# PEEL: Parallel Extraction for Long-Horizon Disassembly Planning via Scale-Invariant Sampling

Servet B. Bayraktar and Andreas Orthey and Zachary Kingston and Marc Toussaint

***Abstract*—Long-horizon multi-part object disassembly requires robots to compute feasible sequences of collision-free removal motions, even in the presence of tight, narrow escape corridors. To efficiently solve such disassembly problems, we propose Parallel Extraction for Long-Horizon Disassembly (PEEL), an algorithm which efficiently computes disassembly motions for object assemblies and feeds them to a robot manipulator for execution. PEEL uses sampling-based motion planning to compute single-object motions through the use of a scale-invariant sampling scheme, where the object scale is estimated in a burn-in phase and a subsequent directional sampler exploits the scale. This sampling scheme is integrated into a multi-arm bandit rapidly-exploring random tree (MAB-RRT) planner, which switches between different samplers depending on the reward signal received. Using MAB-RRT, the PEEL algorithm runs a batch of planners in parallel to obtain an ordered graph specifying the sequence in which object parts have to be removed. We show that MAB-RRT can efficiently solve single-part disassemblies with 100 percent success rate on 76 assemblies, and that it is robust to its parameters. By integrating MAB-RRT into PEEL, we solve four long-horizon disassembly problems using the Fetch manipulator robot involving 10 to 17 individual object parts. Further animations, code, and videos can be found at https://peel-disassembly.surge.sh/.**

## I. Introduction

Multi-part object disassembly is a fundamental robot skill with applications in recycling, remanufacturing, repair, material recovery, and building deconstruction. Disassembling arbitrary objects requires detecting and identifying parts, removing them without damaging the assembly, and computing action sequences that systematically dismantle an object. A core challenge hereby is disassembly planning, where we need to determine the order in which parts should be removed while ensuring geometric feasibility.

Prior approaches to disassembly planning have primarily relied on symbolic or graph-based search over part precedence relations [18]. While effective for small assemblies, these methods scale poorly as exhaustive graph search grows exponentially with part count. Physics-based planners [47] capture contacts and forces in detail but are computationally expensive for large assemblies. Geometric planners abstract away physical forces and consider only collision-free motion. While this abstraction may yield solutions requiring refinement for physical execution, geometric planners produce feasible candidates quickly and provide a foundation for subsequent physics-aware reasoning [47].

In this work, we propose an efficient geometric motion planning method that analyzes a given assembly, computes a collision-free disassembly sequence, and generates motion plans which can be executed on a robot platform.

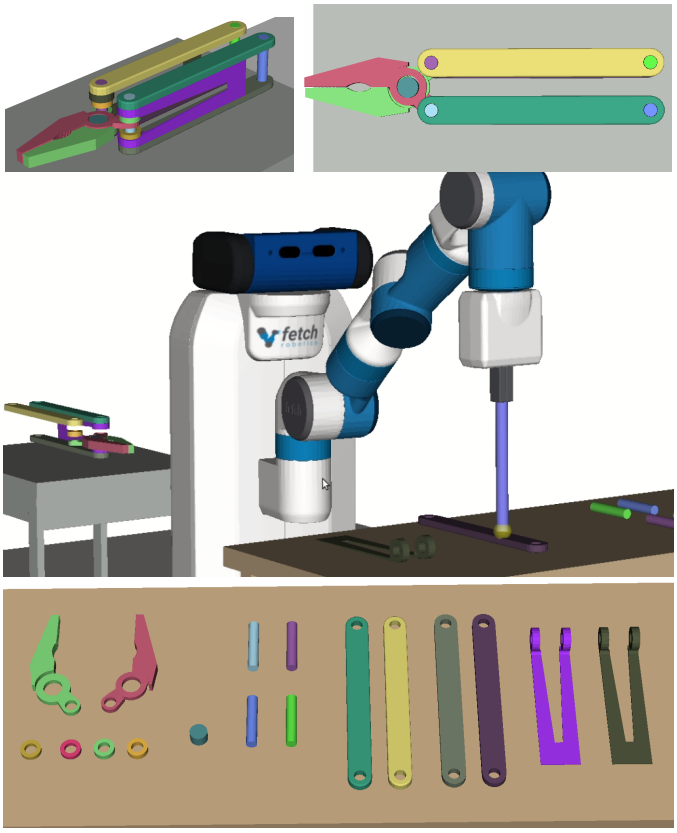



**Figure 1:** A Fetch mobile manipulator executing a disassembly sequence on a pair of pliers. Top: A pair of pliers in its assembled state viewed from its side (left) and from the top (right). Middle: the robot places a disassembled part at its goal location on a table. Bottom: The disassembled pieces are placed side-by-side on the table.

The core difficulty in geometric disassembly differs fundamentally from typical motion planning. Rather than exploring a broad configuration space, the planner must identify narrow escape corridors to move each part from its constrained initial state into free space. For example, to disassemble a pair of pliers (Fig. 1), bolts have to be removed from a thin corridor within an otherwise blocked configuration space. Uniform sampling rarely discovers such corridors [39], and naive directional sampling [13] can become inefficient without proper initialization.

Beyond extracting a single part, multi-part disassembly requires deciding the order in which parts are removed. The number of reachable subassembly states grows exponentially with the part count, so explicitly searching over part precedence relations [18] quickly becomes intractable. We observe, however, that whether a part is currently removable is decided

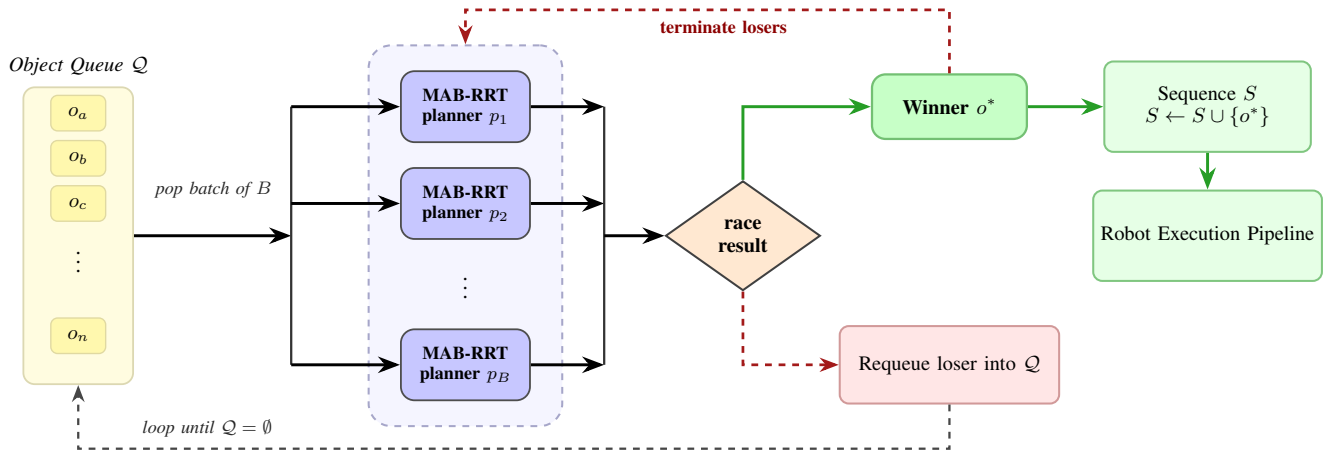



**Figure 2:** Parallel batch protocol (Algorithm 1). At each round, up to $B$ objects are popped from the shuffled queue $\mathcal{Q}$ and assigned to concurrent MAB-RRT planner processes that share the current planning budget $T$. The race resolves into one of two cases. *(Success)* The first planner to finish yields the winner $o^*$, which is committed to the solution sequence $S$; all remaining batch processes (the losers) are terminated, their objects are returned to $\mathcal{Q}$, and $T$ is reset to $T_0$ because the geometry has changed. *(Timeout)* If no planner finishes before $T$ elapses, all processes are terminated and the full batch is requeued. Once every unsolved object has been attempted at the current budget, that is, once one full sweep has failed, $T$ is doubled. The procedure repeats until $\mathcal{Q}$ is empty. The output is a sequence graph $S$, which determines the order of objects and object paths, which can then be used in robot execution.

by the very same query a motion planner already answers: does a collision-free extraction motion exist? This lets us sidestep the symbolic ordering search and allows geometric feasibility itself to reveal the removal order.

Building on this observation, we introduce the **P**arall**E**l **E**xtraction for **L**ong-Horizon Disassembly (PEEL). Rather than committing to a fixed order, PEEL plans the removal of all currently blocked candidate parts concurrently in a batch, using the multi-arm bandit RRT (MAB-RRT) [6] planner as the underlying single-object planner, which uses scale-invariant sampling to quickly extract a part from its surrounding assembly. The first planner to find a valid escape motion wins: its part is removed, all competing planners are terminated, and any unsolved parts are requeued for the next batch on the reduced assembly. This winner-based race avoids combinatorial subassembly enumeration, exploits parallel compute, and adapts naturally as the assembly is dismantled.

To ensure that MAB-RRT works reliably as a single-object planner in PEEL, we benchmark MAB-RRT on 76 assemblies from the Automate dataset [45]. On this benchmark, MAB-RRT achieves a 100% success rate under strict collision checking, outperforming baseline geometric planners [47]. Additionally, we evaluate its hyperparameters and demonstrate robustness across parameter settings. By using MAB-RRT inside PEEL, we validate PEEL on four multi-part assemblies containing 10–17 parts and demonstrate execution on a robot manipulator for all four assemblies.

Our contributions are, therefore, as follows:

- Introduction of the Parallel Extraction for Long-Horizon Disassembly (PEEL), a greedy framework for multi-part disassembly that replaces the search over part precedence relations with concurrent planning races: candidate parts are planned concurrently in batches with MAB-RRT [6].
- Extensive benchmarking of MAB-RRT on 76 benchmark assemblies shows robustness to hyperparameters and improved runtime performance over baseline methods.
- Integration of the results of PEEL into a robot execution pipeline that computes collision-free manipulation actions for a robot manipulator, including possible regrasping actions in case of failure.
- Demonstration of robot execution in a physical simulation on four multi-part assemblies by using the computed precedence graphs.

## II. Related Work

This work is closely related to disassembly sequence planning, object extraction, and parallel motion planning. We review those topics below and discuss their relation to our work.

### A. Disassembly Sequence Planning

Disassembly planning [3] has historically been treated as a combinatorial sequencing problem. The standard representation is the AND/OR graph [18], [19], whose nodes are subassemblies and whose hyperarcs are binary partitions that can be separated. Such graphs represent the assembly space completely, but the number of candidate partitions grows exponentially with the part count [49], and building one from CAD data exhibits $n!$-proportionate computational behavior [32]. These costs are accumulated before any sequence can be selected, which makes the approach impractical beyond roughly 20 parts [32].

Sequencing methods differ in the type of feasibility they establish. Symbolic feasibility encodes part relationships in precedence graphs [33], [48] or interference matrices and treats each removal as atomic. Wilson and Latombe [49] showed that deciding whether a subassembly can separate requires geometric reasoning about blocking relationships in every direction of motion. Their non-directional blocking graph captures these constraints in polynomial time, but tests only infinitesimal motions and therefore does not certify global removability. Geometric feasibility instead integrates path planning into

sequencing to verify that a removal is realizable [44], though such methods still rely on a precomputed precedence structure to generate the candidates [27].

Sequencing methods also differ in when they declare a part separated. The strongest notion requires the part to reach the unbounded free-space component, that is, to be removable to infinity [49]. Most systems instead check a cheaper surrogate, such as convex-hull separation [47], exit from a bounding box [1], or a displacement threshold [46], none of which certifies global separability once rotation is allowed.

The most directly comparable multi-part system is AssembleThemAll (ATA) [47], which casts assembly as disassembly inside a physics simulator: the continuous six-dimensional motion search is replaced by a breadth-first search over a discrete action space of forces applied for fixed time steps, and contacts are resolved by *signed distance field* (SDF) collision detection under a non-zero penetration threshold, since convex-hull decomposition suits neither complex concave geometry nor the small inherent overlaps common in CAD assemblies. Its sequencing layer is a progressive breadth-first search over parts: each iteration attempts every remaining part at a bounded search depth, commits the first that succeeds, and increments the bound until all parts are removed. ASAP [46] extends ATA with gravitational stability, robot integration, and a learned part-selection policy, requiring each candidate subassembly state to be both reachable by a path planner and statically stable under gravity.

**Learning-based ordering:** A complementary line of work attacks the combinatorial cost of sequencing by learning which parts are likely removable, ordering the removal-test queue by a learned prior instead of searching it exhaustively. Cebulla et al. [10] predict per-part removability with a graph neural network over contact-graph representations of CAD assemblies, reducing the number of removal tests by 64% to 90% on five real-world assemblies of bolted aluminium framing, of 29 to 68 parts and so larger than the assemblies we evaluate. The learned policy of ASAP [46] serves the same purpose.

**Parallel disassembly work:** Parallelism has been applied to disassembly before, though not to sequence discovery. Aguinaga et al. [1] parallelize RRT-based path planning for selective disassembly, while Ren et al. [38] and Zhang et al. [55] parallelize execution, respectively scheduling concurrent manipulators under precedence and collision constraints and identifying groups of parts removable simultaneously. Dedicated part-removal query planners [53] and broader metaheuristic surveys [17] complete the picture.

Across these approaches a common pattern emerges: sequencing and the geometric feasibility check are treated as separate stages. A precedence structure is computed first, whether by graph search, interference matrices, learned priors, or progressive search over discrete actions, and motion planning, if performed at all, only validates the candidates that layer proposes. PEEL removes the separation, since each round of its parallel race is a set of per-part motion planning queries whose winner defines the next removal, so no precedence structure is ever constructed. We also check collisions with bounding volume hierarchies at zero penetration tolerance rather than SDF detection under a non-zero threshold [47], which restricts the admissible input, as we filter assemblies for strict separation, but removes any reliance on a penetration tolerance.

### B. Object Extraction

Object extraction, the separation of one object from its surroundings, is the operation on which any disassembly sequence rests. Typically the object is a bolt, screw, nut, or gear that has to be moved through an elongated narrow passage [47]. Special cases admit shortcuts: screw removal reduces largely to choosing the correct tool [54], [29], cluttered environments offer wider passages where generic sampling-based methods suffice [5], and objects piled on each other require methods that keep the pile from collapsing [36], [31]. We target the general case, where narrow passages occur for almost every part.

Two frameworks have proven particularly effective. The first is physics-based simulation [47], [57], which applies random forces to the object; although most point in the wrong direction, their projection onto the correct escape direction still advances it [47]. Behavioral Kinodynamic RRT (BK-RRT) [57] treats such forces as random controls, as kinodynamic systems are moved, while ATA's disassembly breadth-first search [47] explores reachable states with a similarity check that avoids duplicate directions. However, such simulation is costly [47].

The second framework is biased sampling in sampling-based motion planning [34], which avoids simulating dynamics. Samplers bias toward object boundaries [7], [2] or directly toward narrow passages [20]; utility-based sampling scores samples by their expected contribution to planning progress [9]; and the dynamic-domain RRT [51] adapts a per-node radius to each sample's success in extending the tree, which helps overcome narrow passages and even explore zero-measure manifolds for contact planning [52].

Biased sampling can also be specialized to extraction. Manhattan-like RRT [11] plans for the object under consideration and moves the remaining degrees of freedom only when they block the removal. The same decomposition appears in factored state spaces [5], where interpolating one factor at a time within a single tree advances one object while the others stay passive. Targetless-RRT [1] replaces an explicit target with an implicit goal region, such as a separation threshold between object and environment. Mating Vector RRT (MateVec-TRRT) [14] computes mating vectors along which separation from the environment is likely to increase. Most recently, the classical multi-arm bandit (MAB) [4], [8], [41] has been combined with RRT into MAB-RRT [15], [16], [6], letting a bandit arbitrate online between samplers and so admitting dedicated disassembly samplers [6]: scale-invariant sampling first locates the scale at which the free space around the object is best resolved, and a principal component analysis (PCA) [12], [13] of the samples taken at that scale then gives the direction along which to advance locally.

We use MAB-RRT [6] directly as the single-object planner inside PEEL (Sec. IV-B); our contribution is the protocol built around it, which targets full multi-part disassembly rather than

single-object extraction. To establish it as a reliable component in this setting, we benchmark it under strict collision checking and study its parameter sensitivity (Secs. VI-A and VI-C).

### C. Parallel and Distributed Motion Planning

Parallel computation has been applied to motion planning at two levels. Within a single query, planning primitives such as collision checking and nearest-neighbor search are parallelized: PRRT and PRRT* [22] build one shared RRT on multicore CPUs using lock-free concurrency and partition-based sampling, pRRTC [21] runs hundreds of concurrent RRT-Connect iterations on a GPU over shared start and goal trees, and cuRobo [43] batches trajectory optimization across seeds and keeps the best. Across queries, independent planners cooperate: C-FOREST [35] shares improving solutions between trees to enable pruning and tighter sampling bounds, an advantage that diminishes for feasibility problems where the first solution suffices; distributed RRTs partition the configuration space across processors and merge the resulting subtrees [23]; SRT grows independent local trees in parallel and connects them into a roadmap [37]; and dRRT searches the implicit tensor product of independently built per-robot roadmaps [40].

All of these parallelize a single planning query. PEEL instead parallelizes across *candidate parts*, so the race between concurrent planners is what determines the removal order. This winner-takes-all scheme is a form of OR-parallelism [22], [35], applied to discover disassembly sequences rather than to accelerate a single plan. The decomposition also keeps every query low-dimensional: each planner searches $\mathrm{SE}(3)$ for one object on a single core, rather than the composite space of all objects jointly, where sampling-based planners become intractable and partitioning into independent subproblems would otherwise require structured decompositions such as factored state spaces [5]. PEEL needs neither.

## III. Multi-Part Disassembly Problem

We formulate multi-part disassembly as a sequential geometric motion planning problem. Let an assembly be a set of $N$ rigid objects $\{o_1, o_2, \ldots, o_N\}$, each occupying the six-degree-of-freedom configuration space $\mathrm{SE}(3)$. Each object $o_i$ begins at an initial configuration $\mathbf{q}_i^{\text{init}}$ and must reach a goal region $G_i$ while avoiding collisions with all other objects and static obstacles. Objects can only be moved by a robot $R$, with configuration space $C$, that has an end-effector to establish a rigid connection between robot and object which is modeled using a welding joint.

Our formulation is purely geometric: objects are treated as rigid bodies, and feasibility is determined solely by collision constraints. Physical effects such as friction, gravity, and contact forces are not modeled. This abstraction isolates the geometric structure of disassembly planning and focuses on the geometric feasibility problem.

The sequential nature of disassembly induces strong dependencies between planning problems. Removing one object reduces the collision constraints on the remainder, often enabling motions that were previously blocked. The problem is therefore to identify a sequence of collision-free motions that incrementally dismantles the assembly and executes the sequence using robot $R$.

Let us define this more formally. Let $X \subseteq \{o_1, \ldots, o_N\}$ denote all objects that have not yet been removed. An object $o_i \in X$ is *removable in* $X$ if a collision-free path exists for $o_i$ from $\mathbf{q}_i^{\text{init}}$ to $G_i$ treating $X \setminus \{o_i\}$ as static obstacles, and $X$ is *sequentially decomposable* if its objects admit an ordering $o_{\pi(1)}, \ldots, o_{\pi(m)}$ in which every $o_{\pi(k)}$ is removable in $\{o_{\pi(k)}, \ldots, o_{\pi(m)}\}$. We model a removed object as leaving the scene rather than being displaced within it, which is consistent with placing every $G_i$ away from the assembly.

This formulation leaves open how the goal region $G_i$ is defined, that is, when a part counts as *disassembled*. As noted in Sec. II-A, the principled criterion, removal to infinity, is expensive to evaluate. We therefore use a computable surrogate: $G_i$ is a region of free space placed away from the assembly, and a part is removed once its position enters $G_i$. Alternatively, an explicit goal configuration can be used. At execution time we detect a local *free state* with a mobility-rank test (Sec. V).

## IV. Parallel Extraction for Long-Horizon Disassembly

To solve the disassembly problem, we propose the Parallel Extraction for Long-Horizon Disassembly (PEEL) algorithm. PEEL uses, at its core, a single-object planner that employs scale-invariant sampling to separate two parts to extract a single part from the surrounding assembly. Given such a planner, we formulate sequence discovery as a series of *planning races*: at each round, several single-object planners execute concurrently, and the first to succeed determines the winner, after which the round ends. This race-based strategy incrementally identifies removable objects without building an explicit search over all orderings. Fig. 2 illustrates the protocol.

### A. High Level Algorithm

The actual implementation of PEEL is shown as pseudocode in Alg. 1. The algorithm starts in Line 1 by initializing the queue $\mathcal{Q}$ with a random shuffle of all objects; line 2 initializes the solution sequence $S$ to empty. Lines 3 and 4 initialize the planning budget $T$ to $T_0$ and the set $\mathcal{F}$ of objects already attempted at that budget to empty. The main loop (line 5) repeats until $\mathcal{Q}$ is empty. At the start of each iteration, line 6 pops up to $B$ objects from the front of $\mathcal{Q}$ to form the current batch. Lines 8–11 launch one independent planner process per batch object: each planner treats its assigned object as movable and all remaining objects (line 9) as fixed obstacles. Line 13 blocks until either one process succeeds or the budget $T$ elapses.

If a winner is found (lines 14–18), its object $o^*$ is committed to $S$ (line 16), all other processes are terminated (line 17), and their objects are requeued (line 18). This winner-takes-all strategy halts planners working on objects that may become easier to remove once the winner is extracted. Because the geometry has changed, the budget is reset to $T_0$ and $\mathcal{F}$ is cleared (line 19): earlier failures were recorded against an assembly that no longer exists.

**Algorithm 1:** Parallel Extraction for Long-Horizon Disassembly

**Input:** Assembly $\mathcal{A}$ with objects $\{o_1, \ldots, o_N\}$, batch size $B > 0$, initial budget $T_0$, budget cap $T_{\max}$
**Output:** Disassembly sequence $S$

1 $\mathcal{Q} \leftarrow$ SHUFFLE($\{o_1, \ldots, o_N\}$) // Initialize queue
2 $S \leftarrow \emptyset$ // Solved sequence
3 $T \leftarrow T_0$ // Current planning budget
4 $\mathcal{F} \leftarrow \emptyset$ // Already attempted at budget $T$
5 **while** $\mathcal{Q} \neq \emptyset$ **do**
6 batch $\leftarrow$ POPFRONT($\mathcal{Q}, B$) // Up to $B$ objects
7 processes $\leftarrow \emptyset$
8 **foreach** $o_i \in$ *batch* **do**
9 remaining $\leftarrow \mathcal{Q} \cup$ (batch $\setminus \{o_i\}$)
10 $p_i \leftarrow$ LAUNCHPLANNER($o_i, S,$ remaining$, T$)
11 processes $\leftarrow$ processes $\cup \{(p_i, o_i)\}$
12 **end**
13 winner $\leftarrow$ WAITFORFIRSTSUCCESS(processes$, T$)
14 **if** *winner* $\neq$ *null* **then**
15 $(p^*, o^*) \leftarrow$ winner
16 $S \leftarrow S \cup \{o^*\}$ // Record solved object
17 TERMINATE($\{p_i \mid (p_i, o_i) \in$ processes, $o_i \neq o^*\}$) // Terminate losers
18 REQUEUE($\mathcal{Q}, \{o_i \mid (p_i, o_i) \in$ processes, $o_i \neq o^*\}$)
19 $T \leftarrow T_0$, $\mathcal{F} \leftarrow \emptyset$ // Geometry changed: reset
20 **else**
21 TERMINATE(processes) // Budget elapsed
22 REQUEUE($\mathcal{Q}$, batch)
23 $\mathcal{F} \leftarrow \mathcal{F} \cup$ batch // Attempted at $T$
24 **if** $\mathcal{Q} \subseteq \mathcal{F}$ **then**
25 $T \leftarrow 2T$, $\mathcal{F} \leftarrow \emptyset$ // Sweep failed: escalate
26 **if** $T > T_{\max}$ **then return** $S$
27 **end**
28 **end**
29 **end**
30 **return** $S$

If instead the budget elapses with no winner (lines 21–26), all processes are terminated and the entire batch is requeued (line 22), so that difficult objects do not block progress indefinitely and instead receive repeated attempts as the geometry evolves. The batch is added to $\mathcal{F}$ (line 23) to record that its objects have now been attempted at the current budget. Once $\mathcal{F}$ covers the queue (line 24), one full *sweep* over all unsolved objects has failed at budget $T$, and only then is the budget doubled and $\mathcal{F}$ cleared (line 25). Escalating per sweep rather than per batch matters in practice: an object entering a batch for the first time is never charged an inflated budget that it has not yet been shown to need. The cap $T_{\max}$ (line 26) bounds this escalation so that the algorithm also terminates on assemblies that admit no sequential disassembly at all.

### B. Scale-Invariant Sampling Integration

Each single-object planner in PEEL is an instance of MAB-RRT [6], an RRT whose sampler is selected at each iteration by a multi-arm bandit. We summarize the components here and refer to [6] for the full derivation. The *scale sampler* estimates the local free scale of the configuration space around a tree node: starting from an initial sphere radius $r_0$, it grows or shrinks the radius by fixed multiplicative factors until roughly half of a batch of sphere samples are collision-free, the point of maximum information entropy, yielding a scale $r^\star$ at which the surrounding free space is best resolved. The *PCA sampler* then exploits this scale: principal component analysis on the valid samples collected at $r^\star$ gives the dominant direction of local free space, which biases tree extensions along the narrow escape corridor, with the sampling cylinder extended by $h_{\text{ext}} = \delta \cdot r^\star$ to reach further along the passage. The principal component is recomputed online as new valid samples accumulate. A sliding-window upper confidence bound (UCB) bandit arbitrates between the uniform, scale, and PCA samplers based on the reward each provides, so the planner explores when no corridor is evident and exploits the estimated direction once one is found.

### C. Probabilistic Completeness

We argue that PEEL retains probabilistic completeness (PC) [26] on sequentially decomposable assemblies, as defined in Sec. III. This is true because we ensure that every object is chosen infinitely many times while using a probabilistically complete planner for which the timeout is increased to infinity. Let us formalize this:

**Proposition 1.** *Let $X$ be a sequentially decomposable set of objects. Then PEEL with $T_{\max} = \infty$ returns a complete disassembly sequence from $X$ with probability one.*

*Proof Sketch.* Let $X$ be a sequentially decomposable set of $K$ objects. Since it admits an ordering, there must be at least one object $o_k \in X$ which is removable. Let us show that $o_k$ is removed with probability one by PEEL. First, since $o_k$ is in the queue $\mathcal{Q}$, we will enter the while loop in Alg. 1 and stay there until $o_k$ is removed. Since the batch size $B$ is positive and every object that fails is requeued, the queue cycles $o_k$ is selected infinitely often. In each iteration, where $o_k$ is chosen, MAB-RRT is applied to find a solution. If no solution is found, the timeout is increased and will approach infinity. Since $o_k$ is removable and MAB-RRT is probabilistically complete [6], it will find a solution with probability one. Once $o_k$ is removed, it is removed from the queue. Removing $o_k$ from an ordering of $X$ leaves the remaining objects so that the remainder is again sequentially decomposable. □

## V. ROBOT GROUNDING PIPELINE

While PEEL computes collision-free paths in the object's configuration space, executing those paths on a robot requires bridging planning in object-space and planning in robot-space. To accomplish this, we introduce the robot execution pipeline (illustrated in Fig. 3). This pipeline handles two planning modes: robot-only motion, where the manipulator moves independently while the object remains stationary, and attached motion, where the grasped object moves rigidly with the end-effector. To simplify grasping, we use a mobile manipulator equipped with a spherical grasping tool that simulates stable point contacts on arbitrarily shaped surfaces.

PEEL produces a solution queue $\sigma$ defining the extraction order and a set of object-space trajectories. For each object dequeued from $\sigma$, the pipeline executes five phases.

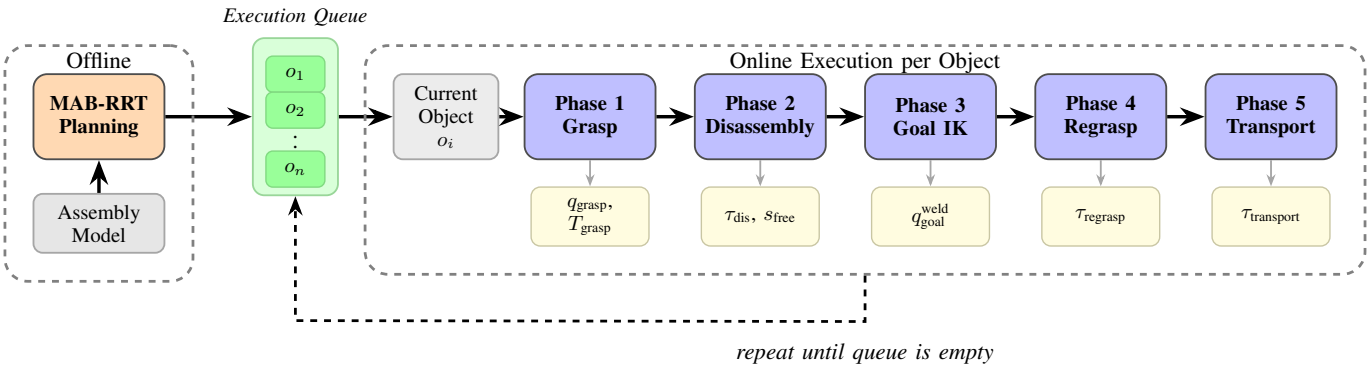



**Figure 3:** Robot execution pipeline for disassembly planning. PEEL uses MAB-RRT [6] to compute collision-free disassembly paths offline and produce an execution queue $\sigma$ defining the extraction order. Objects are dequeued sequentially, and for each object $o_i$ the pipeline executes five phases: (1) grasp planning, (2) disassembly motion following the pre-computed path until the object is free, (3) goal inverse kinematics computation, (4) regrasp motion to find a configuration valid at both the current (post-disassembly) and goal poses, and (5) welded transport to the goal. The loop continues until the queue is empty.

**Phase I: Grasp planning:** The phase begins from the robot's current configuration $q_{\text{init}}$. Grasp candidates are sampled near the object surface with a fixed margin $\mu_g$ and validated through three sequential checks, any of which may reject the candidate and trigger sampling of the next: (i) the candidate end-effector pose must yield a kinematically feasible robot configuration, that is, an inverse-kinematics solution must exist within joint limits; (ii) the resulting configuration must be collision-free with respect to the rest of the assembly, the environment, and the robot itself; and (iii) there must exist a collision-free arm trajectory from $q_{\text{init}}$ to the grasp configuration. The phase records the grasp configuration $q_g$ together with the rigid transform

$$T_{\text{grasp}} \;=\; (T_{\text{obj}}^{(0)})^{-1} \cdot \text{FK}(q_g), \tag{1}$$

which expresses the end-effector pose in the object's local frame and is reused throughout Phase II.

**Phase II: Disassembly motion:** MAB-RRT plans the object's motion as a tree growing from the start configuration into free space, but the path it returns does not explicitly indicate where the narrow-passage portion of the extraction ends and where the object becomes effectively unconstrained. We require this boundary because the goal of Phase II is the narrow-passage portion only: once the object can move freely, the remaining transport to the goal pose can be replanned by a standard RRT in Phase V without the directional bias of MAB-RRT, often along a much shorter path than the one emitted by the search tree.

A first instinct is to cut the path at the first or last waypoint of a particular sampling arm, but this is unreliable. By the time the object is free the search has long converged on the PCA arm and keeps extending toward the distant goal $G_i$, so the last waypoint typically sits far outside the assembly, in mid-air (Fig. 4), and the uniform arm can place a valid sample almost anywhere. Neither arm gives a reliable cut between the narrow-passage and free-space portions of the path.

We instead detect the free state by probing the object's local *mobility* at each waypoint. Around the current object pose $T_{\text{obj}}^{(k)}$, we perturb the object by $\pm\epsilon_t$ along each translational axis and by $\pm\epsilon_r$ about each rotational axis, collision-checking every perturbation against the rest of the assembly. Collecting collision-free directions, the translational mobility rank $r_t \in \{0, \ldots, 3\}$ is the dimension of the subspace they span, computed as the numerical rank of that set of directions; the rotational mobility rank $r_r \in \{0, \ldots, 3\}$ is defined analogously. The waypoint is accepted as a free state when $r_t \geq \rho_t$ and $r_r \geq \rho_r$. With $\rho_t = 3$ and $\rho_r = 2$, the criterion requires the collision-free translational directions to span all three dimensions and the rotational ones to span at least two. This is a conservative geometric definition of *escaped*: the object is unobstructed by the surrounding parts in a local neighborhood, and the manipulator can therefore reorient and translate it with a standard motion planner from this point onward.

The trajectory is followed by mapping each accepted waypoint to a target end-effector pose using

$$T_{\text{ee}}^{\text{target}} \;=\; T_{\text{obj}}^{(k)} \cdot T_{\text{grasp}}, \tag{2}$$

and solving inverse kinematics with the previous configuration as a seed to encourage smooth joint motion. Following a pre-computed object trajectory exposes the manipulator to several failure modes that do not arise during free-space planning. The joint configuration may approach a joint limit as the object rotates relative to the base; the end-effector may approach the boundary of the reachable workspace; the arm may pass through a kinematic singularity at which small target motions demand large joint motions; and the arm itself may collide with another part of the assembly even when the object remains collision-free along its trajectory. Each of these manifests as an IK or collision failure at an otherwise valid object waypoint. We respond with an intermediate *regrasp*: the robot releases the object in place, a new grasp candidate is sampled and validated at the current object pose, a robot-only path is planned to the new grasp configuration, and trajectory following resumes with an updated grasp transform $T_{\text{grasp}}$. Regrasping changes which sub-region of the configuration space the manipulator must traverse to keep the object on its trajectory: a different grasp can sidestep an approaching joint limit, leave the workspace boundary, escape a singular pose, or route the arm clear of an obstacle that the previous grasp pinned it against. As a result, extractions whose total motion exceeds the kinematic,

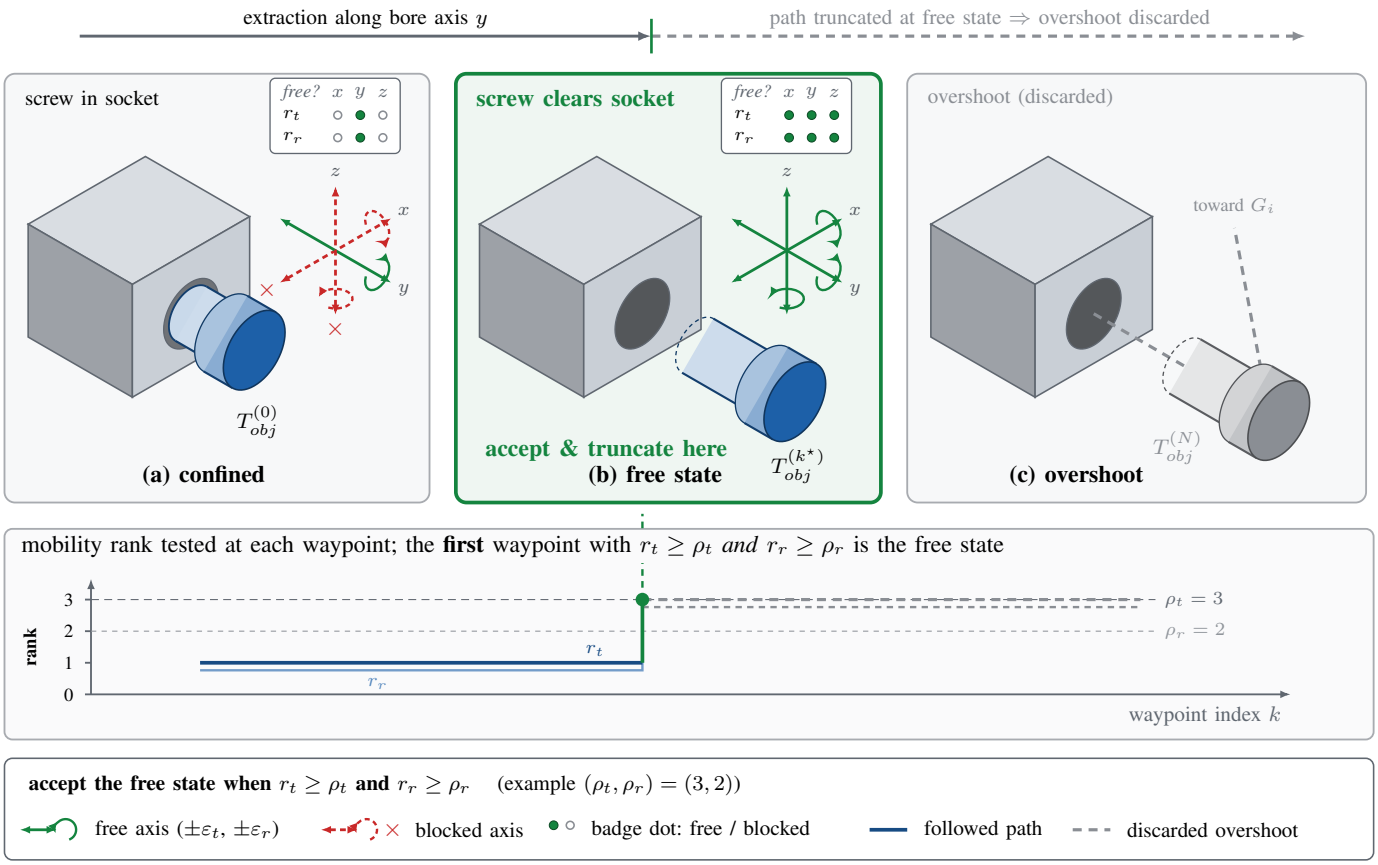



**Figure 4:** Free-state detection while withdrawing a screw from a socket, shown as three isometric stages (mobility rank is inherently three-dimensional, hence the 3D view) with, below, the mobility rank evaluated along the whole extraction path. At each waypoint the local mobility is probed with small $\pm\varepsilon_t$ translations along and $\pm\varepsilon_r$ rotations about the three object axes $x$, $y$, $z$, where $y$ is the bore axis. The translational (rotational) rank $r_t$ ($r_r$) is the dimension spanned by the free directions, which here are axis aligned, and the rank badge repeats it with one column per axis, so every dot names the arrow it stands for. In this example the same axes are free in translation and in rotation, so a single colour per axis carries both. **(a)** While the screw is still in the socket only the axial slide and the axial spin about $y$ are free ($r_t = 1$, $r_r = 1$); both lateral axes $x$ and $z$ are blocked by the socket wall. **(b)** The instant the screw clears the socket all six degrees of freedom open up ($r_t = 3$, $r_r = 3$), so this waypoint $T_{obj}^{(k^\star)}$ is accepted as the free state once $r_t \geq \rho_t$ and $r_r \geq \rho_r$ (here $(\rho_t, \rho_r) = (3, 2)$) and the path is truncated there. **(c)** The raw path continues in open space toward the distant goal region $G_i$, so this overshoot $T_{obj}^{(N)}$ is discarded and replanned as free-space transport. The bottom strip makes the mechanism explicit: the rank stays at 1 while the screw is in the socket and jumps to 3 the instant it clears; the first waypoint crossing the thresholds (green marker, aligned under panel (b)) is the detected free state.

workspace, or local-geometry reach of any single grasp can still complete in a single Phase II execution.

**Phase III: Goal IK computation:** The grasp transform $T_{\text{grasp}}$ accepted in Phase I was selected jointly with the start configuration of the object and the geometry of the surrounding parts. Once the object is free and the remaining task is to place it at its goal pose $T_{\text{obj}}^{\text{goal}}$, the same transform is unlikely to remain feasible: the workspace and the obstacles around the goal are different from those around the start, and the end-effector frame that was suitable for threading the object through a narrow passage may now intersect the table or fall outside the manipulator's reach. We therefore search for a fresh grasp at the goal pose, with the additional constraint that it must also be reachable from the object's current free-state pose so that the robot can switch to it before transport.

In this phase, we sample candidate end-effector poses $p_{\text{ee}}$ near the object surface (with margin $\mu_g$, as in Phase I) and treats each candidate as a tentative goal grasp. For a candidate to be accepted, two inverse-kinematics queries must succeed simultaneously. First, the candidate must yield a collision-free configuration $q_{\text{goal}}$ that places the end-effector at $T_{\text{obj}}^{\text{goal}} \cdot p_{\text{ee}}$, ensuring the object can be physically deposited at its target with the new grasp. Second, the same grasp transform $T'_{\text{grasp}} = (T_{\text{obj}}^{\text{goal}})^{-1} \cdot \text{FK}(q_{\text{goal}})$ must yield a collision-free configuration $q_{\text{br}}$ when applied at the object's current free-state pose $T_{\text{obj}}^{(*)}$, providing the configuration the robot needs to reach before it can re-attach to the object with $T'_{\text{grasp}}$ and transport it. Candidates that fail either query are discarded, and the loop resamples up to $N'_g$ times.

Requiring a single grasp transform to be simultaneously collision-free and within joint limits at both the free-state pose $T_{\text{obj}}^{(*)}$ and the goal pose $T_{\text{obj}}^{\text{goal}}$ is a nontrivial constraint. The two poses can demand geometrically incompatible approach directions, for example when a screw threaded out from above must be set down from the side, so that for some object and

goal pairs no shared grasp exists at all. Phase III searches explicitly for such a grasp, and its failure after $N'_g$ samples signals a genuine infeasibility rather than a mere sampling shortfall.

The resulting triple $(T'_{\text{grasp}}, q_{\text{goal}}, q_{\text{br}})$ defines the boundary configurations of the next two phases. This sequential approach (search at the goal first, then verify at the free state) is enabled by the purely geometric formulation introduced in Sec. III: in the absence of gravity and contact forces, the object can be presumed to remain at $T_{\text{obj}}^{(*)}$ while the robot relinquishes and re-attaches its grasp.

**Phase IV: Bridge motion:** Phase IV realizes the regrasp implied by Phase III. We plan a robot-only path from the Phase II final configuration to the bridge configuration $q_{\text{br}}$, during which the object is presumed to remain stationary at $T_{\text{obj}}^{(*)}$. At the end of the phase the end-effector reaches $T_{\text{obj}}^{(*)} \cdot T'_{\text{grasp}}$, that is, the new grasp transform applied at the object's current pose, and the robot is in position to re-attach to the object for transport.

If the planner fails to find such a path the phase reports failure and recovery requires re-invoking Phase III to draw a fresh goal grasp, since the $q_{\text{br}}$ accepted by Phase III was only tested for being collision-free at $T_{\text{obj}}^{(*)}$ and not for reachability from the Phase II end configuration.

**Phase V: Transport:** The object is rigidly welded to the end-effector via the new grasp $T'_{\text{grasp}}$ and transported to its goal pose by a plan from $q_{\text{br}}$ to $q_{\text{goal}}$. During this phase the object moves rigidly with the end-effector under the welding constraint, so collision checking is performed on the full robot-and-object assemblage rather than on the manipulator alone. The pipeline then dequeues the next object and repeats until $\sigma$ is empty.

## VI. Evaluation

We evaluate PEEL, MAB-RRT, and the robot execution pipeline on two complementary benchmark sets. First, we evaluate MAB-RRT on single-part disassembly using the Automate dataset [45]. This dataset contains over 100 mechanical assemblies. However, since it includes initial collisions while our method requires strict separation, we filter to the 76 collision-free models to establish a common benchmark. Each model is tested under 10 random rotations, yielding 760 trials. The benchmark is described in Sec. VI-A. Second, for multi-part disassembly, we evaluate on 4 assemblies with 10–17 removable parts also taken from the Automate dataset. The benchmark is described in Sec. VI-B. Finally, we demonstrate the robot execution pipeline by using the sequences from PEEL on the four assemblies as input. This is showcased in Sec. VI-D.

**Hardware and Implementation:** All experiments were conducted on Ubuntu 20.04.6 LTS with an Intel Core i7-5820K CPU (3.30 GHz, 12 cores) and 32 GB RAM. MAB-RRT [6] is implemented in C++ as an extension of OMPL [42]. All experiments, covering single-object benchmarks and the robot execution pipeline, run through Robowflex [24], a C++ wrapper around MoveIt that integrates OMPL planners and uses DART-Sim [28] for collision checking. Single-object benchmarks run single-threaded; multi-part batching (Sec. IV) uses 10 concurrent processes per batch. AssembleThemAll (ATA) [47] baselines were executed from the authors' repository[1] without modification.

**Table I:** Combined planner comparison across 76 collision-free Automate assemblies (760 trials, 10 random rotations each).

| Planner | Success Rate (%) | Mean (s) | Median (s) | Std Dev (s) | Runs |
|---|---|---|---|---|---|
| **MAB-RRT** | **100.0** | **4.31** | **3.20** | **3.79** | 760 |
| BFS | 53.9 | 44.59 | 38.23 | 31.26 | 760 |
| RRT (bridge) | 47.5 | 22.15 | 7.63 | 29.93 | 760 |
| RRT (gaussian) | 47.1 | 18.63 | 6.37 | 27.20 | 760 |
| RRT (obstacle) | 45.1 | 20.27 | 6.99 | 27.92 | 760 |
| TRRT | 23.0 | 54.72 | 48.05 | 40.06 | 760 |
| MateVec-TRRT | 22.2 | 56.68 | 49.71 | 39.59 | 760 |
| BK-RRT | 21.8 | 46.31 | 43.26 | 32.76 | 760 |
| RRT (uniform) | 11.4 | 57.52 | 49.45 | 49.13 | 760 |

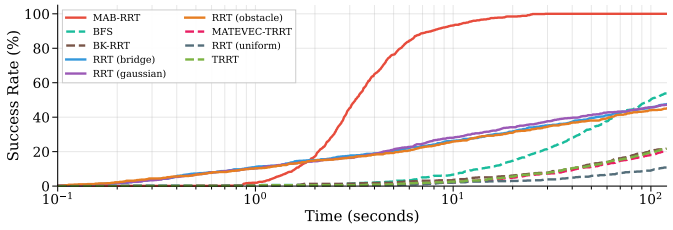



**Figure 5:** Success rate progression across 76 benchmark assemblies (760 trials, 10 random rotations per assembly). Each curve shows the fraction of the 760 trials that completed within elapsed wall-clock time $t$ (denominator fixed at 760). Solid lines represent MAB-RRT and RRT with informed sampling strategies (bridge, Gaussian, obstacle) from OMPL; dashed lines represent ATA baseline planners (BFS, BK-RRT, MateVec-TRRT, RRT uniform, TRRT). MAB-RRT achieves 100% success within seconds, significantly outperforming all baselines.

### A. Single-Object Benchmarks

We compare MAB-RRT against two sets of planners. The first set integrates classical sampling strategies into RRT [25]: bridge sampling [20], Gaussian sampling [7], and obstacle-based sampling [2]. The second set consists of modern strategies from the ATA [47] framework: mating vectors (MateVec-TRRT) [14], behavioral kinodynamic RRT (BK-RRT) [57], [47], disassembly breadth-first search (BFS) [47], Transition-based RRT (TRRT) [47], and plain uniform RRT. For the classical strategies, we use the default parameters as specified in OMPL [42], [30]. For the ATA strategies, we use the default parameters from the ATA software package.

**Success Criteria and Collision Resolution:** MAB-RRT requires an exact collision-free path from the initial configuration to a free-space goal. We set OMPL's `LongestValidSegmentFraction` to $10^{-4}$, forcing a collision check every $0.01\%$ of the workspace extent. ATA success is defined as termination in their planner-defined disassembled state with their standard step size of $10^{-2}$.

**Results:** To evaluate all planners on the 76 collision-free assemblies from the Automate dataset [45] (760 trials total), we use a per-trial timeout of 120 s. Tbl. I summarizes the results, Fig. 5 shows success rate progression over time, and Fig. 6 shows runtime distributions for successful runs.

[1] https://github.com/yunshengtian/Assemble-Them-All

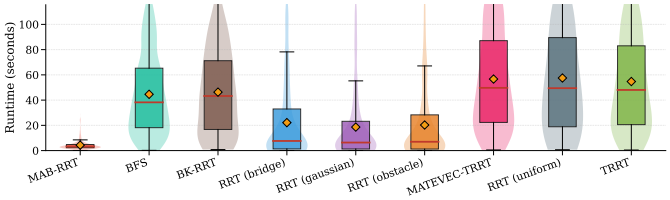



**Figure 6:** Runtime distribution for successful planning runs across all benchmark assemblies. Violin plots show the probability density of solution times, with overlaid box plots displaying quartile statistics. MAB-RRT achieves the fastest median and mean solution times with substantially lower variance than all baselines.

MAB-RRT achieves a 100% success rate with a mean runtime of 4.31 s and a median of 3.20 s. The next best planners are BFS (53.9%), RRT with bridge sampling (47.5%), RRT with Gaussian sampling (47.1%), and RRT with obstacle sampling (45.1%). The ATA-specific planners, MateVec-TRRT (22.2%), TRRT (23.0%), and BK-RRT (21.8%), all remain below 25% success rate, while plain uniform RRT solves only 11.4% of trials. The success rate progression in Fig. 5 shows that MAB-RRT reaches full success within the first few seconds, while all baselines plateau well below 60%. The curve is computed by counting, at each elapsed wall-clock time $t$, the fraction of the 760 trials (76 assemblies $\times$ 10 random rotations) that have completed within $t$. A trial completes when the planner returns a collision-free extraction path or reaches the per-trial wall-clock budget. The denominator is fixed at 760 throughout, so a curve plateauing at 0.6 corresponds to 456 successful trials out of 760.

### *B. Multi-Part Disassembly Benchmark*

To evaluate PEEL, we select four assemblies with 10 to 17 removable parts each: Microscope (12 parts), Disc Brake (10 parts), Coupling Block (14 parts), and Plier (17 parts). Parallel Extraction for Long-Horizon Disassembly uses a batch size of $B = 10$ concurrent processes (one CPU core per object), a per-object timeout of 180 s, and a batch timeout of 190 s. For the MAB-RRT planner [6], we use a window size of $W = 256$, an initial radius $r_0 = 1.0$, and a sample budget of $n = 128$ for the sphere sampler. The baselines (RRT, TRRT, MateVec-TRRT) execute objects sequentially on a single CPU core, following the default ATA disassembly pipeline [47].

For each assembly, the scheduler randomly shuffles the initial object queue and executes until all objects are removed or an overall timeout of 3600 s is reached. Each assembly is tested under 10 random initial shuffles to evaluate robustness to ordering. Fig. 7 shows the four assemblies in their assembled (left) and disassembled or exploded (right) states.

In the benchmarks, we report (1) total disassembly time (wall-clock time from start to complete disassembly) and (2) overall success rate (fraction of runs that fully disassemble the assembly within the overall timeout).

The results are summarized in Tbl. II. PEEL achieves a 100% success rate on all four assemblies with very few per-object timeouts (0–2 per assembly), while the baselines accumulate significantly more due to their sequential execution and less effective sampling in narrow passages. In terms of mean disassembly time, PEEL completes the Disc Brake

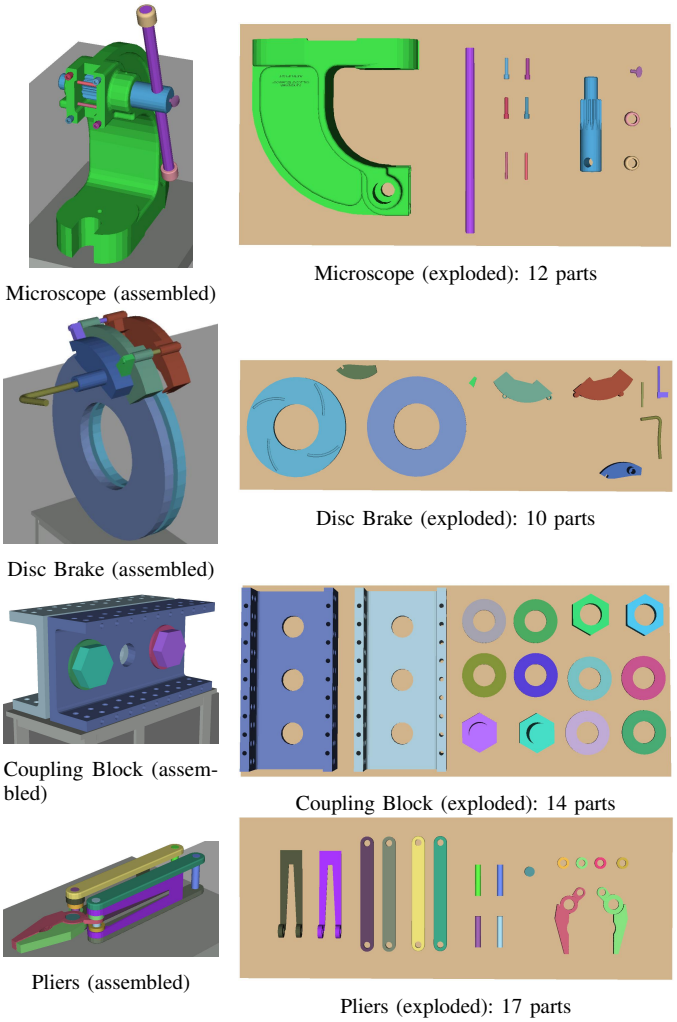



**Figure 7:** The four multi-part assemblies used for evaluation. Left column: assembled state. Right column: all parts separated.

assembly in 189 s (median 189 s), Plier in 406 s (median 397 s), Coupling Block in 419 s (median 411 s), and Microscope in 467 s (median 472 s). The next fastest baselines are MateVec-TRRT and TRRT, which require 2–5$\times$ longer on average, while plain RRT requires up to 7$\times$ longer.

Fig. 9 shows the total disassembly time distributions per assembly. Note that these wall-clock times reflect both the per-object planning quality and the execution protocol: PEEL benefits from parallel batching (10 cores) while the baselines run sequentially (1 core). PEEL consistently clusters at the low end of the time axis, while the baselines exhibit both higher means and substantially larger variance. PEEL's standard deviation ranges from 27 s (Disc Brake) to 110 s (Coupling Block), compared to 306–2446 s for the baselines.

Coupling Block is the most challenging assembly: RRT and TRRT achieve only 90% success rate, while MateVec-TRRT requires a mean time of 1877 s despite reaching 100% success. PEEL solves Coupling Block in 419 s mean time, demonstrating that scale-invariant sampling remains effective even in geometrically complex multi-part settings.

**Disassembly Progression.:** Fig. 8 shows the number of objects disassembled over time on a logarithmic time axis. PEEL (red) begins removing objects within the first second and completes

**Table II:** Environment-Specific Multi-Part Benchmark Results

| Environment | Planner | Success Rate (%) | Mean Time (s) | Median Time (s) | Std Time (s) | Total Runs | Num Timeouts |
|---|---|---|---|---|---|---|---|
| Microscope | **PEEL (ours)** | 100.0 | 467.03 | 472.05 | 44.11 | 10 | 0 |
| | RRT | 100.0 | 1742.87 | 1393.23 | 577.32 | 10 | 40 |
| | TRRT | 100.0 | 914.66 | 806.65 | 511.75 | 10 | 23 |
| | MATEVEC-TRRT | 100.0 | 818.80 | 814.62 | 378.40 | 10 | 20 |
| Disc Brake | **PEEL (ours)** | 100.0 | 188.78 | 189.49 | 27.19 | 10 | 0 |
| | RRT | 100.0 | 1326.93 | 1100.18 | 941.97 | 10 | 36 |
| | TRRT | 100.0 | 667.74 | 382.28 | 473.47 | 10 | 18 |
| | MATEVEC-TRRT | 100.0 | 552.67 | 373.89 | 305.89 | 10 | 15 |
| Coupling Block | **PEEL (ours)** | 100.0 | 419.45 | 410.56 | 109.78 | 10 | 2 |
| | RRT | 90.0 | 1865.87 | 1194.86 | 1386.14 | 10 | 64 |
| | TRRT | 90.0 | 2120.34 | 1479.24 | 1712.07 | 10 | 72 |
| | MATEVEC-TRRT | 100.0 | 1877.13 | 405.89 | 2446.48 | 10 | 51 |
| Plier | **PEEL (ours)** | 100.0 | 405.66 | 397.02 | 30.08 | 10 | 0 |
| | RRT | 100.0 | 2308.05 | 1920.38 | 1356.96 | 10 | 108 |
| | TRRT | 100.0 | 943.45 | 909.63 | 422.90 | 10 | 39 |
| | MATEVEC-TRRT | 100.0 | 1234.75 | 736.38 | 933.60 | 10 | 40 |

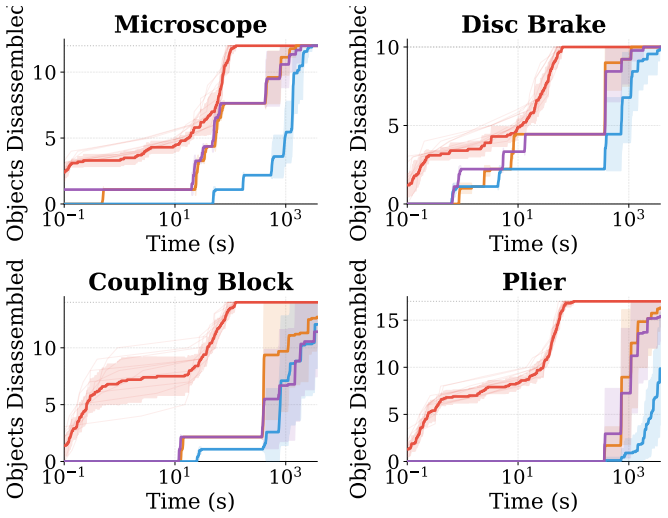



**Figure 8:** Objects disassembled over time (logarithmic time axis) for the four assemblies: PEEL, RRT, TRRT, MateVec-TRRT. PEEL removes objects within the first second and completes full disassembly one to two orders of magnitude faster than the baselines.

full disassembly one to two orders of magnitude faster than the baselines. The baselines (RRT, TRRT, MateVec-TRRT) remain near zero for the first 10–100 s before slowly progressing, with several runs failing to complete within the timeout. The winner-takes-all termination strategy effectively reduces wasted computation: once an object is successfully removed, all concurrent planners working on other objects are immediately terminated, allowing the system to quickly adapt to the updated geometric constraints. The requeuing mechanism ensures that difficult objects receive multiple attempts across successive batches as the assembly evolves.

### C. Parameter Sensitivity

We assess the robustness of MAB-RRT to its main hyperparameters on the single-object benchmark set (760 trials), varying one parameter at a time around the default configuration ($W = 256$, validity band $[10\%, 50\%]$, $r_0 = 1.0$, $n = 128$).

**Table III:** Window size robustness statistics across 760 benchmark trials (76 models $\times$ 10 rotations). Fixed parameters: $n = 128$, $r_0 = 1.0$, $r_{\max} = 15.0$, validity bounds $[0.10, 0.50]$.

| Window Size | Mean (s) | Median (s) | Std Dev (s) | Min (s) | Max (s) | Success Rate (%) |
|---|---|---|---|---|---|---|
| $W = 64$ | 5.77 | 4.37 | 4.67 | 1.14 | 32.51 | 99.7 |
| $W = 256$ | 6.26 | 4.57 | 5.21 | 1.19 | 43.05 | 100.0 |
| $W = 1024$ | 6.58 | 4.98 | 5.53 | 1.41 | 47.85 | 100.0 |

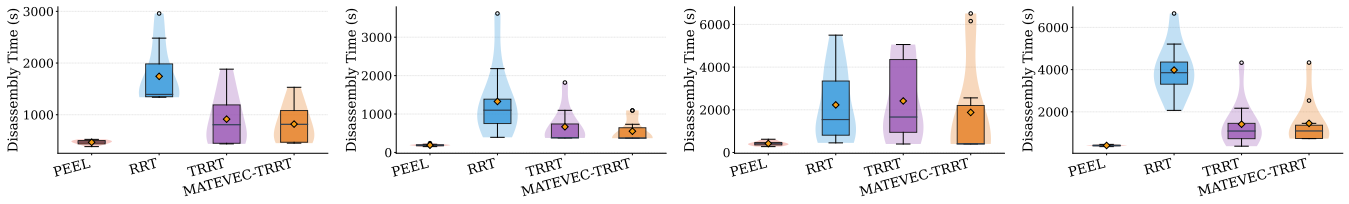
**Table IV:** MAB-RRT validity rate bounds sensitivity across 76 assemblies (760 trials).

| Bounds | Success Rate (%) | Mean (s) | Median (s) | Runs |
|---|---|---|---|---|
| [10%–50%] | 99.9 | 3.73 | 2.63 | 760 |
| [5%–15%] | 98.8 | 4.00 | 2.68 | 760 |
| [15%–25%] | 99.9 | 6.57 | 4.26 | 760 |
| [25%–50%] | 100.0 | 7.01 | 4.74 | 760 |

**Sliding-window size:** The UCB sliding window controls how much sampling history the bandit retains. Across $W \in \{64, \ldots, 1024\}$ the planner is essentially insensitive: median runtime stays at 4–5 s with success at or near 100%. Larger windows slightly favor exploitation in tightly constrained tasks while smaller windows switch samplers faster, but overall performance is comparable. We keep $W = 256$.

**Validity-rate bounds:** The target validity interval $[\alpha_{\min}, \alpha_{\max}]$ sets how aggressively the scale search shrinks or grows the sphere. Across $[5\%, 15\%]$, $[10\%, 50\%]$, $[15\%, 25\%]$ and $[25\%, 50\%]$ all settings exceed 98% success. Lower bounds accept larger radii with fewer valid samples and are fastest (median 2.63–2.68 s), while the highest bound forces further shrinking (4.74 s); the resulting $1.8\times$ spread still lies in a practical range. We use $[10\%, 50\%]$.

**Initial radius:** The initial sphere radius is corrected by the adaptive grow-shrink search, so runtime is largely insensitive to it: $r_0 \in [0.2, 5.0]$ all converge to 3–6 s median at near-100% success. Only extreme values degrade, $r_0 \leq 0.1$ raises adaptation overhead and lowers success to 88–94%, and $r_0 = 15.0$ keeps full success but raises median runtime to 6.2 s. We




**(a)** Microscope **(b)** Disc Brake **(c)** Coupling Block **(d)** Pliers

**Figure 9:** Total disassembly runtime comparison per assembly. Violin plots with overlaid box plots compare PEEL, RRT, TRRT, and MateVec-TRRT across 10 runs each. PEEL consistently achieves an order-of-magnitude speedup with minimal variance.

**Table V:** Radius sensitivity statistics across 760 benchmark trials. Fixed parameters: $n = 128$, $W = 256$, validity bounds $[0.10, 0.50]$.

| Initial Radius | Mean (s) | Median (s) | Std Dev (s) | Min (s) | Max (s) | Success Rate (%) |
|---|---|---|---|---|---|---|
| $r_0 = 10^{-6}$ | 12.19 | 7.75 | 13.57 | 2.07 | 114.25 | 90.8 |
| $r_0 = 0.01$ | 8.72 | 4.92 | 12.73 | 1.02 | 117.77 | 88.1 |
| $r_0 = 0.10$ | 7.63 | 4.25 | 12.10 | 0.59 | 114.29 | 93.5 |
| $r_0 = 0.20$ | 5.26 | 3.40 | 6.65 | 0.56 | 98.79 | 97.8 |
| $r_0 = 0.50$ | 5.00 | 3.67 | 4.24 | 0.89 | 28.34 | 100.0 |
| $r_0 = 1.00$ | 5.61 | 4.31 | 4.48 | 1.24 | 31.85 | 100.0 |
| $r_0 = 2.10$ | 6.72 | 5.31 | 4.53 | 1.89 | 27.57 | 100.0 |
| $r_0 = 2.50$ | 7.05 | 5.23 | 6.27 | 1.66 | 51.34 | 99.9 |
| $r_0 = 5.00$ | 7.10 | 5.38 | 6.11 | 1.81 | 50.74 | 99.9 |
| $r_0 = 15.0$ | 7.81 | 6.22 | 5.80 | 2.55 | 42.16 | 100.0 |

**Table VI:** Sample size impact statistics across 760 benchmark trials. Fixed parameters: $r_0 = 1.0$, $r_{\max} = 15.0$, $W = 256$, validity bounds $[0.10, 0.50]$.

| Sample Size | Mean (s) | Median (s) | Std Dev (s) | Min (s) | Max (s) | Success Rate (%) |
|---|---|---|---|---|---|---|
| $n = 64$ | 2.35 | 1.69 | 2.42 | 0.36 | 39.48 | 99.1 |
| $n = 128$ | 3.33 | 2.48 | 2.79 | 0.60 | 19.40 | 100.0 |
| $n = 256$ | 5.58 | 4.27 | 4.39 | 1.19 | 31.01 | 100.0 |

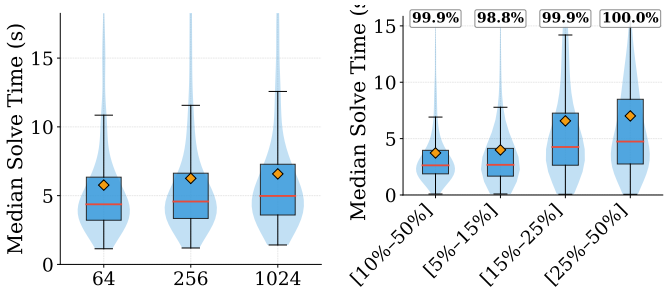
use $r_0 = 1.0$.

**Sample size:** The number of quasi-random samples per sphere trades directional coverage against validation cost. $n = 128$ gives the best balance (100% success, 2.5 s median); $n = 64$ is faster (1.7 s) at slightly lower reliability (99.1%); and $n = 256$ adds collision checks that slow the scale search by roughly $1.7\times$ without improving success. We adopt $n = 128$.

### *D. Robot Execution Demonstration*

After showing that PEEL can successfully generate object sequences to be disassembled, we demonstrate next how those sequences can be translated into feasible robot motions. To validate this, we deployed the five-phase execution protocol (Sec. V) on all four multi-part assemblies in DARTSim [28] via Robowflex [24].

The robot is a simulated Fetch mobile manipulator with 11 degrees of freedom: a 7-DOF arm, a 1-DOF torso lift, and a 3-DOF mobile base. The end-effector is equipped with a spherical grasping tool that simulates stable point contacts on arbitrarily shaped surfaces. During extraction and transport, the grasped object is rigidly attached to the end-effector via a weld joint, which eliminates the need for grasp stability analysis and ensures the object cannot slip during motion. Collision




**(a)** Sliding-window size $W$ **(b)** Validity bounds $[\alpha_{\min}, \alpha_{\max}]$

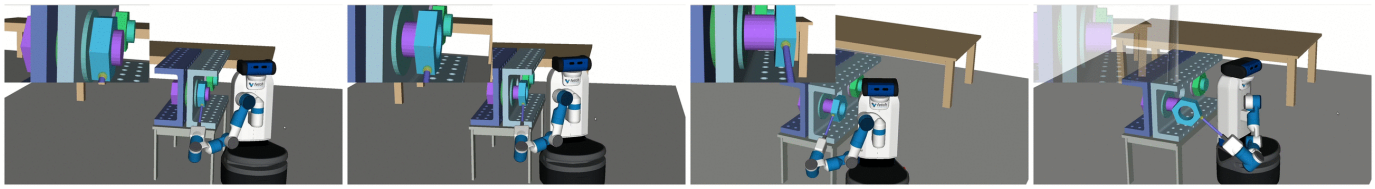
**Figure 10:** Parameter sensitivity of MAB-RRT on the single-object benchmark.

checking remains active throughout all phases: every candidate robot configuration is validated against the full environment, including the assembly, other objects, and the table.

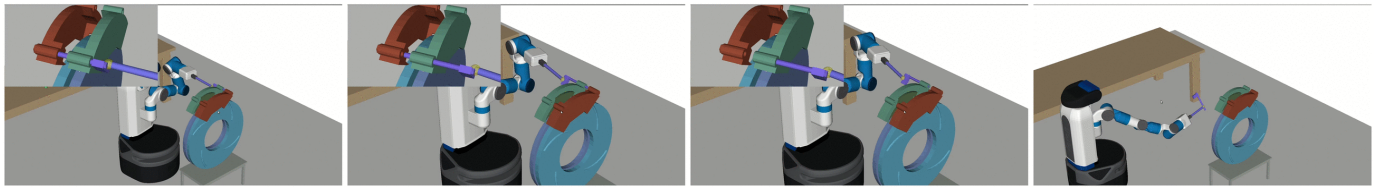
**Parameters:** The following parameters are specific to the five-phase online execution protocol (Sec. V); offline PEEL and MAB-RRT settings are as reported in Secs. VI-B and VI-C.

- **PCA cylinder extension factor** ($\delta$): Scales the principal-component sampling cylinder as $h_{\text{ext}} = \delta \cdot r^\star$ during offline MAB-RRT path generation.
- **Grasp surface sample budget**: Maximum number of random surface points drawn from each object mesh when searching for a valid grasp in Phase I (and during intermediate regrasps in Phase II). Value: 5000.
- **Grasp sampling margin** ($\mu_g$): Offset from the object surface at which candidate end-effector poses are placed before IK and collision validation (Phases I and III).
- **IK seed configuration**: Previous accepted robot configuration used to initialize inverse kinematics at each object waypoint in Phase II, encouraging smooth joint trajectories.
- **Mobility perturbation magnitude** ($\epsilon_t$): Displacement size used to probe translational and rotational object mobility when detecting the free-state boundary in Phase II.
- **Goal grasp sample budget** ($N_g'$): Maximum number of candidate goal grasps sampled and validated in Phase III before reporting failure.

**Results:** All four assemblies (Microscope, Coupling Block, Disc Brake, and Plier) were fully disassembled in simulation. The regrasp mechanism proved essential for parts requiring long extraction motions that exceeded the manipulator's workspace from a single grasp configuration. The free-state detection successfully identified the transition from constrained

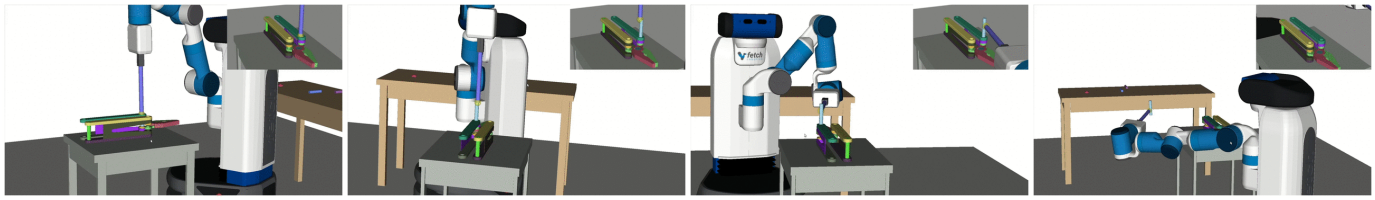


**(a)** Coupling Block



**(b)** Disc Brake

**(c)** Microscope



**(d)** Plier

**Figure 11:** Robot demonstrations using the robot execution pipeline. The disassembly sequences are computed using PEEL and individual escape paths computation uses scale-invariant sampling inside MAB-RRT. All sequences were successfully executed using the Fetch robot.

to unconstrained motion in each extraction, allowing the transport phase to plan freely rather than follow the remainder of the narrow-passage trajectory. The demonstrations confirm that the geometric extraction paths computed by MAB-RRT can be executed on a kinematic robot: the constrained, narrow-passage portion is followed directly through inverse kinematics, while regrasping absorbs kinematic and workspace limits and the remaining free-space transport is replanned. Individual frames of the robot disassembly sequences are shown in Fig. 11 including the microscope (Fig. 11c), the disc brake (Fig. 11b), the coupling block (Fig. 11a), and the pair of pliers (Fig. 11d). Full videos are available on our website[2].

## VII. Conclusion

We presented the Parallel Extraction for Long-Horizon Disassembly (PEEL), a framework that computes geometrically feasible multi-part disassembly sequences and executes them on a robot manipulator. PEEL discovers removal orders through concurrent single-object planning races, each solved by MAB-RRT with scale-invariant sampling to separate parts through tight, narrow passages, and avoids the exponential precedence search of combinatorial sequencing. We demonstrated a 100% success rate on 76 single-object benchmarks and on four multi-part assemblies (Microscope, Disc Brake, Coupling Block, and Plier, 10 to 17 parts each), achieving the lowest runtime against three baselines, and we showed that the resulting sequences are executable by a simulated Fetch manipulator through the five-phase execution pipeline.

While the results are promising, two important opportunities remain for strengthening this approach and pushing our work towards real world deployment. Particularly, those are:

- **Physical Simulation**: Our framework focuses on geometrically feasible paths to achieve a disassembly state. We believe this is an important and necessary step towards the efficient disassembly of real-world objects. Future work will concentrate on using geometrically feasible paths from PEEL as heuristics to guide manipulation systems towards dynamically feasible paths.
- **Non-sequential Disassembly**: Our system currently handles objects which are sequentially decomposable. However, some objects are kinematically interlocked [56], [47] (like the Burr puzzle [50]) meaning that two or more objects have to be moved *simultaneously* to achieve a disassembly state. A straightforward extension of PEEL would involve running MAB-RRT on combinations of pairs (or higher order tuples) if no single-object solutions are found after some time.

[2] https://peel-disassembly.surge.sh/

**Table VII:** Recommended MAB-RRT configuration.

| Group | Parameter | Value |
|---|---|---|
| Goal bias | uniform arm goal bias | 0.5 |
| | PCA arm goal bias | $10^{-7}$ |
| Bandit | sliding-window size $W$ | 256 |
| | forced-uniform streak | 5 |
| Scale search | sample budget $n$ | 128 |
| | initial radius $r_0$ | 1.0 |
| | radius clamp $[r_{\min}, r_{\max}]$ | $[10^{-6}, 15.0]$ |
| | shrink factor $s$ | $\exp(-0.7)$ |
| | grow factor $g$ | $\exp(0.9)$ |
| | validity band $[\alpha_{\min}, \alpha_{\max}]$ | [0.10, 0.50] |
| | max burn-in steps | 50 |
| | Fibonacci-lattice jitter | $\pi/8$ |
| PCA sampler | online PCA recompute | on |
| | cylinder radius offset mult. | 0.1 |
| | cylinder sampling radius mult. | 1.5 |
| | height extension $\varepsilon$ | 2.0 |
| Pre-check / exit | free-sampling probability | 0.20 |
| | uniform pre-check trials | 0 |
| | burn-in / full-validity early exit | off |
| Rewards | uniform valid / invalid | $10^7$ / 0 |
| | PCA valid / invalid | 1.0 / 0 |

**Table VIII:** Per-experiment configuration behind each results table. The parameter varied by a sensitivity study is marked *var.*; all others are held at the listed value. Shrink / grow factors are $\exp(-0.7)$ / $\exp(0.9)$ throughout.

| Experiment | $W$ | $n$ | $r_0$ | $[\alpha_{\min}, \alpha_{\max}]$ |
|---|---|---|---|---|
| Single-object benchmark | 256 | 128 | 1.0 | $[0.10, 0.50]$ |
| Window size | *var.* | 256 | 1.0 | $[0.10, 0.50]$ |
| Initial radius | 256 | 256 | *var.* | $[0.10, 0.50]$ |
| Sample size | 256 | *var.* | 1.0 | $[0.10, 0.50]$ |
| Validity band | 128 | 128 | 1.0 | *var.* |
| Multi-part | 256 | 128 | 1.0 | $[0.10, 0.50]$ |

Despite those open opportunities, PEEL already provides a general-purpose framework for disassembly problems and we have shown that the resulting paths are executable with a manipulator robot. We therefore believe we have made a significant step towards general-purpose disassembly frameworks for robots in the real-world.

## Appendix A
## Planner Configurations

This appendix documents the exact planner configuration behind every result. Tbl. VII gives the recommended MAB-RRT configuration, grouped by role. Every listed key is read and used by the planner (`loadYAMLConfig` in `MAB_RRT.cpp`); the released configuration files are byte-faithful to the runs and record the source file and its checksum. Most studies share this base, but a few were executed on a different one; Tbl. VIII therefore lists the configuration actually used for each results table, so every number is traceable to its exact settings.

The per-trial timeout is 120 s for all single-object studies. The multi-part study additionally uses a batch size of $B = 10$ concurrent planners, a per-object timeout of 180 s, a batch timeout of 190 s, and an overall run timeout of 3600 s.

### A. Sensitivity sweeps

Tbl. IX lists the value set explored by each sensitivity study. Each study varies the single axis shown while the remaining parameters stay at the per-experiment values in Tbl. VIII.

**Table IX:** Parameter-sensitivity sweeps. One axis varies per study.

| Study | Axis | Values |
|---|---|---|
| Window size | $W$ | $\{64, 256, 1024\}$ |
| Initial radius | $r_0$ | $10^{-6}, 0.01, 0.1, 0.2, 0.5, 1.0, 2.1, 2.5, 5.0, 15.0$ |
| Sample size | $n$ | $\{64, 128, 256\}$ |
| Validity band | $[\alpha_{\min}, \alpha_{\max}]$ | $[0.05, 0.15]$, $[0.10, 0.50]$, $[0.15, 0.25]$, $[0.25, 0.50]$ |

**Table X:** Per-assembly free-state thresholds (multi-part pipeline).

| Assembly | $(\rho_t, \rho_r)$ | $\epsilon_t$ | $\epsilon_r$ |
|---|---|---|---|
| Microscope (00003) | $(2, 2)$ | 0.15 m | $10^\circ$ |
| Coupling Block (04370) | $(3, 2)$ | 0.08 m | $6^\circ$ |
| Disc Brake (00580), Plier (zange) | $(2, 2)$ | 0.10 m | $7^\circ$ |

### B. Multi-part free-state condition

The multi-part pipeline reuses the default planner configuration and adds a per-assembly free-state test (Phase II termination): a part is free once its translational and rotational mobility ranks reach $(\rho_t, \rho_r)$, probed with step sizes $(\epsilon_t, \epsilon_r)$. Tbl. X gives the values; assemblies not listed use the default $(\rho_t, \rho_r) = (2, 2)$.

### C. Robot execution pipeline

The five-phase execution protocol adds the parameters in Tbl. XI; the offline PEEL and MAB-RRT settings are unchanged from Tbls. VII and VIII. The free-state thresholds $(\rho_t, \rho_r)$ and probe sizes $(\epsilon_t, \epsilon_r)$ are per-assembly and are given in Tbl. X; the cylinder height extension is the MAB-RRT default ($\varepsilon = 2.0$).

The full configuration files are released with the code (see `reproducibility/README.md`).

## References


[1] I. Aguinaga, D. Borro, and L. Matey, "Parallel rrt-based path planning for selective disassembly planning," *The International Journal of Advanced Manufacturing Technology*, vol. 36, no. 11, pp. 1221–1233, 2008.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Robotics: The Algorithmic Perspective*, P. K. Agarwal, L. E. Kavraki, and M. T. Mason, Eds. CRC Press, 1998, pp. 155–168.

[3] M. E. Asif, A. Rastegarpanah, and R. Stolkin, "Robotic disassembly for end-of-life products focusing on task and motion planning: A comprehensive survey," *Journal of Manufacturing Systems*, vol. 77, pp. 483–524, 2024.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[5] S. B. Bayraktar, A. Orthey, Z. Kingston, M. Toussaint, and L. E. Kavraki, "Solving rearrangement puzzles using path defragmentation in factored state spaces," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4529–4536, 2023.

[6] S. B. Bayraktar, A. Orthey, and M. Toussaint, "Scale-invariant sampling in multi-arm bandit motion planning for object extraction," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2026.

[7] V. Boor, M. H. Overmars, and A. F. Van Der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1018–1023.

[8] S. Bubeck, N. Cesa-Bianchi *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

**Table XI:** Robot execution pipeline parameters (five-phase protocol).

| Phase | Parameter | Value |
|---|---|---|
| I: Grasp | grasp sampling margin $\mu_g$ | 0.05 m |
| | grasp surface sample budget | 5000 |
| | grasp resample rounds | 10 |
| III: Goal IK | goal-grasp resample rounds $N'_g$ | 5 |
| | goal / bridge IK attempts per round | 100 / 100 |
| Planners | approach (Phase I) | RRTConnect |
| | bridge (Phase IV) | RRTConnect |
| | transport (Phase V) | RRTConnect |


[9] B. Burns and O. Brock, "Toward optimal configuration space sampling." in *Robotics: Science and Systems*, vol. 1. Cambridge, USA, 2005, pp. 105–112.

[10] A. Cebulla, T. Asfour, and T. Kröger, "Speeding up assembly sequence planning through learning removability probabilities," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 388–12 394.

[11] J. Cortés, L. Jaillet, and T. Siméon, "Disassembly path planning for complex articulated objects," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 475–481, 2008.

[12] S. Dalibard and J.-P. Laumond, "Control of probabilistic diffusion in motion planning," in *Algorithmic foundation of robotics VIII: selected contributions of the eight international workshop on the algorithmic foundations of robotics*. Springer, 2009, pp. 467–481.

[13] ——, "Linear dimensionality reduction in random motion planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1461–1476, 2011.

[14] T. Ebinger, S. Kaden, S. Thomas, R. Andre, N. Amato, and U. Thomas, "A general and flexible search framework for disassembly planning," in *IEEE International Conference on Robotics and Automation*. IEEE, Sep. 2018, pp. 3548–3555.

[15] M. Faroni and D. Berenson, "Motion planning as online learning: A multi-armed bandit approach to kinodynamic sampling-based planning," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6651–6658, 2023.

[16] ——, "Online adaptation of sampling-based motion planning with inaccurate models," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 2382–2388.

[17] X. Guo, J. Liu *et al.*, "Disassembly sequence planning: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 7, pp. 1308–1324, 2021.

[18] L. S. Homem de Mello and A. C. Sanderson, "AND/OR graph representation of assembly plans," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 188–199, 1990.

[19] ——, "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.

[20] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, vol. 3, 2003, pp. 4420–4426.

[21] C. H. Huang, P. Jadhav, B. Plancher, and Z. Kingston, "pRRTC: GPU-parallel RRT-connect for fast, consistent, and low-cost motion planning," 2025.

[22] J. Ichnowski and R. Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.

[23] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed RRT for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 5088–5095.

[24] Z. Kingston and L. E. Kavraki, "Robowflex: Robot motion planning with MoveIt made easy," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 3108–3114.

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

[26] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[27] D. T. Le, J. Cortes, and T. Simeon, "A path planning approach to (dis)assembly sequencing," in *International Conference on Automation Science and Engineering*, 2009, pp. 286–291.

[28] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[29] R. Li, D. T. Pham, J. Huang, Y. Tan, M. Qu, Y. Wang, M. Kerin, K. Jiang, S. Su, C. Ji *et al.*, "Unfastening of hexagonal headed screws by a collaborative robot," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1455–1468, 2020.

[30] M. Moll, I. A. Şucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, September 2015.

[31] T. Motoda, D. Petit, T. Nishi, K. Nagata, W. Wan, and K. Harada, "Multi-step object extraction planning from clutter based on support relations," *IEEE Access*, vol. 11, pp. 45 129–45 139, 2023.

[32] S. Münker and R. H. Schmitt, "CAD-based AND/OR graph generation algorithms in (dis)assembly sequence planning of complex products," in *Procedia CIRP*, vol. 106, 2022, pp. 144–149.

[33] X. Niu, H. Ding, and Y. Xiong, "A hierarchical approach to generating precedence graphs for assembly planning," *International Journal of Machine Tools and Manufacture*, vol. 43, no. 14, pp. 1473–1486, 2003.

[34] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, no. 1, pp. 285–310, 2024.

[35] M. Otte and N. Correll, "C-FOREST: Parallel shortest path planning with superlinear speedup," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 798–806, 2013.

[36] A. Pathak and R. Muthusamy, "Collapse and collision aware grasping for cluttered shelf picking," *arXiv preprint arXiv:2503.22427*, 2025.

[37] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

[38] Y. Ren, C. Zhang, F. Zhao, H. Xiao, and G. Tian, "An asynchronous parallel disassembly planning based on genetic algorithm," *European Journal of Operational Research*, vol. 269, no. 2, pp. 647–660, 2018.

[39] O. Salzman, M. Hemmer, and D. Halperin, "On the power of manifold samples in exploring configuration spaces and the dimensionality of narrow passages," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 529–538, 2014.

[40] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3–4, pp. 443–467, 2020.

[41] A. Slivkins *et al.*, "Introduction to multi-armed bandits," *Foundations and Trends in Machine Learning*, vol. 12, no. 1-2, pp. 1–286, 2019.

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

[43] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox, "CuRobo: Parallelized collision-free robot motion generation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8112–8119.

[44] S. Sundaram, I. Remmler, and N. M. Amato, "Disassembly sequencing using a motion planning approach," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2001, pp. 1475–1480.

[45] B. Tang, I. Akinola, J. Xu, B. Wen, A. Handa, K. Van Wyk, D. Fox, G. S. Sukhatme, F. Ramos, and Y. Narang, "Automate: Specialist and generalist assembly policies over diverse geometries," in *Robotics: Science and Systems*, 2024.

[46] Y. Tian, K. D. D. Willis, B. A. Omari, J. Luo, P. Ma, Y. Li, F. Javid, E. Gu, J. Jacob, S. Sueda, H. Li, S. Chitta, and W. Matusik, "Asap: Automated sequence planning for complex robotic assembly with physical feasibility," 2024.

[47] Y. Tian, J. Xu, Y. Li, J. Luo, S. Sueda, H. Li, K. D. Willis, and W. Matusik, "Assemble them all: Physics-based planning for generalizable assembly by disassembly," *ACM Trans. Graph.*, vol. 41, no. 6, 2022.

[48] Y. Wang and D. Tian, "A weighted assembly precedence graph for assembly sequence planning," *The International Journal of Advanced Manufacturing Technology*, vol. 83, pp. 99–115, 2016.

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

[50] S. Xin, C.-F. Lai, C.-W. Fu, T.-T. Wong, Y. He, and D. Cohen-Or, "Making burr puzzles from 3d models," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, pp. 1–8, 2011.

[51] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 3856–3861.

[52] A. Yershova and S. M. LaValle, "Motion planning for highly constrained spaces," in *Robot motion and control 2009*. Springer, pp. 297–306.

[53] L. Zhang, X. Huang, Y. J. Kim, and D. Manocha, "D-plan: Efficient collision-free path computation for part removal and disassembly," *Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 774–786, 2008.

[54] X. Zhang, K. Eltouny, X. Liang, and S. Behdad, "Automatic screw detection and tool recommendation system for robotic disassembly," *Journal of Manufacturing Science and Engineering*, vol. 145, no. 3, p. 031008, 2023.

[55] X. F. Zhang, G. Yu, Z. Y. Hu, C. H. Pei, and G. Q. Ma, "Parallel disassembly sequence planning for complex products based on fuzzy-rough sets," *The International Journal of Advanced Manufacturing Technology*, vol. 72, no. 1, pp. 231–239, 2014.

[56] Y. Zhang, Y. Koga, and D. Balkcom, "Interlocking block assembly with robots," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 902–916, 2021.

[57] S. Zickler and M. M. Veloso, "Efficient physics-based planning: sampling search via non-deterministic tactics and skills." in *AAMAS (1)*, 2009, pp. 27–33.