolation)

[1]Realman: `www.realman-robotics.cn`
[2]Vision-Pro:`www.apple.com/apple-vision-pro/`



(a) UTTG trajectory



(b) Realman algorithm trajectory

Fig. 5: 5a and 5b present the comparison of generated trajectories between UTTG and Realman algorithm. Our framework interpolates joint positions before sending commands to the motor, and the Realman algorithm optimizes and sends positions in the internal control loop, which is unreachable thus cannot be migrated to other robot platform.

Our framework exhibits dimensional independence via parallel joint-space processing, enabling complex multi-DoF applications. All experiments are conducted on an Intel NUC computer (i7-1165G7, 16GB RAM).

B. Experiment Result

1) Trajectory Tracking on Realman Robot: This experiment evaluates tracking accuracy and trajectory smoothness.

As shown in Fig. 5, we compared the performance of our framework(5a) with the native control algorithm of Realman robot(5b). Our framework interpolates the issued targets to high-frequency joint trajectories, whereas the Realman algorithm processes raw low-frequency inputs. Analysis of the actuator feedback data (Real Robot Position) in magnified regions demonstrates that our framework generates trajectories with smoother transitions and reduced abrupt changes, particularly during high-velocity segments.

To comprehensively evaluate the performance, we compared multiple metrics including motion smoothness, system latency and tracking accuracy with linear interpolation algorithm and the Realman Method.

In the linear interpolation method [17], the trajectory is generated by connecting the current position and target positions with a straight line as:

$$q(t) = q_{\text{start}} + \frac{q_{\text{target}} - q_{\text{start}}}{T} \cdot t \tag{10}$$

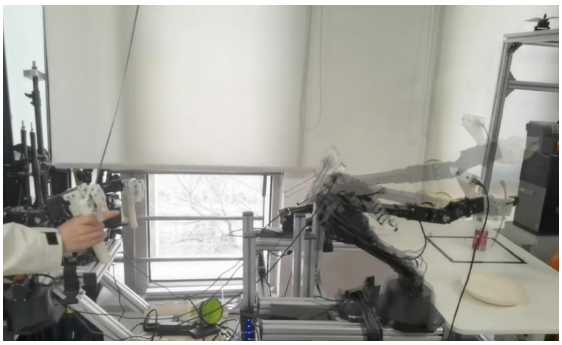


Fig. 6: Visualization of pronounced jitter in the baseline method, caused by the 20Hz joint replication process.

This method only ensures the position is continuous, but it does not guarantee smoothness in velocity or acceleration.

TABLE I: Performance comparison of different trajectory generation methods

Method	Evaluation Metrics		
	MAV(rad/s^2) ↓	Latency(ms) ↓	Traj Error(rad) ↓
Linear Interpolation [17]	11.3	-	-
Realman Method[1]	1.63	119	0.0106
UTTG (Ours)	**0.608**	**39**	**0.005**

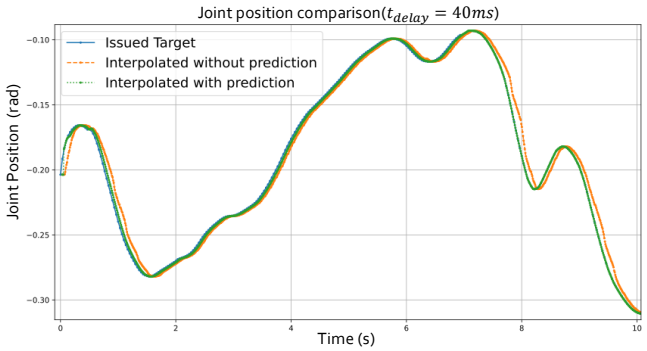
As shown in Table I, our method demonstrates advantages across all evaluation criteria: the MAV(mean absolute value) of joint accelerations shows substantial reductions (95% compared to linear interpolation and 63% compared to Realman Method), indicating better motion smoothness. Additionally, our approach achieves lower latency and reduced trajectory error compared to the existing Realman Method. The online computation time for trajectory generation was consistently measured below 50 ms, ensuring real-time performance without introducing substantial additional latency.

TABLE II: Task success rate with and without UTTG interpolation.

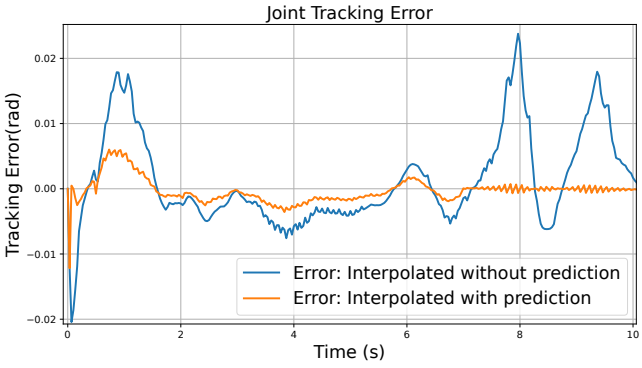
Method	Task Success Rate			
	Toy	Can	Pen	Table
UTTG	1.00	0.92	0.88	0.90
No UTTG	0.82	0.76	0.58	0.54

2) Precision Tasks on Aloha: We conducted 50 trials for each task under teleoperation with and without UTTG. The baseline method operated at 20Hz control frequency, while UTTG achieved 200Hz through online trajectory interpolation. To mitigate user-specific bias, two novice operators (each trained for 3 minutes) performed the tasks, with identical time constraints: each trial was limited to 10 seconds (Table wiping 15 seconds), and timeout was classified as failure.

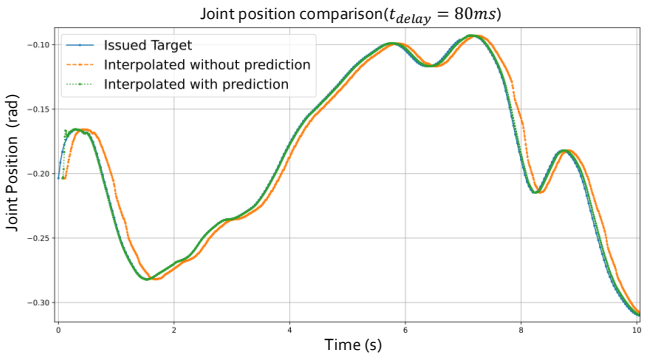
Table II illustrates the success rate across distinct tasks. For single-arm tasks (e.g., Toy/Can), both methods achieve comparable reliability due to inherent task simplicity. In complex bimanual tasks (Table wiping/Pen insertion), the baseline method's pronounced jitter(see Fig. 6) was observed to increase operational difficulty significantly, leading to



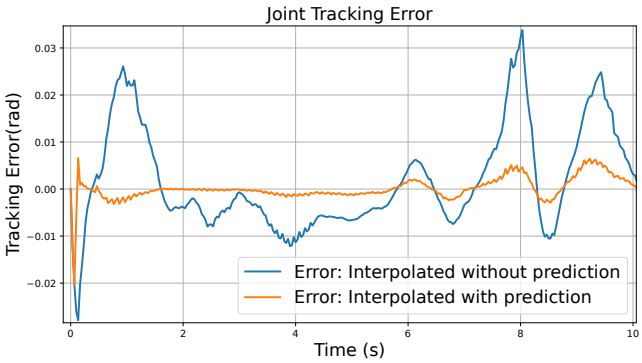
(a) Tracking performance with 40ms delay



(b) Tracking performance with 40ms delay



(c) Tracking performance with 80ms delay



(d) Tracking performance with 80ms delay

Fig. 7: Performance comparison with/without prediction under different time delays.

frequent failures in delicate operations such as simultaneous cloth and cup manipulation during table wiping. In contrast, UTTG maintains robust performance through smooth trajectory generation, effectively reducing operator burden and enabling reliable execution of complex tasks.

3) Predictor Test on Franka: In this experiment, to evaluate the effectiveness of the motion prediction module, we introduced artificial delays (40ms and 80ms) to simulate real-world teleoperation latency. Comparative experiments were conducted with and without the prediction module enabled.

The results demonstrate that our prediction module improves trajectory tracking accuracy under both latency conditions. As delay increases, disabling prediction introduces noticeable lag in the slave device response, which was observed to increase operational difficulty. In contrast, the predictor compensates for delay by anticipating operator intent and generating future states, helping maintain synchronization between human input and robot motion. Fig. 7 provides qualitative trajectory comparisons under both delay settings.

V. CONCLUSION

This paper presents UTTG, a dimension-independent teleoperation framework that enables rapid cross-platform deployment and online continuous trajectory generation. The system utilizes a minimum-stretch spline optimization to achieve smooth motion quality and incorporates a motion prediction module to mitigate latency caused by hardware and communication delays. The purpose of this approach is to facilitate the potential release of low-cost devices for expert data collection, providing an effective tool for the construction of broader embodied intelligence datasets. Experiments across three robotic platforms validate smoother trajectories (95% MAV reduction), better operation performance (36% success rate increases in complex tasks) of our method, and low-latency responsiveness (with delay consistently below 50 ms).

The current framework focuses on trajectory optimization and latency compensation without explicit collision avoidance, instead relying on the operator's situational awareness for obstacle avoidance. We plan to develop reactive obstacle avoidance networks that integrate with the online trajectory generation module to enable real-time collision avoidance during teleoperation even when only end-effector inputs are available.

REFERENCES

[1] K. Darvish, L. Penco, J. Ramos, R. Cisneros, J. Pratt, E. Yoshida, S. Ivaldi, and D. Pucci, "Teleoperation of Humanoid Robots: A Survey," Jan. 2023.

[2] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent Advances in Robot Learning from Demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, May 2020.

[3] B. Siciliano and O. Khatib, eds., *Springer Handbook of Robotics*. Springer Handbooks, Springer International Publishing, 2016.

[4] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning," in *Proceedings of the 5th Conference on Robot Learning*, PMLR, Jan. 2022.

[5] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel, "GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators," July 2024.
[6] D. Honerkamp, H. Mahesheka, J. O. von Hartz, T. Welschehold, and A. Valada, "Zero-cost whole-body teleoperation for mobile manipulation," *arXiv preprint arXiv:2409.15095*, 2024.
[7] B. Xia, X. Tian, B. Yuan, Z. Li, B. Liang, and X. Wang, "Trajectory planning for teleoperated space manipulators using deep reinforcement learning," *arXiv preprint arXiv:2408.05460*, 2024.
[8] J. Elsner, G. Reinerth, L. Figueredo, A. Naceri, U. Walter, and S. Haddadin, "Parti-a haptic virtual reality control station for model-mediated robotic applications," *Frontiers in Virtual Reality*, vol. 3, p. 925794, 2022.
[9] Z. Fu, T. Z. Zhao, and C. Finn, "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation," Jan. 2024.
[10] Y. Ishiguro, T. Makabe, Y. Nagamatsu, Y. Kojio, K. Kojima, F. Sugai, Y. Kakiuchi, K. Okada, and M. Inaba, "Bilateral humanoid teleoperation system using whole-body exoskeleton cockpit tablis," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6419–6426, 2020.
[11] M. Arduengo, A. Arduengo, A. Colomé, J. Lobo-Prat, and C. Torras, "Human to robot whole-body motion transfer," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pp. 299–305, IEEE, 2021.
[12] F. Krebs, A. Meixner, I. Patzer, and T. Asfour, "The kit bimanual manipulation dataset," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pp. 499–506, IEEE, 2021.
[13] Q. Rouxel, K. Yuan, R. Wen, and Z. Li, "Multicontact motion retargeting using whole-body optimization of full kinematics and sequential force equilibrium," *IEEE/ASME Transactions on Mechatronics*, 2022.
[14] R. Ding, Y. Qin, J. Zhu, C. Jia, S. Yang, R. Yang, X. Qi, and X. Wang, "Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning," *arXiv preprint arXiv:2407.03162*, 2024.
[15] S. LaValle, "Planning algorithms: Cambridge university press, 2006," 2006.
[16] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, 2014.
[17] S. Tanaka, Y. M. Baek, N. Sugita, T. Ueta, Y. Tamaki, and M. Mitsuishi, "Minimum-jerk trajectory generation for master-slave robotic system," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, IEEE, 2012.
[18] L. Berscheid and T. Kröger, "Jerk-limited real-time trajectory generation with arbitrary target states," *arXiv preprint arXiv:2105.04830*, 2021.
[19] C. M. Luigi Biagiotti, *Trajectory Planning for Automatic Machines and Robots*. Springer, 2009.
[20] Y. Zhou, G. Sun, Y. Miao, Y. Zhang, X. Chen, and H. Wang, "Spatiotemporal Optimal Trajectory Planning for Safe Planar Manipulation of a Moving Object," *IEEE Transactions on Industrial Electronics*, July 2024.
[21] O. Robotics, "Moveit 2 documentation." `https://moveit.ai/docs/`, 2023. Accessed: 2023-08-20.
[22] R. Schwan, Y. Jiang, D. Kuhn, and C. N. Jones, "Piqp: A proximal interior-point quadratic programming solver," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 1088–1093, IEEE, 2023.
[23] T. A. Várkonyi, I. J. Rudas, P. Pausits, and T. Haidegger, "Survey on the control of time delay teleoperation systems," in *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, 2014.
[24] M. W. Spong, "An Historical Perspective on the Control of Robotic Manipulators," *Annual Review of Control, Robotics, and Autonomous Systems*, May 2022.