Improving the Sequence of Robotic Tasks with Freedom of Execution

Sergey Alatartsev and Frank Ortmeier

Abstract—An industrial robot's workflow typically consists of a set of tasks that have to be repeated multiple times. A task could be, for example, welding a seam or cutting a hole. The efficiency with which the robot performs the sequence of tasks is an important factor in most production domains. The more often a set of tasks can be performed by a robot the more advantages it provides to the company. In most practical scenarios, the majority of tasks have a certain freedom of execution. For example, closed-contour welding task can often be started and finished at any point of the curve. Also the exact orientation of the welding torch is not fixed, but may vary. Currently, these degrees of freedom are used to manually generate robot trajectories. However, their quality highly depends on skills and experience of the robot programmer.

In this paper we propose a method that is able to automatically improve the given sequence of robotic tasks by adding certain freedom to (i) the position of the starting point along the curve, (ii) the orientation of the end-effector and (iii) the robot configuration. The proposed approach does not depend on the production domain and could be combined with any algorithm for constructing the initial task sequence. We evaluate the algorithm on a realistic case study and show that it could significantly improve the production time on the test instances from the cutting-deburring domain.

I. INTRODUCTION

The common workflow of industrial robots is to perform a set of tasks that have no constraints on the order of execution. Performing such tasks consists of the two alternating stages: effective and supporting movements/tasks. Effective movement is the stage when a particular task is being processed (e.g., moving along the welding seam with the switched-on torch). The way of effective movement execution highly depends on the production process requirements. The state of the art is to manually program effective movements using offline or online approaches [1][2]. Supporting movements are needed to move from one effective task to another (e.g., moving between welding seams). Often they do not have constraints on the execution and are programmed manually or calculated with collision-free path planners [3].

Industrial robots have to repeat the same task sequence multiple times. It is important to minimize the execution cost as it greatly influences the production efficiency. For example, the cost can be time, distance or energy. One way to minimize the cost is to optimize the task sequence. There are many approaches that aim at minimizing the sequence of strictly defined tasks and also considering robot kinematics. Some approaches consider the freedom of effective tasks, however, they ignore the robot kinematics and use Euclidean metric instead, see Section II.

In this paper we propose an approach to improve the given task sequence obtained by any of the existing algorithms. This approach offers the following relaxations:

- the tasks are closed contours that can be started/finished at any point of the curve;
- each point within a curve has a window for possible end-effector orientations;
- each point and the corresponding set of allowed orientations could be reached with several robot configurations.

The proposed approach improves the given task sequence by optimizing the entry points of the tasks, end-effector orientation and inverse kinematics solution of each point.

We assume that the working environment is not cluttered. Thus, explicit collision-free planning can be omitted during optimization and applied afterwards. The proposed approach works under the assumption that the freedom of the task execution is defined depending on the industrial process limitations and, therefore, does not harm the quality of the performed work. We also assume that any defined entry point for a particular effective task does not influence the cost of performing this task.

The remainder of the paper is organized as follows. Section II outlines the background and state-of-the-art approaches. Problem specification is covered in Section III. Section IV describes the proposed approach. An evaluation is given in Section V. We conclude and provide an outlook to the future work in Section VI.

II. BACKGROUND

This section gives a brief introduction to robotics and state of the art in path-planning and sequencing problems.

A. Basic Robotic Mathematical Models

In robotics, two different representations of a robot's position are widely used: T-space and C-space. T-space – task space \( SE(3) \), is the space of possible robot end-effector positions \( \mathbb{R}^3 \) and orientations \( SO(3) \), i.e., \( SE(3) = \mathbb{R}^3 + SO(3) \). Often a point in this space is described with homogeneous coordinates [4]. C-space – configuration space \( C \), is the space of possible robot joint angles. Typical industrial robots have 6 joints or degrees of freedom, i.e., \( C = \mathbb{R}^6 \).

There are two mappings that describe the relation between the T-space and the C-space. Forward Kinematics (FK) takes the robot joint angles as input and calculates the corresponding end-effector position and orientation: \( GetFK : C \rightarrow SE(3) \). On the other hand, Inverse
Kinematics (IK) takes the end-effector position and orientation and produces the set of possible robot configurations: \( \text{GetIK} : SE(3) \rightarrow C \). Although, each of these configurations brings the robot’s tool to the same position in T-space, time for the movement between tasks depends heavily on the chosen C-space values. For further information on kinematics we refer to [5].

**B. Task Sequencing**

1) **Sequencing of Simple Tasks**: The majority of approaches represent the sequencing problem as the Traveling Salesman Problem (TSP), which aims at finding the minimal-cost cyclic tour through a set of points. One of the first applications of the TSP to robotics was done by Dubowsky et al. [6]. Later Zacharia et al. [7] involved the robot kinematics and applied a Genetic Algorithm to optimize the task sequence. Baizid et al. [8] extended their approach by optimizing location of the robot’s base. Saha et al. [9] proposed the approach that involves collision-free path planning into the problem of task sequence calculation.

These approaches do not consider the execution freedom of effective tasks during the task sequence optimization. Nevertheless, many of them involve the knowledge about the robot kinematics into sequencing.

2) **Sequencing of Tasks with Freedom of Execution**: In reality not every robotic task can be efficiently approximated as a point. Often robots execute tasks with the complicated shapes like closed contours, open-end curves or 3D volumes. Recently extra freedom was applied for laser welding task specification [10]. The position from which the task could be performed is represented as a truncated cone. Gentilini et al. [11] addressed the task sequencing problem for camera inspection tasks, where the robot has to travel from one area to another and take a picture with a camera mounted on the end-effector. The goal is to optimize both end-effector positions in the corresponding areas and the tour cost. They modeled the problem with Mixed-Integer Nonlinear Programming and introduced a heuristic to speed up the calculation time. Alatartsev et al. [12], [13] solved the task sequencing problem for the industrial use case when a robot has to cut out the holes from a plastic detail. They proposed the Constricting Insertion Heuristic (CIH).

The approaches stated above involve the freedom of task execution by calculating task entry points in addition to the sequence. However, due to the increase of the search space, they ignore the information about the robot kinematics and use Euclidean distance as a cost.

**C. Path Planning with End-effector Pose Constraints**

Often the end-effector pose (i.e., position and orientation) is strictly defined by the programmer. Only few applications define a set of possible poses. There are several approaches from the path planning domain that are capable of specifying the constraints on the end-effector pose for the goal and/or along the path. Stilman [14] represented the constrained task as a combination of a task frame, coordinate system and a motion constrained vector. The last one is a vector of binary values that corresponds to each of the coordinates. A value of one means that the end-effector movement should not change the corresponding coordinate. Later Berenson et al. [15] proposed the concept of Task Space Region (TSR). The idea is to represent the end-effector goal in the T-space as a continuous region – a box that limits possible translations and rotations. TSRs are also used to describe the constraints on the end-effector along the path. Yao et al. [16] addressed the Path planning Problem with General End-effector Constraints (PPGEC). In PPGEC, the starting robot configuration and the desired pose are given. The aim is to find the goal configuration and the desired collision-free path while satisfying the pose constraints.

**D. Summary**

Researchers omit either the kinematics or the task freedom, as considering both in task sequencing is unpractical and leads to a large search space that is not possible to solve in a reasonable time. In contrast to the state-of-the-art approaches, in this paper we propose an approach that takes the task sequence obtained by any sequencing algorithm stated above and adapts it to the robot. The adaptation is done by optimizing entry points, end-effector orientations and corresponding kinematics solutions. We make use of task description methods from the path planning domain. Unlike other path planning approaches, the task entry points are calculated globally for the whole closed loop sequence but not locally, i.e., goal-to-goal. In this paper, we are not focusing on building paths between the entry points but rather minimizing the entry point positions and orientations.

**III. PROBLEM SPECIFICATION**

**A. Task Freedom Specification**

Often programmers define task geometry by a finite number of T-space points. However, in reality the task geometry is typically continuous, e.g., a closed contour. Thus, tasks with discrete representation can be defined by splines. By sampling in the interval \([0, 1]\), one can obtain a corresponding point of the task geometry that robot has to visit with its tool, i.e., \(T : [0, 1] \rightarrow SE(3)\).

In this section we propose the extension, where instead of a single T-space point, the task returns a special data container called node point \(NP\). A node point is a tuple \(NP = (M, ([a_l, a_u], [b_l, b_u], [c_l, c_u]))\), where:

- \(M\) is a homogeneous matrix that shows the position \(x_{np}, y_{np}, z_{np}\) and orientation \(X_w, Y_w, Z_w\) of the node point in relation to the base coordinate system \(X, Y, Z\):

\[
M = \begin{bmatrix}
X_{wx} & Y_{wx} & Z_{wx} & x_{np} \\
X_{wy} & Y_{wy} & Z_{wy} & y_{np} \\
X_{wz} & Y_{wz} & Z_{wz} & z_{np} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- \([a_l, a_u]\) represent the lower and upper bounds for the polar angle in the spherical coordinates, therefore, the maximum limit is \([0, 180]\).
**B. Task Entry Point Specification**

Let us define the structure of the entry point that will be used further in the optimization. It is denoted as $EP_i = (T_i, k, a, b, c, conf_{best})$, where:

- $k \in [0, 1]$ is the interpolation parameter that is used to obtain node point $NP_k$ of the task $T_i$.
- $a, b, c$ angles are within the orientation window of the node point $NP_k$.

**Entry point $EP_i$ for the $T_i$ and nodepoint $NP_k$**

- $conf_{best}$ is the minimal-cost robot configuration to reach the end-effector pose that is calculated according to Section III.C.

An entry point $EP_i$ allows us to obtain the information about the point (i.e., $x_{np}$, $y_{np}$, $z_{np}$, from matrix $M \in NP_k$) from which the robot tool has to start processing the task $T_i$. Furthermore, $EP_i$ specifies the end-effector orientation (i.e., $a, b, c$) and robot configuration (i.e., $conf_{best}$). Entry point $EP_i$ of the task $T_i$ is visualized in Fig. 1.B.

**C. Calculating the Robot End-effector Pose**

Note that the entry point is defined in $X_w, Y_w, Z_w$ coordinate system, i.e., in relation to the task. However, the robot end-effector pose for this entry point has to be defined in global coordinates. In this section, we provide the details on calculation of the end-effector pose $Goal_i$ in global coordinates $X, Y, Z$ for a particular entry point $EP_i = (T_i, k, a, b, c, conf_{best})$. The position of the $Goal_i$ is depicted with $(x_{ef}, y_{ef}, z_{ef})$ point and the orientation with $X_{ef}, Y_{ef}, Z_{ef}$ in Fig. 1.C. Angles $a$ and $b$ are used to compute the approaching vector of the end-effector, i.e., vector $X_{ef}$. Angle $c$ denotes the desired orientation along the tool axis and allows us to calculate normal and sliding vectors of the end-effector, i.e., vectors $X_{ef}, Y_{ef}$.

The first step to get the end-effector $Goal_i$ point is to calculate the corresponding node point $NP_k$ for the current $k$ by calling function $T_i$. Then calculate the rotation matrix for the corresponding angles $a, b, c$ of entry point $EP_i$:

$$M_{rot} = RotZ(b) \cdot RotY(a) \cdot RotZ(c),$$

where $a, b, c \in EP_i$.

$RotY$ and $RotZ$ are the rotation matrices about the corresponding axis in three dimensional space. Then calculate the robot end-effector position $EF = (x_{ef}, y_{ef}, z_{ef})^T$:

$$x_{ef} = R \cdot \sin(a) \cdot \cos(b)$$

$$y_{ef} = R \cdot \sin(a) \cdot \sin(b)$$

$$z_{ef} = R \cdot \cos(a)$$
$R$ is the length of the robot tool e.g., knife, drill. We assume that $R$ is constant during the whole working process. Finally, calculate the $Goal_i$ T-space point for the robot to reach:

$$Goal_i = M \cdot \begin{bmatrix} M_{rot} & EF \\ 0 & 1 \end{bmatrix} \cdot RotX(180)$$

$RotX$ is the rotation matrix about X axis in three dimensional space. The function that returns the end-effector pose $Goal_i$ for a particular entry point $EP_i$ is denoted as: $GetGoal(EP_i) → Goal_i$.

D. Problem Statement

Given a sequence of $n$ tasks $Job = (T_1, ..., T_n)$ with freedom of execution. The goal is to find an entry point sequence $EPS = (EP_1, ..., EP_n)$, where $EP_i = (T_i, k, a, b, c, conf_{best})$ such that cost of robot supporting movement between entry point configurations $conf_{best}$ is minimized. Entry point is defined in such a way that $a, b, c$ are within the windows of the corresponding node point $T(k) → NP_k$, and T-space point $GetGoal(EP_i) → Goal_i$ is reachable by the robot, i.e., $GetIK( Goal_i ) \neq \emptyset$.

Presented problem is similar to the Touring Polygon Problem (TPP) [17]. The goal of TPP is to find a minimal-cost cyclic tour through the sequence of polygons such that it visits every polygon in at least one point. Nevertheless, the problem presented in this paper is more complicated than TPP. In our case the cost function depends not only on the position of the point, but also on the chosen robot configuration. In presented problem the areas are not simple polygons but rather complicated 3D shapes. The stated problem is an extension of the TPP and therefore is also NP-hard.

IV. ALGORITHM DESCRIPTION

A. Problem Decomposition

It is computationally difficult to solve the presented problem, as it leads to the large and multidimensional search space. We propose to apply hierarchical optimization. Its goal is to perform a fast local search on every nested stage instead of applying slow global optimization techniques to the whole problem. We specify three nested stages:

Stage 1: for each entry point $EP_i$ optimize parameter $k$. This gives a new position of the $NP_k$.

Stage 2: optimize orientation angles $a, b, c$, that are within the orientation window of the node point $NP_k$ selected on Stage 1.

Stage 3: optimize robot configuration $conf_{best}$ for the position $k$ and orientation $a, b, c$ selected on Stages 1 and 2.

Each outer stage depends on the results of optimization on the inner stage, see Fig. 2. In other words, to optimize the cost of the solution, i.e., the cost of the entry point sequence for robotic tasks, one has to optimize the position of each entry point, i.e., parameters $k$. The cost of the point position on the curve depends on the chosen orientation of the end-effector, i.e., angles $a, b, c$. The cost of the orientation depends on a best inverse kinematics solution for this position and orientation, i.e., $conf_{best}$. The cost of the kinematics solution in a particular entry point depends on the possible kinematics solutions in the entry points of other tasks.

During the optimization process the outer stage executes the inner stage multiple times. For example, while searching for the near-optimal orientation of the end-effector (Stage 2) in some point on the curve, the costs of multiple orientation vectors have to be compared. For each of them inverse kinematics optimization (Stage 3) has to be performed.

We applied the Pattern search [19] for optimization on the Stages 1 and 2. The exhaustive search is applied for the Stage 3.

Initialization

One has to initialize entry point sequence $EPS = (EP_1, ..., EP_n)$ for the given task sequence $Job = (T_1, ..., T_n)$ with extra freedom defined with notation from Section III-A. It is done by defining an entry point $EP_1 = (T_1, k, a, b, c, conf_{best})$ for every task $T_i$. In general, components of $EP_i$ can be chosen randomly such that $a, b, c$ are within the node point $NP_k$.

It is also possible to make use of the algorithms from Section II-B.2 to obtain parameter $k$. In this paper, CH [12] was applied. Then the angles $a, b, c$ are set to the middle of the corresponding orientation windows from the node point $NP_k ← T_i(k), k ∈ EP_i$.

Afterwards, one has to calculate the best robot configuration $conf_{best}$ for every $EP_i$. It can be done by obtaining the list of possible robot configurations for each entry point and constructing a graph, where every configuration from entry point $EP_i$ is connected with every configuration from entry point $EP_{i+1}$. Then the graph search algorithm obtains the shortest path that visits only one configuration in every entry point. For this purpose Dijkstra search is used in this paper.

Note that the entry point sequence $EPS$ is transferred between all stages and represents a container that accumulates all information about the end-effector poses and robot configurations for a particular task sequence.

Main Loop

The underlying idea is based on the Rubber-band algorithm (RBA) [18]. The principle of the RBA is to iterate over the areas and improve entry points, one at a time. The proposed modification of RBA takes a sequence of entry point $EPS$ and outputs the optimized $EPS$.

The overall workflow is shown in Algorithm 1. We iteratively run the index $i$ from 1 to the number of entry points $n$ and call $OptimizePosition$ (line 3, Algorithm 1). It returns the optimized $EPS$ and the cost. This approach is described in the Stage 1. The stopping condition could be elapsed computational time, desired precision of the solution or a maximum number of iterations.

The whole entry point sequence is passed to the algorithm $OptimizePosition$ instead of a single $EP_i$, as information about all entry points in the sequence is required to calculate the cost at the Stages 1 and 3.
can not improve the current cost. One way to determine the inner Stages 2 and 3 come to the local minimum and many times. This happens when local search strategies on might not be found even if the step $\Delta$ first improvement was found. However, the improvement repeat the process.

Algorithm 1: RBA: Main loop

**Input**: Entry point sequence $EPS = (EP_1, ..., EP_n)$

**Output**: Entry point sequence $EPS$

1. while stopping condition is not satisfied do
   2. for $i \leftarrow 1$ to $n$ do
      3. $EPS, cost \leftarrow$ OptimizePosition($EPS, i$);
   4. end
   5. return $EPS$

Stage 1: Optimization of the Entry Point Position

In this section, the OptimizePosition algorithm is described. Its goal is to choose the best parameter $k$ for $EP_i$ from the given $EPS$. Its workflow is shown in Algorithm 2. At first, Stage 1 obtains the current cost $cost_{cur}$, initializes the step $\Delta_k$ as a value from $[0, 1]$ and saves the initial $EP_i$ (lines 1–3, Algorithm 2). $GetSequenceCost$ is a user-defined cost function that represents the optimization criterion. It returns the cost of the supporting movement in regard to a desired metric, e.g., time or traveled distance in C-space.

Then, the algorithm calculates the cost for the values: $k + \Delta_k$ and $k - \Delta_k$, see Stage 1 in Fig. 2 and lines 6 and 9 in Algorithm 2. After each step we call the OptimizeOrientation algorithm (described in Stage 2) to obtain the costs: $cost_{k-\Delta}$, $cost_{k+\Delta}$ and the new entry point sequences $EPS_{k-\Delta}$ and $EPS_{k+\Delta}$ respectively (lines 7 and 10, Algorithm 2). Note that OptimizeOrientation is called with different $EPS$, where parameter $k$ of $EP_i$ has different values: $k + \Delta_k$ and $k - \Delta_k$. If one of these steps brings an improvement, then stop the algorithm and return the improved entry point sequence and the new cost (lines 13 and 15, Algorithm 2). If these actions did not improve the cost then decrease the step $\Delta_k$ (line 17, Algorithm 2) and repeat the process.

Algorithm returns the entry point sequence when the first improvement was found. However, the improvement might not be found even if the step $\Delta_k$ was decreased many times. This happens when local search strategies on the inner Stages 2 and 3 come to the local minimum and can not improve the current cost. One way to determine this is to compare $cost_{k-\Delta}$ and $cost_{k+\Delta}$ values between two iterations. When there was no improvement, then the rollback for $EP_i$ should be made (line 19, Algorithm 2). This approach guarantees that during optimization the overall sequence cost will decrease or stay the same.

Stage 2: Optimization of the End-effector Orientation

This section covers details on the OptimizeOrientation algorithm. This algorithm optimizes $a, b, c$ of the $EP_i$ from a given $EPS$ for the value $k$ defined by the outer Stage 1. The output is an optimized entry point sequence $EPS$ and a
new cost. In contrast to Stage 1, where optimization is done until the first improvement was found, the optimization of the orientation is done until a stopping condition is met. The stopping condition can be the reaching of a solution desired precision.

At first, OptimizeOrientation initializes its main variables in lines 1–3 in Algorithm 3. The algorithm initializes the starting angles \( a, b, c \). The middle angles of the orientation window can be applied, e.g., \( a = a_1 + (a_u - a_1)/2 \). Then the initial cost is calculated with the algorithm OptimizeConfiguration described in Stage 3. Finally, three steps \( \Delta_a, \Delta_b, \Delta_c \) are initialized. The size of the step depends on the allowed interval of each angle of the current node point, e.g., initially \( \Delta_a \) is a half of the interval \([a_1, a_u]\). \( \Delta_b \) and \( \Delta_c \) are obtained analogously.

The main idea of Stage 2 is to iteratively improve the orientation angles. Within one iteration (lines 4–18, Algorithm 3) six angle combinations are checked, i.e., \((a - \Delta_a, b, c), (a + \Delta_a, b, c), (a, b - \Delta_b, c), (a, b + \Delta_b, c), (a, b, c - \Delta_c), (a, b, c + \Delta_c)\). In pseudocode we show only the case \((a - \Delta_a, b, c)\), as the other combinations are similar.

Angle \( a \) takes the value \( a - \Delta_a \). If the \( a - \Delta_a \) is not within the range \([a_1, a_u]\), then the step \( \Delta_a \) is decreased (lines 7 and 13, Algorithm 3). If angle \( a - \Delta_a \) is within the orientation window, then the optimized \( E P_{a-\Delta} \) and \( cost_{a-\Delta} \) are obtained by optimizing the robot configuration with the function OptimizeConfiguration (line 8, Algorithm 3). If the newly obtained cost is less than the initial cost, then save the new sequence and its cost (lines 9–11, Algorithm 3). Then, the angle \( a \) takes the initial value and the other five combinations are checked (lines 14 and 15, Algorithm 3).

If no improvement was found, the steps are decreased (lines 16–17, Algorithm 3). Finally, the algorithm returns the optimized entry point sequence \( E P_{best} \) and its cost \( cost_{cur} \) to the Stage 1.

Stage 3: Optimization of the Robot Configuration

The goal of Stage 3 is to choose the near-optimal robot configuration to reach the position \( k \) given by Stage 1 and the orientation \( a, b, c \) given by Stage 2. The choice of the best configuration might be nontrivial. It could cause a change of configurations in the entry points of other tasks. It is effective to use the graph search described in the initialization section to optimize configurations in all entry points. As the Stage 3 is executed more often than other stages, having there a global algorithm dramatically increases the computational time. Therefore, a local strategy that optimizes \( conf_{best} \) only in the current \( E P_i \) is preferred.

The approach of the Stage 3 is implemented as the function OptimizeConfiguration. It takes \( E P \) and the current index \( i \) and returns \( E P_{best} \) with \( conf_{best} \) being optimized only for \( E P_i \) and its cost. The approach is described in Algorithm 4. At first, the function GetGoal calculates the \( Goal_i \) for the \( E P_i \) according to Section III-C. The function GetIK obtains a set of possible robot configurations \( IKS \) for the \( Goal_i \) according to Section II-A. If no inverse kinematics solution was found, i.e., the \( Goal_i \) is not reachable by the robot, then the algorithm returns \( E P \) and infinite cost. The core part is to choose such IK solution from the \( IKS \) that minimizes the cost to configurations of the neighboring entry points \( E P_{i-1} \) and \( E P_{i+1} \) (lines 6–10, Algorithm 4). Function GetCost returns the cost between two robot configurations. In this paper Euclidean distance.

**Algorithm 3: OptimizeOrientation**

**Input:** Entry point sequence \( EPS \), current index \( i \)

**Output:** Entry point sequence \( EPS_{best} \), cost of the \( EPS \)

```
1. \( a, b, c \leftarrow \text{InitializeStartingAngles} \);
2. \( EPS_{best}, cost_{cur} \leftarrow \text{OptimizeConfiguration}(EPS, i) \);
3. \( \Delta_a, \Delta_b, \Delta_c \leftarrow \text{InitializeSteps} \);
4. while stopping condition is not satisfied do
   5.     \( a_{temp} \leftarrow a \);
   6.     \( a \leftarrow a - \Delta_a \), where \( a \in EP_i, EP_j \in EPS \);
   7.     \( \text{if } (a_i < a) \text{ and } (a < a_{u_i}) \text{ then} \)
   8.          \( EPS_{a-\Delta}, cost_{a-\Delta} \leftarrow \text{OptimizeConfiguration}(EPS, i) \);
   9.          \( \text{if cost}_{a-\Delta} < cost_{curr} \text{ then} \)
             10.              \( EPS_{best} \leftarrow EPS_{a-\Delta} \);
             11.              \( cost_{curr} \leftarrow cost_{a-\Delta} \);
   12.     \( else \)
   13.          \( \Delta_a \leftarrow \Delta_a / 2 \);
   14.          \( a \leftarrow a_{temp} \);
   15.          \( \ldots \)
   16.          \( \text{Check } (a + \Delta_a, b, c), (a, b - \Delta_b, c), (a, b + \Delta_b, c), (a, b, c - \Delta_c), (a, b, c + \Delta_c) \text{ by analogy} \);
   17.          \( \ldots \)
   18.          \( \text{if } (\text{no cost improvement}) \text{ then} \)
             19.              \( \Delta_a \leftarrow \Delta_a / 2; \Delta_b \leftarrow \Delta_b / 2; \Delta_c \leftarrow \Delta_c / 2 \);
20.     \( \text{return } EPS_{best}, cost_{cur} \).
```

**Algorithm 4: OptimizeConfiguration**

**Input:** Entry point sequence \( EPS \), current index \( i \)

**Output:** Entry point sequence \( EPS \), cost of the \( EPS \)

```
1. Goal_i \leftarrow \text{GetGoal}(EP_i) \);
2. IKS \leftarrow \text{GetIK}(Goal_i) \);
3. \( \text{if } IKS = \emptyset \text{ then} \)
   4. \( \text{return } EPS, \infty \);
   5. \( cost_{IK_{min}} \leftarrow \infty \);
6. foreach \( IK_j \in IKS \) do
   7. \( cost_{IK_{temp}} \leftarrow \text{GetCost}(IK_j, conf_{best} \in EP_{i-1}) + \text{GetCost}(IK_j, conf_{best} \in EP_{i+1}) \);
   8. \( \text{if } cost_{IK_{temp}} < cost_{IK_{min}} \text{ then} \)
       9. \( cost_{IK_{min}} \leftarrow cost_{IK_{temp}} \);
       10. \( conf_{best} \leftarrow IK_j \), where \( conf_{best} \in EP_i, EP_j \in EPS \);
   11. \( cost \leftarrow \text{GetSequenceCost}(EPS) \);
12. \( \text{return } EPS, cost \);
```
adaptation stage

Time

5.885
62.24
6.021
Time

19.55 s.
30
impr. (%)
value
103.29 s.
26.17
5.758
51.37
482.85 s.
20.844
6.332
8.450
102.20
16.02
10.973
2.036
Metric
1.899
value
value
4.063
2.087
impr. (%) 32.96
54.73
1.934
impr. (%) 6.829
value
1.761
value
2.224
24.459
100
30
Metric
allowed orientation window.

angles a, b, c, d, e and f represent the middle of the corresponding allowed orientation window.

The main contribution of the paper is the demonstration that even a small freedom of the end-effector orientation and position as well as the choice of robot configuration could

Fig. 4. Results of adapting the task sequence to the robot. One iteration is the optimization of the one task.

We evaluate the sequence with three metrics: T-space, Time and C-space. The Time metric for the tour is the time in seconds required to perform all supporting movements. It is obtained by the build-in OpenRAVE function RetimeActiveDOFTrajectory() using the Parabolic Trajectory Retimer. The C-space cost is calculated as a Euclidean distance between robot configurations, i.e., the distance between two 6D points. It is calculated without joint weight coefficients [22]. C-space cost for the tour is the sum of all C-space costs for all supporting movements. During the evaluation we minimized two parameters: Time and C-space cost. The results are presented in Fig. 3 and in Table I. The convergence speed is presented in Fig. 4. We report the results up to 30 and 100 iterations for tests A and B respectively.

The use of the Time metric appears to be less efficient than that of the C-space metric, as the robot trajectory has to be recalculated multiple times and it increases the time for calculation. Since the two metrics, distance traveled in joint space and the time are interconnected, it is possible to use C-space metric to optimize the time. Using the Time metric makes sense for applications where the execution time is critical. The proposed approach is capable of reducing the time required for supporting movements by 45% for 6 tasks in 20s and by 38% for 15 tasks in 68s.

V. EVALUATION

The evaluation of the proposed approach was done on two test cases inspired by real-world applications: cutting the holes in plastic cover 1 and dashboard casing 2. The first detail is the plastic cover that consists of 6 closed-contour effective tasks (test A). The second is the car dashboard casing that consists of 15 closed-contour effective tasks (test B). The orientation window is \( a_1 = 30, a_2 = 150, b_1 = -15, b_2 = 15, c_1 = 0, c_2 = 10 \) and it is the same for all node points of all tasks.

For the evaluation we used an Intel Core 2 Quad CPU, 2.83 GHz with 8 Gb RAM, running Ubuntu 12.04. The approach is implemented using Python in OpenRAVE [20]. Inverse kinematics is calculated analytically with the IKfast generator for the KUKA KR30L16 robot. All robot configurations are checked for being not in a collision with environment. Full evaluation results and a demo video are available on-line 3

The initial solution (i.e., the sequence of tasks and Cartesian entry points) is obtained by CIH [12] and improved with 2-Opt [21] using T-space cost. T-space cost is measured in meters and represents a Euclidean length of the tour of end-effector positions (i.e., point \( x_{ef}, y_{ef}, z_{ef} \) in Fig.1.C). For the given Cartesian entry points, the end-effector orientation angles \( a, b, c \) are chosen as the middle of the corresponding allowed orientation window.

We evaluate the sequence with three metrics: T-space, Time and C-space. The Time metric for the tour is the time in seconds required to perform all supporting movements. It is obtained by the build-in OpenRAVE function RetimeActiveDOFTrajectory() using the Parabolic Trajectory Retimer. The C-space cost is calculated as a Euclidean distance between robot configurations, i.e., the distance between two 6D points. It is calculated without joint weight coefficients [22]. C-space cost for the tour is the sum of all C-space costs for all supporting movements. During the evaluation we minimized two parameters: Time and C-space cost. The results are presented in Fig. 3 and in Table I. The convergence speed is presented in Fig. 4. We report the results up to 30 and 100 iterations for tests A and B respectively.

The use of the Time metric appears to be less efficient than that of the C-space metric, as the robot trajectory has to be recalculated multiple times and it increases the time for calculation. Since the two metrics, distance traveled in joint space and the time are interconnected, it is possible to use C-space metric to optimize the time. Using the Time metric makes sense for applications where the execution time is critical. The proposed approach is capable of reducing the time required for supporting movements by 45% for 6 tasks in 20s and by 38% for 15 tasks in 68s.

VI. CONCLUSION

This paper considers the problem of improving a given task sequence. The improvement is achieved by allowing a certain degree of freedom for the effective tasks and optimizing the entry points as well as the corresponding inverse kinematics solutions. We present a way to describe the freedom of curve-like effective tasks. The main idea is to decompose the problem into three stages and solve each of them with local search techniques. The proposed approach significantly improves the initial solution in a short computational time. The algorithm is evaluated on two scenarios from the cutting-deburring industrial domain.

<table>
<thead>
<tr>
<th>Test</th>
<th>Metric</th>
<th>value</th>
<th>value</th>
<th>impr. (%)</th>
<th>value</th>
<th>value</th>
<th>impr. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T-space</td>
<td>1.899</td>
<td>2.036</td>
<td>107.24</td>
<td>1.934</td>
<td>101.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>4.063</td>
<td>2.224</td>
<td>54.73</td>
<td>2.087</td>
<td>51.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-space</td>
<td>8.450</td>
<td>1.354</td>
<td>16.02</td>
<td>1.761</td>
<td>20.84</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>Metric</th>
<th>value</th>
<th>value</th>
<th>impr. (%)</th>
<th>value</th>
<th>value</th>
<th>impr. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>T-space</td>
<td>5.758</td>
<td>6.332</td>
<td>109.96</td>
<td>5.885</td>
<td>102.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>10.973</td>
<td>6.829</td>
<td>62.24</td>
<td>6.021</td>
<td>54.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-space</td>
<td>24.459</td>
<td>6.401</td>
<td>26.17</td>
<td>8.063</td>
<td>32.96</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I
EVALUATION RESULTS FOR TESTS A AND B

Calculation time: 19.55 s., 103.29 s.
Calculation time: 68.04 s., 482.85 s.
greatly influence the supporting movement cost. However, in order to obtain more realistic results, the approach could be further improved. Currently the approach works under the assumption that any defined entry point of a particular effective task does not influence or insignificantly influence the cost of performing this task. This is true for multiple scenarios, e.g., camera inspection tasks (i.e., single point to take picture but with possible orientation window) or cutting small contours. The future research direction is to simulate the whole robot movement for the contour and integrate it into the cost function. Currently, we assume that no point of the tasks creates a singularity for the robot, so that the number of inverse kinematics solutions is finite. The future direction is to involve a technique that will avoid singularities. Currently, Euler angles were used to describe the tasks, as they are easily understandable by the programmer. In order to make the interpolation more stable, Quaternions could be used instead. One more future step is to apply collision-free planners to industrial problems in order to compare the real-world cost to the estimated cost in the algorithm.

**References**


![Fig. 3. Evaluation was performed on the examples: car dashboard casing (row A) and engine cover (row B). The robot end-effector path is indicated with black and task geometries are indicated with red.](image-url)