HAL
open science

Optimization of the robot and positioner motion in a redundant fiber placement workcell

Jiuchun Gao, Anatol Pashkevich, Stéphane Caro

To cite this version:

Jiuchun Gao, Anatol Pashkevich, Stéphane Caro. Optimization of the robot and positioner motion in a redundant fiber placement workcell. Mechanism and Machine Theory, 2017, 114, pp.170 - 189. 10.1016/j.mechmachtheory.2017.04.009 . hal-01692417

HAL Id: hal-01692417

https://hal.science/hal-01692417v1

Submitted on 2 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimization of the Robot and Positioner Motion in a Redundant Fiber Placement Workcell

Jiuchun Gao[a,b,*], Anatol Pashkevich[a,b] and Stéphane Caro[a,c]

[a] Ecole des Mines de Nantes, 4 rue Alfred-Kastler, Nantes 44307, France.
[b] Institut de Recherches en Communications et Cybernétique de Nantes, UMR CNRS 6597, 1 rue de la Noë, 44321 Nantes, France.
[c] Centre National de la Recherche Scientifique (CNRS), France.
* Corresponding author at: C022B, Ecole des Mines de Nantes, 4 rue Alfred-Kastler, Nantes 44307, France. Tel.: +33 6 58 48 14 16. E-mail address: jiuchun.gao@irccyn.ec-nantes.fr (J. Gao).

Abstract

The paper proposes a new methodology to optimize the robot and positioner motions in redundant robotic system for the fiber placement process. It allows user to find time-optimal smooth profiles for the joint variables while taking into account full capacities of the robotic system expressed by the maximum actuated joint velocities and accelerations. In contrast to the previous works, the proposed methodology possesses high computational efficiency and also takes into account the collision constraints. The developed technique is based on conversion of the original continuous problem into a discrete one, where all possible motions of the robot and the positioner are represented as a directed multi-layer graph and the desired time-optimal motions are generated using the dynamic programming that is applied sequentially for the rough and fine search spaces. To adjust the optimization results to the engineering requirements, the obtained trajectories are smoothed using the spline approximation. The advantages of the proposed methodology are confirmed by an application example that deals with a planar fiber placement robotic system.

Keywords

Redundant robotic system; Optimal motion planning; Graph-based representation; Dynamic programming; Robotized fiber placement

1. Introduction

Currently, composite materials are increasingly used in industry [1, 2]. Compared to traditional ones, they have good strength-to-weight ratio, durability, flexibility of shaping and corrosion resistance [3, 4]. Accordingly, they are extremely attractive for fabrication of large dimensional parts in aerospace, automotive and marine industries. The conventional way of manufacturing composite based components is labor intensive tape laying procedure. This manual process is quite slow and expensive, with low repeatability [3]. For efficient fabrication of composite parts, automated tape laying (ATL) is a more productive method. This is a specific technique which drives a technological tool laying a continuous composite tape onto molds automatically. However, ATL has a drawback that it is limited by the mold shapes. For a mold surface with high variation curvatures, tape wrinkling and overlap appear, and it decreases the mechanical performance of the composite dramatically [3, 5, 6]. For this reason, automated fiber placement (AFP) was proposed as an extension to ATL for fabricating composites with arbitrary surfaces [6, 7]. Instead of laying a single tape in ATL, heated fiber tows (12 to 32) are placed onto molds with desired orientations side-by-side and simultaneously. A pressure roller reinforces the placed fibers in situ [1, 3, 7]. Consequently, AFP becomes to the most widespread composite manufacturing technique because of the increasing demand for complex structural components [6].

The fiber placement process can be implemented by using either specifically designed machines or robotic systems, which are redundant in this application [3]. In the first case, general CNC machines equipped with specific placement head are used. They have no limitations on the workpiece size, but usually are quite expensive and require large work floor areas [7-9]. Compared to the process-dedicated machines, the robotic systems are relatively cheap and flexible allowing changing the product type easily [9]. In comparison to their size, they can provide a large working area. Industrial robotic systems are usually composed of a 6-axis serial robotic manipulator, a specific fiber placement end-effector and an actuated positioner

with one or two degrees of freedom [10, 11]. The workpiece is mounted on the positioner flange and changes its orientation with rotation of the positioner axis.

In robotic fiber placement process, manipulator motion planning is an important issue. The main difficulty here arises because of robotic system redundancy with respect to the manufacturing task. To generate the desired motion, it is required decomposing the given task into a robotic manipulator motion and a positioner motion. In literature, several works deal with this issue. A conventional way is based on redundancy resolution via the generalized inverse (pseudo inverse) of the kinematic Jacobian [12-15]. It gives a unique solution for the differential kinematic equations in the sense of least squares, which corresponds to the smallest Euclidean norm of the displacement vector in the joint space. However, this technique can hardly generate the time-optimal solution taking into account velocity and acceleration constraints of actuators, which are quite important in real-life industrial applications. Another approach of motion planning for redundant robotic system [16, 17] is based on the idea of "master-slave", where the trajectories of the "master" manipulator are assigned firstly, and the corresponding conjugate trajectories of the "slave" are then determined. This method is rather simple and computationally efficient, but assigning the master trajectory is not trivial, especially for complex shape objects. Besides, here it is not possible to take into account the actuator constraints in an explicit way.

To generate the manipulator motion for complex shape workpiece taking into account constraints imposed by the actuators and avoiding collisions between the workcell components, several alternative methods have been developed that are based on direct optimization of an objective function describing the total travelling time, the trajectory smoothness, the quantity of manipulator motions, etc. One of such techniques [18, 19] is based on transformation of the original continuous time-optimal problem into a discrete one, where the robotic manipulator and the positioner joint spaces are discretized and the desired trajectory is represented as the shortest path on the corresponding graph. Then, the conventional discrete optimization techniques are applied to obtain the approximate time-optimal trajectory. A key issue here is related to assigning distances between the adjacent nodes of the graph, which are usually computed considering the velocity constraints only and omitting the acceleration constraints. The latter may lead to an optimal solution that corresponds to non-smooth manipulator trajectories, with essential oscillations in actuator velocities. Besides, conventional methods (Dijkstra algorithm, etc.) are rather time consuming for discretization steps acceptable in practice. Slightly different approach was proposed in [20-23] for the applications with constant Cartesian speed of the technological tool (robotic laser-cutting, spraying and arc-welding). In this case, the travelling time between the adjacent graph nodes is constant, so the required distances may be defined as the largest increments in the joint coordinates corresponding to the node-to-node movements. This allows to reduce oscillations in actuator velocities and to improve the trajectory smoothness, but the basic assumption of these techniques (constant tool speed) is not valid for the fiber placement. Nevertheless, some ideas from the above mentioned works, such as application of dynamic programming and elimination of nodes sequences violating the acceleration constraints, are useful for the problem studied in this paper.

For the fiber placement process, the problem of the manipulator motion planning was studied by Martinec [24] and Mlynek [25], who also assumed that the tool velocity is constant. Another work [26] focuses on the tool path optimization in Cartesian space while the optimal utilization of the robotic system redundancy was replaced by smoothing the end-effector trajectory. Obviously, this kinematic improvement does not allow using full capacities of the actuators, but it contributes to the trajectory smoothness in the configuration space and manufacturing efficiency by minimizing variations of the joint coordinates. To the best of authors' knowledge, there are no techniques directly addressing the problem of the time-optimal motion planning for robotic fiber placement and it is still an open issue.

Besides optimization of the travelling time, improvement of the manipulator motion smoothness is another important issue in fiber placement applications. Smooth trajectories allow considerable reduction of vibrational phenomenon and mechanical wear of the robotic system. Known techniques in trajectory smoothing focused mainly on bounding the jerk value (defined as the derivative of the acceleration with respect to time) during the trajectory planning. For example, the maximum absolute value of the jerk is limited in [27-29], and the integral of the squared jerk along the trajectory is minimized in [27, 30-33]. Among the known works, it is also worth mentioning [30, 31] where an efficient minimum time-jerk trajectory planning algorithm was proposed for non-redundant robotic manipulators. However, these techniques cannot be directly applied to the motion planning for the redundant robotic system.

This paper focuses on optimization of manipulator motions in redundant systems taking into account two main objectives: (i) total travelling time, and (ii) trajectory smoothness. The proposed algorithm is an enhancement of our previous technique [34], it is based on the search space discretization and relevant combinatorial optimization search. In order to reduce computing time, the algorithm is composed of three stages: rough search, local optimization and smoothing. First, the original problem is presented as a combinatorial one after a rough discretization of the search space, and it is solved by using dynamic programming. An initial solution is generated quickly here. Then, the initial solution is sequentially improved by gradually

reducing the discretization step in its neighborhood and applying the same dynamic programming procedure. Further, the trajectory smoothing technique based on polynomial approximation of the positioner displacement profile is applied, and the corresponding robot displacement profiles are updated. This idea of sequential optimization allows us to obtain the desired manipulator motions in practically acceptable time.

The remainder of the paper is organized as follows. Section 2 presents the considered fiber placement robotic system. Section 3 deals with the system and task models and also presents the problem statement. Section 4 proposes the motion planning methodology, including the graph-based presentation of the search space and the developed optimization algorithm. Section 5 contains an application example that confirms the efficiency of the proposed methodology. In Section 6, some limitations and open issues are discussed. Section 7 concludes the contributions and defines future research directions.

2. Robotic fiber placement system

Typical robotic fiber placement system includes a 6-axis robotic manipulator and an actuated positioner with one or two degrees of freedom (see Fig. 1). The robotic manipulator is equipped with a specific technological tool ensuring fiber feeding, heating and compaction. The workpiece is mounted on the positioner allowing changing its orientation in order to improve accessibility of certain zones by the technological tool. In practice, a one-axis positioner is usually sufficient if the workpiece is not too large and its shape is simple. In the case of large dimension workpiece, the robotic manipulator can be located on a linear track increasing the robot workspace and its number of degrees of freedom. It is clear that this architecture including at least 7 actuators is redundant with respect to the considered technological task which requires 6 degrees of freedom only. This redundancy provides user with some flexibility because the same workpiece point can be reached with different configurations of the robotic manipulator and the positioner. On the other side, this essentially complicates the robotic system programming and poses a problem of using the redundancy in the best way.

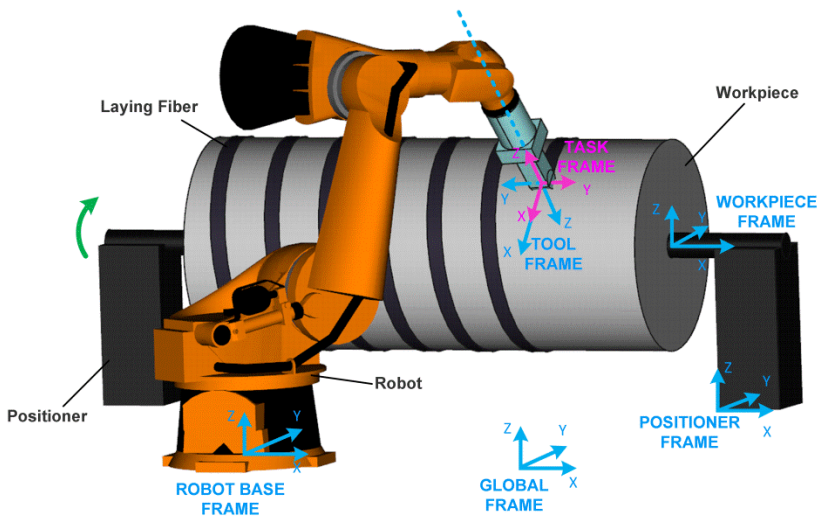

Fig. 1 Typical robotic fiber placement system (6-axis robot and one-axis positioner)

To take into account particularities of the considered application, let us provide some details concerning the robot end-effector that is also called the fiber placement head [3, 6, 35]. This technological tool is usually composed of a material feeder, a fiber heater and a compaction roller, as shown in Fig. 2. The fiber tows are fed and guided towards the desired task locations with desired orientations. To heat fiber tows, a diode laser is used with active control during the placement process. The temperature of the material is controlled by adjusting the laser power, allowing maintaining the prescribed level and avoiding the material overheating. Finally, the heated fiber tows are consolidated in situ by compaction roller and cooled down.

In practice, the fiber placement head may have different designs in order to adapt it to the particular task. For workpieces with approximately cylindrical shape (liners of vessels, etc.) there are winding-oriented head designs. For workpieces with complicated 3D surfaces, there are laying-oriented designs of the head. It is worth mentioning that geometry of the fiber placement head imposes some extra constraints on the robotic manipulator configurations that must be taken into account in the motion planning procedure. In this paper, these constraints are evaluated at the stage of the search space generation, where some nodes of the original task graph are eliminated (they are treated as inadmissible ones because of collisions). In more details, this issue will be addressed in the section 3.3.

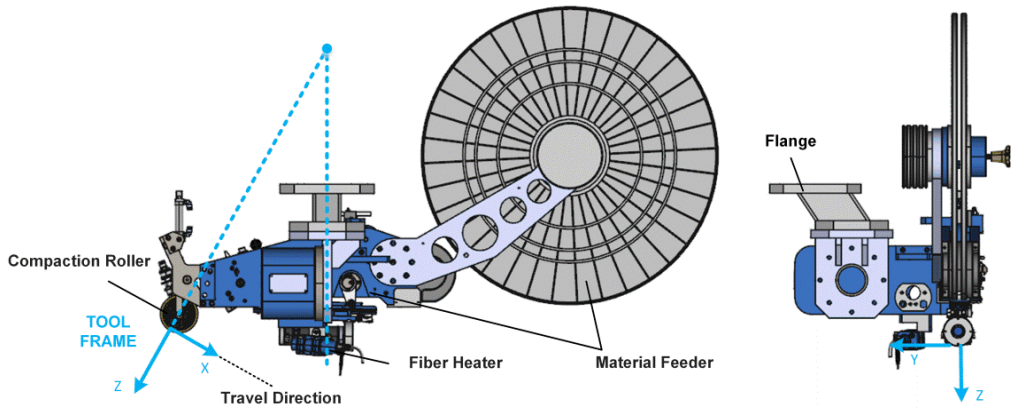

Fig. 2 Robot end-effector for automated fiber placement (AFPT GmbH)

In current industrial practice, programming of robotic system essentially relies on a set of software modules allowing user to prepare the program in off-line mode and avoid time consuming manual teaching. To take into account the system redundancy, we propose the following robot programming procedure presented in Fig. 3. It is assumed that the workpiece model was already created in a CAD system. It is used as the input of the dedicated CAM system that provides the fiber placement path generation in Cartesian space and its discretization (with respect to the workpiece frame). Further, the obtained set of path points and related frames (task model) is entered in the graph generator, which transforms it into a so-called "task graph" that describes all possible motions of the robot and the positioner joints ensuring the desired fiber placement path on the workpiece. It should be mention that this graph takes into account the robotic system redundancy because each graph node represents certain configurations of robot and positioner but each task location corresponds to a number of nodes (graph layer). Using this graph, the next software module provides generation of optimal motions for the robot and positioner. These motions are presented as the "best" path on the graph connecting two boundary layers corresponding to the initial and final locations of the fiber placement task. Finally, the obtained robot and positioner motions are converted into the robotic system program by the post processor. It worth mentioning that this idea of graph-based representation of possible robot/positioner motions was previously successfully used for robotic laser cutting applications [20-22] but these results cannot applied directly here for the considered technology. The open questions are related to formalization of the "best" path notion and development of dedicated algorithm for the "best" path search. These issues are addressed in this paper.

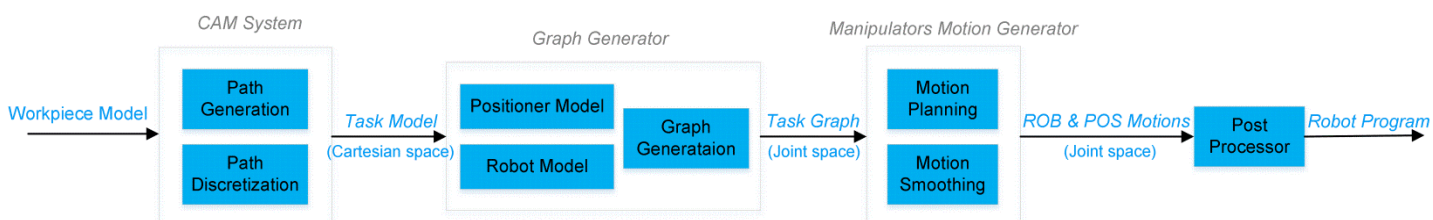

Fig. 3 Preparation of manufacturing process for robot-based fiber placement

3. System model and task formalization

The key problem addressed in this paper is to find optimal motions of robot and positioner ensuring desired trajectory of the technological tool with respect to the workpiece and satisfying some constraints imposed by the system kinematics and the fiber placement technology. To present this problem in a formal way, let us consider in details the system geometry and define optimality criteria for the robot and positioner motions.

3.1 Fiber placement task model

Let us assume that desired fiber placement path is presented as a 3D-augmented line in relevant CAD system and discretized in n segments. This presentation allows obtaining both the Cartesian coordinates of the path points $\mathbf{p}_i = (x_i, y_i, z_i)^T$ and also

the unit vectors $\mathbf{a}_i = (a_{xi}, a_{yi}, a_{zi})^T$ defining the workpiece surface normal directions. To get corresponding end-effector locations required for the robot programming, let us introduce a set of coordinate frames associated with each path point (see Fig. 4). These frames are defined as follows: (i) the frame origin is located at the path point $\mathbf{p}_i$; (ii) the X-axis shows the motion direction and coincides with the path tangent; (iii) the Z-axis points the surface normal direction and coincides with the vector $\mathbf{a}_i$; (iv) the Y-axis is perpendicular to the axes X and Z to complete a right-handed frame. The main difficulty here is related to computing of the X and Y directions described by the unit vectors $\mathbf{n}_i = (n_{xi}, n_{yi}, n_{zi})^T$ and $\mathbf{s}_i = (s_{xi}, s_{yi}, s_{zi})^T$ respectively, which usually cannot be extracted directly from the CAD model. To find these two vectors, the following procedure can be applied. First, the X-direction is estimated approximately using the path increment $\Delta\mathbf{p}_i = (\mathbf{p}_{i+1} - \mathbf{p}_i)$. Then, the Y-direction is computed via the vector product $\mathbf{s}_i = \mathbf{a}_i \times \Delta\mathbf{p}_i$ and normalized as $\mathbf{s}_i := \mathbf{s}_i / \|\mathbf{s}_i\|$. Finally, the vector of the X-direction is computed straightforwardly using the formula $\mathbf{n}_i = \mathbf{s}_i \times \mathbf{a}_i$.

Using the above frames, the fiber placement task can be presented as a set of locations that should be visited sequentially by the robot end-effector in minimum time. Assuming that each location is described by a 4×4 homogenous transformation matrix, the considered task is formalized as follows

$$ {}^{w}\mathbf{T}_{task}^{(1)} \rightarrow {}^{w}\mathbf{T}_{task}^{(2)} \rightarrow \ldots {}^{w}\mathbf{T}_{task}^{(i)} \rightarrow \ldots \rightarrow {}^{w}\mathbf{T}_{task}^{(n)} \tag{1} $$

where

$$ {}^{w}\mathbf{T}_{task}^{(i)} = \begin{pmatrix} \mathbf{n}_i & \mathbf{s}_i & \mathbf{a}_i & \mathbf{p}_i \\ 0 & 0 & 0 & 1 \end{pmatrix}_{4\times4} ; \quad i = 0, 1, 2, \ldots n-1 $$

and all vectors are expressed with respect to the workpiece base frame (see superscript "w"). Equivalent presentation of the considered task relies on 6×1 vectors that include three Cartesian coordinates and three Euler angles that can be easily computed from the orientation matrices $(\mathbf{n}_i, \mathbf{s}_i, \mathbf{a}_i)_{3\times3}$ [36].

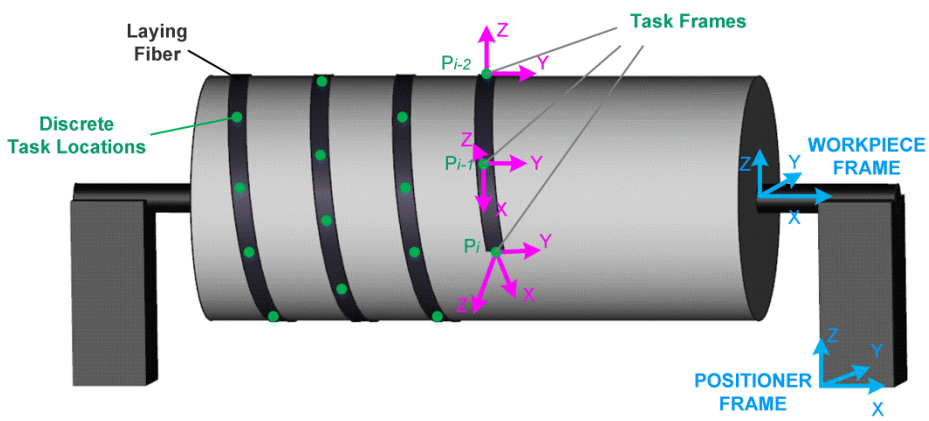

Fig. 4 Discretization of the fiber placement path and definition of the task frames

3.2 Robotic fiber placement system model

As shown in Fig .1, the considered robotic fiber placement system includes two principle mechanisms: an industrial robot (a technological tool manipulator) and a positioner (a workpiece manipulator). It is assumed that their kinematics is known and spatial locations of the output frames depend on the vectors of actuated joint coordinates $\mathbf{q}_R$ and $\mathbf{q}_P$ respectively. Let us derive the kinematic model of the whole system that includes the closed loop created by the robot, workpiece and positioner.

The robot kinematic model describes the end-effector frame location (position and orientation) with respect to the robot base as a function of the actuated joint coordinates. For the serial architecture, the robot model can be presented as a product of the 4×4 homogenous transformation matrices

$$g_R(\mathbf{q}_R) = \prod_{i=1}^{6} {}^{i-1}\mathbf{T}_{Ri}(q_{Ri}) \tag{2}$$

depending on corresponding joint angles [36]. This model is also often referred to as the direct kinematics of the robot. Particular expressions for the matrices ${}^{i-1}\mathbf{T}_{Ri}$ can be obtained using the Denavit-Hartenberg (DH) technique [37], where they are presented as a composition of elementary rotations and translations ${}^{i-1}\mathbf{T}_{Ri}(q_{Ri}) = \mathbf{R}_X(\alpha_{i-1}) \cdot \mathbf{D}_X(a_{i-1}) \cdot \mathbf{R}_Z(q_{Ri}) \cdot \mathbf{D}_Z(d_i)$ depending on both the joint variables q_{Ri} and the parameters a_i, d_i, α_i describing the links/joints geometry.

In practice, it is also often used the inverse kinematic model $g_R^{-1}(.)$ that allows user to compute the vector of the actuated joint variables $\mathbf{q}_R$ corresponding to the desired end-effector location described by a given homogeneous 4×4 matrix $\mathbf{T}$. Compared to the direct kinematics, the inverse one is not trivial since the solution is usually not unique and cannot be expressed in a closed form. For this reason, special types of manipulator architecture are used to satisfy the Pieper condition [38] that ensures the closed-form solution. Otherwise, there are also some numerical techniques to deal with this problem [39, 40]. In our study, the serial robot with last three intersecting axes is employed, which corresponds to the majority of industrial applications. It should be also stressed that in order to obtain a unique solution, the inverse kinematic function augments also include the so-called configuration index μ that determines the posture of the manipulator shoulder, elbow and wrist. For this reason, in the following sections the inverse kinematic function will be referred to as $g_R^{-1}(\mathbf{T}, \mu)$.

The positioner kinematic model describes the positioner mounting flange location with respect to its base as a function of the actuated coordinate vector $\mathbf{q}_P$. Similarly to the above case, it can be also presented as a product of the matrices

$$g_P(\mathbf{q}_P) = \prod_{i=1}^{N_P} {}^{i-1}\mathbf{T}_{Pi}(q_{Pi}) \tag{3}$$

that may be derived using the Denavit-Hartenberg technique (usually $N_P = 1,2$, since typical positioner has one or two degrees of freedom). Because of lacking degrees of freedom, the inverse kinematics for the positioner cannot be solved in strict way [41]. But it is not important for technique developed in this paper that employs the positioner direct kinematic model only.

To obtain the kinematic model of the whole system, let us assume that for technological reasons the end-effector frame and the task frame must be aligned in such way that: (i) the origins of these two frames coincide; (ii) their X-axes are directed similarly; (iii) their Z-axes are opposite. This assumption corresponds to the following homogeneous transformation matrix defining a relation between the end-effector and task frames

$$ {}^{tool}\mathbf{T}_{task} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{4}$$

It allows us to express global locations of all task frames in two ways, using either the robot or positioner kinematic models. In the first case, the matrices ${}^{0}\mathbf{T}_{task}^{(i)}$ describing the task frames are presented as follows

$$ {}^{0}\mathbf{T}_{task}^{(i)} = {}^{0}\mathbf{T}_{Rbase} \cdot g_R(\mathbf{q}_R^{(i)}) \cdot {}^{tool}\mathbf{T}_{task}; \qquad i = 1,2,...n \tag{5}$$

where ${}^{0}\mathbf{T}_{Rbase}$ defines the robot base location relative to the global system that is denoted by the superscript "0" (see Fig. 1). In the second case, similar matrices are expressed as

$$ {}^{0}\mathbf{T}_{task}^{(i)} = {}^{0}\mathbf{T}_{Pbase} \cdot g_P(\mathbf{q}_P^{(i)}) \cdot {}^{w}\mathbf{T}_{task}^{(i)}; \qquad i = 1,2,...n \tag{6}$$

where ${}^{0}\mathbf{T}_{Pbase}$ defines the positioner base location relative to the global system and the superscript "w" denotes the workpiece coordinate system.

After equating these two presentations, one can get the set of equations

$$ {}^{0}\mathbf{T}_{Rbase} \cdot g_R(\mathbf{q}_R^{(i)}) \cdot {}^{tool}\mathbf{T}_{task} = {}^{0}\mathbf{T}_{Pbase} \cdot g_P(\mathbf{q}_P^{(i)}) \cdot {}^{w}\mathbf{T}_{task}^{(i)}; \qquad i = 1,2,...n \tag{7} $$

describing relations between the actuated coordinates of the robot and the positioner, which ensure implementation of the given technological task. It is clear that because of the system redundancy, the above equations do not allow to obtain a unique solution for $\mathbf{q}_R$ and $\mathbf{q}_P$ corresponding to each task point. On the other hand, it provides us with some space for the robot/positioner motion optimization where the above equations are treated as the hard constraints.

3.3 Problem of robot/positioner motions planning

To utilize the redundancy in the best way, it is reasonable to partition the desired motion between the robotic manipulator and the positioner in such a way that the technological tool executes the given task as fast as possible and the robot and positioner movements are smooth enough. These objectives are obviously constrained by the actuator capacities that can be expressed as the maximum allowable joint velocities and accelerations.

To present this problem in a more formal way, let us introduce the functions $\mathbf{q}_R(t)$ and $\mathbf{q}_P(t)$ that describe the robot and positioner motions on the time interval $t \in [0,T]$. In addition, let us define the time instances $\{t_1, t_2, ... t_n\}$ corresponding to the cases where the robot end-effector visits the task frames defined by Eq. (1), where $t_1 = 0$, $t_n = T$. Using this notation, the problem can be presented as minimization of the total travelling time

$$ T \rightarrow \min_{\mathbf{q}_R(t),\, \mathbf{q}_P(t)} \tag{8} $$

over the set of continuous functions $\mathbf{q}_R(t)$ and $\mathbf{q}_P(t)$ subject to the equality constraints imposed by the prescribed task

$$ {}^{0}\mathbf{T}_{Rbase} \cdot g_R(\mathbf{q}_R(t_i)) \cdot {}^{tool}\mathbf{T}_{task} = {}^{0}\mathbf{T}_{Pbase} \cdot g_P(\mathbf{q}_P(t_i)) \cdot {}^{w}\mathbf{T}_{task}^{(i)}; i = 1,2,...n \tag{9} $$

and the inequality constraints describing the capacities of the robot/positioner actuators

$$ q_{Rj}^{\min} \le q_{Rj}(t_i) \le q_{Rj}^{\max} \qquad q_{Pj}^{\min} \le q_{Pj}(t_i) \le q_{Pj}^{\max} \tag{10a} $$

$$ \dot{q}_{Rj}^{\min} \le \dot{q}_{Rj}(t_i) \le \dot{q}_{Rj}^{\max} \qquad \dot{q}_{Pj}^{\min} \le \dot{q}_{Pj}(t_i) \le \dot{q}_{Pj}^{\max} \tag{10b} $$

$$ \ddot{q}_{Rj}^{\min} \le \ddot{q}_{Rj}(t_i) \le \ddot{q}_{Rj}^{\max} \qquad \ddot{q}_{Pj}^{\min} \le \ddot{q}_{Pj}(t_i) \le \ddot{q}_{Pj}^{\max} \tag{10c} $$

where j is the index for the robot and positioner joint variables: $j = 1,2,...6$ for the robot and $j = 1,2$ or $j = 1$ for the positioner.

Besides, the collision constraints that are extremely important for the considered technological application must be taken into account. They are defined as follows:

$$ cols(\mathbf{q}_P(t), \mathbf{q}_P(t)) = 0; \ \forall t \in [0,T] \tag{11} $$

where the binary function *cols*(.) verifies the intersections between the robotic system components (manipulator and positioner links, workpiece, fixture, etc.). It should be mentioned that the fiber placement head geometry may cause very restrictive collision constraints (see Fig. 2). To take into account this issue, additional verifications of intersection between the workpiece and the robot end-effector are required. In practice, relevant collision detection functions may be implemented using standard routines of commercially valuable industrial robotic CAD packages.

For this optimization problem, which aims at finding continuous functions $\mathbf{q}_R(t)$ and $\mathbf{q}_P(t)$ describing the robot manipulator and positioner motions, there is no known technique that can be straightforwardly applied to. The main difficulty here is related to the equality constraints that are written for the unknown time instants $t_1, t_2, ... t_n$. Besides, this problem is highly nonlinear and includes redundant variables. It should be mentioned that for the non-redundant case without the collision constraints the problem was solved by Bobrow *et al.* [42]. Another technique proposed in [18, 19] deals with the redundant case, but the collision constraints are verified at the post-optimization stage. For these reasons, this paper proposes a discrete optimization based methodology that is able to take into account simultaneously the system redundancy, the actuator capacities and the collision constraints.

4 Motion Planning Methodology

In order to solve the considered problem, a combinatorial optimization based methodology is proposed in this section. It includes three principal steps: a) search space discretization; b) path planning; c) motion smoothing. Using this methodology, the considered Cartesian task is represented as a directed multi-layer graph, which allows us easily applying the dynamic programming. To improve the computational efficiency, the optimization is performed in two stages combining "global rough search" and "iterative local fine search". Also, several strategies are proposed to adjust the trajectory to the requirements of the robot controller.

4.1 Search Space Discretization

For the considered robotic system, which includes a 6-axis robot and a one/two-axis positioner, there are exactly one or two redundant variables with respect to the given task (1) that is defined as a sequence of the robot end-effector locations in 6-dimensional space. First, let us present the one-axis positioner case and generalize it for the two-axis positioner further.

It is clear that for the one-axis positioner any of the manipulator joint variables included in the vector $\mathbf{q}_R$ can be treated as a redundant one, but it is more convenient to consider the positioner joint angle q_P as the redundant variable. This allows us, after solving the positioner direct kinematics for any given q_P , to apply straightforwardly the manipulator inverse kinematic function $g_R^{-1}(\mathbf{T},\mu)$ and find corresponding vectors $\mathbf{q}_R$ for the robot (which are not unique because of multiplicity of possible configurations defined by the parameter μ, see Section 3.2). This approach permits to take into account explicitly the equality constraints presented by Eq. (7) and substantially reduce the search space dimension.

To present the problem in a discrete way, let us sample the allowable domain of the redundant variable $q_P \in [q_P^{\min}, q_P^{\max}]$ with the step Δq_P

$$q_P^{(k)} = q_P^{\min} + \Delta q_P \cdot k; \quad k = 0, 1, \dots m \tag{12}$$

where $m = (q_P^{\max} - q_P^{\min})/\Delta q_P$ .Then, applying sequentially the positioner direct kinematics and the manipulator inverse kinematics in accordance with Eq. (9), one can get a set of possible configuration states for the robotic system. For the unique mapping from the task space to joint space, let us also take into account the configuration index μ that corresponds to the manipulator posture, and specifies the shoulder, elbow and wrist configurations. Then, the manipulator configuration corresponding to the given $q_P^{(k)}$ can be obtained as follows:

$$\mathbf{q}_R^{(k)}(t_i) = g_R^{-1}\left({}^{Rbase}\mathbf{T}_0 \cdot {}^{0}\mathbf{T}_{Pbase} \cdot g_P(q_P^{(k)}(t_i)) \cdot {}^{w}\mathbf{T}_{task}^{(i)} \cdot {}^{task}\mathbf{T}_{tool}, \mu \right); \quad \forall \mu;\ k = 0, 1, \dots m;\ \ i = 1, 2, \dots n; \tag{13}$$

where ${}^{Rbase}\mathbf{T}_0$ and ${}^{task}\mathbf{T}_{tool}$ denote the inverse of ${}^{0}\mathbf{T}_{Rbase}$ and ${}^{tool}\mathbf{T}_{task}$ respectively; t_i specifies the unknown time instant corresponding to the task location ${}^{w}\mathbf{T}_{task}^{(i)}$; the functions $g_P(.)$ and $g_R^{-1}(.)$ denote the positioner direct kinematics and the manipulator inverse kinematics. Therefore, for each task location we can generate a number of configuration states, i.e., ${}^{w}\mathbf{T}_{task}^{(i)} \rightarrow \mathbf{L}_{task}^{(k,i)}$; $\forall k, i$, where $\mathbf{L}_{task}^{(k,i)} = (\mathbf{q}_R^{(k)}(t_i), q_P^{(k)}(t_i))$ will be further referred to as the task location cell.

Taking into account that the task locations are strictly ordered in time, the original sequence of ${}^{w}\mathbf{T}_{task}^{(i)}$ described by Eq. (1) may be converted into the directed graph presented in Fig. 5. It should be noted that some of the configurations generated using Eq. (13) should be excluded from the graph because of the collision constraints (11) violation or exceeding of the actuator joint limits (10a). Besides, from an engineering point of view, it is prudent to avoid some configurations that are very close to the manipulator singular postures. These cases correspond to the "inadmissible nodes" in Fig. 5, which are not connected to any of the neighbors. It is clear that due to time-irreversibility, the allowable connections between the graph nodes are limited to the subsequent configuration states $\mathbf{L}_{task}^{(k',\,i)} \rightarrow \mathbf{L}_{task}^{(k'',\,i+1)}$ and the edge weights correspond to the minimum travelling time that are restricted by the maximum actuator velocities expressed by the constraints (10b).

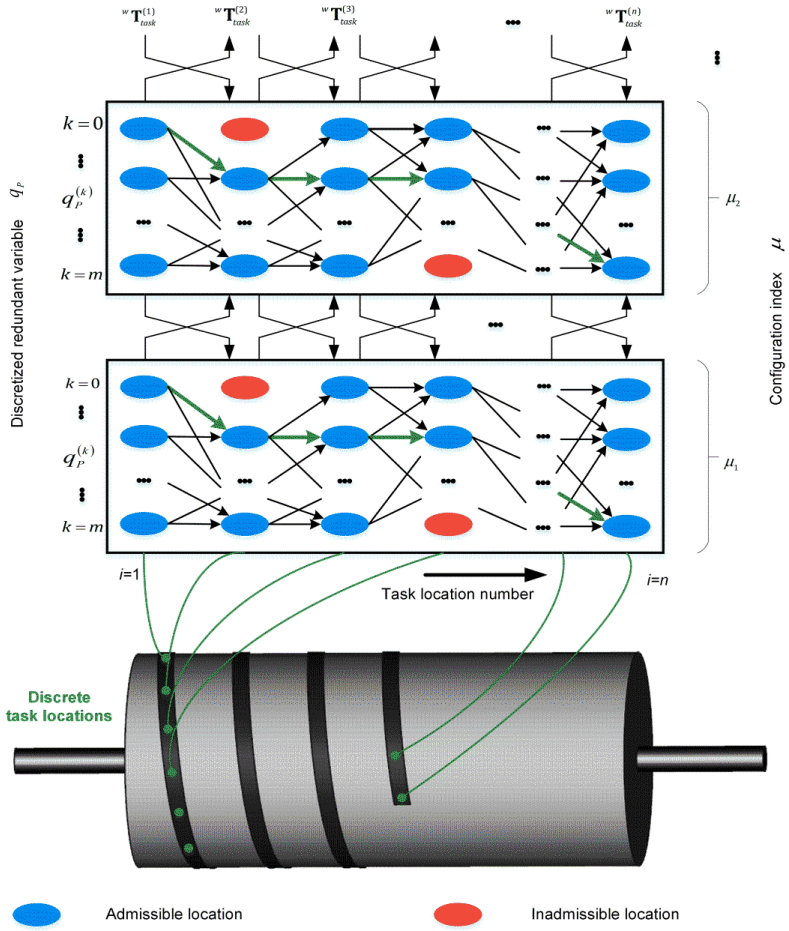

Fig. 5 Graph-based presentation of the discrete search space (for one-axis positioner)

An outline of the graph generation Algorithm 1 is presented below (the algorithm is written in generic mathematical language, but it has been tested using the Matlab and C++ environment, see Section 5). The algorithm input is a sequence of 4×4 location matrices $Task(i)$ corresponding to ${}^{w}\mathbf{T}_{task}^{(i)}$. The upper and lower limits of the redundant variable are defined as $q_P^{\max}$ and $q_P^{\min}$ respectively. The discretization density m determines the number of the discrete values of the redundant variable. The algorithm operates with the positioner direct kinematic function g_P (.) and the robot inverse kinematic function to transform the task locations $Task(i)$ into the joint space. The procedure is composed of two basic steps. The first step discretizes the redundant variable in the interval $[q_P^{\min}, q_P^{\max}]$ by implementing formula (12), and $m\times n$ matrix $\{q_P(k,i)\,|\,i=1,2,...n; k=1,2,...m\}$ is obtained. In the second step, the functions g_P (.) and g_R^{-1} (.) are applied sequentially, and the robot configuration states corresponding to $\{q_P(k,i)\,|\,\forall i;\forall k\}$ are computed. It should be mentioned that the configuration index μ here is treated as a constant input parameter, while in practice the step (2) should be re-executed for each possible μ. The sub-step (2a) checks the solution of the robot inverse kinematics for each $\{\mathbf{q}_R(k,i)\,|\,\forall i;\forall k\}$ and applies the collision test function for each configuration state $\{q_P(k,i),\mathbf{q}_R(k,i)\,|\,\forall i;\forall k\}$. Finally, the task graph is generated with nodes $L(k,i)=(q_P(k,i),\mathbf{q}_R(k,i))$. If a collision is detected or the function g_R^{-1} (.) returns the "*null*" denoting that the robot inverse kinematics is not solvable (because of the joint limits violation, etc.), the "*null*" value is assigned to the current node (it is marked as "inadmissible" one on Fig. 5).

Algorithm 1: ***Search space generation*** **G(.)**

Input: Task location matrices $\{Task(i) \mid i = 1,2,...n\}$
Upper limits of redundant variable $\{q_P^{\max}(i) \mid i = 1,2,...n\}$
Lower limits of redundant variable $\{q_P^{\min}(i) \mid i = 1,2,...n\}$
Discretization density m
Robot configuration index μ

Output: Matrix of locations $\{L(k,i) \mid k = 1,2,...m;\ i = 1,2,...n\}$

Notations: q_P, $\mathbf{q}_R$ - positioner and robot joint coordinates
T_P, T_R - Matrices of robot tool and positioner flange in local frames

Invoked functions: Robot inverse kinematics $g_R^{-1}(.)$ in local frame
Positioner direct kinematics $g_P(.)$ in local frame
Transformation from robot base to positioner base $Trans(.)$
Collision test function $cols(.)$

(1) *Redundant variable discretization*

For $i = 1$ to n **do**
  $\Delta q_P(i) := (q_P^{\max}(i) - q_P^{\min}(i))/(m-1)$;
  For $k = 1$ to m **do**
    $q_P(k,i) := q_P^{\min}(i) + (k-1) \cdot \Delta q_P(i)$;

(2) *Location matrix creation*

For $i = 1$ to n **do**
  For $k = 1$ to m **do**
    (a) $T_P := g_P(q_P(k,i)) \cdot Task(i)$;
      $T_R := Trans(T_P)$;
      $\mathbf{q}_R(k,i) := g_R^{-1}(T_R, \mu)$;

    (b) **If** $(\mathbf{q}_R(k,i) = null) \parallel (cols(q_P(k,i), \mathbf{q}_R(k,i)) = 1)$
      $L(k,i) := null$;
      else
      $L(k,i) := \{q_P(k,i), \mathbf{q}_R(k,i)\}$;

Using this presentation of the search space, the considered problem can be transformed to the searching of the shortest path on the directed graph shown in Fig. 5, where each column corresponds to the same task location ${}^{w}\mathbf{T}_{task}^{(i)}$ and each row represents the same value of the positioner joint coordinate $q_P^{(k)}$. In accordance with the physical sense, the initial and final path nodes must belong to the sets $\{\mathbf{L}_{task}^{(k_1,1)}, \forall k_1\}$ and $\{\mathbf{L}_{task}^{(k_n,n)}, \forall k_n\}$ respectively. In the frame of this notation, the desired solution can be represented as the sequence of the nodes

$$\{\mathbf{L}_{task}^{(k_1,1)}\} \rightarrow \{\mathbf{L}_{task}^{(k_2,2)}\} \rightarrow ... \{\mathbf{L}_{task}^{(k_n,n)}\} \tag{14}$$

corresponding to the robot and positioner configuration states $(\mathbf{q}_R^{(k)}(t_i), q_P^{(k)}(t_i))$.

In accordance with the actuator constraints (10b), the distance between subsequent graph nodes can be evaluated as the displacement time for the slowest joint

$$dist(\mathbf{L}_{task}^{(k_i,i)}, \mathbf{L}_{task}^{(k_{i+1},i+1)}) = \max_{j=0,..6} \left(\left| q_{j,i}^{(k_i)} - q_{j,i+1}^{(k_{i+1})} \right| \Big/ \dot{q}_j^{\max} \right) \tag{15}$$

where $j = 0$ corresponds to the positioner joint variable q_P and $j = 1,2,...6$ correspond to the joint coordinates of the robot. The latter allows us to present the objective function (total travelling time) as the sum of the edge weights

$$T = \sum_{i=1}^{n-1} dist(\mathbf{L}_{task}^{(k_i,i)}, \mathbf{L}_{task}^{(k_{i+1},i+1)}) \tag{16}$$

that depends on the indices $k_1, k_2, ... k_n$. It should be noted that the applied method of edge weights computing automatically takes into account the velocity constraints (10b), but the acceleration constraints must be examined for each considered path. It can be done by applying a formula

$$2\left|\Delta t_i(q_{j,i+1}^{(k_{i+1})}-q_{j,i}^{(k_i)})-\Delta t_{i+1}(q_{j,i}^{(k_i)}-q_{j,i-1}^{(k_{i-1})})\right|\Big/\Delta t_{i+1}\Delta t_i(\Delta t_{i+1}+\Delta t_i)\le \ddot{q}_j^{\max} \tag{17}$$

which is based on the second order approximation of the corresponding functions $\mathbf{q}_R(t)$ and $q_P(t)$ in the time domain, and where $\Delta t_{i+1}=dist(\mathbf{L}_{task}^{(k_i,i)},\mathbf{L}_{task}^{(k_{i+1},i+1)})$ and $\Delta t_i=dist(\mathbf{L}_{task}^{(k_i,i)},\mathbf{L}_{task}^{(k_{i-1},i-1)})$.

It is clear that for the one-axis positioner any of the manipulator joint variables included in the vector $\mathbf{q}_R$ can be treated as a redundant one, but it is more convenient to consider the positioner joint angle q_P as the redundant variable. This allows us, after solving the positioner direct kinematics for any given q_P, to apply straightforwardly the manipulator inverse kinematic function $g_R^{-1}(\mathbf{T},\mu)$ and find corresponding vectors $\mathbf{q}_R$ for the robot (which are not unique because of multiplicity of possible configurations defined by the parameter μ, see Section 3.2). This approach permits to take into account explicitly the equality constraints presented by Eq. (7) and substantially reduce the search space dimension.

For the case of a two-axis positioner, where the degree of redundancy is equal to two, the above presented methodology can be generalized in the following way. Similarly to the previous case, let us treat the robot joint angles $\mathbf{q}_R$ as non-redundant variables and the positioner joint angles $q_{P1}\in[q_{P1}^{\min},q_{P1}^{\max}]$ and $q_{P2}\in[q_{P2}^{\min},q_{P2}^{\max}]$ as the redundant ones which are sampled with the step Δq_{P1} and Δq_{P2} respectively

$$q_{P1}^{(k_1)}=q_{P1}^{\min}+\Delta q_{P1}\cdot k_1;\quad k_1=0,1,\dots m_1 \qquad q_{P2}^{(k_2)}=q_{P2}^{\min}+\Delta q_{P2}\cdot k_2;\quad k_2=0,1,\dots m_2 \tag{18}$$

where $m_1=(q_{P1}^{\max}-q_{P1}^{\min})/\Delta q_{P1}$ and $m_2=(q_{P2}^{\max}-q_{P2}^{\min})/\Delta q_{P2}$. Then, applying sequentially the positioner direct kinematic and robot inverse kinematic transformations in accordance with (9), a set of possible configuration states for each task location can be found. This allows us to obtain a 3D directed graph where each Cartesian task location is presented by a set of nodes located on the plane (see Fig. 6) in contrast to the one-axis positioner case where each task location corresponds to a column (see Fig. 5). Using this graph, the desired robot/positioner motion can be presented as a shortest path that sequentially connects the nodes belonging to the above mentioned planes (exactly one node for each plane). It is clear that the same methodology can be applied for the robot with extra degrees of freedom, mounted on a linear track for example. In the latter case the set of redundant variables includes both the positioner actuated coordinates and the additional coordinates of the robot. This allows us to utilize analytical expressions for the robot inverse kinematic transformations.

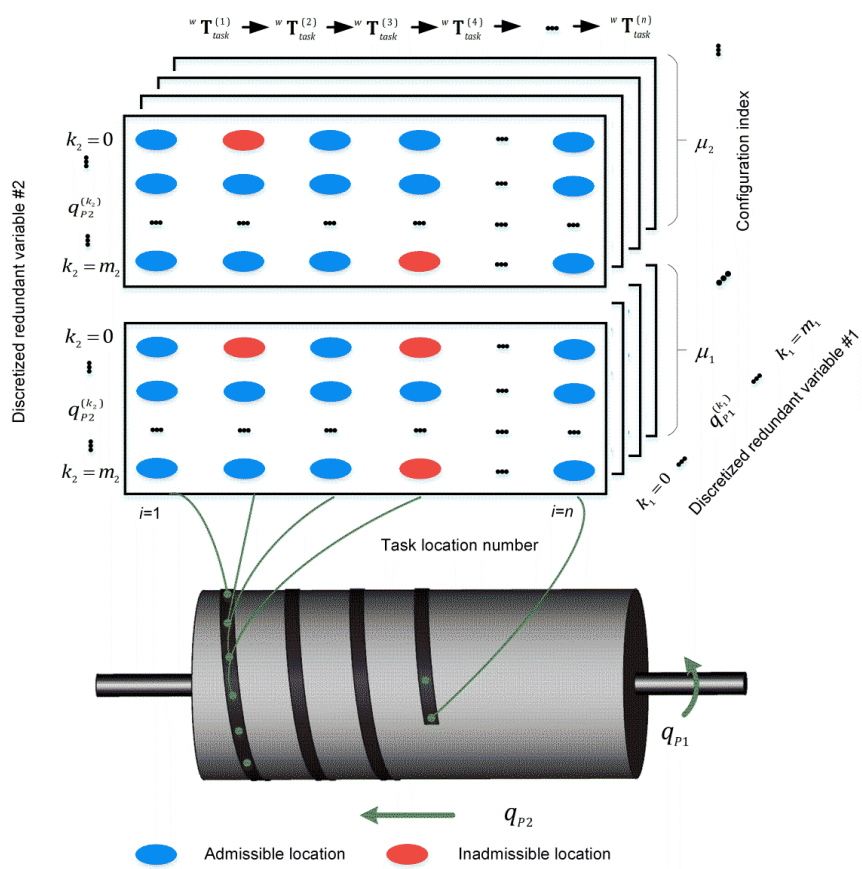

Fig. 6 Graph-based presentation of the discrete search space (for two-axis positioner)

4.2 Motion Planning Algorithm

After generation of the discrete search space, the original optimization problem (8)-(10) can be converted to a combinatorial one, which is aimed at searching of the shortest path connecting the subsets of the nodes corresponding to the given task locations. For computational convenience, these subsets are grouped in columns for the one-axis positioner (see Fig. 5) and in planes for the two-axis positioner (see Fig. 6). A conventional way to solve this discrete optimization problem is to use the classical shortest path search on the directed graph (e.g. Dijkstra, Belford, etc.) created by adding two virtual nodes to the graph-based search space at the beginning and the end [18-20]. This path should visit exactly one node in the candidates for each task location. However, as follows from the dedicated study, the classical techniques are extremely time-consuming, and they can be hardly accepted for real-life industrial applications. For example, it takes over 20 hours using Matlab (and about 2minutes in C++) to find desired solution even in the relatively simple case that deals with a two-axis robot and one-axis positioner, where the search space was built for 100 task points and the discretization step 1° (Win7SP1 with Intel® i5 @2.67 GHz 2.67 GHz). Besides, classical methods are not able to take into account the acceleration constraints expressed by inequalities (10c), which are important here because of technological requirements. For these reasons, a special problem-oriented algorithm is proposed below, which takes into account particularities of the graph describing the search space.

4.2.1 Motion generation using dynamic programming

To present the proposed algorithm for the general case, let us assume that the graph nodes corresponding to the same task points $\{{}^{w}\mathbf{T}_{task}^{(i)}\}$ are organized as one-dimensional arrays $\{\mathbf{L}_{task}^{(k,i)};\ i=const\}$ that are indexed by the superscript $k=0,1,...n$. Such presentation is straightforward for the one-axis positioner, and it can be easily obtained by simple node renumbering in the case of the two-axis positioner. This allows us to describe the algorithm in similar way independent on the number of the redundant variables.

The developed algorithm is based on the dynamic programming principle, which breaks down the full-size problem into a set of sub-problems of smaller sizes [43], aiming at finding the shortest path from the initial node set to the current node set. To present the basic idea, let us denote $d_{k,i}$ as the length of the shortest path connecting one of the initial nodes $\{\mathbf{L}_{task}^{(k_1,1)},\forall k_1\}$ to the current node $\{\mathbf{L}_{task}^{(k',i)}\}$. Then, taking into account the additivity of the objective (16), the shortest path for the nodes corresponding to the next task point $\{\mathbf{L}_{task}^{(ki+1)},\forall k\}$ can be found by combining the optimal solutions for the previous column $\{\mathbf{L}_{task}^{(k',i)},\forall k'\}$ and the distances between the nodes with the indices i and $i+1$.The latter corresponds to the formula

$$d_{k,i+1}=\min_{k'}\{d_{k',i}+dist(\mathbf{L}_{task}^{(k,i+1)},\mathbf{L}_{task}^{(k',i)})\}; \tag{19}$$

that is applied sequentially starting from the second task point, i.e. $i=1,2,...n-1$. Finally, after selection of the minimum length $d_{k,i+1}$ corresponding to the final task point and applying the backtracking, one can get the desired optimal path. Therefore, the desired solution for n sets $\{\mathbf{L}_{task}^{(k_1,1)}\}\rightarrow\{\mathbf{L}_{task}^{(k_2,2)}\}\rightarrow...\{\mathbf{L}_{task}^{(k_n,n)}\}$ is obtained by increasing sequentially the i-index, computing $d_{k,i+1}$, and finding the minimum value among $d_{k_n,n}$. The desired path is described by the recorded indices $\{k_1,k_2,...k_n\}$.

An outline of the proposed algorithm is presented below (see Algorithm 2). The input is the matrix of the locations $\{L(k,i)|\ i=1,2,...n;\ k=1,2,...m\}$, which contains information on the configuration states satisfying the equality constraint (9) and the collision constraint (11). The algorithm operates with two tables $D(k,i)$, $P(k,i)$ that include the minimum distances for the sub-problem of lower size (for the path $1\rightarrow i$) and the pointers to the previous locations respectively. The procedure is composed of four basic steps. The step (1) initializes the distance and pointer matrices by defining their first columns. In step (2), the recursive formula (19) is implemented. The computing starts from the second column and tries all possible connections between the nodes in the current column $\{L(k,i),\forall k\}$ and the previous one $\{L(j,i-1),\forall j\}$. For the admissible configuration states, the acceleration constraints are examined using the expression (17) for each candidate path connecting the nodes with the indices i, $i-1$ and $i-2$. In the algorithm, the acceleration constraint test is included in sub-step (2a) where $accl\ (\cdot)=0$ indicates that the current path satisfies this constraint, and it begins from the third column. It should be mentioned that the function $accl(.)$ requires three inputs $L(*,i)$, $L(*,i-1)$, $L(*,i-2)$, according to the expression (17), and the location $L(*,i-2)$ is determined using the pointer $P(j,i\text{-}1)$ to the previous location in the current path. Then, sub-step (2b) finds the minimum path from the current node $L(k,i)$ to the first column $\{L(j,1),\forall j\}$ and records the reference to $\{L(j,i-1),\forall j\}$ into the pointer matrix. In steps (3) and (4), the optimal solution is finally obtained and corresponding path is extracted by means of the backtracking.

Algorithm 2: ***DP-based Path planning*** **DP(.)**

Input: Matrix of locations $\{L(k,i) \mid i=1,2,...n;\ k=1,2,...m\}$

Output: Minimum path length $D_{\min}$

Optimal path indices $\{k^0(i) \mid i=1,2,...n\}$

Notations: $D(k,i)$, $P(k,i)$ - distance and pointer $m \times n$ matrices

Invoked functions: Distance between nodes $dist(L(k_1,i_1),L(k_2,i_2))$

Acceleration test for nodes $accl\big(L(k,i),L(j,i-1),L(P(j,i\text{-}1),i-2)\big)$

(1) **Set** $D(k,1):=0$; $P(k,1):=null$, $\forall k=1,2,...m$

(2) **For** $i=2$ to n **do**

For $k=1$ to m **do**

(a) **For** $j=1$ to m **do**

If $\big(L(k,i)\neq null\big)\&\big(L(j,i-1)\neq null\big)$

If $\big(i=2\big)\big\|\big(accl\big(L(k,i),L(j,i-1),L(P(j,i\text{-}1),i-2)\big)=0\big)$

$r(j):=D(j,i-1)+dist(L(k,i),L(j,i-1))$;

else

$r(j):=inf$;

(b) **Set** $D(k,i):=\min_j(\{r(j) \mid j=1,2,...m\})$;

$P(k,i):=\arg\min_j(\{r(j) \mid j=1,2,...m\})$;

(3) **Set** $D_{\min}:=\min_k(\{D(k,n) \mid k=1,2,...m\})$;

$k^0(n):=\arg\min_k(\{r(j) \mid j=1,2,...m\})$;

(4) **For** $i=n$ to 2

Set $k^0(i-1):=P(k^0(i),i)$

It is worth mentioning that the above presented algorithm is written in generic mathematical language, but it has been tested using the Matlab and C++ environment (see Section 5). As follows from a dedicated study, the proposed algorithm is rather efficient and suits industrial requirements in certain degree. In particular, it takes about 10 minutes using Matlab (and about 1 second using C++) to find the optimal solution for the above mentioned example (two-axis robot and one-axis positioner, 100 task locations, discretization step 1°). It should be stressed that in contrast to the conventional techniques, this algorithm can generate smooth trajectories by taking into account the acceleration constraints which are expressed by inequalities (10c). Nevertheless, since the inequalities (17) are checked for each candidate path during searching, the computation time may be still too high for real-life fiber placement applications where 6-axis robots are usually used and the number of the task points is essentially larger. This motivates us to improve the developed algorithm further.

4.2.2 Enhancement of the motion generation algorithm

To reduce the computing time, let us apply the following heuristic technique: solve the problem applying twice the same optimization routine (Algorithm 2) in different search spaces. It is reasonable to start with a rather big discretization step Δq_P (stage I), and further improve the initial solution in its neighborhood using relatively small Δq_P (stage II). In this technique, in spite of the repetition of the basic optimization routine, both stages operate with the search space of smaller size compared to the straightforward optimization presented above, which allows us essentially decreasing the amount of the computations. The details of this technique are presented in Fig. 7.

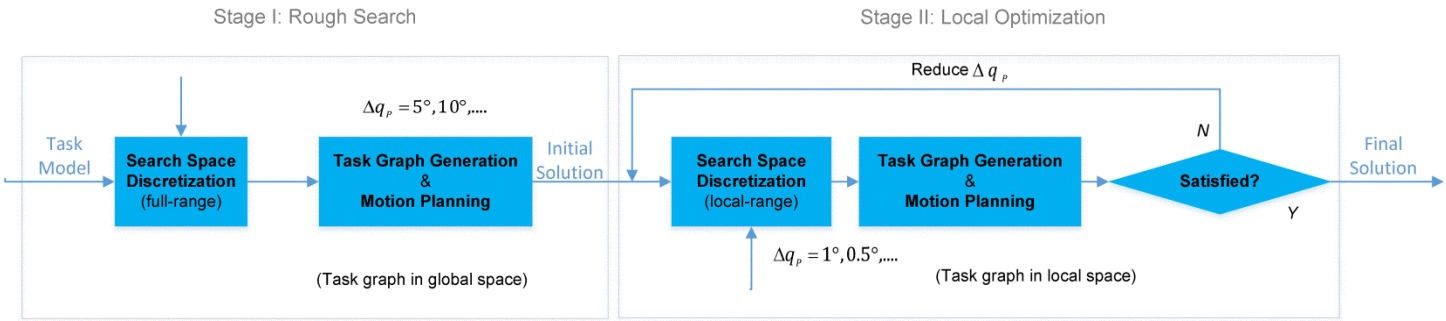

Fig. 7 Two-stage technique for robot and positioner motion generation

The stage I, named here as the *global rough search*, is based on the full-range discretization of the redundant variable with relatively big increment step (compared to that in the fine search algorithm presented before). By applying the dynamic programming principle expressed by (19), an initial solution can be obtained quite rapidly (it takes about one minute using Matlab for the above presented example that deals with the two-axis robot and one-axis positioner, 100 task locations). It should be mentioned that the motion obtained at this stage may be not satisfactory from the engineering point of view since the discretization step is large. So, it is reasonable to optimize the obtained solution further.

At the stage II (named as the *iterative local fine search*), a secondary discretization with smaller step is applied in the neighborhood of the initial solution obtained at the stage I. The neighborhood size is defined by the user (Fig. 8). It is clear that this local discretization produces the search space of relatively small size (compared to straightforward approach, see Algorithm 1) allowing us quickly generate an optimal solution that is obviously better than the initial one. For instance, for the above mentioned example, it takes about two minutes using Matlab in total (stages I and II) while the one-stage technique (see section 4.2.1) requires more than 10 minutes to obtain a similar result. It should be mentioned that the stage II can be repeated several times, sequentially reducing the discretization step.

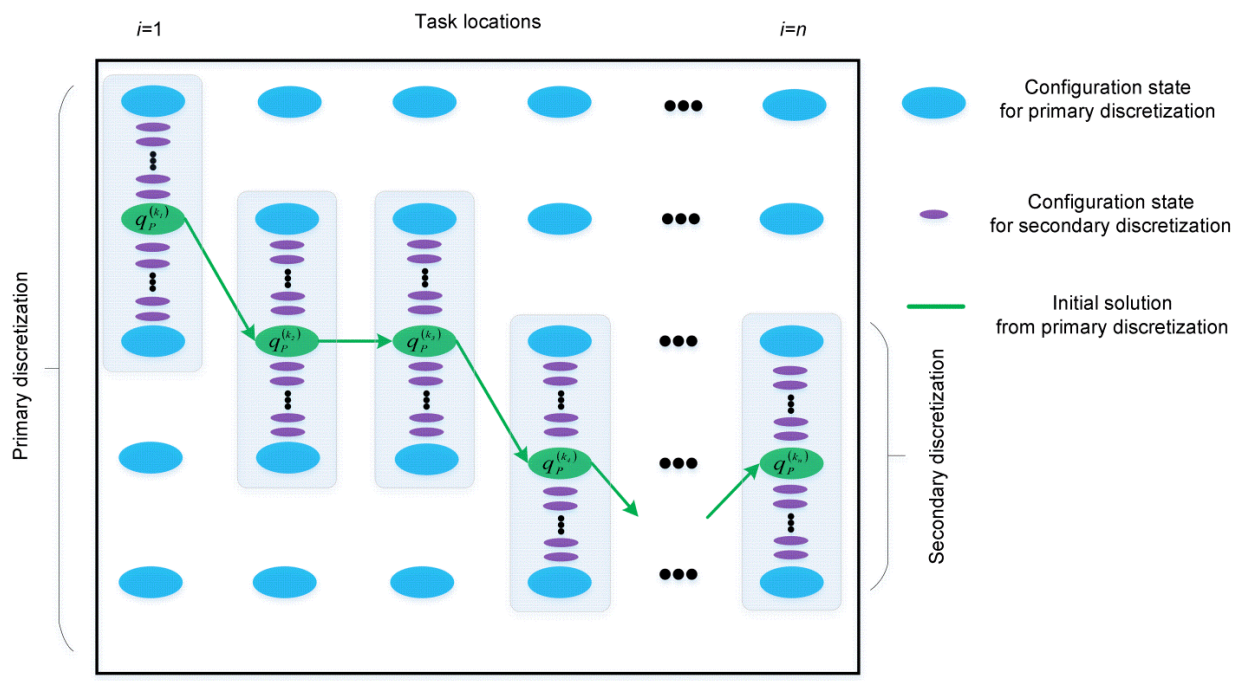

Fig. 8 Search space evolution for the two-stage algorithm

An outline of the two-stage algorithm is presented below (see Algorithm 3). The input includes the task location matrices $\{Task(i) \mid i = 1,2,...n\}$, the boundary limits of the redundant variable $q_P^{\max}$ and $q_P^{\min}$, the neighborhood size Rg_P for the stage II, and the discretization densities m and m' for both stages. It outputs the minimum path length $D'_{\min}$ and optimal path indices $\{k'^0(i) \mid i = 1,2,...n\}$. At the first stage, the $m \times n$ task graph $\{L(k,i) \mid k = 1,2,...m; i = 1,2,...n\}$ is generated using the full-range discretization with the density m (step Ia). The initial solution $\{k^0(i) \mid i = 1,2,...n\}$ is obtained by applying the path planning function $DP(.)$ based on algorithm 2 (steps Ib, Ic). At the second stage, the discretization domain is redefined (step IIa), and a

new graph $\{L'(k',i) \mid k'=1,2,...m'; i=1,2,...n\}$ of the size $m' \times n$ is generated in the neighborhood of the initial solution (step IIb). After applying again the planning function $DP(.)$ (step IIc), the optimal solution of the length $D'_{\min}$ and corresponding path indices $\{k'^0(i)\}$ are obtained. The optimal trajectory is extracted at the step (IId).

An alternative way to speed up the motion generation algorithm is to replace the stage II by simply smoothing the curve $q_P(t)$ corresponding to the redundant variable without changing the time instants obtained from the stage I (if it is acceptable from technological point of view). This can be achieved by applying the polynomial approximation to the function defined by the nodes $\{q_P(i), t(i) \mid i=1,2,...n\}$. From our experience, the 5th order polynomials are quite satisfactory for this procedure, which generates modified values of the redundant variable $\{q'_P(i) \mid i=1,2,...n\}$ ensuring better profile of the curve $q'_P(t)$. Then, in order to guarantee that the task locations are exactly visited by the end-effector, the robot joint coordinates should be also regenerated using the new values of the redundant variable $\{q'_P(i) \mid i=1,2,...n\}$ and sequentially reapplying the positioner direct kinematic and robot inverse kinematic transformations (see section 4.1). It is worth mentioning that the smoothing-based technique is very time efficient because it excludes the optimization at the stage II. However, in most cases studied by the authors the two-stage technique including the local optimization (see Fig. 6) provided better results.

Algorithm 3: Two-stage path planning

Input: Task location matrices $\{Task(i) \mid i=1,2,...n\}$
Upper limit of redundant variable $q_P^{\max}$
Lower limit of redundant variable $q_P^{\min}$
Neighborhood size Rg_P
Primary discretization density m
Secondary discretization density m'

Output: Minimum path length $D'_{\min}$
Optimal path indices $\{k'^0(i) \mid i=1,2,...n\}$

Notations: $\{L(k,i)\}$ - $m \times n$ matrix of locations for the first stage
$\{L'(k',i)\}$ - $m' \times n$ matrix of locations for the second stage

Invoked functions: Graph generation $G(.)$; (*algorithm 1*)
Path planning $DP(.)$; (*algorithm 2*)

Stage I:

(a) *Generate an initial graph using rough discretization in full domain of* q_P
Set $\{L(k,i) \mid i=1,2,...n; k=1,2,...m\} := G\left(\{Task(i) \mid i=1,2,...n\}, [q_P^{\min}, q_P^{\max}], m\right)$;

(b) *Apply algorithm DP*(.)
Set $\{\{k^0(i) \mid i=1,2,...n\}; D_{\min}\} := DP\left(\{L(k,i) \mid i=1,2,...n; k=1,2,...m\}\right)$;

(c) *Extract initial trajectory*
For $i=1$ to n **do**
$InitTraj(i) = L(k^0(i), i)$;

Stage II:

(a) *Redefine the discretization domain*
For $i=1$ to n **do**
$q'^{\min}_P(i) = \max\{(InitTraj(i).q_P - Rg_P), q_P^{\min}\}$;
$q'^{\max}_P(i) = \min\{(InitTraj(i).q_P + Rg_P), q_P^{\max}\}$;

(b) *Generate local graph using fine discretization in local domain of* q_P
Set $\{L'(k',i) \mid k'=1,2,...m'; i=1,2,...n\} = G\left(\{Task(i) \mid i=1,2,...n\}, \{[q'^{\min}_P(i), q'^{\max}_P(i)] \mid i=1,2,...n\}, m'\right)$;

(c) *Apply algorithm DP*(.) again
Set $\{\{k'^0(i) \mid i=1,2,...n\}; D'_{\min}\} := DP\left(\{L'(k',i) \mid i=1,2,...n; k'=1,2,...m'\}\right)$;

(d) *Extract optimal trajectory*
For $i=1$ to n **do**
$OptTraj(i) = L(k'^0(i), i)$;

4.3 Optimal Motion Implementation in Robotic System

The above presented algorithm allows user to find the optimal robot and positioner motion, which are presented as the sequence of the nodes defining the joint coordinates $\{\mathbf{q}_{Ri}, \mathbf{q}_{Pi} \mid i=1,2,...n\}$ corresponding to the time instants $\{t_i \mid i=1,2,...n\}$. It should be stressed that the time instants obtained from the motion planner are distributed irregularly in the interval $[0, t_n]$. For this reason, while implementing this motion in industrial environment, it is necessary to take into account specific

constraints imposed by the robot programming languages. In the simplest case, in the robot controller the desired motions can be described by a sequence of PTP or LIN commands. Their execution emploies real-time linear interpolation between the trajectory nodes in the joint or cartesian space. It is worth mentioning that here the travelling time is recomputed by the robot controller taking into account the velocity and acceleration constraints fixed in the software (so the new time intervals can be slightly different from those computed by the motion planner). The samples of relevant programs are presented below.

PTP node#1		LIN node#1
PTP node#2		LIN node#2
PTP node#3	*or*	LIN node#3
…		…
PTP node#n		LIN node#n

Fig. 9 Continuous trajectory implementation using standard PTP and LIN sequences

However, in the robotic controller, standard PTP and LIN trajectory genereation algorithms assume that each arc connecting adjacent nodes includes three segments (accelerating, constant speed and decelerating) and the speed at the arc ends is equal to zero. It is clear that such "start-stop" implementation is not accepetable for the considered fiber placement technology where the end-effector Cartesian speed should be as high as possible and the motion stops at the nodes are not permited. To avoid these stops, robot programming languges usually include motion commands with approximate positioning at the end points that are based on the overlapping of deceleration and acceleration segments for the subsequent arcs (so-called "blended" motions). For example, in KUKA programming language [44], for the PTP-PTP sequence the commands can include the C_PTP suffix allowing to approximate the end point and to ensure the overlapping producing continuous motion . It is clear that for our application the complete 100% overlapping is required which is achived by setting relevant system variable $APO.CPTP=100. In contrast, for the LIN-LIN sequence, the approximate positionning has several options and the motion command may include the suffixes C_DIS, C_ORI or C_VEL defing the trajectory point where the overlapping begins. They correspond to the switching criteria based on the translational / orientational distance to the target node and the velocity level at the beginning of the approximation. Similar to the previous case, the complete overlapping of the deceleration and acceleration segments is achieved by setting the system variable $APO.CVEL=100. This allows user to impelement non-stop continuous motions for which the end-effector passes the nodes neighbourhood only instead of visiting the nodes exactly, which can be unacceptable in some cases. It is worth mentionning that using the fine discretization in the motion planner (see Section 4.2.2) is not very helpful for the robot programming via the PTP-PTP or LIN-LIN sequences because increasing of the node number does not influence essentially on the positioning accuracy (in fact, only the node neighbouhood is visited). Besides, implementation of very short arcs connecting the nodes is rather difficult for the robot controller that may be not capable to ensure the desired Cartesian speed because of controller constraints on minimum duration of each trajectory segment. The samples of relevant programs are presented below.

$APO.CPTP=100			$APO.CVEL=100	
PTP node#1	C_PTP		LIN node#1	C_VEL
PTP node#2	C_PTP		LIN node#2	C_VEL
PTP node#3	C_PTP	*or*	LIN node#3	C_VEL
…			…	
PTP node#n-1	C_PTP		LIN node#n-1	C_VEL
PTP node#n			LIN node#n	

Fig. 10 Continuous trajectory implementation using approximated PTP-PTP and LIN-LIN sequences

An alternative way to implenment the desired motion is based on using of the spline commands. Their execution employ the high-order polynomial interpolation between the trajectory nodes, instead of the first-order one that is used for LIN or PTP. In KUKA programming language [44], the spline motion can be defined both in the Cartesain and joint spaces. Compared to the approximated LIN-LIN and PTP-PTP sequences, here the end-effector exactly passes the given nodes on the fiber placement path without stops. In addition, the desired velocity for each segment (or the travelling time between the nodes) can be also defined. In order to program the spline motion, the sequence of the nodes must be grouped inside of a relevant program block. In KUKA language for example, the spline block is limited by the SPLINE and ENDSPLINE statements and contains a sequence of commands SPL describing the trajectory nodes and corresponding velocities (or the time intervals). The latter is achieved by setting the system variable $VEL.CP or using special TIME_BLOCK containing the travelling times between the nodes. A sample of relevant program is presented below.

```
SPLINE
    SPL node#1 WITH $VEL.CP = v1
    SPL node#2 WITH $VEL.CP = v2
    SPL node#3 WITH $VEL.CP = v3
    …
    SPL node#n WITH $VEL.CP = vn
ENDSPLINE
```

Fig. 11 Continuous trajectory implementation using spline motions with assigned velocities

Comparison of the above presented approaches for the robot motion implementation is given in Table 1. It takes into account both the positioning accuracy (with respect to the trajectory nodes) and the end-effector speed defined via the time setting options. It should be noticed that, similar to PTP-PTP and LIN-LIN sequences, the spline motion is also rather sensitive to the inter-node travelling times directly related to the path sampling step used in the task model (see Section 3.1). In fact, if the discretized path segments are too short, it is impossible for the robot controller to apply the polynomial interpolation correctly because of hardware constraint on the minimum travelling time between the nodes. The latter may lead to the reduction of the real Cartesian speed that can be lower than the required one. So, before implementing the desired motion in the robot programming language, it is reasonable to regenerate the obtained trajectory by redefining the nodes to be sure that all inter-node traveling times are big enough and perfectly suit the controller capacities.

Table 1 Comparison for the three types of robot motion programming

	PTP / LIN	PTP-PTP / LIN-LIN (approximate positioning)	SPL
Positioning	Exact	Approximate	Exact
Movement	Start-Stop	Continuous	Continuous
Time setting	No	No	Yes
	Inappropriate	Inappropriate	Appropriate

To adjust the trajectory to the controller requirements, several strategies can be applied. The simplest one (i) is based on straightforward elimination of some nodes located too close at the time axis (it is worth reminding that in the Cartesian space the nodes are located at equal distance, but the speed varies). The second strategy (ii) ensures the node location regularization in time, i.e. shifting the nodes along the given fiber placement path in such a way that all inter-node travelling times are equal to each other, but big enough to be implemented by the controller. To achieve this goal, the spline interpolation can be applied to both the end-effector path and the redundant variable profile $q_p(t)$. Then, in order to guarantee that new task locations are exactly visited by the end-effector, the robot joint coordinates should be regenerated by sequentially reapplying the positioner direct kinematic and robot inverse kinematic transformations (see section 4.1). It is also possible to apply the third strategy (iii) where the profiles of the redundant and non-redundant variables are regenerated in the same way, without straightforward modification of the path nodes in the Cartesian space. In practice, an appropriate strategy is selected by the user who must also verify the velocity and acceleration constraints and slightly increase the inter-node times if the constraints are violated.

Hence, sequentially applying algorithms and strategies presented in this section, user can generate the optimal robot and positioner motions, adjust them to the capabilities of the robot controller and to describe the obtained motions in the robot programming language. The following section presents a case study justifying the developed methodology.

5 Application Example

To demonstrate the efficiency of the proposed methodology, let us apply it to a redundant robotic system composed of a two-axis planar manipulator and a one-axis positioner. The geometrical parameters of the system are given in Fig. 12. This system possesses a 1-dof redundancy with respect to the planar task. The ellipsis-type fiber placement path was discretized in 100 segments uniformly. The corresponding discrete path points are denoted as $\mathbf{p}_i = (x_i, y_i)^T$ where $i = 1,2,...100$. These Cartesian positions should be visited sequentially by the end-effector in minimum time by using in the best way the motion capabilities of both the robot and the positioner. The latter are limited by the actuator constraints presented in Table 2. The algorithm programming environments are Visual C++ and Matlab in Win7SP1 with Intel® i5 @2.67 GHz 2.67 GHz. To simplify

the algorithm testing, the computing times in following text are all referred to that in Matlab. However, to show the computational limits for different case studies, the final table includes results obtained using both Matlab and C++.

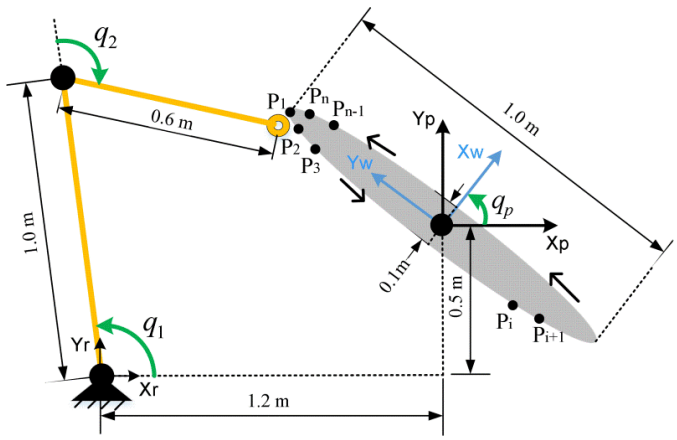

Fig. 12 Planar redundant robotic system

Table 2 Actuator constraints for the robotic system

Joint coordinates	*Velocity*	*Acceleration*
$-180° \le q_P \le 180°$	$-48°/s \le \dot{q}_P \le 48°/s$	$-192°/s^2 \le \ddot{q}_P \le 192°/s^2$
$-180° \le q_{R1} \le 180°$	$-12°/s \le \dot{q}_{R1} \le 12°/s$	$-48°/s^2 \le \ddot{q}_{R1} \le 48°/s^2$
$-180° \le q_{R2} \le 180°$	$-30°/s \le \dot{q}_{R2} \le 30°/s$	$-120°/s^2 \le \ddot{q}_{R2} \le 120°/s^2$

In order to employ the proposed motion planning algorithm, the positioner joint coordinate q_P was treated as the redundant variable and firstly was discretized in the interval $[-180°, 180°]$ with the step $\Delta q_P = 3°$ (for the stage I) . Then, by sequentially applying the positioner direct kinematic and robot inverse kinematic transformations, a column of 121 candidate configuration states was obtained for each of 100 path points. To ensure that no collision happens, each candidate configuration state was verified. For this operation, the robot links and the path segments were presented as short straight lines, and the intersections of the robot links and path segments were checked. If the intersection was detected, the current configuration state was marked as "inadmissible" one. Besides, similar mark was used for the nodes for which the robot inverse kinematics is insolvable. After this procedure, the original sequence of the task points $\mathbf{p}_i$ was converted into the task graph composed of 12100 nodes containing the joint coordinates of the robot and positioner. The obtained graph was used for the stage I of the optimization algorithm. The computing time for the graph generation was 43 seconds.

At the stage I, after applying the developed DP algorithm to this graph, the initial solution was found with the total travelling time of 8.8 sec (computing time for the path planning is about 98 sec). Relevant results are presented in Fig. 13. As follows from them, the joint coordinate profiles are quite smooth but the velocity profiles are very disturbed. It was surely caused by the relatively large discretization step $\Delta q_P = 3°$ used for the task graph generation at the stage I.

At the stage II, to improve the above results, the task graph was regenerated in the neighborhood of the initial trajectories with smaller discretization step $\Delta q'_P = 1°$. The neighborhood size for the redundant variable was defined as $\pm 30°$, the corresponding task graph was generated in 30 sec and was composed of 6100 nodes. It took 52 sec for the DP algorithm to find a better solution. To enhance the results further, the stage II was repeated for the discretization step $\Delta q''_P = 0.5°$ and the neighborhood size $\pm 15°$. The optimization results obtained after these two iterations are presented in Fig. 14. As follows from them, the local optimization allowed us to reduce the total travelling time from 8.8 to 7.0 sec (i.e. by 20% roughly). However, because of the discretization, there is still some non-smoothness in the velocity profiles that is undesirable from the engineering point of view [45]. To adjust the optimization results to the engineering requirements, the obtained trajectories were smoothed using the spline approximation of the fifth order. Relevant profiles for the joint coordinates and velocities are presented in Fig. 15.

For comparison purposes, the considered problem was also solved in the straightforward way, i.e. by applying the developed DP algorithm to the fine task graph obtained by the global discretization of the redundant variable in the full interval $[-180°, 180°]$ with the small step $\Delta q_P = 0.5°$. Corresponding task graph was composed of 72100 nodes. In this case, it took about 40 min to find the optimal trajectories with similar total travelling time of 7.0 sec. In contrast, the proposed method

requires less than 5 min to go through all steps and procedures (graph generation/regeneration and iterative motion planning). More details concerning computational efficiency of the known and proposed algorithms are given in Table 3, which confirms the advantages of the developed technique.

Table 3 Comparison for the known and proposed algorithms

	Classic algorithm (Dijkstra shortest path)	One-stage DP algorithm (global fine search)	Two-stage DP algorithm (global rough search + local fine search)
$\Delta q_P = 3°$	T= 7.83 sec Without acceleration constr. Computation: MATLAB: 3.2 min / C++: 0.3 s	T= 8.57 sec With acceleration constr. Computation: MATLAB: 2.4 min / C++: 0.3 s	T= 8.57 sec With acceleration constr. Computation: MATLAB: 2.4 min / C++: 0.3 s (as rough solution from stage I)
$\Delta q_P = 1°$	T= 5.94 sec Without acceleration constr. Computation: MATLAB: 20.9 h / C++: 2 min	T = 7.59 sec With acceleration constr. Computation: MATLAB: 15.5 min / C++: 1.4 s	T = 7.94 sec With acceleration constr. + Computation: MATLAB: 85 s / C++: 0.1 s (stage II-local size:60°)
$\Delta q_P = 0.5°$	T= 5.79 sec Without acceleration constr. Computation: MATLAB: 350 days*/ C++: 10 h*	T = 7.04 sec With acceleration constr. Computation: MATLAB: 1 h / C++: 4.3 s	T = 7.07 sec With acceleration constr. + Computation: MATLAB: 85 s / C++: 0.1 s (stage II-local size:30°)
$\Delta q_P = 0.3°$	-	T = 7.01 sec With acceleration constr. Computation: MATLAB: 2.6 h / C++: 11.7 s	-
Trajectory property	Unsmooth	Smooth	Smooth

* value is estimated

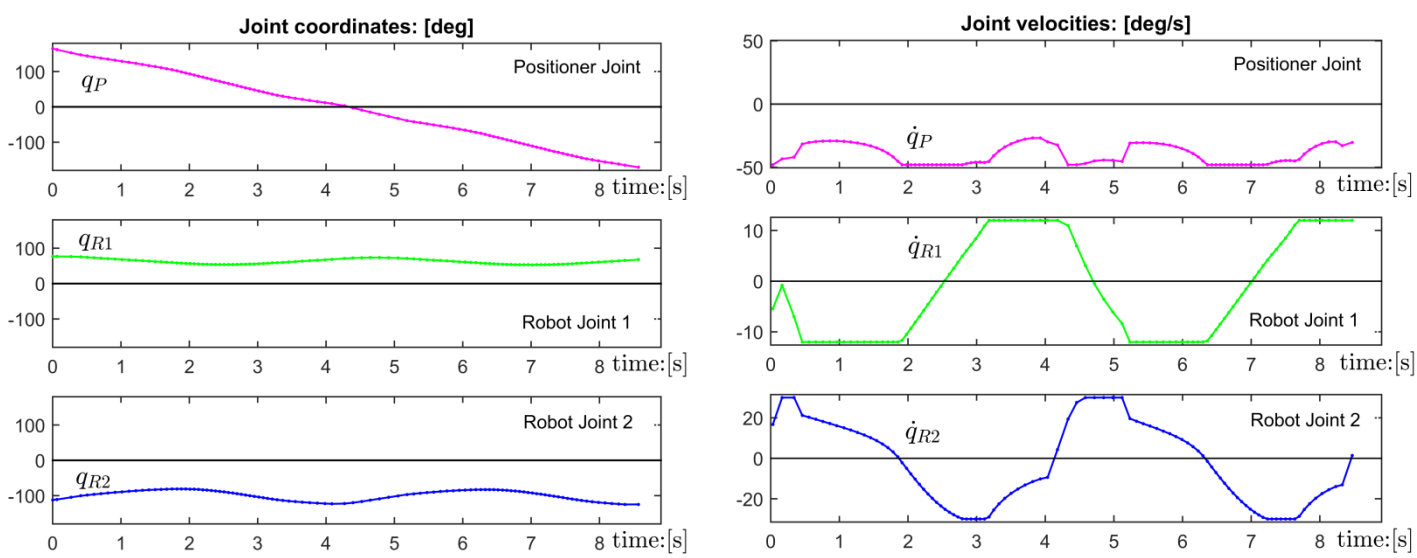

Fig. 13 Joint coordinate and velocity profiles after the Stage I (T = 8.57 s)

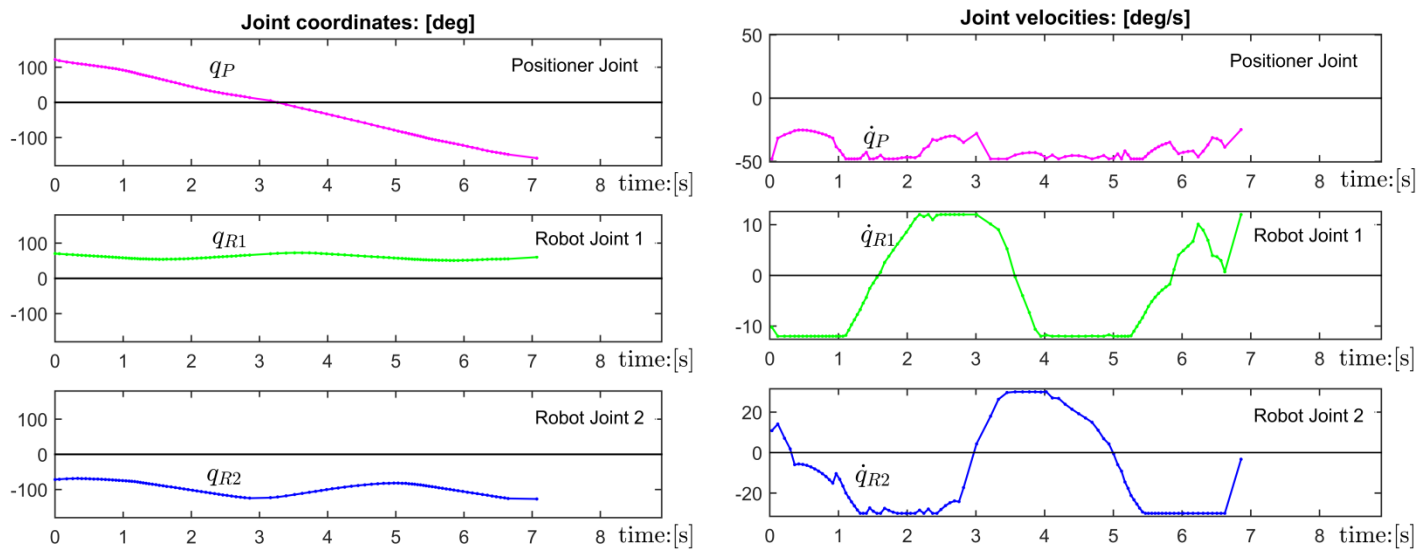

Fig. 14 Joint coordinate and velocity profiles after the Stage II (T = 7.07 s)

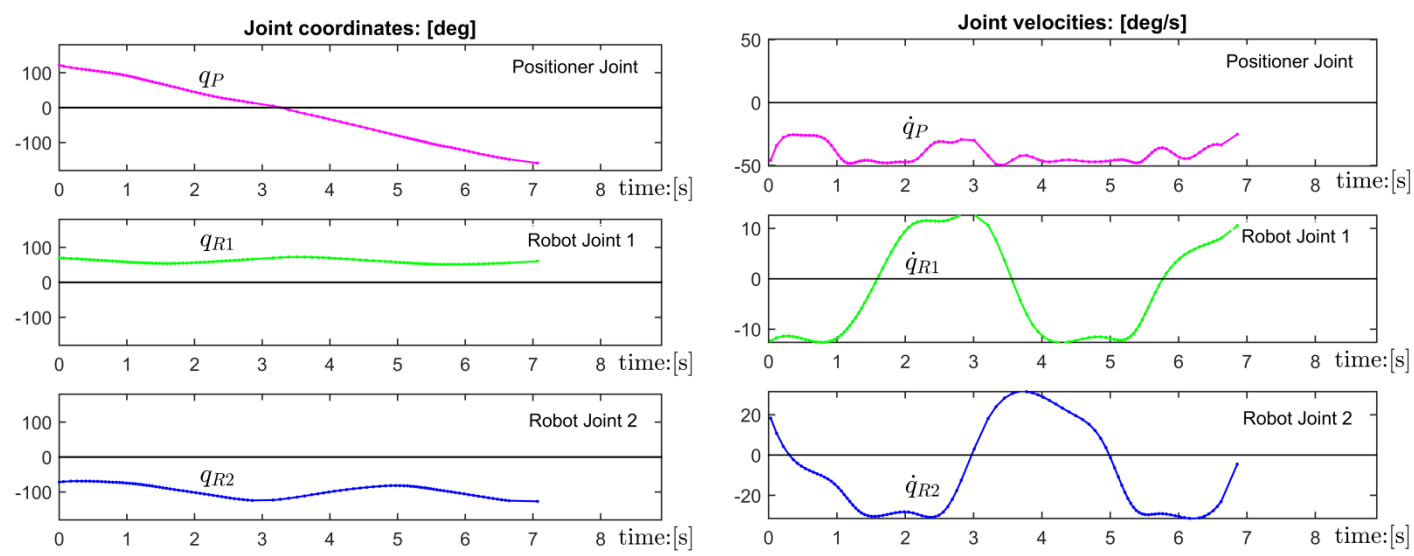

Fig. 15 Joint coordinate and velocity profiles after the smoothing (T = 7.07 s)

It is worth mentioning that the above presented example demonstrates the efficiency of the developed algorithms for a relatively simple case study (two-axis robot and one-axis positioner), but the developed algorithms were also successfully applied for real-life industrial problems that deals with automated fiber placement using a 6-axis robot and an one-axis positioner. The optimal motions generated by the developed algorithm were recorded in a form of data table where the trajectories were presented as a sequence of joint coordinates of all actuators. The obtained table was loaded into simulation software (CATIA package) in order to debug the robot program before implementation on the factory floor. This software offers a 3D graphical simulation environment with visual programming tool and flexibilities in robot selection, robot placement, motion simulation, etc. For the considered fiber placement application, corresponding workcell is presented in Fig. 16. It includes a one-axis actuated positioner and a 6-axis industrial robot equipped with a fiber placement head. The robot and positioner motions were programmed using SPL commands. After simulating and debugging, the trajectories obtained using the developed algorithm are ready for practical implementation.

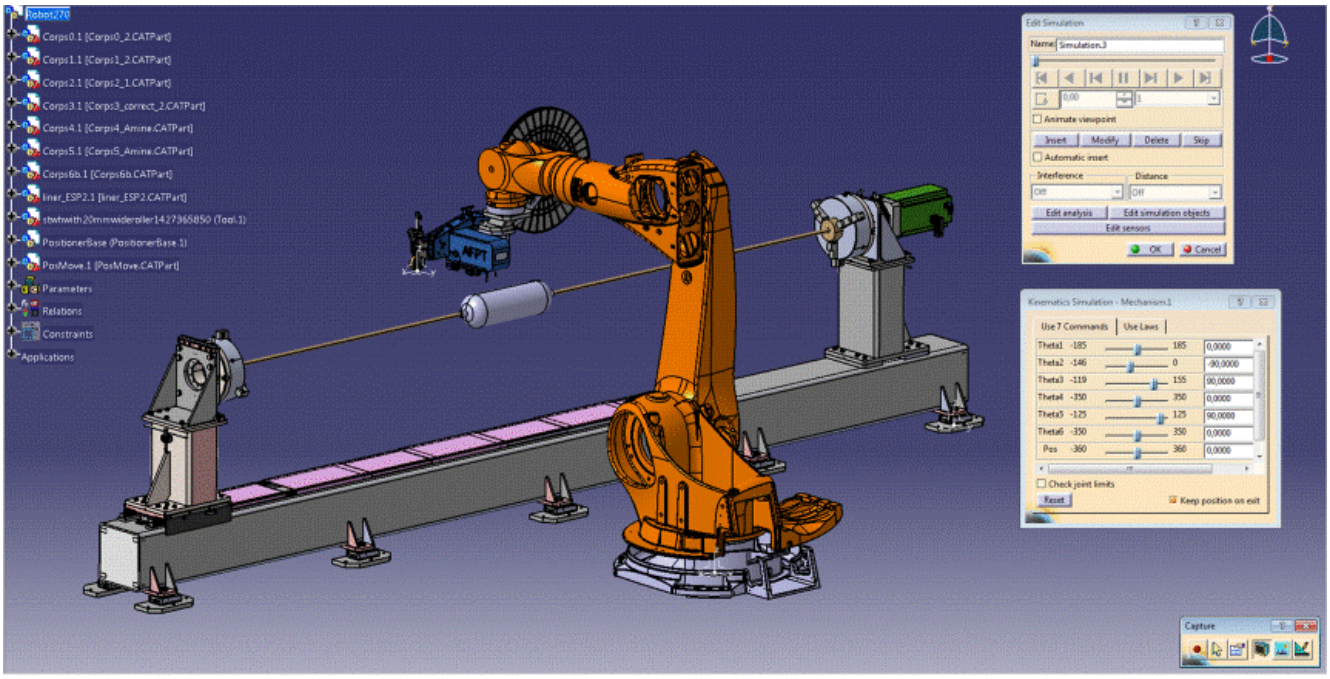

Fig. 16 Verification of the robot/positioner motions in CAD environment

6 Discussions

In spite of the obvious advantages of the proposed technique (high computational efficiency, capability to take into account both the velocity and acceleration constraints), there are a number of limitations and open questions to be discussed. The first of them is related to the selection of the discretization step for the redundant variables. It is clear that a smaller discretization step may provide faster and smoother motions but the relevant computation cost is extremely high. On the other side, excessive discretization does not yield significant improvement of the primary objective that is here the minimization of the total motion time. For instance, in the above presented example, reduction of the discretization step from $\Delta q_P = 0.5°$ to $\Delta q_P = 0.3°$ gives almost negligible improvement of the total motion time (by 0.4% only, from 7.04 to 7.01 sec, see Table 2). At the same time, it requires 3.9 hours of computations instead of 40 min for $\Delta q_P = 0.5°$. Hence, for our case study, the final discretization step around 0.5° is sufficient to obtain a suitable solution in acceptable time. However, in general case, it is reasonable to repeat Stage II (local optimization) several times in order to verify that more fine-grained discretization of the redundant variables is useless.

Another issue to be discussed is related to the sampling of the fiber placement path (i.e. the task discretization). Since the original task is defined as a continuous Cartesian curve, it seems reasonable to describe it using relatively high number of discrete points. However, the excessive sampling of the path leads to essential computational efforts but it provides just a slight benefit for the fiber placement quality, since relevant technology is not extremely sensitive to the positional accuracy of the robot end-effector. In particular, some positional errors can be compensated by the end-effector that provides pressure for the fiber compacting. Besides, this technology requires some overlaying of the successful fiber coats, and the overlaying degree is not very critical here. From our experience, the fiber placement path discretization with 100 to 200 nodes is quite sufficient for the above presented example. But this number should be certainly increased for complicated objects.

It is worth also mentioning that there are two discretization procedures here: 1) Discretization of the given fiber placement path across the workpiece surface (performed in the task space); 2) discretization of the redundant variable (performed in the joint space). It is clear that both of them can be either uniform or non-uniform. For the discretization of the fiber placement path, the simplest way is to maintain the uniform distance between the subsequent task locations. However, in some cases it is reasonable to perform sampling using constant increments of the polar angle related to the workpiece geometric model (which obviously leads to non-uniform path discretization with respect to the distances between the nodes). The latter approach is reasonable for the workpieces with non-circular cross-section, where it allows generating smoother motion at the curved segments with high density of task locations. However, in some cases, it may cause too excessive discretization that does not provide reasonable benefit for the fiber placement quality, because extremely short segment cannot be implemented by the robot controller with the desired speed (since in practice there is hardware constraint on the minimum travelling time

between the nodes). For the redundant variable(s), which in this study are related to the positioner, it is reasonable to apply the uniform discretization in the corresponding joint space. It should be also noted that an alternative way in selection of redundant variable(s) is not reasonable because of essential computational difficulties in inverse kinematic transformations.

Also, some carefulness is required in application of the smoothing technique applied after the motion planning (DP algorithm). It is aimed at the elimination the oscillations in derivatives (see Fig. 14) and is based on the approximation that can be applied either to each joint trajectory independently or to the redundant variable only. It is clear that the approximation causes some positional deviations from the given fiber placement path. However, such small deviations are usually admissible for the considered technology because of the end-effector pressure and the fiber overlaying. Besides, in practice, only part of the nodes is used in robot programming while intermediate segments (between subsequent nodes) are generated online using the spline interpolation. It is clear that the latter provides some smoothing but also introduces some deviations from the given fiber placement path. It is worth mentioning that in reality some additional smoothing is provided by actuators and mechanical inertia of robot/positioner components. So, from engineering point of view, it is not reasonable to solve the optimization problem presented in section 3.3 with extremely high accuracy and to pay for this by huge computational expenses, while knowing that some small errors introduced at the implementation stage are ignored in practice. Nevertheless, the solution produced by the developed motion generator should be carefully verified via simulation, i.e. the obtained path must be compared with the given one.

7 Conclusions

This paper proposes a new methodology to optimize the robot and positioner motions in redundant robotic system for the fiber placement process. In contrast to the previous works, the proposed approach allows using full capacities of the robotic system expressed by the maximum velocities and accelerations in the actuated joints. It generates the time-optimal smooth trajectories with high computational efficiency and also takes into account the collision constraints.

The core of the developed methodology is in conversion of the original continuous problem into a discrete one, which allows describing all possible motions of the robot and the positioner in combinatorial way and considering the robotic system redundancy. Owing to this approach, the considered Cartesian task is represented as a directed multi-layer graph in the discretized joint space, which allows us easily applying the dynamic programming technique in order to find the desired motion for both the robot and positioner. To improve further the computational efficiency, the methodology is enhanced by applying the two-stage optimization strategy. It uses the heuristic idea combining the “global rough search” at the initial stage and the “iterative local fine search” at the final search. This combination provides significant reduction of the computing time for the time-optimal motion planning. To adjust the optimization results to the engineering requirements, the obtained trajectories are smoothed using the spline approximation. Also, several strategies are proposed to adjust the trajectory to the requirements of the robot controller.

To confirm the advantages of the developed methodology, the paper includes an application example that deals with a planar fiber placement robotic system. Current work deals with industrial implementation of the developed algorithms. In future, the developed approach will be generalized and applied to wider class of redundant robotic systems which include two-axis positioner and workspace extension units.

Acknowledgments

This work was supported by the China Scholarship Council [grant numbers 201404490018]. The authors also acknowledged CETIM for the motivation of this research work.

References

[1] D. Gay, Composite materials: design and applications, CRC press, 2014.
[2] G. Marsh, Automating aerospace composites production with fibre placement, REINFORCED plastics, 55 (2011) 32-37.
[3] S.T. Peters, Handbook of composites, Springer Science & Business Media, 2013.
[4] A.K. Kaw, Mechanics of composite materials, CRC press, 2005.

[5] L. Li, D. Xu, X. Wang, M. Tan, A survey on path planning algorithms in robotic fibre placement, Control and Decision Conference (CCDC), 2015 27th Chinese, IEEE, 2015, pp. 4704-4709.
[6] H.-J.L. Dirk, C. Ward, K.D. Potter, The engineering aspects of automated prepreg layup: History, present and future, Composites Part B: Engineering, 43 (2012) 997-1009.
[7] D. Rabeneck, Boosting composite production, aerotec. The engineering and technology magazine for the aerospace industry, (2010) 28-30.
[8] A. Özer, S.E. Semercigil, An event-based vibration control for a two-link flexible robotic arm: Numerical and experimental observations, Journal of Sound and Vibration, 313 (2008) 375-394.
[9] C. GALLET-HAMLYN, Multiple-use robots for composite part manufacturing, JEC composites, (2011) 28-30.
[10] C. Krombholz, M. Perner, M. Bock, D. Röstermundt, Improving the production quality of the advanced automated fiber placement process by means of online path correction, 28th Congress of the International Council of the Aaeronautical Sciences, 2012, pp. 1-10.
[11] B. Shirinzadeh, G. Cassidy, D. Oetomo, G. Alici, M.H. Ang Jr, Trajectory generation for open-contoured structures in robotic fibre placement, Robotics and Computer-Integrated Manufacturing, 23 (2007) 380-394.
[12] K. Kazerounian, A. Nedungadi, Redundancy resolution of serial manipulators based on robot dynamics, Mechanism and machine theory, 23 (1988) 295-303.
[13] F. Flacco, A. De Luca, Discrete-time redundancy resolution at the velocity level with acceleration/torque optimization properties, Robotics and Autonomous Systems, 70 (2015) 191-201.
[14] L. Wu, K. Cui, S.-B. Chen, Redundancy coordination of multiple robotic devices for welding through genetic algorithm, Robotica, 18 (2000) 669-676.
[15] S.R. Buss, Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods, IEEE Journal of Robotics and Automation, 17 (2004) 16.
[16] E. Tabarah, B. Benhabib, R.G. Fenton, Optimal motion coordination of two robots—a polynomial parameterization approach to trajectory resolution, Journal of robotic systems, 11 (1994) 615-629.
[17] Y. Gan, X. Dai, D. Li, Off-Line Programming Techniques for Multirobot Cooperation System, International Journal of Advanced Robotic Systems, 10 (2013).
[18] L.B. Gueta, R. Chiba, J. Ota, T. Ueyama, T. Arai, Coordinated motion control of a robot arm and a positioning table with arrangement of multiple goals, Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, IEEE, 2008, pp. 2252-2258.
[19] L.B. Gueta, R. Chiba, T. Arai, T. Ueyama, J. Ota, Hybrid design for multiple-goal task realization of robot arm with rotating table, Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, IEEE, 2009, pp. 1279-1284.
[20] A. Dolgui, A. Pashkevich, Enhanced Optimisation Techniques for Off-line Programming of Laser Cutting Robots, Research Report G2I-EMSE 2007-500-015, November 2006, Ecole des Mines de Saint-Etienne, 2006.
[21] A. Dolgui, A. Pashkevich, Manipulator motion planning for high-speed robotic laser cutting, International Journal of Production Research, 47 (2009) 5691-5715.
[22] A.P. Pashkevich, A.B. Dolgui, O.A. Chumakov, Multiobjective optimization of robot motion for laser cutting applications, International Journal of Computer Integrated Manufacturing, 17 (2004) 171-183.
[23] B. Zhou, X. Zhang, Z. Meng, X. Dai, Off-line programming system of industrial robot for spraying manufacturing optimization, Proceedings of the 33rd Chinese Control Conference, 2014.
[24] T. Martinec, J. Mlýnek, M. Petrů, Calculation of the robot trajectory for the optimum directional orientation of fibre placement in the manufacture of composite profile frames, Robotics and Computer-Integrated Manufacturing, 35 (2015) 42-54.
[25] J. Mlynek, T. Martinec, Mathematical model of composite manufacture and calculation of robot trajectory, Mechatronics-Mechatronika (ME), 2014 16th International Conference on, IEEE, 2014, pp. 345-351.
[26] P. Debout, H. Chanal, E. Duc, Tool path smoothing of a redundant machine: Application to Automated Fiber Placement, Computer-Aided Design, 43 (2011) 122-132.
[27] H. Dong, M. Cong, D. Liu, G. Wang, An effective technique to find a robot joint trajectory of minimum global jerk and distance, Information and Automation, 2015 IEEE International Conference on, IEEE, 2015, pp. 1327-1330.
[28] A. Piazzi, A. Visioli, Global minimum-jerk trajectory planning of robot manipulators, Industrial Electronics, IEEE Transactions on, 47 (2000) 140-149.
[29] A. Piazzi, A. Visioli, An interval algorithm for minimum-jerk trajectory planning of robot manipulators, Decision and Control, 1997., Proceedings of the 36th IEEE Conference on, IEEE, 1997, pp. 1924-1927.
[30] A. Gasparetto, A. Lanzutti, R. Vidoni, V. Zanotto, Experimental validation and comparative analysis of optimal time-jerk algorithms for trajectory planning, Robotics and Computer-Integrated Manufacturing, 28 (2012) 164-181.
[31] A. Gasparetto, V. Zanotto, A technique for time-jerk optimal planning of robot trajectories, Robotics and Computer-Integrated Manufacturing, 24 (2008) 415-426.
[32] J. Wu, H. Wu, Y. Song, Y. Cheng, W. Zhao, Y. Wang, Genetic algorithm trajectory plan optimization for EAMA: EAST Articulated Maintenance Arm, Fusion Engineering and Design, (2016).

[33] P. Wang, H. Yang, K. Xue, Jerk-optimal trajectory planning for Stewart platform in joint space, Mechatronics and Automation (ICMA), 2015 IEEE International Conference on, IEEE, 2015, pp. 1932-1937.
[34] J. Gao, A. Pashkevich, S. Caro, Manipulator Motion Planning in Redundant Robotic System for Fiber Placement Process, in: P. Wenger, P. Flores (Eds.) New Trends in Mechanism and Machine Science: Theory and Industrial Applications, Springer International Publishing, Cham, 2017, pp. 243-252.
[35] B. Denkena, C. Schmidt, P. Weber, Automated Fiber Placement Head for Manufacturing of Innovative Aerospace Stiffening Structures, Procedia Manufacturing, 6 (2016) 96-104.
[36] J.J. Craig, Introduction to robotics: mechanics and control, Pearson Prentice Hall Upper Saddle River, 2005.
[37] J. Denavit, A kinematic notation for lower-pair mechanisms based on matrices, Trans. of the ASME. Journal of Applied Mechanics, 22 (1955) 215-221.
[38] D.L. Peiper, The kinematics of manipulators under computer control, DTIC Document, 1968.
[39] A. Pashkevich, Real-time inverse kinematics for robots with offset and reduced wrist, Control Engineering Practice, 5 (1997) 1443-1450.
[40] M.L. Husty, M. Pfurner, H.-P. Schröcker, A new and efficient algorithm for the inverse kinematics of a general serial 6R manipulator, Mechanism and machine theory, 42 (2007) 66-81.
[41] A.P. Pashkevich, A.B. Dolgui, K.I. Semkin, Kinematic aspects of a robot-positioner system in an arc welding application, Control Engineering Practice, 11 (2003) 633-647.
[42] J.E. Bobrow, S. Dubowsky, J. Gibson, Time-optimal control of robotic manipulators along specified paths, The international journal of robotics research, 4 (1985) 3-17.
[43] D.P. Bertsekas, D.P. Bertsekas, D.P. Bertsekas, D.P. Bertsekas, Dynamic programming and optimal control, Athena Scientific Belmont, MA, 1995.
[44] K. Roboter, Kuka System Software 5.5-Operating and Programming Instructions for System Integrators, 2010.
[45] A. Gasparetto, P. Boscariol, A. Lanzutti, R. Vidoni, Path Planning and Trajectory Planning Algorithms: A General Overview, Motion and Operation Planning of Robotic Systems, Springer, 2015, pp. 3-27.