Robust 2D Assembly Sequencing via Geometric Planning with Learned Scores

Tzvika Geft[1], Aviv Tamar[2], Ken Goldberg[3] and Dan Halperin[1]

***Abstract*— To compute robust 2D assembly plans, we present an approach that combines geometric planning with a deep neural network. We train the network using the Box2D physics simulator with added stochastic noise to yield robustness scores – the success probabilities of planned assembly motions. As running a simulation for every assembly motion is impractical, we train a convolutional neural network to map assembly operations, given as an image pair of the subassemblies before and after they are mated, to a robustness score. The neural network prediction is used within a planner to quickly prune out motions that are not robust. We demonstrate this approach on two-handed planar assemblies, where the motions are one-step linear translations. Results suggest that the neural network can learn robustness to plan robust sequences an order of magnitude faster than simulation.**

I. Introduction

Given a set of parts and their relative positions in a product, the assembly sequencing problem is to find a sequence of collision-free motions that will merge the separated parts into the final assembly [1]. While the problem is PSPACE-hard in general [2], efficient algorithms have been developed for restricted types of motions, which guarantee to find a valid assembly sequence when one exists. However, even under such restrictions, there may be exponentially many valid sequences and so the computational hardness remains for the task of choosing an optimal sequence (based on a desired measure). We focus on such a task in this paper – finding a *robust* assembly sequence, which we define as one having a high probability to succeed under small deviations from the planned motions. Such deviations arise in real scenarios where noise is introduced due to uncertainties in control and sensing.

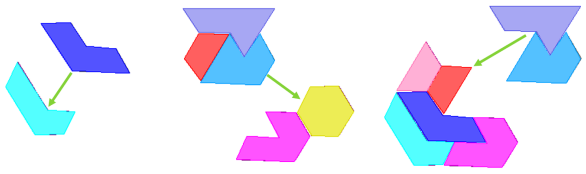

Fig. 1: Examples of 3 highly-robust operations. In each operation one subassembly moves as a rigid body in the direction shown while the other one is fixed.

*Work by D.H and T.G. has been supported in part by the Israel Science Foundation (grant no. 825/15), by the Blavatnik Computer Science Research Fund, and by grants from Yandex and from Facebook. A.T. was partly supported by Siemens. K.G. is supported in part by donations from Siemens, Google, Toyota Research Institute, Autodesk, ABB, Knapp, Honda, Intel, Hewlett-Packard.

[1]Blavatnik School of Computer Science, Tel-Aviv University, Israel

[2]Technion, Haifa, 3200003, Israel. Part of the work was done at University of California, Berkeley.

[3]Dept. of EECS, University of California, Berkeley

To provide some intuition about the problem and the effects of deviations, we present examples of assembly operations and their robustness levels, starting with highly robust ones in Figure 1. Note that the poly-lines along which the subassemblies in this figure are mated are characterized by wide "funnels", where mechanical compliance can guide subassemblies into alignment. Together with an appropriate direction for mating, these motions guide the moving subassemblies into the correct final position. We contrast these operations to ones that are not robust in the next figure:

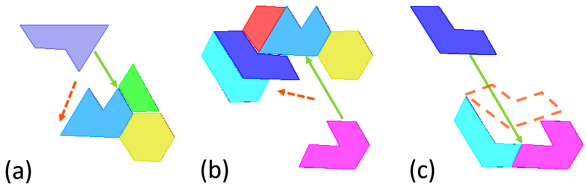

Fig. 2: Examples of operations that are not robust, in which noise along the planned path (shown by the green arrow) can result in undesirable motions (represented by red dashed arrows). We assign operations (a) and (b) a score of 0.5 (out of 1), and operation (c) a score of only 0.14 (see text for how it is obtained). We provide an explanation for the low scores: In operation (a) any deviation of the moving part to the left would result in it sliding away from the target along the part that is below it. The mating direction aggravates the situation (a better one would bring the moving part from the North-Northeast). In operation (b), only a single direction can be used and a similar scenario can occur if the moving part deviates left. In both (a) and (b), a deviation to the right should still allow successful completion. In contrast, operation (c) has a particularly low success rate since a deviation either way would likely result in failure: Missing right would cause the moving part to rotate and settle in a configuration (shown by the red dashed line) that can only be recovered from by fully backing up. A deviation to the left could end up the same way, since recovering from such a deviation would entail some momentum to the right.

These are just a few examples out of the exponentially many possible operations for the assembly in Figure 3. Our task is to find assembly sequences with robust operations (like those in Figure 1) by avoiding difficult configurations of parts already assembled (like those in Figure 2) and choosing appropriate mating directions.

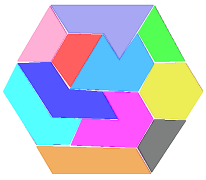

Fig. 3: Example of a final assembly.

In order to evaluate the robustness of assembly operations, we use a physics simulator, as we can exploit its immediate ability to capture the dynamics of the task, compared to analytic approaches (which would require handling potentially challenging intricacies, as we exemplify in Section VI-B.1). In particular, in this work we evaluate robustness by simulating

object mating using a proportional feedback controller [3] that moves an object along the assembly trajectory in the presence of actuation noise. In this case, robustness corresponds to the ability of the planned motion to tolerate noise.

Since the noise we add is random, multiple simulations are required to get the score, rendering it computationally burdensome. The effect on run time is compounded by the fact finding a good sequence involves scoring a myriad of candidate operations. Although planning robust assembly sequences can be performed offline for a given assembly, efficient algorithms can facilitate efficient design for assembly (DfA), where small changes in geometry can increase robustness and planning is executed inside an optimization cycle. We therefore introduce a convolutional neural network (CNN) that has potential to efficiently identify high scored assembly operations, given their visual representation, using binary classification.

We plan assembly sequences by obtaining a set of possible operations at each step using geometric planning and then greedily selecting the most robust one among them using the CNN. In case the CNN classifies all available operations as *not* highly robust, we fall back on the physics simulator to choose the best one (though this rarely occurred in our experiments).

While our proposed method handles planar assemblies, where the motions are one-step linear translations (like those in Figures 1 and 2), it is an initial step towards handling spatial assemblies. The planar case already has a high combinatorial complexity and raises interesting challenges, as we illustrate in Section V.

Contribution. We present the Robust Assembly Planning (RAP) algorithm, which takes a planar assembly and returns a robust two-handed assembly sequence using one-step translations (or determines that no feasible sequence exists under such motions). RAP also considers assembly operations that are not linear (i.e., ones which bring together more than one part into an existing subassembly).

II. Related Work

A. Assembly Planning and Optimization

Assembly planning is a well-studied problem in manufacturing and robotics. Some early planners employ a potentially exponential running-time generate-and-test approach, which enumerates all possible operations and tests their feasibility [4] while others pose questions to a human expert in order to establish precedence between operations [5]. A pioneering work by Wilson and Latombe [6], on which our geometric planner is based, introduces the *non-directional blocking graph*, which uses geometric reasoning and examines assembly operations in the space of the allowable assembly motions. This approach avoids the inherent combinatorial trap and leads to polynomial time algorithms for motions such as one-step translations and infinitesimal rigid motions [1].

Given the algorithmic success in finding feasible assembly sequences, a natural goal is to find ones that meet desired properties or optimality criteria. Goldwasser et al. [7], [8] show that optimizing assembly sequences under simple motions can be NP-hard even to approximate (we list some of their cost measures in Section VI-C). Assembly optimization is therefore typically solved with heuristic techniques. As the subject is vast, with many possible optimization criteria [9], we refer to the following surveys: [10], [11].

B. Robustness

A great deal of work revolves around planning and executing fine motions to handle challenging peg-in-hole style assembly tasks (e.g [12], [13]). We, on the other hand, seek to plan inherently robust sequences that minimize difficult tasks during assembly. We mention a few works that plan sequences using quality measures reminiscent of ours. Heger [14] focuses on the assembly environment by considering planar assemblies inside a constrained workspace, where robots would be operating. The work combines symbolic and motion planning to generate sequences and greedily optimizes them for two relevant measures: clearance along the path around the part being inserted and the reachability of its docking locations in each step. Moving the partial assembly between steps is allowed for improving these measures. Wan et al. [15] optimize sequences based on a score for the ease of 3D part insertion derived from contact normals (ignoring potential disturbances that can occur along the motion). Anders et al. [16] also learn from simulation to plan pushing motions for planar objects. They approach uncertainty with conformant planning, in which actions are sequenced to guarantee a successful configuration without intermediate sensing. They acquire a belief-state transition model for pushing squares into an arrangement by training a random-forest regressor.

C. Learning

The use of neural networks in assembly planning has been explored in [17], [18], [19]. In these works, supervised learning was used to learn a mapping from geometric features in an assembly to a complete assembly sequence, with the goal of replacing the search computation with a learned heuristic. In our work we use neural networks differently: we use them to replace physics simulation in predicting the robustness of an assembly operation, from a visual representation of the proposed assembly. We incorporate the predictions from this neural network into the assembly planning framework.

Neural networks have been used to learn controllers for assembly tasks, using inverse models [20], and reinforcement learning [21]. Neural networks have also been used for solving various planning problems in robotics, such as motion planning [22], [23], grasping [24], and pose estimation [25]. In this work, we apply deep CNNs to learning the robustness of an assembly operation, namely the motion that puts two subassemblies together to form one larger subassembly.

III. Preliminaries

A. Assembly Planning

An **assembly** is a collection of bodies (also called parts) in some given relative placements, such that no two bodies overlap. A **subassembly** is a subset of the bodies composing an assembly A in their relative placements in A. We say that

two subassemblies are **separated** if they are arbitrarily far apart from one another. Given an assembly A, an assembly operation is a motion that merges s ($s \geqslant 2$) pairwise separated subassemblies of A into a new subassembly of A. During this motion, each subassembly moves as a single rigid body and no overlapping between bodies is allowed. [1] In this paper the assembly operations are **two-handed**, i.e., every operation merges exactly two subassemblies. The reverse of an assembly operation is a **partitioning** operation. We denote both types of operation as a tuple (S_1, S_2, m), where $S_1, S_2 \subset A$ are the subassemblies merged/partitioned by the motion m.

An **assembly sequence** is a total ordering on assembly operations that merges the separated parts composing an assembly into this assembly. A common approach in assembly planning is **assembly-by-disassembly**, whereby a *dis*assembly sequence is obtained and then reversed. As the final assembly configuration is the most constrained, the approach reduces the search space and naturally offers valid operations. Disassembly consists of partitioning, first the input assembly A into subassemblies and then, recursively, the resulting subassemblies that are not individual parts. When only considering geometric feasibility, and assuming that the parts are rigid, the sequences are symmetric, though this is not true if we also care about robustness (e.g., the peg-in-hole scenario). Nevertheless, the approach is suitable for our setting, since we treat partitioning operations as the respective assembly operations when evaluating robustness. Due to this interchangeability of assembly and disassembly, we sometimes simply use the term **sequence**.

B. The Motion Space Approach

To complete the description of the disassembly process, we give an overview of the partitioning procedure using the so-called **motion space approach** [1]. The **motion space** is defined to be the space of parametric representations of all allowable motions for partitioning operations: every point in it uniquely defines a path of the subassemblies moved by an operation. A key concept in this approach is the **directional blocking graph** (DBG) [6]. Given a specific motion, a DBG is a directed graph that represents the blocking relation between parts: each node represents a part and an edge from part A to part B exists if applying the motion on part A results in a collision with part B. Given a DBG, it is sufficient to compute its strongly connected components in order to find a valid partition that uses this motion (or determine that no such partition exists, as is the case when the DBG is strongly connected). An important insight is that a single DBG can represent many motions, as the blocking relations are not necessarily affected by small changes to a given motion. This fact allows for decomposing the motion space into regions called **cells**, such that all motions in the same cell induce a single DBG. The decomposed motion space is called the **non-directional blocking graph** (NDBG) [6]. See Figure 4 for an illustration of DBGs for one-step translations.

[1] As is common in *assembly planning* we allow motion of parts in contact, such as one part sliding over the other, but forbid overlap of the interior of parts during the motion.

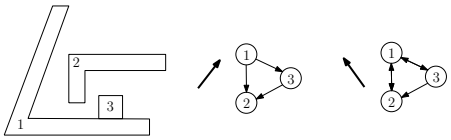

Fig. 4: Examples of DBGs for two directions used for one-step translation (adapted from [1]).

Once an NDBG is obtained for a given assembly, partitioning it involves finding a feasible partition for any of the DBGs (i.e., motions) it contains, meaning the NDBG captures all the geometric information required for partitioning. Its structure, particularly the number of cells in the decomposition, is therefore a critical run-time factor that depends on the type of allowed motions. For one-step translations (in both 2D and 3D) this number is polynomial in the number of parts in the assembly (and their complexity), which allows for finding an arbitrary assembly sequence in polynomial time (exact bounds for our setting appear in Section VI-C). For more details on NDBG construction and the application of the motion space approach for a few families of motions see [1].

IV. Problem Statement

Given a planar assembly with n parts, we would like to find the most robust two-handed sequence for it, where the motions are one-step translations, such as those in Figure 1. To evaluate a sequence for robustness we use the simulator to score the individual assembly operations in it, resulting in the scores $r_1, \ldots, r_{n-1} \in [0, 1]$ (note that a two handed sequence for n parts always has $n - 1$ steps). As we define the robustness of a single operation as its success probability under noise, the score we assign to the whole sequence is the product of these individual scores, $R := \prod r_i$, and our goal is to find a sequence that maximizes it.

As the search space involved here is potentially exponential, we restrict it to a polynomial one by greedily choosing assembly operations and in a few other ways, which will be described in the sequel.

V. The Hardness of Finding Optimal Sequences

In this section we illustrate the hardness of finding optimal sequences by considering a generic cost function, which takes an assembly sequence and returns a real value as its cost.

proposition. *Finding an optimal assembly sequence is NP-hard.*

Proof: We give a reduction from the PARTITION problem, which is: given n positive integers $a_1, \ldots, a_n$, decide whether they can be partitioned into two subsets whose sums are equal. Given such an instance I, we define an assembly A to be a row of n axis-aligned rectangles with the same height, where the width of rectangle i is a_i. We define a cost function that only takes into account the last operation, which merges two subassemblies, $S \subset A$ and $A \backslash S$, into the final assembly: The cost is the absolute difference between the sum of the widths of the rectangles in S and the corresponding sum for the rectangles in $A \setminus S$. Clearly,

under this cost function $I \in$ PARTITION if and only if the optimal cost of assembling A is 0.

A possible motivation for the cost function used is a desire to have the weights of the two subassemblies be close for balance reasons. Note that PARTITION was also used to show the NP-hardness of finding a feasible sequence for a planar rectilinear block puzzle, where each block is allowed to move multiple times using translations that are also rectilinear [26].

More NP-hard (dis)assembly optimization goals under one-step translations include minimizing the number of directions used to mate subassemblies and minimizing the number of parts that need to be removed from an assembly in order to remove a key part [8]. These hardness results motivate us to follow an approximate solution to robust assembly sequencing, which we describe next.

VI. The RAP Algorithm

We first give an overview and then explain how the planning, physics simulation, and CNN are used. Given a planar assembly A, we wish to efficiently find an assembly sequence close to optimal. We use a greedy approach, which selects the most robust assembly operation available at each step using the generic scoring procedure, **select_best_partition**. To this end, we first construct the NDBG for A and then apply Algorithm 1 on the complete assembly. The algorithm recurses on subassemblies introduced by partitions, following the assembly-by-disassembly approach.

Algorithm 1: A generic greedy approach for finding a robust assembly sequence.

Input : Subassembly $S \subseteq A$
Output : Assembly sequence for S
1 P $\leftarrow$ List of feasible partitions for S obtained from NDBG
2 **if** *no feasible partitions exist* **then**
3 | return failure
4 $(S_1, S_2, \overrightarrow{d}) \leftarrow$ **select_best_partition**(P)
5 Output $(S_1, S_2, \overrightarrow{d})$
6 Continue recursively on S_1 and S_2 (when more than a single part remains)

Section VI-A explains how the NDBG is used (line 1). As for the procedure **select_best_partition** (line 4), we have two implementations: an efficient CNN-based one, which we use for RAP, and baseline that only uses the physics simulator to score partitions (returning the one with highest score). We describe the former in Section VI-B, providing more details on the simulator in Section VII, and compare the two implementations with experiments in Section IX.

A. The Use of the NDBG

We first describe the structure of the NDBG in our setting and then explain how we use it to obtain partitions. Since the motions we allow are one-step translations, a single angular parameter representing the direction of the translation defines the motion. The motion space is therefore a unit circle $\mathcal{S}^1$ that is decomposed into an arrangement of vertices and arcs. Our NDBG implementation is an adaption of [27] to the planar case, for which the theory is given in [1].

To obtain a choice of possible partitions for a given subassembly $S \subseteq A$, we examine each cell in the NDBG and its associated DBG (restricted to S). As outlined in Section III-B, each DBG encodes all possible partitions for the directions of motion represented by the cell. In order to keep the overall running time polynomial, we restrict ourselves to outputting only a single partition per DBG (out of a potentially exponential number, e.g., when no part is blocked by another part in some direction). This entails choosing two subassemblies, $S_1, S_2 \subset S$, which is done arbitrarily among the valid options, and a direction, for which we have the freedom of choice only when the cell is an arc of $\mathcal{S}^1$. Such a cell represents a range of directions along which S_1 and S_2 can be mated and we select the middle of that range. This is a heuristic, which in many cases would result in a large clearance for the assembly operation, relative to other directions in the range. [2] While experiments mostly support this choice of direction, a more sophisticated selection can be considered in future work (e.g. by learning it as well). In Section VI-B.1 we further discuss clearance as a robustness measure.

We make two remarks on the restriction of one partition per DBG: First, our approach remains complete, as all directions are considered. Another fact that remains is that the space of feasible sequences is still potentially exponential. What reduces it further to a polynomial one is the greedy selection of the best partition.

B. Scoring and Selecting Partitions

As a preprocessing step, the CNN is trained on examples that are scored by the simulator and learns to classify operations as either *high*, i.e., having a score in the range [0.95, 1], or *not high*. Given a list of assembly operations as an argument to **select_best_partition** in the main algorithm, we first classify them using the CNN. If there are operations classified as *high*, we choose the one with the highest *confidence*, as measured by the softmax output of the network [28]. Otherwise, we resort to the simulator, scoring all partitions with it, and choose the one with the highest score. While a finer output from the CNN might be more desirable, we find that our use of binary classification is suitable, as the simulator does not need to be queried in almost all our experiments (see Section IX).

1) Clearance as an Attempted Analytic Robustness Measure: In this section we introduce clearance, an alternative *analytic* scoring method, and present its pitfalls. We define this score as the average minimum distance between the subassemblies during the mating motion. Intuitively, a higher clearance should minimize physical interaction between subassemblies that could hinder the success of the operation under noise. Indeed, we observe an overall positive correlation between this score and the simulator score, but we also find

[2] For this to hold, adjacent NDBG cells that admit the same subassemblies in their partitions need to be merged. Indeed this is what our algorithm does. We omit the details.

that it can mislead, as we show in Figure 5. While (d) has the highest clearance score among the four examples, it has the lowest simulator score. A deviation of the moving part to the right that brings the marked edges in (d) into contact, would result in an almost unrecoverable slide away from the target. This possible failure is not captured by clearance, as (c) has a slightly lower clearance even though its shape makes it robust to similar failures. Similarly, if we had to decide between (a) and (b) based on clearance, we would also end up forgoing the robust operation for one that is not. In this case, the part above the square in (a) reduces clearance, but allows the subassembly to slide to the correct place in case it misses to the left and touches the top rectangle. The above cases exemplify just a few intricate details that would have to be accounted for in an analytic solution, justifying the use of a simulator.

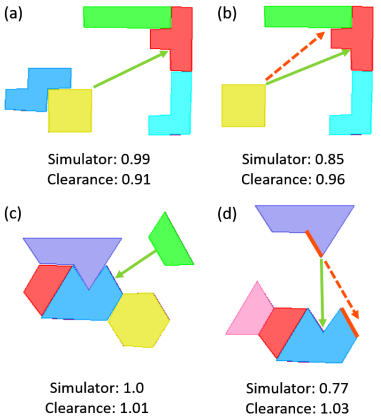

Fig. 5: Examples where the clearance does not correlate with robustness.

C. Time Complexity

We present the time bounds, for which more details are available in [1]. Let n be the number of parts in the input assembly A and q be the maximal number of edges in a single part. The NDBG has $O(n^2)$ cells and we compute it once in the beginning using $O(n^2(\log n + q^2))$ time.

In line 1 we obtain partitions by computing the strong components of each cell's DBG. This allows obtaining a partition using $O(n^2)$ time per cell, resulting in a total number of $O(n^2)$ partitions for the subassembly. In line 4 we score the partitions by feeding their images to the CNN and (if required) by simulating the assembly operations they represent. Let $t(n,q)$~~$t(A)$~~ be the time bound for scoring a single partition in this manner (a polynomial that mainly depends on the simulator's internals). We thus require $O(n^2(n^2 + t(n,q)))$ time for lines 1-4. These lines are repeated exactly $n-1$ times, as each time they run an assembly operation is chosen and that is the length of the sequence. The overall run time is therefore $O(n^2q^2 + n^3(n^2 + t(n,q)))$.

VII. Physics Simulation

To evaluate robustness, we use the Box2D physics engine [29] to simulate a controlled assembly motion of two objects with actuation noise. We start the simulation when the two objects are separated, and apply translational forces on one object to drive it to an assembled configuration, while the second object is held fixed. Rotational forces are also applied, though only for correcting the orientation of the moving object. We used a proportional feedback controller [3] to track a linear path connecting the object's initial position and the goal (see Figures 1 and 2).

We add actuation noise at each control loop iteration as follows: Let F be the correct force that the controller should currently apply. We add to F a random noise component drawn uniformly from the interval $[-\eta\|F\|, \eta\|F\|]$ on the axis perpendicular to F and output the sum as the noisy force, resulting in a stochastic system. Currently we set $\eta = 9$ as it gives a reasonable amount of noise when visualizing the operations while also resulting in a sufficiently varied distribution of robustness scores.

We measure the robustness of the operation as the success rate of the controller in driving the object sufficiently close to the goal within a fixed time, after performing 100 trials. At the beginning of each trial, we place the moving object such it is completely outside of the bounding box of the static object (aligned with the direction of motion) and is also located at least a fixed distance away from it.

VIII. Neural Network

Using a physics simulator to predict the success probability of an assembly operation, as outlined in the previous section, adds significant computational overhead to the planning algorithm. We hypothesize, however, that in a practical setting, it is possible to exploit similarities between different assemblies to reduce the computational burden. For example, it may be that peg-in-hole type assemblies are particularly difficult for our robot, and we can identify such structures and avoid them without requiring extensive simulations. Here, we propose a general approach for identifying such structural properties by using supervised learning [30].

In our approach, we generate N random assembly operation instances, $x_1, \ldots, x_N$. A single assembly operation instance x_i includes the initial, unassembled, position of two subassemblies, and their assembled position. For each instance x_i, we use the physics simulator to obtain a robustness score y_i. We propose to use supervised learning to learn a mapping f from some features of the assembly instance $\phi(x)$ to their score y, such that during planning, we could use f instead of the physics simulator to obtain robustness scores.

Selecting features that are relevant to the robustness score is not trivial. Here, we build on the recent success of deep convolutional neural networks (CNNs;[31]) in automatically learning features from image input. We observe that in a 2D setting, an image of the subassemblies contains all the geometric information that is input to the physics simulator, and is therefore sufficient, in principle, for a CNN to decode features relevant to predict robustness. For each instance x_i, we therefore generate two images (see Figure 6): **start image**, where the subassemblies are in their initial position, and **goal image**, where the subassemblies are assembled. These images

are input to a CNN, which predicts whether the operation has high robustness.

A. CNN Architecture

CNNs have been previously used for various geometric tasks such as shape recognition [32], pose estimation [25], and evaluating robotic grasp success [24]. Here, we follow a similar approach, and design a CNN for evaluating assembly robustness, as described in Figure 6. Our input is represented as a 2-channel 64×64 image, where each channel corresponds to a grayscale rendering of the start and goal images. In principle, this image contains all the information about the planned trajectory and the geometry of the parts involved, and we expect that a CNN based architecture would be able to extract this information to predict a robustness score.

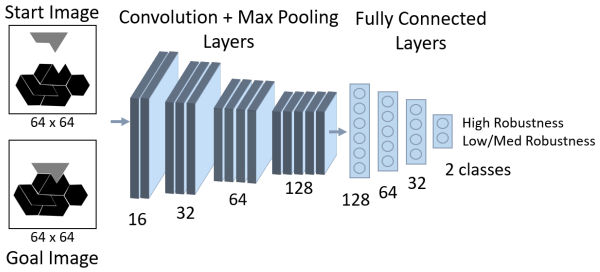

Fig. 6: CNN architecture for predicting robustness. Given 64×64 grayscale images of the starting position and goal position of two subassemblies, the network predicts whether the assembly operation is highly robust or not (see text for details). The CNN is composed of four convolution layers with 4×4 kernels, ReLU activations and max-pooling, followed by three fully connected layers with ReLU activations, and a final linear connection to the output layer.

We use the popular CNN architecture of several convolutions and pooling layers, followed by fully connected layers with dropout [31], [33], [34], as depicted in Figure 6. For the output, we found that the numerical value of the robustness score was hard to predict accurately, and instead we quantize the score into 2 classes: *high*, corresponding to the score range [0.95, 1], and *not high*. We trained our networks in PyTorch [35], using Adam [36] with default parameters to optimize the cross-entropy loss [31].

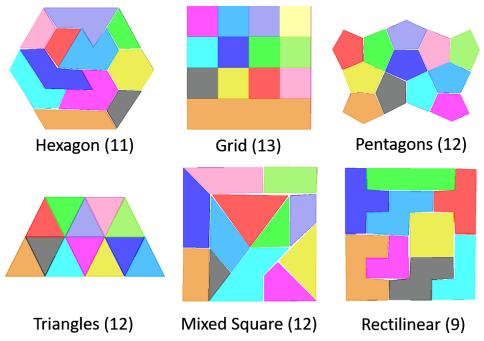

Fig. 7: The assembly tasks in our experiments.

B. Dataset Normalization

In our experimental setting, gravity is not considered and therefore the robustness score is invariant to the orientation of the operation as a whole. We therefore preprocessed the input images by rotating them such that moving subassembly translates downward, which resulted in a small performance gain. In the absence of such symmetry, similar gains can be obtained by increasing the data size.

IX. Experiments

We now evaluate the prediction accuracy of the CNN and RAP's run time and sequence robustness. Figure 7 shows the assembly tasks in our experiments and the number of parts in each of them. We chose to analyze these puzzle-like assemblies due to their highly constrained nature, which increases the interaction between parts and requires careful planning.

A. Performance of the CNN

One question that arises when using a machine-learning approach to predicting robustness is how well the trained model generalizes to assemblies not in the training data. We investigate this question by performing 6-fold cross-validation on the data obtained from the assemblies in Figure 7, consisting of 40,000 examples for each of them. [3] We test the CNN model on data for each one of the assemblies in turn, while using the rest for training. In Table I we report the accuracy in predicting robustness. These results suggest that the CNN can indeed generalize and predict robustness for assemblies it had not seen during training. Thus, with sufficient training data obtained off-line, we can learn an effective model for predicting robustness during online planning, thereby reducing planning time.

Test Assembly	Accuracy	Low Robustness	High Robustness
Hexagon	84.79	93.17	69.87
Mixed Square	84.73	77.77	89.48
Pentagons	74.93	94.17	72.09
Rectilinear	82.64	80.70	90.99
Grid	81.70	70.98	90.35
Triangles	93.13	88.77	96.41

TABLE I: Overall accuracy of our CNN model in predicting robustness alongside individual class accuracy. Each row shows results for a CNN trained on examples from all assemblies other than the one it was evaluated on.

B. Performance of RAP

1) RAP vs Simulator-Only Baseline: For each assembly task we compare the time taken to find a sequence and its robustness score using RAP and the baseline. Regardless of the approach used, sequences are evaluated using the simulator (as it is our "ground truth"). Also note that when applying RAP on an assembly, we use the CNN that has not seen the data for that assembly in training.

The average run time on all examples for the simulator-only baseline is 21.58 seconds (with low variance) while for RAP it is 2.38 seconds. In all but the Rectilinear assembly task RAP does not query the simulator, resulting in an average run time

[3] We obtain many examples for each assembly using its NDBG by exhaustively considering all possible partitions for each subassembly, as opposed to choosing just one like in RAP's online phase (i.e. we generate all possible sequences).

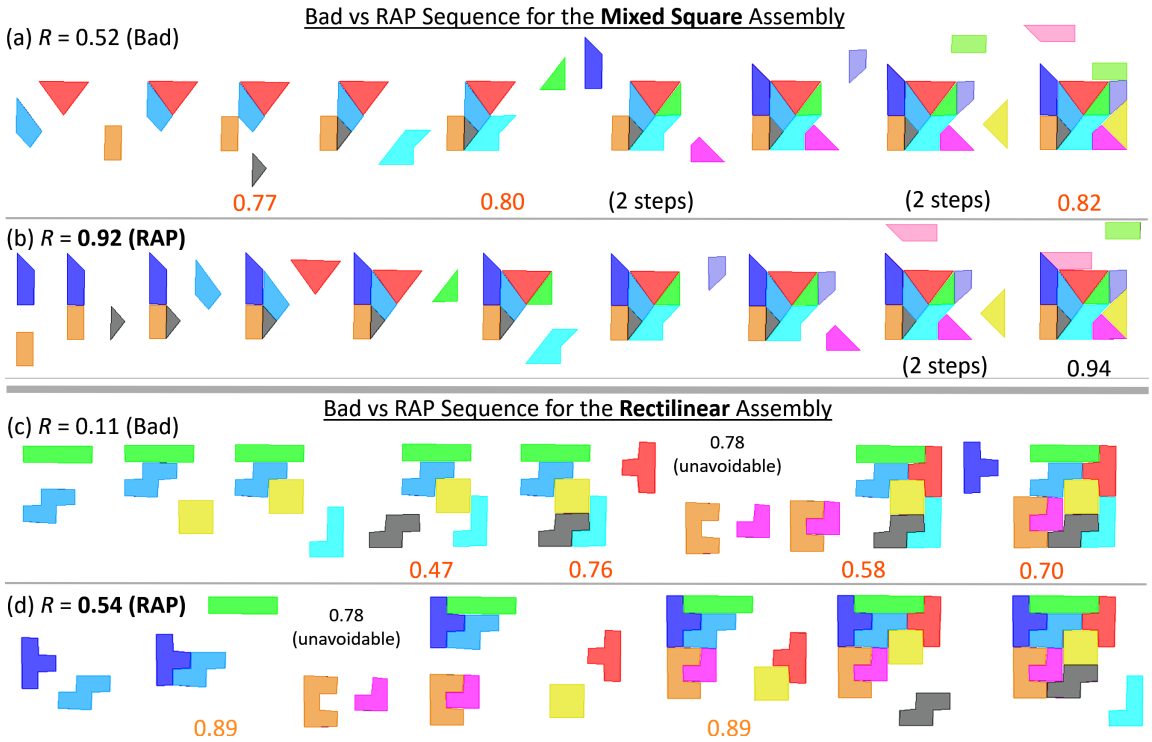

Fig. 8: Two sequences each for the Mixed Square assembly, (a) and (b), and the Rectilinear assembly, (c) and (d). For each of them the bottom sequence is returned by RAP, while the top one is an arbitrary one with low robustness. Individual motion scores are shown for non-robust steps (i.e., with a score less than 0.95). Total robustness score R is the product of individual scores. Sequence (d) shows how crucial it is to consider non-linear sequences, as the operation that results in the second to last subassembly (before the S-shaped part is inserted) would be a tight insertion if we only allowed a single part to be inserted.

of 1.4 seconds for these tasks. In the Rectilinear example, the search leads to a subassembly for which the CNN classifies all possible operations as not robust (even though one is), leading to a simulator query for them, which results in a run time of 7.0 seconds. Nevertheless, even in this case RAP improves run time by more than 3-fold (compared to 10-fold overall).

As for robustness, the scores end up very close: the average difference between the baseline and RAP is only -0.005 (scores for RAP are shown in Figure 9). These results indicate that the time improvements resulting from using a CNN within RAP do not come at the expense of robustness.

2) Evaluating Robustness: For this evaluation we must first note that absolute robustness scores do not necessarily indicate the quality of the optimization, since an assemblies' sequences can naturally tend towards a certain level of robustness. For example, there might be unavoidable steps that are not robust, such as the C-shaped part in the Rectilinear assembly, which induces a peg-in-hole scenario (see "unavoidable" examples in (c) and (d) in Figure 8). To take such inherent difficulties into account, we compare the sequences obtained by RAP to both optimal and many random sequences. For the purpose of the comparison we fix the possible operations available for each subassembly, i.e. we always use the NDBG as we describe in Section VI-A, which maintains consistency in the returned partitions. We find optimal sequences using an exhaustive search that considers all possible partitions for each subassembly and scores all of them using the simulator (a process which is more than 100 times slower than RAP). As for the random sequences, in which each next step is selected uniformly at random, we find the average score of 50 such sequences. We present the comparison in Figure 9.

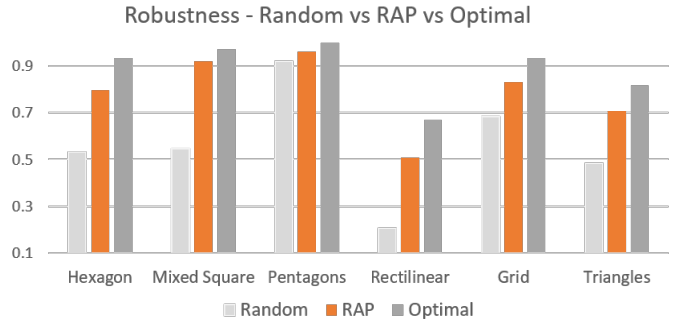

Fig. 9: Comparison of average scores of 50 random sequences, scores of sequences obtained by RAP, and the optimal ones for the different assembly tasks. Indeed our learning-based approach offers significant robustness gains, which are close to optimal, using a fraction of the running time required by the optimal procedure.

In Figure 8 we present sequences for two assemblies, comparing two obtained by RAP alongside two with a low robustness score. For both assemblies, the sequence with the lower score introduces configurations that hinder the successful insertion of parts, with a clear negative impact on robustness. The impact in the Rectilinear example is especially severe, since four more peg-in-hole operations are performed compared to the only (unavoidable) one in the sequence returned by RAP. We observe similar benefits of RAP in the other instances as well.

X. Conclusions and Future Work

In this work, we learned a robustness metric by using a physics simulator to provide the ground truth labels. A recent study noted the difficulty of such CNNs in capturing geometric properties such as coordinate transforms [37]. We conjecture that a special purpose architecture would further improve our results.

Even after a sequence is fixed, individual motions can be optimized. As we pointed out, within one NDBG cell, one can typically choose from an infinitude of directions. In the current work we choose the middle direction, which is well defined in the one-dimensional motion space. Selecting a good direction in general is in itself an interesting problem, requiring first to define what makes a direction better than others. More generally, one could replace a one-step translation with an arbitrary path, e.g. to rely on compliant motion to further improve robustness.

References

[1] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3-4, pp. 577–601, 2000.

[2] B. K. Natarajan, "On planning assemblies," in *Proceedings of the fourth annual symposium on Computational geometry*. ACM, 1988, pp. 299–308.

[3] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and J. D. Powell, *Feedback control of dynamic systems*. Addison-Wesley Reading, MA, 1994, vol. 3.

[4] L. H. De Mello and A. C. Sanderson, "A correct and complete algorithm for the generation of mechanical assembly sequences," *IEEE transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240, 1991.

[5] A. Bourjault, "Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires," *These D'etat, Universite de Franche-Comte*, 1984.

[6] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[7] M. Goldwasser, J.-C. Latombe, and R. Motwani, "Complexity measures for assembly sequences," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 2. IEEE, 1996, pp. 1851–1857.

[8] M. Goldwasser, "Complexity measures for assembly sequences," Ph.D. dissertation, stanford university, 1997.

[9] R. E. Jones and R. H. Wilson, "A survey of constraints in automated assembly planning," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 2. IEEE, 1996, pp. 1525–1532.

[10] P. Jiménez, "Survey on assembly sequencing: a combinatorial and geometrical perspective," *Journal of Intelligent Manufacturing*, vol. 24, no. 2, pp. 235–250, 2013.

[11] S. Ghandi and E. Masehian, "Review and taxonomies of assembly and disassembly path planning problems and approaches," *Computer-Aided Design*, vol. 67, pp. 58–86, 2015.

[12] Y.-L. Kim, B.-S. Kim, and J.-B. Song, "Hole detection algorithm for square peg-in-hole using force-based shape recognition," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2012, pp. 1074–1079.

[13] T. Tang, H.-C. Lin, Y. Zhao, W. Chen, and M. Tomizuka, "Autonomous alignment of peg and hole by force/torque measurement for robotic assembly," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2016, pp. 162–167.

[14] F. W. Heger, "Generating robust assembly plans in constrained environments," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 4068–4073.

[15] W. Wan, K. Harada, and K. Nagata, "Assembly sequence planning for motion planning," *Assembly Automation*, vol. 38, no. 2, pp. 195–206, 2018.

[16] A. S. Anders, L. P. Kaelbling, and T. Lozano-Perez, "Reliably arranging objects in uncertain domains," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1603–1610.

[17] D. Hong and H. Cho, "A neural-network-based computational scheme for generating optimized robotic assembly sequences," *Engineering applications of artificial intelligence*, vol. 8, no. 2, pp. 129–145, 1995.

[18] C. Sinanoğlu and H. Rıza Börklü, "An assembly sequence-planning system for mechanical parts using neural network," *Assembly Automation*, vol. 25, no. 1, pp. 38–52, 2005.

[19] W.-C. Chen, P.-H. Tai, W.-J. Deng, and L.-F. Hsieh, "A three-stage integrated approach for assembly sequence planning using neural networks," *Expert Systems with Applications*, vol. 34, no. 3, pp. 1777–1786, 2008.

[20] J. Kim and H. S. Cho, "A neural net-based assembly algorithm for flexible parts assembly," *Journal of Intelligent and Robotic Systems*, vol. 29, no. 2, pp. 133–160, 2000.

[21] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," *arXiv preprint arXiv:1803.07635*, 2018.

[22] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," *arXiv preprint arXiv:1609.07910*, 2016.

[23] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *arXiv preprint arXiv:1709.05448*, 2017.

[24] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, p. eaau4984, 2019.

[25] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017.

[26] B. Chazelle, T. Ottmann, E. Soisalon-Soininen, and D. Wood, "The complexity and decidability of separation," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1984, pp. 119–127.

[27] E. Fogel and D. Halperin, "Polyhedral assembly partitioning with infinite translations or the importance of being exact," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 2, pp. 227–241, 2013.

[28] Y. Geifman and R. El-Yaniv, "Selective classification for deep neural networks," in *Advances in neural information processing systems*, 2017, pp. 4878–4887.

[29] E. Catto, "Box2d: A 2d physics engine for games," 2011.

[30] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[32] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," in *Advances in Neural Information Processing Systems*, 2018, pp. 9628–9639.