Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

A Learning-Based Approach for the Programming of Interior Layout Implementation

Hanqing Wang hanqingwang@bit.edu.cn
Wei Liang liangwei@bit.edu.cn
Lap-Fai Yu craigyu@gmu.edu

GMU-CS-TR-2019-1

Abstract

Given a scene layout like a room or a courtyard composed of objects, it is usually implemented manually due to the complicated state which results in a large searching space for the machine. In this paper, we propose a learning-based approach to program the implementation automatically. Our approach has two components. The main structure of our approach is the Monte Carlo Tree which searches the most valuable move for the current state. A neural network estimates the value for the leaf nodes of the searching tree. With the power of deep reinforcement learning, the network learns how to move the objects through millions of trial and error. We demonstrate our approach on different scenes and compare the performance of our approach with human performance in our experiments.

1 Introduction

After Christmas or New year, on spring break, or before summer, many people want to rearrange their rooms for a change. To make the rearrangement, people usually take two steps: design a new layout, and move all furniture into their new positions. Most recently, many attentions have been given to automatically synthesize an interior furniture configuration which is compatible to the room. Those works [40, 11, 37] achieved good results comparing to human designs and can be used to deal with the task in the first step. But how to move furniture to the desired positions is ignored. The moving strategy is also very important since it is an essential process to realize the design ideas.

Many people have such experiences. They have a new designed layout of a room. But when they are going to move the furniture into the designed positions in practice, they find they have to accomplish the task through trial-and-error with a lot of useless efforts. For example, he may move a piece of furniture to the desired position. Then when he tries to move another one, he finds the former moved furniture blocks the way. So he has to move the former one again so that it can make way for others.

During the moving process, human makes a sequential decisions. The furniture must be moved to the target position piece by piece in certain orders in case that some placed furniture block other's way to get the target positions. This sequential decision-making problem is a NP problem. Even finding an approximation solution could be hard. There are mainly two difficulties to solve this problem: (1) *Dynamic changes.* As the furniture is being moved, the scene state changes all the time. It brings more variations to the scene and influence the further decision. (2) *Long-term decision making.* Each moving in the whole decision sequence is not isolated. Moving an object to a target position usually is not a one-step process. For example, to move a dining table to a new position, we may need to move all chairs first. Generating such a sequence of actions is pretty hard since the searching complexity is exponential.

To address these problems, we propose an automatic move planning approach to yield a moving suggestion, i.e. a sequence of objects being moved and the corresponding moving path of each object. To deal with the dynamic changes, we use both history and current scene states as the input of the inference. Then we leverage the power of deep reinforcement learning to learn the mapping from the scene space to the probability distribution of moving actions by training a policy network. In order to make long-term decision, we introduce Monte Carlo Tree Search (MCTS) to embed the causality between the current action and the future states. We also define a sparse action space where the actions are composed of some primary actions to shorten the length of the action sequence. It helps our approach make long-term decisions.

There are mainly two stages in our approach. In the first stage, we discretize the current scene and the target scene into grids where each grid is annotated as empty

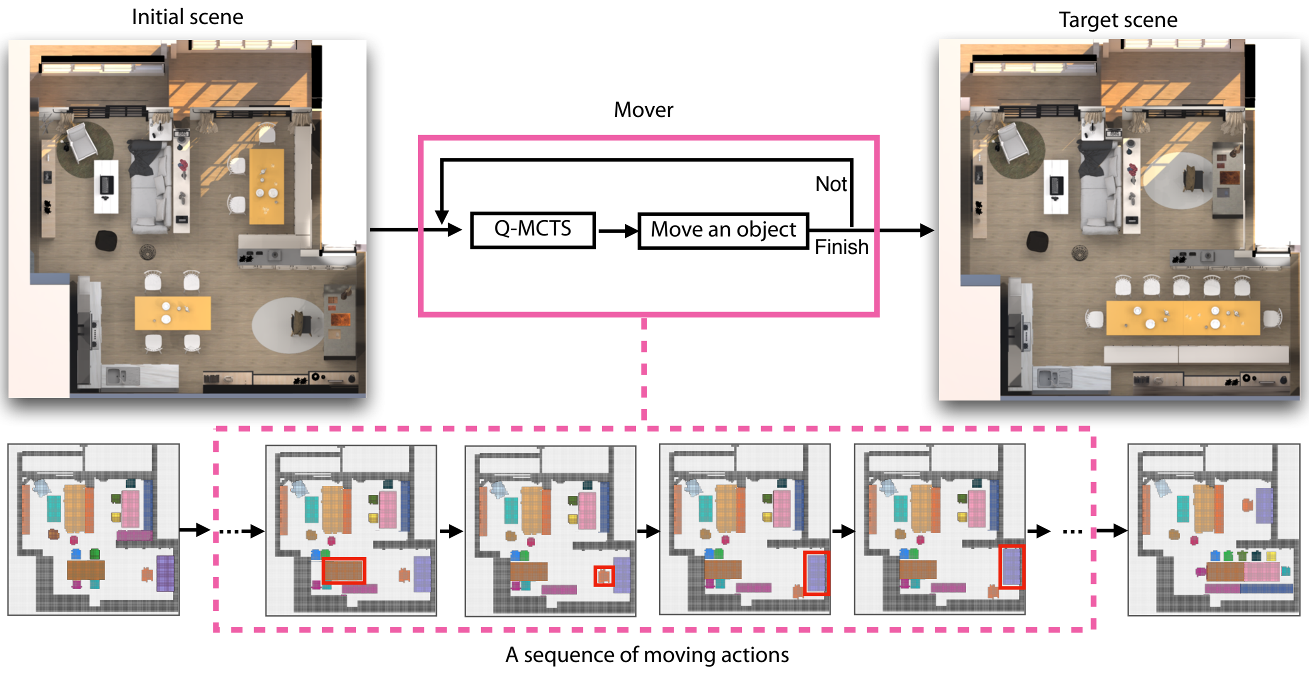

Figure 1: The pipeline of our algorithm. Given the initial layout and the target layout, our approach is able to iteratively generate a sequence of moving actions in which the initial layout is transformed into the target layout.

or occupied. In the second stage, we infer the action sequence. Specifically, for each step, we feed the discretized current scene into the network embedded Monte Carlo Tree and select the action that gains most rewards. This procedure is repeated until all objects reach the target positions. An example is illustrated in Figure 2.

Our contributions are summarized as follows:

- We propose a new problem of move planning to turn an initial scene into a target scene automatically, which is the implementation of scene synthesis.
- We introduce a reinforcement learning based Monte Carlo Tree search approach to solve the problem of move planning.
- We perform human perception studies to evaluate the efficiency of our approach and perform ablation studies to analyze our framework. We also present applications in different scenes to demonstrate the generalization.

2 Related Work

2.1 Scene Synthesis

Recent years, there are numerous efforts in synthesizing indoor scene layout, furniture arrangement or building layout. A lot of automatic scene synthesis approaches [40, 11, 30] are proposed to generate reasonable scenes automatically, considering aesthetics, functionality, and general human activities.

Merrell et al. [26] created plausible arrangements for a set of specified objects according to interior design principles. Yeh et al. [39] specified priors over which objects may occur and their spatial relationships to arrange objects in scenes. Another stream focused on data-driven scene synthesis. Yu et al. [40] extracted hierarchical and spatial relationships for various furniture objects, encoding them into priors associated with ergonomic factors, to yield realistic furniture arrangements. Fisher et al. [9] modelled the pairwise spatial relationships between objects by a directed graphical and a Gaussian mixture model. Qi et al. [30] learned the layout distributions from an indoor scene dataset and sample to generate new layouts using Monte Carlo Markov Chain. Human contexts as contextual relations were encoded. Wang [37] used deep convolutional networks to learn priors over which objects should be in a scene and how they should be arranged. Merrell et al. [25] proposed a datadriven method to generate residential building layouts. A Bayesian network are trained based on 120 examples of architectural programsthat to model the relationships among different rooms.

Our approach mainly focuses on how to implement the design ideas, which may cooperate with those scene synthesis approaches to fill the gap between the design and the implementation. For example, a user wants to rearrange his room. An synthesis approach can be applied to generate a layout for him. Then the user may use our approach to suggest a sequence of moving actions, consisting of the a sequence of objects being moved and the corresponding moving path of each piece of furniture.

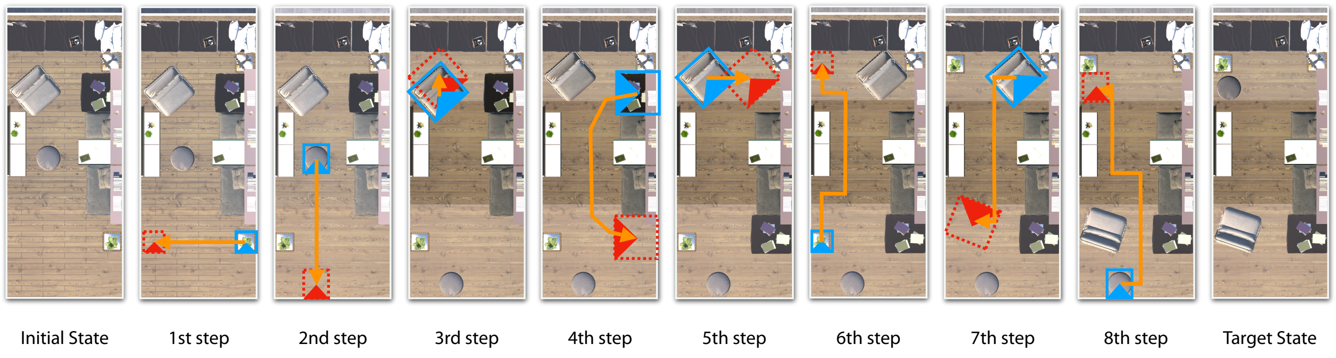

Figure 2: An example of our approach. The bounding boxes represent the position and the orientation of objects. The blue bounding box represents the object to move in this step. The red boundling box with dashed line represents the target position of this object. The orange line represents the moving path.

2.2 Path Planning

Tomas [24] defined one geometric problem, which involved finding where to place objects (Findspace) and how to move objects from one place to another without collision (Findpath). The latter is also regarded as path planning problem. It is researched broadly to apply on robot navigation, aerial vehicles, and digital animation. Please refer to a recent survey [13] for a comprehensive review.

Roughly, the path planning approaches were divided into three categories: roadmap, cell decomposition, and artificial potential. The roadmap methods such as visibility graphs [20, 2], Voronoi diagrams [36, 12], Delaunay triangulation [18], mapped the free space connectivity into a system of one dimension curves. Cell decomposition methods [6] divided the free space into small cells, and used connectivity graph to represent the adjacency relations between cells. Then searching graph algorithms were applied to find the solution. The artificial potential approaches considered the moving object as a point, which subjected to a potential field generated goal configuration and the obstacles [31, 14].

To search in a graph constructed by the roadmap and the cell decomposition approaches, search algorithms such as Dijkstra [8] and A* [15] were used to find an optimal solution in a connectivity graph.

Different from the merely path planning problem, our approach solves the problem of moving strategy, which considers not only searching for a plausible path for an object but also inferring a global strategy of the objects' moving order to ensure each object to reach the target location.

2.3 Deep Reinforcement Learning

Reinforcement learning is an experience-driven learning framework, in which an agent learns through trial-and-error interactions in a dynamic environment. With the great achievements of deep learning, reinforcement learning also benefited from the deep learning algorithms and defined deep reinforcement learning (DRL) techniques. DRL helped to break the bottleneck of solving high-dimensional problems.

Levine et al. [21, 22] proposed to learn the control policies for robots directly from camera inputs in the real world. Mnih et al. [28] introduced Deep Q-Network (DQN), which stabilized the training of Q action value function approximation with deep neural networks. The algorithm perform well on 49 Atari games. Schaul et al. [3] cast the design of an optimization algorithm as a learning problem, allowing the algorithm to learn more efficiently. ODonoghue et al. [29] proposed to combine policy gradient with off-policy Q-learning (PGQ), to benefit from experience replay.

In our approach, we leverage the techniques of deep reinforcement learning to train a Q-Network to estimate the long-term reward for each moving action being taken.

2.4 Sequential Decision Making

Sequential decision making refers to a procedural approach to decision-making, or as a step by step decision theory, where the earlier decisions influences the later available choices [10]. It is the act of answering the question "What should I do now?" [23]. Here, "now" is one of a finite set of states; "do" is a finite set of actions; "should" is maximize a lomng-run measure of reward; and "I" is an automated planning or learning system(agent). Sequential decision problems usually incorporate utilities, uncertainty, and sensing, include search and planning problems as special cases, which are not one-shot or episodic decision problems [32].

Sequential decision problems commonly modeled as Markov decision processes (MDPs) [5], occur in a range of real-world tasks such as robot control [19], game playing [38, 33], military planning [1] and so on.

Using sequential decision methods to play games is one of important applications of sequential decision making and have attracted considerable attention, e.g. computer chess [7], Shaogi [17], Go [34, 33], and Atari 2600 [16]. Anthony et al. [4] presented Expert Iteration (EXIT), a novel reinforcement learning algorithm which decomposes the problem into separate planning and generalisation tasks. The proposed EXIT showed good performance on training a neural network to play the board game Hex. Silver et al. [34] applied AlphaZero to the games of chess and shogi as well as Go, without any additional domain knowledge except the rules of the game, demonstrating that a general-purpose reinforcement learning algorithm can achieve, tabula rasa, superhuman performance across many challenging domains.

In our work, we model the problem of moving a set of objects to the target locations as a sequential decision making problem,and formulate it as a Markov decision process. Our approach is different from the previous works in terms of problem definition and learning strategy.

3 Overview

Our goal is to find an optimal moving plan, following which all objects in the initial scene can be moved into the positions assigned by the target scene. In the moving plan, all actions are coherent, that is, the action of moving an object will affect the action of moving another one. For example, after being moved to the right position assigned by the target scene, an object may get in the way of another one, which leads to the failure of the task or extra efforts to move the placed objects.

To move objects efficiently and to accomplish the moving task at the end, searching strategy is crucial. We adopt Monte Carlo Tree Search (MCTS) to search for an optimal decision, which searches to maximum depth without branching at all by sampling long sequences of actions from a policy. The search tree is built in an incremental and asymmetric manner. For each step, a tree policy is used to find the most urgent node of the current tree. A simulation is then run from the selected node and the search tree is updated according to the simulation. As the tree grows iteratively and more simulations are executed, the relevant values become more accurate. The tree policy of finding the urgent node determines how MCTS explores the search tree. We leverage the learning ability of reinforcement learning to train a policy network and reduce the effective depth and breadth of the search tree.

Fig. 3 depicts the overview of our approach. The left demonstrates the process of MCTS. Given a current and a target scene, the search tree grows a new node in four steps: node selection, expansion, simulation, and back propagation. In the selection phase, the most valuable node is selected. In the expansion phase, an unexplored action of the selected node is choosed as the edge connectting the node and a new child node which represents the next state after the execution of the action. In the simulation phase, a quick simulation under some policy is performed to estimate the value of the new child node. In the back propagation phase, a chain of nodes from the new node to the root node is updated bottom-up. The right of Fig. 3 illustrates the procedure of node selection, expansion and the simulation under the guide of the Q-Net.

4 Problem Formulation

Formally, we denote the initial scene and the target scene as x_i and x_f, respectively, where $x_i, x_f \in \mathcal{X}$ and $\mathcal{X}$ is the state space of the scene. We consider this task in which an agent interacts with an environment $\mathcal{E}$ through a finite sequence of actions $a = (a_1, a_2, ..., a_n)$, $a_t \in \mathcal{A}$ where $\mathcal{A}$ is the action space, observations $x = (x_1, x_2, ..., x_n, x_{n+1})$, where $x_1 = x_i$ and rewards $r = (r_1, r_2, ..., r_n), r_t \in \mathbb{R}$. In this case, the moving sequence is a series of actions selected from the action space. We define 5 actions for each object, the first four actions are moving straightly untill it reaches obstacles in up, down, left, right directions, noted as straight move, the 5th action is moving the object to its new arrangement if there is an available route, noted as route move. It's worth noting that the rotation operation is embeded in the route searching to reduce the burden of the agent. This action space constrains the space of scene variations and short the length of action sequence. At moment t, the agent observes the state s_t and select an action a_t. The action is passed to the environment, an emulator $e : (\mathcal{X}, \mathcal{A}) \to \mathcal{X}$ and a reward function $re : (\mathcal{X}, \mathcal{A}) \to \mathbb{R}$ are executed to get the next state x_{t+1} and the reward r_t. The rewards received after executing an action are defined as following:

Basic penalty: Each step receives -10 score as the penalty to limit the length of sequence.

Fail penalty: If an illegal action is selected, -600 score received. The illegal actions include that select an action of some object which does not exist in the scene or the object can not be moved as the action instructs.

Duplicate penalty: If the state appeared before, -60 score received.

Arrival reward: When an object arrives its destination for the first time, 500 score received. If it is not the first time the object arrives its destination, 50 score received.

Leave penalty: If it is the first time an object leave its destination, -600 score received. If it is not the first time, -60 score received.

Finish reward: If all objects are moved to their destination, 10000 score received, the task terminates.

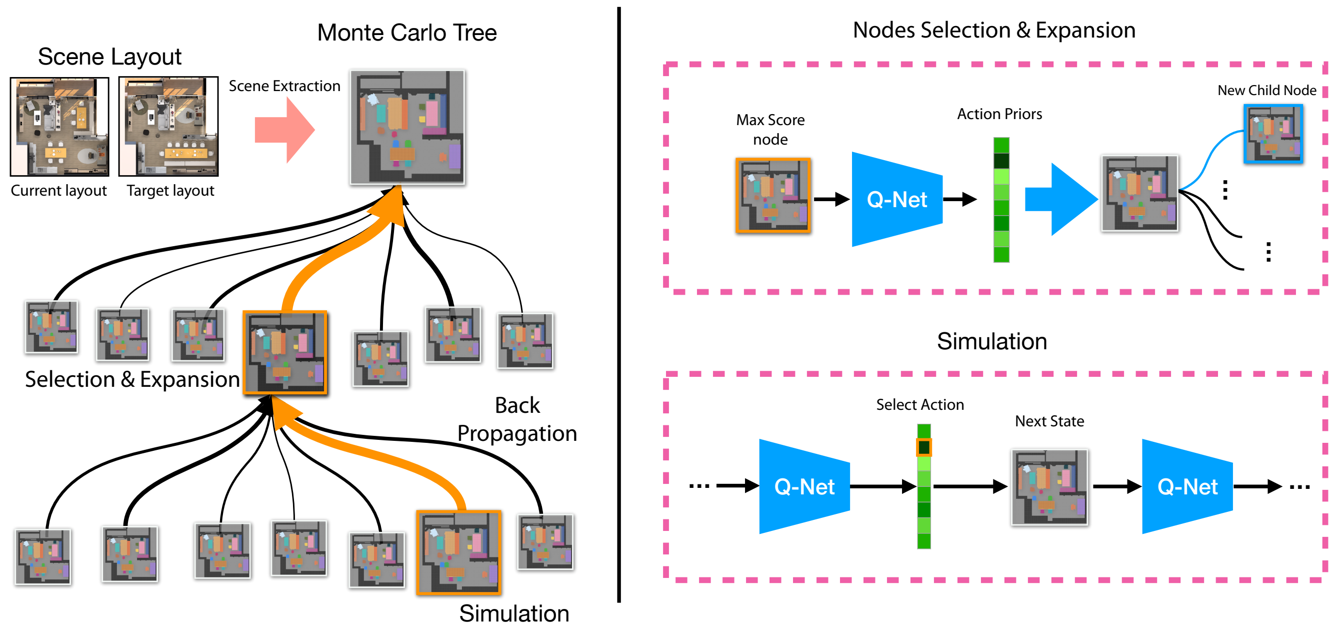

Figure 3: The overview of our framework. Our approach is a Monte Carlo Tree with deep neural network embeded. The thickness of the branch represents the value of the child node. In the inference phase, the search tree grows in four steps: a) Node Selection, we choose the most valuable tree node to expand; b) Expansion, the Q-Net estimates the action priors for the state of the selected node and sample an action. The selected node grows a child node after make the action. c) Simulation, we do a quick simulation using the Q-Net until the task is finished or it reaches the maximum number of steps. Save the gained rewards to the new expanded leaf node. d) Back Propagation. The child node back propagates the accumulated rewards to its ancient nodes recursively. Then we can find the best action from the children of the root node.

Thus, the goal of the agent is to maximize the accumulated rewards during the procedure. Formally, it can be defined as

$$\begin{aligned} &\max_{a} \sum_{t=1}^{\|a\|} re(x_t, a_t), \\ &s.t. \quad x_{\|a\|+1} = x_f. \end{aligned} \tag{1}$$

.

5 Scene Infromation Extraction

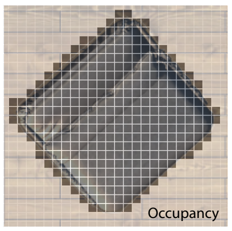

Figure 5: The discretization of a object in the scene.

As illustrated in Figure 3, when the initial state and the target state of the 3D scene is given, the first step is to extract the information from the scenes. Though the scene is 3D, in our problem, the states of the scene can be mostly characterized by a 2D x-z plane layout. Many prior works about scene synthesis phrase this problem in terms of determing 2D positions and 1D orientations of objects from a top-down view. Similarly, we discretize the 3D scene into a $n * n$ grids representation which is friendly to the convolution network. Specifically, we discretize the scene into a mutli-channel top-down grid map noted as F. As shown in Figure 4, the first channel is the wall mask channel. In this channel, we discretize the static objects like walls and pillars. The following channels are object channels. Each channel describes the coarse shape and the position of one object. To describe the object orientation, we use the euler angle between the current pose and the canonical pose. For each channel, a grid on the grid map takes a value of 0 or 1 which represents that this part is empty or occupied. To ensure the consistency of collisions between the representation and the real 3D scene, we adopt a simple dilation sample for object discretization. The details of discretization are illustrate in Figure 5. Thus a scene containing m objects is described by 1 wall mask channel and m object channels. Since the initial state and the target state share the same wall mask channel, the grid map of the inital and target state has $2 * m + 1$ channels. This coarse geometry representation is feed to the convolution network to learn to detect the semantic features of the scene.

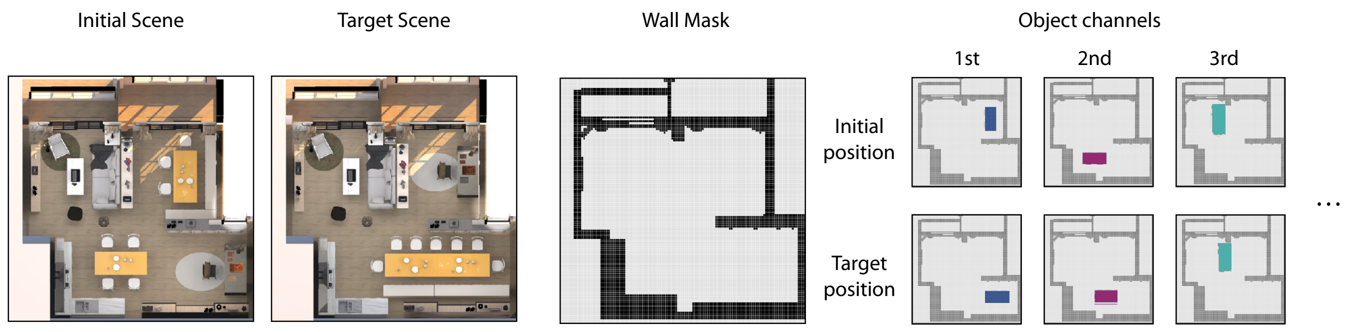

Figure 4: The illustration on the channels of the scene. In this example, there are 20 objects in total. The input of our algorithm which represents this scene has 41 channels. The first channel is the wall mask which characterizes the impassable regions. The following channels are object channels. Each object possesses an intial position channel and a target position channel.

6 Technical Approach

The goal of our apporach is to find an action squence that maximize the total rewards which is typically a reinforcement learning problem. We leverage the power of neural network and MCTS to solve it. The way using the neural network as the searching hints is similar to AlphaGo[33] which achieved a great break through on the challenging problem. In this section, we introduce the details of the our policy network and the searching strategies with network embeded.

6.1 Policy Network

Our policy network is a convolution network which is adopted as the action prior and a fast rollouts agent in our approach. It tries to model the optimal policy of finishing the task. According to the formulation in Section 4, the sequence of actions and observations is actually a finite Markov decision process (MDP). The goal of the agent is to interact with the environment by selecting the actions through which the future rewards will be maximized. Similar to previous works, here we make an assumption that the future rewards are decayed by a factor γ as time step increase. Specifically, we consider the sequence $s_t = x_1, a_1, x_2, a_2, ..., x_t$ at time t, the future accumulated rewards at t is $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the time step at which the game is terminated. Since the action space is discrete, we adopt deep Q-learning[27] and define the optimal action-value function as

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi], \tag{2}$$

where π is the policy the agent follows when selecting the future actions. According to Bellman equation[35], if the optimal value $Q^*(s', a')$ of the squence s' at the next time step is known for all possible action a', then the optimal strategy is to select the action a' maximizing the expected value of $r + \gamma Q^*(s', a')$, Equation 2 can be adapted as

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s', a') | s, a].$$

Here we use the convolution network with weights θ to approximate the Q function, an iterative update is adopted. For iteration i, the Q-Network is trained by minimizing the following loss function

$$Loss_i = \mathbb{E}_{s,a \sim \rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2] \tag{3}$$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q(s', a'; \theta_i - 1) | s, a]$ is the update target and ρ is the distribution over sequences s and actions a.

The input s of our network is composed of the feature map described in Section 5, the finishing tag which indicates the history arrival information and the conflict matrix which describes the spatial relations between obejcts. Since our policy network need to do fast rollouts, it is inefficient to consume all history actions and states. According to our observation, without the history records, the network would be easily trapped in a local minima by jumping between several states that happened before. As a trade-off, we propose a vector v named as finishing tag to mark the arrival history, $v \in \{0, 1\}^m$, where m is the number of object. The ith position of the finishing tag takes a value of 1 or 0, representing the object i has reached its position or not. Figure 6 illustrates the function of finishing tag. With the hint of the arrival information, the agent jumps out of the loop. The output of our network is the estimate of the value for each action.

As described in Section 5, for a input map having $2 * m + 1$ channels, this representation can describe the scene containing number of objects from 1 to m. Note that for a scene containing more than m objects, it is needed to train an larger fixed-dimensional network. In our experiments, the maximum permited object number is x which covers most of the circumstances. We discuss it in limitation and future works.

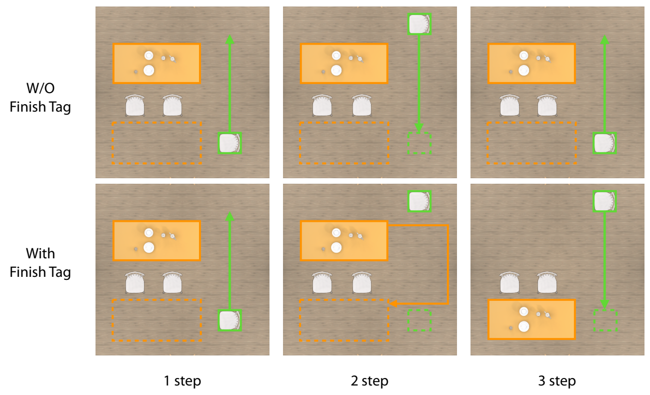

Figure 6: The different results with or without the finishing tag. The first row is the result without the finishing tag. The solid boundling box presents the object, the dashed bounding box represents the target position. The action in the second step is a sub-optimal solution since that after the action of second step, the state goes back to the initial state. The second row is the result with the finishing tag. The agent finds the best solution.

6.2 Searching with Q-Network

Our approach embeds the policy network into an MCTS algorithm which select an action according to lookahead simulations. Each node of the search tree can be characterized by state s. It stores the state, the value of this node $V(s)$, the visit count $N(s)$ and the action a^* to the most valuable child node. Each edge (s,a) stores the selected action a, the reward received after execution $r(s,a)$ and the action prior $P(s,a)$. At the beginning, the tree only has a root node which restores the current state. The tree grows as the searching proceeds. At each simulation time step t, we first select a tree node to expand. Formally, the selection of the tree node s_t is

$$s_t = \arg\max_s (V(s) + u(s))$$

where $V(s)$ is normalized to $[0,1]$ and

$$u(s) \propto \sqrt{\frac{log(N(s_p))}{1+N(s)}}.$$

$u(s)$ is the trade-off term between exploration and the depth of searching. Then we select an legal action for this tree node and expand a leaf node according to the prior $P(s_t,a)$ estimated by the policy network. The leaf node is evaluated through a fast playing out until the game is terminated or reach the maximum number of steps. In the fast rollout, the policy network is used as the policy of the agent. At the end of the simulation, the nodes are updated from the leaf node recursively

$$a^* = \arg\max_a V(s_a)$$
$$V(s) = \gamma V(s_{a^*}) + r(s,a^*)$$
$$N(s) = \sum_{a\in\mathcal{A}} N(s_a)$$

When the search is complete, different from the zero-sum game, the algorithm chooses the best action a^* of the root node as the solution. Here we present the pseudo code of our algorithm 1.

ALGORITHM 1: Q Monte Carlo Tree Searching

```
Input: the current searching tree $\mathcal{T}_t$, the
       Q-network $Q(\cdot,\cdot)$, sampling amount $N$,
       maximum simulation depth $D$, node
       selection function $S(\cdot)$, simulation
       function $E(\cdot,\cdot)$, reward function $R(\cdot,\cdot)$
Output: The best action $a^*$, the next step
        searching tree $\mathcal{T}_{t+1}$
for $i = 1$ to $N$ do
    $n = S(\mathcal{T}_t)$;
    $s = n.state$;
    $a = \arg\max_a Q(s,a)$, $a$ is a legal action and
     not be expanded;
    Expand a new node $n'$ as the child of $n$, the
     edge is action $a$;
    $s' = s$;
    $a' = a$;
    $sum_{rewards} = 0$;
    for $j = 1$ to $D$ do
        $s' = E(s',a')$;
        $reward = R(s',a')$;
        $a' = \arg\max_a Q(s,a)$;
        $sum_{rewards} += reward$;
        if $s'$ is the terminal state then
            break;
        end
    end
    Recursively back-propagate from node $n'$,
     update the reward and the visit count;
end
$a^* = \arg\max_a(\mathcal{T}_t.root[a].reward)$;
$\mathcal{T}_{t+1}$ = the subtree $\mathcal{T}_t.root[a^*]$;
return $a^*$, $\mathcal{T}_{t+1}$;
```

It is worth noting that two different tree nodes may share the same state since much actions are reversible. To prevent this happening, we would check if the state appears before when expanding a new node.

7 Implementation

In this section, we introduce the details of the virtual environment the agent interacts with, the synthesis of our training data and our training details.

7.1 Virtual Environment

To perform deep reinforcement learning, we define a virtual environment to interact with the neural network, execute the action and return the reward as well as the terminal signals. The virtual environment need to perform five kinds of actions, the first four are moving straightly in four directions which is trivially simulated. The fifth is searching a legal route from the current place to the target place. Since the orientation of the object at the initial state may differ from the orientation at the target state, the object needs an operation to rotate to the target orientation. We embed this operation into the fifth kind action because the discretized rotation operation will greatly enlarge the action space and the solution space which makes it harder to search a good solution. The fifth action is performed through an A^* search algorithm[15] on grids map. At each step, the object is able to move to four adjacent grids if there is no collision with other obstacles. During the searching of path, we allow the object rotates to dodge obstacles and adjust its orientation. Specifically, we discretize the orientation of objects into 24 bins and detect collisions for a continous orientation interval. If there is no such a route, the state will not be changed and fail penalty would be returned.

7.2 Data Synthesis

Since our approach only need the layout of the scene to train the network, we randomly synthesize the layouts in some manner. The synthesis of our case takes three stages. At the first stage, we randomly generate some walls. The growth of the wall follows the Bernoulli distribution. In details, a wall starts from a grid at the boundary district and has a intial direction to grow. At each step it grows one unit in the current direction. It chooses a new direction to grow with probability p and terminates the growth with probability q. Here we set $p = 0.5$ and $q = 0.2$. At the second stage, we randomly generate some objects described by rectangles in different size. Though the coarse shape of real objects are arbitrary, the different size of rectangles force the convolution network to learn how to handle some shape variations. At the last stage, we generate the initial position and the target position for each object without overlap. Due to the space limitation, please refer to our supplementary materials for the illustration of synthesized data.

7.3 Training Details

We adopt ϵ-greedy strategy and experience replay during training[27] where $\epsilon \propto exp(i)$. Learning rate is 0.0001. According to Bellman equation, the converge of the policy is guaranteed. In practice, the convergence might be very slow partially due to that y_i in Equation 3 is a bad estimate for the current state. With the aid of MCTS, more accurate estimate of current can be performed by simulation and search, in the meantime, higher computation is needed. Considering the limitation of equipmements and training efficiency, our policy network is trained independently.

8 Experiments

In this section, we design several experiments to evaluate the our approach. We make a user study to compare the performance of our approach with human manipulation. We also do ablation studies to analyse the effects of each component in our approach. Finally, we demonstrate our approach on different interior scenes.

8.1 Comparison against Human Perception

For a more intuitive understanding of how good our algorithm performs, we compare our results with human practices. We develope a mobile application to record human manipulations. The goal of this game is to move the rectangles to their assigned location through 5 given actions. Considering the limitation of screen space on mobile devices, the experiment cases are performed on $35 * 35$ rather than $64 * 64$ grid since the bigger grid will be more easily to tap. To observe the performance of the agent on different problem difficulty, in this experiment, the cases are designed as 4 levels where the case has 5, 9, 13, 17 objects respectively. In each level, we randomly synthesize 20 cases. Every player should pass 1 randomly selected case in each level and one can have access to the next level only after the current level has been passed. This rule is made to avoid that the user lost the interest of the game when they face a hard level at their first playing. It introduces bias to user data between different levels. The records of higher levels are from more experienced players.

We finally collect 1106 playing records from 224 users. Each record saves the moving sequence of a playing round. We compare the mean number of steps needed between our algorithm and human. Since the map size is different from the input of our neural network, we upsample the map to $64 * 64$ to fit the size of input. As shown in Figure 7, the number of mean steps rises rapidly as the rise of object number. Intuitively, the worst complexity of this problem is $O(n^2)$ since the objects may stand in the way of each other's destination. When the problem scale is not large, the decision made by human is somewhat rational. As the problem scale increase, the number of steps needed for human to finish increases rapidly even the players has been trained

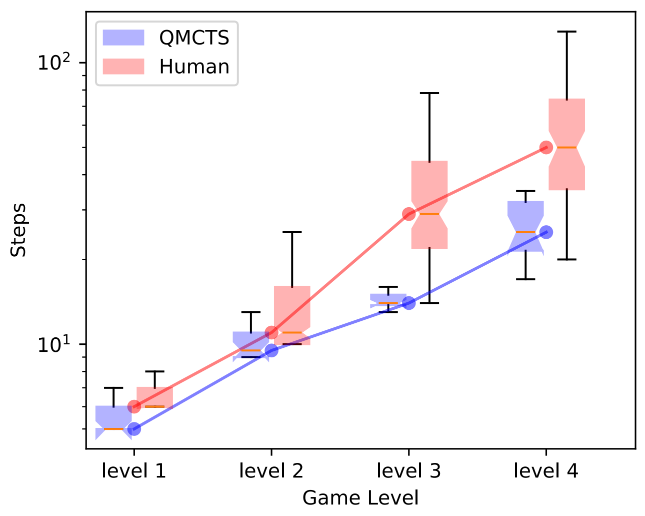

Figure 7: The box plot illusrates the number of steps needed to finish the tasks. Noting that the y-axis is in log scale. Our algorithm significantly outperforms the human practice especially on the complicated situations.

through playing easier levels. To our observation, the addtional steps for human are often used for trial. Human tend to adopt simple and less efficient policy to handle the complex situation because they do not have a good sense of the global state, the attenttions are drawn in local regions. With the power of tree searching, our approach has a better global perception and often gives excellent moves.

8.2 Approach Analysis

The conventional Monte Carlo Tree Search methods suffer from low sampling efficiency and the bad evaluation of the leaf nodes. Adopting the neural network as the expansion prior and the rollout agent gives a way to overcome the two problems. In this section, we compare our approach with a rule-based strategy and do ablation study to analyse our approach. We propose a naive heuristic method as our baseline. This baseline method bases on the observation that the route move often gains reward. This heuristic method selects the action following the manners below: 1) If there is an object can be moved to its destination through the 5th action, select this action, if not, select a random action. 2) If there are more than one objects can select the 5th action, select the one with bigger size. 3) An object will not be moved in two adjacent moving steps.

As described in Section 6.1, our Q-Net is introduced as priors and a quick simulation agent. To demonstrate the effect of it, we compare our approach with the heuristic method and the MCTS hinted by the naive heuristic method mentioned above. Generally speaking, Monte Carlo methods approximate the true solution space as the sampling size of nodes increase. More sampling nodes gives more opportunity to find good solutions. We compare the performance of those approaches with 100 and 200 sampling nodes. In this experiment, our Q-network is trained for 20,000 iterations. We use the synthesized data to evaluate those approaches. Specifically, we synthesize 20 cases. The cases are performed on $64 * 64$ grids, each case has 13 objects in its scene. The results are illustrated in Table.1. The rule-based heuristic approach has some cases that cannot be solved since the rule-based method is esaily trapped in some simtuations that cannot be covered by the rule. With the help of the searching tree, the agent is able to jump out those situations. Our approach achieves the same success rate but with less sampling nodes, higher total rewards and less steps comparing to MCTS. Comparing to the baseline, the MCTS methods need more time during inference beacuse it need to do tree searching and simulation at the leaf nodes. A way to optimize our approach is to do the searching and simulation in parallel. It will be done in our future work.

8.3 Large Scene Performance

In this section, we demonstrate our approach on real large scenes. Beside the Living Room, we show 4 additional different scenes, the Bed Room, the Library, the Restaurant and the Gym. Table 2 illustrates the number of objects, the size of scenes, the number of steps and the running time. We download the scenes from the Internet and rearrange them by our designer as the target arrangement. We first fill the topdown view of the scene into a bounding square and then discretize it into a $128 * 128$ grids map. The resolution of our grids in this experiment ranges from 5.2cm to 15.6cm. We find that the complexity of the task not just depends on the number of objects but more depends on the objects needed to move and the crowded degree. For example, the Gym has more objects than the Bedroom, but it spends less steps to finish the tranformation since the Gym have more free space for objects to move around. The number of objects in the Restaurant is almost twice as the number of objects in the Living Room, but they need almost the same steps to finish the task. It is because that the number of objects in the Restaurant needed to be moved are similar to it in the Living Room.

9 Conclusions

In this paper, we present a novel problem statement of interior object moving procedure. To solve this problem, we also introduce a novel framework using a neural network embeded MCTS method for the automatically programming of the moving sequence. We compare the performance of our approach with human performance. Our algorithm achieves better results. We also demonstrate our algorithm on real large scenes where our algorithm also performs good.

Table 1: The comparison against different agents.

Agent	Sampled Nodes	Mean Steps	Mean Reward	Success Rate	Mean Time (Sec)
Heuristic	-	52.9	11383.0	0.67	0.26
MCTS	100	15.3	13369.3	1.0	137.01
MCTS	200	14.9	13460.0	1.0	225.91
MCTS + RollNet	100	14.2	13608.2	1.0	216.32
MCTS + RollNet	200	13.9	14002.1	1.0	389.47

Table 2: The results on different scenes.

	Objects	Map Size (Meter)	Steps	Total Time (Sec)
Living Room	20	10.0*10.0	34	375.41
Bed Room	6	2.7*6.6	8	35.51
Library	32	13.0*10.5	14	179.42
Restaurant	43	20.0*20.0	36	332.15
Gym	10	10.0*18.5	5	27.87

References

[1] Aberdeen, D., Thiébaux, S., and Zhang, L. Decision-theoretic military operations planning. In *International Conference on Automated Planning and Scheduling* (2004).

[2] Alexopoulos, C., and Griffin, P. M. Path planning for a mobile robot. *IEEE Transactions on systems, man, and cybernetics 22*, 2 (1992), 318–322.

[3] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems* (2016), pp. 3981–3989.

[4] Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems* (2017), pp. 5360–5370.

[5] Bellman, R. A markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[6] Brooks, R. A., and Lozano-Perez, T. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, 2 (1985), 224–233.

[7] Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. Deep blue. *Artificial intelligence 134*, 1-2 (2002), 57–83.

[8] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[9] Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., and Hanrahan, P. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 135.

[10] Frankish, K., and Ramsey, W. M. *The Cambridge handbook of artificial intelligence.* Cambridge University Press, 2014.

[11] Fu, Q., Chen, X., Wang, X., Wen, S., Zhou, B., and Fu, H. Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *Acm Transactions on Graphics 36*, 6 (2017), 201.

[12] Garrido, S., Moreno, L., and Lima, P. U. Robot formation motion planning using fast marching. *Robotics and Autonomous Systems 59*, 9 (2011), 675–683.

[13] Gasparetto, A., Boscariol, P., Lanzutti, A., and Vidoni, R. Path planning and trajectory planning algorithms: A general overview. In *Motion and operation planning of robotic systems.* Springer, 2015, pp. 3–27.

[14] Ge, S. S., and Cui, Y. J. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation 16*, 5 (2000), 615–620.

[15] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[16] Hausknecht, M., and Stone, P. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527 7*, 1 (2015).

[17] Iida, H., Sakuta, M., and Rollason, J. Computer shogi. *Artificial Intelligence 134*, 1-2 (2002), 121–144.

[18] Jan, G. E., Sun, C.-C., Tsai, W. C., and Lin, T.-H. An $o(nlogn)$ shortest path algorithm based on delaunay triangulation. *IEEE/ASME Transactions On Mechatronics 19*, 2 (2014), 660–666.

[19] Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research 32*, 11 (2013), 1238–1274.

[20] Lazano-Perez, T. Spatial planning: A configuration approach. *IEEE Trans. on Computers 100*, 2 (1983), 108–120.

[21] Levine, S., and Koltun, V. Guided policy search. In *International Conference on Machine Learning* (2013), pp. 1–9.

[22] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research 37*, 4-5 (2018), 421–436.

[23] Littman, M. L. Algorithms for sequential decision making.

[24] Lozano-Perez, T. Spatial planning: A configuration space approach. *Autonomous robot vehicles* (1990), 259–271.

[25] Merrell, P., Schkufza, E., and Koltun, V. Computer-generated residential building layouts. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 181.

[26] Merrell, P., Schkufza, E., Li, Z., Agrawala, M., and Koltun, V. Interactive furniture layout using interior design guidelines. In *ACM transactions on graphics (TOG)* (2011), vol. 30, ACM, p. 87.

[27] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[29] ODonoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. Pgq: Combining policy gradient and q. *arXiv preprint arXiv:1611.01626* (2016).

[30] Qi, S., Zhu, Y., Huang, S., Jiang, C., and Zhu, S. C. Human-centric indoor scene synthesis using stochastic grammar.

[31] Rimon, E., and Koditschek, D. E. Exact robot navigation using artificial potential functions. *IEEE Transactions on robotics and automation 8*, 5 (1992), 501–518.

[32] Russell, S. J., and Norvig, P. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[33] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484.

[34] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature 550*, 7676 (2017), 354.

[35] Sutton, R. S., and Barto, A. G. Reinforcement learning: An introduction.

[36] Takahashi, O., and Schilling, R. J. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation 5*, 2 (1989), 143–150.

[37] Wang, K., Savva, M., Chang, A. X., and Ritchie, D. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 70.

[38] Wiering, M., and Van Otterlo, M. Reinforcement learning. *Adaptation, learning, and optimization 12* (2012), 51.

[39] Yeh, Y.-T., Yang, L., Watson, M., Goodman, N. D., and Hanrahan, P. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 56.

[40] Yu, L.-F., Yeung, S. K., Tang, C.-K., Terzopoulos, D., Chan, T. F., and Osher, S. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph. 30*, 4 (2011), 86.

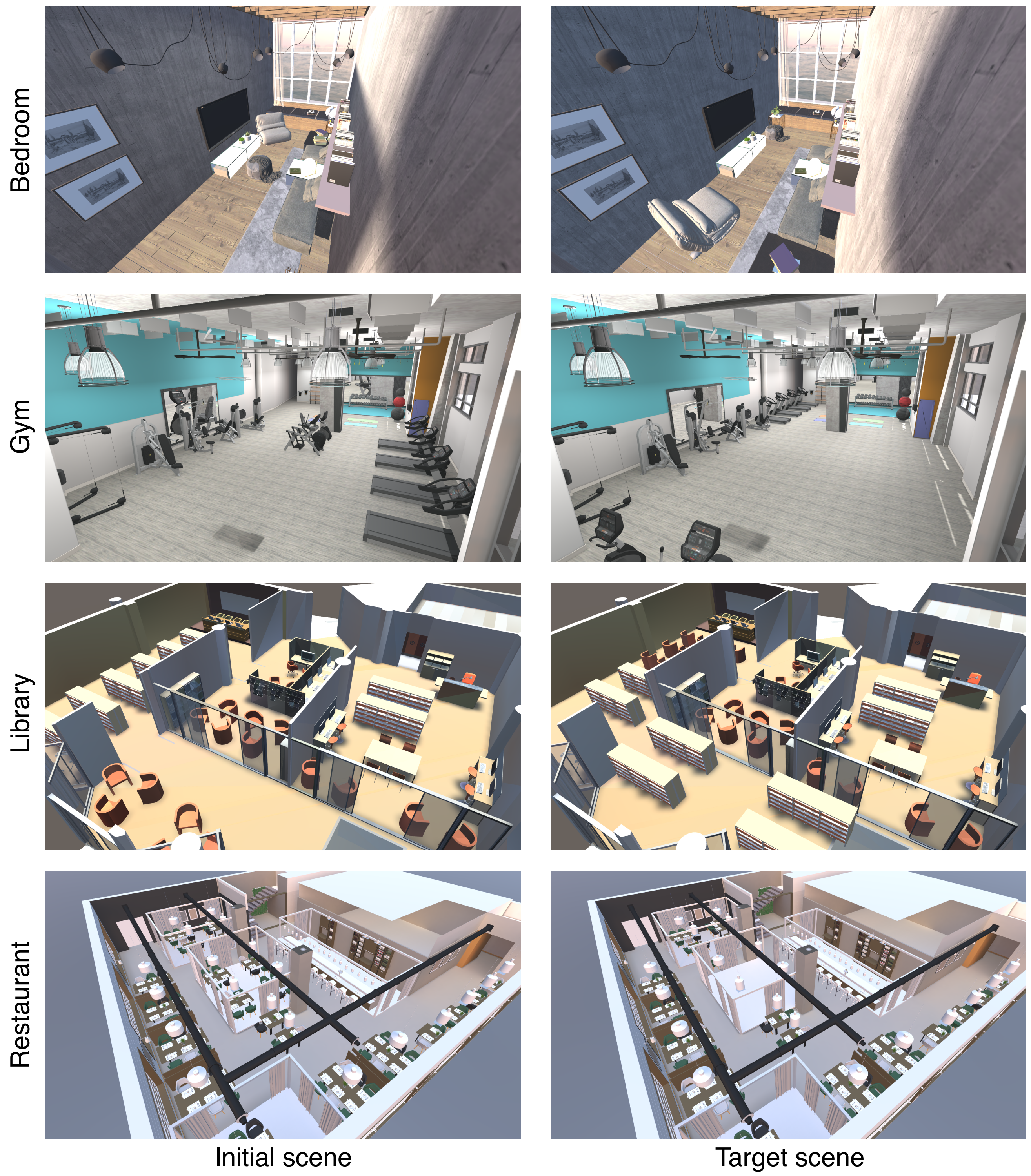

Figure 8: The demonstration of our approach on living room, library, restaurant and factory. There are three columns in each case, the left is the empty layout, the middle is the initial layout and the right is the target layout.