arXiv:1904.10348v1 [cs.RO] 23 Apr 2019

Monte-Carlo Tree Search for Efficient Visually Guided Rearrangement Planning

Sergey Zagoruyko [a,b,*,‡], Yann Labbé [a,b,*], Igor Kalevatykh [a,b], Ivan Laptev [a,b], Justin Carpentier [a,b], Mathieu Aubry [c] and Josef Sivic [a,b,d]

***Abstract*— In this paper, we address the problem of visually guided rearrangement planning with many movable objects, i.e., finding a sequence of actions to move a set of objects from an initial arrangement to a desired one, while relying directly on visual inputs coming from a camera. We introduce an efficient and scalable rearrangement planning method, addressing a fundamental limitation of most existing approaches that do not scale well with the number of objects. This increased efficiency allows us to use planning in a closed loop with visual workspace analysis to build a robust rearrangement framework that can recover from errors and external perturbations.**

The contributions of this work are threefold. First, we develop an AlphaGo-like strategy for rearrangement planning, improving the efficiency of Monte-Carlo Tree Search (MCTS) using a policy trained from rearrangement planning examples. We show empirically that the proposed approach scales well with the number of objects. Second, in order to demonstrate the efficiency of the planner on a real robot, we adopt a state-of-the-art calibration-free visual recognition system that outputs position of a single object and extend it to estimate the state of a workspace containing multiple objects. Third, we validate the complete pipeline with several experiments on a real UR-5 robotic arm solving rearrangement planning problems with multiple movable objects and only requiring few seconds of computation to compute the plan. We also show empirically that the robot can successfully recover from errors and perturbations in the workspace. Source code and pretrained models for our work are available at `https://github.com/ylabbe/rearrangement-planning`

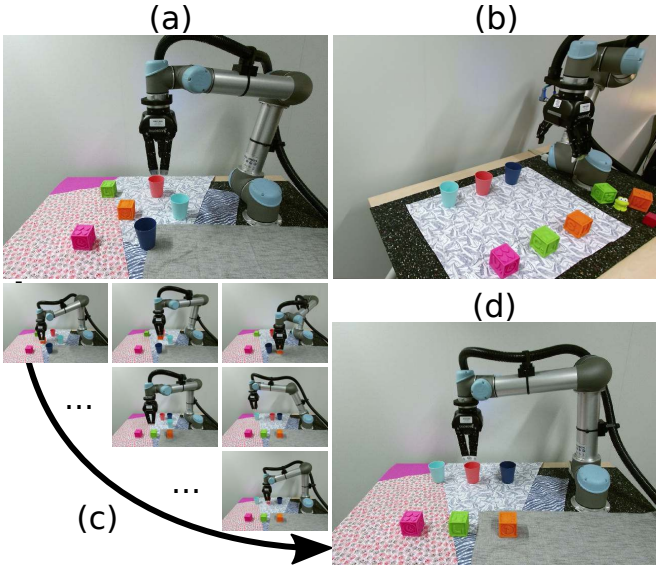

Fig. 1: **Visually guided rearrangement planning.** Given a source (a) and target (b) RGB images depicting a robot and multiple movable objects on a table, our approach estimates the positions of objects in the scene without the need for explicit camera calibration and efficiently finds a sequence of robot actions (c) to re-arrange the source scene into the target scene. Final object configuration after re-arrangement by the robot is shown in (d).

I. Introduction

Using a robot to clean up a room is a dream shared far beyond the robotics community. This would require a robot to localize and re-arrange many objects. This paper presents an approach that makes a step towards the efficiency, scalability and robustness required for such a task. Fig. 1 shows the problem we consider, where colored objects have to be moved from an initial position to a target position. Rearrangement planning has a long history in robotics [1]–[6] and remains an active research topic [7]–[9] in the motion planing community. The goal of rearrangement planning is to find, for a given robot, a sequence of *transit* and *transfer* motions [1], [5] to move a set of objects from an initial arrangement to a target arrangement, while avoiding collisions with the environment. This leads to a complex sequential decision process, whose complexity depends both on the number of objects to move, on the free-space available around the objects, as well as on the kinematics of the robot considered. In order to solve this problem in a reasonable amount of time, several solutions have been proposed in the literature which can be roughly classified into two groups.

Methods in the first group [6], [8], [9] rely on the *task* and *motion* planning hierarchy where a high-level task planner is combined with a local motion planner [3] to solve the rearrangement problem. Methods in the second group [2], [4] aim at solving a single but unified formulation of the problem by using classic probabilistic algorithms such as Rapidly-Exploring Random Tree (RRT) [3]. While both approaches have been shown to work well in practice with few objects, existing methods are typically not able to scale to a large set of objects, because the number of possible action sequences increases exponentially with the number of objects to move. In this work we describe an efficient and trainable approach for rearrangement planning and show empirically that it scales well with the number of objects, by

[a] Département d'informatique de l'ENS, École normale supérieure, CNRS, PSL Research University, 75005 Paris, France
[b] INRIA, France
[c] LIGM (UMR 8049), École des Ponts, UPE, France
[d] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague
[*] equal contribution
[‡] corresponding author: szagoruyko@fb.com

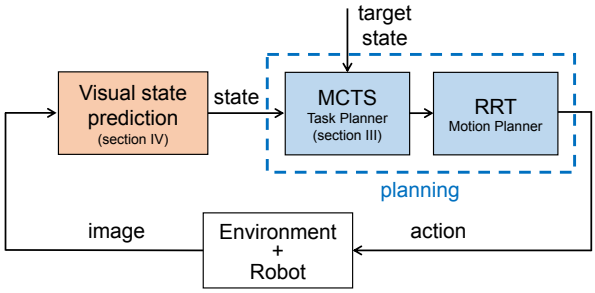

Fig. 2: **Approach overview.** Given an image of the scene, the visual state prediction module outputs a list of objects and their coordinates in the robot coordinate frame. Together with a target state, this serves as input to the planning module, which combines trainable Monte-Carlo Tree Search (MCTS) with low-level robot motion planning.

taking only a few seconds to plan complex rearrangement scenarios for multiple objects.

To demonstrate the benefits of our approach we incorporate it into a complete visually guided rearrangement planning pipeline using a real robot, illustrated in Fig. 2. This pipeline is composed of three main stages. The goal of the first, visual state prediction stage (section IV), is to estimate the position of multiple objects relative to a robot given a single non-calibrated photograph of the scene. This is achieved by extending the deep neural network architecture for calibration-free position estimation of a single object, introduced in [10], to handle multiple objects. The second stage is the high-level planner: according to the current estimate of the object positions, it decides which object to move as well as its new desired position in order to move towards the target state. To break the complexity of the decision process we take inspiration from the recent success of AlphaGo [11] that has demonstrated great planning capabilities in the game of Go. In detail, we rely on the combination of Monte-Carlo Tree Search (MCTS) [12] and neural networks to learn how to select good placement actions according to the current positions of objects and their target arrangement. This stage is our main technical contribution and is detailed in Sec. III. Finally, the third and last stage corresponds to the local controller, a classic RRT-based algorithm allowing to plan grasping, transit and transfer robot movements given the high-level plan computed by the MCTS planner.

II. Related work

We build on results in robotics, reinforcement learning and computer vision, which we review below.

Rearrangement planning. Rearrangement planning is NP-hard [13]. As a result, existing hierarchical [6], [8], [9] and randomized [2], [4] methods do not scale well with the number of objects. Recent works have investigated employing (deep) reinforcement learning for solving non-prehensile rearrangement planning but either required human demonstration [14] or needed a large amount of training episodes to converge towards a valid deep neural network policy [15] and was only demonstrated in simulation. In contrast, our approach is able to learn an efficient policy with only one thousand episodes and is able to directly operate from real images.

Monte-Carlo Tree Search (MCTS) is a reinforcement learning based tree search algorithm [16]. It was used in the core of AlphaGo, the first system able to achieve human performance in the game of Go [17], combined with neural networks to help facilitate the search. First versions of AlphaGo were trained using human knowledge, and only recently AlphaGo Zero and AlphaZero [11] were able to train from scratch using self-play only. Applications of MCTS in robotics are typically constrained to discrete action spaces. Applying MCTS to a continuous action space remains an open research problem [18]. Furthermore, very few open source implementations are available [19]. To the best of our knowledge, the AlphaZero algorithm has not yet been applied to robotics. Yet we think its applications could lead to remarkable advances in the field and we make the first steps towards this perspective in this work.

Vision-based object localization. In robotics, fiducial markers are commonly used for detecting the objects and predicting their pose relative to the camera [20]–[22] but their use limits the type of environments the robot can operate in. This constraint can be removed by using a trainable object detector architecture [23]–[27]. However, these methods often require gathering training data for the target objects at hand. In addition, these methods estimate the pose of the objects in the frame of the camera and using these predictions for robotic manipulation requires calibration of the camera system with respect to the robot. The calibration procedure [28], [29] is time-consuming and must be redone each time the camera is moved. More recently, [10] proposed to directly predict the position of a single object in the robot coordinate frame, using hundreds of thousands of synthetically generated images using domain randomization [10], [30]. We build on [10]'s state-of-the-art approach and extend it for predicting the 2D positions of multiple objects relative to a robot.

III. Re-arrangement Planning with Monte-Carlo Tree Search

This section explains how we exploit the combination of Monte-Carlo Tree Search and a trainable neural network in order to learn an optimistic policy for solving the underlying decision process of rearrangement planning. We first review Monte-Carlo Tree Search and we then explain how we adapt it for rearrangement planning.

A. Review of Monte-Carlo Tree Search

Monte-Carlo Tree Search [12] is a reinforcement learning-based algorithm which allows to efficiently explore and exploit a sequence of actions in order to solve complex decision processes, such as the game of Go or chess. The MCTS decision process is represented by a tree of states and actions, where each node is associated to a state value s, and each edge represents an action, denoted by a. For each pair (s, a) in the tree, a reward function r gives a scalar representing the level of accomplishment of the task to solve.

The goal of MCTS is then to find the most optimistic path, i.e. the path that maximizes the expected reward, starting from the root node and leading to a leaf node solving the task.

MCTS is an iterative algorithm which builds the decision tree incrementally, by random sampling of the space of possible actions and the set of already visited states. MCTS is composed of four main stages: *selection, expansion, simulation* and *backpropagation*:

i) during the *selection stage*, both the action a and the node s to expand are selected using the Upper Confidence Bound (UCB) formula:

$$U(s,a) = Q(s,a) + \sqrt{\frac{2\log N}{n}}, \tag{1}$$

where N is the total number of node visits, n is the number of times a child node has been visited, and $Q(s,a) = \frac{w(s,a)}{n}$ is the expected value at state s when choosing the action a, with $w(s,a)$ the cumulative reward computed during the backpropagation stage (see below). The second term in $U(s,a)$ seeks to encourage the visit of sparsely visited nodes (with low visit counter value n). The node to expand is then selected as the action a_{selected} which maximizes $U(s,a)$ ($a_{\text{selected}} = \text{argmax}_a\, U(s,a)$);

ii) in the *expansion stage*, a child node is added to the tree using the selected optimistic action a_{selected};

iii) in the *simulation stage*, the simulation is performed by taking a sequence of actions from the newly expanded node until a terminal node is encountered, and an outcome reward r is collected;

iv) this reward signal is then back-propagated towards the root node, where the cumulative rewards w of all the parent nodes are updated with the outcome reward r according to $w(s,a) = w(s,a) + r$ with $w(s,a) = 0$ at the beginning). In addition, during the back-propagation phase, all the counters n and N are incremented to reflect the new visit of the nodes.

The search process is run iteratively until the problem is solved or a maximal number of iterations or the time limit are reached.

B. MCTS planning with goal-aware parameterization

We now describe how we apply the MCTS algorithm to our planning setup. Without loss of generality, we limit the scope to prehensile rearrangement planning problems where objects lie on the same surface, similarly to [15], even if our solution may be applied to other contexts. Therefore, in the following we assume that the complete state of the object is given by its 2D position on the surface, and this information is sufficient for grasping the object. The movements are constrained by the limited workspace, and actions should not result in collisions with other objects or in an object being placed outside of the workspace. Then, given the source and target arrangements of the objects, the planner has to compute a sequence of actions, which transforms the source arrangement into the target arrangement. An example of source and target configurations is depicted in Fig. 3. As the number of objects increases, the complexity of the planning procedure grows exponentially, and the planner needs to plan a dozen or more steps in advance to satisfy all the collision and workspace constrains.

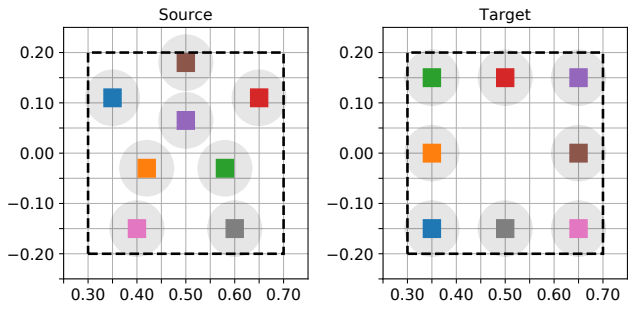

Fig. 3: Examples of source and target object configurations. The planner has to find a sequence of actions (which object to move and where to displace it inside the workspace), while avoiding collisions with other objects. Workspace limits are shown as dashed lines. Grey circles depict the collision radius for each object.

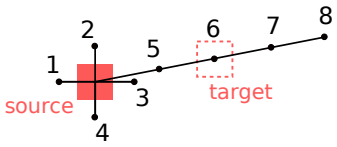

Fig. 4: Goal-aware parameterization with $k = 8$ different discrete actions. The four first actions correspond to the fixed distance lateral motion, while executing, for example, the action 6 brings the object directly into the target location.

The rest of this section is organized as follows. First, we define the goal-aware action space parametrization, which provides a discrete action space on which MCTS operates. Then we describe our approach for solving rearrangement planning using MCTS. Finally, we show how to improve the efficiency of exploration in MCTS by learning the action policy.

Goal-aware parametrization. MCTS operates on a discrete set of actions in order to build and explore the decision tree. In the context of rearrangement planning, for a given state s we first need to decide which object to move *and then* to which new position we want to move the selected object. While the first decision is discrete by nature, we need to discretize the displacement action. There is a large number of ways to perform this discretization. We divide the space of actions into 8 different choices, depicted in Fig. 4: four lateral motions of a fixed distance and four motions of different distances but following the direction between the source and target positions. As shown in Fig. 4, this last choice enables the planner to reach the targeted position in a single pass. It is important to notice that those displacements are only valid if no collision occurs (the collision volume being represented by bounding circles, see Fig. 3).

Monte-Carlo Tree Search-based planner. In addition to a discrete action space, we need to provide a suitable reward function to MCTS in order to give an informative feedback

on the current action being made. We represent the reward function by the following continuous function over the state space:

$$r(s_{\text{leaf}}|s_{\text{target}}) = d(s_{\text{init}}, s_{\text{target}}) - d(s_{\text{leaf}}, s_{\text{target}}), \quad (2)$$

which computes the negated distance between the current state s_{leaf} and the known target state s_{target} plus distance between initial state s_{init} and target. The first term is constant, but important for learning value estimators. This reward signal is only evaluated when expanding a leaf node and its value is back-propagated towards the root node as explained in section III-A. The depth of the tree is limited to $n_{\text{objects}}+\gamma$, where n_{objects} is the number of objects to rearrange and $\gamma = 3$ is a hyperparameter. While this value may appear to be relatively low, this maximum depth is largely sufficient to solve all the rearrangement planning problems considered in Section V.

Training policy networks for MCTS. Even if MCTS seeks to lower at best the number of expansions of non-optimistic branches in the decision tree (thanks to the UCB equation (1)), the number of iterations needed to solve a rearrangement planning problem can become very large in practice. The number of iterations grows exponentially with the number of objects to move and the number of actions. To speed up the search we propose to apply a simplified version of the AlphaGo Zero [11] resolution strategy.

To achieve that, we first collect few thousand of MCTS rollouts assuming a uniform prior over the action space and exploiting the UCB equation (1) to expand the tree in an optimistic way. Using MCTS rollouts, we then train a neural network to predict both the probability $\pi(s, a, s_{\text{previous}})$ of the best action a as well as the expected value $\nu(s, a, s_{\text{previous}})$ given the values of the current and previous states s and s_{previous}. Adding the state history helps the neural network to learn and predict the most optimistic policies and also prevents from looping when predicting the next best action to perform.

To train network with parameters θ on successful MCTS playouts, we use a loss function similar to the one used in AlphaGo Zero [11]:

$$l(s, a, s_{\text{previous}}) = (Q - \nu)^2 - \pi^\top \log p + c\|\theta\|^2, \quad (3)$$

where the first term is the mean-squared error between the estimated value $Q(s, a)$ and the value ν predicted by the neural network, the second term is the cross-entropy loss between the estimated probability value p of an action to occur and the same value π predicted by the neural network. The last term is regularization of the parameters. We omit the arguments of the functions to simplicity the notations.

We use a multi-layer perceptron (MLP) composed of two hidden layers as a policy network, which is fast to evaluate and simple to optimize. We apply batch normalization on each layer and employ ReLU activations between each layer. In order to reduce the amount of noise, we only select nodes whose visit counter (n) is higher than a certain threshold. The downside of the current approach is that a new network has to be retrained with the changing number of objects. Alternatively, a convolutional network could be employed to deal with any number of objects, where the state could be converted into a top-down image. Another benefit of using convolutional networks would lie in implicit modeling of collisions. We plan to explore this direction in future work.

When exploiting the neural network to predict the next action, we rely on a variant of UCT with predictor [31]:

$$U(s, a) = Q(s, a) + c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N_b}}{n + 1}, \quad (4)$$

where $\sum_b N_b$ is a sum of visits of all nodes on a path to the root node, and c_{puct} is a constant controlling the ratio between exploitation and exploration.

Finally, we found that training such a neural network is a relatively simple and quick process, allowing to significantly reduce the number of MCTS iterations required to solve rearrangement, as we show experimentally in Section V.

IV. Visual scene state prediction with multiple objects

Our visual system takes as input a single photograph of a scene taken from an uncalibrated camera and predicts a workspace state that can be used for rearrangement-planning. More precisely it outputs the 2D positions of a variable number of objects in the coordinate system of the robot. In contrast to [30], we do not assume that the different types of objects are known at training time.

This problem is difficult because the scene can contain a variable number of objects, placed on different backgrounds, in variable illumination conditions, and observed from different viewpoints, as illustrated in Fig. 1. To address these challenges, we design a visual recognition system that does not require explicit camera calibration and outputs accurate 2D positions which can then be used for grasping. Moreover, while we deploy our system on a real robot, we show that it can be trained entirely from synthetic data using domain randomization, avoiding the need for real training images.

Our approach is summarized in Fig. 5. We first use a convolutional neural network (CNN) that outputs a dense 2D position field in the robot coordinate frame and a foreground-background mask. We combine these outputs by associating to each pixel within the foreground mask a 2D point in the robot coordinate frame. We then cluster the associated 2D point set using mean shift [32] to identify the individual objects and obtain their 2D coordinates in the scene. The result of this clustering procedure can be transformed into an image-level instance segmentation that we exploit to extract image patches around each object. We use these patches and their CNN features to recognize each object in the current image as one of the reference objects. In particular, if the target state is described by a photograph of the target workspace, the same instance segmentation process can be applied to the target image to obtain reference images for each object. In the rest of this section, we present in more details the different components of our recognition system.

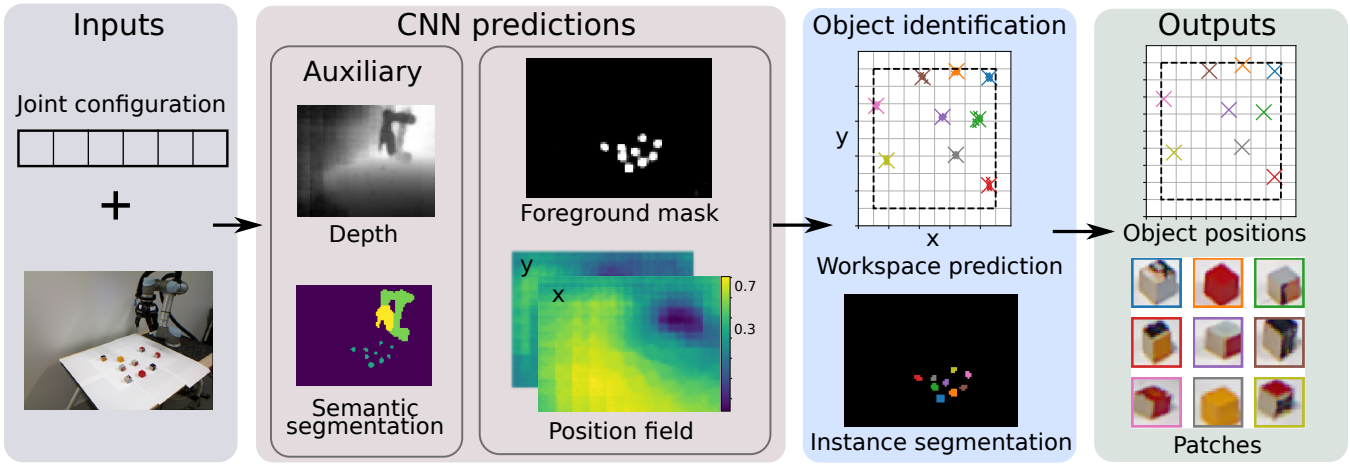

Fig. 5: **The visual recognition system.** The input is an image of the scene captured by an uncalibrated camera together with the joint configuration vector of the depicted robot. Given this input a convolutional neural network (CNN) predicts the foreground-background object segmentation mask and a dense position field that maps each pixel of the (downsampled) input image to a 2D coordinate in a frame centered on the robot. Including auxiliary tasks of predicting scene depth and semantic segmentation improves the overall performance. We then combine the estimated masks and position fields to obtain a set of 2D candidate object positions in the workspace. This set of candidate positions is clustered using mean shift to identify the individual object instances and their segmentation.

Fig. 6: Examples of synthetic training images. We generate images displaying the robot and a variable number of objects in its workspace. The images are taken by cameras with different viewpoints and depicting large scene appearance variations.

A. *Position prediction network.*

In this section, we give details about the network we use for predicting the dense 2D position field and the object segmentation mask.

Architecture. Our architecture is based on ResNet-18 [33]. We remove the average pooling and fully connected layer and replace them by four independent heads. All the heads use the same architecture: four transposed convolution layers with batch normalization and leaky ReLU activations in all but the last layer. We use a dilation factor of 2 for the first two transposed convolution layers. The resolution of the input image is 320 $\times$ 240 and the spatial resolution of the output of each head is 85 $\times$ 69. We add the 6D joint configuration vector of the robot to the network by copying it into a tensor of size 320 $\times$ 240 $\times$ 6, and simply concatenating it with the three channels of the input image. The four heads predict a depth map, a pixel-wise semantic segmentation, an object mask and a 2D position field. While only these last two are used at test time, we found that predicting depth and semantic segmentation during training increased the localization accuracy at test time. These outputs are visualized in the Fig. 5.

Synthetic training data. Following [10], [30], we use domain randomization for training our network without requiring any real data. We generate two million images displaying the robot and a variable number of objects with various shapes (cubes, cylinders and triangles) in its workspace. In each scene, we randomly sample from 1 up to 6 objects, with various dimensions between 2.5 and 8 cm. Examples of training images are shown in Fig. 6. Randomization parameters include the textures of the robot and objects, the position of the gripper, the position, orientation and field of view of the camera, the positions and intensities of the light sources and their diffuse/ambient/specular coefficients. In addition, we add random crops and small random 2D rotations. Note that the large variation in appearance of the generated training data is central to the success of the method.

We also render depth and semantic segmentation for the following five classes: cube, cylinder, triangle, robot arm, gripper. To make training easier, the position and depth targets are centered and scaled to have zero mean and unit variance on the full training dataset.

Training procedure. We train our network by minimizing the following loss:

$$\mathcal{L} = \mathcal{L}_{\text{pos}} + \mathcal{L}_{\text{mask}} + \mathcal{L}_{\text{segm}} + \mathcal{L}_{\text{depth}}, \tag{5}$$

where the individual terms are explained next. For the position field loss we use $\mathcal{L}_{\text{pos}} = \sum_{i,j} \delta_{i,j} \left[(\hat{x}_{i,j} - x_{i,j})^2 + (\hat{y}_{i,j} - y_{i,j})^2 \right]$ where (i, j) are the pixel coordinates in the output; $\delta_{i,j}$ is the binary object mask; $x_{i,j}$, $y_{i,j}$ are the ground truth 2D coordinates of the center of the object (that appears at pixel (i, j)) and $\hat{x}_{i,j}, \hat{y}_{i,j}$ are the components of the predicted position field. We express the 2D coordinates of the objects of the scene in a coordinate frame that rotates with the base of the robot in order to compensate for the lack of non moving robot parts. At test time, we use the (known) joint configuration of the robot to transform them to the fixed robot coordinate

frame. For $\mathcal{L}_{\text{mask}}$, $\mathcal{L}_{\text{segm}}$ and $\mathcal{L}_{\text{depth}}$ we use, respectively, the following standard losses: binary cross entropy loss, cross entropy loss and mean squared error. Note that $\mathcal{L}_{\text{segm}}$ and $\mathcal{L}_{\text{depth}}$ are auxiliary losses used only for training, similar to [34]. We use the Adam optimizer [35] and train the network for 20 epochs, starting with a learning rate of 10^{-3} and decreasing it to 10^{-4} after 10 epochs.

B. Identifying individual objects

The model described above predicts a dense 2D position field and an object mask but does not distinguish individual objects in the scene. Hence, we use a simple post-processing step to group pixels belonging to each individual object.

Applying a threshold to the predicted mask yields a binary object segmentation. The corresponding pixels of the 2D position field provide a point set in the robot coordinate frame. We use the mean-shift algorithm [32] to cluster the points corresponding to the different objects and obtain an estimation of the position of each object. The resulting clusters are then used to identify pixels belonging to each individual object, i.e. obtain instance segmentation of the input image. We use the resulting instance segmentation to extract image patches that describe the appearance of each object in the scene.

C. Source-Target Matching

To perform rearrangement, we need to associate each object in the current image to an object in the target configuration. Here, we assume that both the source and target objects are visually represented using image patches. These image patches are obtained using the visual recognition system described in the previous subsections. Our goal is thus to match the patches extracted from the current image to the image patches of the target objects. We designed a simple procedure to obtain matches robust to the exact position of the object within the patches, their background and some amount of viewpoint variations. We rescale the input patches to 64×64 pixels and extract conv3 features using an AlexNet [36] network trained for ImageNet classification. We then run a Hungarian algorithm [37] to find the one-to-one matching between source and target images maximizing the sum of cosine similarities between the extracted features.

V. Experiments

We start by evaluating planning and visual scene state estimation separately, demonstrating that: (i) our MCTS task planner scales with the number of objects and is efficient so that it can be used used online, i.e. recomputing the plan after each movement; (ii) the visual system detects and localizes the objects with an accuracy sufficient for grasping. Finally, we evaluate our full pipeline in challenging setups and demonstrate that it can perform the task efficiently and can recover from errors and perturbations as also shown in the supplementary video.

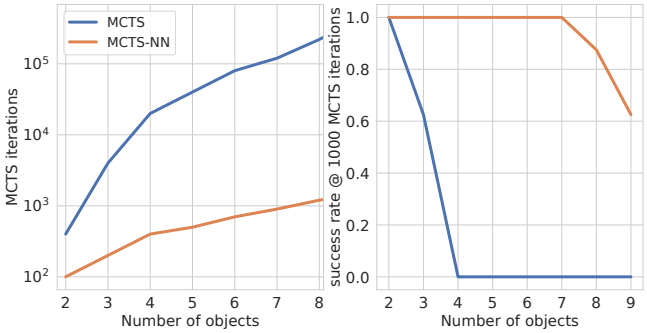

Fig. 7: Left: The number of MCTS iterations required to solve rearrangement planning (log-scale), right: success rate at 1000 MCTS iterations in 8 random source and target initializations. The results demonstrate the significant speed-up (left) and improved success rate (right) of our MCTS planner with learnt policy network (MCTS-NN) compared to the vanilla MCTS planner (MCTS).

A. MCTS planning

To evaluate planning capabilities and efficiency we randomly sample source and target configurations of 2 to 9 objects in workspace, and run MCTS with and without pretrained policy networks. It is difficult to find a valid configuration for more than 9 objects, due to our workspace constraints. We then report success rate as a ratio of playouts matching target after $n+3$ or less steps to the total number of playouts. We find that 8 playouts is representative of planning capabilities. Fig. 7 shows the results. Left plot shows the approximate number of iterations required to achieve 100% success rate over 8 playouts. It can be seen that the number of MCTS iterations is exponential in the number of objects in workspace, and MCTS with pretrained policies (MCTS-NN) is able to achieve the same success rate using order of magnitude less iterations. In Fig. 7 (right) we plot the success rate at 1000 iterations. It can be seen that success rate of pretrained policies drops at 8 objects, which is due to the difficulty of obtaining training data from successful playouts that need very large number of iterations. This limitation can be overcome by using AlphaZero, which uses the network during training to generate new samples. We leave this direction of improvements to future work.

Overall, we find that our unoptimized implementation of MCTS with pretrained policies is 4-5 times faster than highly efficient C++ MCTS. For example, it takes around 1s to compute 6 object plan with 2-layer policy MLP with 32 neurons in each hidden layer, on a single core of 2 GHz Xeon E5-2620 CPU. The implementation can be highly optimized in several directions: policy inputs could be batched, network computed on GPU, batch normalization and nonlinearity fused into a single operation with matrix multiplication. We have also implemented a model-free RL baseline for the planning task, using state-of-the-art PPO training algorithm [38]. It is able to produce 6-7 step plans for 2-3 objects, but struggles to solve environments with more objects, due to the large and imbalanced action space.

Fig. 8: Example images from the dataset that we use to evaluate the accuracy of object position estimation as a function of number of objects.

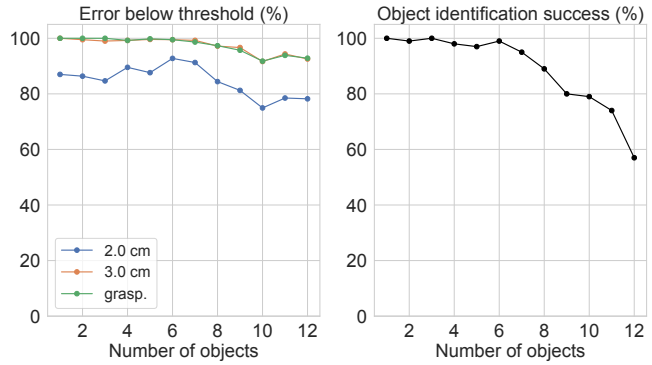

Fig. 9: Evaluation of our visual system for a variable number of objects. We report the localization accuracy (left) and the percentage of images where all objects are correctly identified (right).

B. Visual scene state estimation

Experimental setup. To evaluate our approach, we created a dataset of real images with known object configurations. We used 1 to 12 small 3.5 cm cubes, 50 configurations for each number of cube, and took photographs from 2 cameras for each configurations, leading to a total of 1200 images from 600 different configurations. Examples of evaluation images are presented in Fig. 8.

Single object accuracy. When a single object is present in the image, the mean-shift algorithm always succeeds and the precision of our object position prediction is 1.2 ± 0.5cm. This is comparable to the results reported in [30] for positioning of a known object with respect to a known table without occlusions and distractors, 1.3 ± 0.6cm, and to results reported in [10] for the coarse alignment of a single object with respect to a robot 1.7 ± 3.4cm. The strength of our method however is that this accuracy remains constant for up to 6 previously unknown objects, a situation with which neither [30] nor [10] can deal with.

Accuracy for multiple objects. In Fig. 9, we report the performances of the object localization and object identification modules as a function of the number of objects. For localization, we report the percentage of objects localized with errors below 2 and 3 cm. The 3 cm accuracy approximately corresponds to the success of grasping, that we evaluate using a simple geometric model of the gripper.

Note that for up to 6 objects, for which the visual recognition network is trained, the identification and grasping success rates are close to 100%. For more objects, the accuracy of both the object identification and object localization slightly tends to decrease because objects start to occlude or overlap each other.

Benefits of auxiliary predictions. To show the benefits of our auxiliary losses on depth and semantic segmentation, we compute the average performance of our pipeline for 600 images containing less than 6 objects. The average error is 1.6 ± 0.9cm without the auxiliary losses and 1.2 ± 0.6cm with the auxiliary losses, demonstrating they help in training the network despite not being directly used at test time.

Discussion. Our experiments demonstrate that our visual predictor of the scene state works almost perfectly in the scenario for which it has been trained, i.e. 1 to 6 objects well separated from each other. They also show that the visual predictor generalizes reasonably well to more challenging cases for which it has not been trained, such as more objects closer to each other. Our method is fast, recognizing objects in the scene in 100 ms on a modern GPU. Thus, it can be used for online rearrangement planning. Finally, note that our experiments demonstrate that we are able to cope with an unknown number of objects, which is not the case for most existing methods, in particular [30] and [10].

C. Real robot experiments using full pipeline

We evaluated our full pipeline, performing both online visual scene estimation and rearrangement planning by performing 38 rearrangement tasks, each with 6 objects. In each case, the target configuration was described by an image of a configuration captured from a different viewpoint, with a different table texture and a different type of camera. Examples of target image configurations and initial images are shown in Fig. 10, where the differences in texture and viewpoint are visible.

Despite the very challenging nature of the task, our pipeline succeeded in correctly solving $33/38$ of the experiments. In case of success, our pipeline uses on average 10.7 steps. The 5 failure cases were due to:

- the robot stopped because it collided with an object due to imprecise object localization in 2 cases;
- one object out of the 6 was not detected in 2 cases;
- in one case, the process was not finished after 21 moves and seemed to be looping (adding a longer history as input of the neural network present at the high planning level might be helpful).

Interestingly, the successful cases were not always perfect runs, in the sense that the policy was not optimal or that the visual estimation confused two objects at one step of the matching process. However, our system was able to recover robustly from these failures because it is applied in a closed-loop fashion.

In the supplementary video[1], we show that our system can also recover from external perturbations, such as somebody moving cubes during the rearrangement. The supplementary video also shows additional experiments including objects other than cubes, visual prediction of scene states with different backgrounds and using a hand-held camera.

[1] https://www.youtube.com/watch?v=fS5tTa_Tl1Y

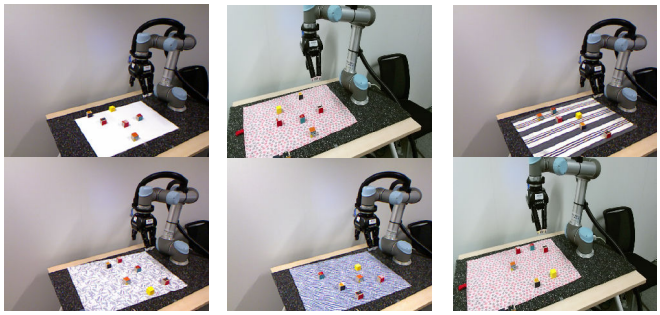

Fig. 10: Images used for our full pipeline experiments on a real robot. Each column shows an example of source (top) and target (bottom) configurations that we give as input to our system.

VI. Conclusion

We have demonstrated a robust and efficient system for online rearrangement planning, that can scale to many objects and recovers from perturbations, without requiring any calibrated camera or fiducial markers on objects. To the best of our knowledge, such a system has not been shown in previous work. At the core of our approach is the idea to apply recent advances in reinforcement and deep learning to rearrangement planning, which leads to important speed-ups that allow to perform online planning. Currently, one of the main limitations is the range of objects that can be recognized and grasped. While our approach is tolerant to certain amount of object variation as we show in the supplementary video, localizing and manipulating arbitrary objects remains an open challenge. Another limitation is in planning, where we need to manually specify the set of possible actions. One promising direction to overcome this issue lies in the extension of MCTS towards continuous action representations, as introduced in [39] but not yet applied to robotics applications.

Source code

Source code to reproduce our experiments is available online[2]. We also provide pretrained models for the UR5 robot and 3-finger Robotiq gripper.

Acknowledgements

We thank Loïc Esteve and Ignacio Rocco for helpful discussions. This work was partially supported by the DGA RAPID projects DRAAF and TABASCO, the MSR-Inria joint lab, the Louis Vuitton - ENS Chair on Artificial Intelligence, the ERC grant LEAP (No. 336845), the CIFAR Learning in Machines&Brains program, and the European Regional Development Fund under the project IMPACT (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000468).

References

[1] R. Alami, T. Simeon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *The fifth international symposium on Robotics research*. MIT Press, 1990, pp. 453–463.

[2] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

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

[4] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *ICRA*, 2007.

[5] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now." 2011.

[7] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach," in *ICRA*, 2014.

[8] A. Krontiris and K. E. Bekris, "Dealing with difficult instances of object rearrangement." in *Robotics: Science and Systems*, 2015.

[9] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach." in *Robotics: Science and systems*, 2016.

[10] V. Loing, R. Marlet, and M. Aubry, "Virtual training for a real application: Accurate object-robot relative localization without calibration," *IJCV*, 2018.

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. R. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[12] R. Munos *et al.*, "From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning," *Foundations and Trends® in Machine Learning*, vol. 7, no. 1, pp. 1–129, 2014.

[13] G. Wilfong, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.

[14] J. E. King, V. Ranganeni, and S. S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," *ICRA*, 2017.

[15] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," *ICRA*, 2018.

[16] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Euro. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Germany: Springer, 2006, pp. 282–293.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[18] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0c: Alpha zero in continuous action space," *CoRR*, vol. abs/1805.09613, 2018.

[19] Yuandong Tian, Jerry Ma*, Qucheng Gong*, Shubho Sengupta*, Zhuoyuan Chen, James Pinkerton, and C. Lawrence Zitnick, "Elf opengo: An analysis and open reimplementation of alphazero," *CoRR*, vol. abs/1902.04522, 2019.

[20] C. Feng, Y. Xiao, A. Willette, W. McGee, and V. R. Kamat, "Towards autonomous robotic in-situ assembly on unstructured construction sites using monocular vision," in *Proceedings of the 31th International Symposium on Automation and Robotics in Construction*, 2014, pp. 163–170.

[21] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern Recognit.*, vol. 51, pp. 481–491, Mar. 2016.

[22] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, June 2014.

[23] J. Wu, B. Zhou, R. Russell, V. Kee, S. Wagner, M. Hebert, A. Torralba, and D. M. S. Johnson, "Real-time object pose estimation with pose interpreter networks," in *IROS*, 2018.

[24] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," in *Robotics: Science and Systems*, 2018.

[25] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. T.

[2] https://github.com/ylabbe/rearrangement-planning

Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *CoRL*, 2018.

[26] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker Jr, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge," in *ICRA*, 2016.

[27] T. Hodaň, J. Matas, and Š. Obdržálek, "On evaluation of 6D object pose estimation," in *ECCV Workshops*, 2016.

[28] R. Horaud and F. Dornaika, "Hand-Eye calibration," *Int. J. Rob. Res.*, vol. 14, no. 3, pp. 195–210, June 1995.

[29] J. Heller, M. Havlena, A. Sugimoto, and T. Pajdla, "Structure-from-motion based hand-eye calibration using L-∞ minimization," in *CVPR*, 2011.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *IROS*, 2017.

[31] C. D. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, pp. 203–230, 2010.

[32] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[34] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-Real via Sim-to-Sim: Data-efficient robotic grasping via Randomized-to-Canonical adaptation networks," *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[37] B. Y. H. W. Kuhn, "The hungarian method for the assignment problem," in *Naval Res. Logist. Quart*, 1955.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[39] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous Upper Confidence Trees," in *LION'11: Proceedings of the 5th International Conference on Learning and Intelligent OptimizatioN*, Italy, Jan. 2011.