Where To Start?
Transferring Simple Skills to Complex Environments
–Supplementary Material–

Vitalis Vosylius and Edward Johns
The Robot Learning Lab
Imperial College London
United Kingdom
{vitalis.vosylius19, e.johns}@imperial.ac.uk

1 Behaviour Cloning Skills

In our experiments, we test our method on two robotic skills: 1) grasping novel objects and 2) placing a held object inside novel containers. However, our approach is not limited to these types of tasks. We acquired these skills using Behavioural Cloning (BC) solely in simulated environments. Our learnt skills operate in 6D end-effector action space using segmented point clouds as observations.

1.1 Training Data

We create a set of expert demonstrations ($D_{BC} = \{(\mathcal{P}^0_{target}, \mathcal{T}^0), (\mathcal{P}^1_{target}, \mathcal{T}^1), ..., (\mathcal{P}^T_{target}, \mathcal{T}^T)\}^N_1$) using a simulated environment and scripted policies that use privileged information such as object geometries and their poses. Here $\mathcal{P}^t_{target}$ and $\mathcal{T}^t$ represent the point cloud of the target object and end-effector $\mathbb{SE}(3)$ pose, respectively.

To generalise across novel objects for grasping, we use 40 different categories of objects from ShapeNet [1], and their annotated grasps from the ACRONYM [2] dataset [1]. In total, we use 200 different objects. For the placing policy, we use different types of containers from ShapeNet [1] (Bowls, Pans, Trash Bins, Baskets etc.) totalling 200 different objects. For both policies, we create 1000 expert demonstrations from close proximity to the target. All expert demonstrations are create in a tabletop environment without any obstacles present. For a better chance of a sim2real transfer, we add noise to the depth images according to [3], use noisy camera extrinsic and imperfect segmentations.

1.2 Training

We train Behaviour Cloning policies that map the point cloud of the object represented in the end-effector frame to the relative transformation between subsequent $\mathbb{SE}(3)$ end-effector frames in the expert trajectory ($a^*_{eef} = \mathcal{T}_t^{-1}\mathcal{T}_{t+1}$) and a binary action for the gripper a^*_{grip}. To train the BC policies, we use a combination of the Point Matching loss defined in Equation 1 [4], that jointly optimises translation and orientation of the 6D pose and a Binary Cross-Entropy loss for predicting the action for the gripper.

$$Loss_{pose}(\mathcal{T}_1, \mathcal{T}_2) = \frac{1}{|X_g|} \sum_{x \in X_g} ||\mathcal{T}_1(x) - \mathcal{T}_2(x)||_1 \tag{1}$$

Here, X_g is a set of points defined on the gripper, $\mathcal{T}_1$, $\mathcal{T}_2 \in \mathbb{SE}(3)$. Our network outputs the 3D translation and 6D representation of rotation that we use to create homogeneous transformations in

[1] We adjust and re-validate all the used grasp poses through the physics simulation due to the use of a different gripper from the one used to create the ACRONYM [2] dataset.

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

a differentiable manner. We then use these transformations in Equation 1 to calculate the $Loss_{pose}$. The complete loss function used to train BC skills is

$$Loss_{BC} = \alpha Loss_{pose}(a_{eef}, a^*_{eef}) + \beta BCE(a_{grip}, a^*_{grip}) \tag{2}$$

Here we use α and β to scale different parts of the loss function in order to keep the losses of comparable magnitude for each part of the action. In our experiments, we set α and β to 1 and $1e^{-3}$, respectively. In addition, because a^*_{eef} is only equal to 1 at the end of an expert trajectory, we weigh positive examples in the BCE loss part with a factor of 10. All training was done using a single machine with AMD Ryzen 9 5900X CPU and NVIDIA GeForce RTX 3080ti GPU and took approximately 2 hours per skill.

During deployment, the a_{eef} outputted by the BC skill is time-scaled to obtain end-effector velocities that are applied to the robot at a 30Hz rate. When the skill predicts a_{grip}, the closed-loop skill execution is stopped, and the gripper closes or opens depending on the skill (grasping or placing).

1.3 Architectures

We train separate models for the two skills we are considering. However, the network architecture for both skills is exactly the same. Our policy network is composed of the PointNet++ [5] followed by an LSTM cell. PointNet++ is composed of two Set Abstraction layers followed by a global PointNet layer compressing the point cloud observation into a 512-dimensional vector. This vector is then passed to an LSTM cell with a hidden dimension of 128. Finally, separate linear layers predict 3D translation and 6D representation of rotation. The total size of the model is around 2M trainable parameters. We use Adam [6] as the optimiser for training and keep a constant learning rate throughout the training at $1e^{-4}$. We experimented with different learning rates and their schedulers but found no significant improvement.

2 Graph-based Affordance Model

2.1 Architecture

The network architecture used to learn the affordance model is composed of PointNet++ Set Abstraction (SA) layers [5] for coarsening the point clouds and a heterogeneous graph neural network.

We use separate SA layers to process the point clouds of the target object and the surrounding obstacles. For both, we use 2 SA layers that produce a set of points with 256-dimensional vectors describing their local neighbourhood. In total, point clouds are coarsened to $\sim 3\%$ of their original size. We represent the robot configuration in Cartesian space as a set of positions of the centre of mass of different links of the robot and one-hot encoding specifying which link it represents. In our experiments, we use eight robot key points. We use these feature vectors and the ones obtained by coarsening the point clouds to construct a fully connected heterogeneous graph. We represent edge features as 3-dimensional vectors - relative translations in XYZ between the nodes.

For our model's graph neural network part, we use two layers of GATv2 operations [7]. We transform a homogeneous version of our graph neural network into a heterogeneous version using the Pytorch-Geometric Deep Learning Library [8]. For GATv2 operations, we use a single attention head and encode each node in the heterogeneous graph into a 256-dimensional vector. A global vector describing the graph is then obtained using a global attention pooling layer [9] and passed through a Sigmoid activation to get an affordance score estimate. The whole network is trained end-to-end. In total, our model has around 3.5M trainable parameters.

2.2 Training

We train the affordance model using a dataset composed of positive and negative samples created by rolling out the learnt skills from the different configurations and recording the outcome. Skills rollouts are done in different randomised versions of the environment, such that the wrist camera is pointing toward the object, and the robot is not colliding with the environment. In addition, we add extra negative examples to the dataset where the wrist camera is pointing away from the object, knowing that the skill can not succeed without seeing the object. This could be circumvented by

adding additional constrain that the target object needs to be in the field of view of the camera to the optimisation problem at inference.

We use the Binary Cross-Entropy loss function, and Adam [6] as the optimiser for training, keeping a constant learning rate throughout the training at $1e^{-4}$. We experimented with different learning rates and their schedulers but found no significant improvement.

All training was done using a single machine with AMD Ryzen 9 5900X CPU and NVIDIA GeForce RTX 3080ti GPU and took approximately 1.5 hours per affordance model.

2.3 Deployment

During deployment, we want to maximise the predicted affordance score with respect to the robot's configurations (joint angles). We do this using gradient-based optimisation. Because we add additional, non-linear constraints, this constitutes solving constrained, non-convex optimisation problem. In practice, instead of maximising the predicted affordance score, we minimise the negative logits of our model.

Through the optimisation, the observation (segmented point cloud) does not change. Therefore, we only need to create a heterogeneous graph once and only update the edge features representing relative positions between the nodes. This can be efficiently done using Forward Kinematics.

We start the optimisation with a robot configuration that does not collide with the environment, and the wrist camera points towards the target object. We find this configuration using the same procedure used to generate the dataset for training the affordance model, resulting in initialisation that is in distribution for the learnt affordance model. We run gradient-based optimisation from 3 different configurations to avoid local minima and use the one with the highest affordance score, ensuring that the target object can be seen by the wrist camera.

Note that this approach could be easily extended to a full trajectory optimisation framework without needing an additional sampling-based motion planner. However, we found that it is unnecessary and can even hinder performance due to a higher susceptibility to local minima.

3 Simulated Environments

To create a dataset of rollouts and evaluate our method, we procedurally generate a set of simulated environments with varying amounts of clutter. We do so by placing a target object on the tabletop, selecting a number of obstacles in the scene and placing them, ensuring there are no collisions. We randomise all the poses of objects and the number of obstacles for each generated environment. We use a mixture of primitive shapes and random objects from ShapeNet as obstacles. To accurately represent collision geometries, we use convex-decomposition of watertight meshes of ShapeNet objects before adding them to the environment. For environments 3-6, we construct a structure with random dimensions and orientation, ensuring the target is inside. Examples of the randomised environments can be seen in Figure 1.

Finally, we validate that it is possible to complete the considered task in generated environments using the same oracle used to create demonstrations for Behaviour Cloning Skills. The oracle is designed to provide demonstrations in an open and unconstrained environment and does not take obstacles into account. Therefore, if it can complete the task without any collisions, a converged policy trained on demonstrations obtained from it should also be able to do it.

4 Experimental Results

Here we present the same results as in the main paper but in the form of graphs to better showcase the underlying trends. In addition, we further discuss the baselines used in our experiments and the reasons for choosing them.

4.1 Comparison Against Learning a Single End-to-End Policy

In our experiments, we use skills for grasping and placing acquired using Behaviour Cloning. Therefore, for a fair comparison, our single policy baselines trained end-to-end in cluttered environments

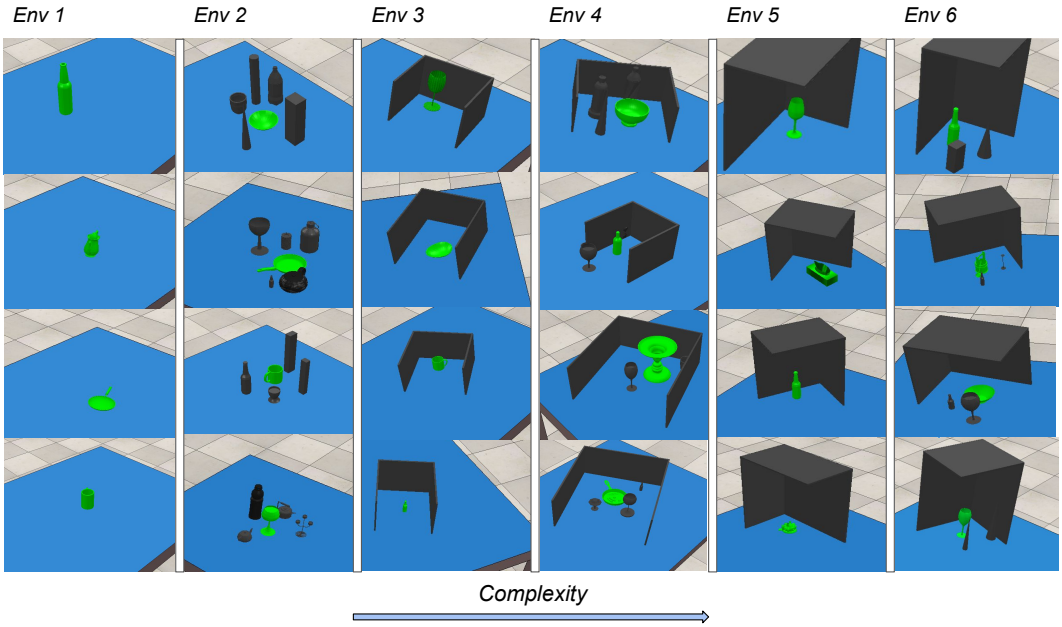

Figure 1: Examples of randomised evaluation environments used in our simulation experiments. Inherently different structures introduce a different complexity in each type of environment. Target object and obstacles are depicted in green and black, respectively.

are also acquired using Behaviour Cloning. We did not include additional baselines that main to learn manipulation skills in constraint environments in our evaluation as it would not have allowed for direct comparison with using *a priori* acquired skill in unconstrained scenes.

These baselines do not use *a priori* acquired skills but rather control the robot for the entire trajectory, from the initial configuration to the interaction with the object. This requires learnt policies to reason about both avoiding obstacles and completing the task. On the other hand, our approach has two networks: one for finding a suitable starting configuration for a skill and one for executing that skill without avoiding obstacles.

We compare our approach against two BC policies that use different action spaces (end-effector velocities and joint velocities). We do this because in some environments, controlling the entire configuration of the robot is necessary to avoid collisions, but completing the task is more straightforward when considering only the end-effector. Results for this set of experiments in a graph form are shown in Figure 2.

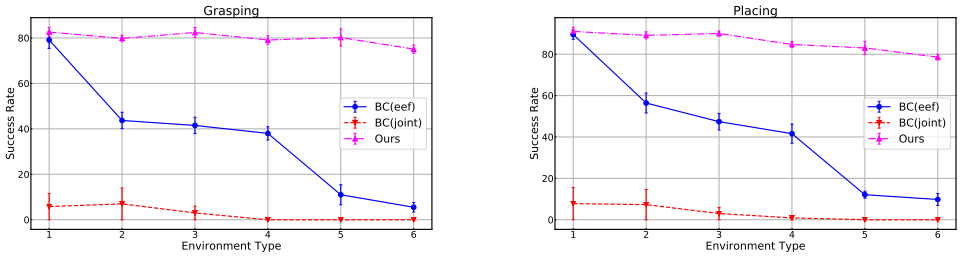

Figure 2: Success rates of our method and single end-to-end BC policies trained in cluttered environments. The curves represent the mean of the success rates averaged over 6 experiments, each with a different random seed. Error bars represent the standard deviation.

We can see that the more complex the environment the higher the performance gains of our method over learning a single end-to-end policy. This is mainly because end-to-end policies need to generalise across the whole workspace and learn how to complete the task, navigate free space and avoid

obstacles. By using *a priori* acquired skills that focus on completing the task and a motion planner to navigate the free space and reach the predicted starting configuration, the search space is drastically reduced which results in a better performance.

4.2 Different Ways of Finding Starting Configurations

Our objective is to deploy skills acquired in unconstrained environments directly in constrained ones without the need to re-learn them, which can be extremely inefficient in environments where small adjustments to the original skills are not enough. Therefore, we did not include baselines that try to adjust already acquired skills based on the new environment. Instead, we devised several different baselines to compare our method against. These baselines also reach a specific configuration and execute the *a priori* acquired skill.

The *naive* baseline does not consider the obstacles in the environment, only ensuring that the camera is pointing towards the target object and the robot is not colliding with the obstacles at the start. It aims to validate the need to learn the starting configurations for *a priori* acquired skills.

The *generative* baseline directly regresses joint angles from which the skill should be executed. It allows us to validate our approach to finding these starting configurations. The distribution of suitable starting configurations is inherently multimodal. Therefore, for a fair comparison, we train a Conditional Variational Auto Encoder (CVAE) [10] capable of capturing such distributions.

Finally, the *BC2* baselines jointly learn the distribution of suitable starting configurations and control policy on how to reach them. This can be seen as a more complex task and allows us to validate the added benefit of the second step in our framework - reaching a starting configuration using a motion planner. Again, we use two different BC policies using different action spaces for the same reasons as in Section 4.1. Results for this set of experiments in a graph form are shown in Figure 3.

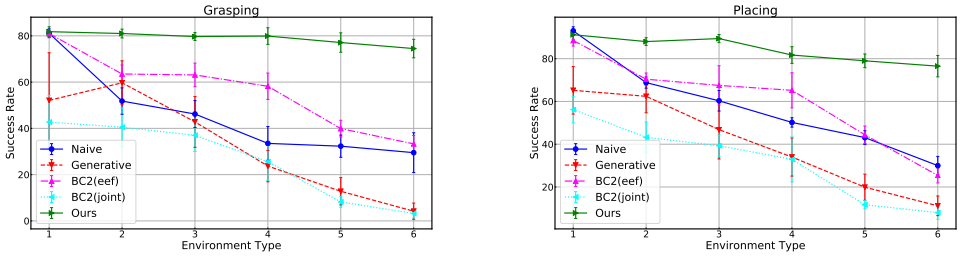

Figure 3: Success rates of our method and other considered baselines. The curves represent the mean of the success rates averaged over 6 experiments, each with a different random seed. Error bars represent the standard deviation.

4.3 Ablations

Results for our ablation set of experiments in a graph form are shown in Figure 4.

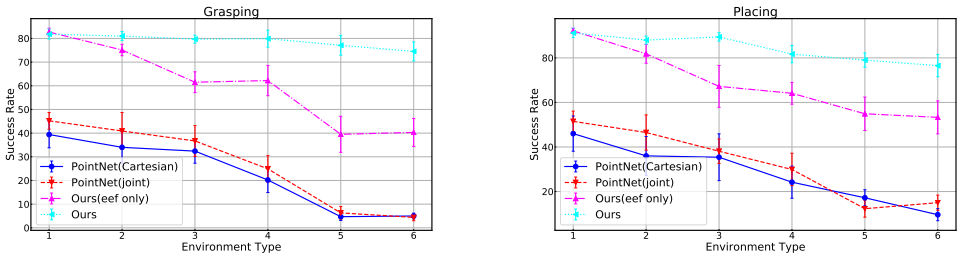

Figure 4: Success rates of our method when using different types of architecture to learn the affordance model. The curves represent the mean of the success rates averaged over 6 experiments, each with a different random seed. Error bars represent the standard deviation.

References

[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[2] C. Eppner, A. Mousavian, and D. Fox. Acronym: A large-scale grasp dataset based on simulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6222–6227. IEEE, 2021.

[3] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong. Analysis and noise modeling of the intel realsense d435 for mobile robots. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 707–711. IEEE, 2019.

[4] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *European Conference on Computer Vision*, pages 574–591. Springer, 2020.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[8] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[9] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[10] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.