SeGMan: Sequential and Guided Manipulation Planner for Robust Planning in 2D Constrained Environments

Cankut Bora Tuncer[1,†], Dilruba Sultan Haliloglu[1,†], and Ozgur S. Oguz[1]

***Abstract*— In this paper, we present SeGMan, a hybrid motion planning framework that integrates sampling-based and optimization-based techniques with a guided forward search to address complex, constrained sequential manipulation challenges, such as pick-and-place puzzles. SeGMan incorporates an adaptive subgoal selection method that adjusts the granularity of subgoals, enhancing overall efficiency. Furthermore, proposed generalizable heuristics guide the forward search in a more targeted manner. Extensive evaluations in maze-like tasks populated with numerous objects and obstacles demonstrate that SeGMan is capable of generating not only consistent and computationally efficient manipulation plans but also outperform state-of-the-art approaches. https://sites.google.com/view/segman-lira/**

I. Introduction

Sequential manipulation is used in complex and constrained environments where tasks require multiple steps. It involves generating a feasible motion plan by decomposing a primary goal into manageable subgoals that can be addressed individually. However, in constrained environments, finding suitable subgoals is a challenging process that demands the identification of critical objects and placement locations. Due to the high dimensionality of the environments, the subgoal generation process requires coordinated reasoning through multiple stages of interaction, such as grasping, reorienting, pushing, and removing objects, in a structured manner. However, even minor errors, such as a misaligned object, can lead to failures in subsequent steps, necessitating re-planning and extending plan generation time. Thus, the quality of the generated subgoals not only directs the motion planning process but also impacts the robustness and efficiency of the final solution.

Several studies have investigated methods for identifying and addressing critical subgoals in sequential motion planning. [1] proposes a heuristic-driven search to identify intermediate subgoals, which are then solved using Task-and-Motion Planning (TAMP). However, the heuristics lack generalization, often misguiding the search process and increasing computation time. A different approach is taken in [2], where the subgoals are generated with a sampling-based method and utilized in motion plan optimization. While effective, this method struggles to propose feasible obstacle relocation points and has limited adaptability to varying environmental factors. In [3], [4], novel subgoal generation algorithms are proposed to decompose complex problems into simpler, more manageable subgoals. However, a common limitation among these approaches is that their proposed planners lack the capability for fine manipulation of objects in constrained environments. Thus, current methods do not offer a framework capable of identifying critical subgoals for constrained sequential manipulation while offering robustness and computational efficiency.

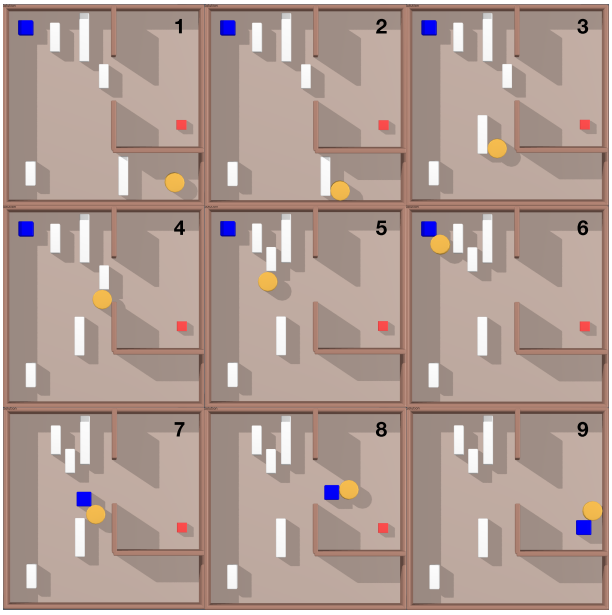

Fig. 1: The example illustrates a pick-and-place task where the agent (yellow) must move the goal object (blue) to the goal (red). Before doing so, it selectively removes the obstacles (white) to ensure it can access the goal object.

Fig. 1 illustrates a simplified task where a 2DOF agent (yellow cylinder) must execute sequential pick-and-place actions to move the goal object (blue square) to its goal (red square) by interacting with movable obstacles (white squares) [1]. The solution of the problem requires relocation of two critical movable obstacles (Fig. 1, 1-5) so that a feasible motion plan can be realized from agent to goal object (Fig. 1, 6) and goal object to the goal (Fig. 1, 7-9). This problem highlights the key challenges in sequential manipulation, including effective subgoal selection and precise object manipulation to iteratively solve each subgoal.

In this paper, we introduce SeGMan — Sequential and Guided Manipulation Planner — a hybrid motion planner that integrates sampling and optimization-based techniques

†These authors contributed equally to this work.
[1]Dept. of Computer Engineering, Bilkent University.
*This work was supported by TUBITAK under 2232 program with project number 121C148 ("LiRA").
Corresponding author: Cankut Bora Tuncer, Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey. Email: bora.tuncer@bilkent.edu.tr

with a heuristic-driven forward search. By using waypoints from a sampling-based planner as subgoals [2], SeGMan adaptively selects the subgoals to optimize object manipulation, which enhances efficiency across various constrained environments. Moreover, the proposed generalizable forward search heuristics guide the search on critical obstacles and relocation positions, improving robustness.

Our main contributions are as follows:

- A generalizable hybrid motion planning framework by introducing a guided search for obstacle manipulation in constrained environments.
- A novel algorithm that identifies critical obstacles and leverages generalizable heuristics to propose and evaluate obstacle relocation subgoals for guided forward search in scenarios where unguided search would be impractical.
- A method that adaptively selects subgoals and iteratively adjusts granularity, improving the robustness and efficiency of the framework.

The framework was evaluated across 15 distinct problems, similar to the task in Fig. 1, each presenting unique challenges with varying levels of complexity. Overall, SeGMan outperformed baseline methods in robustness and demonstrated a balance between computation time and solution quality.

II. Related Work

Motion planning for object manipulation involves determining a sequence of collision-free and feasible configurations for both the object and the agent, from the initial state to the goal state. This is typically achieved using optimization-based, sampling-based, or hybrid approaches.

A. Optimization-Based Methods

The traditional optimization-based methods optimize trajectories for smoothness and efficiency. While CHOMP [5]–[7] and variants rely on gradient-based optimization, STOMP [8] uses stochastic sampling to explore multiple trajectories. TrajOpt [9] introduces convex collision avoidance constraints, and KOMO [10] extends trajectory optimization by incorporating higher-order kinematic constraints and leveraging second-order Newton methods. Although optimization-based methods offer computational efficiency, they are limited to finding locally optimal solutions, reducing their effectiveness in high-dimensional, long-horizon tasks.

B. Sampling-Based Methods

In sampling-based manipulation planning, the typical approach involves the exploration of high-dimensional spaces by randomly sampling predefined primitives. CBiRRT [11] employs grasping primitives, RMR* [12] relies on constraint manifold-based motion primitives, and IMACS [13] utilizes implicit constraint-compliant primitives. While these primitives help guide the sampling process, their predefined nature limits generalizability. To address this limitation, CMGMP [14] introduces an approach that automatically generates motion primitives to enhance adaptability. However, despite these advancements, sampling-based methods generally require further trajectory optimization and remain computationally expensive in complex environments.

C. Hybrid Methods

Hybrid methods utilize both sampling and optimization-based methods to generate realizable and optimized motion plans in complex environments. One recent example is H-MaP [2], a hybrid sequential manipulation planner that decouples object trajectory planning from agent motion planning, leveraging informed contact sampling and waypoint generation to enhance robustness. Another recent work addresses the challenges of sequential manipulation planning in the presence of obstacles and narrow passages by decomposing complex tasks into a sequence of easier pick-and-place subproblems [1]. The approach integrates forward search with an optimization-based Task-and-Motion Planning (TAMP) solver [15], [16], allowing the agent to generate a feasible manipulation plan in spatially constrained environments more effectively. Other TAMP-related hybrid approaches focus on modeling modes with grasps [17]–[20]. Although recent advancements in hybrid motion planning have improved performance in complex and constrained environments, generating robust and computationally efficient solutions remains challenging, particularly in identifying critical subgoals and key objects.

D. Rearrangement Planning

Rearrangement planning is a crucial aspect of sequential manipulation, particularly in environments where clutter obstructs the completion of a task. [3] introduces factored state spaces to model rearrangement puzzles, allowing for efficient problem decomposition where [4] utilizes a learning-based approach to decompose long-horizon tasks into smaller, parallelized subproblems.

The rearrangement planning has conceptual similarities with the Navigation Among Movable Objects (NAMO) and Manipulation Among Movable Objects (MAMO) [21]. Recent advances have improved efficiency and adaptability through search-based planning [22], affordance-driven strategies [23], socially-aware object placement [24] and contact-aware motion planning via pushing and grasping [25].

Currently, few approaches integrate manipulation planning with obstacle relocation in constrained and complex environments. Our work proposes a generalizable framework that can generate robust and computationally efficient manipulation plans in the presence of obstacles.

III. Preliminaries

In our proposed framework, we use a hybrid motion planning approach to solve maze-like tasks in cluttered environments based on tasks from [1].

A. Optimization Based Motion Planning with KOMO

The SeGMan algorithm employs k-order motion optimization KOMO [10] for motion planning between subgoals generated by Bi-RRT [26].

$$\begin{aligned} \min_{x_{0:T}} \quad & \sum_{t=0}^{T} f_t(x_{t-k:t})^T f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall t : g_t(x_{t-k:t}) \leq 0, \quad h_t(x_{t-k:t}) = 0, \end{aligned} \tag{1}$$

where $x_t \in \mathcal{X}$, and $\mathcal{X}$ is the configuration space for the scene, including agents with n-DoFs and m movable objects. The set of x_{t-k} represents consecutive k+1 states from x_{t-k} to x_t. f_t, h_t, and g_t represent the cost function, equality, and inequality constraints, respectively, for timestep t.

In KOMO, logical constraints are translated into geometric constraints that define the motion feasibility of the agent and objects. In our work, we used logical constraints *touch*, *stable*, *positionDiff*, and *stableOn*, which correspond to the following geometric constraints in KOMO. The *touch* constraint ensures that the Euclidean distance between the agent and the object's grasp point is zero, effectively modeling contact as an equality constraint. The *stable* constraint ensures that an object remains static at specific timesteps by preventing changes in its position and orientation, which is also formulated as an equality constraint. The *positionDiff* constraint maintains a bounded relative position between two objects or between the agent and an object, represented as an inequality constraint on the Euclidean distance between them. Finally, the *stableOn* constraint ensures that an object remains stably placed on a surface by constraining its contact points and enforcing stability conditions, combining both equality and inequality constraints. These geometric formulations allow KOMO to optimize motion while satisfying stability, contact, and positioning requirements needed for executing not only a single but also a sequence of pick-and-place actions jointly, which is crucial for the problems we consider.

B. Problem Formulation

Our aim is to solve pick-and-place problems where the goal object, o_{goal}, should be placed in goal position, g, in a cluttered environment, often with narrow passages. We denote the initial configuration as x_0 and goal configuration as x_{goal}. The problem involves a combination of trajectory optimization and sampling-based planning to ensure that the object can be reached and placed at the goal location while respecting physical constraints.

Moreover, the environments might include movable objects that obstruct the path from the agent to the goal object o_{goal} and from the goal object to the goal position, g. In this case, the agent needs to perform pick-and-place actions on these obstructing objects as well. We denote movable objects (other than the goal object, o_{goal}) in the environment as $O = \{o_1, o_2, \ldots, o_{m-1}\}$ for $m-1$ movable objects and the minimal subset containing the obstructing objects as $O_{\text{min}} \subseteq O$ such that removing all the elements of O_{min} will result in a feasible plan P.

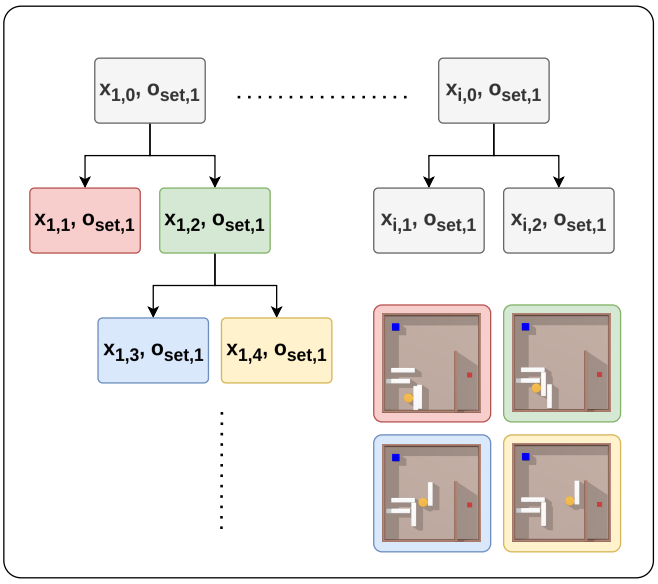

Fig. 2: The forest of o_{set} trees.

We divide the problem into several pick-and-place subproblems to reach the goal configuration. The final motion plan X is the concatenation of pick plans $X^{o_p}_{\text{pick}}(t)$ and place plans $X^{o_p}_{\text{place}}(t)$ for each subproblem p. This can be mathematically represented as:

$$X = \bigoplus_{p} \left(X^{o_p}_{\text{pick}}(t) \oplus X^{o_p}_{\text{place}}(t) \right)$$

where:

- X is the final motion plan consisting of a sequence of pick and place actions.
- $X^{o_p}_{\text{pick}}(t)$ is the pick motion plan for object o_p as a function of timestep t for subproblem p.
- $X^{o_p}_{\text{place}}(t)$ is the place motion plan for object o_p as a function of timestep t for subproblem p.
- $\oplus$ denotes concatenation, meaning that the pick and the place plans are executed sequentially.
- $\bigoplus_p$ represents the concatenation over all subproblems p.

This problem can simply be stated as $X = X^{o_{\text{goal}}}_{\text{pick}}(t) \oplus X^{o_{\text{goal}}}_{\text{place}}(t)$ when there are no obstacles present in the environment. However, most of our tasks include obstacles where brute-force trial and error exponentially grows the problem of pick and place. Therefore, we define the problem of obstacle subset, O_{set}, selection from the movable objects O in the scene as follows. We want to generate a O_{set} such that $O_{\text{min}} \in O_{\text{set}}$, $O_{\text{set}} \subseteq \mathcal{P}(O)$ where $\mathcal{P}(O)$ is the power set of O, and $|O_{\text{set}}|$ is minimal.

IV. Sequential and Guided Manipulation Planner

The framework given in Fig. 3 demonstrates the modules and how they interact with each other in the SeGMan algorithm. It consists of 2 main components: hybrid manipulation with adaptive subgoal selection and guided forward search.

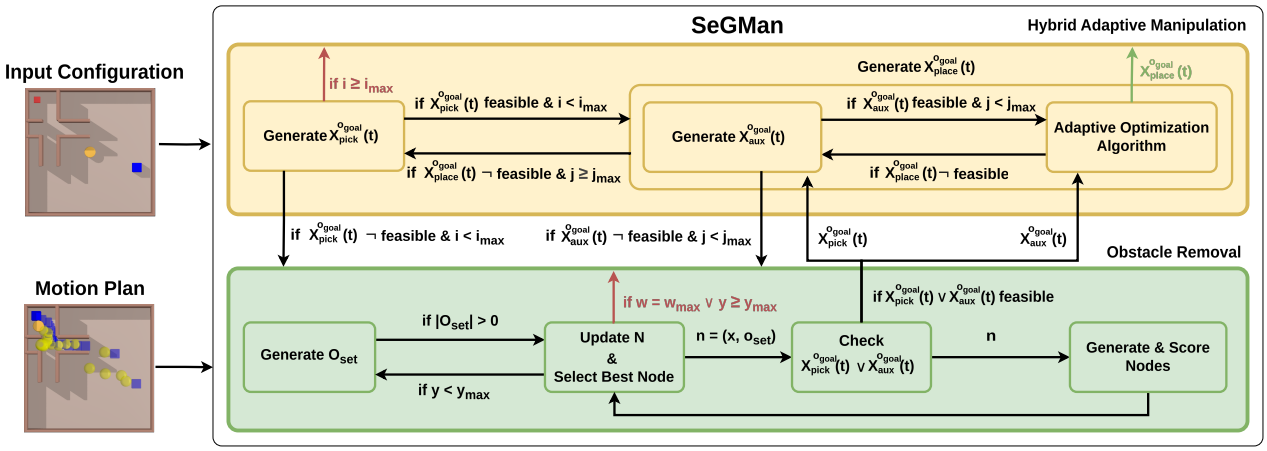

Fig. 3: System flowchart of SeGMan's two-phase architecture: (I) hybrid manipulation of goal object to goal position with adaptive subgoal selection, (II) forward search of determining obstacle objects and their removal. If no pick or place plan can be realized for the goal object, the obstacle removal phase clears the path. Once the paths are cleared, the framework continues to pick-place the goal object from where it left off.

A. Hybrid Manipulation with Adaptive Subgoal Selection

In this phase, a motion plan is generated for an object-goal tuple. The motion plan X consists of intermediate subgoals that are sequentially solved. The object manipulation phase consists of pick-and-place sequences between the agent and o_{goal} to generate an end-to-end motion plan.

1) Pick Plan: First, a valid pick configuration, $x^{o_{\text{goal}}}_{\text{pick}} = X^{o_{\text{goal}}}_{\text{pick}}(T)$, where T is the final timestep, is found using the non-linear constraint optimization explained in Eq. 1. Once a viable grasping configuration is found, a feasible trajectory $X^{o_{\text{goal}}}_{\text{pick}}(t)$ from x_0 to $x^{o_{\text{goal}}}_{\text{pick}}$ is generated with Bi-RRT. If a feasible trajectory cannot be found from the initially proposed grasping configuration, new grasp configurations are tried until either a valid $X^{o_{\text{goal}}}_{\text{pick}}(t)$ is found or the trial limit is exceeded. If no feasible $X^{o_{\text{goal}}}_{\text{pick}}(t)$ is found, x_0 is checked for obstacles. If movable objects exist in x_0, their relocation is attempted using the methodology described in sec. IV-B.

2) Manipulation Plan: Upon successful generation of $X^{o_{\text{goal}}}_{\text{pick}}(t)$, a waypoint sequence from o_{goal} to g is generated. Waypoint generation is performed in an auxiliary configuration $x^{o_{\text{goal}}}_{\text{aux},0}$ where the agent is removed and the o_{goal} is treated as an agent. A trajectory $X^{o_{\text{goal}}}_{\text{aux}}(t)$ from $X^{o_{\text{goal}}}_{\text{aux}}(0)$ to x_{goal} is computed using Bi-RRT. If a feasible $X^{o_{\text{goal}}}_{\text{aux}}(t)$ cannot be found, the guided forward search explained in sec. IV-B is utilized.

Once $X^{o_{\text{goal}}}_{\text{aux}}(t)$ is found, the intermediate configurations on the trajectory, $x^{o_{\text{goal}}}_{i}$, are used as a sequence of subgoals to generate $X^{o_{\text{goal}}}_{\text{place}}(t)$. Rather than utilizing every $x^{o_{\text{goal}}}_{i}$ as a subgoal, the Adaptive Subgoal Selection Algorithm (Alg. 1) iteratively selects closer subgoals within narrow passages, while maintaining efficiency in less constrained areas by selecting farther subgoals when feasible. If a feasible $X^{o_{\text{goal}}}_{\text{place}}(t)$ cannot be found within limited attempts, the trajectory is resampled with a finer resolution. If increasing the trajectory granularity still does not result in a feasible $X^{o_{\text{goal}}}_{\text{place}}(t)$, a different trajectory $X^{o_{\text{goal}}}_{\text{aux}}(t)$ is generated. This iterative process continues until either a $X^{o_{\text{goal}}}_{\text{place}}(t)$ is found or the iteration limit is reached, at which point the algorithm goes back to generating $X^{o_{\text{goal}}}_{\text{pick}}(t)$ where possible obstacles can be removed in the subsequent steps. The planner stops when a final motion plan is found or if a maximum trial limit is reached.

Algorithm 1: Adaptive Subgoal Selection Algorithm

Input: Goal object trajectory $X^{o_{\text{goal}}}_{\text{aux}}$, step incrementation threshold θ
Output: Motion plan $X^{o_{\text{goal}}}_{\text{place}}$
Initialize: $step_{\text{max}} \leftarrow |X^{o_{\text{goal}}}_{\text{aux}}| - 1$, $p_i, step_i \leftarrow step_{\text{max}}$, $p_{prev} \leftarrow 0$, $\beta \leftarrow 0$, $X^{o_{\text{goal}}}_{\text{place}} \leftarrow []$
while o_{goal} *not reached* g **do**
  $x^{o_{\text{goal}}}_{i} \leftarrow X^{o_{\text{goal}}}_{\text{aux}}[p_i]$
  $feasible \leftarrow False$
  $X^{o_{\text{goal}}}_{\text{place}}, feasible \leftarrow$ **KOMO**$(x^{o_{\text{goal}}}_{i})$
  if $feasible$ **then**
    $\beta \leftarrow \beta + 1$
    $p_{prev} \leftarrow p_i$
    if $\beta \geq \theta$ **then**
      $step_i \leftarrow \min\{step_i \times 2,\ step_{\text{max}}\}$
    end
    $p_i \leftarrow \min\{p_i + step_i,\ step_{\text{max}}\}$
  end
  else
    $\beta \leftarrow 0$
    if $step_i = 1$ **then**
      return $None$
    end
    $step_i \leftarrow \max\{1, step_i\ /\ 2\}$
    $p_i \leftarrow \min\{p_{prev} + step_i,\ step_{\text{max}}\}$
  end
end
return $X^{o_{goal}}_{place}$

B. Obstacle Removal

In this section, we first introduce our forward search strategy for obstacle removal. Then, we describe how the forward search is guided by the node selection policy and

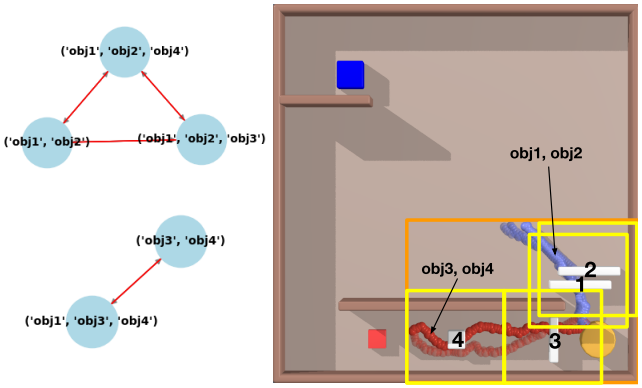

Fig. 4: Left: The clustered object sets (arrow direction is not important). Right: Two different clusters (red and blue) with each path are generated for $o_{set} \in C_i$. The more transparent the color of the path is, the less the o_{set} is prioritized .

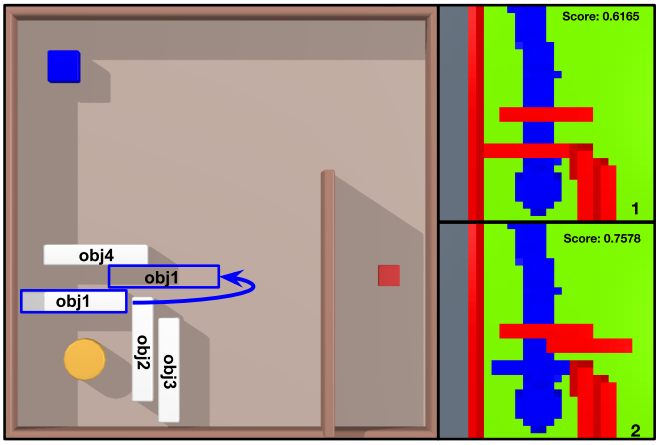

Fig. 5: Left: The Four Block scene, *obj1* is relocated (blue arrow). Right: Local occupancy grid (LOG) for *obj4*. The colors red and green, respectively, represent objects and free spaces. The color blue represents o_{set} trajectory, the initial positions of each object in o_{set}, and the initial position of the agent. When *obj1* moves, blue and green areas expand; thus, the configuration score increases.

local occupancy grids (LOG). Finally, we explain how we create new nodes to expand the search.

1) Forward Search: If a clear path from the agent to the goal object, o_{goal}, or from the goal object to the goal position cannot be found, our framework performs a guided forward search over possible configuration-obstacle set tuples, (x, o_{set}), which we call a *node* $n \in N$. Our method performs a search in a forest (see Fig. 2), where each tree is associated with a different obstacle subset, $o_{\text{set}} \subseteq O$. Nodes in a particular tree represent different configurations of the obstacles associated with this tree. We generate y number of candidate subsets using the subset selection policy (sec. IV-B.2) and create a forest of y number of trees. We score each new node using a scoring policy and select the node with the highest score in the forest to expand (sec. IV-B.4). When we are expanding a node, subgoal locations are generated for the tree's associated obstacle set (sec. IV-B.5), guiding the search through only a subset of objects. After the existing y trees reach a certain depth threshold, we add new trees for y more subsets until all object sets are explored to balance the depth and breadth of the search. The search is terminated when a node is found such that the configuration x contains a feasible path or the maximum search limit is reached.

2) Object Set Generation: In complex environments with numerous objects, it is important to guide the forward search toward a subset of objects rather than considering all movable objects. Preferably, object sets that include critical objects blocking the path should be selected.

For a subset $o_{\text{set}} \in \mathcal{P}(O)$ where $\mathcal{P}(O)$ is the power set of O, first a preliminary elimination is done by checking whether a clear path exists when the objects in o_{set} are completely removed from the configuration. If a solution does not exist, we eliminate this subset as a possible feasible subset.

After the preliminary elimination, similar sets are clustered based on the similarity of their generated paths, using Dynamic Time Warping (DTW) [27]. The sets containing redundant objects follow similar paths to the set with no redundant objects since redundant objects contribute little to path variation. However, they still might be clustered into different groups due to randomization in Bi-RRT. To minimize variations introduced by Bi-RRT, we check the similarity of only a small path segment. The path segment in the smallest region containing each $o \in O$ (Fig. 4) is used for similarity. Finally, the clustered sets, $C = \{C_1, C_2, ..., C_i\}$, are scored using S_{o} to prioritize sets that include fewer redundant objects.

Within the same cluster, the smallest subset contains the most critical objects, as other sets in the cluster are likely supersets that include redundant objects. Thus, selecting only the smallest subset in each cluster might seem preferable to scoring all sets and prioritizing the smallest ones. However, in certain scenes (Lock problem in Fig. 6), removing some redundant objects might be necessary to successfully relocate the objects in the smallest subset. For this reason, we adopt a soft threshold with subset priority scoring rather than strictly selecting the smallest subset, as this approach enhances the generalizability of the proposed framework.

3) Local Occupancy Grid (LOG): LOG is proposed to guide the search toward generating configurations that are more likely to yield feasible $X_{\text{pick}}^{o_{\text{goal}}}(t)$ or $X_{\text{aux}}^{o_{\text{goal}}}(t)$. LOG is used for configuration scoring and subgoal generation.

For configuration scoring, LOG is generated for each $o \in o_{\text{set}}$, localized around the initial position of o. Fig. 5 shows an example LOG, where red and green areas represent free space and objects, respectively. Blue areas correspond to the path taken by the agent if $o \in o_{\text{set}}$ were removed, the initial position of each o, and the agent's initial position. The configuration score S_{x} is obtained by calculating the green and blue areas where each color has a score per unit area defined with blue areas weighted twice as much. The S_{x} prioritizes configurations that free the object's initial position without further obstructing the agent or its trajectory.

The subgoals for node generation (sec. IV-B.5) are ob-

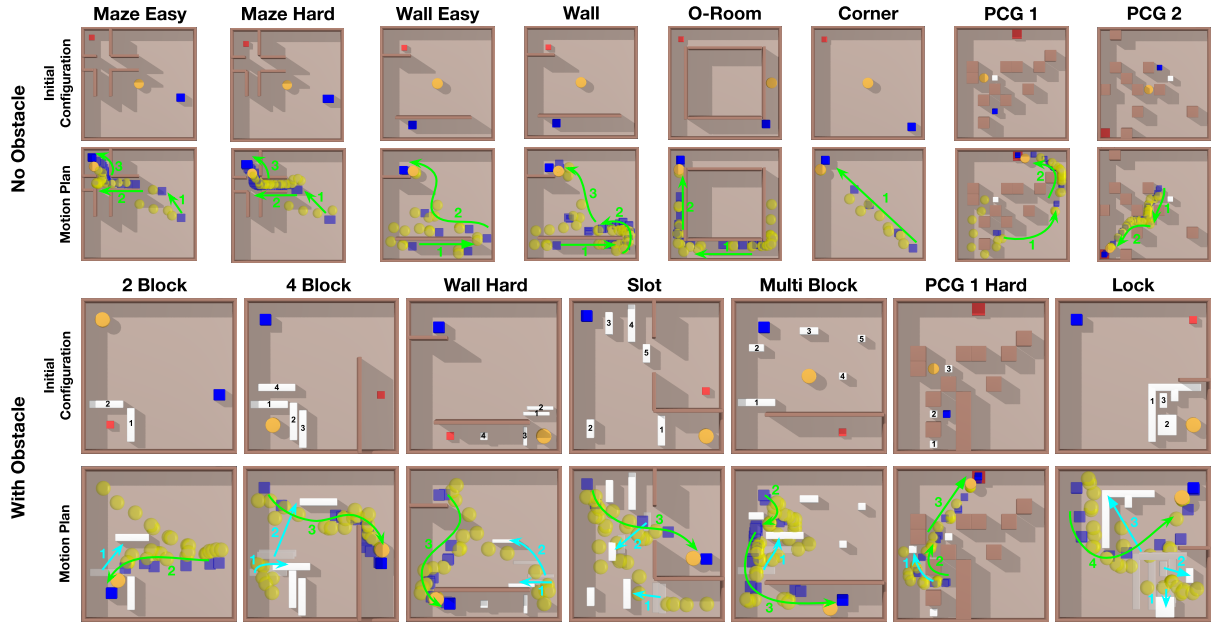

Fig. 6: Illustration of 15 tasks used to evaluate the proposed framework and baselines. The upper rows show the initial configuration of the scene, while the lower rows show the final motion plan trajectories. Green arrows indicate o_{goal} movements, whereas turquoise arrows represent the relocation of objects $o \in O$.

tained from the LOG, which is localized around the object's current position in x_i. The green areas in the LOG are proposed as candidate subgoal locations.

4) Node Selection: We use the following node selection policy, which is:

$$\arg\max_{n \in N} S(n) = \alpha \cdot \sqrt{\frac{1}{V_n}} + \gamma^{V_n} \cdot \frac{|C_i|}{\arg\max_{C_k \in C} |C_k|} \cdot \left(\frac{1}{|o_{\text{set}}|} \cdot S_{\text{r}}(n) \cdot S_{\text{x}}(n) \right)^2 \quad (2)$$

where $o_{\text{set}} \in C_i$, $n = (x, o_{\text{set}})$ and $S(n)$ is the node score. Since our goal is to find the minimal feasible subset for object removal, we first expand the nodes that have smaller o_{set} size, $|o_{\text{set}}|$. Therefore, the score is inversely proportional to the size of the subset. Moreover, the minimal subset is more likely to be in a larger cluster of subsets, C_i since the supersets of the minimal subset will also yield feasible configurations; thus, such clusters receive higher scores.

S_{r} is the reachability score. For a node $n = (x, o_{\text{set}})$, $S_{\text{r}}(n) = r + 1$, where r is the number of reachable objects $o \in o_{\text{set}}$ in configuration x. S_{x} is the configuration score obtained using the local occupancy grid as explained in sec. IV-B.3. The remaining terms are used to ensure the exploration of nodes; α is the exploration factor, V_n is the visit count of the current node n and γ is the discount factor.

5) Node Generation: Node with the highest score, $n = (x, o_{\text{set}})$, is expanded by repositioning the reachable objects in o_{set} to the proposed subgoal positions. The subgoals for each object are obtained from the LOG, and a randomly sampled subset of these subgoals is tested. The pick-and-place sequence is optimized using Eq. 1 for each subgoal.

The *positionDiff* logical constraint is given to the optimizer as a soft constraint in this phase, allowing it to position the object in a feasible location near the specified subgoal rather than at the exact position. This increases the computational efficiency in cases where the exact subgoal location is not feasible.

If the motion plan is feasible, the reachability score S_{r} is calculated. Finally, each new node is scored using $S(n)$, sorted, and the best subset of generated nodes is added to the complete node set N.

V. Experiments

We evaluated SeGMan on 15 pick-and-place tasks (Fig. 6), which are representative of complex and constrained environments. The tasks are adapted and extended from [1]. The agent's (yellow cylinder) task is to place the goal object (blue square) in the goal position (red square), interacting with movable objects (white squares) when required. The tasks are divided into two categories:

- **No-obstacle tasks** The solution does not require interaction with movable objects.
- **With Obstacle tasks** The solution requires relocating key movable objects to find a feasible motion plan.

The tasks require multiple pick-and-place sequences to manipulate objects through narrow passages while avoiding obstacles. Choosing the right subgoals for the right objects is important as many objects in the tasks are either redundant or interdependent. Thus, the tasks effectively represent the key challenges of sequential manipulation in constrained and complex environments. We compare SeGMan with two versions of [1] and H-MaP [2]. [1]

[1]As the source code for [1] was not available, we re-implemented two variations of their algorithm. For [2], we re-implemented the code in Python.

TABLE I: The computation time, solution success rate, and solution quality (pick and place count) results for the SeGMan, [1] Random, [1] Bottleneck, and H-MaP [2] after running each task 10 times. "-" indicates failure to solve the task after 1000 seconds.

Task Name	SeGMan				[1] Random				[1] Bottleneck				H-MaP [2]			
	Mean (s)	Std Dev (s)	Solved (%)	PnP Count	Mean (s)	Std Dev (s)	Solved (%)	PnP Count	Mean (s)	Std Dev (s)	Solved (%)	PnP Count	Mean (s)	Std Dev (s)	Solved (%)	PnP Count
Maze Easy	2.66	0.41	100	15.9	18.59	24.07	100.0	3.2	190.47	276.03	100.0	3.8	0.30	0.03	100.0	53.0
Maze Hard	3.96	1.94	100.0	18.2	–	–	–	–	–	–	–	–	0.28	0.02	100.0	47.1
Wall Easy	1.63	0.74	100.0	9.3	21.18	7.41	100.0	3.0	71.44	10.61	100.0	3.0	0.39	0.04	30.0	75.33
Wall	4.00	3.53	100.0	11.6	395.57	194.30	100.0	5.3	360.78	218.87	90.0	5.56	0.51	0.43	90.0	58.78
O-Room	4.04	2.50	100.0	16.0	146.35	105.85	100.0	3.8	473.23	237.72	90.0	6.33	0.43	0.07	70.0	76.43
Corner	0.62	0.08	100.0	3.7	0.19	0.32	100.0	2.0	0.31	0.09	100.0	2.0	0.20	0.02	100.0	51.1
PCG 1	5.49	4.95	100.0	11.9	57.39	36.22	100.0	3.1	125.05	33.69	90.0	3.67	0.45	0.07	100.0	63.6
PCG 2	13.33	9.99	100.0	19.3	10.61	2.22	100.0	3.0	38.82	10.37	100.0	3.1	0.28	0.01	70.0	41.57
Two Blocks	8.99	0.55	100.0	4.7	1.77	0.91	100.0	2.3	6.11	1.05	100.0	4.0	–	–	–	–
Four Blocks	46.73	22.23	100.0	10.1	877.18	0.0	10.0	5.0	684.81	0.0	10.0	8.0	–	–	–	–
Wall Hard	37.42	2.53	100.0	8.9	–	–	–	–	–	–	–	–	–	–	–	–
Slot	26.10	3.38	90.0	7.77	577.52	77.20	60.0	5.33	594.00	221.52	90.0	4.89	–	–	–	–
Multi Block	16.79	2.39	90.0	10.66	232.58	322.49	90.0	3.89	305.33	219.44	30.0	5.0	–	–	–	–
PCG 1 Hard	46.49	4.95	90.0	4.7	373.48	0.0	10.0	4.0	616.41	292.22	60.0	3.5	–	–	–	–
Lock	378.93	326.51	90.0	10.4	719.86	123.05	40.0	4.75	900.42	178.56.0	30.0	5.0	–	–	–	–

The primary evaluation metrics are solution success rate and computation time. Additionally, the solution quality is assessed based on the number of pick-and-place sequences, where a higher count indicates lower quality. Each experiment is repeated 10 times to ensure statistical significance. If the computation time exceeds 1000 seconds, the run is aborted and recorded as a failure.

VI. Results

The success rate, computation time, and PnP (Pick-and-Place) count are given in Table I. SeGMan successfully generated a feasible motion plan for all tasks in under 1000 seconds, while baseline methods struggled to solve some tasks within the given time limit.

A. No Obstacle

SeGMan successfully generated feasible motion plans for all "No Obstacle" cases while demonstrating computational efficiency compared to the baselines. Moreover, its sampling-based approach results in consistent subgoal generation across trials, as reflected in the low standard deviation in computation time.

We observe that the PnP count is small for tasks with clear spaces, such as the Corner task, while the count increases as the task includes narrow passages, like the Maze task. This is due to the adaptive subgoal selection method; in narrow passages, subgoals are chosen in finer granularity to ease the manipulation.

Both variations of [1] performed as expected, as most of these tasks were originally from their work. They generated the highest quality solutions for all tasks within 1000 seconds, except for the Maze tasks. Among all methods, H-MaP [2] had the shortest computation time but struggled with robustness and solution quality. While SeGMan and [2] share a similar sampling-based approach, SeGMan's adaptive subgoal selection method led to higher-quality solutions. This is because [2] does not select a subset of the generated subgoals and utilizes all of them.

B. With Obstacle

To solve the "With Obstacle" tasks, SeGMan employs the guided forward search for obstacle relocation. Despite the increased complexity of the puzzles, SeGMan successfully found solutions in nearly all trials. Compared to the baselines, the proposed framework generates significantly faster solutions. SeGMan consistently produced robust, computationally efficient solutions while maintaining relatively high solution quality.

Compared to "No obstacle" tasks, the solutions of these tasks produce lower PnP counts. There are two reasons for this behavior. Firstly, the environments in this category include more movable objects in the configuration but have fewer narrow passages. Secondly, in the case of PCG 1 Hard, the solution quality is better than PCG 1, because Bi-RRT generates a path passing through clear spaces in PCG 1 Hard, whereas in PCG 1, the path goes through narrower spaces.

The baseline results provide insight into the difficulty of generating a solution for these tasks and highlight the performance of SeGMan. H-MaP's object relocation strategy could not find feasible subgoals for the obstacles; thus, no solution was found for the tasks.

Similarly, [1] struggles to generate a solution to these tasks, failing to find solutions for 1 out of 7 cases. The primary reason [1] struggles to generate solutions is that when there are many obstacles in the scene, subgoals are generated for every reachable obstacle, increasing the search space considerably while SeGMan selectively moves a subset of movable objects in the configuration.

The subgoal scoring heuristic in [1] does not differentiate configurations with varying obstacle placements, rendering it less effective, especially when obstacles need to be relocated to critical positions, as seen in Wall Hard and Slot tasks. While both SeGMan and [1] implement forward search, SeGMan's LOG and clustering-based heuristics effectively reduced the search space and prioritized subgoals that cleared obstacles, leading to shorter solution generation times.

C. Limitations & Discussion

There are several limitations of our work. Firstly, the proposed heuristics guide the forward search toward feasible configurations; however, there is a computational overhead in simpler cases. In scenarios like Corner and Two Blocks, [1] outperforms SeGMan in most metrics, benefiting from its underlying TAMP framework. Moreover, while the sampling-based subgoal generation increases robustness, the solution quality decreases due to the size of the subgoal set. Although the subgoals are selected adaptively, the generated end-to-end motion plan is not necessarily optimal in solution quality.

The results reveal a trade-off between computation time and solution quality, with [1] performing better in solution quality, while H-MaP is computationally faster. SeGMan offers a balanced approach, producing robust motion plans while offering efficient computation performance and relatively high solution quality.

VII. Conclusion

In this work, we introduced SeGMan, a hybrid motion planner that enhances sequential manipulation in constrained, cluttered environments. By combining a hybrid motion planning method and heuristic-guided forward search, SeGMan efficiently identifies critical subgoals and generates robust motion plans.

Our approach addresses key challenges such as obstacle selection, subgoal feasibility, and fine object manipulation, where existing methods often struggle. Adaptive subgoal selection and iterative obstacle relocation with guided forward search enable generalization across environments. Empirical results show that SeGMan outperforms baselines, achieving a strong balance between computation time and solution quality.

SeGMan provides a framework that can be implemented in other domains, such as mobile robots and prehensile manipulation. Future work will focus on extending SeGMan to higher-dimensional domains and improving adaptability to real-world uncertainties, contributing to more intelligent and autonomous robotic planning.

References

[1] S. Levit, J. O. de Haro, and M. Toussaint, "Solving sequential manipulation puzzles by finding easier subproblems," in *ICRA*, pp. 14924–14930, 2024.

[2] B. Cicek, A. S. Yenicesu, C. B. Tuncer, K. Demiray, and O. S. Oguz, "H-map: An iterative and hybrid sequential manipulation planner," *arXiv preprint*, vol. arXiv:2403.10436, 2024.

[3] 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.

[4] Y. Zhang, A. Razmjoo, and S. Calinon, "Learn2decompose: Learning problem decomposition for efficient task and motion planning," 2024.

[5] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *IJRR*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[6] A. D. Dragan, N. D. Ratliff, and S. S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *IEEE ICRA*, pp. 4582–4588, 2011.

[7] A. D. Dragan, G. J. Gordon, and S. S. Srinivasa, "Learning from experience in manipulation planning: Setting the right goals," in *Robotics Research: The 15th International Symposium ISRR*, pp. 309–326, Springer, 2017.

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE ICRA*, pp. 4569–4574, 2011.

[9] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *IJRR*, vol. 33, no. 9, pp. 1251–1270, 2014.

[10] M. Toussaint, "Newton methods for k-order markov constrained motion problems," *ArXiv*, vol. abs/1407.0414, 2014.

[11] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE ICRA*, pp. 625–632, 2009.

[12] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *IEEE ICRA*, pp. 3426–3432, 2017.

[13] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *IJRR*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[14] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d," in *IEEE ICRA*, pp. 2730–2736, 2022.

[15] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning.," in *IJCAI*, pp. 1930–1936, 2015.

[16] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *RSS*, 2018.

[17] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[18] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3426–3432, IEEE, 2017.

[19] Z. Kingston, A. M. Wells, M. Moll, and L. E. Kavraki, "Informing multi-modal planning with synergistic discrete leads," in *IEEE International Conference on Robotics and Automation*, pp. 3199–3205, 2020.

[20] W. Vega-Brown and N. Roy, "Asymptotically optimal planning under piecewise-analytic constraints," in *Algorithmic Foundations of Robotics XII*, pp. 528–543, Springer, 2020.

[21] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3327–3332, IEEE, 2007.

[22] Z. Ren, B. Suvonov, G. Chen, B. He, Y. Liao, C. Fermuller, and J. Zhang, "Search-based path planning among movable obstacles," *arXiv preprint arXiv:2410.18333*, 2024.

[23] M. Wang, R. Luo, A. Ö. Önol, and T. Padir, "Affordance-based mobile robot navigation among movable obstacles," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2734–2740, IEEE, 2020.

[24] B. Renault, J. Saraydaryan, and O. Simonin, "Modeling a social placement cost to extend navigation among movable obstacles (namo) algorithms," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11345–11351, IEEE, 2020.

[25] H. Wang, Q. Wang, F. Gao, and S. Shen, "Contact-aware motion planning among movable objects," *arXiv preprint arXiv:2502.03317*, 2025.

[26] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, 2000.

[27] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.