An Efficient and Time-Optimal Trajectory Generation Approach for Waypoints under Kinematic Constraints and Error Bounds

Jianjie Lin, Nikhil Somani, Biao Hu, Markus Rickert, and Alois Knoll

***Abstract*— This paper presents an approach to generate the time-optimal trajectory for a robot manipulator under certain kinematic constraints such as joint position, velocity, acceleration, and jerk limits. This problem of generating a trajectory that takes the minimum time to pass through specified waypoints is formulated as a nonlinear constraint optimization problem. Unlike prior approaches that model the motion of consecutive waypoints as a Cubic Spline, we model this motion with a seven-segment acceleration profile, as this trajectory results in a shorter overall motion time while staying within the bounds of the robot manipulator's constraints. The optimization bottleneck lies in the complexity that increases exponentially with the number of waypoints. To make the optimization scale well with the number of waypoints, we propose an approach that has linear complexity. This approach first divides all waypoints to consecutive batches, each with an overlap of two waypoints. The overlapping waypoints then act as a bridge to concatenate the optimization results of two consecutive batches. The whole trajectory is effectively optimized by successively optimizing every batch. We conduct experiments on practical scenarios and trajectories generated by motion planners to evaluate the effectiveness of our proposed approach over existing state-of-the-art approaches.**

I. Introduction

In industrial applications, it is important to guarantee fast and accurate motions for robot manipulators to perform a well-defined task, such as assembly and welding. Limiting the motion jerk is an essential requirement for avoiding the manipulator's mechanical resonances and improving the trajectory accuracy. Once a task is defined, the robot is typically optimized for fast cycle times, therefore requiring a time-optimal trajectory with kinematic constraints. On the other side, the generated trajectory should also be as smooth as possible, especially for applications such as spray painting.

The most common motion profiles used during teach-in are point-to-point motions in configuration space, straight lines and circular motions in Cartesian space, and in a limited number of industrial controllers a spline interpolation in Cartesian space. Controllers typically offer blending options in configuration or Cartesian space to smoothen the trajectory and to avoid stops at each waypoint. With these different motion options, a programmer can impose constraints on the behavior of the robot and its end effector, e.g., for following an outline of a specific object. A robot with more degrees of freedom than required by this constraint is able to optimize its motion according to other goals. Given an acceptable tolerance in following a straight line in Cartesian space, an optimization algorithm can modify the trajectory even further to improve the cycle time of the process or the durability of the mechanical system.

With challenges arising from small lot sizes, where manual programming is no longer feasible, path planning algorithms can be used to move from manual programming to an automatic generation of robot motions. They consider geometric information from the robot and its environment to generate collision-free paths between specified start and goal configurations. This problem is already PSPACE-hard [1] when considering a state space solely based on positions and that complexity increases for kinodynamic algorithms that add velocity information and generate a trajectory instead of a path. The solution path is however only valid with the same interpolation used during planning, typically a linear interpolation in configuration space. Trajectory generation algorithms based on the solution path have to perform expensive collision checking in order to prevent this. Any deviation without an explicit upper bound error model in the collision checking may result in a collision during execution of the motion and simply sending a list of waypoints to an industrial controller without a model of its trajectory generation can lead to undesired behavior.

This work presents a new approach to trajectory generation that represents the trajectory between two adjacent waypoints in a different way and is able to explore the robot manipulator's full potential in reducing the motion time. In the same way as Haschke et al. [2] and Kröger et al. [3], the trajectory between two consecutive waypoints are generated with trapezoidal acceleration profiles, which are also referred to as seven-segment acceleration profiles [4]. Compared to cubic splines, seven-segment trajectories offer more optimization possibilities, while increasing the optimization complexity.

The main problem arises from an optimization complexity that increases exponentially with the number of waypoints. It will become impossible to perform the optimization when the waypoints exceed a certain amount. In this paper we develop an approach that can efficiently reduce this problem to a linear complexity, thus making the optimization adaptable to any number of waypoints. This approach is inspired by Model Predictive Control [5], that predicts the system future state based on a formalized model. By decomposing all waypoints into many consecutive batches, where each one is bridged by two overlapping waypoints, the motion states of these waypoints can be predicted and updated by the

Jianjie Lin and Markus Rickert are with fortiss, An-Institut Technische Universität München, Munich, Germany.

Nikhil Somani and Alois Knoll are with Robotics and Embedded Systems, Department of Informatics, Technische Universität München, Munich, Germany.

Biao Hu is with College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China.

optimization of two adjacent batches. In the end, the whole trajectory is solved by successively optimizing every batch. On the other hand, the optimization performance is very sensitive to the initial point. We discuss in detail how to find an appropriate initial point that enables the optimal results to be obtained in a short time and with a high success rate. In addition, the algorithm is extended to also consider trajectory behavior in Cartesian space.

II. Related Work

The problem of generating time-optimal and smooth trajectories has been studied extensively in previous work. The proposed trajectory planning techniques can be generally categorized into two categories: online real-time planning and offline planning. Online real-time trajectory planning targets a dynamic and fast modification of the planned trajectories in case of unforeseen events. The online approaches often rely on the manipulator's current state and the goal state to generate the motion trajectory. For example, by using quintic splines, an online trajectory generation with jerk bound was shown in [6]. Besides, Haschke et al. [2] presented an online trajectory planner with an arbitrary starting state and an end velocity of zero. Kröger et al. [3], [7] and Lange et al. [8] explored in more depth the online generation of trajectories for manipulators with arbitrary start and goal states. However, those online approaches are often limited to two waypoints. For a complicated path with many intermediate waypoints, it is often very difficult to apply these approaches for finding the time-optimal trajectory.

For a well-defined task, the offline trajectory planning aims at finding an optimal trajectory in space or time. To represent the trajectory, a polynomial curve is often used. Generally, the higher the degree of the polynomial curve, the more precise can the motion trajectory properties be tuned. A third-degree polynomial curve is necessary to provide a limit on the jerk of the motion. In order to find the optimal trajectory with a bounded jerk, cubic splines and B-splines are often used to represent the trajectory between two successive waypoints. Thompson and Patel [9] proposed an approach to approximately construct joint trajectories using B-splines. Saravanan et al. [10] developed an optimal trajectory planning approach based on the evolutional theory using uniform cubic B-splines. Gasparetto et al. [11], [12] adopted cubic splines to optimize an objective function composed of execution time and squared jerk. Liu et al. [13] took further steps to optimize the motion time with the combination of cubic splines in Cartesian space and septuple B-splines in joint space. Although cubic splines and B-splines simplify the trajectory planning problem, they are not able to explore the robot manipulator's full capacity to minimize its cycle time.

Dahl et al. [14] address trajectories that are computed to reach the motor's torque limits and introduce two methods for online scaling of calculated trajectories that allow compensating for modeling errors, disturbances, and uncertainties. The algorithm in [15] also considers online scaling with a focus on torque values for tracking an existing trajectory.

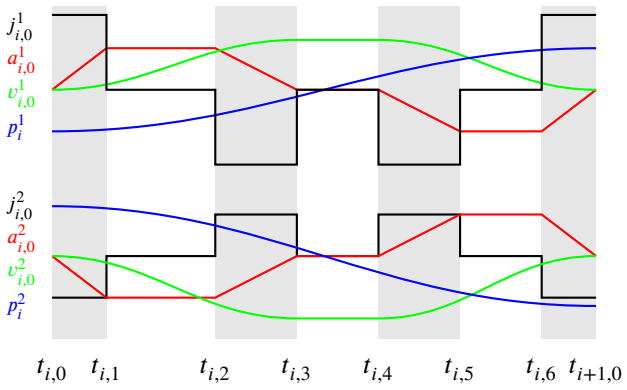


Fig. 1: The seven-segment motion profiles for two axes, where the segments of each dimensional motion are synchronized.

Antonelli et al. [16] present an offline algorithm that supports several constraints on the trajectory, including limits on the jerk and torque values. They propose blending at waypoints by overlapping segments and superimposing them with a clotoid blend that fulfills the specified constraints.

III. Problem Formulation

This paper assumes that a geometric path designed for the robot manipulator to perform a task is available. This path consists of a sequence of waypoints defined in either Cartesian or joint space. The studied problem is how to generate a motion trajectory so that the robot manipulator can pass through those waypoints with the minimum time and without violating its kinematic constraints.

A. Trajectory Model

Given the waypoints required for the task, we have to ensure that the motion of each joint dimension is synchronized at every waypoint. In contrast with prior work [2], [3], [6] that only synchronize motions at every waypoint, this paper requires that each dimensional motion is synchronized in every segment as shown in Fig. 1, leading to two benefits. Firstly, the trajectory becomes more smooth by sharing the three motion phases in every dimension. In particular, by sharing the same constant velocity phase, the robot manipulator is able to perform tasks that demand high-precision and high-stability in this phase, e.g., gluing and painting. Secondly, the search space in the optimization is reduced as each dimensional motion has the same time segments.

The waypoints in the k-th dimension ($k < m$) are indicated as $(p^k_0, p^k_1, ..., p^k_{n-1})$, where m is the number of degrees of freedom, n is the total number of waypoints, and p^k_i indicates the position in the dimension k at the waypoint i. For a path in configuration space, p^k_i presents the joint position. In a Cartesian space path, this refers to the Cartesian position. The trajectory between any two consecutive waypoints is modeled as a trapezoidal acceleration or seven-segment acceleration profile.

We consider a trajectory between p^k_i and p^k_{i+1}. A typical trajectory profile is shown in Fig. 1, where the motion time is

divided into seven segments or three phases. The acceleration phase from $t_{i,0}$ to $t_{i,3}$ has increasing velocity. It is followed by the constant velocity phase from $t_{i,3}$ to $t_{i,4}$. The velocity is decreasing in the final deceleration phase from $t_{i,4}$ to $t_{i,7} = t_{i+1,0}$. The jerk profile has a fixed value of zero in the three time segments $(t_{i,1}, t_{i,2})$, $(t_{i,3}, t_{i,4})$, and $(t_{i,5}, t_{i,6})$ due to a constant acceleration in these segments.

We denote the acceleration, velocity and position at $t_{i,h}$ as $a_{i,h}$, $v_{i,h}$, and $p_{i,h}$, respectively. The jerk in the segment $(t_{i,h-1}, t_{i,h})$ is labeled as $j_{i,h}$ with $h \in [0, \dots, 6]$. Then, for the time $t \in (t_{i,h}, t_{i,h+1})$, the time segment can be defined as $\Delta t = t - t_{i,h}$. The acceleration, velocity, and position profiles can be derived from the previous segment $(t_{i,h-1}, t_{i,h})$ as:

$$\begin{aligned} j^k_{i,h+1}(t) &= j^k_{i,h+1}\,, \\ a^k_{i,h+1}(t) &= a^k_{i,h} + j^k_{i,h+1}\Delta t\,, \\ v^k_{i,h+1}(t) &= v^k_{i,h} + a^k_{i,h}\Delta t + \frac{1}{2} j_{i,h+1}\Delta t^2\,, \\ p^k_{i,h+1}(t) &= p^k_{i,h} + v^k_{i,h}\Delta t + \frac{1}{2} a^k_{i,h}\Delta t^2 + \frac{1}{6} j_{i,h+1}\Delta t^3\,. \end{aligned} \tag{1}$$

B. Kinematic Constraints

In any segment, the motion should not violate the kinematic constraints of the robot. Note, that the acceleration is a monotonous function with time and piecewise smooth. Hence, to guarantee the kinematic constraints within a segment, we only need to guarantee the kinematic constraints at both ends of this segment.

Our aim is to find the time $t_{i,h}$ and jerk $j_{i,h}$ such that the motion time of the robot manipulator passing through all waypoints can be minimized. Also, we aim to minimize the overall time of the trajectory, i.e., $t_{n,0}$. The kinematic constraints $\forall i \in [0, \dots, n-1]$ are defined as:

$$\begin{aligned} p^k(t_{i,7}) &= p^k(t_{i+1,0}) = p^k_{i+1} \\ v^k(t_{0,0}) &= v^k(t_{n,0}) = 0 \\ a^k(t_{0,0}) &= a^k(t_{n,0}) = a^k(t_{i,3}) = 0 \\ j^k_{i,h} &= 0\,, && \forall h \in [1,3,5] \\ |a^k(t_{i,h})| &\le a^k_{\max}\,, && \forall h \in [0, \dots, 6] \\ |v^k(t_{i,h})| &\le v^k_{\max}\,, && \forall h \in [0, \dots, 6] \\ |j^k_{i,h}| &\le j^k_{\max}\,, && \forall h \in [0,2,4,6] \end{aligned} \tag{2}$$

At the initial and final points, i.e., at $t_{0,0}$ and $t_{n,0}$, the acceleration and velocity are equal to zero. Setting $a^k(t_{i,3}) = 0$ guarantees that the velocity will remain constant during the constant velocity phase.

In (2), since $v^k(t_{0,0})$ and $a^k(t_{0,0})$ are equal to zero and p^k_1 is known, the whole trajectory can be generated based on $t_{i,h}$ and $j^k_{i,h}$. As $j^k_{i,h}$ is equal to zero $\forall h \in [1,3,5]$, the unknown variables are $t_{i,h}$ $\forall i \in [0, \dots, n-1]$ and $\forall h \in [0, \dots, 6]$, as well as $j^k_{i,h}$ $\forall i \in [0, \dots, n-1]$ and $\forall h \in [0, 2, 4, 6]$. Therefore, the total number of unknown variables is $7\,(n-1)$ for the time variables and $4\,(n-1)$ for the jerk variables.

C. Constraints in Cartesian Space

In the previous section, a trapezoidal model is implemented in configuration space. However, in typical robot applications, robot arm motions in Cartesian Space are important as well. By only optimizing the trajectory in configuration space, the behavior in Cartesian space might be unpredictable. Hence, we propose an approach to include Cartesian space constraints in the optimization process. We use the forward kinematics function FK to get the Cartesian position $\boldsymbol{x} \in \mathrm{SE}(3)$ of the end effector for a given joint configuration $\boldsymbol{q} \in R^m : \boldsymbol{x} = \mathrm{FK}(\boldsymbol{q})$, where $\boldsymbol{q} = \{q^0, \dots, q^{m-1}\}$. Here, the straight line interpolation between waypoints in the Cartesian space will be considered. Since the trapezoidal model cannot guarantee a strict straight line, an approximated straight line behavior will be targeted. We introduce an additional term in the objective function to regulate the distance of the calculated trajectory to the straight line:

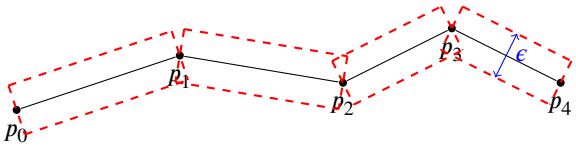


Fig. 2: The error bound for the straight line deviation. The maximum allowed line variation ϵ is calculated based on the variable λ and the length of the individual path segment.

$$\begin{aligned} f_1 &= \sum_{i=0}^{i=n-1} \sum_{h=0}^{h=6} g_1(\boldsymbol{x}) \\ d_1 &= \frac{\|\Delta x_{i,h\to 0} \times \Delta x_{i+1\to i,0}\|}{\|\Delta x_{i+1\to i,0}\|} \\ g_1(\boldsymbol{x}) &= \begin{cases} 0 & \text{if } d_1 < \epsilon_1, \\ d_1 & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

where $\Delta \boldsymbol{x}_{i,h\to 0} = \boldsymbol{x}_{i,h} - \boldsymbol{x}_{i,0}$, $\Delta \boldsymbol{x}_{i+1\to i,0} = \boldsymbol{x}_{i+1,0} - \boldsymbol{x}_{i,0}$, and $\boldsymbol{x}_{i,h}$ is the Cartesian position at the waypoint i and at the time segment $t_{i,h}$. The variable ϵ_1 describes the error bound for a path segment (Fig. 2) and is defined by a scalar λ and the length of a path segment:

$$\epsilon_1 = \|\lambda\, \Delta x_{i+1\to i,0}\|\,. \tag{4}$$

For n waypoints, the optimization function has to consider $7n$ additional scalar terms.

D. Linearity Constraints in Configuration Space

Here, the straight line interpolation $l_{i,i+1}$ between waypoints in the configuration space will be considered. Similar to Section III-C, we introduce an additional term in the objective function to minimize the distance of the calculated trajectory to the straight line in configuration space:

$$\begin{aligned} f_2 &= \sum_{i=0}^{i=n-1} \sum_{h=0}^{h=6} g_2(\boldsymbol{x}) \\ d_2 &= \mathrm{distance}(\boldsymbol{p}_{t_{i,h}}, l_{i,i+1}) \\ g_2(\boldsymbol{x}) &= \begin{cases} 0 & \text{if } d_2 < \epsilon_2, \\ d_2 & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

where $l_{i,i+1}$ is the straight line connecting $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ and distance$(\boldsymbol{p}_{t_{i,h}}, l_{i,i+1})$ is calculated as

$$\left\| (\boldsymbol{p}_{t_{i,h}} - \boldsymbol{p}_i) - \frac{(\boldsymbol{p}_{t_{i,h}} - \boldsymbol{p}_i)\,(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i)^2}{\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|} \right\|. \tag{6}$$

IV. Optimization Approaches

The trajectory generation can be formulated as a nonlinear and non-convex constraint optimization problem, which can be solved by means of the sequential quadratic programming algorithm that decomposes the non-convex problem into sequential convex problems. We use the the SLSQP [17] solver from NLOpt [18] for our implementation. With the high dimensionality of the configuration space and a large number of waypoints, the complexity of the optimization problem and the optimization space can increase significantly. A good initial estimate of the trajectory noticeably affects the convergence of the optimization routine. To handle this problem, we apply concepts from Model Predictive Control (MPC).

A. Optimization Problem Formulation

The purpose of the constrained optimization problem is to optimize the time of the whole trajectory by making use of the trapezoidal acceleration model. The optimization parameters X for this problem consist of the time and jerk values at each segment of the trajectory. Using these values, the acceleration, velocity, and position can be derived automatically using (1). For the time optimization part of the objective function, we try to minimize $\Delta t_{i,h} = t_{i,h} - t_{i,h-1}$ for every value of h and i. The objective function is then defined as

$$f(X) = \sum_{i=0}^{i=n-1} \sum_{h=0}^{h=6} \Delta t_{i,h}\,, \tag{7}$$

where

$$\begin{aligned} X &= (\{\boldsymbol{t}_0, \dots, \boldsymbol{t}_i, \dots, \boldsymbol{t}_{n-1}\}, \{\boldsymbol{j}_0, \dots, \boldsymbol{j}_i, \dots, \boldsymbol{j}_h\})\,, \\ \boldsymbol{t}_i &= \{\Delta t_{i,0}, \dots, \Delta t_{i,6}\}\,, \\ \boldsymbol{j}_i &= \{\{j^0_{i,0}, j^0_{i,2}, j^0_{i,4}, j^0_{i,6}\}, \dots, \{j^{m-1}_{i,0}, j^{m-1}_{i,2}, j^{m-1}_{i,4}, j^{m-1}_{i,6}\}\}\,. \end{aligned}$$

Besides the object function, the constraints also play an important role in solving this problem. Roughly, the constraints comprise nonlinear inequality and equality constraints as well as lower and upper bounds.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & f(X) \\ \text{subject to} \quad & \text{lb} \le X \le \text{ub} \\ & c(X) \le 0 \\ & \text{ceq}(X) = 0 \end{aligned} \tag{8}$$

The multidimensional trapezoid model consists of $7n{+}4nm$ optimization variables, a total of $14mn$ nonlinear inequality constraints, and $mn + nm$ nonlinear equality constraints.

Algorithm 1 Trajectory optimization based on Model Predictive Control

Input: $\boldsymbol{p}$, $\boldsymbol{v}_{\max}$, $\boldsymbol{a}_{\max}$, $\boldsymbol{j}_{\max}$
Output: Ω *time and jerk profile*

$T_{\text{tentative}} \leftarrow \varnothing$
$J_{\text{tentative}} \leftarrow \varnothing$
for i = 0 to $n - h$ **do** *h: receding horizon*
  if i == 0 **then**
    $(\boldsymbol{t}_i, \{\boldsymbol{t}_{i+1}, \dots, \boldsymbol{t}_{i+h}\}, \boldsymbol{j}_i, \{\boldsymbol{j}_{i+1}, \dots, \boldsymbol{j}_{i+h}\}) \leftarrow$ InitialPoint($\{\boldsymbol{p}_i, \dots, \boldsymbol{p}_{i+h}\}$)
    $X_{\text{init}} \leftarrow (\boldsymbol{t}_i, \{\boldsymbol{t}_{i+1}, \dots, \boldsymbol{t}_{i+h}\}, \boldsymbol{j}_i, \{\boldsymbol{j}_{i+1}, \dots, \boldsymbol{j}_{i+h}\})$
  else
    $(\boldsymbol{t}_{i+h}, \boldsymbol{j}_{i+h}) \leftarrow$ InitialPoint($\{\boldsymbol{p}_{i+h-1}, \boldsymbol{p}_{i+h}\}$)
    $X_{\text{init}} \leftarrow (T_{\text{tentative}}, \boldsymbol{t}_{i+h}, J_{\text{tentative}}, \boldsymbol{j}_{i+h})$
  end if
  $X_{\text{optimized}} \leftarrow$ JointSolver($X_{\text{init}}, \boldsymbol{v}_{\max}, \boldsymbol{a}_{\max}, \boldsymbol{j}_{\max}$) *Eq. 8*
  if optimize Cartesian constraints **then**
    $X_{\text{optimized}} \leftarrow$ OptimizeCartesian($X_{\text{optimized}}, \boldsymbol{v}_{\max}, \boldsymbol{a}_{\max}, \boldsymbol{j}_{\max}$)
  end if
  $(\boldsymbol{t}_{\text{optimized}}, T_{\text{tentative}}, \boldsymbol{j}_{\text{optimized}}, J_{\text{tentative}}) \leftarrow X_{\text{optimized}}$
  $\Omega \leftarrow \Omega \cup \{\boldsymbol{t}_{\text{optimized}}, \boldsymbol{j}_{\text{optimized}}\}$ *Output optimized profile for the part between p_i to p_{i+1}*
end for
return Ω

Algorithm 2 OptimizeCartesian() function for optimizing Cartesian linearity constraints

Input: $X_{\text{optimized}}, \boldsymbol{v}_{\max}, \boldsymbol{a}_{\max}, \boldsymbol{j}_{\max}$
Output: $X_{\text{optimized}}$

for j = 1 to $7h$ **do** *trajectory profile with 7 segments*
  $X_{\text{optimized}} \leftarrow$ CartesianSolver($X_{\text{optimized}}, \boldsymbol{v}_{\max}, \boldsymbol{a}_{\max}, \boldsymbol{j}_{\max}$) *Eq. 9*
end for
return $X_{\text{optimized}}$

B. Optimization for Linearity Constraints

In order to include linearity constraints in the Cartesian or configuration space (Algorithm 2), the optimization problem from (8) is updated as:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & f(X) + w_1 f_1(X) + w_2 f_2(X) \\ \text{subject to} \quad & \text{lb} \le X \le \text{ub} \\ & c(X) \le 0 \\ & \text{ceq}(X) = 0 \end{aligned} \tag{9}$$

The weights w_1 and w_2 denote the relative importance of the Cartesian or configuration space linearity constraints. These parameters affect the optimization problem and need to be tuned according to the use case.

C. MPC-based Optimization Approach

With a high number of degrees of freedom and multiple waypoints, the optimization problem is a highly nonlinear and non-convex constraint problem. It is not realistic to optimize all waypoints within one optimization step. However, the waypoints can be divided into consecutive batches. The remaining problem is then to connect all the individual batches. Directly connecting all batches will result in a zero velocity at all connecting points. In order to avoid this situation, we adopt the idea behind Model Predictive

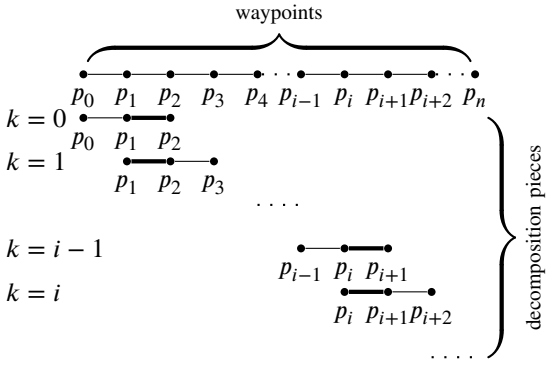


Fig. 3: The waypoints have been decomposed into $n - 2$ batches. Each batch consists of three waypoints, where the former and last two waypoints (*bold lines*) are respectively overlapped with their former and subsequent batches.

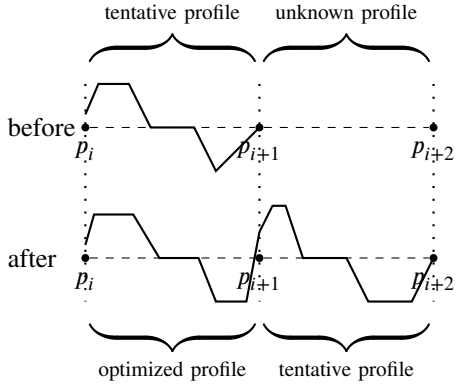


Fig. 4: The conceptual acceleration profile of batch i before and after its optimization.

Control approaches [19]. As shown in Fig. 3, we first need to set a receding horizon. We use a value of two for this, therefore only the next two waypoints p_i and p_{i+1} will be taken into account at waypoint p_{i-1} and the batch contains the waypoint set $\{p_{i-1}, p_i, p_{i+1}\}$. The batch including these three waypoints is used as an input for the optimization solver, but only the result between p_{i-1} and p_i will be used and the result between p_i and p_{i+1} will be discarded. However, we can still use this result as initial guess for the next optimization step, since the next batch will contain the waypoints $\{p_i, p_{i+1}, p_{i+2}\}$. Hence, every batch shares two waypoints with its former and subsequent batch. In this way, the trajectory profile between two overlapping waypoints can be predicted by optimizing the former batch and updated by optimizing the subsequent batch.

As shown in Fig. 3, the whole set of waypoints has been successively decomposed into $n - h$ batches, where h is the receding horizon. The optimization problem for the complete set of waypoints is then posed as multiple ($i = n - h$) successive optimization problems, one for each batch. The overlapping waypoints act as a bridge, connecting the optimization results from one batch to the next one. The detailed optimization procedure is presented in Algorithm 1. Now we analyze the optimization of batch i, where the overlapping part with its former batch is between waypoints p_i and p_{i+1}. After the optimization of batch $i-1$, a tentative acceleration profile is predicted on the overlapping part. Then, after the optimization towards the batch i, the tentative profile will become the optimized profile and the unknown profile between p_{i+1} and p_{i+2} becomes the tentative acceleration profile, as illustrated in Fig. 4. Here, the overlapping part is optimized twice so that the trajectories of two successive batches can be bridged efficiently.

In order to keep the consistency at p_i, the acceleration and velocity at p_i that have been optimized from optimization step $i - 1$ will act as initial conditions of optimization step i. The trajectory of batch i is required to end with zero velocity and acceleration. There are two benefits out of this assumption. Firstly, the static state improves the optimizer flexibility to predict a good tentative profile. For example, the trajectory is assumed to be static at p_{i+1} after the optimization on batch $i - 1$. Then, when optimizing the batch i, the static point at p_{i+1} leaves the optimizer with the full possibility of generating a good tentative profile between p_{i+1} and p_{i+2}. Secondly, the initial point for the unknown profile will be chosen by means of the solution without blending, which requires zero velocity and acceleration as initial conditions. Therefore, the initial point can be consistent—a very important property for optimization.

After this optimization on the $n - h$ decomposed batches, the whole trajectory is generated and optimized. There are $h + 1$ waypoints in every batch and the optimization of each batch has the same computational complexity. Therefore, the computation of the whole trajectory scales linearly with n.

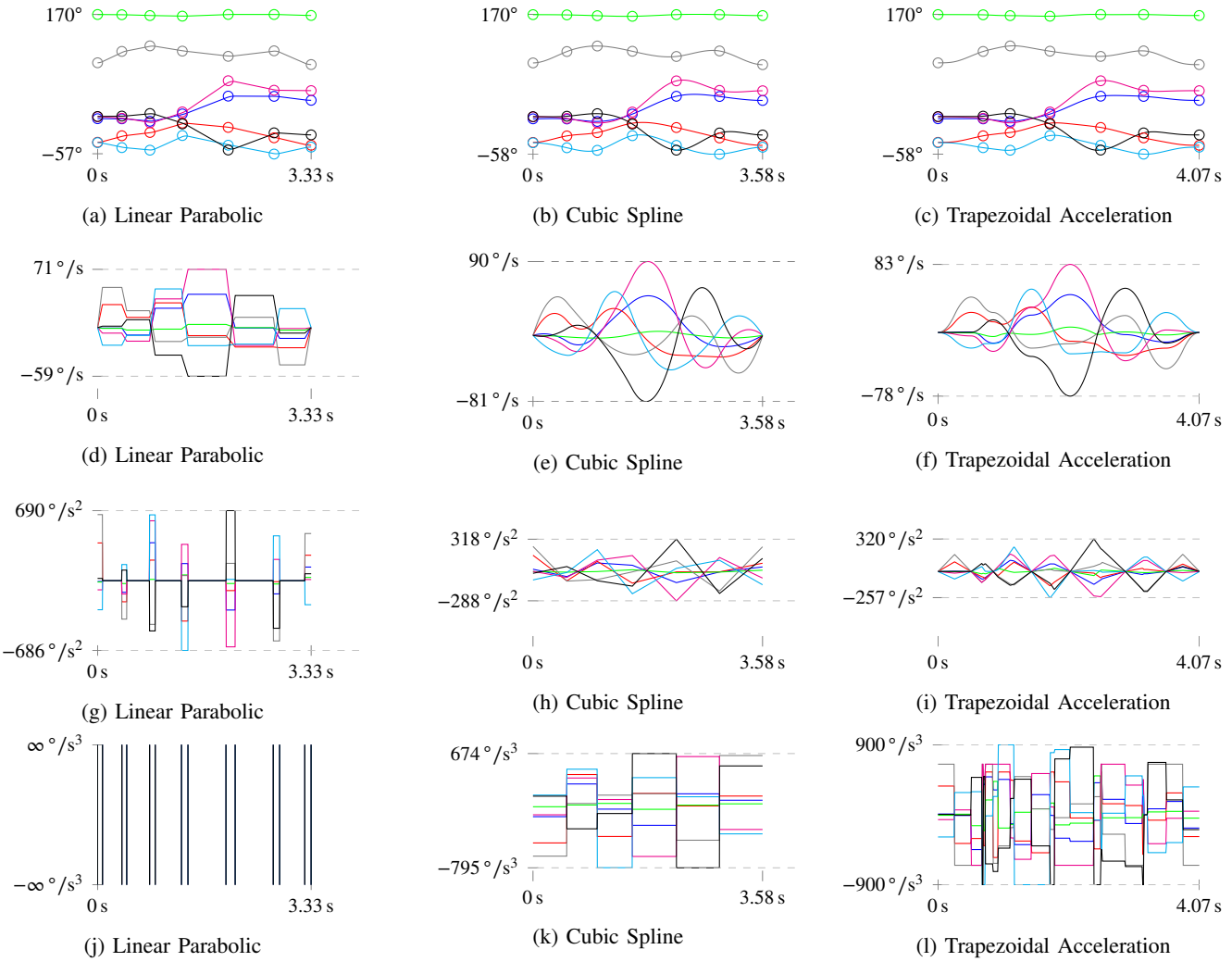
D. Initial Point

Although the optimization complexity has been lowered to a linear scale, the optimization on a specific batch is still complex. As an example, for a robot manipulator with six degrees of freedom and a receding horizon of two, the optimization problem has 62 variables, 48 nonlinear equality and 168 nonlinear inequality constraints (Section III-A). Hence, to efficiently find the optimal result, choosing a good initial point is a very important step.

For the optimization of batch i, the tentative profile is a very promising starting point for the part between p_i and p_{i+1} as this profile has already been optimized tentatively. Regarding the starting point for the other part, i.e., from p_{i+1} to p_{i+2}, a simple way is to derive a profile where the trajectories in all dimensions are synchronized at the endpoint with the shortest time, which is also a non-smooth trajectory. This profile can be derived analytically as follows. First, we independently compute the profile with the shortest motion time for each dimension. Suppose

$$t_{i+1}^k = f(p_{i+1}^k, p_{i+2}^k, v_{\max}^k, a_{\max}^k, j_{\max}^k) \tag{10}$$

represents the shortest time for the k-th dimension trajectory, which is a sum of seven time segments. Secondly, the



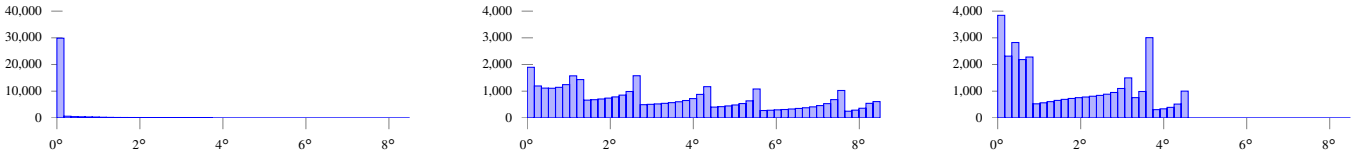
(a) Linear Parabolic (b) Cubic Spline (c) Trapezoidal Acceleration

(d) Linear Parabolic (e) Cubic Spline (f) Trapezoidal Acceleration

(g) Linear Parabolic (h) Cubic Spline (i) Trapezoidal Acceleration

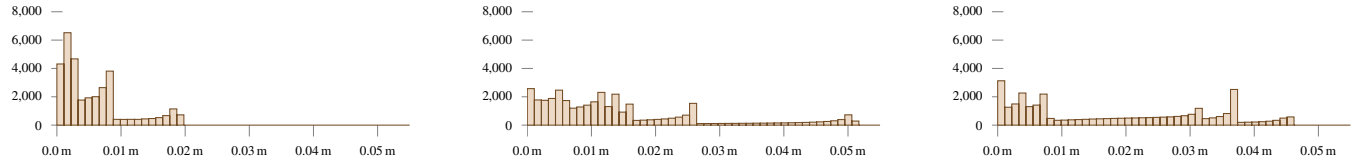
(j) Linear Parabolic (k) Cubic Spline (l) Trapezoidal Acceleration

Fig. 5: A comparison of three different trajectory profiles for the 7-DOF Kuka LWR example in Fig. 8a–8d. The individual plots show the respective (a)–(c) position, (d)–(f) velocity, (g)–(i) acceleration, and (j)–(l) jerk profiles.



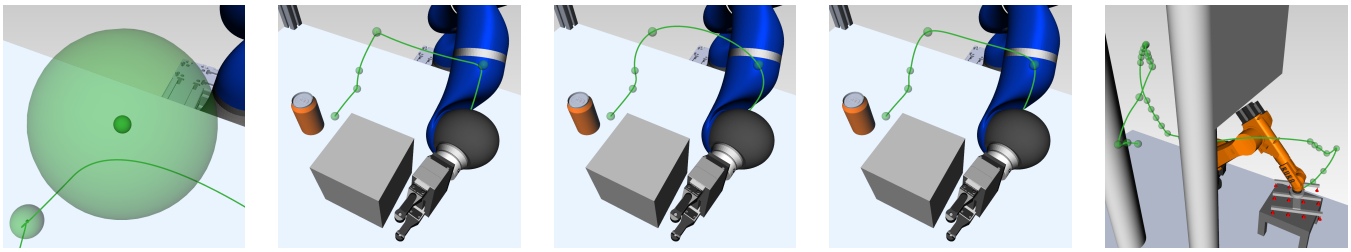
(a) Joint dev. w/ Linear Parabolic (b) Joint dev. w/ Cubic Spline (c) Joint dev. w/ Trapezoidal Acceleration

Fig. 6: A comparison of the deviations from straight lines in configuration space for the Kuka LWR example of Fig. 8a–8d. This property is especially important for ensuring collision free paths calculated by a motion planner. In this example, the error bound $\epsilon_2 = 0.1$ and optimization weights are $w_1 = 0.1$ and $w_2 = 0.1$.



(a) Cartesian dev. w/ Linear Parabolic (b) Cartesian dev. w/ Cubic Spline (c) Cartesian dev. w/ Trapezoidal Acc.

Fig. 7: A comparison of the deviations from straight lines in Cartesian space for the Kuka LWR example of Fig. 8a–8d. This property is especially important for following paths of a Cartesian task. In this example, the error bound ϵ_1 is calculated based on a value of $\lambda = 0.1$ and optimization weights are set as $w_1 = 0.1$ and $w_2 = 0.1$.



(a) Linear Parabolic (b) Linear Parabolic (c) Cubic Spline (d) Trapezoidal Acc. (e) Trapezoidal Acc.

Fig. 8: Trajectories for different interpolation models on path planning use cases for (a)–(d) a Kuka LWR next to a table with a parallel gripper and (e) a Kuka KR60-3 next to a wall and three columns with a vacuum gripper.

maximum time among all dimensions is chosen as

$$t_{i+1}^{\max} = \max t_{i+1}^{k}\,, \quad \forall k \in [0, \ldots, m-1]\,. \tag{11}$$

Then, in order to scale up the motion time of all dimensions to t_{i+1}^{max}, the maximum jerk of other dimensions is adjusted so that kinematic constraints at each dimension are not violated:

$$j_{\max}^{k'} = (t_{i+1}^{k} / t_{i+1}^{\max})^3 j_{\max}^{k}\,, \quad \forall k \in [0, \ldots, m-1]\,. \tag{12}$$

Lastly, the time profile of the dimension that has the maximum time and the modified jerk of all dimensions are used as the initial point for the optimization.

We have to mention that there are two exceptions that are not optimized twice. In Algorithm 1, the parts $\{p_1, p_2\}$ of the first batch and $\{p_{n-1}, p_n\}$ of the last batch will only be optimized once as they do not overlap with any other batch. In addition, even after the initial point is chosen as described above, this does not guarantee that a solution can be found. Therefore, after the chosen initial point fails, a uniform random noise is added to this initial point and the optimization is restarted to avoid getting stuck in a local minimum [20].

V. Experimental Evaluations

We evaluate the performance of the proposed optimization approaches and two state-of-the-art approaches: Linear Parabolic interpolation [21] and Cubic Splines [22]. The Linear Parabolic model divides each interpolation step into three parts: two parabolic blends with the previous and next trajectory segments and a linear interpolation with constant velocity in the middle. We test these approaches on two path planning examples and present a detailed analysis of their performance and properties. For both the Cubic Spline and Linear Parabolic models, we ensured that the acceleration and velocity limits of each joint are satisfied by scaling the timescales of the trajectory segments. For the Trapezoid Acceleration model, this is handled intrinsically by our optimization routine.

We show how we fulfill two key requirements of path planning algorithms, i.e., reaching waypoints exactly and maintaining near-linear trajectories in the configuration space between successive waypoints. Fig. 8b–8d shows a path planning example for the Kuka LWR consisting of seven waypoints indicated by green spheres. The interpolated trajectories are shown by green lines. For the Cubic Spline and our approach, the calculation will consider all kinematic constraints, as described in Algorithm 1. For the Linear Parabolic model, we calculate the time segment for each dimension by only taking into account the velocity and acceleration constraints. The time for each segment is chosen according to the most constrained dimension/joint, ensuring that all robot joints are within their velocity and acceleration limits. The collision-free paths in the examples are computed using the Robotics Library [23], which is also used for kinematics calculations and simulation. All evaluations were performed on a laptop with a 2.60 GHz Intel Core i7-6700HQ and 16 GB of RAM.

The trajectory calculated by the Cubic Spline model (Fig. 8c) deviates significantly from a straight line interpolation between successive waypoints. The trajectory from the Linear Parabolic method (Fig. 8b) follows a near-linear interpolation but is clearly non-smooth. Our method (Fig. 8d) satisfies both requirements. We plot the configuration space positions, velocities, and accelerations for each of these models in Fig. 5. Cubic Spline and Trapezoidal Acceleration methods are guaranteed to pass through all waypoints exactly. Linear Parabolic interpolation blends around the waypoints and fails to hit inner waypoint as demonstrated in Fig. 8a. The percentage of the trajectory segment that is blended influences this deviation from the waypoints. We set this to 20% for our experiments, resulting in deviations of 0.0°, 2.3°, 1.3°, 3.0°, 4.5°, 2.2°, and 0.0° for the seven waypoints.

Cubic Spline has a movement time of 3.58 s, while the approach presented in this paper is slightly longer with 4.07 s. The Linear Parabolic one only takes 3.33 s and is the shortest one. Note, that Linear Parabolic cannot consider the jerk limits, which are very important to prevent any damage to the motors. In contrast to our approach, Cubic Spline does not start and end with an acceleration value of zero and it mathematically only allows one point at the maximum value at each path segment. The trapezoidal acceleration model can hold the maximum velocity and acceleration for a longer time. Over 20 runs, the optimization for the Linear Parabolic each takes 1.45 ± 0.03 s, Cubic Spline 0.07 ± 0.01 s, and the more complex trapezoidal acceleration model 9.58 ± 2.05 s.

The simulation of these three trajectories is shown in Fig. 8b–8d. A key property from the underlying path plan-

ning algorithm is that collision-free paths are only guaranteed as long as the interpolation between the waypoints follows a straight line in configuration space. In practice, the path planning algorithms usually consider a minimum distance from obstacles as a safety margin. Hence, the straight line interpolation requirement can also be relaxed according to this distance. We evaluate the deviations in configuration space, as shown in Fig. 6a–6c. Note, that in our approach we add an additional term to minimize the deviation error. It can be considered as a soft constraint and the problem will be optimized to achieve minimum deviation. However, it cannot be guaranteed to stay within the error bound.

In Fig. 7a–7c, the deviation from the straight line between waypoints in Cartesian space is illustrated. It can be seen that the Linear Parabolic shows the smallest error as it consists of linear movement and a blending around the waypoints. The trapezoidal model cannot guarantee straight line movement. However, by adjusting the duration of the constant velocity phase, the motion can be made closer to a straight line interpolation. In our approach, we employ an error bound object function to minimize the deviation from the straight line. From the results, we can observe that the Cartesian errors are lowest for Linear Parabolic, and highest for Cubic Spline while ours lies in the middle. Our approach allows us to adjust this deviation by tuning λ, w_1, and w_2.

In Fig. 8e, we evaluate our approach in a more complex example consisting of 31 waypoints. Optimizing the trajectory using the whole set of waypoints in one optimization problem requires several hours, whereas our approach can find each solution in 222.50 ± 49.75 s over a total of 10 attempts. The robot is guaranteed to pass through all waypoints while exhibiting a smooth 16.63 s trajectory and obeying all kinematic constraints. The optimization of the Linear Parabolic's 11.61 s trajectory requires 6.05 ± 0.41 s and the Cubic Spline's 13.89 s solution takes 0.33 ± 0.01 s.

VI. Conclusion

In this paper, we have presented an approach to find a time-optimal trajectory passing through given waypoints under kinematic constraints. Unlike prior approaches that model the trajectory between two adjacent waypoints as a Cubic Spline or Linear Parabolic segments, this approach adopts a trapezoidal acceleration profile to represent the trajectory. We require this trajectory to move through all waypoints while exploiting the manipulator's capabilities in order to reduce the motion time and ensuring that the kinematic limitations are always satisfied. Our proposed bridged optimization approach has linear complexity with the number of waypoints compared to a full trajectory optimization. Evaluations of two practical examples from path planning have shown how the three approaches compare to each other.

There are also some limitations that we observed in our experiments. In cases where there are large rotations between successive waypoints, our method can take longer to converge and sometimes not provide an optimal solution. Additionally, the Cartesian linearity constraint is only included as a soft constraint and might not work in applications with more stringent linearity requirements such as welding or deburring. In such cases, the Cartesian linearity constraint should be formulated as an inequality constraint instead. Future work includes handling more complicated cases and examples from planners that support Cartesian constraints.

References

[1] J. Reif, "Complexity of the mover's problem and generalizations," in *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1979, pp. 421–427.

[2] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 3248–3253.

[3] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

[4] R. H. Castain and R. P. Paul, "An on-line dynamic trajectory generator," *The Int. J. of Robotics Research*, vol. 3, no. 1, pp. 68–72, 1984.

[5] E. F. Camacho and C. B. Alba, *Model predictive control*, ser. Advanced Textbooks in Control and Signal Processing. Springer, 2007.

[6] S. Macfarlane and E. A. Croft, "Jerk-bounded manipulator trajectory planning: Design for real-time applications," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, 2003.

[7] T. Kröger, *On-line trajectory generation in robotic systems*, ser. Springer Tracts in Advanced Robotics. Springer, 2010, vol. 58.

[8] F. Lange and A. Albu-Schäffer, "Path-accurate online trajectory generation for jerk-limited industrial robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 82–89, 2016.

[9] S. E. Thompson and R. V. Patel, "Formulation of joint trajectories for industrial robots using B-splines," *IEEE Transactions on Industrial Electronics*, no. 2, pp. 192–199, 1987.

[10] R. Saravanan, S. Ramabalan, and C. Balamurugan, "Evolutionary optimal trajectory planning for industrial robot with payload constraints," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 11, pp. 1213–1226, 2008.

[11] A. Gasparetto and V. Zanotto, "A technique for time-jerk optimal planning of robot trajectories," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 415–426, 2008.

[12] ——, "Optimal trajectory planning for industrial robots," *Advances in Engineering Software*, vol. 41, no. 4, pp. 548–556, 2010.

[13] H. Liu, X. Lai, and W. Wu, "Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 309–317, 2013.

[14] O. Dahl and L. Nielsen, "Torque-limited path following by online trajectory time scaling," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 5, pp. 554–561, 1990.

[15] J. Moreno-Valenzuela, "Tracking control of on-line time-scaled trajectories for robot manipulators under constrained torques," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2006, pp. 19–24.

[16] G. Antonelli, C. Curatella, and A. Marino, "Constrained motion planning for open-chain industrial robots," *Robotica*, vol. 29, no. 3, pp. 403–420, 2011.

[17] D. Kraft, "A software package for sequential quadratic programming," Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln, Cologne, Germany, Forschungsbericht DFVLR-FB–88-28, 1988.

[18] S. G. Johnson, "The NLopt nonlinear-optimization package." [Online]. Available: http://ab-initio.mit.edu/nlopt

[19] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[20] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Martí, "Scatter search and local NLP solvers: A multistart framework for global optimization," *INFORMS Journal on Computing*, vol. 19, no. 3, pp. 328–340, 2007.

[21] L. Biagiotti and C. Melchiorri, *Trajectory planning for automatic machines and robots*. Springer, 2008.

[22] C. de Boor, *A Practical Guide to Splines*. Springer, 1978.

[23] M. Rickert and A. Gaschler, "Robotics Library: An object-oriented approach to robot applications," in *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2017, pp. 733–740.