

Appendix

A. limitations

Observation model Our algorithm can show differences in performance depending on how we set up the observation model. Our algorithm approximates the belief through a particle filter, and the weight of the particles is proportional to the observation likelihood. If an inappropriate observation model is set, the updated belief will not properly reflect this information even if the information gathering action is taken. Most failures in fetch domain occurred because there was no particle similar to the true state. For example, We set the Gaussian model using Chamfer distance as the observation model in fetching domains, as described in section 4. This model can sometimes be calculated to have a small distance between the actual and other class objects when parts of objects in different classes are similar to each other. This was the main reason for the task to fail when we set the number of particles to small. You could try to deal with this problem by increasing the number of particles, but this is not practical because it increases planning time. Therefore, in order for the algorithm to work effectively with a small particle, it is necessary to carefully set the observation space and observation model so that the observation obtained through the information gathering action can effectively reflect information on the true state.

Belief generator Our algorithm is difficult to deal with unseen objects in fetching domain due to the limitations of the belief generator. As far as we know, there is no perception algorithm yet that can predict the shape and pose of all objects present in the scene through one image obtained from a fixed view and propose particles with reflecting uncertainty by occlusion. Therefore, we limited the classes of possible objects in the domain and assumed that if we could know the class of each object, we could know the shape of the object, in order to construct a generating pipeline that can operate by combining existing algorithms, such as Mask R-CNN [1] with dropout sampling [2] and PointNet [3]-based pose estimator.

Planning time As you can see in the supplementary video, our algorithm can make a plan with less time than unguided search. However, it is not enough to apply it as a real-time yet. Our algorithm performed fairly well with a small number of particles through guided search, but more particles are needed for a higher success rate. Our algorithm based on serial tree search has a trade-off between planning time and performance improvement due to an increase in the number of particles. In [4], they proposed an algorithm that parallelly performs MCTS. In [5], they proposed an algorithm that operates at a real-time level in combination with this algorithm and a physical simulator that is advantageous for parallel processing. If we can also adapt these methods to tree search in parallel, we expect to effectively reduce the planning time of our algorithm.

B. Implementation details for training

B.1. Architecture of networks

Backbone transformer

- Light-dark room domain
 - Number of layers: 3
 - Number of attention heads: 1
 - Embedding dimension: 128
 - Input encoding
 - * Observation input dimension: 2
 - * Action input dimension: 2
 - Nonlinearity function: GeLU
 - Sequence length: 61

- 46 • Fetching domains
- 47 – Number of layers: 3
- 48 – Number of attention heads: 2
- 49 – Embedding dimension: 256
- 50 – Input encoding
- 51 * Observation input dimension:
- 52 · RGB-D: 64×4
- 53 · Grasp detection identifier: 1
- 54 * Image encoder:
- 55 · Number of convolutional layers: 4
- 56 · Number of max pool layers: 4
- 57 · Kernel size: 3
- 58 · Stride: 1
- 59 · Dimension of feature maps: (3, 16, 32, 64, 128)
- 60 * Action input dimension: 8
- 61 – Nonlinearity function: GeLU
- 62 – Sequence length: 13

63 Header of value networks

- 64 • Light-dark domain
- 65 – Input dimension: 128
- 66 – Output dimension: 1
- 67 – Number of layers: 1
- 68 • Fetching domains
- 69 – Input dimension: 256
- 70 – Output dimension: 1
- 71 – Number of layers: 2
- 72 – Hidden dimension: 256

73 Header of policy network

- 74 • Light-dark domain
- 75 – Input dimension: 2
- 76 – Output dimension: 2
- 77 – Condition dimension: 128
- 78 – Latent dimension: 64
- 79 – Number of encoder layers: 3
- 80 – Number of decoder layers: 3
- 81 – Beta
- 82 * Light-dark
- 83 · Imitation: 0.25
- 84 · Preference: 0.5
- 85 * Fetching domains
- 86 · Imitation: 0.25
- 87 · Preference: 0.5
- 88 • Fetching domain
- 89 – Input dimension: 3

- 90 – Output dimension: 3
- 91 – Condition dimension: 256
- 92 – Latent dimension: 64
- 93 – Number of encoder layers: 4
- 94 – Number of decoder layers: 4
- 95 – Beta
 - 96 * Light-dark
 - 97 · Imitation: 0.25
 - 98 · Preference: 0.5
 - 99 * Fetching domains
 - 100 · Imitation: 0.25
 - 101 · Preference: 0.5

102 B.2. Hyperparameters

- 103 • Light-dark domain
 - 104 – Batch size
 - 105 * Value
 - 106 · 128 for the dataset with 10 search trees
 - 107 · 512 for the dataset with 50 search trees
 - 108 · 2048 for the dataset with 100 or larger number of search trees.
 - 109 * Policy
 - 110 · 64
 - 111 – Learning rate: 1e-4 (1e-5 for dataset with 10 search trees)
 - 112 – Learning rate scheduler: None
 - 113 – Optimizer: AdamW
 - 114 – Weight decay: 1e-5
 - 115 – Dropout: 0.1
- 116 • Fetching domains
 - 117 – Batch size
 - 118 * Value
 - 119 · 128 for the dataset with 10 search trees
 - 120 · 512 for the dataset with 50 search trees
 - 121 · 2048 for the dataset with 100 or larger number of search trees.
 - 122 * Policy
 - 123 · 64
 - 124 – Learning rate: 0.0001 (0.00001 for dataset with 10 search trees)
 - 125 – Learning rate scheduler: Multi-step scheduler with gamma = 0.1 and milestone=4000
 - 126 – Optimizer: Adam
 - 127 – Dropout: 0.1

128 C. Guiding POMCPOW with the learned functions

129 Using \tilde{V} , \tilde{Q} , and $\hat{\pi}_\theta$, we guide POMCPOW in order to speed up planning. Algorithm 1 shows the
 130 pseudocode for the guided POMCPOW. The guided search takes in the trained value function and
 131 policy, initial belief, the total number of simulations n , and the maximum planning horizon T . The
 132 variables used throughout the algorithms are: an execution history $h_t = (o_0, a_1, o_1, \dots, a_t, o_t)$, a list
 133 of child nodes C , a number of visits to a node N , a number of times that a given observation node
 134 has been generated M , and B and W are the list of the list of belief states and the weight associated
 135 with it respectively. C, N, M, B, W are implicitly initialized to \emptyset or 0.

136 Procedure GUIDEDSEARCH in Algorithm 1 describes the overall process of guided POMCPOW.
 137 The procedure takes initial belief $b_0(s)$, \tilde{V} , $\hat{\pi}_\theta$, n , T . For each iteration, we sample a particle s
 138 from $b_0(s)$, then run SIMULATE procedure. iterate through the particles by simulating with each
 139 of them, and outputs action a with the highest Q value backed up by the simulation.

140 The new action is sampled with a procedure ACTIONPROGWIDEN similar to the one proposed in
 141 POMCPOW[6]. However, to efficiently sample an action rather than resorting to a random policy,
 142 we use $\hat{\pi}_\theta$ to produce a new action sample (9). Then, an action is selected according to UCB1 (11).

143 The SIMULATE is a recursive function that terminates when it reaches the maximum search depth
 144 of 0. Otherwise, it samples an action with ACTIONPROGWIDEN, expands observation node (line 6
 145 to 11) and action node (line 12 to 18). However, guided POMCPOW differs in that it leverages \tilde{V}
 146 instead of rollout by random policy (line 14) to reduce planning time.

147 There are several hyperparameters for POMCPOW: progressive widening constants $(k_a\alpha_a, k_o, \alpha_o)$
 148 and exploration constant c for UCB. We set the progressive widening parameter as $(k_a\alpha_a, k_o, \alpha_o) =$
 149 $(0.5, 0.5, 0.5, 0.5)$ for 2D light-dark room domain and $(k_a\alpha_a, k_o, \alpha_o) = (3.0, 0.15, 3.0, 0.15)$ for
 150 two object fetching domains. Since the value outputs of each method in each domain have different
 151 scales, we set the exploration constant c differently for each method. In 2D light-dark room domain,
 152 we set it as 20 for PGP and SF-PGP, and 50 for IGP and unguided search. In object fetching domain
 153 with known object class, we set it as 0.5 for PGP and SF-PGP, and 100 for IGP and unguided search.
 154 In object fetching domain with unknown object class, we set it as 20 for PGP and SF-PGP, and 200
 155 for IGP and unguided search.

Algorithm 1 Guided POMCPOW

<pre> 1: procedure GUIDEDSEARCH($b_0(s), \tilde{V}, \hat{\pi}_\theta, n, T$) 2: $Q \leftarrow -\infty, h \leftarrow \emptyset$ 3: for $i \leftarrow 0$ to n do 4: $s \sim b_0(s)$ 5: SIMULATE($s, h, T, \hat{\pi}_\theta, \tilde{V}$) 6: return $\underset{a}{\operatorname{argmax}} Q(ba)$ 7: procedure ACTIONPROGWIDEN(h) 8: if $C(h) \leq k_a N(h)^{\alpha_a}$ then 9: $a \sim \hat{\pi}_\theta(h)$ 10: $C(h) \leftarrow C(h) \cup \{a\}$ 11: return $\underset{a \in C(h)}{\operatorname{argmax}} Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$ </pre>	<pre> 1: procedure SIMULATE($s, h, d, \hat{\pi}_\theta, \tilde{V}$) 2: if $d = 0$ then 3: return 0 4: $a \leftarrow \text{ACTIONPROGWIDEN}(h, \hat{\pi}_\theta)$ 5: $s', o, r \leftarrow G(s, a)$ 6: if $C(h) \leq k_a N(h)^{\alpha_a}$ then 7: $M(hao) \leftarrow M(hao) + 1$ 8: else 9: $o \leftarrow \text{select } o \in C(ha) \text{ w.p. } \frac{M(hao)}{\sum_o M(hao)}$ 10: append s' to $B(hao)$ 11: append $\mathcal{Z}(o s, a, s')$ to $W(hao)$ 12: if $o \notin C(ha)$ then 13: $C(ha) \leftarrow C(ha) \cup \{o\}$ 14: $total \leftarrow r + \gamma \tilde{V}(hao)$ 15: else 16: $s' \leftarrow \text{select } B(hao)[i] \text{ w.p. } \frac{W(hao)[i]}{\sum_{j=1}^m W(hao)[j]}$ 17: $r \leftarrow R(s, a, s')$ 18: $total \leftarrow r + \gamma \text{SIMULATE}(s', hao, d - 1)$ 19: $N(h) \leftarrow N(h) + 1$ 20: $N(ha) \leftarrow N(ha) + 1$ 21: $Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$ 22: return $total$ </pre>
---	--

156 **D. Experiment setup for fetching domains**

157 To ensure that each pick or place actions are in contact with the object, we use rejection sampling.
 158 To sample PICK, we first choose a particle from a belief state, and for the given object, we choose

a point on the upper surface of the object as a contact point. Then, we check two things: the *cosine similarity* between the normal vector at the contact point and the z-axis must be within 0.95, and the entire suction cup must be enclosed inside the object when we project it down along the z-axis. If these are not met, we sample another PICK and repeat the procedure. For PICK that passed the tests, we check the existence of an IK solution and collision-free motion plan using biRRT [7], and if they do not exist, we sample another PICK and restart from the beginning. PLACE simply samples a pose that is either inside the cabinet or the goal region.

Perception System

In real-world object fetching with unknown object classes domain, we use Mask R-CNN [1] with dropout sampling [2] to model the class distribution of an object, $p(c|o)$. For efficient data procurement, We utilize the Bullet physics simulator [8] to collect RGB images and mask annotation. Nonetheless, the data from the simulation, especially the RGB image, does not generalize to the real-world scene due to the sim2real gap. To overcome this, we render the image with NVISII [9] to create photo-realistic texture while using the mask annotation from Bullet simulator and train Mask R-CNN with these images.

To estimate $p(q|c, o)$, we train a pose estimator per each class. It is a PointNet [3]-based architecture with 2 heads of MLP layers for orientation and position. We bin the orientation into 2048 classes and train the orientation head to predict the unit quaternion closest to the ground truth orientation while the position head is trained with regression. During an inference, we decide which pose estimator to use according to the class label predicted by Mask R-CNN. Then, we feed the partial point cloud obtained from the depth image and the segmentation result and get $p(q|c, o)$. To generate the data, we randomly sample about 130k positions on the cabinet and yaw angle (roll and pitch angles of the objects are fixed to 0). Then, we load random objects to the Bullet simulator with the sampled pose, capture the partial point cloud using the depth camera, and label them with corresponding object poses given by the simulator.

Initial belief generator

In real-world object fetching with unknown object classes domain, the initial belief generator uses the result from the perception system to sample initial belief over the state $p(s|o)$. For class uncertainty, we sample N_{shape} number of class labels from the class distribution output by Mask R-CNN. Then, for each class, we run our pose estimator to sample N_{pose} number of poses. There are total $N_{shape} \times N_{pose}$ candidates for beliefs, and we randomly choose $N_{particle}$ particles for the initial belief with rejection sampling. The criteria for rejection is the existence of collision between objects or cabinet, grasp affordance of the vacuum gripper, and the misclassification of target(red) objects (i.e. hammer, drill, driver) to the non-target object class label (i.e. box) or vice-versa.

E. Evaluation

Qualitative results for object fetching with known object classes

Figure 1 (a) shows the visualization of the outputs of value networks, which are normalized between 0 and 1, for PLACE action on the same scene. After the robot picks the non-target object, to achieve information-gathering actions, the robot should prefer the region where the target object can be visible, such as the side of the area, except the area where the target object can be invisible. Our approach can tell the clear difference between such areas, whereas other approaches do not show clear differences, or even show the same value across all areas. (b) and (c) from Figure 1 shows the action samples when PICK on both target and non-target object. When a target object is picked, the robot should prefer the action samples around the goal region. Here, our model shows a high preference in that area. When the non-target object is picked, the robot should prefer the action samples where the target object can be visible, and also not the goal area. Our approach avoids around the target object, whereas others take action samples around the target object and goal area.

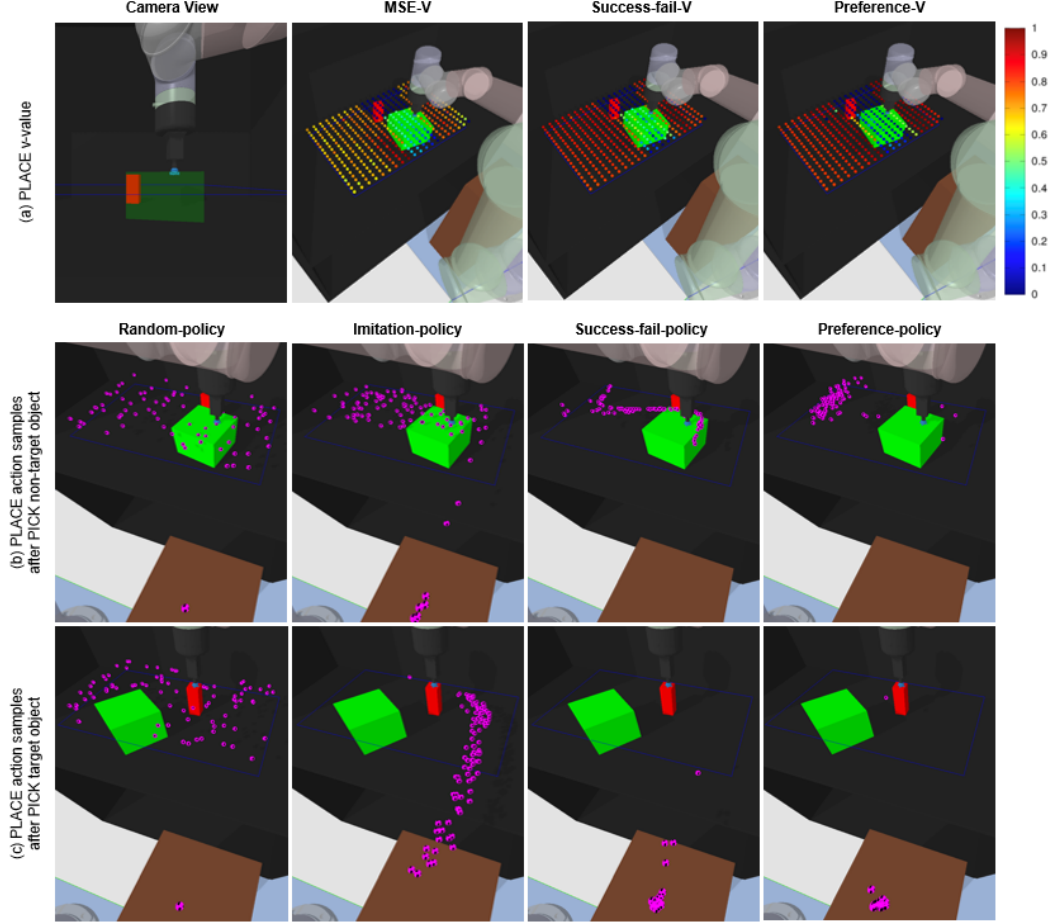
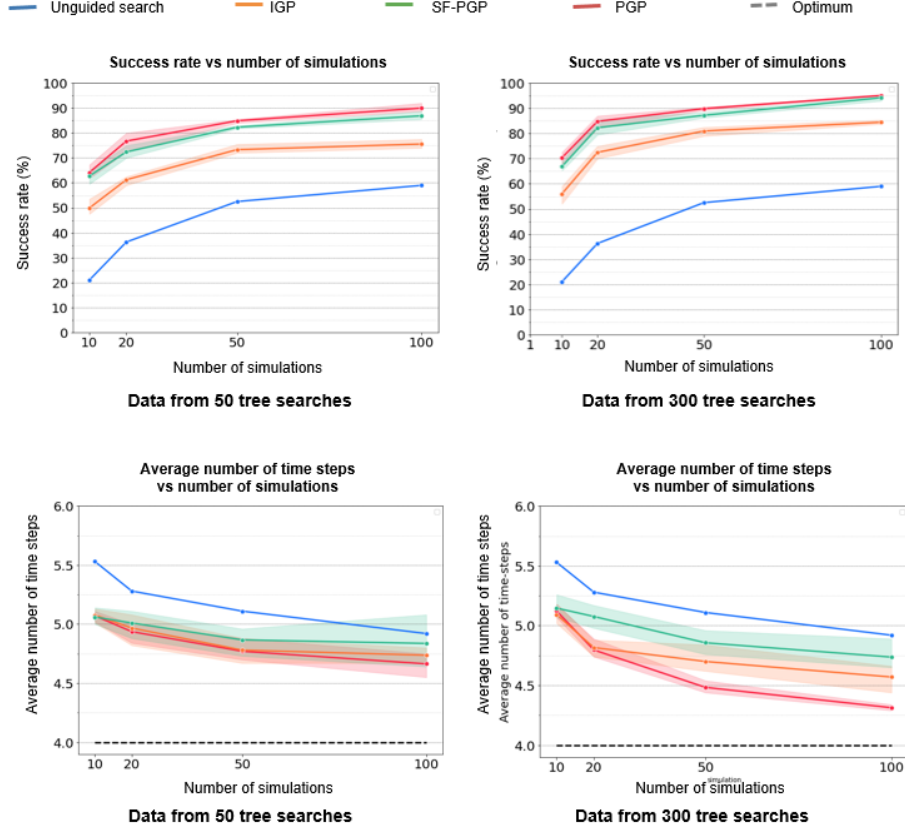


Figure 1: **Qualitative comparisons on object fetching with known object classes.** (a) shows normalized V-values of PLACE actions obtained from 3 different value networks (b) purple dots show 100 PLACE action samples from the trained policy after PICK non-target object. (c) purple dots show 100 PLACE action samples from the trained policy after PICK target object when the target object is visible. The networks used in these figures were trained using data from 50 search trees.

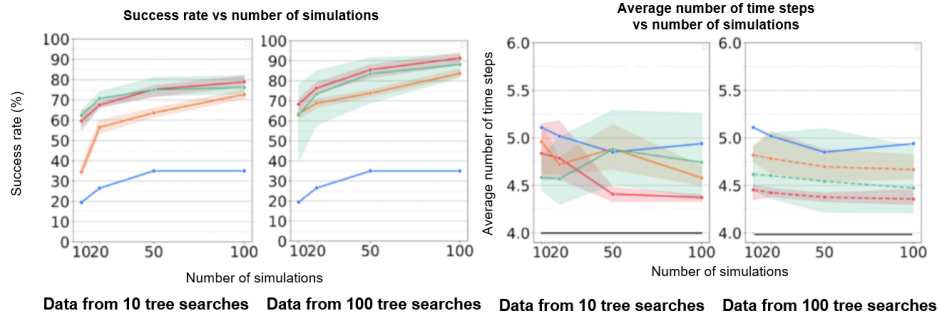
206 Quantitative results of fetching domains with different sizes of data

207 We compared the results of a guided search using networks trained in various data sizes to evaluate
 208 data efficiency. Figure 2 (a) shows the result of object fetching with known object classes domain,
 209 using data from 50 and 300 search trees in addition to the results shown in the main document.

210 The first row shows that the results are consistent with our hypotheses. With 50 and 300 tree
 211 searches, SF-PGP and PGP show higher success rates than IGP. The second row shows that
 212 PGP achieves a more optimal solution than other approaches. With 300 tree searches, PGP finds
 213 near-optimal plans, while other approaches struggle to do so. IGP comes close, but note that it has
 214 a lower success rate than PGP. Figure 2 (b) shows the result of real-world object fetching with
 215 unknown object classes domain evaluated in PyBullet [8] simulation, using data from 10 and 100
 216 search trees. The two columns on the left show that the results are consistent with our hypotheses
 217 again. With 10 and 100 tree searches, SF-PGP and PGP again show higher success rates than IGP.
 218 The two columns on the right show that PGP again finds the most optimal plans.



(a) **Quantitative results of object fetching with known object classes.** Data from 50 search trees and 300 search trees are compared. The first row shows the success rates of guided search using networks learned from data obtained from the different number of search trees. The second row shows on the right show the average time-step of success trajectories obtained from these searches.



(b) **Quantitative results of real-world object fetching with unknown object classes in PyBullet [8] simulation.** Data from 10 search trees and 100 search trees are compared. The two columns on the left show the success rates of guided search using networks learned from data obtained from the different number of search trees. The two columns on the right show the average time-step of success trajectories obtained from these searches.

Figure 2: **Quantitative comparisons on object fetching domains** All plots depict the mean and 95% confidence intervals (CIs) based on 400 experiments conducted with trained networks using three different random seeds.

References

- [1] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [2] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ArXiv*, abs/1506.02142, 2015.
- [3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.
- [4] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik. Parallel monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 60–71. Springer, 2008.
- [5] B. Huang, A. Boularias, and J. Yu. Parallel monte carlo tree search with batched rigid-body simulations for speeding up long-horizon episodic robot planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1153–1160. IEEE, 2022.
- [6] Z. Sunberg and M. Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 259–263, 2018.
- [7] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.
- [8] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2021.
- [9] N. Morrical, J. Tremblay, Y. Lin, S. Tyree, S. Birchfield, V. Pascucci, and I. Wald. Nvisii: A scriptable tool for photorealistic image generation, 2021.