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

Accelerating Trajectory Optimization by Exploiting B-Spline Gradient Structure

Nikos Doiron, Thomas Duquette, Gilde Vanel Tchane Djogdom, and André Gallant

Abstract— **This work presents a discrete-time trajectory optimization framework that achieves near real-time performance for robotic manipulators. This is achieved by drastically speeding up constraint gradient computations. The approach leverages the analytical and structural properties of B-splines to introduce three key speedups: exploiting gradient sparsity from local control, using a hybrid-analytical method to replace most finite differences with closed-form derivatives, and aggregating constraints per knot-span to reduce the problem size. Validated on a simulated UR5e across 64 tasks in a cluttered workspace, these cumulative speedups reduce computation time by up to 96.3% (a 26.9x speedup) relative to a finite-difference baseline, without compromising trajectory quality, success rate, or fidelity to kinematic, dynamic, and collision constraints.**

Index Terms— **Motion and Path planning; Optimization and Optimal Control; Collision Avoidance**

I. Introduction

Trajectory optimization enables serial manipulators to execute precise, smooth, and efficient motions while enforcing kinematic, dynamic, and safety constraints. Although continuous-time formulations offer elegant theoretical representations, their computational burden limits their applicability to real-time control and online planning. A discretized approach is typically used to ensure satisfaction of constraints in a more reasonable computation time. Despite this advantage, solving discrete-time trajectory optimization problems remains computationally demanding, particularly in scenarios requiring fast re-planning or adaptation to dynamic environments.

Smoothness is an essential component of trajectory planners, since it is necessary for trajectories to be tractable by actuators. To ensure smoothness, parametric curves such as splines have become widely used. Particularly, B-splines have become commonplace in robotic applications, as they offer many useful properties such as local support and the convex hull property [1], [2]. They facilitate the enforcement of smoothness and constraints at discrete points along the trajectory, while maintaining a compact representation. Existing approaches primarily focus on continuous-time formulations or on exploiting B-spline properties for the satisfaction of local constraints. The efficient computation of their gradients is paramount to the computational efficiency of the trajectory optimization algorithm, but it has historically been an understudied subject.

One of the highest-performing gradient-based optimization solvers is Sequential Quadratic Programming (SQP). It has emerged as a standard tool to handle constrained nonlinear optimization [3], [4]. However, its practical efficiency is heavily influenced by the cost of gradient computations, which often dominate the overall computation time. Prior efforts to reduce this burden have taken advantage of problem-specific structures, including sparse system dynamics [5] and reduced order models [6]. We take inspiration from such works to exploit the parametric form of the B-spline trajectory representation itself.

In this work, we propose a discrete-time trajectory optimization framework that takes advantage of the structure of B-splines to speed up gradient evaluation in gradient-based solvers such as SQP. The method optimizes directly over the position control points, enabling efficient computation of position, velocity, and acceleration gradients. This framework is further extended to compute torque, tool center point (TCP) speed, and collision gradients. The local control property is used to skip unnecessary gradient evaluations in spans where we know the gradient to be zero. This structured formulation substantially reduces computation time while maintaining trajectory feasibility under discrete-time constraints.

The proposed framework is validated on a simulated Universal Robot UR5e serial robotic manipulator, demonstrating substantial reductions in computation time and enabling near real-time trajectory optimization. Results highlight the potential of exploiting the B-spline structure for efficient gradient-based optimization in practical robotic systems.

II. Related Work

Motion planning for serial manipulators has been extensively studied in the last decade, with approaches broadly classified into sampling-based and optimization-based approaches.

Sampling-based planners, such as Rapidly-exploring Random Trees (RRT) [7] and Probabilistic Roadmaps (PRM) [8], excel in exploring high-dimensional or cluttered spaces. Extensions include multi-robot coordination [9], [10] and dynamic environments [11]. Although these methods provide probabilistic completeness and, in some variants, asymptotic optimality [12], they often produce coarse trajectories that require post-processing for smoothness, limiting their applicability for time-optimal tasks.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under scholarship BESC D-589586-2024, by Mitacs and Kinova through the Mitacs Accelerate grant, by Mitacs and Hadeon Robotics through the Mitacs Accelerate Entrepreneur grant, and by the Fondation Baxter & Alma Ricard scholarship.

The authors are with the Dynamium Laboratory, Université de Moncton, Moncton, NB E1A 3E9 Canada (emails: nikos.doiron / thomas.duquette / gilde.vanel.tchane.djogdom / andre.gallant @umoncton.ca). (Corresponding author: Nikos Doiron.)

Data and source are available on-line at https://github.com/Dynamium-Lab/blast.

On the other hand, optimization-based approaches directly compute trajectories by minimizing a cost function subject to kinematic, dynamic, and collision constraints. Classical methods include CHOMP [13], STOMP [14], TrajOpt [15], and GPMP2 [16]. TrajOpt partially bridges discrete and continuous formulations, using linear approximations for collision checking, yet accuracy suffers in fast or rotational motions. CHOMP and STOMP provide high-quality trajectories and can handle local constraints effectively, but the computation load remains a significant challenge. GPMP2 is very fast and provides even faster re-planning for collision-free motions. However, constraints on manipulator dynamics are not considered; the goal is to find a smooth, feasible path, and the focus is not on time-optimality.

Within optimization-based planners, B-spline-based methods have been widely used to represent trajectories compactly. In the discrete setting, the B-splines allow the enforcement of position, velocity, and acceleration constraints at the discretization points, taking advantage of the local control property to reduce computation [17], [18]. Common objectives for trajectory optimization using B-splines include trajectory smoothness, time, jerk, or energy optimization [19], [20]. Other methods exploit warm-starting, hybrid optimization pipelines (B-spline + SQP or LCQP), or analytical derivatives to speed up convergence [21], [22]. However, these approaches primarily focus on accelerating solver convergence or simplifying constraint handling, rather than reducing the intrinsic computational complexity of gradient evaluation within each SQP iteration. In typical implementations, gradient computation remains proportional to the total number of control points and active constraints, since the recursive derivative structure of B-splines with respect to control points is not explicitly exploited. As a result, computational cost scales unfavorably when increasing spline order, knot density, or when introducing additional constraints such as torque limits, TCP speed bounds, or collision avoidance.

Gradient speedup in SQP has been investigated through the exploitation of problem sparsity or analytical derivative structures [4], [5]. Sommer et al. [21] further demonstrated efficient derivative computation for cumulative B-splines on Lie groups, achieving significant speedups in state-estimation problems. However, these approaches do not address the discrete-time B-spline control-point formulation commonly used in manipulator trajectory optimization, nor do they incorporate constraints on angular position, velocity, and acceleration, torque, TCP speed, or collisions.

Efforts to combine spline parameterizations with gradient-based nonlinear optimization have shown promise, particularly for enforcing nonlinear constraints in trajectory planning. For example, Tang et al. [18] proposed a B-spline–based formulation to enforce collision constraints using Bézier conversion for tighter convex-hull bounds, and several recent works employ B-spline representations with analytical derivatives in optimization pipelines for UAV trajectory planning [21]. These studies highlight the advantages of spline structures for constraint handling and solver performance. Nonetheless, their approaches typically rely on conservative convex-hull approximations or linearizations to guarantee feasibility, which can reduce trajectory optimality. They remain limited in simultaneously achieving efficient gradient computation while maintaining high fidelity to complex constraints.

In robotic manipulator applications, B-spline + SQP methods remain scarce. Most approaches use splines primarily to enforce smoothness or convex hull constraints, without leveraging the analytical structure of B-spline derivatives to speed up gradient computation. As a result, while they can handle position, velocity and acceleration, or collision constraints at a discrete set of samples, they remain computationally heavy when extended to higher-order splines, dense knot placements, or additional torque limits.

A. Contribution

Our work distinguishes itself from these methods by directly exploiting the recursive structure of B-spline derivatives for efficient gradient evaluation in discrete-time SQP. This enables the solver to simultaneously manage position, velocity and acceleration, torque, TCP speed, and collision constraints while significantly reducing the computational overhead that limits prior hybrid approaches.

Our contribution fills this gap by

- leveraging the local support property of B-splines to reduce the number of required constraint gradient evaluations,
- exploiting the analytical derivatives of B-splines with respect to control points to speed up gradient computation within an SQP solver,
- reducing the number of constraints passed to the SQP solver while maintaining the same fidelity.

This positions our work at the intersection of discrete B-spline trajectory optimization and structure-aware SQP solving, enabling fast and constraint-compliant discrete-time planning.

III. Preliminaries

This work presents a near-optimal trajectory optimization algorithm for serial manipulators. The goal is to generate a collision-free trajectory that satisfies the constraints while minimizing motion time. The algorithm's inputs include the manipulator's kinematics, dynamics, collision model, and joint limits. A task is defined by an initial joint state $\mathbf{X}_0$ and a final joint state $\mathbf{X}_T$, both of which specify the joint positions $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, and accelerations $\ddot{\mathbf{q}}$. T denotes the total motion time.

A. B-splines

A central criterion of the optimization procedure is to generate smooth trajectories that satisfy the prescribed boundary conditions from a given optimization vector. In this work, we use *quintic ($p = 5$) clamped uniform B-splines*, which guarantees smoothness and local control, making them well suited for manipulator trajectories.

Formally, the B-splines are defined as

$$\mathbf{q}(u) = \sum_{k=1}^{K} \mathbf{c}_k B_k(u) \tag{1}$$

where K is the total number of control points and $\mathbf{c_k}$ are the control points defined in joint space. Each control point is associated to a basis function, $B_k(u)$. The independent variable u represents the normalized time, $u = t/T$, where t is an instant in time and T is the total duration of the trajectory.

One advantage of B-splines is that, by discretizing the trajectory in the u-domain, the basis functions become time-independent and thus can be precomputed before starting the optimization. Furthermore, the basis functions can be differentiated with respect to u, making the derivative basis functions of the B-spline likewise time-independent. Thus, the joint kinematic parameters of the trajectory can be computed at predefined u_i values as

$$\mathbf{q}(u_i) = \sum_{k=1}^{K} \mathbf{c}_k B_k(u_i) \tag{2}$$

$$\dot{\mathbf{q}}(u_i) = \tfrac{1}{T} \sum_{k=1}^{K} \mathbf{c}_k \dot{B}_k(u_i), \tag{3}$$

$$\ddot{\mathbf{q}}(u_i) = \tfrac{1}{T^2} \sum_{k=1}^{K} \mathbf{c}_k \ddot{B}_k(u_i), \tag{4}$$

where $B(u_i)$, $\dot{B}(u_i)$, and $\ddot{B}(u_i)$ denote the basis functions for position, velocity, and acceleration, respectively, precomputed at predefined u_i discretization values.

Note: the traditional method of computing the velocity and acceleration is to express them as a B-spline of lower degree with new control points as linear combinations of the position control points. In this work, we avoid this approach to reduce the overhead of computing new control points and to simplify gradient computation, as detailed in subsequent sections.

Another important property of B-splines that will be heavily exploited in this work is their *local control*. Indeed, the basis functions are only non-zero on a portion of the u-domain defined by *knots*. We use uniform B-splines, thus, the knots are uniformly distributed in u. Fig. 1 shows the position basis functions along with the knot locations. It can be shown that exactly $p+1$ basis functions are non-zero between knots. Therefore, each control point only affects a portion of the total generated motion, as it is non-zero along $p+1$ knot-spans. Many other parametrization techniques such as Bézier curves, standard polynomials, and interpolating splines do not possess this property.

B. Problem Statement

Satisfying boundary conditions on $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ requires imposing the first three and last three control points, leaving $K-6$ mutable control points. The optimization vector is then formed by stacking these mutable control points across

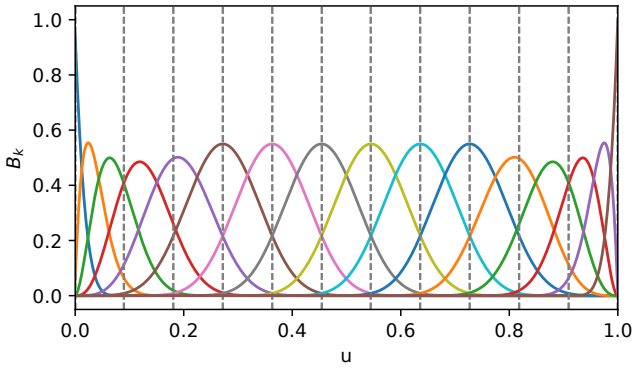

Fig. 1. Basis functions $B_k(u)$ of a quintic uniform clamped B-spline ($K = 16$). Vertical dashed lines indicate knots.

all joints, together with the time-scaling factor T as

$$\mathbf{z} = \begin{bmatrix} c_{4,1} \\ \vdots \\ c_{K-3,1} \\ \vdots \\ c_{4,N_j} \\ \vdots \\ c_{K-3,N_j} \\ T \end{bmatrix} \tag{5}$$

where $c_{k,j}$ is the value of control point k for joint j and N_j is the number of manipulator joints (degrees of freedom). The length of the vector $\mathbf{z}$, denoted N_z throughout the paper, is $(K-6) \times N_j + 1$.

A feasible trajectory must respect the capabilities of the manipulator. The method proposed in this work is capable of considering a *large variety of constraints*, *e.g.*, joint position, velocity, acceleration, and torque, as well as tool center point speed and collisions. Let $\mathbf{y} \in \mathbb{R}^{N_j}$ denote the stacked constraint values at a given u, with elementwise bounds $\mathbf{y}^{\min}, \mathbf{y}^{\max} \in \mathbb{R}^{N_j}$. In this work, the following bounds are imposed

$$\mathbf{q}(u_i) \in [\mathbf{q}^{\min}, \mathbf{q}^{\max}], \qquad \dot{\mathbf{q}}(u_i) \in [-\dot{\mathbf{q}}^{\max}, \dot{\mathbf{q}}^{\max}],$$
$$\ddot{\mathbf{q}}(u_i) \in [-\ddot{\mathbf{q}}^{\max}, \ddot{\mathbf{q}}^{\max}], \quad \boldsymbol{\tau}(u_i) \in [-\boldsymbol{\tau}^{\max}, \boldsymbol{\tau}^{\max}],$$
$$v_{\text{tcp}}(u_i) \in [0, v_{\text{tcp}}^{\max}].$$

Define the elementwise midpoint and range

$$\mathbf{m} = \tfrac{1}{2}(\mathbf{y}^{\max} + \mathbf{y}^{\min}), \qquad \mathbf{r} = \mathbf{y}^{\max} - \mathbf{y}^{\min}.$$

Then the normalized constraints are given elementwise

$$\mathbf{g} = \frac{2\,|\,\mathbf{y} - \mathbf{m}\,|}{\mathbf{r}} - \mathbf{1} \;\leq\; \mathbf{0}, \tag{6}$$

where all operations are applied elementwise.

The discrete time–optimal motion problem is

$$\begin{aligned} \min_{\mathbf{z}} \quad & T \\ \text{s.t.} \quad & \left.\begin{aligned} & \mathbf{g}_i \leq \mathbf{0} \\ & d_{\text{self}}(\mathbf{q}(u_i)) \geq 0 \\ & \mathbf{d}_{\text{coll}}(\mathbf{q}(u_i)) \geq \mathbf{0} \end{aligned}\right\} \quad i = 0, \ldots, N_{pt} \\ & \mathbf{X}(0) = \mathbf{X}_0 \\ & \mathbf{X}(T) = \mathbf{X}_T \end{aligned} \tag{7}$$

where d_{self} is the minimum signed distance to a self-collision and $\mathbf{d}_{\text{coll}}$ is the vector of minimum signed distances from each link to its nearest collision object. Constraints are enforced at N_{pt} uniformly spaced discretized points located at $u_i = i/N_{pt}$. For sufficiently large N_{pt}, this approximates the continuous-time problem. Since calculating the objective function (the total time T) is straightforward, the main computational challenge lies in enforcing the numerous and complex constraints.

The total number of nonlinear inequality constraints for problem (7) is given by

$$N_{con} = N_{pt}(4N_j + 1_{TCP} + 1_{self} + N_{link}) \tag{8}$$

where $4N_j$ represents the $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\tau}$ constraints per joint, 1_{TCP} and 1_{self} are used to indicate that there is one constraint imposed per discrete point for TCP speed and self-collisions, respectively, and N_{link} is the number of links in the manipulator and represents the external collisions.

Note: empirical evidence indicates that considering only the worst self-collision constraint is sufficient, whereas enforcing one external collision constraint per link enhances optimization convergence.

While kinematic constraints $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ can be obtained directly from the B-spline parameterization, other constraints require additional evaluation. Joint torques $\boldsymbol{\tau}$ are computed using the recursive Newton–Euler algorithm; v_{tcp} is obtained using the manipulator Jacobian; and collision distances d_{self} and $\mathbf{d}_{\text{coll}}$ are obtained using forward kinematics and distance computations between robot links, represented as capsules, and the environment, represented as primitives such as Oriented Bounding Boxes (OBBs), spheres, and capsules.

The method proposed in this work seeks near-optimal solutions to Eq. (7), with emphasis on efficient handling of the constraints. In this work, the optimization is carried out using a gradient-based Sequential Quadratic Programming (SQP) algorithm implemented in the `nlopt` C++ library [23]–[25].

IV. Proposed Speedups

To highlight the benefits of the proposed speedups, we begin by establishing a baseline. As a baseline, we consider the naive approach where, at each iteration, all constraints are computed at each discrete point and gradients are computed by finite differences with respect to every parameter in the optimization vector $\mathbf{z}$. This method is straightforward to implement but scales poorly: computing the gradient of each constraint by finite difference scales with the number of control points K, the number of discretization points N_{pt}, and the number of joints N_j. This renders this approach impractical.

To address this challenge, we introduce three methods to speed up constraint gradient computation and to reduce the problem size, namely:

- *SpdUp 1*: exploiting the local control of B-splines, *i.e.*, only $p+1$ basis functions are non-zero at a given u;
- *SpdUp 2*: computing the gradients analytically where possible and introducing a hybrid finite difference method that no longer scales with the number of control points K;
- *SpdUp 3*: applying an order-reduction technique that aggregates constraints on a per–knot-span basis, rather than enforcing them at every discretization point.

Each method builds on the previous, *e.g.*, *SpdUp 3* also applies *SpdUp 1* and *SpdUp 2*. Together, these speedups form a cumulative strategy that significantly reduces the computational burden of constraint gradients and optimization.

A. SpdUp 1: Exploiting Local Control of B-splines

The first speedup exploits the fact that optimization variables affect only a subset of constraints. Indeed, the $N_z \times N_{con}$ *constraint gradient matrix* denoted by $\nabla g(\mathbf{z})$ is sparse. Each set of $(K-6)$ optimization variables only affects a given joint's trajectory. Therefore, the gradient with respect to all other variables is zero for kinematic constraints. Furthermore, the local property of B-splines indicates that a given control point only affects $(p+1)$ knot-spans along a trajectory. The gradient of constraints outside the affected knot-spans will inherently be zero. All other constraints are joint-coupled; therefore, the gradients with respect to all joints must be computed. However, the local control property of B-splines still applies.

This induces a block-diagonal structure in the constraint gradients, schematically

$$\nabla g(\mathbf{z}) = \left[\begin{array}{cccc|c} \boxed{G_1} & 0 & \cdots & 0 & D_1 \\ 0 & \boxed{G_2} & \cdots & 0 & D_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \boxed{G_{N_j}} & D_{N_j} \\ \hline \multicolumn{4}{c|}{\partial g/\partial T} & D_T \end{array}\right] \tag{9}$$

where G_j correspond to decoupled joint constraints (position, velocity, acceleration) and D_j, D_T correspond to coupled constraints, *i.e.*, torque, TCP speed, collision. The bottom row corresponds to the derivatives w.r.t. T, which are inherently different and cannot be treated with the same process.

B. SpdUp 2: Analytic and Hybrid Constraint Gradients

The second speedup concerns the introduction of analytical and hybrid gradient computations. For position, velocity, and acceleration constraints, analytical expressions are available in closed form. For coupled constraints such as torque, TCP speed, and collision distances, we adopt a hybrid approach that combines finite differences on the constraints with respect

to position, velocity, and acceleration, which are analytically mapped to the control points.

In the B-spline representation, $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ are linear combinations of the control point vector $\mathbf{c}_k$. Their derivatives with respect to the control points are

$$\frac{\partial \mathbf{q}(u)}{\partial \mathbf{c}_k} = B_k(u), \tag{10}$$

$$\frac{\partial \dot{\mathbf{q}}(u)}{\partial \mathbf{c}_k} = \tfrac{1}{T}\, \dot{B}_k(u), \tag{11}$$

$$\frac{\partial \ddot{\mathbf{q}}(u)}{\partial \mathbf{c}_k} = \tfrac{1}{T^2}\, \ddot{B}_k(u). \tag{12}$$

It should be noted that the partial derivatives in Eq. (10, 11, 12) are already available because we can precompute all basis functions at predefined u_i values as explained in section III-A.

These matrices are joint-local, sparse, and will be reused across all constraints. For notational brevity, the normalized time parameter (u) is omitted in what follows. Differentiating (6) with respect to $\mathbf{y}$ yields

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \frac{2\,\mathrm{sign}\big(\mathbf{y}-\mathbf{m}\big)}{\mathbf{r}}, \tag{13}$$

where all operations including $\mathrm{sign}()$ are applied elementwise. Given that $\mathrm{sign}(0) = 0$, the gradient is nullified at $y = m$. Because this point represents a non-differentiable local minimum, any movement away from m results in an immediate degradation of the constraint state. The gradients with respect to control points can be computed using the chain rule

$$\frac{\partial \mathbf{g}^q}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^q}{\partial \mathbf{q}}\,\frac{\partial \mathbf{q}}{\partial \mathbf{c}_k} = \frac{2\,\mathrm{sign}\left(\mathbf{q} - \frac{\mathbf{q}^{\max}+\mathbf{q}^{\min}}{2}\right)}{\mathbf{q}^{\max}-\mathbf{q}^{\min}} B_k, \tag{14}$$

$$\frac{\partial \mathbf{g}^{\dot{q}}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\dot{q}}}{\partial \dot{\mathbf{q}}}\,\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{c}_k} = \frac{\mathrm{sign}(\dot{\mathbf{q}})}{\dot{\mathbf{q}}^{\max}}\,\frac{1}{T}\,\dot{B}_k, \tag{15}$$

$$\frac{\partial \mathbf{g}^{\ddot{q}}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\ddot{q}}}{\partial \ddot{\mathbf{q}}}\,\frac{\partial \ddot{\mathbf{q}}}{\partial \mathbf{c}_k} = \frac{\mathrm{sign}(\ddot{\mathbf{q}})}{\ddot{\mathbf{q}}^{\max}}\,\frac{1}{T^2}\,\ddot{B}_k. \tag{16}$$

The other constraints are more difficult to derive but ultimately are simply functions of $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$. By the chain rule, their gradients with respect to the control points are expressed as combinations of the analytic mappings $\partial\mathbf{q}/\partial\mathbf{c}_k$, $\partial\dot{\mathbf{q}}/\partial\mathbf{c}_k$, and $\partial\ddot{\mathbf{q}}/\partial\mathbf{c}_k$. As an example, torque depends on all derivatives

$$\frac{\partial \mathbf{g}^{\tau}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\tau}}{\partial \mathbf{q}}\,\frac{\partial \mathbf{q}}{\partial \mathbf{c}_k} + \frac{\partial \mathbf{g}^{\tau}}{\partial \dot{\mathbf{q}}}\,\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{c}_k} + \frac{\partial \mathbf{g}^{\tau}}{\partial \ddot{\mathbf{q}}}\,\frac{\partial \ddot{\mathbf{q}}}{\partial \mathbf{c}_k} \tag{17}$$

where the partial derivatives $\partial\mathbf{g}^{\tau}/\partial\mathbf{q}$, $\partial\mathbf{g}^{\tau}/\partial\dot{\mathbf{q}}$, and $\partial\mathbf{g}^{\tau}/\partial\ddot{\mathbf{q}}$ are obtained by finite difference.

Other global constraints follow the same pattern with fewer terms:

$$\frac{\partial \mathbf{g}^{\text{TCP}}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\text{TCP}}}{\partial \mathbf{q}}\,\frac{\partial \mathbf{q}}{\partial \mathbf{c}_k} + \frac{\partial \mathbf{g}^{\text{TCP}}}{\partial \dot{\mathbf{q}}}\,\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{c}_k}, \tag{18}$$

$$\frac{\partial \mathbf{g}^{\text{self}}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\text{self}}}{\partial \mathbf{q}}\,\frac{\partial \mathbf{q}}{\partial \mathbf{c}_k}, \tag{19}$$

$$\frac{\partial \mathbf{g}^{\text{coll}}}{\partial \mathbf{c}_k} = \frac{\partial \mathbf{g}^{\text{coll}}}{\partial \mathbf{q}}\,\frac{\partial \mathbf{q}}{\partial \mathbf{c}_k}. \tag{20}$$

Thus, $\mathbf{g}^{\tau}$ depends on $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, $\mathbf{g}^{\text{TCP}}$ on $(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{g}^{\text{self}}$ and $\mathbf{g}^{\text{coll}}$ only on $\mathbf{q}$. All are vector-valued functions and their gradients reuse the same analytic building blocks.

The elements of the last row of the gradient matrix $\nabla\mathbf{g}(\mathbf{z})$ are $\partial\mathbf{g}/\partial\mathbf{T}$. Some partial derivatives can be computed as

$$\begin{aligned} \frac{\partial \mathbf{q}}{\partial T} &= 0, \\ \frac{\partial \dot{\mathbf{q}}}{\partial T} &= -\frac{\dot{\mathbf{q}}}{T}, \\ \frac{\partial \ddot{\mathbf{q}}}{\partial T} &= -\frac{2\,\ddot{\mathbf{q}}}{T}. \end{aligned} \tag{21}$$

Using (6) and (21), the decoupled bound constraints yield

$$\frac{\partial \mathbf{g}^{q}}{\partial T} = 0, \tag{22}$$

$$\frac{\partial \mathbf{g}^{\dot{q}}}{\partial T} = -\frac{|\dot{\mathbf{q}}|}{\dot{\mathbf{q}}^{\max} T}, \tag{23}$$

$$\frac{\partial \mathbf{g}^{\ddot{q}}}{\partial T} = -\frac{2\,|\ddot{\mathbf{q}}|}{\ddot{\mathbf{q}}^{\max} T}. \tag{24}$$

For coupled constraints, the chain rule yields

$$\frac{\partial \mathbf{g}^{\tau}}{\partial T} = -\frac{1}{T}\left(\frac{\partial \mathbf{g}^{\tau}}{\partial \dot{\mathbf{q}}}\,\dot{\mathbf{q}} + 2\,\frac{\partial \mathbf{g}^{\tau}}{\partial \ddot{\mathbf{q}}}\,\ddot{\mathbf{q}}\right), \tag{25}$$

$$\frac{\partial \mathbf{g}^{\text{TCP}}}{\partial T} = -\frac{1}{T}\frac{\partial \mathbf{g}^{\text{TCP}}}{\partial \dot{\mathbf{q}}}\,\dot{\mathbf{q}}, \tag{26}$$

$$\frac{\partial \mathbf{g}^{\text{self}}}{\partial T} = 0, \tag{27}$$

$$\frac{\partial \mathbf{g}^{\text{coll}}}{\partial T} = 0. \tag{28}$$

where $\partial\mathbf{g}^{\tau}/\partial\dot{\mathbf{q}}$, $\partial\mathbf{g}^{\tau}/\partial\ddot{\mathbf{q}}$, and $\partial\mathbf{g}^{\text{TCP}}/\partial\dot{\mathbf{q}}$ were previously obtained by finite difference for Eq. (17).

This provides a notable gain in computation time, given that, instead of scaling with $N_j(K-(p+1))$, it only scales with N_j. We get one finite difference per joint per kinematic variable ($\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$).

C. SpdUp 3: Constraint Computation by Knot-Span

The third speedup reduces the number of constraints, thereby decreasing the cost of gradient evaluations and the size of the optimization problem passed to the SQP solver. We still evaluate every constraint at every discretized point along the trajectory, but for each knot-span and each constraint type, we retain only the worst violation (e.g., worst $\mathbf{g}^q$ per joint, worst $\mathbf{g}^{\tau}$ per joint, worst $\mathbf{g}^{coll}$ per link, etc.). We then compute the constraint gradient only at these worst-case

samples, turning per-point gradient computation into per-span computation without risking missed violations. Ultimately, this leads to a reduced number of gradient evaluations.

The order reduction technique reduces the number of constraints passed to the SQP solver from being proportional to the number of sample points, $O(N_{pt})$, to the number of knot-spans, which is directly affected by the number of control points, $O(K)$. While accurately finding the worst-case violation still requires computing the constraints with a fine sampling resolution (an $O(N_{pt})$ operation), *SpdUp 3* can then compute its gradient, $\nabla \mathbf{g}(\mathbf{z})$, with a cost that is independent of sampling density.

Consequently, we can afford to sample finely to ensure accuracy while still reducing the number of constraints for the solver and gradient computations by a factor of $N_{pt}/(K-p)$, which generally leads to an order of magnitude reduction. Reducing the number of constraints given to the solver has the added benefit of making its subproblem smaller and its operations faster.

V. Comparison Methodology

The proposed speedup methods were evaluated on a simulated Universal Robot UR5e ($N_j = 6$) in an instance of MotionBenchMaker's [26] kitchen environment where all doors are open. Even though most boxes are axis-aligned, note that we use only OBBs to improve the fidelity of our results to real-life scenarios. Kinematics, dynamics, and joint limits were extracted from the URDF and the collision model from the SRDF available in the official *ros-industrial* repository [27]. To enforce collision constraints, each link was approximated by a least-volume capsule, defined by a line segment and a radius, fitted from the SRDF data. The UR5e manipulator consists of $N_{link} = 7$ capsules, while the scene is composed of $N_{obs} = 14$ collision objects that are represented by OBBs.

Eight start and eight goal configurations were selected, yielding a total of 64 benchmark tasks. Fig. 2 shows the UR5e in the kitchen environment, along with the eight start (red) and eight goal (green) end-effector poses. The corresponding joint configurations were obtained by solving the inverse kinematics for all 16 poses.

Note: although two start positions are very close together, different inverse kinematics solutions were taken and thus we provide the optimizer with different tasks.

Four implementations (*baseline*, *SpdUp 1*, *SpdUp 2*, and our final proposed method, *SpdUp 3*) were benchmarked on three key metrics: *computation time*, *trajectory quality (i.e. its duration T)*, and *success rate*.

All benchmarks were executed single-threaded on an Intel Core i9-13900HX processor (5.4 GHz). The optimization vector $\mathbf{z}$ was initialized with entries sampled uniformly from $[-1, 1]$ and the initial duration T was drawn from $[0.1, 2.1]$. To ensure statistically robust results, each optimization task was solved 100 times, yielding a total of 6,400 runs per implementation. To ensure fairness, all four methods shared the same initial guess per run. We used a relatively fine resolution of 10 discretization points per knot-span, for a total of $N_{pt} = 110$ points.

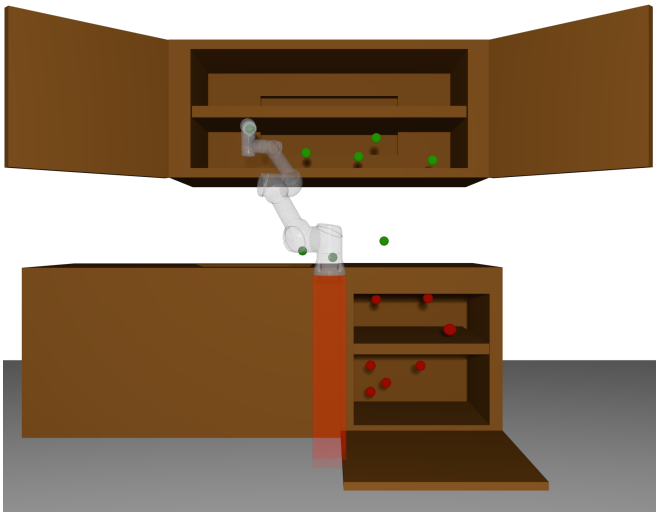

Fig. 2. UR5e in the kitchen environment, illustrating the eight start (red) and eight goal (green) end-effector poses.

Trajectories were represented as quintic clamped uniform B-splines with $K = 16$ control points, resulting in 11 knot-spans (see Fig. 1). Fig. 3 illustrates two representative trajectories.

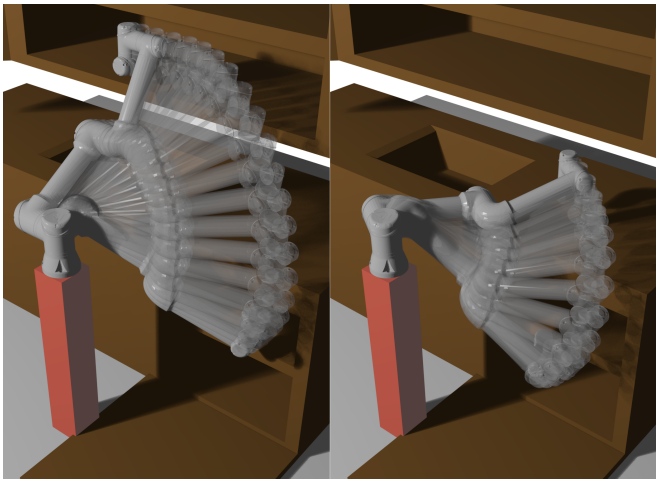

Fig. 3. Two representative trajectories generated for the simulated UR5e in the kitchen environment.

This setup highlights the main advantage of our approach. In the *baseline*, *SpdUp 1*, and *SpdUp 2* methods that rely on pointwise discretization, the total number of constraints is

$$N_{con} = 110 \times (4 \times 6 + 1 + 1 + 7) = 3,630.$$

By contrast, our *SpdUp 3* method enforces constraints on a per-span basis, dramatically reducing the problem size by an order of magnitude to

$$N_{con} = 11 \times (4 \times 6 + 1 + 1 + 7) = 363.$$

VI. Results

The aggregated results across all 64 tasks comparing the four methods (*baseline*, *SpdUp 1*, *SpdUp 2*, and *SpdUp 3*) are summarized in Fig. 4.

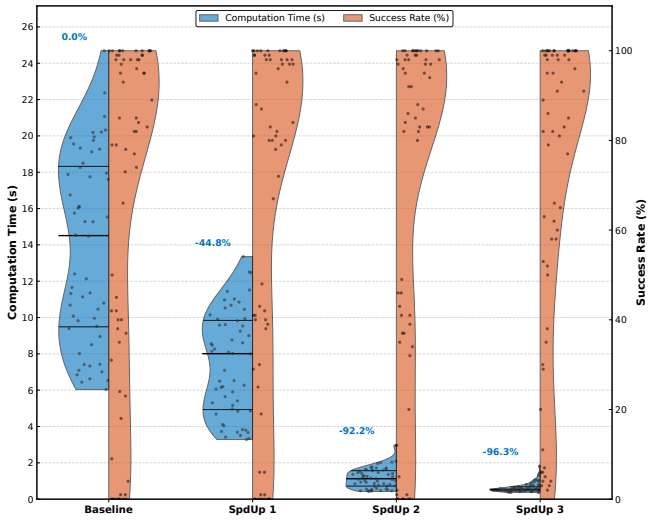

Fig. 4. Aggregated computation time [s] and success rate [%] for *baseline*, *SpdUp 1*, *SpdUp 2*, and *SpdUp 3* across all 64 tasks. *SpdUp 3* achieves a speedup factor of $26.9\times$. For computation time, the median, the first quartile, and the third quartile are represented by black horizontal lines.

For all three speedups, *trajectory quality (i.e. duration)* and *average success rate* do not differ significantly from the *baseline*. This shows consistency in the solver's convergence. These results show a clear progression of improvements relative to the *baseline* for *computation time*.

Tab. I presents the computation time distribution across the baseline and speedups for the gradient computation, the constraint computation, and the SQP computation, as well as the average number of required SQP iterations.

TABLE I

Computation time distribution and number of SQP iterations (N_{eval}) across speedups.

	Baseline	SpdUp 1	SpdUp 2	SpdUp 3
Gradients	92% (11.78 s)	84% (5.62 s)	11% (0.110 s)	9% (0.050 s)
Constraints	2% (0.259 s)	4% (0.271 s)	20% (0.200 s)	77% (0.424 s)
SQP	6% (0.777 s)	12% (0.812 s)	69% (0.692 s)	14% (0.077 s)
N_{eval}	42	43	42	74

The computation of the gradients dominated the computation time in the *baseline* algorithm (11.78 seconds) and was reduced to 50 milliseconds after applying the proposed computation speedups. While the first two speedups keep a similar average of $42-43$ SQP iterations, equivalent to that of the *baseline*, iterations increase to 74 on average for *SpdUp 3*. Since less information is given to the optimizer, this is consistent with the expected outcome. However, this increase in iterations is worthwhile, since it allows for each iteration to be much faster, leading to a lower overall computation time.

SpdUp 1 reduces computation time consistent with theoretical expectations. The five central knot-spans each have $p+1$ active mutable points. As a result, gradient evaluations per constraint drop from $(K-6)\times N_{pt}$ in the *baseline* to $(p+1)\times N_{pt}$ with *SpdUp 1*. Furthermore, the first three knot-spans have only three, four, and five active mutable points in order to enforce boundary conditions as described in Sections III-A and III-B. Considering the active control points averaged over all knot-spans, this yields a theoretical reduction of 50% with $K=16$ and $p=5$. Considering that the gradient computation accounted for $\sim 92\%$ of total computation time, the measured 52.3% reduction in gradient computation time therefore falls within a plausible range. This produces a 44.8% reduction in total computation time. However, gradient computation still accounts for $\sim 84\%$ of the computation time.

SpdUp 2 further reduces the gradient computation time. In addition to the factors described for *SpdUp 1*, *SpdUp 2* eliminates most finite-difference evaluations for constraint gradients. For decoupled joint constraints (position, velocity, and acceleration), analytic gradients are available at negligible computation cost. For joint-coupled constraints, instead of computing $(K-6)\times N_j$ finite-difference gradients, only $3\times N_j$ (with respect to $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$) are required, a reduction of 70% with $K=16$. Moreover, hybrid gradients are computed in-place during constraint evaluation, avoiding redundant passes and leveraging constraint-specific information (e.g., collisions are evaluated only against the nearest collision object). This enables further savings: instead of $N_{obs}\times N_{link}$ distance queries per finite difference, only $1\times N_{link}$ is required, yielding an approximate 93% reduction in collision computations for the gradient with $N_{obs}=14$. According to our tests, external collision constraints account for roughly 96% of total constraint cost, so their impact is significant. Collectively, these improvements explain the measured 99.1% reduction in gradient computation time (92.2% reduction in total computation time). Importantly, *SpdUp 2* also reduces the proportion of time spent on gradient evaluations from $\sim 84\%$ after *SpdUp 1* to $\sim 11\%$, showing that gradients are no longer the computation bottleneck.

SpdUp 3 reduces the computation time associated with the SQP solver by reducing the problem size. Although it is difficult to ascertain the precise theoretical gain, the improvement can be attributed to a reduction in the number of constraints, which in turn speeds up the quadratic subproblem solved at each SQP iteration. With $N_{pt}=110$ and $K=16$, the problem size is reduced by 90%. After *SpdUp 3*, gradient computations account for only $\sim 9\%$ of runtime, while constraint evaluation and SQP overhead become the dominant costs ($\sim 77\%$ and $\sim 14\%$, respectively). A reduction in problem size was accompanied by an increase in the average number of iterations, as reported in Table I, which is explained by the decrease in information given to the SQP solver.

Overall, the progression from *baseline* to *SpdUp 3* demonstrates how exploiting control-point locality, replacing finite differences with analytic and hybrid gradients, and reducing the number of active constraints together yield 96.3% reduction in total computation time, a **26.9x** improvement, while preserving trajectory quality and success rate.

VII. Conclusion

This paper introduced a structure-aware, discrete-time trajectory optimization framework that leverages the analytical and local control properties of quintic clamped uniform B-splines to speed up constraint-gradient computations in SQP. Three cumulative speedups were proposed. Firstly (*SpdUp 1*), we exploited per-joint separability and B-spline local control to avoid computing the gradient of constraints when we know it to be zero from context. Secondly (*SpdUp 2*), we replaced finite differences on control points with analytic gradients for joint-decoupled constraints and a hybrid chain-rule method for joint-coupled constraints (torque, TCP speed, and collisions). Finally (*SpdUp 3*), we reduced gradient and SQP solver workloads by aggregating constraints per knot-span and differentiating only at relevant samples (*i.e.* u_i at worst constraint).

On a simulated UR5e case study with 64 tasks in a cluttered environment, these speedups reduced computation time by 44.8%, 92.2%, and 96.3% (a **26.9x** reduction in computation time) for *SpdUp 1*, *SpdUp 2*, and *SpdUp 3*, respectively, relative to a finite-difference *baseline*. These were achieved while maintaining or slightly improving trajectory quality and success rates. The gains stem from bounded sparsity, analytic reuse of shared $\partial\mathbf{q}/\partial\mathbf{c}_k$, $\partial\dot{\mathbf{q}}/\partial\mathbf{c}_k$ and $\partial\ddot{\mathbf{q}}/\partial\mathbf{c}_k$, and an order reduction technique that preserves constraint fidelity.

Future work is focused on improving the computation time and convergence of the algorithm. Note that the constraint functions themselves can be interchanged and the speedups will still be valid. Therefore, improving the efficiency of constraint computation is a viable simultaneous avenue for improving the computation time of trajectory optimization algorithms. Secondly, since SQP is a deterministic algorithm, it is highly dependent on its initial guess. Improving the quality of the initial guess has the potential to improve both the computation time and convergence of the algorithm.

References

[1] C. De Boor and C. De Boor, *A practical guide to splines*. springer New York, 1978, vol. 27.

[2] L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed. Berlin: Springer, 1997.

[3] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[4] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," in *Nonlinear model predictive control: towards new challenging applications*. Springer, 2009, pp. 391–417.

[5] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. Philadelphia: SIAM, 2010.

[6] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli, "Automatic differentiation of rigid body dynamics for optimal control and estimation," *Advanced Robotics*, vol. 31, no. 22, pp. 1225–1237, 2017.

[7] S. Lavalle, "Rapidly-exploring random trees : a new tool for path planning," *Research Report 9811*, 1998.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[9] P. Švestka and M. H. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125–152, 1998.

[10] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic foundation of robotics VIII*. Springer, 2009, pp. 449–464.

[11] M. G. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics Auton. Syst.*, vol. 100, pp. 171–185, 2018.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[13] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494.

[14] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[15] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[16] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs," in *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016.

[17] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

[18] L. Tang, H. Wang, Z. Liu *et al.*, "A real-time quadrotor trajectory planning framework based on b-spline and nonuniform kinodynamic search," *Journal of Field Robotics*, vol. 38, no. 3, pp. 452–475, 2021.

[19] B. Rezali, B. Ibari, M. Hebali, M. Berka, M. Bennaoum, K. Bouzgou, R. Ayad, and L. Benchikh, "Optimal trajectory planning for industrial robots: Minimizing time, jerk, and energy consumption using lstm for energy profile modeling," *Journal of Vibration and Control*, p. 10775463251333481, 2025.

[20] Y. Zhou, G. Han, Z. Wei, Z. Huang, and X. Chen, "A time-optimal continuous jerk trajectory planning algorithm for manipulators," *Applied Sciences*, vol. 13, no. 20, 2023.

[21] C. Sommer, V. Usenko, D. Schubert, N. Demmel, and D. Cremers, "Efficient derivative computation for cumulative b-splines on lie groups," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 145–11 153.

[22] Y. Chen and L. Li, "Collision-free trajectory planning for dual-robot systems using b-splines," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417728021, 2017.

[23] S. G. Johnson and J. Schueller, "Nlopt: Nonlinear optimization library," *Astrophysics Source Code Library*, pp. ascl–2111, 2021.

[24] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[25] D. Kraft, "Algorithm 733: Tomp–fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[26] C. Chamzas, C. Quintero-Peña, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "MotionBenchMaker: A tool to generate and benchmark motion planning datasets," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 882–889, 2022.

[27] ROS-Industrial, "universal_robot," GitHub repository, 2025, [Online; accessed 15-September-2025]. [Online]. Available: https://github.com/ros-industrial/universal_robot