Comparing Online Robot Joint Space Trajectory Optimization for Task Space Applications

Maximilian Krämer*, Rodrigo J. Velasco-Guillen†, Philipp Beckerle† and Torsten Bertram*
*Institute of Control Theory and Systems Engineering
TU Dortmund University, Dortmund, Germany
Email: maximilian.kraemer@tu-dortmund.de
†Chair of Autonomous Systems and Mechatronics
Friedrich-Alexander University Erlangen-Nürnberg, Erlangen, Germany
Email: philipp.beckerle@fau.de

Abstract—Online trajectory planning allows robotic manipulators to adapt to changing environments and dynamic tasks, which becomes increasingly relevant through human-robot interaction. Optimizing motion objectives is a common way of planning but is usually conducted in joint space, whereas many planning objectives are rather defined in task space. Joint Space Planning (JSP) first transforms target poses into joint space and then performs conventional optimization. Implicit Task Space Planning (ITSP) uses a special objective function to directly consider task quantities and is similar to economic model predictive control. Remarkably, comparative studies of both approaches with respect to their performance are scarce. This contribution aims at filling this gap by providing a systematic analysis of both methods with reference to a 6-DoF collaborative robot in practical experiments. Results show that the first method is less sensitive to local minima and profits from active reconfiguration abilities while it generates less straight-lined motions and is not able to account for target manifolds in the task space. The second method can easily consider target manifolds and performs more straight-lined motions while it is more sensitive to local minima and provides active reconfiguration only partially. Although the cost function of the second method is significantly more complex, the computational effort relativizes during planning.

Index Terms—Task Space Motion Planning, Online Trajectory Optimization, Human-Robot-Collaboration.

I. INTRODUCTION

Online trajectory planning for robotic manipulators is a challenging task, and with the rapid growth of collaborative robots, it is becoming increasingly important. The most important advantages of online trajectory planning include proactive collision avoidance with dynamic obstacles such as coworkers in the workspace. But also the frequently changing workspaces in which collaborative robots are deployed require feedback of the environment, as utilized in online trajectory planning. In addition, collaborative robots are also exposed to dynamic tasks, such as human-robot handover, where the handover location is implicitly determined by the momentary motion of the human and must be continuously updated during trajectory planning [1].

In recent years, several approaches to online trajectory planning have emerged [2]–[4]. One group is formed by optimization-based methods [5]. These methods offer to optimize cost functions as a more general way to encode different types of motion objectives. For example, optimization can support vibration damping of elastic robots [6]. Repeatedly optimizing a complete trajectory online is often too time-consuming so these approaches utilize different methods to simplify the problem. One solution is dividing the trajectory into small parts for which the optimization returns faster, which allows the robot to move before the entire trajectory was computed [7]. Planners based on Model Predictive Control (MPC) act in a similar way by restricting themselves to a horizon, that covers the current state and reaches into the future [8]–[10]. Since only a small part of the optimized horizon is performed by the robot between two optimization instances, most of it remains valid in the next instance. This allows distributing the optimization across different time instances without having to wait for the solver to fully converge in each instance. This type of optimization-based planner shows promising results and constitutes an active field of research [8], [9].

The optimization variables of MPC-based planners mostly originate from joint space, for example, in the form of joint angles, velocities, and accelerations [7]–[12]. However, constraints such as collision avoidance or pose targets are usually defined in the task space and therefore require a transformation between both spaces. Especially for handling end-effector pose targets, there are two ways, which have essentially different properties and considerable impact on the resulting trajectories. Joint Space Planning (JSP) continuously transforms task space set-points into joint space by inverse kinematics and considers them in the optimization afterward [9]. Potential redundancies between both spaces are actively handled during the transformation. Implicit Task Space Planning (ITSP) connects both spaces within the optimization using an economic cost function that utilizes forward kinematics [10], [11]. It differs from JSP in that the target configuration is only implicitly given and the approach optimizes towards a task space quantity. While both methods are established and theoretically analyzed [13], [14], experimental studies evaluating and comparing their online trajectory planning performance are scarce. Previous work mainly focuses on one method and often does not consider all critical cases, which are important to evaluate the practical effectiveness of trajectory planning in human-robot interaction [9]–[11]. This paper derives and
summarizes the theory of both methods and contributes a comparative analysis in five practically related experiments using a collaborative robot. Additionally, the study provides guidance on which parameterizations for particular situations can be analyzed in more depth in future work. Both methods base on a recent approach by [8] to benefit from its computational performance and improve comparability.

Section II introduces the basic optimization problem of trajectory planning. Sections III and IV elaborate on the details of JSP and ITSP, respectively, whereas Section III introduces set-point transformation. Section V presents the experiment setup and the evaluation of five scenarios. Section VI finishes this paper with a conclusion and an outlook.

II. ONLINE TRAJECTORY OPTIMIZATION

MPC-based trajectory planning optimizes partial trajectories at closed-loop time instances \( t_n \) for \( n = 0, 1, \ldots \) and \( t_{n+1} - t_n = \Delta t \). A partial trajectory \( \tau_{0,K} \) consists of \( K + 1 \) states \( \tilde{x}_k \in \mathcal{X} \) and \( K \in \mathbb{N} \) controls \( \tilde{u}_k \in \mathcal{U} \):

\[
\tau_{0,K} := \left( \begin{array}{c}
\tilde{x}_0 := \tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_K \\
\tilde{u}_{0,K-1} := \tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_K
\end{array} \right) ,
\]

based on the following uniform time grid:

\[
\tilde{t}_0 < \tilde{t}_1 < \ldots < \tilde{t}_K < \ldots < \tilde{t}_K = T, \quad \tilde{t}_{k+1} - \tilde{t}_k = \Delta t. \tag{2}
\]

Sets \( \mathcal{X} \subset \mathbb{R}^N \) and \( \mathcal{U} \subset \mathbb{R}^M \) for \( N, M \in \mathbb{N} \) represent box-constraints for states and controls, respectively:

\[
\mathcal{X} := \left\{ \tilde{x} \in \mathbb{R}^N \mid \tilde{x}_{\min} \leq \tilde{x} \leq \tilde{x}_{\max} \right\} ,
\]

\[
\mathcal{U} := \left\{ \tilde{u} \in \mathbb{R}^M \mid \tilde{u}_{\min} \leq \tilde{u} \leq \tilde{u}_{\max} \right\} . \tag{3}
\]

In each instance \( t_n \) a portion of the optimal continuous-time control trajectory \( \tilde{u}^*(t) \) is sent to the robot:

\[
\mu(t) := \tilde{u}^*(t) \big|_{t_0 = t_n}^{t_1 = t_{n+1}} \quad \text{for} \quad t \in [t_n,t_{n+1}], \tag{4}
\]

with \( \tilde{u}(t) \) as the continuous-time control trajectory of \( \tau_{0,K-1} \):

\[
\tilde{u}(t) := \tilde{u}_k \quad \text{const. for} \quad t \in [\tilde{t}_k, \tilde{t}_{k+1}). \tag{5}
\]

By setting \( \tilde{t}_0 = t_n \) in (4), the time grid of each partial trajectory starts at the current closed-loop time instance \( t_n \).

States \( \tilde{x}_k \) and controls \( \tilde{u}_k \) are introduced as optimization parameters of a nonlinear program:

\[
\min_{\tilde{x}_{0,K-1}} c_K(\tilde{x}_{0,K}, \cdot) + \sum_{k=0}^{K-1} c_k(\tilde{x}_k, \tilde{u}_k). \tag{6a}
\]

subject to

\[
\tilde{x}_k - \tilde{x}_{k+1} + \Delta t f(\tilde{x}_k, \tilde{u}_k) = 0, \tag{6b}
\]

\[
\tilde{x}_k \in \mathcal{X}_c(\tilde{t}_k), \tag{6c}
\]

\[
\tilde{x}_0 = \tilde{x}(t_n). \tag{6d}
\]

Cost function \( c_k : \mathbb{R}^N \times \mathbb{R}^M \rightarrow \mathbb{R}^+_0 \) in (6a) constitutes the running costs and takes into account the target state, and other objectives, which are defined in Section III and IV for JSP and ITSP, respectively. For the sake of brevity, the final state cost \( c_K(\tilde{x}_K, \cdot) \) equals \( c_K(\tilde{x}, \tilde{u}) \) but without evaluating \( \tilde{u} \).

Equality constraint (6b) enforces consistency between \( \tilde{x} \) and \( \tilde{u} \) regarding the system behavior \( f : \mathbb{R}^N \times \mathbb{R}^M \rightarrow \mathbb{R}^N \) of the robot. The set \( \mathcal{X}_c(\tilde{t}_k) \subset \mathcal{X} \) includes all collision-free states and ensures a collision-free trajectory via equality constraint (6c). Since the environment is allowed to be dynamic, \( \mathcal{X}_c \) depends on time \( \tilde{t}_k \), which allows incorporating predicted states of the environment. From a practical point of view, this constraint is usually implemented by distance functions and thresholds between robot links and the environment [8].

The last constraint (6d) enforces the partial trajectory to start at the robot’s current state \( x(t_n) \).

III. JOINT SPACE PLANNING

This section introduces the transformation of task space set-points into joint space targets and defines the objective function of JSP.

A. Set-Point Transformation

Set-point transformation refers to the systematic analysis and online update of target states \( x_{\text{ref}} \in \mathcal{X} \) prior to and during online planning to exploit the kinematic flexibility ([15]) between task space and joint space as well as to enable direct joint space planning of task space set-points. Consider a task space set-point \( s_{\text{ref}} \):

\[
s_{\text{ref}} := (p_f, \Theta_f)^T, \tag{7}
\]

with \( p_f \in \mathbb{R}^3 \) and \( \Theta_f \in \mathbb{H} \) as the reference position and reference unit quaternion, respectively. Set-point transformation then utilizes inverse kinematics to obtain a set \( q_{\text{ref}} \in \mathbb{R}^D \) of potential target configurations \( q_{\text{ref},i} \) for \( i = 1, 2, \ldots, |Q_{\text{ref}}| \) and a robot with \( D \in \mathbb{N} \) DoFs:

\[
Q_{\text{ref}} := \left\{ q \in \mathbb{R}^D \mid h(q) = s_{\text{ref}} \right\} , \tag{8}
\]

with forward kinematic function \( h(q) \). Without loss of generality, it is assumed that the first components of the state vector \( x \) constitute joint angles \( q \). Thus, a potential target state \( x_{\text{ref},i} \) is defined as follows:

\[
x_{\text{ref},i} := (q_{\text{ref},i} - 0 \ldots 0)^T. \tag{9}
\]

The set \( Q_{\text{ref}} \) is reduced to \( Q'_{\text{ref}} \) containing only those configurations \( q_{\text{ref},i} \) for which the corresponding target state \( x_{\text{ref},i} \) satisfies \( x_{\text{ref},i} \in \mathcal{X}_c(t_n) \). The final target configuration \( q_{\text{ref}}^* \in Q'_{\text{ref}} \) minimizes the objective function \( V : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^+_0 \):

\[
q_{\text{ref}}^* = \arg \min_{q \in Q_{\text{ref}}} V(q, q(t_n)), \tag{10}
\]

which might further consider the robot’s current joint configuration \( q(t_n) \). The final target state \( x_{\text{ref}}^* \) follows from (9) and \( q_{\text{ref}}^* \). The particular choice of \( V \) is dealt with in Section V.
B. Objective Function
The previously transformed target state $x_{\text{ref}}$ is now directly available in joint space and allows the use of a quadratic objective:

$$c_Q(\hat{x}) := (\hat{x} - x_{\text{ref}})^T Q (\hat{x} - x_{\text{ref}}),$$  \hspace{1cm} (11)$$
with positive definite weights $Q \in \mathbb{R}^{N \times N}$. This cost term ensures that solving (6) optimizes the partial trajectory $\tau_{0:K}$ towards the target state $x_{\text{ref}}$. It is often advantageous to evaluate controls:

$$c_R(\tilde{u}) := \tilde{u}^T R \tilde{u},$$  \hspace{1cm} (12)$$
with positive definite weights $R \in \mathbb{R}^{M \times M}$. Hereby, unnecessary motions of the robot will be avoided and the convergence rate when solving (6) increases [16]. Although (6c) already represents hard constraints for collision avoidance, additional cost potentials provide a way to keep the robot away from obstacles in general and induce a more prophylactic avoidance behavior:

$$c_C(\tilde{x}) := \sum_{o \in \mathcal{O}} \rho(\rho),$$  \hspace{1cm} (13)$$
with $\rho \in \mathcal{O}$ as pairs of link and obstacle collisions. The proximity function $\rho: \mathcal{O} \rightarrow \mathbb{R}_0^+$ rises, the closer a robot link approaches an obstacle:

$$\rho := \begin{cases} w \left( \frac{d(o)}{d_{\text{min}}} - 1 \right)^2 & \text{if } d(o) < d_{\text{min}}, \\ 0 & \text{if } d(o) \geq d_{\text{min}}, \end{cases}$$  \hspace{1cm} (14)$$
with weight $w \in \mathbb{R}_0^+$ and activation threshold $d_{\text{min}} \in \mathbb{R}^+$. The function $d: \mathcal{O} \rightarrow \mathbb{R}$ calculates the signed distance between a collision pair $o$.

Finally, joint space planning uses the following running costs:

$$c_k^{(\text{JSP})}(\tilde{x}, \tilde{u}) := c_Q(\tilde{x}) + c_C(\tilde{x}) + c_R(\tilde{u}).$$  \hspace{1cm} (15)$$

C. Initialization
Initialization plays an important role for gradient-based methods to solve (6) since it has a significant influence on the solution if multiple minima exist. In this case, initialization uses a straight line between $\tilde{x}_0$ and $x_{\text{ref}}$ for instance $t_0$. In subsequent instances $t_1, t_2, \ldots$, the method can profit from solutions of the previous instances $t_0, t_1, \ldots$ (warm start). When warm start is inactive the straight line initialization is used for each instance.

IV. IMPLICIT TASK SPACE PLANNING
This section introduces the concept of ITSP based on an economic cost function. The target state $x_{\text{ref}}$ is not explicitly known but implicitly emerges during optimization towards the task space set-point $s_{\text{ref}}$.

A. Objective Function
The economic cost function defines the distance between the current end-effector pose $\hat{s}_k$ and the task space set-point $s_{\text{ref}}$ in task space. The current end-effector pose $\hat{s}_k$ for states $\hat{x}$ of the partial trajectory $\tau_{0:K}$ is determined with the help of direct kinematics $h(\hat{x})$:

$$\hat{s} = h(\hat{x}) := (\hat{p} \quad \hat{\Theta})^T.$$  \hspace{1cm} (16)$$
The position related cost function is then simply:

$$c_p(\tilde{x}) := (p_t - \tilde{p})^T P (p_t - \tilde{p}),$$  \hspace{1cm} (17)$$
with positive semi-definite weights $P \in \mathbb{R}^{3 \times 3}$. The error component for orientation is first defined as unit quaternion $\Delta \Theta$:

$$\Delta \Theta := (\Delta \tilde{\eta} \quad \Delta \tilde{e}) = \Theta_0 \times \Theta^{-1},$$  \hspace{1cm} (18)$$
with $\times$ describing the quaternion multiplication and $\Delta \tilde{\eta} \in \mathbb{R}$ as well as $\Delta \tilde{e} \in \mathbb{R}^3$ the quaternion scalar and vector component, respectively. By decomposing $\Delta \Theta$ into axis-angle notation, the magnitude $\phi \in [-\pi, \pi]$ rad of the orientation error and the corresponding axis $\tilde{r} \in \mathbb{R}^3$ are obtained:

$$\tilde{r} = \frac{\Delta \tilde{e}}{1 - \Delta \tilde{\eta}^2}.$$  \hspace{1cm} (19)$$
Both components are considered in the orientation related cost function:

$$c_O(\tilde{x}) := \left( \begin{array}{c} \tilde{r} \\ \tilde{\eta} \end{array} \right)^T O \left( \begin{array}{c} \tilde{r} \\ \tilde{\eta} \end{array} \right),$$  \hspace{1cm} (20)$$
with positive semi-definite weights $O \in \mathbb{R}^{3 \times 3}$.

In addition to (17) and (20), $c_R(\tilde{x})$ and $c_C(\tilde{x})$ introduced in Section III-B are included so that the final cost function is as follows:

$$c_k^{(\text{ITSP})} = c_k^{(\text{JSP})} + c_O(\tilde{x}) + c_C(\tilde{x}) + c_R(\tilde{u}).$$  \hspace{1cm} (21)$$

Remark: Since $\tilde{r}$ is a unitary vector, $\tilde{\phi}$ already would be a sufficient measure of the orientation error. However, by utilizing the orientation error vector in (20), it is possible to weight the angular components individually. It is also possible to use the vector component $\Delta \tilde{e}$ as the orientation error [17]. Substituting the mapping from axis-angle $\Delta \tilde{e} = \sin \left( \frac{\phi}{2} \right) \tilde{r}$ reveals the similarity between both representations:

$$c_k^{(\text{ITSP})} = \left( \sin \left( \frac{\phi}{2} \right) \tilde{r} \right)^T O \left( \sin \left( \frac{\phi}{2} \right) \tilde{r} \right).$$  \hspace{1cm} (22)$$
When comparing (22) and (20), the former leads to a squared sinusoidal component while the latter leads to a parabolic component.
B. Initialization

Unlike JSP from Section III, ITSP does not have an explicit target state in joint space and thus, a direct initialization is not possible. However, there are still options to initialize:

1. Initial state: via the robot’s current state \( x(t_n) \),
2. Initial configuration: with the robot’s default configuration (\( q = 0 \)),
3. Initial position: via \( x_{\text{ref}} \) based on the transformation from Section III-A.

All variants will be considered in the evaluation. Similar to JSP, ITSP also benefits from warm starting with a previous solution or will reinitialize in every instance if warm starts are inactive.

V. Evaluation

This section introduces the robot and the implications for approaches JSP and ITSP as a basis to present and analyze five different evaluation scenarios in the remainder. While results apply to the real robot, some figures show the robot in a simulation environment for improved illustration and control of the environmental conditions. The experiments investigate the behavior at local minima, the exploitation of kinematic flexibilities, target manifolds, and computational cost.

A. Experiment Setup

The scenarios employ a UR10 collaborative robot, which has \( D = 6 \) DoFs. The predicted state vectors \( \tilde{x} \) as well as actual states \( x \) represent joint angles (\( N = 6 \)) and the predicted controls \( \tilde{u} \) are defined as joint velocities\(^1\) (\( M = 6 \)). Consequently, the target state \( x_{\text{ref}} \) equals the target configuration \( q_{\text{ref}} \). By considering these quantities, the most common motion constraints and set-points can be taken into account. Acceleration limits are realized via the additional constraint \( \tilde{u}_k \in U_\Delta \), with:

\[
U_\Delta := \left\{ \tilde{u}_k \in \mathcal{U} \mid a_{\min} \leq \frac{\tilde{u}_k - \tilde{u}_{k-1}}{\Delta t} \leq a_{\max} \right\}.
\]

The robot has sufficient gear reduction which, together with the manufacturer’s joint velocity controller, allows the system behavior \( f \) between target velocities and actual joint angles to be described by six parallel integrators [8], [17]:

\[
f(\tilde{x}_k, \tilde{u}_k) := \tilde{u}_k.
\]

Collision avoidance uses swept-sphere-volumes, like spheres or capsules to model the robot’s links and obstacles. They allow an efficient trade-off between the effort of distance calculations and their fit, which is why they are often used in online trajectory planning [18], [19]. Set-Point transformation uses a generated algorithm to calculate the analytical inverse kinematics, which is advantageous over numerical methods since kinematic flexibility is preserved in the form of multiple solutions. The following objective function is used in (10):

\[
V(q, q(t_n)) := ||q - q(t_n)||_2.
\]

It prefers solutions, which are closest to the robot’s current configuration.

\(^1\)Using \( \tilde{u}_k, \tilde{u}_{k-1} \), and \( \Delta t \) to compute accelerations \( \tilde{a}_k \) similar to (23) allows for commanding a robot via accelerations if needed, without changing state and control vector.

Table I summarizes the parameter values of the following scenarios. Unless otherwise specified, they are the same for all scenarios. The state related components of the final state cost \( c_k(\tilde{x}_k, \cdot) \) use their own weight matrices \( Q_k, P_k, \) and \( O_k \) instead of those from the running costs \( c_k(\tilde{x}_k, \tilde{u}_k) \). Warm start is active for both methods by default. In principle, when combining costs the relative ratios of the weights are decisive instead of their absolute values. The weights \( Q, P \) and \( O \) are usually larger than \( R \), as otherwise the motion will slow down increasingly early before reaching the target due to decreasing target costs and constant motion costs. Since orientation is influenced by all joints and positioning mainly by the first three, positioning also leads to a change in the orientation, which means that the latter tends to trail behind. Choosing \( O > P \) compensates this effect slightly. The final state weights \( Q_f, P_f, \) and \( O_f \) are chosen to be much larger than the other weights, since they are evaluated only on the final state and serve a special purpose. Since the final state is usually closest to the target and potentially reaches behind obstacles, it can relativize the costs of evasive motions caused by the middle part of the trajectory.

Both motion planning methods are implemented in C++ under ROS (Robot Operating System) and run on Ubuntu 18.04 using an Intel i7-8700K CPU at 4.8 GHz and 32 GB RAM. Trajectory optimization benefits from a previously published hypergraph representation, which efficiently encodes the nonlinear optimization problem (6) and exploits sparsity [20]. To solve the optimization problem, both methods adopt the interior point solver IPOpt [21] with a tolerance of 0.5 and up to 20 iterations using the linear solver MA27 [22]. Refer to [23] for necessary and sufficient optimality conditions.

B. Joint Limits

The objective function (21) of ITSP optimizes directly towards the task space set-point \( s_{\text{ref}} \) without taking joint limits into account. As a result, optimization may stop in a local minimum in front of a joint limit enforced by box constraints (3). This scenario investigates to what extent the initialization

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_{\text{max}} )</td>
<td>175°</td>
<td>Identical for all joints</td>
</tr>
<tr>
<td>( x_{\text{min}} )</td>
<td>-175°</td>
<td>Identical for all joints</td>
</tr>
<tr>
<td>( u_{\text{max}}, \tilde{u}_{\text{min}} )</td>
<td>( \pm 0.5 ) rad s(^{-1} )</td>
<td>Identical for all joints</td>
</tr>
<tr>
<td>( a_{\text{max}}, \tilde{a}_{\text{min}} )</td>
<td>( \pm 1.0 ) rad s(^{-2} )</td>
<td>Identical for all joints</td>
</tr>
<tr>
<td>( K )</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>( \Delta t )</td>
<td>0.1 s</td>
<td></td>
</tr>
<tr>
<td>( w )</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>( \alpha_{\text{min}} )</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>( Q )</td>
<td>diag(2, 2, 2, 2, 2, 2)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( Q_f )</td>
<td>100 diag(2, 2, 2)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( R )</td>
<td>diag(1, 1, 1, 1, 1, 1)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( P )</td>
<td>diag(3, 3, 3)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( P_f )</td>
<td>100 diag(2, 2)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( O )</td>
<td>diag(5, 5, 5)</td>
<td>Diagonal matrix</td>
</tr>
<tr>
<td>( O_f )</td>
<td>100 diag(2, 2)</td>
<td>Diagonal matrix</td>
</tr>
</tbody>
</table>
of ITSP has an impact on this effect and how this method generally performs compared to JSP. The three start configurations for experiments 1, 2, and 3 are \( \mathbf{q} = (-90, 0, 0, 0, 0, 0) \)° with different values of the first joint (−90°, −120° and −150°). The final configurations \( \mathbf{q}_{\text{ref}} \) simply flip the sign of the first joint by turning to 90°, 120° and 150°, respectively. For ITSP, \( \mathbf{q}_{\text{ref}} \) is translated into a task space set-point \( \mathbf{s}_{\text{ref}} \) corresponding to the respective final configuration.

Figure 1 shows the path of the end-effector in the \( x-y \)-plane for both methods in experiment 3. The determination of the motion is a fundamental difference implied by the cost functions. ITSP takes a more directed path around the robot’s base (black square) towards the set-point, which can be beneficial for human-robot scenarios where plausible robot motions are easier to understand for humans and lead to more safety and acceptance. However, ITSP cannot reach the target pose in every case. Table II gives an overview of the conditions under which planning succeeds/fails. The first form of initialization \((I_1)\) is successful only in experiment 1. In experiment 2 and 3 it simply does not provide any helpful information that could put the optimization on the right track, even for large horizons like \( K = 100 \). The problem is the local minimum at the joint limit, from which to get out requires a temporary increase of the costs, which gradient-based methods usually do not support. Although \( I_2 \) and \( I_3 \) provide helpful initializations, they require long horizons for experiment 3, because otherwise, the optimization has to oppose the initialization due to the constraints. However, long horizons slow down the runtime considerably. As expected, JSP has no problems reaching the target configuration. Since only the first initialization matters, warm start has no effect except to reduce the runtime of both methods, which is discussed in detail in Section V-F.

The experiments also show another difference related to the convergence of planning. Figure 2 compares the translational and rotational error of both methods in experiment 1. The timestamps indicate when the thresholds for position or orientation are reached. The errors are defined as follows:

\[
\Delta p = \| \mathbf{p}_f - \mathbf{p} \|_2, \quad \Delta o = \left\| \frac{\phi \mathbf{r}}{\pi} \right\|_2, \tag{26}
\]

with the threshold values \( T_p = 0.5 \) cm and \( T_o = 1 \)°. As can be seen, JSP already converges completely after 8.6 s, while ITSP needs 14.7 s. The slow convergence of the position part is amplified due to the target pose, which represents a fully extended arm. In this case, some joint motions have only a marginal effect on the singular degree of freedom.

### C. Kinematic Flexibility

Even robots with \( D \leq 6 \) can have multiple configurations for a task space set-point. This scenario examines how both methods react when a cylindrical obstacle \((l = 0.5 \text{ m}, r = 0.1 \text{ m})\) intrudes into the robot’s current configuration \( \mathbf{q} = (0.45 \ 180 \ -45 \ 0 \ 0) \)° to different extends, forcing both methods to reconfigure. The obstacle center is located at \((0.4 \ 0.5 \ 0.85) \text{ m})\). Figure 3 illustrates the setup of this scenario and the different angles at which the obstacle rotates towards the robot. The obstacle performs the rotation within 10 s.

Table III summarizes the results. Reconfiguration with ITSP and active warm start relies on the obstacle actively pushing

---

**TABLE II**

**RESULTS FOR SCENARIO 1**

<table>
<thead>
<tr>
<th>Exp</th>
<th>JSP</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>√( ( K \geq 30 ) )</td>
<td>√( ( K \geq 30 ) )</td>
<td>√( ( K \geq 30 ) )</td>
<td>√( ( K \geq 30 ) )</td>
</tr>
<tr>
<td>2</td>
<td>√( ( K \geq 30 ) )</td>
<td>×( ( K \leq 100 ) )</td>
<td>√( ( K \geq 36 ) )</td>
<td>√( ( K \geq 36 ) )</td>
</tr>
<tr>
<td>3</td>
<td>√( ( K \geq 30 ) )</td>
<td>×( ( K \leq 100 ) )</td>
<td>√( ( K \geq 96 ) )</td>
<td>√( ( K \geq 96 ) )</td>
</tr>
</tbody>
</table>

’re: success, ×: fail.
the robot and therefore does not work reliably, for example $-60^\circ$ is not sufficient while $-75^\circ$ is. Since reconfiguration happens during planning, any initialization for ITSP is only effective if warm start is deactivated. Without warm start, ITSP can reliably reconfigure with the initialization $I_3$, as set-point transformation from Section III-A actively reacts to the dynamic situation. The active management of redundancies via set-point transformation allows JSP to always reliably reconfigure as soon as the current configuration becomes invalid. This can also be seen in Figure 4, where the target values of joints 2, 3, and 4 change to a new configuration after 0.3 s.

### D. Target Manifolds

The weighting with $P$, $P_f$ as well as $O$ and $O_f$ allows ITSP to evaluate individual components of a task space set-point to different degrees or create target manifolds in the task space in a simple and intuitive way. This scenario investigates how well redundancies in the set-point can be exploited in case of a temporary disturbance by a dynamic obstacle. Figure 5 outlines the setup in this scenario, where the robot holds a target pose and the center of a cylindrical obstacle ($\tilde{l} = 0.25 \text{ m}, r = 0.1 \text{ m}$) travels from $(0.84, 0.26, 0.3) \text{ m}$ straight up for 1 m through the center of the end-effector within 10 s. ITSP uses new weights $P = \text{diag}(0.02)$ and $P_f = \text{diag}(0.02, 0.02)$ to create a manifold in $x$-$y$ plane at the $z$-level of the set-point. The robot starts at the same configuration as in the previous scenario and is commanded to hold the associated task space set-point.

Figure 6 compares the $z$-component of the position error between JSP and ITSP. It shows how ITSP can easily exploit target manifolds in the task space through the semidefinite weight matrices and avoid the obstacle mainly within the $x$-$y$ plane. Set-point transformation cannot map manifolds from the task space to the joint space since the method cannot represent infinite solutions symbolically online. Supporting this is already quite complex in general, but integrating them into the cost function of JSP is another challenge. In this scenario, JSP avoids the obstacle until the target configuration is free again, and therefore returns after 10.1 s, while ITSP has reached its alternative pose after 4.9 s.

### E. Obstacle Avoidance

Evasive motions often cause a temporary increase in costs, which can lead to local minima and, in the case of gradient-based methods, to a standstill of the robot. Something similar has already been dealt with by the first scenario. This scenario investigates to what extent JSP and ITSP are affected when circumnavigating obstacles and what impact initializations and different obstacle sizes have. Figure 7 shows the setup of this scenario consisting of cylindrical obstacles with three different lengths ($l_1 = 0.4 \text{ m}, l_2 = 0.6 \text{ m}, l_3 = 0.8 \text{ m}, r = 0.1 \text{ m}$). There are two start configurations with the first joint angle $-30^\circ$ (I) and $-60^\circ$ (II). The target configurations use $60^\circ$ for the first joint. All remaining joints are set to $0^\circ$ for start and target configurations. The center of all obstacles are located at the position of the left obstacle $(0.84, 0.26, 0.9)$, but are shown side by side for the sake of clarity. A more detailed analysis for collision avoidance with dynamic human obstacles for JSP can be found in [8]. Again, the target configuration is converted to a corresponding task space set-point $s_{\text{ref}}$ for ITSP.

![Fig. 4. Evolution of the robot’s joint angles with switching reference due to set-point transformation at 0.3 s.](image)

![Fig. 5. Experiment setup for scenario 3.](image)

![Fig. 6. $z$-component of the deviation from the task space manifold during obstacle disturbance.](image)

<table>
<thead>
<tr>
<th>Exp.</th>
<th>JSP</th>
<th>ITSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-60^\circ$</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>$-75^\circ$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$-90^\circ$</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

✓: success, ×: fail, *: no warm start

**TABLE III**

RESULTS FOR SCENARIO 2

<table>
<thead>
<tr>
<th>Joint Angle</th>
<th>$\Delta p,z [\text{rad}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint 2</td>
<td>Blue</td>
</tr>
<tr>
<td>Joint 3</td>
<td>Red</td>
</tr>
<tr>
<td>Joint 4</td>
<td>Orange</td>
</tr>
</tbody>
</table>

Time $t \ [\text{s}]

Error $\Delta p,z \ [\text{m}]$

$t = 4.9$ $t = 10.1$

![Fig. 5. Experiment setup for scenario 3.](image)
Table IV summarizes the results of this scenario. Up to a length of 0.4 m, ITSP has no problems avoiding the obstacle. From a length of 0.6 m, the initialization $I_1$ is needed, because it reaches behind the obstacle and together with the final state weights $(P_f, O_f)$, which are considerably higher than for the running costs, the local minima is passed. This difference in weights ensures that getting the end of the horizon closer to the target reduces the cost more than the evasive motion in the middle part of the horizon increases it. For this to work, an initialization like $I_3$ is required that reaches as far as possible towards the target. However, length $l_3$, is too large for an horizon of $K = 30$. Similar to the first scenario, the initialization is shortened during optimization to meet the constraints so that the tip of the horizon does not get behind the obstacle and remains in the local minimum in front of it. This effect is amplified if the start and target configurations are further apart. Only when warm start is deactivated ITSP regains its success as it eventually reaches a configuration similar to case I from which initialization worked. JSP has the advantage that it initializes towards the target configuration in the joint space by default and thus automatically benefits from high final state weights. Therefore, it does not have any problems with up to $l_2$ regardless of warm start. However, the length $l_3$ is also too large for the same reasons as for ITSP.

### E. Runtime Analysis

The cost functions of both methods differ in (11) for JSP and (17) and (20) for ITSP. Cost functions (17) and (20) are not only more costly due to the direct kinematics but also introduce nonlinearities that potentially slow down the convergence of the solver. This experiment compares the computational complexity of (17) and (20) with that of (11) and analyzes the impact on solver iterations. Planning starts in the start configuration $q = (-70 - 60 60 - 180 180 0)°$ and plans towards the target configuration $q_{ref} = (70 60 -60 0 0 180)°$.

To determine the computation time of the cost functions, 100 000 calls with randomized states and target values are calculated and then averaged. Cost function (11) takes 77 ns per evaluation and cost functions (17) and (20) combined take 348 ns. The impact of forward kinematics is clearly visible with a factor of 4.5. However, the resulting average runtime throughout planning with warm start until the thresholds for position and orientation are reached is relatively similar with 36.9 ms for JSP and 43.5 ms for ITSP ($I_1$). This indicates a constant solver-internal delay, which covers the factor measured above. For this comparison, warm start is active as initialization has hardly any influence on the computation time in that case. Figure 8 offers another perspective by showing the distribution of the required solver iterations. ITSP finishes more often after 3 iterations than JSP. However, this results mainly from slow convergence towards the target pose, where the solutions hardly change. Of greater importance is that JSP requires no more than 12 iterations, while ITSP requires up to 20, which occur in the beginning of the motion.

Figure 9 compares the histograms of initializations $I_2$ and $I_3$.
I₂ and shows the impact of different initializations without warm start. Initialization I₃ performs worse when the home configuration is no longer between the current and the target configuration. This results in constantly high iterations, since the initialization points in the wrong direction. Initialization I₃ always acts in a target-oriented way and manages to require less iteration (down to 10) while at the same time reduces the count for 20 iterations. However, the average runtimes until the thresholds for position and orientation are reached are almost equal with 54.5 ms for I₂ and 57.3 ms for I₃.

VI. CONCLUSION AND OUTLOOK

This paper presents JSP and ITSP for online trajectory planning of task space targets, which belong to the MPC-based methods. Both methods are introduced theoretically and examined in detail in five different practical scenarios.

The results show that JSP has no problems with local minima caused by joint limits and, thanks to set-point transformation, it can actively exploit the kinematic flexibility of the robot. However, set-point transformation requires an inverse kinematic approach that, unlike conventional numerical methods, determines all possible solutions. This may not be possible for every robot. On the other hand, JSP generates less straight-lined motions in the workspace and cannot easily take into account target manifolds in the task space. The advantages of ITSP are the more straight-lined motions in the workspace, as well as the easy way to consider target manifolds in the task space. Even though the cost function of ITSP is much more complex to calculate and nonlinear, the resulting computation time relativizes during optimization.

Both methods have shortcomings with extensive evasive motions, which can lead to local minima due to the optimization-based nature of both methods. For ITSP to be on par with JSP in this matter, warm start must be deactivated and set-point transformation is required for initialization. The same is also necessary for reliable exploitation of the kinematic flexibility, which is otherwise only passive. Without warm start, initializing by home configuration (I₂) or by transformed pose (I₃) is almost equally fast, so the latter should be considered if possible. Further remarks of ITSP are the slower convergence to the target pose, which is worsened by kinematic singularities, and the sensitivity to stop in local minima in front of joint limits.

The experiments have highlighted challenges of online planning and current limitations as well as strengths of the approaches. Given the high potential of model predictive planning, future work should address the problem of local minima in particular, for example, by using additional waypoints. In order not to abandon warm start for ITSP in general, adaptive or parallel approaches are possible, which switch or use both variants. In addition to the utilized interior point solver, sequential quadratic programming solvers have demonstrated their performance in the field of MPC in the past and are also intended to be part of further analysis.

REFERENCES