Smooth Point-to-Point Trajectory Planning in $SE(3)$ with Self-Collision and Joint Constraints Avoidance

Reinhard Grassmann, Lars Johannsmeier and Sami Haddadin

***Abstract*— In this paper we introduce a novel point-to-point trajectory planner for serial robotic structures that combines the ability to avoid self-collisions and to respect motion constraints, while complying with the requirement of being C^4 continuous. The latter property makes our approach also suited for 4^{th} order dynamics flexible joint robots, which gained significant practical relevance recently. In particular, we address the problem of generating a smooth, kinematically nearly time-optimal $SE(3)$ trajectory while simultaneously avoiding potential collisions of the robot end-effector with its base as well as respecting the Cartesian unreachable states induced by the joint limits of the proximal kinematic structure.**

I. Introduction

Consider a robot manipulator that moves its end-effector from start pose to goal pose with a point-to-point (p2p) planner. Essentially there are two alternatives for achieving this. Namely, planning in joint space or in Cartesian space. Due to the fact that a planned path in Cartesian space may generate unreachable intermediate points [1], the robot manipulator could collide with itself or run into its mechanical joint limits. Consequently, the robot cannot follow a given trajectory, which will cause it to stop or produce an undesirable and potentially dangerous motion. Thus, it represents a safety risk to both itself as well as its environment including humans. In contrast, a planned path in joint space can avoid unreachable points with relatively small planning effort. However, Cartesian obstacles cannot be considered efficiently [2]. Furthermore, it may produce undesirable trajectories which are neither intuitive nor predictable for human co-workers.

a) State of the Art: In order to avoid self-collisions and calculate unreachable intermediate points, the Configuration Space approach [3] can be applied to different path planning algorithms [2], [4]. Also online optimization has been commonly used [5]. Noticeably, the problem of self-collision avoidance is theoretically solved, however, often requiring high computational loads [6], which is not necessarily useful or needed for all cases [4].

The generation of such trajectories is well covered by standard robotics textbooks [1] and is extensively studied in [7], [8]. Note that it is necessary for both the geometric path and the velocity profile to be continuous in order for the trajectory to be considered continuous. In trajectory planning especially two properties are of high relevance, namely smoothness and duration. Most of the trajectories in literature are designed for rigid industrial robots and show at most C^3 smoothness or less. For kinematically time-optimal motions trapezoidal-like velocity profiles are commonly used [7]. Usually, these profiles strictly adhere to motion constraints such as desired velocity. This may lead to overshooting in case the given trajectory is too short. A common countermeasure for this is to recalculate the motion constraints and the total displacement in an iterative way (see e.g. [9]) or the problem is neglected [10], [7]. Another method for constructing trapeziodal-like velocity profiles, which is based on convolution, is proposed in [10]. In [11], [12] a C^4 smooth trajectory was developed. However, the method requires many trajectory segments, resulting in increasing complexity w.r.t. the design of a feasible overall trajectory.

Furthermore, relatively few publications (e.g. [13]) cover the simultaneous interpolation of both position and orientation. Most approaches consider position or a single degree-of-freedem (DOF) only. Interpolation of orientation is only rarely treated. However, it is indeed well known and accepted that quaternions are so far best suited for representing [14], [15] as well as interpolating [16] rotations. Slerp (spherical linear interpolation) [17] is the standard for interpolation between two rotations. However, it generates C^0-smooth trajectories, which are not sufficient for robotic applications due to the fact that such trajectories cause oscillations and result in low performance tracking. In [9] a quaternion based trajectory planner is developed which is jerk-bounded and thus C^3-smooth.

b) Contribution and Organization: In the present work we introduce a complete approach for simultaneous translation and orientation interpolation. We propose a trajectory generator, which employs a simple, yet effective and smooth trajectory that respects a defined virtual obstacle. In particular, following contributions are made:

- A C^4 smooth trapezoidal-like path velocity profile with three segments is developed and its formalism provided.
- Interpolation of position and orientation in Cartesian space, taking into account motion constraints such as maximum velocity and acceleration.
- A novel quaternion-based interpolator is introduced that generalizes the well-known Slerp method [17].
- A new path generator is introduced that calculates a feasible path w.r.t. self-collisions caused by the end-effector with the base prior to execution. Potential collisions are avoided by making use of the Problem of Apollonius.
- An interpolator framework is formulated that combines the other contributions for application to a 7-DOF robot with only little parameterization effort.

Figure 1 depicts the overall organization of the paper. In Sec. II the considered problem is defined. Section III describes the proposed trajectory planner. A comparison with

R. Grassmann is with Laboratory for Continuum Robotics, Faculty of Mechanical Engineering, Gottfried Wilhelm Leibniz Universität Hannover, 30167 Hannover, Germany grassmann@lkr.uni-hannover.de

L. Johannsmeier and S. Haddadin are with Munich School of Robotics and Machine Intelligence, Chair of Robotics Science and Systems Intelligence, Department of Electrical and Computer Engineering, Technical University of Munich, 80797 Munich, Germany firstname.lastname@tum.de

This work has been conducted, when all authors were still with Institute of Automatic Control at Gottfried Wilhelm Leibniz Universität Hannover.

 DOI: 10.1109/IROS.2018.8594339

two different standard p2p trajectory planners is carried out in Sec. IV. Finally, the paper concludes in Sec. V.

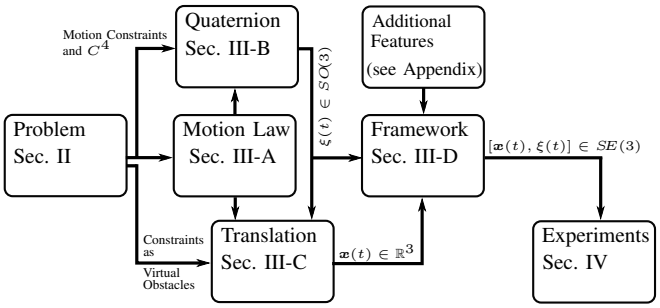

Fig. 1: Organization of the paper.

II. Problem Statement

In this section, we formalize the concept of unreachable intermediate points of the robot end-effector when moving in Cartesian space between a start pose $\boldsymbol{x}_{\text{start}}$ and a goal pose $\boldsymbol{x}_{\text{goal}}$. Also, we motivate the need for C^4 continuous trajectories in flexible-joint robotics. Note that in Sec. II-A only translation is considered for sake of clarity, the extension to orientation is provided in Sec. II-C.

A. Self-Collision and Smoothness for 6-DOF

The class of systems we consider in this paper is a flexible joint robot with six joints, indicated by dashed joint axes in Fig. 2a. Each of its joint angles q_i has inequality constraints due to mechanical end limits, restricting the joint range to interval $[q_{i,\text{lb}}, q_{i,\text{ub}}]$. For sake of simplicity and practical relevance, we assume $q_{i,\text{ub}} = -q_{i,\text{lb}}$.

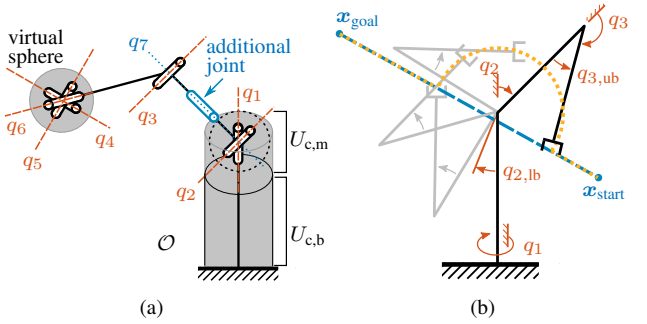

Fig. 2: (a) Exemplary 6-DOF manipulator (dashed joint axes) with extension to the 7-DOF case (dotted joint axis), (b) Cartesian motion imposed by joint limitation $q_{3,\text{ub}}$. The dashed line is the desired trajectory, the dotted line is the possible deviation caused for running into joint limits and being pushed back.

Specifically, we make two basic assumptions in order to restrict the problem practically. First, the first two joint axes are assumed to intersect, and the second and third joint axes to be genuine parallel. Secondly, we assume that the first link is cylindrically shaped or at least this is a resonable approximation.

In Fig. 2b a desired trajectory that leads from $\boldsymbol{x}_{\text{start}}$ to $\boldsymbol{x}_{\text{goal}}$ in Cartesian space is shown (dashed line). The dotted line depicts a possible feasible movement of the robot. Note that this is only one possible movement e.g. produced by a compliant Cartesian impedance controlled robot. In this case the end-effector moves along the upper Cartesian constraint imposed by th joint constraint $q_{3,\text{ub}}$ of q_3. The real behavior of the robot when moving on this path obviously depends on many factors. A position controlled robot would usually simply stop and abort the motion entirely.

From this observation and based on the stated assumptions, we define the following sets of unreachable points.

Definition 1: The set $U_{\text{j,m}}$ contains all unreachable positions w.r.t. mechanical end stops in joint space of the 2nd and 3rd joint. $U_{\text{c,m}}$ is its mapping to Cartesian space. ■

Definition 2: $U_{\text{c,b}}$ is the set containing all Cartesian points belonging to a cylindrical shape that encompasses the robot's first link. $U_{\text{j,b}}$ is its representation in joint space. ■

Definition 3: We define $U_{\text{j}} = U_{\text{j,m}} \cup U_{\text{j,b}}$ as the set of all unreachable points in joint space and $U_{\text{c}} = U_{\text{c,m}} \cup U_{\text{c,b}}$ as the set of all unreachable points in Cartesian space. We also call these sets unreachable intermediate points. ■

We approximate $U_{\text{c,m}}$ with a sphere with radius r_{m} and center point $\boldsymbol{x}_m$, which coincides with the intersection of the first two joints. $U_{\text{c,b}}$ is a cylinder with radius r_{b} and a central axis that is collinear to the first joints axis.

Definition 4: From these definitions we can define the virtual obstacle $\mathcal{O}$, see Fig. 2a. $\mathcal{O}$ is a cylinder with radius $r_{\mathcal{O}} = \max\{r_{\text{m}}, r_{\text{b}}\}$ and height $h_{\mathcal{O}} = \boldsymbol{x}_{\text{m},z} + r_{\text{m}}$. ■
Note that subsequently we only use the task space representation of the virtual obstacle.

Now that the virtual obstacle is defined, the robot requires the capability to automatically avoid it. Since it is more intuitive to represent the obstacle in Cartesian space and joint trajectories are known to be counterintuitive for humans, we subsequently design a Cartesian trajectory planner, for which the overall problem statement is as follows:

Find a C^4 continuous Cartesian trajectory from start pose to goal pose while avoiding the virtual obstacle $\mathcal{O}$.

Generally, smooth trajectories are desirable because they avoid structural oscillations [7], [13], increase accuracy in the tracking of the end effector [18] and reduce energy consumption [9]. Moreover, as a benefit to human-robot collaboration, they "appear" more natural [13], resulting in improved user acceptance.

B. Notions on the Extension to 7-DOF

The previous assumptions can be easily extended to the standard kinematics 7-DOF case. Note that for sake of simplicity we insert the additional 7th joint (dotted joint axis) after the 2nd joint in order to avoid restructuring the example from Fig. 2a.

The set $U_{\text{c,b}}$ remains since it only depends on the first link. The set $U_{\text{c,m}}$ is still the same sphere because when moving the additionally inserted 7th joint the end effector follows the surface of the sphere $U_{\text{c,m}}$, see Fig. 3. In conclusion, the 3D case holds for both kinematically redundant (7-DOF) and non-redundant robot manipulators that have the well-established structure depicted in Fig. 2a.

C. End-Effector Orientation

Incorporating the orientation of the robot end-effector can be done in a conventional fashion by encasing it with a virtual sphere that is configuration independent. The center of this sphere is located in the geometric center of the end-effector. In our case, and for sake of simplicity, we assume the center point of the end-effector to be at the intersection of the last three joints, see Fig. 2a.

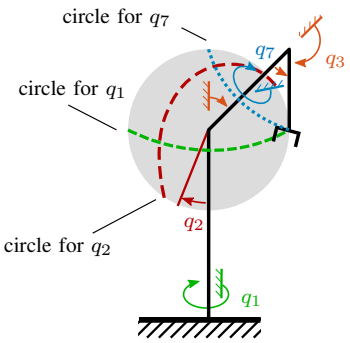

Fig. 3: Effect of the 7th joint on the sphere, resulting from joint limits.

Shrinking the radius of this virtual sphere into a point has no effect with regard to potential self-collisions if the radius and height of the virtual obstacle $\mathcal{O}$ are increased by the same amount. Thus, we do not have to consider the orientation explicitly in the following. Note the similarity to the Configuration Space approach [2].

Obviously, a more accurate approach with regard to exploiting available workspace would involve matching exact geometric shapes for the end-effector. However, this also results in significantly more complicated algorithms for detecting intersections with the virtual obstacle $\mathcal{O}$ and subsequently avoiding them. We intend to address this topic in future work.

III. Approach

In this section we introduce our approach to solve the problem of planning a Cartesian C^4 smooth trajectory in the presence of a virtual obstacle $\mathcal{O}$. First, we define a motion law. Then, taking this law into account, we present a group of geometric paths that can avoid $\mathcal{O}$, while complying with C^4 requirement. Finally, we describe the framework as a whole, unifying the components in a logical decisional structure.

Note that the geometric paths for the translation and orientation components are defined separately and then synchronized in order to design the complete desired path. Without a separate definition of both components an undesired screw motion could be generated and its translation component is generally not a straight path [13]. Furthermore, since obstacle avoidance for orientation is considered in Sec. II-C, it is only necessary to design the geometric paths for the translation such that $\mathcal{O}$ is avoided.

A. Motion Law

The following motion law is built based on a trapezoidal-like velocity profile with three segments. They are denoted as *lift-off*, *cruise* and *set-down*. For sake of simplicity, we assume the motion constraints to be symmetric w.r.t. upper and lower values. In order to achieve a C^4 smooth trajectory the velocity profile is designed with C^3 polynomial functions for the first and the third segment and a constant second segment with durations $T_{\text{lift-off}}$, $T_{\text{set-down}}$ and T_{cruise}, respectively. Furthermore, $T := T_{\text{lift-off}} + T_{\text{cruise}} + T_{\text{set-down}}$ is the total trajectory duration.

The normalized polynomial function $v(\zeta) \in [0,1]$ for the *lift-off* and the *set-down* velocity segments is

$$v(\zeta) = -20\zeta^7 + 70\zeta^6 - 84\zeta^5 + 35\zeta^4, \tag{1}$$

where $\zeta \in [0,1]$ is the time primitive [7]. The coefficients in (1) fulfill the null boundary condition, meaning that the first three derivatives of (1) are zero at $\zeta = 1$ and $\zeta = 0$. By integrating (1) w.r.t. ζ we obtain

$$V(\zeta) = -2.5\zeta^8 + 10\zeta^7 - 14\zeta^6 + 7\zeta^5, \tag{2}$$

which is utilized for the position segments. The three segments of the velocity profile $\dot{s}$ are designed as

$$\dot{s}_{\text{lift-off}}(\tau) = \lambda v_{\max} v(\tau), \tag{3}$$
$$\dot{s}_{\text{cruise}}(\tau) = \lambda v_{\max} = \text{const.}, \tag{4}$$
$$\dot{s}_{\text{set-down}}(\tau) = \lambda v_{\max} v(1-\tau), \tag{5}$$

where λ is a scaling factor that is properly defined in (11), $v_{\max}$ is the maximum velocity. $\tau(t)$ with $t \in [0,T]$ is defined as

$$\tau = \begin{cases} \dfrac{t}{T_{\text{lift-off}}} & \text{for } 0 \le t < T_{\text{lift-off}} \\ \dfrac{t - T_{\text{lift-off}}}{T_{\text{cruise}}} & \text{for } T_{\text{lift-off}} \le t < T - T_{\text{set-down}} \\ \dfrac{t - T_{\text{lift-off}} - T_{\text{cruise}}}{T_{\text{set-down}}} & \text{for } T - T_{\text{set-down}} \le t \le T \end{cases} \tag{6}$$

The position profile is obtained by integrating (3), (4) and (5), leading to the definition of the motion law $s(t)$.

Definition 5: $s(t)$ is a piece-wise defined function

$$s(t) = \begin{cases} 0 & \text{for } t < 0 \\ s_{\text{lift-off}}(t) & \text{for } 0 \le t < T_{\text{lift-off}} \\ s_{\text{cruise}}(t) & \text{for } T_{\text{lift-off}} \le t < T_{\text{cruise}} + T_{\text{lift-off}} \\ s_{\text{set-down}}(t) & \text{for } T_{\text{cruise}} + T_{\text{lift-off}} \le t < T \\ L & \text{for } T < t, \end{cases} \tag{7}$$

with

$$s_{\text{lift-off}}(t) = \lambda v_{\max} T_{\text{lift-off}} V(\tau(t)), \tag{8}$$
$$s_{\text{cruise}}(t) = \lambda v_{\max} (T_{\text{cruise}} \tau(t) + T_{\text{lift-off}}/2), \tag{9}$$
$$s_{\text{set-down}}(t) = L - \lambda v_{\max} T_{\text{set-down}} V(1-\tau(t)), \tag{10}$$

where L is the total displacement of the geometric path. ■

Examples generated by the motion law $s(t)$ are depicted in Fig. 4. Note that $s(t)$ is not a path primitive since it is not generally defined for $[0,1]$ but for $[0,L]$ instead, as it is not independent of L.

In case L is too short such that the maximum velocity $v_{\max}$ and acceleration $a_{\max}$ cannot be reached, both motion constraints ($v_{\max}$ and $a_{\max}$) are automatically adapted by the scaling factor λ, which is defined as

$$\lambda = \begin{cases} 1 & \text{for } T_{\text{cruise}}(\lambda = 1) \ge 0 \\ \dfrac{16\, a_{\max}}{35\, v_{\max}^2} |L| & \text{otherwise.} \end{cases} \tag{11}$$

The durations of the respective segments are calculated as follows. Computing the maximum derivative from (3), which is an acceleration and can be set equal to $\lambda a_{\max}$, leads to

$$T_{\text{lift-off}} = T_{\text{set-down}} = \frac{35\, v_{\max}}{16\, a_{\max}}. \tag{12}$$

The scaling factor λ is canceled out for the first and third segment, leading to a constant duration w.r.t. λ. The duration T_{cruise} for the constant velocity part is given by

$$T_{\text{cruise}} = \frac{L}{\lambda v_{\max}} - \frac{35\, v_{\max}}{16\, a_{\max}}. \tag{13}$$

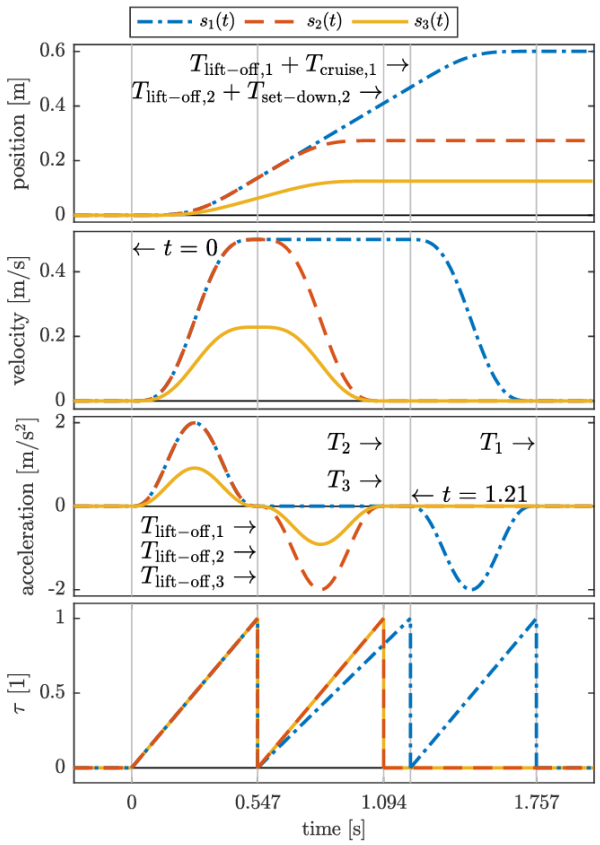

Fig. 4: Three examples of positions, velocities and accelerations of $s(t)$ with different total displacements L_1, L_2 and L_3, respectively. The first trajectory (L_1) has a constant velocity segment, while the second (L_2) does not. The third trajectory (L_3) is even shorter and cannot reach $v_{\max}$ and $a_{\max}$. Therefore, the maximal velocity and acceleration is scaled by λ.

Equation (13) can be found by combining and reordering the sum of the length, which can be obtained from (8)-(10). The velocity profiles are determined with $v_{\max}$ and $a_{\max}$. Although the maximum jerk $j_{\max}$ can be considered as well it would result in an over-determined equation system. In order to make use of it, we compare the required times $T_{\text{set-down, jerk}}$ and $T_{\text{set-down, acc}}$. Since $T_{\text{lift-off}} = T_{\text{set-down}}$ and $T_{\text{lift-off}}$ is obtained w.r.t. the maximum acceleration $a_{\max}$ from (12) we denote this quantity by $T_{\text{lift-off, acc}}$. Analogous to (12) we may calculate the required time as

$$T_{\text{lift-off,jerk}} = \sqrt{\frac{84}{5\sqrt{5}}} \sqrt{\frac{v_{\max}}{j_{\max}}}. \tag{14}$$

The new duration of segement *lift-off* and *set-down*, respectively, is then

$$T_{\text{lift-off}} = \max\left(T_{\text{lift-off,jerk}}, T_{\text{lift-off,acc}}\right) \tag{15}$$

Note that only either maximum acceleration $a_{\max}$ or maximum jerk $j_{\max}$ can be reached. In case that (14) is larger than (12), the maximum velocity and maximum acceleration need to be recalculated as

$$v_{\max} = \frac{L}{\lambda\left(T_{\text{cruise}} + T_{\text{lift-off}}\right)} \quad \text{and} \tag{16}$$

$$a_{\max} = \frac{35L}{\lambda 16 T_{\text{lift-off}}\left(T_{\text{cruise}} + T_{\text{lift-off}}\right)}. \tag{17}$$

(16) and (17) can be derived from (12) and (13), respectively. Note that (16) and (17) can be used to synchronize n degrees of freedom by rewriting them as

$$v_{\text{scal},i} = \frac{L_i}{\lambda\left(T_{\text{cruise,max}} + T_{\text{lift-off,max}}\right)} \quad \text{and} \tag{18}$$

$$a_{\text{scal},i} = \frac{35L_i}{\lambda 16 T_{\text{lift-off,max}}\left(T_{\text{cruise,max}} + T_{\text{lift-off,max}}\right)}, \tag{19}$$

where $T_{\text{cruise,max}} = \max_i\left(0, T_{\text{cruise},i}\right)$. The index $i = 1, \ldots, n$ indicates the respective degree of freedom in Cartesian space and the *max* subfix denotes the maximum duration of the respective segment considering all n degrees of freedom. This is used later to synchronize different components of a trajectory e.g. translation and orientation components or components of a straight line trajectory.

B. Geometric Path and Trajectory - Orientation

After having defined the motion law, let us now introduce the aforementioned geometric paths, starting with orientation. As mentioned in Sec. II the interpolation of the orientation is independent of the consideration of the virtual obstacle $\mathcal{O}$. Thus, it can be viewed independently from the translation and the obstacle.

a) Quaternion Interpolation: A unit quaternion [19] is a hypercomplex number denoted by $\xi = \eta + \epsilon_1 \imath + \epsilon_2 \jmath + \epsilon_3 k$ with property $\eta^2 + \epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 = 1$, where $\imath$, $\jmath$ and k follow Hamilton's rule $\imath^2 = \jmath^2 = k^2 = \imath\jmath k = -1$. A quaternionic conjugate of ξ is defined as $\xi^* = \eta - \epsilon_1\imath - \epsilon_2\jmath - \epsilon_3 k$, which is also the inverse ξ^{-1}. The associative and non-commutative product of two quaternions ξ and ξ' expands to

$$\begin{aligned}\xi\xi' = {} & \left(\eta + \epsilon_1\imath + \epsilon_2\jmath + \epsilon_3 k\right)\left(\eta' + \epsilon_1'\imath + \epsilon_2'\jmath + \epsilon_3' k\right) \\ = {} & \left(\eta\eta' - \epsilon_1\epsilon_1' - \epsilon_2\epsilon_2' - \epsilon_3\epsilon_3'\right) \\ & + \left(\eta\epsilon_1' + \eta'\epsilon_1 + \epsilon_2\epsilon_3' - \epsilon_2'\epsilon_3\right)\imath \\ & + \left(\eta\epsilon_2' + \eta'\epsilon_2 + \epsilon_3\epsilon_1' - \epsilon_3'\epsilon_1\right)\jmath \\ & + \left(\eta\epsilon_3' + \eta'\epsilon_3 + \epsilon_1\epsilon_2' - \epsilon_1'\epsilon_2\right)k. \end{aligned} \tag{20}$$

The embedding of a vector $\boldsymbol{p} = (x, y, z)^T \in \mathbb{R}^3$ is defined as $\rho = 0 + x\imath + y\jmath + zk$, which is needed later for rotating a vector. Such quantities are called pure quaternions.

Unit quaternions can be used to represent orientation, while quaternion multiplication can be used to retrieve the result of a rotation. Let ${}^{\text{base}}\xi_{\text{start}}$ and ${}^{\text{base}}\xi_{\text{goal}}$ denote the unit quaternion expressing the orientation of the two frames w.r.t. the base frame. The superscript denotes the frame in which a unit quaternion is expressed. And the subscript denotes the frame to which it refers. The result of the quaternion product

$${}^{\text{base}}\xi_{\text{d}}\left(\theta\right) = {}^{\text{base}}\xi_{\text{start}}\,{}^{\text{start}}\xi_{\text{d}}(\theta) \tag{21}$$

describes an interpolation between start orientation ${}^{\text{base}}\xi_{\text{start}}$ and the goal orientation ${}^{\text{base}}\xi_{\text{goal}}$. The transition between them is described by the quaternion ${}^{\text{start}}\xi_{\text{d}}(\theta)$. The quaternion ${}^{\text{start}}\xi_{\text{d}}$ complies with ${}^{\text{start}}\xi_{\text{d}}(\theta = 0) = 1$ and ${}^{\text{start}}\xi_{\text{d}}(\theta = \theta_L) = {}^{\text{start}}\xi_{\text{goal}}$. To achieve those conditions, the unit quaternion

$$\xi\left(\theta\right) = \cos\left(\theta\right) + \left(n_x\imath + n_y\jmath + n_z k\right)\sin\left(\theta\right) \tag{22}$$

is used, where $\boldsymbol{n} = (n_x, n_y, n_z)^T$ with $\|\boldsymbol{n}\|_2 = 1$ is a fixed axis and θ is the angle between the two given orientations.

The product ${}^{\text{start}}\xi_{\text{base}}\,{}^{\text{base}}\xi_{\text{goal}}$ yields the axis $\boldsymbol{n}$ and the goal angle θ_L in (22), where ${}^{\text{start}}\xi_{\text{base}}$ is the inverse to ${}^{\text{base}}\xi_{\text{start}}$.

Applying the proposed motion law from Def. 5 to the geometric path defined by (21) and (22) leads to

$$ {}^{\text{base}}\xi_{\text{d}}\left(\theta(t)\right) = {}^{\text{base}}\xi_{\text{start}}\,{}^{\text{start}}\xi_{\text{d}}\left(s\left(t\right)\right), \tag{23} $$

a C^4 continuous p2p trajectory in $SO(3)$ with $L = \theta_L$, $v_{\text{max}} = \omega_{\text{max}}$ and $a_{\text{max}} = \alpha_{\text{max}}$, being the orientation motion constraints. Note that other interpolation methods are also applicable such as (1), leading to $\theta(t) = \theta_L v(t/T)$.

Orientation motion constraints can easily be taken into account because the different axes of the angular velocity $\boldsymbol{\omega} = \left(\omega_x, \omega_y, \omega_z\right)^T$ and acceleration $\boldsymbol{\alpha} = \left(\alpha_x, \alpha_y, \alpha_z\right)^T$ w.r.t. the base frame are uncoupled and linear:

$$ \omega_x = n_x\dot{\theta}(t)\,, \quad \omega_y = n_y\dot{\theta}(t)\,, \quad \omega_z = n_z\dot{\theta}(t)\,, \tag{24} $$

$$ \alpha_x = n_x\ddot{\theta}(t)\,, \quad \alpha_y = n_y\ddot{\theta}(t)\,, \quad \alpha_z = n_z\ddot{\theta}(t)\,. \tag{25} $$

Equations (24) and (25) hold because the rotation axis $\boldsymbol{n}$ is fixed during interpolation. Note that if Euler angles are used, orientation motion constraints cannot be expressed in algebraic form [2]. Hence, none of the twelve sets of Euler angle triples (e.g. RPY) are able to take into account the orientation motion constraints efficiently.

b) Relation to Slerp: In the following, we show that (21) with the interpolated angle $\theta_L v(\zeta)$ is the generalized version of Slerp [17] w.r.t. the exponent h and its smoothness with $v(\zeta)$ being an arbitrary path primitive, while ζ is its time primitive. Slerp is defined as

$$ \text{Slerp}\left(\xi_0, \xi_1; h\right) = \xi_0\left(\xi_0^*\xi_1\right)^h, \tag{26} $$

where ξ_0, and ξ_1 are quaternions and h is linearly interpolated in the interval $[0, 1]$. The quaternion ξ_0^* is the quaternionic conjugate of ξ_0. The power of a quaternion raised to an arbitrary exponent h is taken from [20] as

$$ \xi^h = \cos\left(h\theta\right) + \left(n_x\imath + n_y\jmath + n_z k\right)\sin\left(h\theta\right), \tag{27} $$

where the angle θ is multiplied by the exponent $h \in \mathbb{R}$. Substituting θ in (22) with $\theta_L v(\zeta)$ results in the right side of (27), from which we obtain

$$ {}^{\text{start}}\xi_{\text{d}}\left(\theta_L v\left(\zeta\right)\right) \overset{(27)}{=} \left({}^{\text{start}}\xi_{\text{d}}\left(\theta_L\right)\right)^{v(\zeta)} \tag{28} $$

$$ = \left({}^{\text{start}}\xi_{\text{base}}\,{}^{\text{base}}\xi_{\text{goal}}\right)^{v(\zeta)}. \tag{29} $$

By comparing (29) with (26), we can conclude that (21) is identical to (26) if θ_L is interpolated with a linear path primitive i.e. $v(\zeta) = \zeta = t/T$. The linear path primitive unveils that Slerp in terms of (26) generates only a C^0 smooth trajectory. Furthermore, it inherits the property of being the shortest great arc between the two quaternions on the unit quaternion sphere.

One can say that the exponent h in (26) is the time primitive ζ of the path primitive $v(\zeta)$. Hence, an implementation of Slerp can be extended by inserting h into the path primitive i.e. $v(\zeta) = v(h)$, leading to smoother orientation interpolation (cf. (24) and (25)). A more pratical implementation of Slerp adapted from [17] is then given by

$$ {}^{\text{base}}\xi_{\text{d}}\left(\theta_L v\right) = \frac{\sin\left((1-v)\theta_L\right)}{\sin\left(\theta_L\right)}\,{}^{\text{base}}\xi_{\text{start}} + \frac{\sin\left(v\theta_L\right)}{\sin\left(\theta_L\right)}\,{}^{\text{base}}\xi_{\text{goal}}, \tag{30} $$

with $v = v(\zeta)$. By applying the substitution $h = v(\zeta)$ we may now generalize Slerp even to C^n. We may achieve the same result with our approach by using $\theta(t) = \theta_L v(t/T)$. It is important to note that we can use an arbitrary function for $\theta(t)$ such as Def. 5 and thus (21) is not limited to having a path primitive, i.e. $\theta(t) = \theta_L v(t/T)$, as it is the case with Slerp. Therefore, (21) is more general than Slerp can be.

C. Geometric Path and Trajectory - Translation

In the following we discuss three relevant cases of geometric paths regarding the avoidance of $\mathcal{O}$ and, therefore, unreachable intermediate points. As an initial situation we always have a starting position $\boldsymbol{x}_{\text{start}}$ and goal position $\boldsymbol{x}_{\text{goal}}$, both assumed to be outside of $\mathcal{O}$. For more readable notation we define $\mathcal{O}_{xy}$ as the projection of $\mathcal{O}$ onto the xy-plane of the robot base frame. Similarly, we define $\boldsymbol{x}_{\text{start},xy}$ and $\boldsymbol{x}_{\text{goal},xy}$ as the projection of $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ onto the same plane, respectively.

We distinguish three different cases regarding intersections of a line, leading from $\boldsymbol{x}_{\text{start}}$ to $\boldsymbol{x}_{\text{goal}}$ with $\mathcal{O}$.

1) The first case is that no intersection occurs between the obstacle and the line, see paragraph a).
2) In the second case the line intersects the obstacle and either $\boldsymbol{x}_{\text{start},xy}$ or $\boldsymbol{x}_{\text{goal},xy}$ lie in $\mathcal{O}_{xy}$, see paragraph b).
3) All other possible situations are dealt with by the third case, see paragraph c).

a) Straight Line Trajectory: The first case of possible intersections results is a straight line in Cartesian space. By applying the motion from Def. 5 we get

$$ \boldsymbol{x}_{\text{d}}(t) = \boldsymbol{x}_{\text{start}} + \begin{bmatrix} s_x(t) & s_y(t) & s_z(t) \end{bmatrix}^T \tag{31} $$

with $\begin{bmatrix} L_{\text{x}} & L_{\text{y}} & L_{\text{z}} \end{bmatrix}^{\text{T}} = \left\|\boldsymbol{x}_{\text{goal}} - \boldsymbol{x}_{\text{start}}\right\|_2$.

Note that we assign $v_{\text{max},i}$ and $a_{\text{max},i}$ component-wise. We may derive them from a desired path velocity v_{d} and acceleration a_{d} by making e.g. use of the Pythagorean theorem.

In order to synchronize the Cartesian axes, the scaling factor λ defined by (11) and the durations $T_{\text{lift-off,max}}$ and $\max\left(0, T_{\text{cruise,max}}\right)$ are computed w.r.t. the largest displacement $L_{\text{max}} = \max_i\left\{L_i\right\}$ using (18) and (19).

b) Circular Arc Trajectory: The second case creates a circular arc. When this case is projected to the 2D-case, $\boldsymbol{x}_{\text{start},xy}$ lies inside $\mathcal{O}_{xy}$ and $\boldsymbol{x}_{\text{goal},xy}$ outside of it. Note that, by constructing a path with linear and circular segments C^2 continuity cannot be obtained [7]. Therefore, we construct a circular arc path by reutilizing the quaternion-based interpolator (21). In order to make use of it, we need to introduce following considerations.

Given the well known sandwich product $\xi\rho\xi^* = \rho'$ which rotates an arbitrary vector $\boldsymbol{p}$ via the pure quaternion (see Sec. III-B for a recap) ρ into the rotated vector $\boldsymbol{p}'$ embedded into the pure quaternion ρ'. The multiplication with the conjugated unit quaternion ξ^* corresponds to a correction term [19]. This term is needed due to the fact that a single multiplication generally produces a non-vanishing real part η of the quaternion and scales the length of vector $\boldsymbol{p}$. If $\boldsymbol{p}$ is perpendicular to axis $\boldsymbol{n}$, no correction is required and the sandwich product can be simplified. The simplification can be $\rho' = \rho\xi^*$ or

$$ \rho' = \xi\rho, \tag{32} $$

which rotates vector $\boldsymbol{p}$ with the embedding ρ about the axis $\boldsymbol{n} = (n_x, n_y, n_z)^T$. Note that the full angle θ is required instead of $\theta/2$ due to the fact that no correction is needed.

In order to determine a circular arc, a midpoint $\boldsymbol{m}$, a start position $\boldsymbol{x}_{\text{start}}$ and a goal position $\boldsymbol{x}_{\text{goal}}$ may be used. Note that the calculation of $\boldsymbol{m}$ for our particular case is provided in Sec. III-D and the start and goal position are obviously given.

In order to simplify the calculation of the circular arc we transform $\boldsymbol{x}_{\text{start}}$, $\boldsymbol{x}_{\text{goal}}$ and $\boldsymbol{m}$ into a new frame $\mathbf{A}$ such that ${}^{\mathbf{A}}\boldsymbol{m}$ is the origin and ${}^{\mathbf{A}}\boldsymbol{x}_{\text{start}}$ coincides with the x-axis of $\mathbf{A}$ and the z-axis coincides with the cross product of ${}^{\mathbf{A}}\boldsymbol{x}_{\text{start}}$ and ${}^{\mathbf{A}}\boldsymbol{x}_{\text{goal}}$. Note that $\boldsymbol{m}$ has to coincide with the origin of $\mathbf{A}$ since we rotate ${}^{\mathbf{A}}\boldsymbol{x}_{\text{start}}$ into ${}^{\mathbf{A}}\boldsymbol{x}_{\text{goal}}$ as explained in (32).

Then we define the quaternion $\xi_{\text{arc}}(\psi(t))$ with the angle $\psi(t)$ around the z-axis of $\mathbf{A}$ which is interpolated via the motion law from in Def. 5

$$\psi(t) = s(t). \tag{33}$$

Note that $L = \psi_L$ can be obtained by calculating the angle between ${}^{\text{A}}\boldsymbol{x}_{\text{start}}$ and ${}^{\text{A}}\boldsymbol{x}_{\text{goal}}$. Furthermore, $v_{\max} = v_{\text{d}}/r_{\text{arc}}$ and $a_{\max} = a_{\text{d}}/r_{\text{arc}}$, where v_{d} and a_{d} are the path velocity and acceleration, respectively, and the radius r_{arc} is the length of ${}^{\mathbf{A}}\boldsymbol{x}_{\text{start}}$.

In order to obtain the current position ${}^{\mathbf{A}}\boldsymbol{x}_{\text{d}}(t)$ we embed $\psi(t)$ into a pure quaternion $\rho_{\text{d}}(t)$ via

$$\rho_{\text{d}}(t) = \xi_{\text{arc}}(\psi(t))\rho_{\text{start}}, \tag{34}$$

where ρ_{start} is the embedding of ${}^{\mathbf{A}}\boldsymbol{x}_{\text{start}}$. Now, we can retrieve ${}^{\mathbf{A}}\boldsymbol{x}_{\text{d}}(t)$ and transform it back to the robot base frame resulting in the eventual circular arc trajectory.

c) Helicoidal Trajectory: The third case covers all situations not met by the two other cases, meaning that the line between $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ intersects with $\mathcal{O}$ and both $\boldsymbol{x}_{\text{start},xy}$ and $\boldsymbol{x}_{\text{goal},xy}$ are outside of $\mathcal{O}_{xy}$. Note that a cross section of $\mathcal{O}$ with a plane that contains $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ as well as the shortest possible path between them while not entering $\mathcal{O}$ is in general a (partial) ellipse. Therefore, a circular arc path is not suitable, since it is impractical to handle it in this setting. However, a helicoidal path solves this problem.

Generating a helicoidal path is divided into two parts. First, the circular arc path is utilized by applying $\boldsymbol{x}_{\text{start},xy}$, $\boldsymbol{x}_{\text{goal},xy}$ and $\boldsymbol{m}$ with the z component set to zero. Second, a straight line path along the z-axis is generated based on the z components of $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$. Afterwards both trajectory parts are synchronized similar to the straight path case.

D. Assembling the system

Now that the motion law, all necessary geometric paths and the resulting trajectories have been defined, the system can be assembled into one framework. Figure 5 depicts the overall scheme.

Fist, we make use of a logical selection mechanism that decides which of the three aforementioned cases has occurred. The determination of the respective case is done by utilizing the problem of Apollonius which then selects the correct path strategy, see Fig. 5. Also, we show the calculation of a midpoint $\boldsymbol{m}$ for the case of the circular arc path and the helicoidal path.

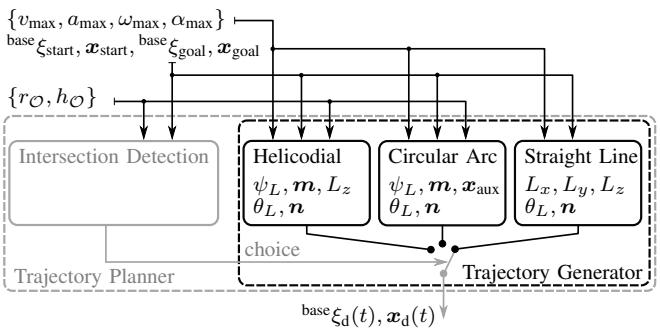

Fig. 5: The trajectoy planner chooses a suitable path depending on the relation of $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ with $\mathcal{O}$.

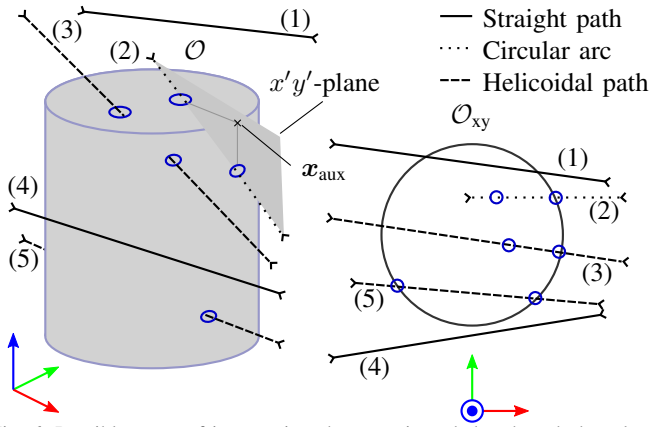

Fig. 6: Possible cases of intersections between intended path and obstacle. We distinguish between straight path (solid lines, (1), (4)), circular arc (dotted line, (2)) and helicoidal path (dashed lines, (3), (5)). The intersections are indicated by circles. Note that the depicted coordinate frame is not the robot base frame but only a visualization aid.

a) Problem of Apollonius: Finding a tangent circular arc path to the obstacle $\mathcal{O}$ is desirable. This requires the calculation of a midpoint $\boldsymbol{m}$ and leads to the so-called Problem of Apollonius. The Problem of Apollonius describes how to construct circles that are tangent to three given objects, where the objects represent any combination of points, lines, or circles [21]. We take advantage of two methods, namely the *PPP*-method and the *CPP*-method. The *PPP*-method requires three distinguishable points and provides the midpoint $\boldsymbol{m}$. By requiring one circle and two different points, the *CPP*-method provides midpoints $\boldsymbol{m}$ of two possible circles. Note that the desired circular arc path (segment of a circle) is selected choosing the shortest one.

b) Choice of geometric path: The trajectory planner will check for an intersection between the virtual obstacle $\mathcal{O}$ and the straight line path leading from $\boldsymbol{x}_{\text{start}}$ to $\boldsymbol{x}_{\text{goal}}$. The concrete method for detecting the intersection type is straightforward and uses simple geometry. Figure 6 depicts how the three cases which require the following steps can be distinguished.

1) *Straight line path*: No intersection is detected. A straight line path between $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ is constructed.
2) *Circular arc path*: If an intersection is detected and either $\boldsymbol{x}_{\text{start},xy}$ or $\boldsymbol{x}_{\text{goal},xy}$ lie inside $\mathcal{O}_{\text{xy}}$ a circular arc path on a suitable $x'y'$-plane is constructed. The $x'y'$-plane can be derived from $\boldsymbol{x}_{\text{start}}$, $\boldsymbol{x}_{\text{goal}}$ and an auxiliary point $\boldsymbol{x}_{\text{aux}}$ which is defined as the intersection point between $\mathcal{O}_{xy}$ and the line between $\boldsymbol{x}_{\text{start},xy}$ and $\boldsymbol{x}_{\text{goal},xy}$, see Fig. 6. From this the *PPP*-method as

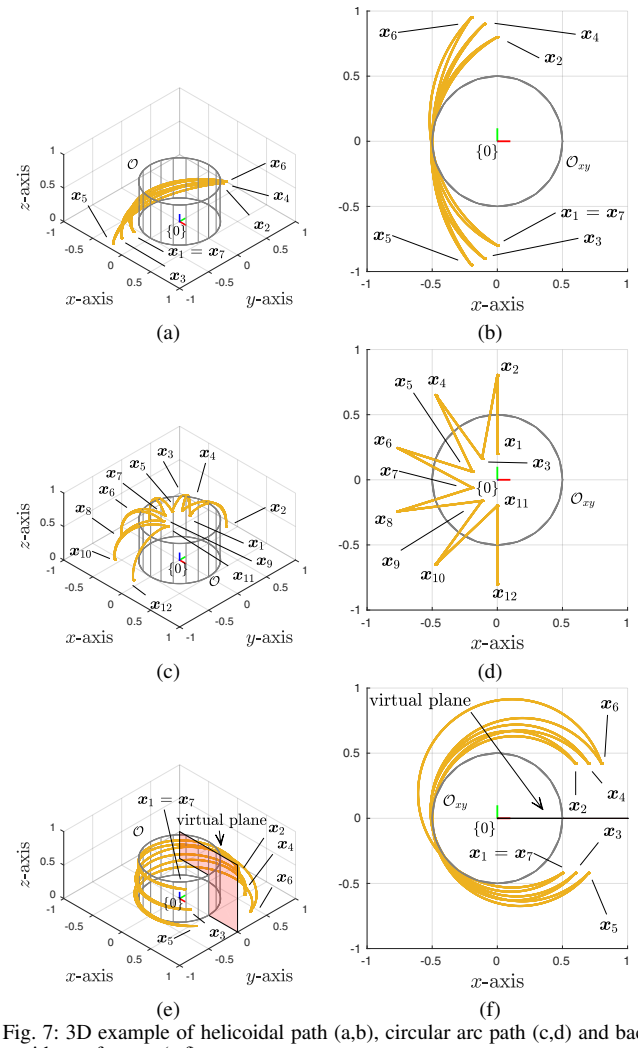

Fig. 7: 3D example of helicoidal path (a,b), circular arc path (c,d) and back avoidance feature (e,f).

mentioned in the problem of Appolonius can be used to determine the midpoint $\boldsymbol{m}$. The construction of the circular arc path can then be completed as discussed in the previous section.

3) *Helicoidal path*: An intersection is detected and neither $\boldsymbol{x}_{\text{start},xy}$ nor $\boldsymbol{x}_{\text{goal},xy}$ are located inside $\mathcal{O}_{xy}$. Therefore, a helicoidal path is chosen. The general idea is to take advantage of the projections $\mathcal{O}_{xy}$, $\boldsymbol{x}_{\text{start},xy}$ and $\boldsymbol{x}_{\text{goal},xy}$ making use of the *CPP*-method, where $\mathcal{O}_{xy}$ is the required circle.

As described in Sec. III-C the motion law in Def. 5 is applied to the respective geometric path after its selection. Then, the translational component is synchronized with the rotational one, which is interpolated according to Sec. III-B. Synchronization is achieved by applying (18) and (19).

E. Back Avoidance

Typically, robot manipulators can be limited by mechanical stops to follow a path traversing behind the robot in Cartesian space. Especially manipulators with six joints suffer from this limitation mostly sternly from the 1st joint. By implementing a virtual plane that coincides with the origin of the robot base frame and the center of the unreachable angle interval of the 1st joint the intersection detection discussed in Sec. III-D can be easily extended. In the case of an intersection of the path between $\boldsymbol{x}_{\text{start}}$ and $\boldsymbol{x}_{\text{goal}}$ and the virtual plane behind the robot the helicoidal trajectory is chosen by the planner, see Fig. 7f and Fig. 7e. This is a simple yet efficient and useful extension of the mechanical joint limit avoidance to the 1st joint.

IV. Experiments

For the following experiment we use a 7-DOF DLR Lightweight Robot III [22]. First, we compare our trajectory planner as defined in Sec. III-D with two simple p2p-trajectory generators, one in Cartesian space (Cartesian interpolator) and one in joint space (joint interpolator). Both generate a straight line in their respective domain. The accompanying video provides additional visual aid.

a) Setup: In the experiment, a trajectory is planned between a start and a goal position specified by $\boldsymbol{x}_{\text{start}} = (-0.12, 0.68, 0.45)^T\,\text{m}$ and $\boldsymbol{x}_{\text{goal}} = (-0.12, -0.68, 0.45)^T\,\text{m}$ w.r.t. the robots base frame. Since it intersects with the first robot link it violates the virtual obstacle $\mathcal{O}$. The proposed trajectory generator is initialized with a description of the virtual cylinder $\mathcal{O}$ (i.e. $r_{\mathcal{O}} = 0.36\,\text{m}$ and $h_{\mathcal{O}} = 0.5\,\text{m}$).

b) Result: As can be seen in Fig. 9 the Cartesian interpolator ($\boldsymbol{x}_{\text{Cart}}$, $\boldsymbol{q}_{\text{Cart}}$) produces an undesirable motion, leading to high joint velocities, unnecessary joint movement and violation of the joint limits. Moreover, its executed path significantly differs from the straight line path, following the silhouette of the virtual obstacle $\mathcal{O}$. The joint interpolator ($\boldsymbol{x}_{\text{joint}}$, $\boldsymbol{q}_{\text{joint}}$) generates the shortest path in joint space. However, in comparison, the executed path in Cartesian space covers the longest distance. In contrast, the proposed trajectory planner ($\boldsymbol{x}_{\text{C4}}$, $\boldsymbol{q}_{\text{C4}}$) executes the helicoidal trajectory leading to a suitable path which unifies the advantages of the other trajectory generators. First, the generated path is short in terms of covered distance, see Fig. 9 and Fig. 8. Moreover, it avoids unreachable intermediate points and, therefore avoids reaching the mechanical stops potentially creating a dangerous situation, see Fig. 8.

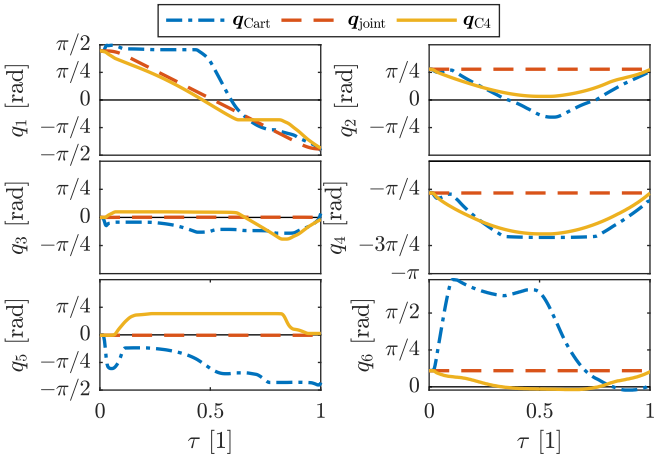

Fig. 8: Time normalized joint angle for joints generated by the joint interpolator ($\boldsymbol{q}_{\text{joint}}$), the p2p Cartesian interpolator ($\boldsymbol{q}_{\text{Cart}}$) and our interpolator ($\boldsymbol{q}_{\text{C4}}$). Note that the robots joints are labeled in increasing order as opposed to the example in Fig. 2. Noted that q_7 is omitted.

V. Conclusion

In this paper we introduced a novel trajectory generator that unifies the generation of C^4 continuous point-to-point trajectories in $SE(3)$ and the ability of avoiding unreachable

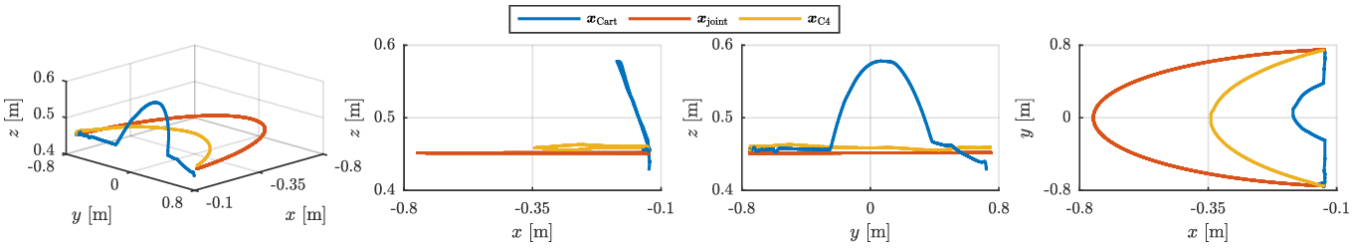

Fig. 9: Comparison of the trajectories generated by the standard joint interpolator ($\boldsymbol{x}_{\text{joint}}$), the p2p Cartesian interpolator ($\boldsymbol{x}_{\text{Cart}}$) and the proposed framework ($\boldsymbol{x}_{\text{C4}}$).

intermediate points for a typical and relevant class of manipulators. In order to define these unreachable intermediate points we introduced a general virtual obstacle for this robot class, encompassing mechanical joint limits as well as the proximal kinematic structure. The trajectory planner distinguishes between three general cases with regard to the virtual obstacle based on the Problem of Apollonius, and chooses a strategy accordingly. Furthermore, we developed a new orientation interpolation algorithm that is a generalization of the well-known Slerp algorithm. To validate our approach we tested it on a real robot. The next steps we intend to take are the generalization of our trajectory planner to via points and online replanning to be able to react to unforeseen events.

Acknowledgment

We greatly acknowledge the funding of this work by the KBee AG.

References

[1] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson Education International Press, 2005, vol. 3.
[2] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic, 1991.
[3] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE transactions on computers*, no. 2, pp. 108–120, 1983.
[4] Y. K. Hwang and N. Ahuja, "Gross motion planning - a survey," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 219–291, 1992.
[5] T. Bellmann, M. Otter, and G. Hirzinger, "The DLR robot motion simulator part ii: Optimization based path-planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4702–4709.
[6] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.
[7] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Springer, 2008.
[8] T. Kröger, *On-Line Trajectory Generation in Robotic Systems*, ser. Springer Tracts in Advanced Robotics, S. Bruno, O. Khatib, and F. Groen, Eds. Springer, 2010, vol. 58.
[9] M.-X. Kong, C. Ji, Z.-S. Chen, and R.-f. Li, "Application of orientation interpolation of robot using unit quaternion," 2013.
[10] D. Beckmann, M. Schappler, M. Dagen, and T. Ortmaier, "New approach using flatness-based control in high speed positioning: Experimental results," in *Industrial Technology (ICIT), 2015 IEEE International Conference on*, 2015, pp. 351–356.
[11] K. Ahn, W. K. Chung, and Y. Youm, "Arbitrary states polynomial-like trajectory (aspot) generation," 2004.
[12] S.-H. Nam and M.-Y. Yang, "A study on a generalized parametric interpolator with real-time jerk-limited acceleration," *Computer-Aided Design*, vol. 36, no. 1, pp. 27–36, 2004.
[13] M. Žefran, V. Kumar, and C. B. Croke, "On the generation of smooth three-dimensional rigid body motions," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 579–589, 1998.
[14] J. Stuelpnagel, "On the parametrization of the three-dimensional rotation group," *SIAM review*, vol. 6, no. 4, pp. 422–430, 1964.
[15] E. B. Dam, M. Koch, and M. Lillholm, *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998, vol. 2.
[16] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of graphics tools*, vol. 3, no. 3, pp. 29–48, 1998.
[17] K. Shoemake, "Animating rotation with quaternion curves," in *ACM SIGGRAPH computer graphics*, vol. 19, no. 3. ACM, 1985, pp. 245–254.
[18] A. De Luca and W. Book, "Robots with flexible elements," in *Springer Handbook of Robotics*. Springer, 2008, pp. 287–319.
[19] I. L. Kantor and A. S. Solodownikow, *Hyperkomplexe Zahlen (German)*. Teubner, 1978.
[20] W. R. Hamilton, "On a new species of imaginary quantities connected with a theory of quaternions," in *Proceedings of the Royal Irish Academy*, vol. 2, no. 1844, 1843, pp. 424–434.
[21] H. S. M. Coxeter, "The problem of apollonius," *The American Mathematical Monthly*, vol. 75, no. 1, pp. 5–15, 1968.
[22] A. Albu-Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger, "The DLR lightweight robot: design and control concepts for robots in human environments," *Industrial Robot: an international journal*, vol. 34, no. 5, pp. 376–385, 2007.

Appendix: Additional Features

The trajectory planner presented in Sec. III can be used as a basis for various useful features. In the following, two of them are introduced.

a) Path scheduling: It is a feature that enables the robot to shift and/or clinch the orientation component of a trajectory w.r.t. the translation component, and vice versa. This feature is especially useful in combination with teaching-by-demonstration. The parameters α_{start} and $\alpha_{\text{goal}} > \alpha_{\text{start}}$ define start and end of the orientation interpolation, respectively. Hence, T_{trans} and T_{rot} are set by

$$T_{\text{trans}} \leftarrow \max\{T_{\text{trans}}, T_{\text{rot}}\} \quad \text{and} \tag{35}$$

$$T_{\text{rot}} \leftarrow (\alpha_{\text{goal}} - \alpha_{\text{start}})\, T_{\text{trans}}, \tag{36}$$

denoting the total duration of translation and orientation, respectively. Figure 10 depicts more details.

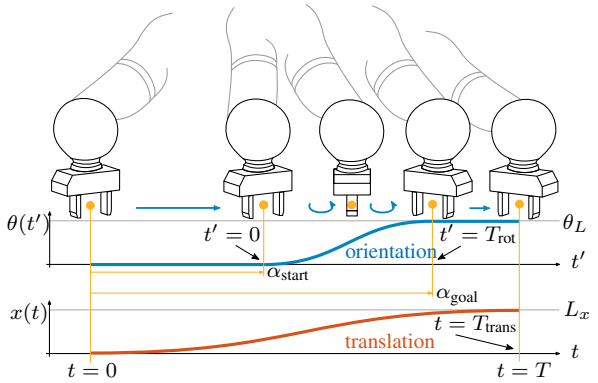

Fig. 10: Path scheduling example with $\alpha_{\text{start}} = 0.4$ and $\alpha_{\text{goal}} = 0.8$

b) Obstacles: Note that we do not consider any external obstacles yet. However, this issue could be approached in future work by shifting the origin of the xy-plane towards the geometric center of an external obstacle. The same computational procedure as introduced in this work would be applicable and result in a path that circumvents the addressed obstacle. Eventually a decision process would be required if multiple obstacles including the robot itself are present to decide on a valid path through the workspace.