Toward Fast and Optimal Robotic Pick-and-Place on a Moving Conveyor

Shuai D. Han   Si Wei Feng   Jingjin Yu

Abstract— Robotic pick-and-place (PNP) operations on moving conveyors find a wide range of industrial applications. In practice, simple greedy heuristics (e.g., prioritization based on the time to process a single object) are applied that achieve reasonable efficiency. We show analytically that, under a simplified telescoping robot model, these greedy approaches do not ensure time optimality of PNP operations. To address the shortcomings of classical solutions, we develop algorithms that compute optimal object picking sequences for a predetermined finite horizon. Employing dynamic programming techniques and additional heuristics, our methods scale to up to tens to hundreds of objects. In particular, the fast algorithms we develop come with running time guarantees, making them suitable for real-time PNP applications demanding high throughput. Extensive evaluation of our algorithmic solution over dominant industrial PNP robots used in real-world applications, i.e., Delta robots and Selective Compliance Assembly Robot Arm (SCARA) robots, shows that a typical efficiency gain of around 10%-40% over greedy approaches can be realized.

I. INTRODUCTION

We present a study aiming at developing fast algorithms for optimal robotic pick-and-place (PNP) on a moving conveyor. Modeling typical industrial robotic PNP scenarios, we examine the setting where a robotic arm is tasked to continuously pick up objects, one at a time, from a moving conveyor and drop them off at a fixed location. Based on our investigation, it would appear that greedy approaches had been used in practice because of the fast online nature of the task, which is explained in two aspects. First, estimating the poses of multiple objects requires advanced sensing techniques, whereas it is much easier to detect an object as it enters the scene (e.g., by using a laser scanner). Nowadays, however, computer vision algorithms are fast enough to accurately report the poses of many objects. Second, on a fast-moving conveyor, very limited computation can be done before an object becomes inaccessible.

In this letter, we first work with a simplified robot model to show analytically that commonly used greedy approaches do not produce time-optimal solutions in general. Then, we develop dynamic programming based algorithms capable of computing (near-)optimal solutions for tens to hundreds of objects under a second. Because the running time can be accurately bounded for a given number of objects, our algorithmic solution can be customized for real-time PNP operations. Extensive simulation studies on both simplified and practical robot models including Delta and SCARA (Selective Compliance Assembly Robot Arm) robots show that

S. D. Han, S. W. Feng, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. E-Mails: {shuai.han, siwei.feng, jingjin.yu} @ rutgers.edu. This work is supported by NSF awards IIS-1617744, IIS-1734419, and IIS-1845888.

our methods consistently yield about 10%-40% efficiency gain with respect to the number of objects that can be successfully picked.
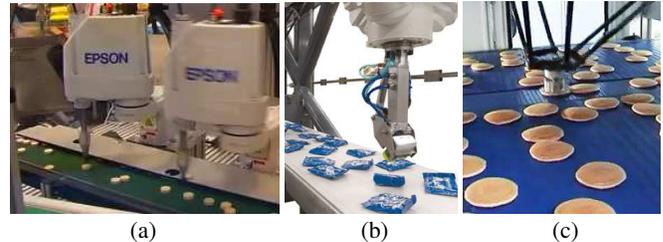

Fig. 1.   A few conveyor-based robotic PNP systems (a) Two Selective Compliance Assembly Robot Arm (SCARA) robots working on picking and placing machine parts (b) (c) Delta robots packing food items.

The invention and development of conveyor belt systems for material handling have revolutionized many industries over the years [1]. With advances in computer vision and robotic manipulation, conveyor-based robotic PNP solutions [2]–[6] have seen rapid adoptions that have yielded increasing levels of automation (see Fig. 1). The intrinsic goal in deploying such systems is to realize continuous and fast PNP operations. Therefore, a natural algorithmic question to ask here is how optimal is a given solution [7], [8] and how better algorithms may be designed to improve system throughput.

A fairly thorough fundamental algorithmic study of conveyor-based robotic PNP is carried out [9], where several polynomial-time approximation algorithms are provided for a variety of PNP problems. The work also pointed out that many such problems are at least NP-Hard [10], [11], given the similarity between robotic PNP and the traveling salesperson problem (TSP). This and other studies, e.g., [12], also link the robotic PNP problem to classical vehicle routing problems (VRP), which has many variations on its own [13]–[16]. We note that while polynomial-time approximation algorithms for PNP have been proposed [9], the algorithms optimize over metrics like L_1 and the approximately optimal solutions are not practical. The study also does not sufficiently consider robot geometry and dynamics, which are very important factors in real-world applications.

When it comes to *practically efficient* algorithmic solutions for PNP operations over a conveyor, the first proposed solutions resorted to a first-in first-out (FIFO) rule for prioritizing the object picking order [17], [18]. As pointed out, the FIFO heuristic can result in fairly sub-optimal solutions [19]. To address this, a job scheduling rule called shortest processing time (SPT) [20] was employed [19]. With further improvements, SPT and variants are shown to be consistently superior to FIFO. Since [19], research on PNP over conveyor appears to have shifted to using multiple robot arms to further

boost the throughput. Among these, non-cooperative game theory was explored [21] whereas FIFO and SPT heuristics are employed [22]. A recent approach combines randomized adaptive search with Monte Carlo simulation [23].

Contributions. The main contributions of this work are two. First, after observing and analytically characterizing sub-optimality of existing greedy PNP solutions, we develop a dynamic programming-based optimal finite-horizon PNP algorithm that applies to arbitrary robot models for which the dynamics can be simulated. Within a second, our algorithm is capable of computing optimal solutions for over 20 objects, which requires the exact processing of 20! possible picking sequences. With additional locality-based heuristics, we can compute near-optimal solutions for over 100 objects in under one second. Second, through extensive simulation study over typical industrial PNP robots (e.g., Delta and SCARA), we show that our algorithmic solutions are computationally efficient and outperform the existing state-of-the-art including FIFO and SPT variants by 10% to 40% in real-time settings. Such improvement is significant when it comes to real-world applications, where a few percentages of efficiency gain could provide a company a large competitive edge.

Organization. The rest of the letter is organized as follows. The problem setting studied in the letter is explained in detail in Section II. In Section III, we demonstrate the sub-optimality of greedy methods and estimate the maximum potential gain via optimization. Sections IV and V detail our algorithmic development, with Sections IV focusing on how to quickly obtain optimal PNP time for complex robots and Section V describing how we deal with the combinatorial explosion as we seek optimal finite-horizon solutions. A selection of our extensive evaluation effort of the algorithms is presented in Section VI, demonstrating the superior real-time performance of our proposed methods. We then conclude with Section VII.

II. PRELIMINARIES

Consider a robotic pick-and-place (PNP) system composed of a robot arm and a moving conveyor belt. Such systems [9], [20] are generally modeled as residing in a two-dimensional bounded rectangular workspace $\mathcal{W} \in \mathbb{R}^2$. Let the base of the robot arm be located at $(x_A, y_A = 0)$. We assume that the reachable area on the conveyor by the robot end-effector for PNP actions is an axis-aligned rectangle $\mathcal{W}$ with the lower left coordinate being $(x_L, y_B = 0)$ and upper right coordinate being (x_R, y_T) (see. Fig. 2). The task for the robot is to pick up objects located within $\mathcal{W}$ and drop them off at the origin $(x_D = 0, y_D = 0)$. We assume that the rest position of the end-effector is also at the drop-off location. Since the robot will execute a large number of PNP actions in a single run, this assumption has little effect on optimality. The robot is assumed to know all object locations within $\mathcal{W}$. The assumption that $y_A = y_B = y_D = 0$ is for convenience and has no effect on computational complexity and has negligible effects on solution optimality.

Without loss of generality, we assume that the conveyor belt moves at unit speed, e.g., $v_b = 1$, from the right toward
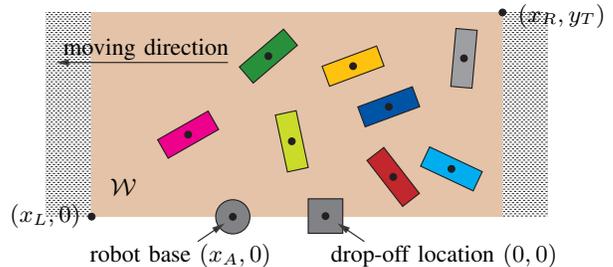

Fig. 2. Illustration of a conveyor workspace where the base of the robot arm is located at $(X_A, 0)$. The end-effector picks up objects within a region $\mathcal{W}$ with a lower left corner of $(x_L, 0)$ and an upper right corner of (x_R, y_T), and drops off objects at the drop-off location $(0,0)$.

the left. For the robot arm, we work with two types of motion models. In a *simplified model*, the end-effector of the arm is assumed to be able to extend or retract at a fixed speed $v_e > 1$. That is, the absolute speed of the end-effector along the straight line between the robot base and the end-effector location is v_e. In other words, the robot arm behaves like a telescoping arm. We use this model for the structural analysis as well as potions of the simulation studies.[1] Notice that in general, it is never beneficial for a robot to run at a lower speed.

For the study, it is assumed that the robot can pick up an object when its end-effector stops at the center of the object on the conveyor belt. The pickup action and the drop-off action are assumed to be instantaneous, i.e., they do not induce delays. We make such an assumption because the time involved in these actions is comparatively small in applications. Whereas we assume infinite acceleration and de-acceleration for the simplified telescoping robot model since it is a velocity based model, as already mentioned, dynamics are carefully considered for Delta and SCARA robots. The overall goal is then to execute as many PNP actions as possible in a given amount of time.

In developing the algorithms, we work with two object distribution models. Under a *one-shot* setting, we fix the dimensions of $\mathcal{W}$ and the number of objects n, and let the n objects be uniformly distributed in a subset of $\mathcal{W}$. More precisely, the objects are spawned in a rectangular area with the same y span as $\mathcal{W}$ and also the same maximum reach on the x axis, i.e., both end at $x = x_R$ on the right. The left end of the object spawning area has an x value larger than x_L because otherwise, objects appearing close to x_L may immediately move out of $\mathcal{W}$ on the conveyor, rendering it impossible to pick them. Under a *continuous* setting, which models after real application setups, the objects, following a some spatio-temporal distribution, appear at $x \geq x_R$ continuously for a period of time. For example, the distribution may be a Poisson process with rate λ followed by a uniform distribution of $y \in (0, y_T)$. That is, as a new event is generated by the Poisson process, a new object is placed at (x_R, y) where $y \in [0, y_T]$ is uniformly selected.

---

[1] We note that the algorithms we develop directly apply to Delta and SCARA robots that are dominant in relevant industrial applications. We use *accurate models* in Sections IV for Delta and SCARA robots that consider both robot geometry and dynamics with bounded acceleration.

Our proposed methods will be compared with greedy approaches, namely, FIFO and SPT [19], which pick objects following simple heuristics. FIFO follows the first-in first-out rule and always picks the object which enters the workspace the earliest, i.e., with the smallest x location [19]. On the other hand, SPT always picks the object with the smallest PNP time, following the *shortest processing time* rule [19]. In addition, we add EUCLIDEAN which uses the Euclidean distance between an object's location and the drop-off location instead of x for prioritizing. It is clear that these approaches require little computational effort.

III. ANALYSIS OF THE OPTIMAL SOLUTION STRUCTURE

As greedy approaches (e.g., SPT, FIFO [19]) work with a very short horizon, it can be expected that they are generally sub-optimal. It appears, however, no quantitative analysis has been performed in the literature to study this sub-optimality. Thus, we begin our study with an analytical characterization on the benefit of using a longer horizon for optimization.

A. Non-Optimality of Greedy Strategies

As mentioned in the introduction, the most commonly used heuristics for PNP appear to be FIFO and SPT [19]. Whereas these best-first like heuristics runs in $O(n)$ time for selecting a single picking candidate with n being the number of objects accessible on the conveyor, the overall solution is generally sub-optimal in terms of efficiency over long term. We now establish this under a fairly general setting through examining the PNP of two objects o_1 and o_2 located at (x_1, y_1) and (x_2, y_2), respectively (Fig. 3). For the analysis, we further assume that $x_A = x_D = 0$, i.e., the robot base and drop off location are the same (we note that the analysis that follows readily generalizes beyond this assumption).
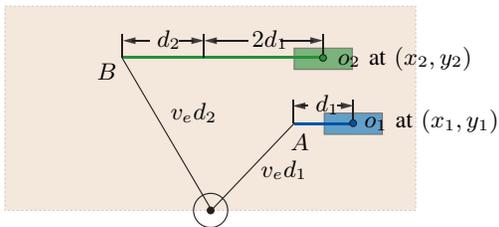

Fig. 3. The relevant distances when object o_1, initially located at (x_1, y_1), is picked up first by the robot.

Under the setup, we can compute the time it takes the end-effector to carry out PNP actions on the two objects for the two possible picking orders (i.e., first picking the object o_1 or first picking the object o_2). Assuming that object o_1 is picked first, we can compute the earliest location A where the end-effector can pick up the object. At A, object o_1 has traveled some distance d_1; the end-effector, with speed v_e, would have traveled a distance of $v_e d_1$. We then have a single unknown d_1 and the quadratic equation $(x_1 - d_1)^2 + y_1^2 = v_e^2 d_1^2$, with which we can solve for d_1. We omit the solution here, which is lengthy to write down. Based on d_1 and similar reasoning, we can compute the point B where the end-effector can pick up the second object after dropping the first object at the origin. At this point, the second object would have traveled a

distance of $2d_1 + d_2$ for some d_2. The equation for computing d_2 is readily obtained as $(2d_1 + d_2 - x_2)^2 + y_2^2 = v_e^2 d_2^2$.

The total time required for handling both objects this way is $2(d_1 + d_2)$ since the conveyor runs at unit speed. We denote this time as $t_{12}(x_1, x_2, y_1, y_2, v_e)$. Similarly, we may compute the time required if o_2 is picked up first; denote the time as $t_{21}(x_1, x_2, y_1, y_2, v_e)$. It can be shown that, setting $x_1 = x_2$, there is no general dominance between t_{12} and t_{21}.

Proposition III.1. *For two objects o_1 and o_2 initially located at (x, y_1) and (x, y_2), the optimal pick-and-place sequence of the objects depends on the horizontal offset x.*

Proof. We define a function δt as
$$\delta t(x_1, x_2, y_1, y_2, v_e) = t_{12}(x_1, x_2, y_1, y_2, v_e)$$
$$-t_{21}(x_1, x_2, y_1, y_2, v_e).$$
To prove the proposition, we only need to show that for some fixed y_1, y_2, and v_e, varying $x = x_1 = x_2$ will flip the sign of the function $\delta t(\cdot)$. For this purpose, we let $y_1 = 0.4, y_2 = 0.7$ and $v_e = 2$ and examine
$$f(x) = \delta t(x, x, 0.4, 0.7, 2).$$
Solving for $f(x) = 0$ with the restriction of $x > 0$ yields a single solution $x_0 \approx 0.65$. This means that when $x > x_0$ holds, it is more optimal to pick o_2 first. When $x < x_0$, it is more optimal to pick o_1 first. $\square$

Continuing from the proof of Proposition III.1, if we plot $f(x)$ over $x \in [0.4, 1.4]$, Fig. 4 is obtained which clearly shows that in this case, picking o_1 first is only better when initial x is less than 0.65. It also shows that δt can have relatively large positive and negative values, meaning that the conclusion of Proposition III.1 holds for proper $x_1 \neq x_2$ and also for $x_A \neq x_D$ when other conditions are proper.
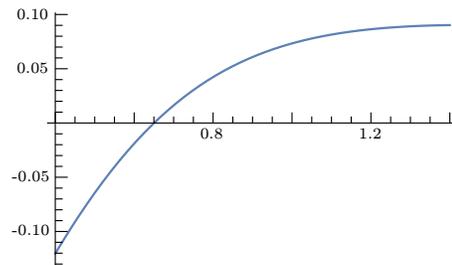

Fig. 4. Plot of $f(x) = \delta t(x, x, 0.4, 0.7, 2)$ for $x \in [0.4, 1.4]$.

Practical Concerns. As one might expect, as indicated by the SPT rule, it is generally better if the closer object is picked first. In the example from the proof of Proposition III.1, the maximum value of $f(x)$ is reached when $x \approx 1.45$, which yields $f(1.45) \approx 0.09$. The value of t_{21} in this case is approximately 0.77 (therefore, $t_{12} \approx 0.86$). That is, picking o_2 first in this case will lead to at most an optimality loss of $0.09/0.77 \approx 12\%$. On the other hand, always picking o_2 first can lead to an optimality loss of about 40% when $x \approx -0.1$. Nevertheless, a 12% of optimality loss is very significant and should be avoided in practice whenever possible. Lastly, since the analysis is based on $x_1 = x_2$, it directly applies to the FIFO setting as well, where the more sub-optimal choice of picking o_2 first can happen if x_1 is just slightly larger than x_2.

B. Structure of Optimal PnP Solutions

The analysis and resulting observation from Section III-A leads us to develop optimal PnP solutions via exhaustive search (see Section V-A). Through running the exhaustive search algorithm, we computed optimal solutions under various configurations and studied the distribution pattern of the optimal picking sequences. A typical outcome under the one-shot setting is illustrated in Fig. 5. In the figure, the small discs correspond to the initial locations of objects for 100 problems with $n = 10$ each, with $2 \le x \le 8, 0 \le y \le 3$, and $v_e = 5$. After computing the optimal solution for each of the 100 problems, we color the object that is picked first (out of the 10 objects in a problem instance) dark red and the last picked object dark blue. The colors for the other eight objects are interpolated between these two. The majority of objects are picked before their x coordinates fall below $x = -2$.
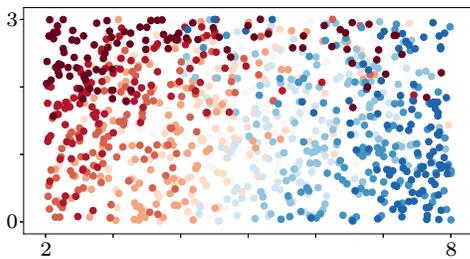

Fig. 5.   Illustration of picking orders of 10 objects in 100 optimal solutions.

From the figure, we make the observation that the first few objects that get picked are concentrated toward the left, though a few are also from the far right. Generally, however, they all have relatively large y coordinates. The last few objects, on the other hand, fall more on the far right and are not concentrated in terms of the y coordinate. The objects that are picked in the middle in an optimal sequence tend to fall in the middle, which more or less is as expected.

IV. Computation of Shortest PnP Time

A significant challenge in the design and implementation of object picking sequence selection algorithms is how to deal with the geometry and dynamics of the robots (see, e.g., [24]) that are involved. We encapsulate the complexity caused by robot geometry and dynamics in a routine, GETPNPTIME, that returns the best available PnP time for a given robot model and the initial location of the moving object to be picked up. This is achieved through a two-step process. First, a principled method is designed for estimating a single optimal PnP time. Second, we build a table of pre-computed PnP times to enable real-time look-up in practice.

A. Computing Shortest PnP Time for Simple Robots

If the robot has trivial dynamics (note that this is a BIG if that almost never happens in practice), it may be possible for GETPNPTIME to compute the PnP time directly and analytically. In the case of the simplified telescoping robot, we may do so via solving the quadratic equation

$$(\sqrt{x_A^2 + y_A^2} \pm v_e t)^2 = [(x - v_b t) - x_A]^2 + (y - y_A)^2.$$

The sign of $v_e t$ depends on whether the arm extends or retracts, which directly correlates to whether the object's

current location (x, y) is in the circle centered at (x_A, y_A) with radius $\sqrt{x_A^2 + y_A^2}$ (see Fig. 6). In this case, the arm extension and retraction take the same amount of time.
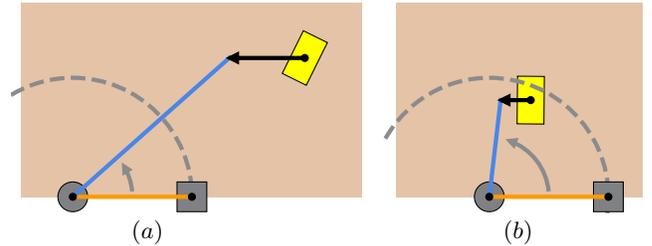

Fig. 6.   Scenarios when robot (a) extends and (b) contracts the arm. The orange and blue lines illustrate the drop-off and pick-up arm poses, respectively. the arced arrows show the rotation movements of the arm.

B. Computing Shortest PnP Time for Complex Robots

Computing optimal PnP time is hard in general as most robots have complex, interacting geometric constraints and physical constraints including robot kinematics, speed/acceleration limits, and so on. For a given robot model, e.g., SCARA, we first need a method for computing the optimal (shortest) time it takes for the end-effector to reach a point (x, y) within the robot's workspace and then the optimal time for the end-effector to return to the drop-off location. The sum of the two times is the optimal total PnP time. In practice, optimal PnP times are estimated [24].

Our implementation of the shortest PnP time computation for a single 2D point is as follows. First, based on the robot's geometric structure, we compute the joint angles of the robot (two for SCARA and three for Delta [24], [25]) at the initial (drop-off) end-effector location and the target pick-up location (x, y). Then, we invoke the Reflexxes Motion Library [26] to obtain an estimated shortest transition time from the drop-off pose to the pick-up pose and also the shortest transition time from the pick-up pose to the drop-off pose, which may be different. We denote this time as $t(x, y)$, from which we can readily obtain $(x + v_b t(x, y), y)$ as the location where the object is before the end-effector starts the PnP operation.

Because each computation of $t(x, y)$ can be relatively time consuming (easier for SCARA and slightly more involved for Delta with 3 degrees of freedom), the procedure cannot directly be used for real-time robot operations. Instead, we build a table of pre-computed PnP times at a given resolution (in this work, a 100×100 discretization is used, with interpolation), with which GETPNPTIME can then be realized extremely efficiently with very high precision.

C. Visualizing Typical PnP Time Profiles

The GETPNPTIME subroutine can be readily adapted to work with other robots. That is, GETPNPTIME is an abstraction layer that isolates the object picking sequence selection from physical robot models. It is clear that different robots can have significantly different PnP time structure. For the three robots that are examined in this work, their PnP time profiles are shown in Fig. 7. We note that the (rotated) profiles are slightly truncated at the bottom. Similarly, there

are some values missing at the top of the figures; this is because objects initially located in these areas will exit workspace before the arm can reach them.
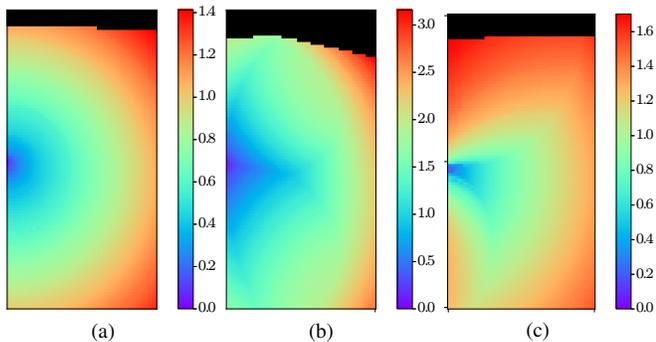

Fig. 7. The (relative) time profile for PNP operations for difference arms. The workspace is rotated 90 degrees clockwise. (a) The simplified telescoping robot. (b) The Delta robot. (c) The SCARA robot.

V. EXACT AND APPROXIMATE ALGORITHMS FOR SELECTING THE BEST PICKING SEQUENCE

The main algorithm developed in this work is an *exhaustive-search*-based method which checks all possible object picking sequences to find the optimal one. In addition, a *local-augmentation*-based method is developed to further boost computational efficiency.

A. Exhaustive Search Methods: OPTSEQ and OPTSEQDP

With the GETPNPTIME routine, a baseline exhaustive search routine is straightforward to obtain. We call such a routine OPTSEQ, which computes the optimal object picking sequence for a given horizon (i.e., number of objects examined at a time). Then, *dynamic programming* is applied to speed up OPTSEQ, yielding the routine OPTSEQDP, which is significantly faster yet without any loss of optimality.

1) OPTSEQ: As it is shown in Alg. 1, OPTSEQ iterates through all permutations of objects (line 2) and finds the picking sequence with the minimum execution time (lines 5–5). The computation time of OPTSEQ is $O(n!n)$, since there are $n!$ permutations to check and for each permutation, the algorithm calls GETPNPTIME n times.

Algorithm 1: OPTSEQ

Input: objects' initial location $(x_1, y_1), \ldots, (x_n, y_n)$
Output: S^*: a time-optimal PNP sequence
1 $t^* \leftarrow \infty, S^* \leftarrow$ none
2 **for** $P \in$ ALLPERMUTATIONS($\{1, \ldots, n\}$) **do**
3      $t \leftarrow 0$
4   **for** $i \in P$ **do** $t \leftarrow t +$ GETPNPTIME $(x_i - v_b t, y_i)$
5   **if** $t < t^*$ **then** $t^* \leftarrow t, S^* \leftarrow P$
6 **return** S^*

---

2) OPTSEQDP: Clearly, OPTSEQ contains redundant computation. For example, for four objects, the time for first picking up objects $(1, 2)$ is calculated twice, during the computation for sequences $(1, 2, 3, 4)$ and $(1, 2, 4, 3)$. To avoid such redundant calculations, we propose OPTSEQDP, a dynamic programming algorithm similar to that in [27]. The pseudo code for OPTSEQDP is provided in Alg. 2. In line 1, two datasets are initialized: S which will contain the time-optimal picking sequences of all the 2^n subsets of

objects, and T which will contain the associated time costs. An n-step iterative process starts from line 2. In line 4, the algorithm updates T amd S for each k-combination U. The update process iterates through all objects $i \in U$, finds the one that minimizes the execution time when picked last:

$$T[U] = \min_{i \in U} \{ T[U \setminus \{i\}] +$$
$$\text{GETPNPTIME}(x_i - v_b T[U \setminus \{i\}], y_i) \}.$$

During this process, $S[U]$ is also updated accordingly to store the subsets' optimal picking sequence. Finally, in line 5, a time-optimal PNP sequence of all n objects is returned.

Algorithm 2: OPTSEQDP

Input: objects' initial location $(x_1, y_1), \ldots, (x_n, y_n)$
Output: a time-optimal PNP sequence
1 $T = \{\varnothing : 0\}, S = \{\varnothing : ()\}$
2 **for** $1 \leq k \leq n$ **do**
3   **for** $U \leftarrow$ ALLCOMBINATIONS($\{1, \ldots, n\}, k$) **do**
4   UPDATE(T, S, U)
5 **return** $S[\{1, \ldots, n\}]$

Proposition V.1. OPTSEQDP *finds the optimal* PNP *sequence.*

Proof. Since it is trivial that S contains the optimal picking sequence of the 0-combination of the objects, it suffices to show that given the optimal sequences of all $(k - 1)$-combinations, then the function UPDATE(T, S, U) in line 4 calculates the optimal sequences of all k-combinations.

Given U as an arbitrary k-combination, its optimal picking sequence must also be optimal when picking the first $k - 1$ objects in this sequence. The update process of $T[U]$ checks all candidate sequences with the first $k - 1$ objects picked in a time-optimal manner. □

OPTSEQDP runs in $O(2^n n)$: there are $O(2^n)$ object subsets, and processing a combination U calls the routine GETPNPTIME $|U|$ times, taking $O(n)$ time. Since $2^n \ll n!$ for large n, OPTSEQDP is much faster than OPTSEQ.

B. A Local Augmentation Method: SUBOPTDP

While working on OPTSEQDP, we attempted many heuristics to further boost its efficiency. Here, we report a particularly effective method that appears to achieve optimality close to OPTSEQDP but scales much better. We call this local augmentation-based method SUBOPTDP, which uses OPTSEQDP as a subroutine. The pseudocode of SUBOPTDP is provided in Alg. 3. In line 1, SUBOPTDP starts with an initial picking sequence S, which can be selected in many different ways, for example, using FIFO. Then, line 2-9 repeatedly call OPTSEQDP over sub-sequences of S to reduce the execution time. Specifically, the algorithm has two parameters m_1 and m_2. The main loop is repeated m_1 times, and for each iteration, we call OPTSEQDP over the kth to $(k + m_2)^{\text{th}}$ elements of S for $1 \leq k \leq n - m_2$.

The computation time for SUBOPTDP is $O(m_1 n 2^{m_2} m_2)$. In our implementation, we found that initializing S using FIFO and assigning $m_1 = n, m_2 = 9$ produce solutions that

Algorithm 3: SUBOPTDP

Input: objects' initial location $(x_1, y_1), \ldots, (x_n, y_n)$
Output: a near-optimal PNP sequence

1   $S \leftarrow$ GETINITIALPICKINGSEQUENCE()
2   **for** m_1 *times* **do**
3     $t \leftarrow 0$
4   **for** $1 \le k \le n - m_2$ **do**
5       $O \leftarrow \varnothing$
6   **for** $i \in S[k : k + m_2]$ **do**
7         $O \leftarrow O \cup \{(x_i - v_b t, y_i)\}$
8       $S[k : k + m_2] \leftarrow$ OPTSEQDP (O)
9       $t \leftarrow t +$ GETPNPTIME $(x_{S[k]} - v_b t, y_{S[k]})$
10   **return** S

---

are often indistinguishable from these computed by OPTSEQDP: in all test cases, the average performance difference between SUBOPTDP and OPTSEQDP never exceeds 0.05%.

Remark (Adapting to a conveyor setting). Since it is expected that a conveyor will run without stopping for extended periods of time, for the continuous setting, OPTSEQDP or SUBOPTDP are invoked repeatedly with real-time locations of all the pick-able objects in the workspace.

VI. EXPERIMENTAL STUDIES

We performed an extensive evaluation of the newly developed algorithms. In this section, we present a small subset of the evaluation that is most representative. In Section VI-A, we measure the computation time of the algorithms under the one-shot setting and conclusively show that our algorithms are sufficiently fast for industrial applications. In Section VI-B, we focus on the one-shot setting and check how much execution time savings are possible. Selected results demonstrate that the projected efficiency gain of our algorithms are very significant across different robot models. Finally, in Section VI-C, we evaluate the performance of the SCARA robot in realistic continuous conveyor settings under two different object arrival distribution models. Again, our proposed algorithm shows a clear lead.

All algorithms are implemented in C++, and all experiments are executed on an Intel® Xeon® CPU at 3.0GHz.

A. Computational Efficiency

Fig. 8 shows the computation time of our original algorithms versus the number of objects n. With dynamic programming, we can solve much larger problem instances (near-)optimally: in around one second, OPTSEQ, OPTSEQDP, SUBOPTDP can solve one-shot problems with 10, 22 and 100 objects, respectively. We note that, while the test is done over the simplified robot model, similar performance is observed for Delta and SCARA robots. The shape of SUBOPTDP is due to the choice of parameters (i.e. $m_2 = 9$).

Our observation indicates that the active workspace in a conveyor PNP system contains a few to low tens of objects. From the figure, we observe that both OPTSEQDP and SUBOPTDP can complete a single sequence computation for ten objects within 10^{-4} seconds and fifteen objects with 10^{-2} seconds. Because, Delta and SCARA-based PNP systems generally do not pick more than a single digit number of
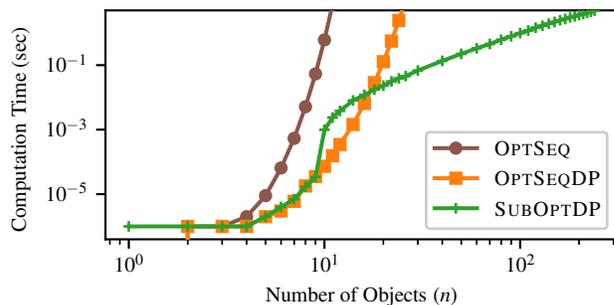

Fig. 8. Average computation time of our algorithms versus the number of objects n. In all experiments, the error is within $\pm 2\%$.

objects per second, OPTSEQDP and SUBOPTDP impose negligible time overhead. As such, they are sufficiently fast for the target industrial applications.

B. PNP *Performance under One-Shot Setting*

Having shown that OPTSEQDP and SUBOPTDP are sufficiently fast, next, we present their performance in one-shot settings where only a single batch of objects are handled. In a typical setting, we let $x_L = -5, x_R = 5, y_B = 0$, and $y_T = 5$, i.e., the workspace is a 10×5 rectangle. All objects are initially uniformly randomly placed between $x = 3$ and $x = 5$. Robots are configured so that they can reach anywhere within the workspace but are forbidden to reach outside. We first present Fig. 9, which illustrates the relative total PNP time of different algorithms. For results in the top two figures, since the number of objects is comparatively small, a faster relative conveyor belt speed is used ($5\times$ of that used in the bottom two figures). The parameters of the robots are set so that all algorithms can successfully pick all objects (for each figure, only a single set of robot parameters are used). Our selection of the robot model is somewhat arbitrary because we observe some but no substantial difference between different robot models. Therefore, we decided to select a diverse set of results given the limited available space.
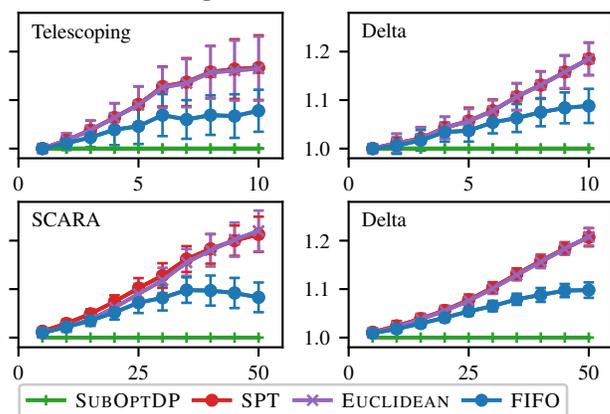

Fig. 9. Execution time ratios of various algorithms as compared with SUBOPTDP. The y-axes are the ratio and the x-axes are the number of objects used in a given run. Each data point is an average over 100 randomly generated instances. Standard deviations are plotted as error bars.

From the figure, we observe that SUBOPTDP (and therefore OPTSEQDP, which is at least as fast as SUBOPTDP) yields significant savings in PNP execution time. For example, for a SCARA robot, if we expect the workspace to have

about ten objects at a time, then FIFO, EUCLIDEAN, and SPT are expected to spend around 10% to 20% more PNP execution time as compared to our proposed solutions.

We also observe that there does not seem to be an upper bound on the ratios as the number of objects increases. Though it appears that the ratio for FIFO is tapering off in some of the figures, adding more objects shows that these ratios will eventually grow again. Though this may not be highly relevant in practice as the the number of objects is not likely to exceed a few tens, the phenomenon is structurally interesting. Our interpretation is that the behavior is perhaps caused by the optimal object picking sequence problem is similar in structure as hard TSPs [28] where polynomial time constant factor approximations are provably impossible.

We also evaluated the impact of different workspace settings with the result given in Fig. 10 for the Delta robot model (again, other robot models yield similar results). We note that the execution time here is relative but provides a meaningful comparison between different workspace settings. 10 objects are used for each run, which are randomly allocated between $x = 3$ and $x = 5$. We conclude that the impact of workspace appears to be small.
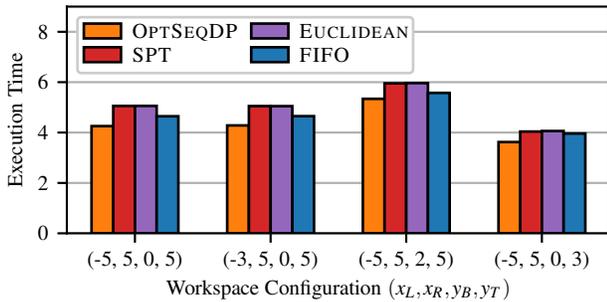
Fig. 10. The execution time for PNP operations on 10 randomly placed objects. Each bar is obtained as an average over 100 runs.

C. PNP *Performance under Continuous Setting*

After the one-shot setting, we examined a more realistic setting and evaluated how the algorithms perform when the conveyor runs for an extended period of time. For this setting, we fixed the conveyor speed and robot parameters so that there are generally a few objects on the conveyor within the workspace, mimicking practical settings. We attempted two distributions with which objects are placed on the conveyor: Poisson and uniform. For all experiments, we sample $n = 10000$ objects and used a workspace with $x_L = -5, x_R = 5, y_B = 0$, and $y_T = 5$. The SCARA robot model is used here for two reasons: *(i)* SCARA is the most widely used industrial PNP robot and *(ii)* the performance of SCARA is similar to Delta and the simplified telescoping model.

For the Poisson setting, a Poisson process with parameter $\lambda > 0$ is started at time $t_0 = 0$. Each time an event is triggered by the process at time $t \geq t_0$ (including at $t = 0$), we sample the uniformly sample (y_B, y_T) to get a y value. An object is then placed at (x_R, y) at time t. For the uniform setting, we sample n points in the unit square and the scale the unit square to have the same height as the workspace. We then adjust the length of the unit square to simulate how densely the objects are placed on the conveyor belt. At $t_0 =$

0, the left side of the scaled unit square is aligned with the line segment between (x_R, y_B) and (x_R, y_T).

In each experiment, we record the total number of objects that can be successfully picked up before some leave the left side of the workspace on the conveyor. The results are then scaled as ratios divided by $n = 10000$ and the data is visualized in Fig. 11 and Fig. 12.
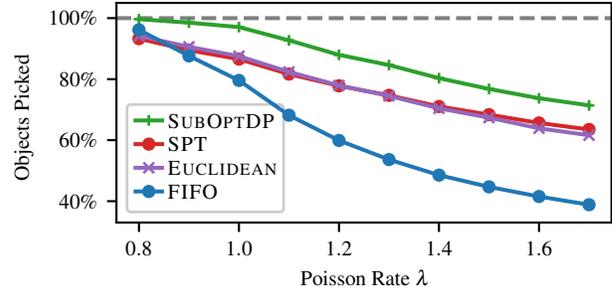
Fig. 11. Percentage of objects picked up out of 10000 using different algorithms. The object distribution is generated by 1D uniform distribution driven by a Poisson process with different Poisson rate λ. We mention that the absolute value of λ does not bear much significance.

Looking at Fig. 11, except when the Poisson rate λ is sufficiently low so that almost all objects can be picked up using any method, SUBOPTDP maintain a lead of around 10% as compared with SPT and EUCLIDEAN. The lead over FIFO is as large as 40%. We also point out that, unlike the one-shot case, FIFO generally performs the worst, though it works better than SPT and EUCLIDEAN initially. This is as expected since FIFO does the least amount of optimization. In the one-shot setting, requiring that all objects can be picked up benefited FIFO since it never let an object travel too far to the left side of the workspace. On the other hand, when there are too many objects, FIFO suffer since it attempts to catch objects that can be very far on the left side of the workspace, inducing penalty.
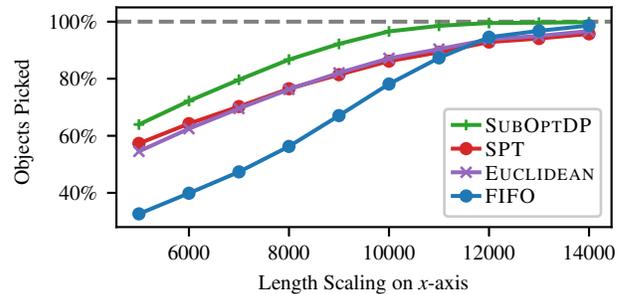
Fig. 12. Percent of objects picked up out of 10000 using different algorithms. The object distribution is generated by uniform distribution over the unit square. The height is then scaled to 5 and length is scaled as indicated in the figure, from 5000-14000.

For the uniform setting, e.g., Fig. 12, we observe a similar outcome as the Poisson setting. The figure looks different because larger scaling of the length of the unit square means sparser object placement and thus an easier setting, whereas larger λ values suggesting denser object placement.

From the experiments, we conclude that OPTSEQDP and SUBOPTDP provide significant performance gain as compared with FIFO, SPT, and EUCLIDEAN.

An illustration video of the proposed algorithms is provided at https://youtu.be/bIomJzjKXyc.

VII. Conclusion and Discussions

Seeing that the state-of-the-art for robotic PNP on a moving conveyor from the literature generally take a greedy approach in deciding the object picking order, we set out to explore how sub-optimal such approaches may be and ways to improve them. Using a simplified telescoping robot model, we show that greedy methods may lead to the loss of 12% to 40% execution time efficiency. To address the shortcomings of greedy methods, we propose the exhaustive OptSeq algorithm that computes optimal picking sequences for a finite look-ahead horizon. The algorithm is then further enhanced with dynamic programming and heuristic techniques to yield OptSeqDP and SubOptDP that are sufficiently efficient for practical setting. In doing so, we also establish a principled method for computing shortest PNP time profiles for complex Delta and SCARA robots, which have highly involved geometric and dynamic constraints.

Extensive simulation-based experimentation indicates that OptSeqDP and SubOptDP fully realize the goal we set out to achieve, as reflected in three aspects: *(i)* both algorithms are computationally efficient for real-time PNP applications, *(ii)* in the one-shot setting, our algorithms deliver up to 20% saving in execution time as compared to FIFO, Euclidean, and SPT, and *(iii)* in realistic PNP operations on continuously running conveyors, our algorithms show 10-40% advantage in terms of the number of picked objects. The magnitude of the efficiency gain has significant practical implications; a few percentage of difference in efficiency can separate success from failure. We observe no substantial difference in performance of our algorithms as we switch between robot models (i.e., telescoping, Delta, and SCARA). We conclude that OptSeqDP and SubOptDP could potentially make sizable impact to industrial PNP systems.

In future work, we would like to further explore two directions. First, a natural next step is to develop algorithms for the collaboration among multiple robots to further enhance overall system throughput, which appears to require more carefully object selection across multiple robots. As a second direction, we are in the process of integrating our algorithms on some real robot hardware, with the hope of bringing our methods one step closer to real-world applications.

References

[1] D. Hounshell, *From the American system to mass production, 1800-1932: The development of manufacturing technology in the United States*. JHU Press, 1985, no. 4.

[2] B. Mirtich, Y. Zhuang, K. Goldberg, J. Craig, R. Zanutta, B. Carlisle, and J. Canny, "Estimating pose statistics for robotic part feeders," in *Proceedings of IEEE international conference on robotics and automation*, vol. 2. IEEE, 1996, pp. 1140–1146.

[3] B. Carlisle, K. Goldberg, A. Rao, and J. Wiegley, "A pivoting gripper for feeding industrial parts," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 1650–1655.

[4] G. C. Causey, R. D. Quinn, N. A. Barendt, D. M. Sargent, and W. S. Newman, "Design of a flexible parts feeding system," in *Proceedings of International Conference on Robotics and Automation*, vol. 2. IEEE, 1997, pp. 1235–1240.

[5] D. R. Berkowitz and J. Canny, "Designing parts feeders using dynamic simulation," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1996, pp. 1127–1132.

[6] K. Goldberg, J. Craig, B. Carlisle, and R. Zanutta, "Estimating throughput for a flexible part feeder," in *Experimental Robotics IV*. Springer, 1997, pp. 486–497.

[7] G. C. Causey, R. D. Quinn, and M. S. Branicky, "Testing and analysis of a flexible feeding system," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 4. IEEE, 1999, pp. 2564–2571.

[8] M. S. Branicky, G. C. Causey, and R. D. Quinn, "Modeling and throughput prediction for flexible parts feeders," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 154–161.

[9] P. Chalasani, R. Motwani, and A. Rao, "Algorithms for robot grasp and delivery," in *2nd International Workshop on Algorithmic Foundations of Robotics*. Citeseer, 1996.

[10] J. K. Lenstra and A. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[11] C. S. Helvig, G. Robins, and A. Zelikovsky, "Moving-target tsp and related problems," in *European Symposium on Algorithms*. Springer, 1998, pp. 453–464.

[12] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps," *The International Journal of Robotics Research*, p. 0278364918780999, 2018.

[13] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: a classification scheme and survey," *Top*, vol. 15, no. 1, pp. 1–31, 2007.

[14] N. Christofides and S. Eilon, "An algorithm for the vehicle-dispatching problem," *Journal of the Operational Research Society*, vol. 20, no. 3, pp. 309–318, 1969.

[15] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation algorithms for some routing problems," in *17th annual symposium on foundations of computer science (sfcs 1976)*. IEEE, 1976, pp. 216–227.

[16] K. Treleaven, M. Pavone, and E. Frazzoli, "Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2261–2276, 2013.

[17] T.-Y. Li and J.-C. Latombe, "On-line manipulation planning for two robot arms in a dynamic environment," *The International Journal of Robotics Research*, vol. 16, no. 2, pp. 144–167, 1997.

[18] G. Pardo-Castellote, S. A. Schneider, and R. Cannon, "System design and interfaces for intelligent manufacturing workcell," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1995, pp. 1105–1112.

[19] R. Mattone, M. Divona, and A. Wolf, "Sorting of items on a moving conveyor belt. part 2: performance evaluation and optimization of pick-and-place operations," *Robotics and Computer-Integrated Manufacturing*, vol. 16, no. 2-3, pp. 81–90, 2000.

[20] L. Schrage, "Letter to the editora proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[21] H. I. Bozma and M. Kalalıoğlu, "Multirobot coordination in pick-and-place tasks on a moving conveyor," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 4, pp. 530–538, 2012.

[22] S. Daoud, H. Chehade, F. Yalaoui, and L. Amodeo, "Efficient meta-heuristics for pick and place robotic systems optimization," *Journal of Intelligent Manufacturing*, vol. 25, no. 1, pp. 27–41, 2014.

[23] Y. Huang, R. Chiba, T. Arai, T. Ueyama, and J. Ota, "Robust multi-robot coordination in pick-and-place tasks based on part-dispatching rules," *Robotics and Autonomous Systems*, vol. 64, pp. 70–83, 2015.

[24] D. Carp-Ciocardia *et al.*, "Dynamic analysis of clavel's delta parallel robot," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 3. IEEE, 2003, pp. 4116–4121.

[25] R. Clavel, "A fast robot with parallel geometry," in *Proc. Int. Symposium on Industrial Robots*, 1988, pp. 91–100.

[26] P. Zsombor-Murray, "Descriptive geometric kinematic analysis of Claveĺs "delta" robot," *Centre of Intelligent Machines, McGill University, USA*, 2004.

[27] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial and Applied mathematics*, vol. 10, no. 1, pp. 196–210, 1962.

[28] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the ACM (JACM)*, vol. 23, no. 4, pp. 665–679, 1976.