Simultaneous Calibration of Extrinsic Laser And Robot Kinematics Parameters with Three Planar Constraints

Teguh Santoso Lembono, Francisco Suárez-Ruiz and Quang-Cuong Pham

***Abstract*— In many industrial robotics applications, such as spot-welding, spray-painting or drilling, the robot is required to visit successively multiple targets. The robot travel time among the targets is a significant component of the overall execution time. This travel time is in turn greatly affected by the order of visit of the targets, and by the robot configurations used to reach each target. Therefore, it is crucial to optimize these two elements, a problem known in the literature as the Robotic Task Sequencing Problem (RTSP). Our contribution in this paper is two-fold. First, we propose a fast, near-optimal, algorithm to solve RTSP. The key to our approach is to exploit the classical distinction between task space and configuration space, which, surprisingly, has been so far overlooked in the RTSP literature. Second, we provide an open-source implementation of the above algorithm, which has been carefully benchmarked to yield an efficient, ready-to-use, software solution. We discuss the relationship between RTSP and other Traveling Salesman Problem (TSP) variants, such as the Generalized Traveling Salesman Problem (GTSP), and show experimentally that our method finds motion sequences of the same quality but using several orders of magnitude less computation time than existing approaches.**

I. Introduction

Robots have been used in many industrial applications, such as pick and place, spray-painting, and spot-welding. In those applications, the robots either do not need very high accuracy (such as in material handling for large objects), or the robots are programmed online (such as using teaching method), where the important parameter is the repeatability of the robot, not the accuracy. However, there are a lot of interests to use robot on many other applications where the accuracy of the robot becomes very crucial, where the robot has to adapt to each task automatically with a great precision. One such example is robot drilling task. In robot drilling task, the robot is supposed to drill several holes at precisely-defined locations on a workpiece, which can change for each task. To program the robot offline for such a task, the robot has to be able to move to a location very accurately. However, the accuracy of the robot is very often unknown, as the manifacturer only provides the repeatability of the robot. Meanwhile, the robot kinematic parameter used in the controller is normally the standard one used for most of the robot, while in fact each robot will have slightly different kinematics parameters (due to inaccuracy in the manufacturing process, assembly, and operating environment).

The accuracy of such a robotic system depends on several factors: The accuracy of the robot arm, the accuracy of the measurement system, and the accuracy of the end effector tool. The accuracy of the robot arm is determined by how closely the kinematic parameters of the robot model resembles the actual kinematic parameter of the physical robot. This is affected by the manufacturing process, the assembly process, and the wear and tear during the operation of the robot. To achieve a better accuracy, a robot calibration is normally conducted, either by using an external measurement system (such as motion capture system, or spinarm, or CMM), or by constraining the motion of the end effector. The accuracy of the measurement system can be divided into two parts: the accuracy of the measurement device itself, and the accuracy of the transformation between the measurement frame to the robot frame. A camera system, for example, can give a lot of information, but it is generallly less precise as compared to a laser system. However, the overall accuracy is still determined by the accuracy of the transformation between the measurement coordinate frame to the robot coordinate frame. To obtain accurate transformation, the information from the CAD model is not sufficient, as there will be error during the assembly. A process known as hand-eye calibration is normally used to find such transformation. Lastly, the accuracy of the end effector transformation. This is the transformation from the robot flange to a pre-determined location on the end effector (such as the tip of the drilling bit). Calibration of the end effector is generally known as TCP calibration.

Here we concern ourselves with solving the first two issues. We propose to use a laser line scanner, attached on the robot arm, to calibrate the kinematics parameters of the robot while at the same time calibrating the transformation between the laser scanner and the robotic arm. The overall procedure is as follows: 1. The scanner is attached on the robot, and three (approximately) perpendicular planes are set around the robot 2. The robot is moved to several positions, such that the laser plane intersects each of the plane (one at a time). The data from the laser as well as the joint angles information are collected 3. An optimization algorithm (Levenberg-Marqueadt) is used to find the optimal robot kinematics parameters, together with the laser-robot transformation, with the following constraints: the projected laser data should correspond to the three planes.

The proposal has several advantages: 1. It does not require an external measurement system. The measurement device that we used to calibrate the robot is also the device that will be used on the actual process. 2. It does not require specially calibrated object to calibrate the robot. The planes coordinates do not need to be known precisely, only approximately. 3. The calibration can be perform for the whole robot

The authors are with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore.

workspace, not only a local portion of it.

II. Related works

- calibration of robot kinematics parameters: using external measurement system: using constraints:1 plane, several plane
- calibration of laser extrinsic parameter using external using constraints
- simultaneous calibration

III. RoboTSP algorithm

A. Setting

Consider n targets in the task-space. A tour in the task-space that visits each target exactly once is called a *task-space tour*[1]. We first compute Inverse Kinematics (IK) solutions for each target – using a suitable discretization for the free-DoFs if necessary. A tour in the configuration-space that starts from the robot "home" configuration, visits, for each target, exactly one IK solution associated with that target, and returns to the "home" configuration is called a *configuration-space tour*. Our objective is to find the fastest *collision-free* configuration-space tour subject to the robot constraints (e.g. velocity and acceleration bounds).

Let m_i be the number of IK solutions found for target i. If we do not take into account obstacles, there are $(n-1)!(\prod_{i=1}^{n} m_i)$ possible configuration-space tours (with straight paths) for this task. One cannot therefore expect to find the optimal sequence by brute force in practical times.

B. Proposed algorithm

As presented in Section I, our method consists in:

1) Finding a (near-)optimal task-space tour in a *task-space metric*;
2) Given the order found in Step 1, finding, for each target, the optimal robot configuration, so that the corresponding configuration-space tour has minimal length in a *configuration-space metric* – collisions are ignored at this stage;
3) Computing fast collision-free *configuration-space* trajectories by running classical motion planning algorithms (e.g. Rapidly-Exploring Random Tree (RRT)-Connect with post-processing [1], [2]) through the configurations found in Step 2 and in the order given by Step 1.

Implementation details and benchmarking results for Steps 1 and 3 are given in Section IV.

Regarding Step 2, we first construct an undirected graph as depicted in Fig. 1. Specifically, the graph has n layers, each layer i contains m_i vertices representing the m_i IK solutions of target i (the targets are ordered according to Step 1), resulting in a total of $\sum_{i=1}^{n} m_i$ vertices. Next, for $i \in [1, \dots, n-1]$, we add an edge between each vertex of layer i and each vertex of layer $i+1$, resulting in a total of $\sum_{i=1}^{n-1} m_i m_{i+1}$ edges. Finally, we add two special vertices: "Start" and "Goal", which are associated with the robot "home" configuration, and connected respectively to the m_1 vertices of the first layer and the m_n vertices of the last layer.

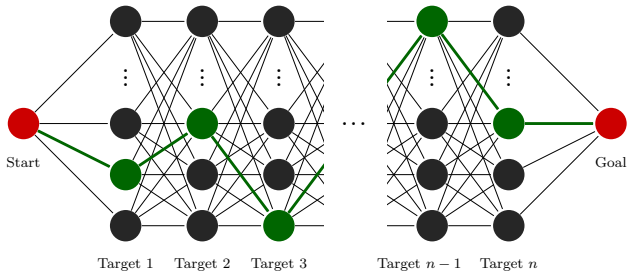

Fig. 1. Graph constructed for Step 2 of the algorithm. The targets are ordered according to Step 1. The shortest path (green lines) connecting the "Start" and "Goal" vertices will yield the optimal configuration-space tour that visits the targets in the order specified by Step 1.

The edge costs are computed according to a configuration-space metric: for instance, the cost for the edge joining vertices $\boldsymbol{q}$ and $\boldsymbol{q}'$ can be given by the Euclidean distance in the configuration space $\sqrt{\sum_{k=1}^{\text{DoF}} (q_k - q'_k)^2}$. Section IV-B examines in detail how the choice of such metrics influences the quality of the final path. One can note here that the metric should be fast to compute – in particular, collisions are ignored at this stage – since the costs must be computed for all $m_1 + \sum_{i=1}^{n} m_i m_{i+1} + m_n$ graph edges.

Finally, we run a graph search algorithm to find the shortest path between the "Start" and "Goal" vertices. By construction, any path between the "Start" and "Goal" vertices will visit exactly one vertex in each layer, in the order specified by Step 1. Conversely, for any choice of IK solutions for the n targets, there will be a path in the graph between the "Start" and "Goal" vertices and going through the vertices representing these IK solutions. Therefore, Step 2 will find the *true optimal* selection of IK solutions that minimize the total cost, according to the specified configuration-space metric, given the order of the targets.

C. Complexity analysis

For Step 1, it is well-known that TSP is NP-complete, which means that finding the true optimal tour for n targets has in practice an exponential complexity. Many heuristics have been developed over the years to find near-optimal tours. For instance, *2-Opt* [3] and Lin-Kernighan-Helsgaun (LKH) [4] can find tours in practical times with an optimality gap bellow 5% and 1% respectively [5].

For Step 2, let M be an upper-bound of the number of IK solutions m_i per target. The number of graph vertices is then $\mathcal{O}(nM)$ and the number of the graph edges is $\mathcal{O}\big(nM^2\big)$. Since Dijkstra's algorithm (with binary heap) has a complexity in $\mathcal{O}(|E| \log |V|)$ where $|E|$ and $|V|$ are respectively the number of edges and vertices, Step 2 has a complexity in $\mathcal{O}\big(nM^2 \log (nM)\big)$.

For Step 3, one has to make $n-1$ queries to the motion planner, yielding a complexity in $\mathcal{O}(n)$. However, as the constant in the $\mathcal{O}()$ (average computation time per motion

[1]Strictly speaking, a tour requires to return to the first target, so we are making a slight abuse of vocabulary here.

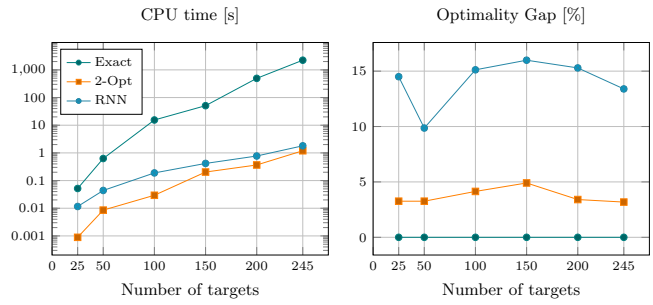

Fig. 2. Benchmarking results for three task-space TSP solvers. (a) CPU times for different number of targets. The Y axis is logarithmic (b) Task execution time for different number of targets. The sets of targets are selected randomly. The last sample considers all the 245 targets.

planning query) is large, the overall computation time is dominated by that of Step 3 in our setting. In general, the computation time of motion planning queries depends largely on the environment (obstacles), see [6], [7] for recent benchmarking results showing the CPU time required when planning practical robot motions.

IV. Experiments

This section evaluates the proposed method when applied to the drilling task shown in Fig. **??**. Our system is formed by a Denso VS060 6-DoF industrial manipulator equipped with a commercial off-the-shelf hand drill. All benchmarks were executed in a system with Intel® Core™ i7 processor and 24 GB RAM, GeForce GTX 960M video card, running Ubuntu 16.04 (Xenial), 64 bits.

A. Benchmarking task-space TSP solvers

To solve the task-space TSP (Step 1 of our algorithm), one may use exact or near-optimal solvers. The choice depends on the trade-off between the available CPU time and the solution quality. Here we evaluate three TSP solvers.

1) *Exact*: Constraint Integer Programming (CIP) can be used to find true optimal TSP tour [5]. Here, we used the SCIP Optimization Suite [8] to implement an exact solver;
2) *2-Opt*: We re-implemented this simple, yet efficient, algorithm to find a near-optimal solution to TSP. The algorithm iteratively improves an initial guess by repeatedly replacing pairs of edges that cross over [5].
3) *RNN*: We re-implemented this algorithm, which consists in iteratively selecting the nearest neighbor as the next target to visit. This process is repeated until all the targets are visited [5]. One can do several *restarts* from different initial targets, and choose the tour with the lowest cost from all the restarts. The drawback of this method is that it tends to corner itself, which requires long edges to get back to unvisited targets.

Fig. 2 shows a benchmark of the three methods. We run the TSP solvers on task-space subsets of 25, 50, 100, 150, 200 random targets as well as on the total 245 targets. One can observe that the *2-Opt* solver yields high-quality tours (less than 5% of sub-optimality) with low CPU time usage (less than 1 s). As for the *Exact* solver, it is not practical for more than 150 targets. Therefore, for all the subsequent experiments, we shall use *2-Opt* as our near-optimal task-space TSP solver.

B. Benchmarking configuration-space metrics

The *configuration-space metric* that defines the edge cost in Step 2 of our algorithm is the key component for the overall performance of the method. Given two robot configurations $\boldsymbol{q}$ and $\boldsymbol{q}'$, the ideal cost of the edge $c^*(\boldsymbol{q}, \boldsymbol{q}')$ is the duration of a time-optimal collision-free trajectory between them. However, since there are thousands of such edges, running full-fledged motion planning algorithms (with collision checks) for every edge would not be tractable. Therefore, one must consider approximate metrics, which should be fast to compute, yet give a good prediction of the corresponding time-optimal collision-free trajectory duration. Here we evaluate three such metrics.

1) *Weighted Euclidean joint distance*: The cost $c(\boldsymbol{q}, \boldsymbol{q}')$ is estimated as the weighted L^2 norm:

$$c(\boldsymbol{q}, \boldsymbol{q}') := \sqrt{\sum_{k=1}^{\mathrm{DoF}} w_k (q'_k - q_k)^2}\,,$$

where w_k is a positive weight for joint k. The weights are chosen in proportion to the maximum possible distance (Euclidean distance in the task-space) traveled by any point on the robot, when moving along the corresponding joint. Similar to [9], in our experiments this metric outperforms consistently the Euclidean joint distance.
2) *Maximum joint difference*: The cost $c(\boldsymbol{q}, \boldsymbol{q}')$ is estimated as follows:

$$c(\boldsymbol{q}, \boldsymbol{q}') := \max_k \left| \frac{(q'_k - q_k)}{\dot{q}_k} \right| .$$

The intuition of this metric is to determine the maximum joint displacement when "moving" from $\boldsymbol{q}$ to $\boldsymbol{q}'$ by simply computing the joint difference, $(q'_k - q_k)$, over the joint k velocity limit, for $k \in [1, \ldots, \mathrm{DoF}]$. Then the maximum value is used;
3) *Linear trajectory interpolation*: the cost $c(\boldsymbol{q}, \boldsymbol{q}')$ is given by the duration of a trajectory obtained by linear interpolation. It only requires to specify the positions, $\boldsymbol{q}$ and $\boldsymbol{q}'$, and guarantees continuity at the position level subject to the robot velocity and acceleration bounds. Moreover, this metric does not consider obstacles which greatly reduces its computing time.

Fig. 3 shows the benchmarking results for the three proposed configuration-space metrics. One can observe that the *Maximum joint difference* metric takes the lowest CPU time and yields task execution times comparable to, in some cases even better than, the *Euclidean* and *Linear Interpolation metrics*. Therefore, for all the subsequent experiments, we shall use *Maximum joint difference* as our metric.

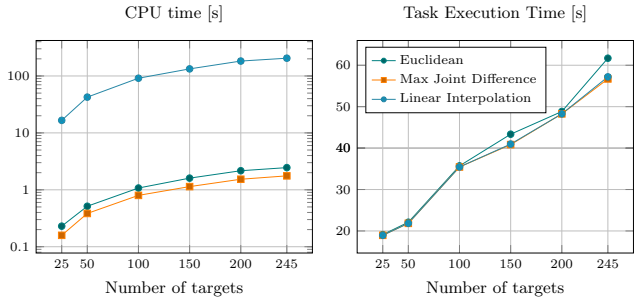

Fig. 3. Benchmarking results for three C-space metrics. (a) CPU times for different number of targets. The Y axis is logarithmic (b) Task execution time for different number of targets. The sets of targets are selected randomly. The last sample considers all the 245 targets.

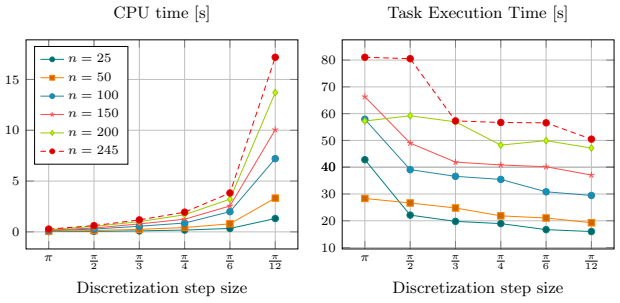

Fig. 4. Benchmarking results showing the effect of the discretization step size of the free DoF.

C. Benchmarking discretization step size for the free DoF

Many industrial tasks such as spot-welding, spay-painting or drilling involve less than 6 degrees of freedom. Therefore, a classical 6-DoF industrial robot has more joints than strictly required to execute such tasks. Specifically, the drilling task at hand involves 5 DoF, since the rotation θ about the drilling direction is irrelevant. One approach to tackle this redundancy can consist in setting a specific value for the irrelevant DoF: for instance $\theta \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$ for a $\frac{\pi}{2}$ discretization step size. For each of the discretized value of θ, we then have a full 6-DoF IK problem. To solve the full 6-DoF IK, we next use OpenRAVE's IKFast [10], which outputs all the IK solutions (here we have a "discrete" redundancy situation – think of the "elbow up" and "elbow down" configurations). We finally group all the IK solutions corresponding to all the discretized values of θ into a single list, which is the list of all IK solutions that will be considered for a given target. Table I gives the total number of IK solutions considered as a function of the discretization step size and of the number of targets.

TABLE I

NUMBER OF ROBOT CONFIGURATIONS DEPENDING ON THE DISCRETIZATION STEP SIZE AND THE NUMBER OF TARGETS

Step	**Number of targets**					
Size	**25**	**50**	**100**	**150**	**200**	**245**
π	209	462	1,049	1,540	2,059	2,486
$\frac{\pi}{2}$	329	723	1,569	2,311	3,100	3,770
$\frac{\pi}{3}$	452	1,053	2,231	3,297	4,425	5,409
$\mathbf{\frac{\pi}{4}}$	**630**	**1,369**	**2,850**	**4,238**	**5,727**	**6,975**
$\frac{\pi}{6}$	898	2,011	4,287	6,363	8,492	10,406
$\frac{\pi}{12}$	1,820	4,038	8,448	12,605	16,933	20,693

One can see that the choice of the *discretization step size* is governed by a trade-off between speed and optimality. Fig. 4 shows the computation time and task execution time as a function of the discretization step size. As expected, the computation time increases as the discretization step size decreases, but interestingly, the task execution time does not change significantly for step sizes below $\frac{\pi}{4}$, which thus yields a good trade-off between CPU time and task execution time. Therefore, for all the subsequent experiments, we shall use $\frac{\pi}{4}$ as our discretization step size.

D. Comparison to other methods

As none of the methods described in Section II provide public implementations, we were unable to reproduce their results and perform a fair comparison with the method herein presented. However, we can note that the computation times reported in previous works are several orders of magnitude higher than ours, yet for problem instances that are smaller [2] than what we consider in this work.

In the following, we compare our method to two of the existing methods, which we could re-implement.

1) *TSP in C-space* [11]: when only one configuration is considered per target, RTSP is reduced to a regular TSP in the configuration space. Here, for each target, we consider the IK solution with the best manipulability index [12];
2) *GLKH*: here the RTSP is formulated as a GTSP [13], [14], [15]. To solve that GTSP, we use the state-of-the-art GLKH solver [16] which makes use of the LKH heuristic [4].

Fig. 5 shows the comparison of our method (*RoboTSP*) to the two methods just described. While the *TSP in C-space* method has a similar running time as *RoboTSP* (indeed both run a TSP on the same number of targets), the time durations of the trajectories it produces are higher than those of *RoboTSP*, since it does not optimize the IK choice per target.

GLKH produces trajectories with similar total duration as *RoboTSP* but the computing time is higher several orders of magnitude.

V. CONCLUSIONS

We have proposed a method to determine a near-optimal sequence to visit n targets with multiple configurations per target, also known as the Robotic Task Sequencing Problem.

[2]The problem instance size is considered in terms of targets and configurations per target

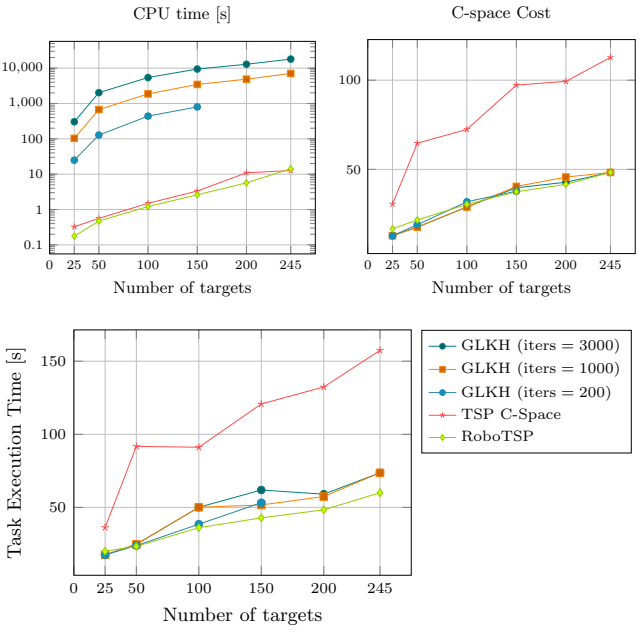

Fig. 5. Comparison of our method (*RoboTSP*) against other methods: i) Solved as TSP in configuration-space; ii) Solved as GTSP using GLKH.

For a complex drilling task, which requires visiting 245 targets with an average of 28.5 configurations per target, our method could compute a high-quality solution in less than a minute. To our knowledge, no existing approach could have solved the same problem in practical times. We have also provided a carefully benchmarked open-source software solution, which can be readily used in complex, real-world, applications such as drilling, spot-welding or spray-painting.

ACKNOWLEDGMENT

This work was supported in part by NTUitive Gap Fund NGF-2016-01-028 and SMART Innovation Grant NG000074-ENG.

REFERENCES

[1] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[2] Q.-C. Pham, "Trajectory Planning," in *Handbook of Manufacturing Engineering and Technology*. London: Springer London, 2015, pp. 1873–1887.

[3] G. A. Croes, "A Method for Solving Traveling-Salesman Problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

[4] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, oct 2000.

[5] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem : a Computational Study.* Princeton University Press, 2011.

[6] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[7] J. Meijer, Q. Lei, and M. Wisse, "Performance study of single-query motion planning for grasp execution using various manipulators," in *2017 18th International Conference on Advanced Robotics (ICAR)*. IEEE, 2017, pp. 450–457.

[8] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, jul 2009.

[9] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2000, pp. 521–528.

[10] R. Diankov, "Automated Construction of Robotic Manipulation Programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, aug 2010.

[11] Y. Edan, T. Flash, U. Peiper, I. Shmulevich, and Y. Sarig, "Near-minimum-time task planning for fruit-picking robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 48–56, 1991.

[12] T. Yoshikawa, "Manipulability of Robotic Mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, jun 1985.

[13] M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sánchez-Ante, "Planning Tours of Robotic Arms among Partitioned Goals," *The International Journal of Robotics Research*, vol. 25, no. 3, pp. 207–223, 2006.

[14] C. Wurll, D. Henrich, and H. Wörn, "Multi-Goal Path Planning for Industrial Robots," in *International Conference on Robotics and Application*, 1999.

[15] C. Wurll and D. Henrich, "Point-to-point and multi-goal path planning for industrial robots," *Journal of Robotic Systems*, vol. 18, no. 8, pp. 445–461, 2001.

[16] K. Helsgaun, "Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun Algorithm," *Mathematical Programming Computation*, vol. 7, no. 3, pp. 269–287, sep 2015.