Robust Policies via Mid-Level Visual Representations:
An Experimental Study in Manipulation and Navigation

Bryan Chen[1*] **Alexander Sax**[1*] **Francis E. Lewis**[2]
Silvio Savarese[2] **Amir Zamir**[3] **Jitendra Malik**[1] **Lerrel Pinto**[1,4]

[1] University of California, Berkeley [2] Stanford University [4] New York University
[3] Swiss Federal Institute of Technology Lausanne (EPFL)

http://midlevel.berkeley.edu

Abstract: Vision-based robotics often separates the control loop into one module for perception and a separate module for control. It is possible to train the whole system end-to-end (e.g. with deep RL), but doing it "from scratch" comes with a high sample complexity cost and the final result is often brittle, failing unexpectedly if the test environment differs from that of training.

We study the effects of using **mid-level visual representations** (features learned asynchronously for traditional computer vision objectives), as a *generic* and *easy-to-decode* perceptual state in an end-to-end RL framework. Mid-level representations encode invariances about the world, and we show that they aid generalization, improve sample complexity, and lead to a higher final performance. Compared to other approaches for incorporating invariances, such as domain randomization, asynchronously trained mid-level representations scale better: both to harder problems and to larger domain shifts. In practice, this means that mid-level representations could be used to successfully train policies for tasks where domain randomization and learning-from-scratch failed. We report results on both manipulation and navigation tasks, and for navigation include zero-shot sim-to-real experiments on real robots.

Keywords: Representation Learning, Mid-Level Representations, Generalization, Transfer Learning, Vision, Reinforcement Learning.

1 Introduction

Over the past few years, impressive success stories such as [1, 2] have helped deep reinforcement learning (deep RL) make inroads into various fields. Deep RL from pixels, in particular, has drawn attention [3, 4, 5] as a unified method for training agents, but it requires that agents can access virtually unlimited data covering every possible input.

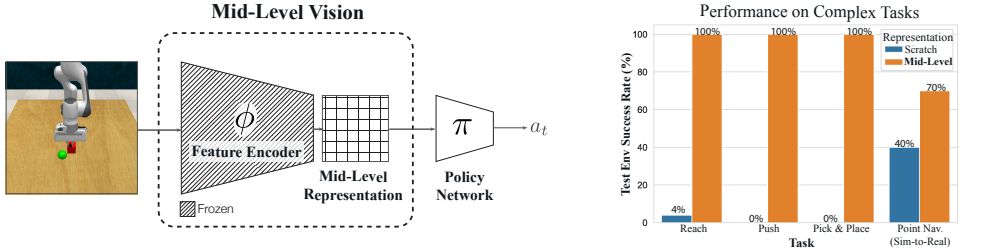

Figure 1: **Mid-level visual representations used for RL. Left:** A feature encoder trained for some mid-level objective provides representations to the agent. **Right:** Agents trained using these mid-level representations were able to generalize, without additional adaptation, to distribution shifts and deployment on physical robots.

In scenarios where agents can access large, but still finite amounts of data, the deep-RL-from-pixels approach proves hard to train, however, and the resulting policies are brittle in the real world [6].

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

Agents trained this way fail, sometimes spectacularly, under mild visual shifts relative to the training environment. For example, simply placing a water bottle in view of the robot [2], or operating in morning light instead of midday sun [7, 8] can be sufficient to completely degrade performance.

This brittleness reflects the fact that the policies have not captured the invariances (and equivariances) necessary to generalize and operate in the real world. The policies do not learn these invariances because there is usually no reason for them to do so. Biased or insufficient training data might permit spurious "cheating" shortcuts or make it easier to memorize features of the training environment rather than learn how to extract those features from the vast space of possible inputs. In any case, agents end up without the right *priors* about the world.

Computer vision provides us with tools that parse complex visual scenes and extract usable perceptual representations. In this paper, we study some of those representations used as a form of mid-level vision. That is, instead of training directly on raw pixels, we first extract representations driven by traditional computer vision objectives and use those as the input observations to RL instead (see Fig. 2-left). The networks which extract the mid-level visual representations are *asynchronously trained*, meaning that they can be trained independently and on a schedule different from the RL training. This approach has shown promise [9, 10, 11], especially in navigation contexts where agents using mid-level vision are able to generalize to unseen (simulated) buildings [9].

The main contribution of our study is experimental. We show that this approach scales to harder tasks than previously shown, even when control is nontrivial (e.g. manipulation with continuous control), or there are drastic domain shifts such as training in simulation and then deploying on physical robots with no additional training. Specifically, we test the following hypotheses:

a Do mid-level visual representations provide a useful way to incorporate invariances for "hard" tasks, when compared to training from scratch? (Answer: **Yes**)

b Do the representations simplify the learning problem (opening up the possibility to successfully train on harder problems that fail otherwise)? (Answer: **Yes**)

c Do the representations improve robustness to distribution shifts? (Answer: **Yes**: both sim-to-real and within simulation)

d How does the mid-level approach compare to other approaches for incorporating invariances? (Answer: **It scales significantly better**)

In summary, we present a large-scale study evaluating the effect of using invariances learned from computer vision objectives plugged into active RL frameworks. We find that the mid-level approach actually performs even better in the harder contexts, relative to alternatives, and provides an avenue for solving harder problems. We analyze this behavior in terms of training performance, generalization performance, and sample complexity. The mid-level approach was able to achieve a 100% success rate in certain test environments when alternatives approaches like learning-from-scratch, using *ground-truth low-dimensional state*, and domain randomization do not learn at all (0% train and 0% test, even after multiple hyperparameter sweeps).The mid-level approach trains faster than the alternatives, almost as fast as using ground truth low-dimensional state (when state succeeds). Further, we compare mid-level representations to other methods of learning invariances (domain randomization), and show that as the tasks become more difficult, domain randomization makes learning the task harder (100% train success $\rightarrow$ 70% with domain randomization, $4\% \rightarrow 20\%$ test) while mid-level simplifies it (near-perfect performance on the both train and test domains).

2 Related Works

Our study is connected to a range of relevant fields and we review the most important ones, within space constraints.

Computer Vision. Computer vision approaches are typically designed to solve stand-alone vision objectives such as depth estimation [12], object classification [13], detection [14], segmentation [15], pose estimation [16, 17]. While approaches may use various levels of supervision [13, 18, 19, 20, 21], the common characteristic across conventional computer vision methods is that they are trained and evaluated on static datasets collected for that stand-alone objective. In contrast, the perception of active agents is fundamentally in service of some downstream goal, and agents are evaluated on that goal in an *online* manner so that a single decision early on in an episode impacts the observations that will follow. In this paper we study how conventional computer vision objectives impact these

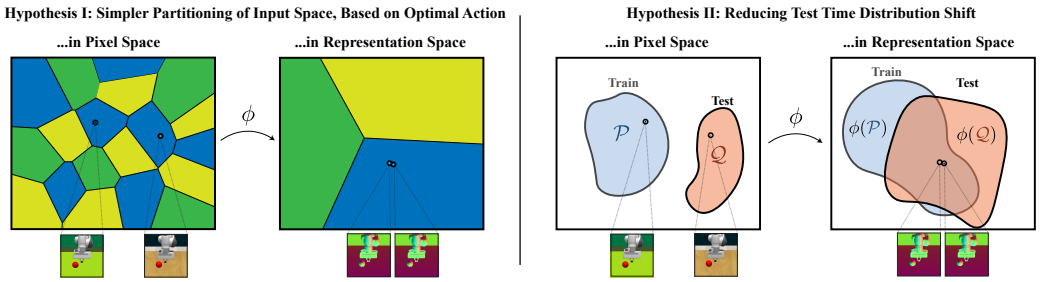

Figure 2: **Mid-level representations transform pixel inputs.** There are two major ways that invariances from mid-level representations could be useful for downstream tasks. **(I) Left:** The invariances simplify the decision boundaries for downstream tasks. In this case, we would expect (i) training on representations to be faster than training on pixels and (ii) to allow us to train agents for more difficult tasks. **(II) Right:** The invariances in representation space align the train and test distributions. In this case we would expect generalization performance to improve relative to training on pixels. In practice, we see behavior consistent with both hypotheses.

downstream tasks, especially when the downstream tasks have strong temporal dependencies and domains different than those used for training the conventional vision approaches.

Representation/Feature Learning. The goal of representation learning is to encode observations in a way that provides benefits over using raw sensor data. One of the most popular approaches is based on Minimum Description Length (MDL) which suggests good representations are those which most compactly describe the data. This includes variational autoencoders [22] and alternatives [23, 24, 25] and has been applied to robotics (e.g. [26]). However, MDL representations tend to be exceedingly sensitive to tiny domain shifts [9], an obervation that we also make. Likelihood-based approaches often try to predict the probability that two patches come from the same image–e.g. Constrastive Predictive Coding (CPC) [27] and variants [28, 29, 30, 31]. Dynamics-based approaches model the environment (e.g. by predicting the next state [32] or related objectives [33, 34, 35, 36, 37, 38]). Dynamics models have the possible advantage that they could be reused for planning. Dynamics are not necessarily visual and are sometimes specialized to the morphology or action space of the agent.

Mid-Level Vision in Other Contexts. Recent works have shown that mid-level visual representations can offer advantages in terms of generalization and sample complexity for downstream active tasks. Many works show one or a couple objectives for single tasks–e.g. a specific semantic representation for semantic driving ([39, 40, 41]) or dense object descriptors for manipulation ([42]). Outside of RL, [11] uses mid-level vision in conjunction with hard-coded grasping policies.

Recently, [9, 10] presented comprehensive studies in simulated navigation contexts using multiple objectives and multiple tasks. This many tasks/many objectives setup allows them to conclude that which objectives perform well depends on the task, which we also find. [9] then computationally derives a generic subset of mid-level representations that should perform as well as the best one in the whole set, but does not show when that best-possible feature is useful. Unlike previous studies, we demonstrate mid-level vision's ability to scale to harder tasks, to handle sim-to-real transfer, and we compare it to other approaches for incorporating invariances (e.g. domain randomization). By examining more complex domains, we find that we are able to train agents using mid-level features even in cases where other approaches fail.

Domain Randomization. One way of building agents with useful invariances is to make them learn it directly from data. [43, 44, 45, 46, 47] suggests using simulators to augment training with all the variation likely to be present at testing. However, training then takes longer and, in practice, varying more than one or two factors complicates the learning problem to the point that learning completely fails. Most importantly, domain randomization involves making the unrealistic assumption that we can enumerate all the invariances that are needed. We cannot, and for those which we can define, the corresponding variation is often difficult to build into a simulator (as evidenced by the numerous open problems in computer graphics). Our study shows that incorporating invariances via mid-level representations has multiple critical advantages over doing so via domain randomization. In particular, the mid-level representations can be trained asynchronously on large static datasets, making RL training simpler and more scalable. In contrast, domain randomization (unnecessarily) delays learning the invariances to be done in conjunction with RL training.

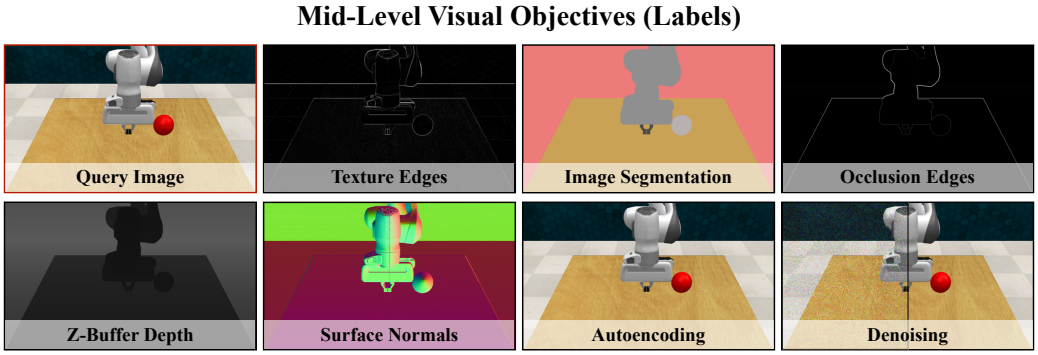

Figure 3: **Sample labels for mid-level visual objectives in the RLBench environment.** The objectives cover various modes of computer vision tasks including 2D, 3D, and semantic tasks.

3 Methodology

Our goal is to study the utility of mid-level visual representations for performing downstream motor tasks. Concretely, motor tasks require mapping observation histories to actions. Agents are trained in a scenario $\mathcal{P}$ but evaluated on a test scenario $\mathcal{Q}$. The goal is to maximize agent's performance in a test setting ($\mathcal{Q}$) that it is related, *but not identical*, to the training distribution. For example, $\mathcal{Q}$ might be a physical robot in scenes contain unseen objects, while training on $\mathcal{P}$ took place in a simulator.

We assume access to a set of 'mid-level' functions, $\Phi = \{\phi_1, \ldots, \phi_m\}$, that transform raw sensory data into potentially useful mid-level representations. The goal is to determine whether agents using Φ could perform better in the test setting $\mathcal{Q}$ than if they never had access to Φ. In this paper, we show that when Φ is a set of mid-level functions, agents access to Φ improves generalization and generalize better final perfomance.

3.1 Using Mid-Level Representations in Active Contexts

Why might a mid-level feature (e.g. image $\rightarrow$ surface normals) aid in downstream tasks, compared to end-to-end learning? This is an important question as preprocessing observations with ϕ might potentially discard information (e.g. color information in surface normals). Good representations preserve important information while discarding spurious details, providing "invariances" that make the train and test set more similar ($\phi(\mathcal{P}) \approx \phi(\mathcal{Q})$). When the train and test set become similar, improving performance on the train set generally improves test-time performance, too.

Really good representations also simplify training by using $\phi(\mathcal{P})$ instead of $\mathcal{P}$. By throwing away unimportant information and providing easily decodable outputs (e.g. linearly separable), great representations can reduce sample complexity and boost performance even on the training set, relative to learning *tabula rasa*. These ideas are illustrated in Figure 2.

We use representations derived from neural networks trained, offline, for computer vision objectives. Figure 1 shows how we use these networks in our active framework. While training agents from mid-level representations we freeze the mid-level network (ϕ) and use it to transform each observed image o_t into a summary statistic $\phi(o_t)$ that is then provided to the agent instead of pixels. Only the policy network is updated during agent training.

Studied Mid-Level Representations In this paper, we study representations from neural networks that were trained, offline, each for a specific vision objective from [48]. 7 of them are visualized in (Figure 3) and they cover various common modes of computer vision objectives: from texture-based (e.g. denoising), to 3D pixel-level (e.g. depth estimation), to semantic (e.g. image segmentation). As the networks were trained on data from indoor scenes significantly different than our manipulation setting, we fine-tune the networks, offline, with images from our simulator. We study the effects of domain shift and feature robustness in Section 4.4. All networks were trained with identical hyperparameters and using a ResNet-50 [49] encoder. For a full list of vision objectives and samples of the networks evaluated in our environments, see the supplementary.

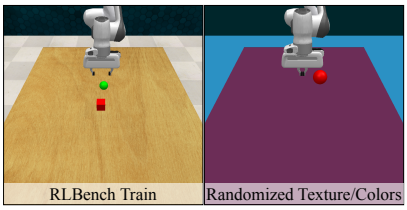

Figure 4: **Train vs. test observations in RLBench. Left:** The default environment for the Pick + Place task. The goal is colored green, object in red. **Right:** Sample observation from the test environment showing held-out randomized textures.

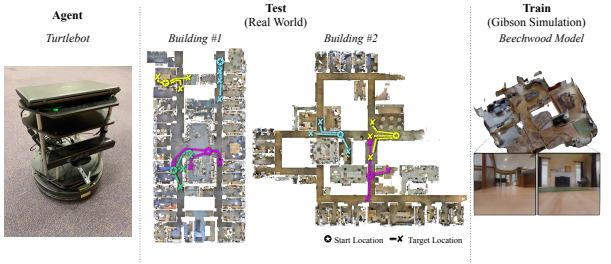

Figure 5: **Zero-shot visual sim-to-real. Right:** We train policies in a single building in simulation. **Middle:** We then directly test them in novel real world buildings, where agents have no prior knowledge of the building and no adaption period. **Left:** The TurtleBot uses only an RGB camera for vision and an IMU for localization, No depth/LiDAR sensors are used.

3.2 Metrics

Final performance: We report performance in both train and test environments using the final, fully-trained models. Theoretically, agents should be able to achieve 100% success in the training environments, even if the agent does not generalize at all.

Generalization: We break down final performance in terms of generalization from train to various test sets. We report both test-set performance with no shift (evaluated in the same simulator as training), under mild domain shift (with various colors swapped out at test-time), and under more drastic domain shift (unseen objects or under zero-shot sim-to-real transfer).

Sample efficiency: We examine whether agents equipped with mid-level vision can learn faster than a comparable agent learning *tabula rasa* (i.e. with raw vision, but no priors about the world). We report this in terms of the number of interactions with the environment.

3.3 Manipulation with Continuous Control

To study mid-level vision on harder tasks, we test the mid-level representations against baselines in three different manipulation tasks requiring continuous control from vision. This section describes the setup at a high level, and complete details for each subsection can be found in the supplementary.

Tasks: We use three common manipulation tasks: *Reach*, *Push*, and *Pick + Place* with sparse rewards (0 if within ϵ of the target, -1 otherwise). All tasks terminate episodes after 50 timesteps. Brief descriptions of each task are below, and full details for each of the the tasks are in the supplementary.

Reach: The agent must move the end effector to a random target position marked by a sphere.
Push: The agent must push a cube to a random target position marked by a sphere.
Pick and Place: The agent must pick up a cube and move it to a target position marked by a sphere. Both object and target locations are randomized.

Observation Space: For each task agents receive visual input from a single camera. Agents trained from pixels receive $64\times64\times3$ RGB images while mid-level approaches receive $16\times16\times8$ latent features. Except for the state baselines, no other information (including proprioception) is supplied.

Action Space: For all tasks, actions are XYZ+gripper values in [-1, 1] specifying end-effector deltas for a position controller. Gripper open/close is calculated via thresholding.

Environment: We adopt the RLBench environment [50] which is built on PyRep [51] and CoppeliaSim [52]. RLBench is suited for vision-based manipulation and offers more visually realistic observations compared to popular environments such as OpenAI Gym [53]. We use the Franka Emika Panda arm in our experiments.

Train/Test Split Agents are trained in the default RLBench tabletop environment shown in Figure 4. All policies are tested on a test set of held-out flat color textures that replace the table, floor, and/or background (Fig 4, right). We also show experiments with domain randomization during training (no texture overlap with the test set) or with held-out objects during testing.

Learning Algorithm. We train all agents using TD3 [54] and HER [55] using hyperparameters optimized for the *tabula rasa* baseline. Mid-level agents used the learning rate optimized for scratch in [9], but we ran additional hyperparameters sweeps for agents trained from scratch on the Reach and Pick+Place tasks. Because all hyperparameter searches were done using the from-scratch approach and the settings then applied everywhere, this setup should be biased *against* mid-level vision.

We followed the guidelines for training laid out in [55]. For some tasks we could not get the from-scratch approach to train using sparse rewards and we had to add additional reward shaping. The shaped dense rewards often help the mid-level approach too (see supplementary). In the main paper we present the best-possible approach for the pixel-based methods, but only show the sparse-trained policies for mid-level based agents since sparse rewards are usually easier to define in practice.

3.4 Generalization to the Real World

In cases where collecting real-world data makes training policies prohibitively expensive, the plentiful and cheap data from simulation provides a path to train policies that can then be deployed in the real world. However, the domain shift between simulators and the real world means that policies trained this way usually fail to generalize. We study this sim-to-real capability for agents trained with mid-level representations via RL.

Manipulation is usually non-quasistatic and, in practice, simulators will trade off physical accuracy (simulating all contact forces) for simulation throughput, resulting in a large environment dynamics sim-to-real gap. As we are primarily focused on perception, we focus on navigation contexts where the visual gap is responsible for the primary domain shift. Navigation-based sim-to-real is still highly non-trivial from a vision perspective as any number of discrepancies between simulated images and real images could cause the agent to fail to generalize; potential discrepancies include lighting variations, stitching artifacts and semantic distribution complexity.

We test sim-to-real generalization for the *point navigation* task from the CVPR19 Habitat Challenge. The task requires an agent to navigate to a target position (specified by coordinates) using visual observations and the agent's onboard odometry. Actions are discrete {`forward, pivot right, pivot left`} and episodes cap at 400 timesteps. We use policies from [9] trained with the same architecture as our manipulation tasks but using PPO [56] in a *single* building in the Gibson environment [57] and are tested in *different* (unseen) real-world buildings. We evaluate the mid-level vision based policies trained in [9] on a Turtlebot with Kobuki base and a Microsoft Kinect camera. Full details are provided in the supplementary.

3.5 State Representation Baselines

We describe the most important control groups here and defer remaining ones to the supplementary.

Tabula Rasa Learning (aka *from scratch*): This is the most common approach for end-to-end learning. The agent receives the raw RGB image as input and uses a randomly initialized AtariNet [1] architecture that is updated during training, along with the policy architecture.

Blind: The blind agent is the same as *scratch* but instead of an RGB image, receives a constant zero tensor. This shows how much can be learned by just exploiting the biases of the task.

State: In this setting the agent has access to the complete environment state: joint positions, the goal centroid and, if applicable, the object center. Since objects are always the same, this should be sufficient.

4 Results

Given that no two setups outside of simulation will be exactly the same, building in invariances into visuomotor policies could be helpful for bridging the gap between the training setup and any testing scenario. The rest of this section covers experiments that dissect whether such invariances are necessary, finding that incorporating them offers notable advantages in terms of final performance, generalization, and sample complexity. We then compare two main methods of building in invariances: either co-learning them during training (via domain randomization) or asyncronously learning them in different stages (via mid-level representations). We show that as the agent needs to learn more invariances, the co-learning problem becomes complicated and learning can collapse. We find that the mid-level approach scales and performs better.

4.1 Final Performance

We find that agents trained using mid-level representations achieve significantly higher success rates than agents learning from pixels, especially in harder tasks such as *Pick + Place*. The mid-level agents performed much better than scratch in the test environment, as shown in Fig. 6. Consis-

tent with Hypothesis II (*mid-level representations simplify training*) they performed better during training, too, especially for harder tasks.[1]

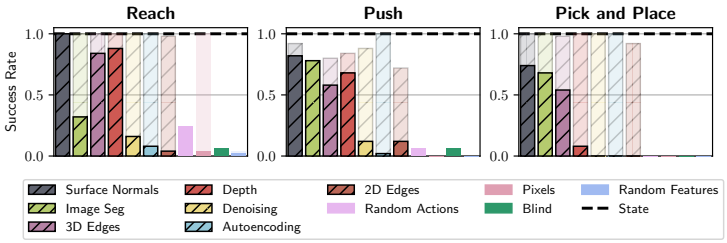

Figure 6: **Final performance on manipulation tasks.** Translucent bars indicate training performance; opaque bars show performance on the test set. Hatched bars indicate agents are trained with mid-level representations.

We found the above results despite the fact that the hyperparameters were optimized for the scratch baseline. In general, we found that the mid-level agents were less sensitive to choices of hyperparameters, and that the same or similar hyperparameters worked across multiple architectures, downstream learning algorithms, and simulators. They also did not require reward shaping.

4.2 Generalization

Comparison to generic state for unseen objects. In order to test whether mid-level representations could provide an easily decodable representation that enables both learning and generalization to unseen objects, we train agents for Pick+Place with 10 red, procedurally generated objects of different shapes. In contrast to the standard environment which only used a red cube, encoding the salient parts of the environment is now more complicated; we represent the object shape by its mesh vertex positions centered by the object centroid. In the more complex training environment, the state-based agent gets a 2% success rate during training (0% on unseen test), shown in Fig. 7. In contrast, using mid-level representations (normals), the agent has a 96% success rate on the training objects (90% on the unseen objects, 96% training objects colored green, and 88% on unseen green objects).

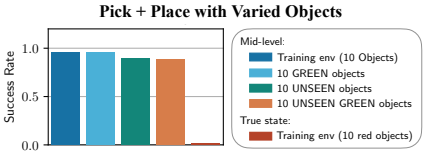

Figure 7: **Generalization to unseen objects.** In all tested environments, the mid-level policy trained to pick+place varied objects outperformed an agent using true state.

Invariance Method		Success Rate			
Mid-Level Representation	**Domain Rand.**	**Reach** Train	Test	**Pick + Place** Train	Test
None (*Scratch*)	No	100%	4%	0%	0%
Image Segmentation		100%	88%	100%	92%
Surface Normals		**100%**	**100%**	**100%**	**100%**
None (*Scratch*)	Yes	70%	20%	0%	0%
Image Segmentation		100%	100%	98%	98%
Surface Normals		**100%**	**100%**	**100%**	**100%**

Table 1: **Policies trained via different invariance-learning approaches.** The mid-level approach scales better to harder tasks, compared to *tabula rasa* or domain randomization.

Learning invariances with mid-level representations vs. domain randomization. Table 1 compares the performance of agents trained with different methods of incorporating invariances. Agents using the asyncronous (mid-level) approach perform better across train and all test environments. In particular, when tested on colors not seen in the training, mid-level vision has a success rate of 100% versus 20% when using pixels with domain randomization. The domain randomization approach trained from scratch also showed signs of learning collapse (100% $\rightarrow$ 70% success rate) as the randomization made the learning problem more difficult, a problem also found in [58, 59].

Sim-to-real transfer. Agents using mid-level vision generalize to new axes of variation not present during training and across large gaps of the simulator vs. physical world. After training in a single simulated building, we test in 24 scenarios in two unseen buildings in the real world. Scenarios vary significantly in length, complexity, and visual characteristics (mean length 5.24m; variance $3.65m^2$). In 594 evaluation runs and over 13 hours of execution time, we found that agents trained from scratch achieved an SPL [60] of 0.319 and a completion rate of 0.4 in the test environment,

[1]These results are intended to compare different features. Training the features longer further improves downstream performance (competitive with the *state* oracle). See Tab. 1 and supplementary for more analysis.

which was not significantly different than blind agents, as shown in Figure 9. Agents using mid-level features achieved a significantly higher SPL of 0.608 and a completion rate of 0.7. The use of the best features therefore provides a 90.6% increase in SPL and 75.3% increase in completion rate over scratch. Because we did no fine-tuning here, we could evaluate a slightly larger set of features here than for the simulation experiments. A full description of the experiment is available in the supplementary, and we provide videos from the agents' onboard cameras during the sim-to-real test episodes on our website.

4.3 Sample Efficiency

Training agents with mid-level representations dramatically increases sample efficiency, allowing for policies to be trained on difficult tasks using sparse rewards. Across all tasks, agents using mid-level representations converge within 2x the number of steps required to train from state (e.g. 450k steps vs 250k in Pick + Place). This is several times faster than learning from scratch (when it is even possible for scratch to learn anything—even with reward engineering, the from-scratch approach never succeeds on the test set). We provide train/test curves in the supplementary.

4.4 Analysis of Features for Downstream Tasks

Relationship between mid-level objectives and downstream tasks.

Given that mid-level objectives are typically defined irrespective of any downstream task, we ask whether representations that perform better on their objective also perform better on downstream tasks. Figure 8 shows the downstream performance of agents trained using surface normal or image segmentation features at various checkpoints during training. We found that generally, when the feature is useful for the task, the two performances are correlated (both features on pick+place).

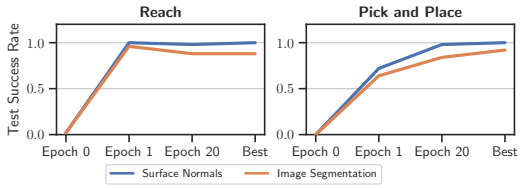

Figure 8: **Performance on mid-level vs. downstream tasks.** If a feature initially performed well on the downstream task, a better version further improved downstream performance.

Feature rank stability in different environments.

We found that within each task, feature order was notably stable, even across large sim-to-real domain gaps: the Spearman's rank correlation between the feature rankings found by testing in the real world vs. testing in simulation (Gibson) was ρ=0.77. [9] compares across simulators, but not in a zero-shot manner, finding that feature rank correlation was ρ=0.88 between Habitat [61] and Gibson when mid-level agents were trained in the respective environments. This inter-environment correlation is not simply because some features are more useful than others (i.e. the same features are always useful): feature ranking in [9] was uncorrelated across tasks.

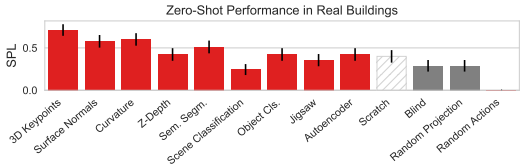

Figure 9: **Performance in simulation vs. after zero-shot transfer to the real world.** Agents using mid-level representations (red) significantly outperform those trained from scratch. Agents along the the x-axis are ordered by descending performance in simulation. Performance for top-performing features (e.g. curvature) was about the same in simulation and on the physical robot [9]. Agents trained from scratch do not significantly outperform blind agents in the real environment (standard error shown in chart).

5 Conclusion

High-level takeaway: Mid-level representations simplify training, improve generalization, and aid training speed. **Mid-level representations should be the preferred input to policies**, especially for harder tasks, where they are more viable than alternative methods of invariance learning.

In the supplementary material we have a more complete discussion of lessons learned, future directions, and also additional experiments. That, as well as demos, visuals, and code, are available on the project website website: `https://midlevel.berkeley.edu`.

Acknowledgments

This material is based upon work supported by ONR MURI (N00014-14-1-0671), an Amazon AWS Machine Learning Award, NSF (IIS-1763268), a BDD grant and Toyota Research Institute (TRI)[2].

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015. URL http://arxiv.org/abs/1504.00702.

[3] Y. Liu, B. Logan, N. Liu, Z. Xu, J. Tang, and Y. Wang. Deep reinforcement learning for dynamic treatment regimes on medical registry data. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 380–385. IEEE, 2017.

[4] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, pages 3814–3823, 2018.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[6] A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. In *Advances in Neural Information Processing Systems*, pages 9094–9104, 2018.

[7] Y. Sun, X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt. Test-time training for out-of-distribution generalization. *arXiv preprint arXiv:1909.13231*, 2019.

[8] N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[9] A. Sax, J. O. Zhang, B. Emi, A. Zamir, S. Savarese, L. Guibas, and J. Malik. Learning to navigate using mid-level visual priors, 2019.

[10] B. Zhou, P. Krähenbühl, and V. Koltun. Does computer vision matter for action? *arXiv e-prints*, art. arXiv:1905.12887, May 2019.

[11] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. URL https://yenchenlin.me/vision2action/.

[12] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014. URL http://arxiv.org/abs/1406.2283.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[14] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

[15] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. *Indoor Segmentation and Support Inference from RGBD Images*, pages 746–760. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33715-4. doi:10.1007/978-3-642-33715-4_54. URL https://doi.org/10.1007/978-3-642-33715-4_54.

[16] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 689–696, Sept 2009. doi:10.1109/ICCVW.2009.5457637.

[17] Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018.

[18] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[19] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[20] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[21] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016.

[22] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi:10.1126/science.1127647. URL http://science.sciencemag.org/content/313/5786/504.

[24] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi:10.1145/1390156.1390294. URL http://doi.acm.org/10.1145/1390156.1390294.

[25] L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR 2017*, 2017.

[26] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 512–519. IEEE, 2016.

[27] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

[28] O. J. Hénaff, A. Srinivas, J. D. Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord. Data-efficient image recognition with contrastive predictive coding. *CoRR*, abs/1905.09272, 2019. URL http://arxiv.org/abs/1905.09272.

[29] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations, 2020.

[30] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning, 2019.

[31] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[2] TRI provided funds to assist the authors but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

[32] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[33] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *CoRR*, abs/1611.01779, 2016. URL http://arxiv.org/abs/1611.01779.

[34] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL http://arxiv.org/abs/1705.05363.

[35] Z. Zhu, T. Oskiper, S. Samarasekera, R. Kumar, and H. S. Sawhney. Ten-fold improvement in visual odometry using landmark matching. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, Oct 2007. doi:10.1109/ICCV.2007.4409062.

[36] A. Raffin, A. Hill, R. Traoré, T. Lesort, N. D. Rodríguez, and D. Filliat. S-RL toolbox: Environments, datasets and evaluation metrics for state representation learning. *arXiv preprint arXiv:1809.09369*, 2018.

[37] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL http://arxiv.org/abs/1606.07419.

[38] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020.

[39] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun. Driving policy transfer via modularity and abstraction. *CoRR*, abs/1804.09364, 2018. URL http://arxiv.org/abs/1804.09364.

[40] A. Mousavian, A. Toshev, M. Fiser, J. Kosecka, and J. Davidson. Visual representations for semantic target driven navigation. *CoRR*, abs/1805.06066, 2018. URL http://arxiv.org/abs/1805.06066.

[41] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. *CoRR*, abs/1810.06543, 2018. URL http://arxiv.org/abs/1810.06543.

[42] P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*, 2018.

[43] 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. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[44] F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[45] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[46] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to manipulate deformable objects without demonstrations. *arXiv preprint arXiv:1910.13439*, 2019.

[47] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[48] A. R. Zamir, A. Sax, W. B. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

[49] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[50] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. RLBench: The robot learning benchmark & learning environment. URL https://arxiv.org/abs/1909.12271v1.

[51] S. James, M. Freese, and A. J. Davison. Pyrep: Bringing V-REP to deep robot learning. *CoRR*, abs/1906.11176, 2019. URL http://arxiv.org/abs/1906.11176.

[52] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[53] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018. URL http://arxiv.org/abs/1802.09464.

[54] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

[55] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. URL http://arxiv.org/abs/1707.01495.

[56] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[57] F. Xia, A. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. Gibson Env: Real-world perception for embodied agents. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] 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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[59] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[60] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[61] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[62] A. Raffin, A. Hill, R. Traoré, T. Lesort, N. Díaz-Rodríguez, and D. Filliat. S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning. *arXiv preprint arXiv:1809.09369*, 2018.

[63] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

[64] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

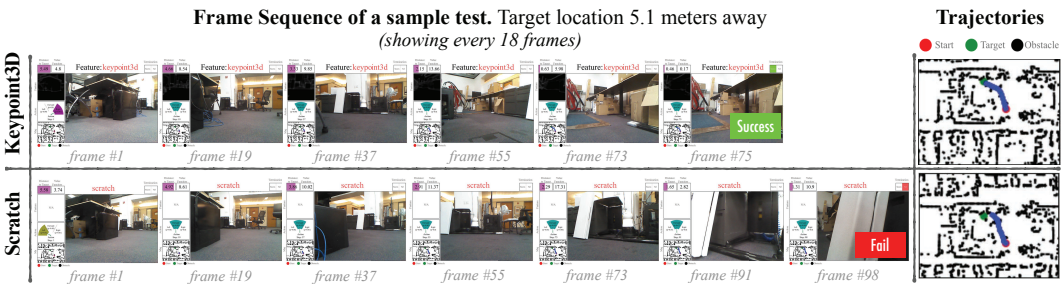

Figure 10: **Frame sequence of a sample test episode.** Selected frames from keypoint3D (**top**) and scratch (**bottom**) experiments. **Mid-level versus no features:** Even when the target is far away and occluded from the agent, mid-level features are able to successfully complete the task - here is an example of using 3D keypoints. The absence of such features leads most of the cases to failed attempts. **We include more than 100 execution videos in the supplementary main video and in the folder `sim_to_real/test_episode_videos`.**

[65] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

6 Project Website

The project website has a nice overview video that shows *various results from the proposed study* including a description of the main hypotheses and comparisons to controls.The clip also shows some sample episode of the sim-to-real generalization experiments, which we find insightful. **We strongly recommend watching the video clip** on the project website http://midlevel.berkeley.edu/.

7 Discussion of Lessons Learned

First, we found that agents trained using mid-level vision could be successfully trained for harder tasks than possible when training from scratch or using domain randomization. Training was less sensitive to the choice of hyperparameters, and training speed improved. Overall, these results ere consistent with the hypothesis that mid-level representations can simplify the input space and make the learning problem easier.

Second, we found the agents trained using mid-level representations were significantly more robust to domain shifts than agents trained from scratch (and also those trained using domain randomization). We showed this in simulation on multiple manipulation tasks under multiple types of domain shift (new objects, textures). We also showed this in the sim-to-real setting, successfully deploying mid-level-based simulator trained policies in unseen real-world buildings. The robust generalization is consistent with our second hypothesis, that mid-level representations also align training and test distributions to improve test-time performance. While approaches for solving mid-level objectives are generally less sensitive than methods trained for robotics using RL, they are still susceptible to domain shift. Improvements to methods for approximating these individual objectives would probably carry good knock-on effects for agents trained using mid-level vision.

Third, which features performed well depended on the choice of task, but not so much on the environment used for training. The advantage of picking a good feature (vs. training from scratch) grew as tasks became more difficult, underscoring the importance of picking a good feature. While we did not explore how to pick a generic set of features, this would be an important avenue and an [9] has proposed an initial (computationally-derived example). Without the dependence on task, these features would be expected to work well in most environments.

8 Where to Go From Here:

That mid-level features work well in harder contexts suggests that we are ready to "close the loop" between features and downstream tasks. Features are currently defined irrespective of any downstream task. Choosing a representative set of benchmark tasks that "cover" downstream robotic tasks would make it possible to choose new computer vision objectives that better benefit downstream motor tasks. This same strategy of a defining of benchmark tasks has made it possible to design ever-better architectures (e.g. ResNets in computer vision and Transformers in language) that work well for most perception tasks in that domain.

9 Videos of sim-to-real test episodes from physical onboard cameras

We ran 594 evaluation episodes in the sin-to-real experiments comprising 13 hours of runtime. In the folder `sim_to_real/test_episode_videos`, we've included videos from onboard RGB cameras during a representative sample of the test episodes. These videos include a dashboard showing useful metrics at each point in time such as distance to goal, and episode success. Frames sequences from two videos videos are shown in Figure 10.

10 Code

We provide all of our code through a Github repository available https://github.com/alexsax/robust-policies-via-midlevel-vision

11 Experiments with Shaped Rewards

Using sparse rewards and HER, we were unable to get the agent trained from scratch to learn useful behaviors in our environments, even after running multiple hyperparameter searches. HER [55] required using shaped rewards to train from pixels, and we were also able to train an agent only by using a dense, shaped reward. Agents trained using mid-level vision outperformed agents trained from scratch even when the pixel-based agents were trained using a dense reward, but dense reward also improved performance for mid-level based agents, shown in Figure 16.

12 Complete Sim-to-Real Episode-Level Results

These are provided in the file `sim_to_real/test_episodes.csv`.

13 Mid-Level Representations: Latent vs. Outputs

We chose to use the latent representations from neural networks as our mid-level representations. These have the advantage that the latent representations as have a uniform shape no matter which mid-level objective they were trained for, which makes comparisons easier. We could have also chosen to use the outputs and we found (on the Reach task using surface normal features) that these often have similar performances to using the latent representations.

14 Feature Network: Initialization and Training Length

In the main paper we showed results that compared feature performance on their objective with the downstream performance of agents trained on top of those features. That experiment used feature networks, trained from scratch, for longer than the feature networks used in the "Final Results" section. In Figure 11 we analyze why those networks outperformed those in the "Final Results" section, finding that it was the longer training, which is consistent with the results of that experiment (longer training improves the features which improves downstream performance). Initializing the features from the Taskonomy-trained models [48] also marginally improved performance and made downstream RL training more consistent.

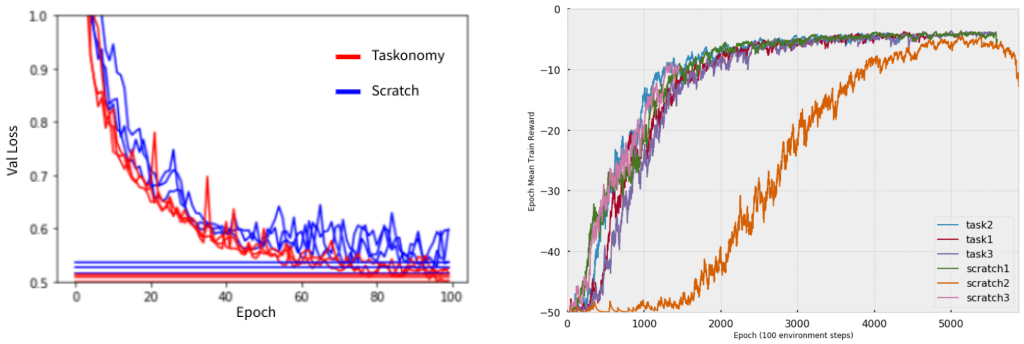

Figure 11: **We re-train networks with 3 random seeds each to compare the effect of initializating from Taskonomy vs. scratch**. Left: Comparing loss during training of the computer vision (surface normals) tasks; horizontal lines indicate the minimum loss reached. Right: Comparison of their downstream training reward in the Pick + Place task.

15 Descriptions of Manipulation Tasks

We provide full descriptions of the task setups within RLBench.

Reward: Sparse reward is given by 0 when the cube is within an $\epsilon = 4$cm distance of the target, and -1 otherwise. Dense/shaped reward is given by the negative euclidean distance between the target and the end effector / object, with a positive reward bonus of 0.1 when within the ϵ distance. In total, we tried rewards of the form $-|g - s_{object}|_2^p, -|g - s_{object}|_1^p, p \in \{1, 2\}$ and a brief search over reward bonuses. [55] finds that more advanced shaped rewards considering future states does not help in learning tasks.

Gripper: For Reach and Push, the gripper is in a fixed position. For Pick and Place, it is controlled by the agent's actions. A positive value is considered a close action if not already closed, and vice versa a negative value an open action.

Extra Task Details:

Reach: The target sphere location is randomized in a cube. The agent begins in a fixed position within the cube.

Push: The cube starts in a fixed location on the table. The target sphere location is randomized in a square around the cube. The agent begins in a fixed position above the cube.

Pick and Place: The cube location is randomized in a square on the table. The target sphere location is randomized in a square at a fixed z position above the table. To aid exploration as in [55], the agent begins half of the episodes gripping the cube. We were also able to train mid-level policies by starting half of the episodes with the target sphere on the table, but present the results using the aforementioned setup.

16 Description of Navigation Tasks

The task under consideration is PointGoal [60] (referred to as Local Planning in [9]). In the PointGoal task, the agent must navigate to a specified (fixed) target location coordinate. The target location is specified as a vector from the agent to the target. The agent receives positive reward for Euclidean distance to goal at each timestep, and receives negative reward for collisions and time spent. During training, the episode terminates successfully when the agent comes within $0.5m$ of the goal location and fails if the agent doesn't reach the goal in 400 actions. In real-world testing, the episode additionally terminates if the agent collides with a wall or obstacle.

Action space Discretized to $A = \{$*forward 0.25m*, *pivot right* 10°, *pivot left* $10^\circ\}$. To abstract away difficulties and imperfections in simulating physical dynamics, Gibson assumes no lag dynamics in either the controller or camera; thus, the sense-action loop for the agent consists of 1) policy receives RGB image from Gibson; 2) policy produces an action; 3) embodied agent teleports to resulting pose; 4) policy receives a new RGB image. We refer the reader to [57] for a detailed discussion of collision handling.

Observation space of the agent consists of Taskonomy [48] encoder representations (with shape $16 \times 16 \times 8$) which output the mid-level features as well as the vector to target location. The vector to target location is encoded as $\cos$ and $\sin$ of the relative heading of the agent to the target, and the magnitude of the distance. To match the shape of the Taskonomy encoders, the target vector is tiled to have shape $16 \times 16 \times 3$.

17 Train and Test Splits

17.1 Manipulation

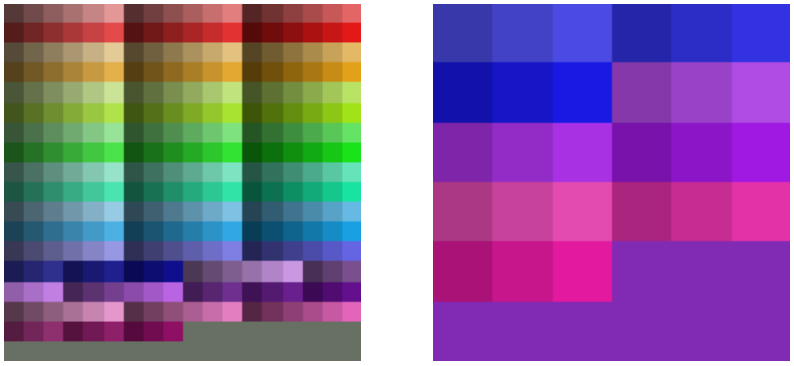

Figure 12: **Train/test split of textures for domain randomization. Left:** Colors used for textures in the *domain randomization* treatment group. Replaced table, floor, and background textures. **Right:** Textures used in the testing environment for all policies.

We create our train/test set through applying color textures to the floor, walls, and table. To generate colors, we generate HSV tuples with equally spaced values across each dimension, throwing away values for visibility. We take 90% of these values along each dimension for train, and hold out the remaining 10% for test. This formed the train and test for policies trained with domain randomization, whereas for policies trained with the regular environment, the test set was given by the domain randomized train set.

17.2 Sim-to-real

The sim-to-real policies are trained in simulation in a single building and tested in two unseen physical buildings. For complete information, please see Section 19.

18 Mid-Level Vision Objectives

We trained our agents using the following mid-level objectives:

All mid-level vision objectives:

Mid-Level Vision Objectives ($\widetilde{\Phi}$)
Autoencoding
Denoising
Depth Estimation, Z buffer
Edge Detection, 2D
Edge Detection, 3D
Image Segmentation
Surface Normal Estimation

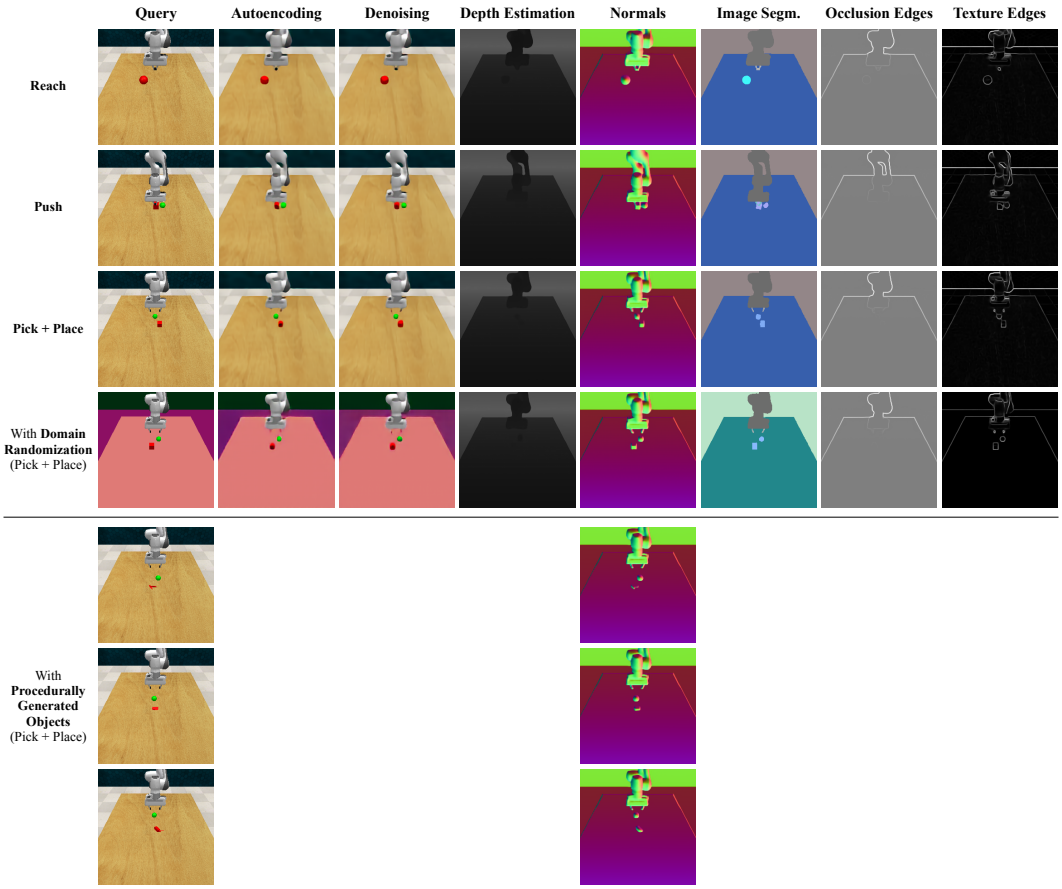

Figure 13: **Mid-level neural networks evaluated in our environments.**

Except for *Image Segmentation* we used the tasks as defined in [48]. Image segmentation is similar to instance segmentation, but in our case each instance of an item `floor, table, arm, gripper, object` are sorted into their own label. The `object` class includes target spheres as well as cubes.

For denoising, we use inputs with added Gaussian noise independent per pixel sampled from $\mathcal{N}(0, 0.05)$.

Dataset Generation: We generate a dataset to train the mid-level vision objectives by taking random actions in various environment setups. The environments we train on are Reach and Push as above, as well as Pick + Place with agent starting above and near the block. For each of these setups, we generate with and without domain randomization as well as with 2 camera angles. We take uniformly random actions for 25 resets and 150 steps per reset. For the procedural objects, we train another dataset which is comprised of 45 objects each with 50 resets and 2 camera angels. We adapted the code from [62].

Training / Fine Tuning: For final performance figures, we initialize from Taskonomy [48], and for the checkpointing figure we initialize from scratch. We use the Adam optimizer with learning rate 3e-4, with batch size 32, and number of epochs 150 and 500 respectively, taking the model with the best validation loss. We use L1 Loss for all takss except for Image Segmentation, where we use Cross Entropy Loss. For the procedural objects, we fine tuned the mid-level normal policy from the final performance figures on just the procedural object dataset, using learning rate 3e-4 with batch size 32 and 500 epochs.

We show (non-cherry-picked) results of the final networks evaluated in our environments in Figure 13.

19 Sim-to-Real Setup

For the sim-to-real setup we test the generalization of the pretrained "Local Planning" (PointGoal) policies from [9]. The policies are trained in the Gibson simulation environment [57], which employs virtualized spaces to render images to the agent that capture much of the inherent visual and semantic complexity of the real world. The training environment, called Beechwood, is a large house with $154.9m^2$ of navigable space.

The policies are AtariNet [63] models trained with Proximal Policy Optimization [**?**] and Generalized Advantage Estimation [64]. A thorough hyperparameter search was conducted for the *scratch* baseline, with the best performing hyperparameters frozen and used for all policies; this method favors the *scratch* baseline, making any gains over *scratch* all the more significant.

19.1 Hardware Setup

We used a Turtlebot with Kobuki base. The Turtlebot is a reliable mobile base platform, with easy interaction using the Robot Operating System (ROS) framework. The on-board computer is quite minimal, and doesn't come with sufficient computing resources to run the policies; we therefore network Turtlebot with a server equipped with an NVIDIA RTX 2080 Ti GPU. We additionally augment the platform with a WiFi AC wireless adaptor and router to minimize communication latency between Turtlebot and the server. The round-trip latency between the main computer and Turtlebot is optimized to roughly .2s per action, resulting in near-continuous action execution.

Our camera is a Microsoft Kinect, which has a 43° vertical field of view and outputs color images as $640 \times 480 \times 3$ tensors. We only use the RGB output of the camera and discard depth.

We discretize the Turtlebot's continuous action space into $A = \{$*forward*, *pivot right*, *pivot left*$\}$ to match that of the policies; each action corresponds to a velocity command to the Turtlebot base, resulting in roughly 10cm movement along robot x-axis and heading changes of -10° and +10° for each of the three actions respectively.

For calculation of vector from agent to target, we utilize the Turtlebot's on-board odometry as ground truth; in particular, we don't use external localization such as a GPS, beacon, or SLAM [**?**] module. We find that the odometry exhibits acceptable performance for our scenarios, with average error of $0.25m$ across all runs.

Our controller communicates with the main computer with a client/server pattern. When the controller receives a motion command, it executes an action for .6 seconds and stops. The next received image is then sent back to the main computer. This pattern is used to ensure images acted on by the policy are the final result of executing the previous action, and so provides close parity with the sense-action-loop training dynamics experienced by the policy in simulation.

19.2 Real-World Testing Scenarios

We test our policies on 24 scenarios consisting of start-goal location pairs across two office buildings with notably diverse interior scenes, seen in Figure 14. Each building comes with pre-existing 3D scans and metric meshes, along with camera positions used for scanning. The scans are only used for planning the study, and the agent does not receive any prior information from them. Scenarios were generated in each building by 1) selecting start locations uniformly at random from the list of camera positions constrained to public areas; 2) sampling a target distance d from a Gaussian with mean $5m$ and standard deviation $4m$; 3) choosing the camera position that has distance from start location closest to the sampled distance; 4) uniformly sampling the initial orientation from $[-\pi, \pi]$. Following this procedure, we generated four starting locations for Building 1 and three starting locations for Building 2, each with three goal locations for a total of 12 scenarios in Building 1 and 9 scenarios in Building 2. We generated a total of 594 evaluation runs across all scenarios and policies, for a total of 13 hours of continuous execution time. A full list of scenarios and their geometric characteristics can be found in the supplementary material file `sim_to_real/test_episode_results.csv`.

Scenarios vary significantly in length, complexity, and visual characteristics. One metric used to evaluate the difficulty or non-linearity of a physical navigation scenario is *navigation complexity* [57], defined as the ratio of geodesic (i.e. optimal path) distance to Euclidean distance. Average geodesic distance among scenarios is $4.92m$ (variance $4.16m^2$) in Building 1, $6.42m$ (variance $0.86m^2$) in Building 2, and $5.24m$ (variance $3.65m^2$) overall, while average navigation complexity among scenarios is 1.18 (variance 0.036) in Building 1, 1.06 (variance 0.001) in Building 2, and 1.15 (variance 0.03) overall. Visual differences between buildings include floor patterns and coloration, lighting sources & conditions, furniture styles, clutter makeup and distribution, and wall material (e.g. glass or drywall).

20 Policy Learning Setup

Architecture: Our architecture follows [9] closely:

Pretrained Feature encoder: For all tasks, we modified a ResNet-50 encoder with no average-pooling and replace the last stride 2 convolution with stride 1. This gives us an output shape of 16 x 16 x 2048. We use a 3 x 3 convolution to transform the output to a shape of 16 x 16 x 8.

Feature readout network: The feature readout network maps the feature encoding to a final representation. It consists of one convolutional (Conv) layer and two fully-connected (FC) layers:

- Conv, 32 channel, 4x4 kernel, stride 4
- FC, ouptut size = 1024
- FC, ouptut size = 512

We train the readout networks on the vision objectives end to end, and at the end of training freeze the encoder output for use as inputs to the downstream policy networks.

Atari-net network: The atari-net network [63] consists of three convolutional layers; we modify it slightly as follows:

- Conv, 32 channel, 4x4 kernel, stride 2
- Conv, 64 channel, 4x4 kernel, stride 1
- Conv, 64 channel, 3x3 kernel, stride 1
- FC, ouptut size = 1024
- FC, output size = 512

Putting it together: For mid-level policies, we use images of size 256 x 256 x 3 as input for the feature encoder, and freeze the output for use as the representation for downstream policy networks. We use the Atari-net network with slight modifications for learning from pixels on images of size 64 x 64 x 3.

Hyperparameters: We conducted a grid search for the best hyperparameters for learning from scratch for Reach, and then used those same hyperparameters for mid-level policies across all tasks, aside from a few noted differences: first, for Reach, noise for exploration was epsilon-greedy with $\epsilon = 0.3$, while for Push and Pick and Place we used Gaussian noise decaying from $\sigma = 0.3$ to 0.1 over 10^6 timesteps. For learning rate we used 1e-5 for both actor and critic when learning from pixels, and 1e-4 for both actor and critic when learning mid-level policies. We find that mid-level policies are still successful using a learning rate of 1e-5, but we are unable to successfully train policies from

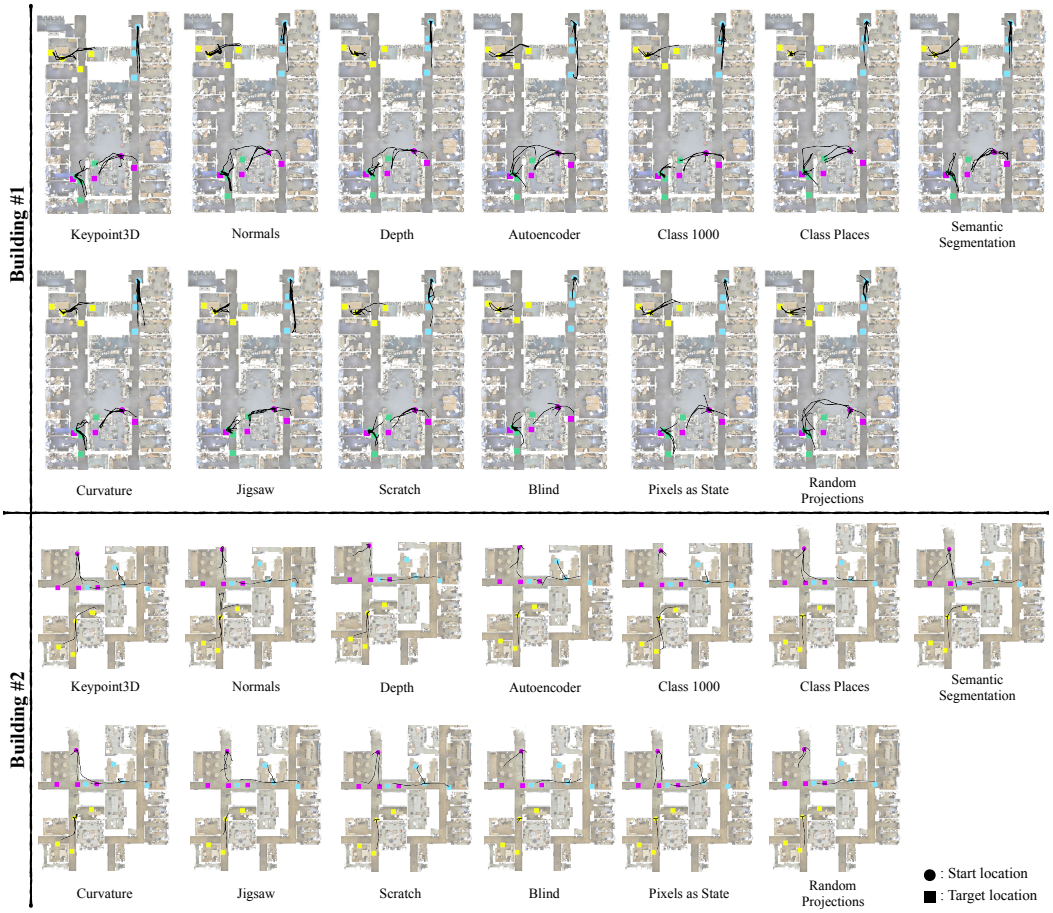

Figure 14: **Recorded Trajectories** of all experiments for the 2 text buildings. We ran a total of 594 evaluations in both buildings, for a variety of start-target locations and features. The recorded trajectories are visualized per feature and start-target pair.

scratch using a learning rate of 1e-4. The rest of the hyperparameters are the same: Batch size of 128, discount factor 0.95, target policy noise 0.2, policy delay of 2, replay buffer size of 1e6, rollouts of size 100 per epoch, gradient updates of size 100 per epoch, rollouts of size 50. We use the Adam optimizer [65] and no frame stacking.

HER: To help with exploration for training from sparse rewards, we implement a visual form of HER [55], where instead of concatenating different goals to a state/proprioception observation, we re-render the observation using different goals. For goal resampling, we use the "final" method with k=4.

20.1 Training Summary

We provide a full description of the main agents we trained on Reach, Push, and Pick + Place. We focused on training agents with dense rewards as well as with sparse rewards in conjunction with HER. Policies trained using state observations were successful for all settings. Policies trained from scratch were only successful on Reach with dense reward, failing on all tasks with sparse reward. Successful mid-level policies were trained on Reach and Push with dense reward, as well as on all tasks with sparse reward.

21 Full Descriptions of Baselines

We use the following baselines, which are the same as [9]. They are summarized in [9] and reproduced here:

Tabula Rasa (Scratch) Learning: The most common approach, *tabula rasa* learning trains the agent from scratch. In this condition (sometimes called *scratch*), the agent receives the raw RGB image as input and uses a randomly initialized AtariNet [63] network.

Blind: The *blind* baseline is the same as *tabula rasa* except that the visual input is a fixed image and does not depend on the state of the environment. This gives us an idea of how much performance can be learned from nonvisual biases, correlations, and structure of the environment.

Random Nonlinear Projections: this is identical to using *mid-level* features, except that the feature encoder is not pretrained but rather randomly initialized and then frozen. The policy then learns on top of this fixed nonlinear projection. This addresses the possibility that the ResNet architecture, not the offline perception task, is responsible the representations' success.

Random Actions: this uniformly randomly samples from the action space. It calibrates how difficult the task is without any specialized method and determines how much can be obtained just from random chance.

Pixels as Features: this is identical to using *mid-level* features, except that we downsample the input image to the same size as the features (16×16), apply a convolutional layer to match output sizes, and use it as the feature. This addresses whether the feature readout network could be an improvement over AtariNet [63].

In addition, we test:

State: In this setting the agent has access to the complete environment state: joint positions, the goal centroid and, if applicable, the object center. Since objects are always the same, this should be sufficient.

22 Train and Test Curves

22.1 Train/test curves for mid-level policies trained with sparse reward

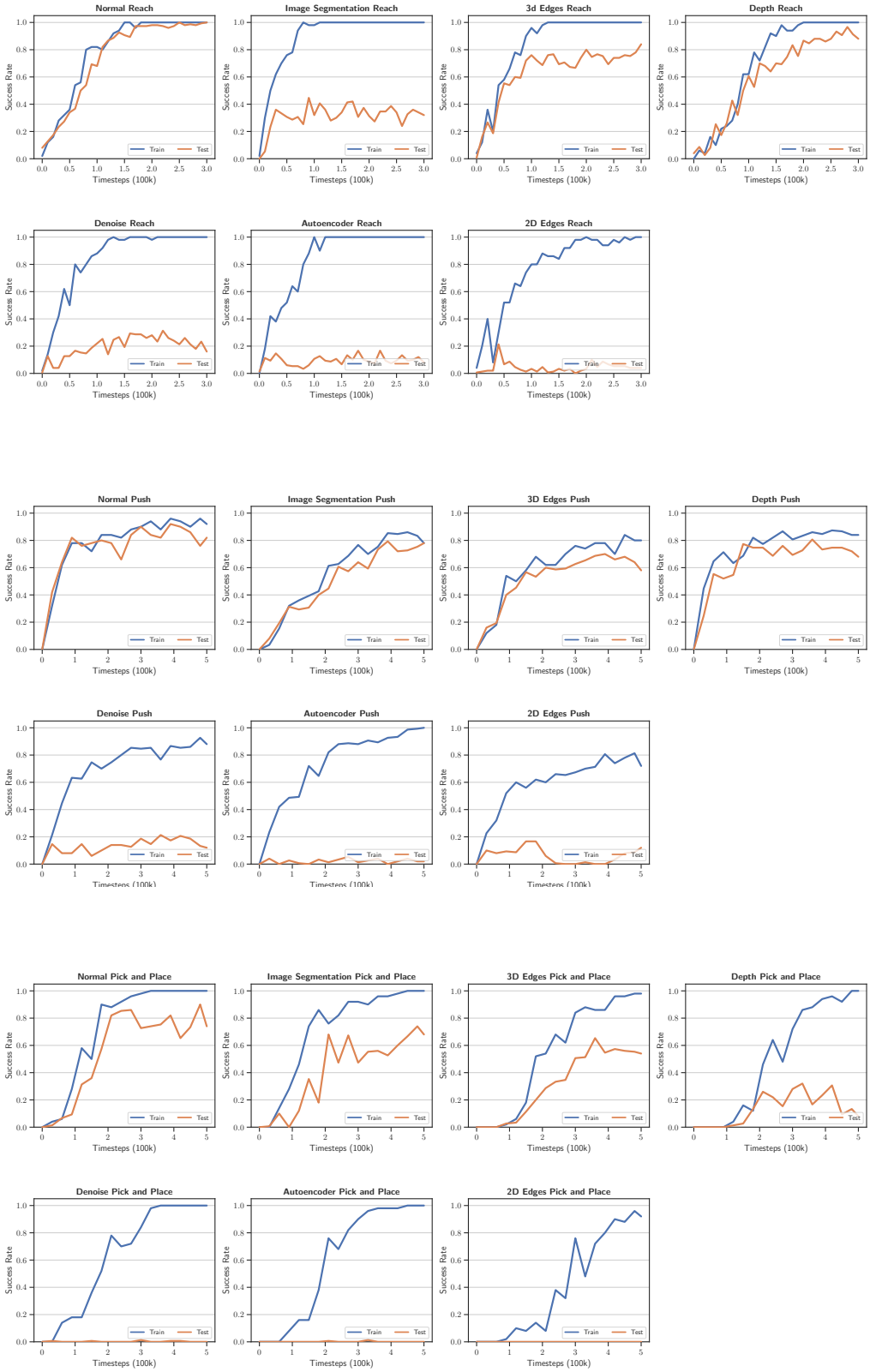

Figure 15: **Train/test curves for policies trained with sparse rewards.** We omit the curves for *scratch* as they failed to train in all environments.

22.2 Train/test curves for policies trained with dense reward

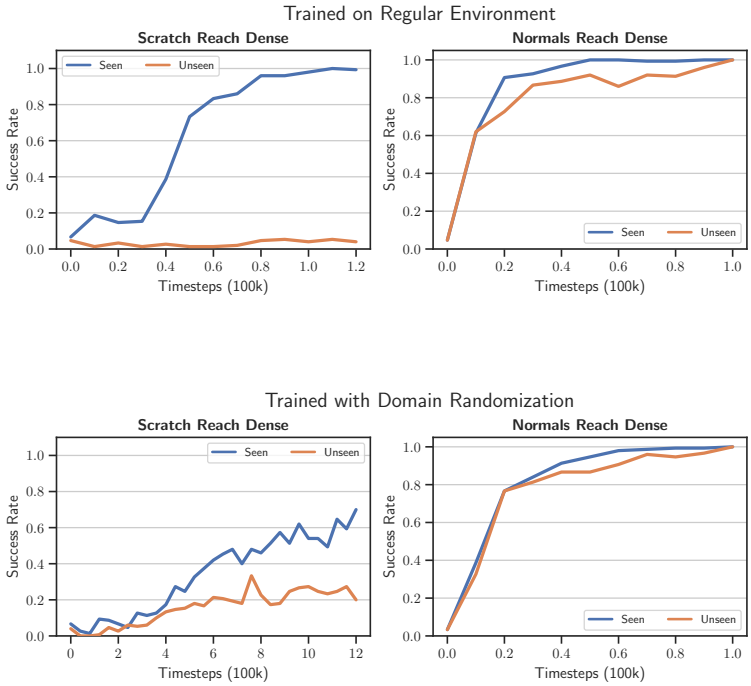

Figure 16: **Train/test curves for policies trained with dense reward.** The reward engineering improved performance and was necessary to get pixels to train at all. However, we found the approach scaled poorly as harder tasks required prohibitive amounts of engineering.

22.3 Train/test curves for policies trained with additional objects

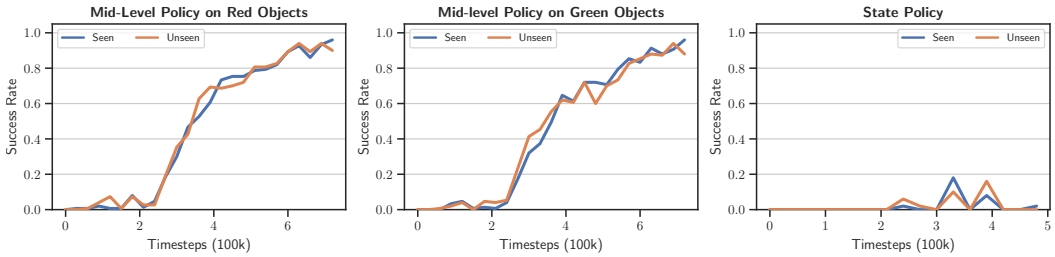

Figure 17: **Train/test curves for policies trained to pick + place various objects.**

22.4 Test curves for policies trained from state

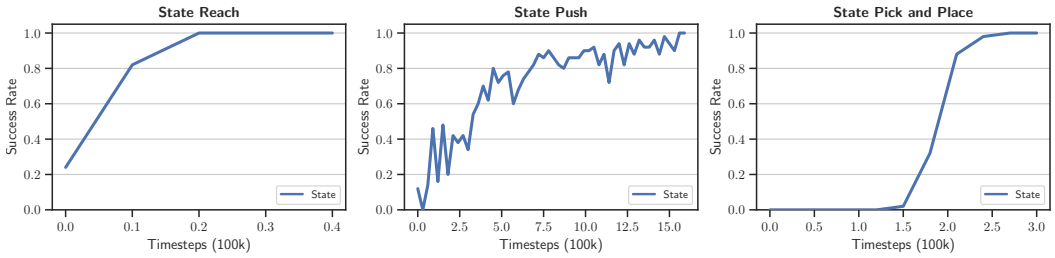

Figure 18: **Train/test curves for policies trained from true environment state.**