
DoorGym: A Scalable Door Opening Environment and Baseline Agent

Yusuke Urakami¹, Alec Hodgkinson¹, Casey Carlin¹, Randall Leu¹, Luca Rigazio^{1,2}

¹Panasonic Beta, CA, USA ²Totemic Inc., CA, USA
yusuke.urakami@us.panasonic.com

Pieter Abbeel

University of California Berkeley, USA
pabbeel@cs.berkeley.edu

Abstract

In order to practically implement the door opening task, a policy ought to be robust to a wide distribution of door types and environment settings. Reinforcement Learning (RL) with Domain Randomization (DR) is a promising techniques to enforce policy generalization, however, there are only a few accessible training environments that are inherently designed to train agents in domain randomized environments. We introduce DoorGym, an open-source door opening simulation framework designed to utilize domain randomization to train a stable policy. We intend for our environment to lie at the intersection of domain transfer, practical tasks, and realism. We also provide baseline Proximal Policy Optimization and Soft Actor-Critic implementations, which achieves success rates between 0% up to 95% for opening various type of doors in this environment. Moreover, the real world transfer experiment shows the trained policy is able to work in the real world. Environment kit available here: <https://github.com/PSVL/DoorGym/>

1 Introduction

Door opening is a fundamental skill for any robot that interacts with humans. In the past, there have been a variety of approaches to solving the door opening task [1–9]. Prior work can be roughly categorized into a few different approaches: Door Keypoint Detection, Model Based, and RL based. Keypoint Detection approaches often involve locating the doorknob location, and axes of rotation, then using motion planning to open the door [1, 7, 8]. There have been several model based approaches that range from creating a model of the door [5] (Karayiannidis et al, 2012) to learning the kinematics of doors [6](Endres et al, 2013). Reinforcement Learning based approaches are beginning to gain more attention. Kalakrishnan et al, 2011 [9] use compliant control and PI^2 [10](Theodorou et al, 2010) to open a door in a very narrow environment. Nemec et al, 2017 [3] expanded the use of compliant control and PI^2 to transfer from simulator to multiple different doors, including a real door. More recently there have been some purely RL based approaches, though they have not been focused on opening doors. Both Gu et al, 2017 [4] and Rajeswaran et al, 2017 [11] have door opening environments. These research achieved an door opening task at certain level, however, they were not focusing on a generalization of a policy. A practical door opening policy must be robust to many different doors, lighting conditions, and different environment settings.

There are various techniques to make a policy robust to many different environments. Domain randomization (DR) is one such technique. DR assumes that it is hard to perfectly model the target domain, but it is easy to create many different simulations that approximate the target domain. With

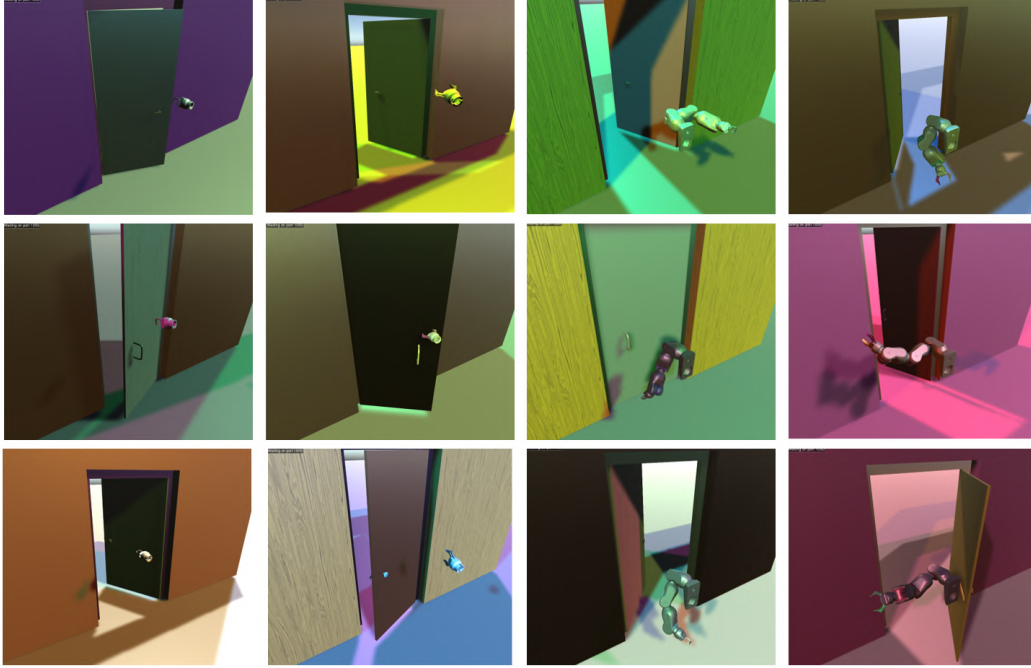


Figure 1: Various doors sampled with different visual and physical characteristics. For a full list of randomizable parameters, see table 5 in the appendix.

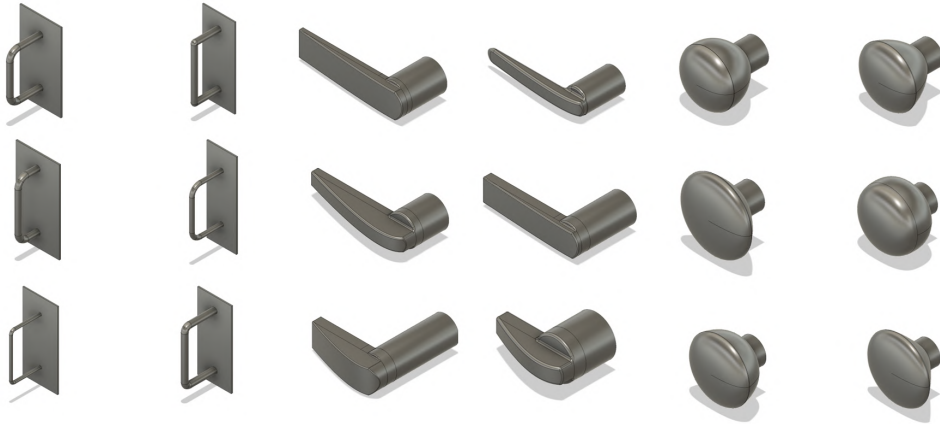


Figure 2: Random Samples from the doorknob distribution. 3000 each for knob are included in the door knob dataset.

this assumption, it is possible to ensemble a variety of simulator environments with different visual or physical properties to generalize to a domain that overlaps the target domain. Rajeswaran et al, 2016 [12] explored making policies more robust in the simple case of the MuJoCo Hopper and Half Cheetah environments using random physical properties. Sadeghi et al, 2017[13] further explored using domain randomization with realistic textures to bridge the reality gap between simulation and the real world. This was done by retexturing CAD models to perform zero shot transfer for drone crash avoidance. Tobin et al, 2017 [14] abandons using realistic textures and instead uses synthetic textures to accomplish fine-grained tasks such as object detection. Sadeghi et al, 2016 [15] used domain randomization to decouple different visual environments from the goal of visual servoing. Peng et al, 2017 [16] used domain randomization to enable zero-shot policy transfer from simulation to the real world. Most recently and perhaps most notably, OpenAI, 2018 [17] performed domain

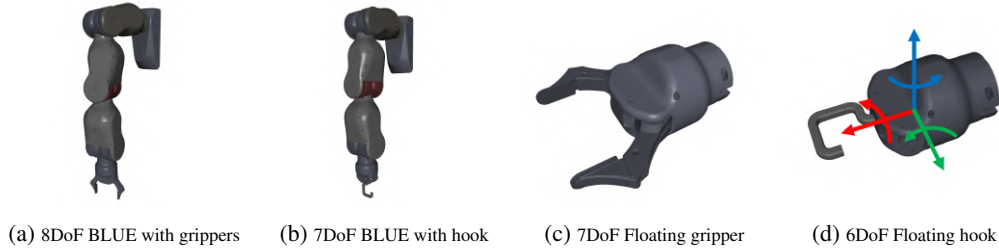


Figure 3: Available arm types

randomization on the 24 DoF shadow hand [18] to transfer a dexterous manipulation task to the real robot.

Various simulator environments have been released with the aim of creating a common benchmark suite [19–21]. In addition there are frameworks based on these popular benchmark libraries to enable more realistic tasks. One such framework is Fan et al’s 2018 [22] SURREAL environment. The SURREAL framework provides a straightforward framework with practical robotic manipulation tasks, but it is not designed to transfer policies between environments. Hence, we are interested in creating an environment that captures the intersection of domain transfer, practical tasks, and realism.

In this paper, we introduce DoorGym, an open-source domain randomized environment for the task of opening various kinds of doors. We also present baseline agents capable of solving this task. We present multiple doorknob types, including round, lever, and pull knobs, with the option of using custom knobs as well. We provide a benchmark environment, and baseline agent capable of opening multiple types of door knobs. We also show that the agent trained in our environment is capable of successfully generalizing to previously unseen environments.

2 DoorGym Door Opening Task Environment Development

Our environment is based on the groundwork laid by the OpenAI Gym framework [21], and the Unity Game Engine¹. Gym offers a variety of discrete as well as continuous environments, including an API for MuJoCo [23]. Combined with wrapper libraries [19, 24], the MuJoCo API, provides the ability to do elaborate physical simulations as shown in [4, 17]. Our environment makes use of Gym and the MuJoCo API to create an easily randomized world by adding an appropriate doorknob STL file and specifying a configuration. While MuJoCo allows us to create a rich environment, it sacrifices visual realism. For this purpose we use Unity to improve the rendering quality. Our simulator also allows the user to modify the simulator reward and observable state easily. We also provide a MuJoCo-Unity plug-in that allows users to access to photorealistic images and various visual effects easily through a Python API. Figure 4 shows the comparison of images by each software.

2.1 Randomized Door-World Generator

We provide a simulation environment devkit that includes the dataset of pull knobs, lever knobs, and round knobs, as well as a script to generate more using Autodesk Fusion360 [25]. Each type of doorknob is parameterized to generate unique instances by changing a few attributes in the CAD models. Sample doorknobs are shown in figure 2. The pull knob is the simplest of the included doorknobs. It does not have a latching mechanism; as such, the pull knob environment can be opened by reaching out, grasping, and pulling. Both the lever and round knob are more complex and involve turning the knob to unlatch the door. The round knob environment is the most difficult to open due to the geometry of the knob. Each knob is saved as several STL files with 3D data, a screenshot of the doorknob, and a JSON file with all doorknob metadata.

We spawn doorknobs in the world randomly using the MuJoCo API. To do this, we generate XML files with all the objects, joints, and physical properties of the world. Though the MuJoCo API supports modifying appearance on the fly, it does not support hot swapping STL models in and out. To deal with this limitation, we generate a new world for every doorknob. The environment itself

¹Unity game engine website: <https://unity3d.com>

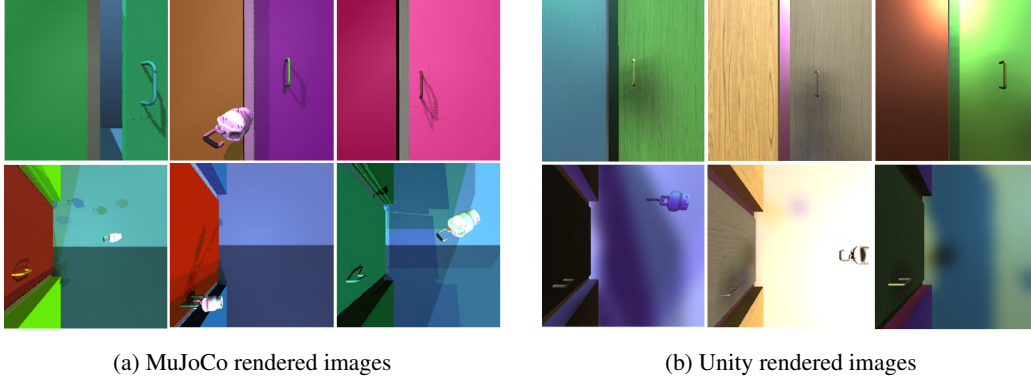


Figure 4: Comparison between the domain randomized images from MuJoCo and Unity. Unity can render realistic shadows and detailed textures.

consists of the robot, door, doorknob, door frame, and wall. All physical properties of the door, doorknob, and robot are randomized in the ranges specified in Table 5. In figure 1, we show examples of randomized environments. At training time, environments will be randomly selected and spawned in the simulator.

2.2 Robot

In our environment, we use the Berkeley BLUE Robot arm [26] by default. We prepare 6 different arm/gripper combinations for BLUE, but provide an interface to add new arms and grippers. The arms that we prepared range from a 6 DoF floating hook to a full 10 DoF arm with gripper and mobile platform. Each arm can be seen in figure 3. The mobile platform variants of the BLUE arm are visually identical, but unlock the x-y plane for an additional two degrees of freedom. Arms 3a and 3b have their configurations expressed by the joint angles, while the floating wrists 3c and 3d are expressed by their x-y-z coordinates and x-y-z rotations.

3 Benchmark

We aim to build a simulation environment that can maintain transfer-ability between domains. We use simple and readily available observations from the environment. An observation consists of the position and the velocity of each joint, as well as the position of the door knob and the position of the end-effector in world coordinates. The position of the doorknob in world coordinates can be obtained directly from the simulator, or using a 256x256 RGB image and vision network. Policy actions use force for linear actuators and torque for rotational actuators, but can be configured to use position control. The size of the action space corresponds to the DoF of each robot. At training time, we use the following reward function for our baseline agent:

$$r_t = -a_0 d_t - a_1 \log(d_t + \alpha) - a_2 o_t - a_3 \sqrt{u_t^2} + a_4 \phi_t + a_5 \psi_t \quad (1)$$

where d_t is the distance between the fingertip of the end-effector and the center coordinate of the doorknob, the second term has been added to encourage higher precision when the agent gets close to the target [27](Levine et al, 2015), α is set to 0.005, o_t is the difference between the current fingertip orientation of the robot and the ideal orientation to hook/grip the doorknob, u_t is the control input to the system, ϕ_t is the angle of the door, and ψ_t is the angle of the door knob. The t subscript indicates the value at time t . Weights are set to $a_0 = 1.0$, $a_1 = 1.0$, $a_2 = 1.0$, $a_3 = 1.0$, $a_4 = 30.0$, and $a_5 = 50.0$. When using the pull knob, door knob angle ψ is ignored.

We define a successful attempt as the robot opening the door at least 0.2 rad within 20 seconds. For attempt i , it can be expressed as

$$\mathbb{1}_i = \begin{cases} 1 & \text{if } \phi > 0.2 \text{ rad and } t < 20 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

This gives rise to two potential evaluation metrics. The average success rate of opening a door, and the average time to open a door. We measure both these metrics over 100 attempts. Average success rate, r_{ASR} , and average time to open a door, r_{AT} can be formalized as

$$r_{ASR} = \frac{1}{100} \sum_{i=1}^{100} \mathbb{1}_i, \quad r_{AT} = \frac{1}{n} \sum_{i=1}^n t_i \quad (3)$$

where t_i is the time to completion for successful attempts, n is the number of successful attempts, and the indicator function is defined as in Eq. 2.

Our environment currently provides a total of 36 possible environment combinations² for evaluation. These environments range in difficulty from the simplest case of a pull knob with floating hook, to the extremely challenging round knob to be opened by BLUE with a gripper.

4 Door opening agent

4.1 Method

To train our baseline agents, we chose an on-policy and off-policy update algorithm. For the on-policy algorithm, Proximal Policy Optimization (PPO) [28] was chosen due to its high stability. Soft Actor Critic (SAC) [29] was chosen as an off-policy algorithm due to its exploration capability and sample efficiency. Details of these algorithms can be found in the appendix in sections 7.1 and 7.2.

4.2 Architecture

Our agent consists of two networks; a vision network to estimate doorknob position, and a policy network to output actions. The architecture of the vision network is shown in figure 10 in the appendix. We jointly predict the x-y-z location of the doorknob using images from the top and front views. The network performs feature extraction on both views using a feature extractor network composed of residual blocks with 4x downsampling. The extracted features are then pooled along the channel dimension using global average pooling. The pooled feature maps are then fed into a regression network which is directly optimized for mean squared error. We regularize the network by adding a cross entropy loss between the heatmaps, and the corresponding ground truth of the doorknob location. To balance the regression and heatmap losses we train with a scaling factor of 10^{-3} for the heatmap loss. We train the network for 35 epochs using the Adam optimizer with a learning rate of 10^{-3} with polynomial learning rate decay and a batch size of 50.

Once we have received an estimate of the doorknob location, we take the difference between our estimate and the end-effector location to produce a direction vector. This direction vector is concatenated with the joint positions and velocities to produce an observation. We then use a network of two fully connected layers with tanh activations to produce the output actions. The structure of our policy network can be seen in figure 11 in the appendix.

4.3 Training Pipeline

The entire training flow is shown in figure 5. A door-world is chosen randomly from the door-world data set and spawned into MuJoCo simulator and Unity. The policy network decides the next action given the observation of the environment. The Gym environment then returns a reward and an observation of the environment for the next time step. During rollouts, observations, actions and rewards are stored in the rollout memory buffer and used for a policy update.

In order to update the policy efficiently while using the randomized data, PPO method runs 8 workers synchronously. During the training time, door-worlds are randomly sampled and fed to each worker after every update. At the start of each episode, the robot position and orientation are randomly initialized. Each episode is 512 time steps (10.2 seconds). The rollout memory buffer of each worker stores 8 episodes for every policy update. During the policy update, rollout memory buffers of all 8 workers are gathered and will be used to update the shared worker policy. We used a learning rate of

²There are 3 grippers, 6 robots, and 2 directions (push/pull)

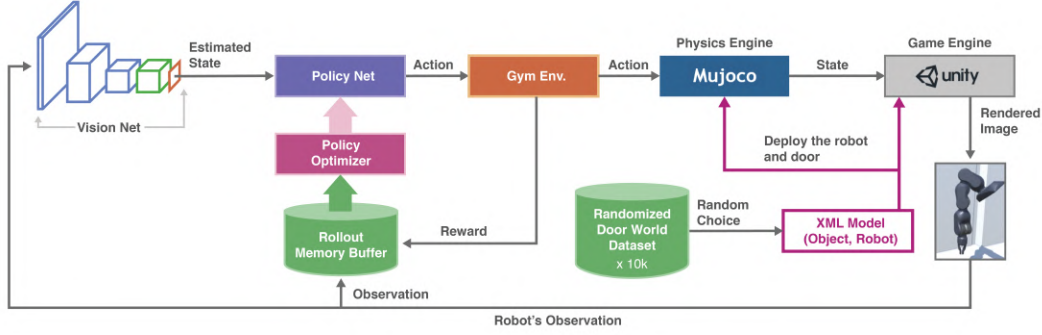


Figure 5: Training Pipeline

10^{-3} for all environments. This PPO training pipeline was structured based on code by Kostrikov, 2018 [30]. In contrast, due to the fact that SAC is an off-policy RL algorithm, it is trained in different way. For each epoch, a single worker performs 10 episodes in randomly sampled door-worlds and stores this data in the rollout memory buffers which has a maximum size of 10^6 time steps. When the policy of worker is updated, an accumulated data in the rollout memory buffers will be used. This training pipeline was structured based on rlkit³.

For PPO training, in order to accelerate the training speed and make the policy converge stably, the policies used with the vision network were pretrained using the door knob position from the simulator, followed a training using the door knob position from the simulator with gaussian noise. The gaussian noise used for pretraining approximates the error introduced by the vision network’s position estimate. The training using vision network for SAC was done without the pretraining process. All policy training hyperparameters are listed in table 6 and table 7 in the appendix.

4.4 Experiments

The following three experiments are conducted to evaluate performance as a representative of the environments. Task 1 has a pull knob environment with floating hook, task 2 has a lever knob with floating hook, and task 3 has a pull knob with the BLUE-with-gripper platform. All experiments are evaluated using the unshaped reward as defined in section 4.1. For each task, we have prepared 100 test door-worlds. The agent has 10.2 seconds to try open the door for each world.

Table 1 shows the results of each task using average success rate and average opening time for different door knob position estimation methods. Each row corresponds to a different knob position estimation method. By using the ground truth of the door knob position from the simulator, the agent has a success rate of over 70% for PPO. In contrast, SAC generally has lower success rates, and specifically it has a 0% success rate for opening the lever knob. During training, SAC shows better exploration capability and its total reward converges faster than PPO, but the result suggests that PPO has better exploitation capabilities than SAC in trade-off of its training speed. PPO’s success rate of all tasks decreases after adding gaussian noise to the door knob position. Tasks that require accurate position information to execute complicated manipulation such as the lever knob suffer more than other tasks. When the door knob position information comes from the vision network, the success rates of both algorithm decrease even more. This result implies that position estimation in the 3D space of the door knob is extremely important for the door opening task. Our vision network achieves ± 3.079 cm accuracy as shown in table 4 in the appendix. Even with this level of accuracy, it was not enough to adapt to open the randomized door. Moreover, even though we train the vision network with the images that has randomly located robot, at the inference time, it is possible that robot is located in right in front of the camera and lessens the accuracy of the position estimation. Figure 6 shows the behavior of the robot in each task in the successful case. Since the door angle reward is weighted more heavily than other rewards, the robot tries to open the door after it hooks or rotates the door knob. Results of all combinations of the environments and the algorithms are shown in table 8, 9 in the appendix.

³<https://github.com/vitchyr/rlkit>

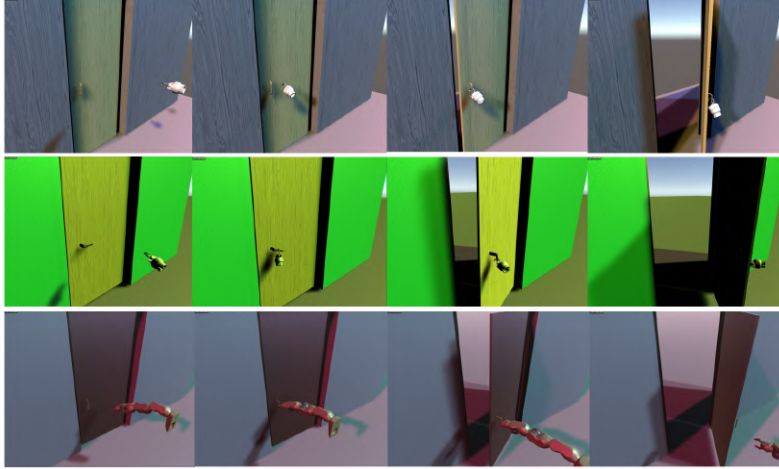


Figure 6: Behavior of successful policies in Task 1, Task 2, and Task 3 respectively.

Table 1: Average Success Ratio and Average Time to Open measured in seconds.

		Task 1		Task 2		Task 3	
		r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}
PPO	Ground Truth Position	0.95	4.51	0.68	4.12	0.71	5.13
	Ground Truth Position + $N(0, \sigma)$	0.78	5.18	0.54	4.11	0.65	4.73
	Vision Network Estimated Position	0.48	6.15	0.29	3.83	0.57	5.12
SAC	Ground Truth Position	0.23	4.29	0	N/A	0.33	5.02
	Ground Truth Position + $N(0, \sigma)$	-	-	-	-	-	-
	Vision Network Estimated Position	0	N/A	0	N/A	0.05	5.06

4.5 Ablation Study

We performed additional experiments to confirm the necessity of DR for policy transferability, and also we show that we were able to transfer our vision network from simulation to real doorknobs. Table 1 below shows the ablation test comparing policies trained with DR and without DR using PPO algorithm. On the left side of Table 2 shows the agent trained in a single environment (env1) achieves 92% of success rate in that environment, but it struggles in randomized environments (50% success). It has a similar trend using the vision network for position estimation (From 46% to 0%).

On the right side of Table 2, the agents are trained under randomized environment and it is showing its robust performance. The agent trained in randomized environments have high success rates in both env1(99%) and randomized env(95%) with the ground truth doorknob position. By conducting the same test using the vision network estimated doorknob position, the randomized trained policy have almost the same performance no matter the test environments are randomized or fixed parameters environments (50% and 48%).

For the vision network, we collected and annotated 20 round knob images in homes and performed inference on the vision network. The network trained with DR achieved an accuracy of 4.95cm in the real world, whereas the network trained without DR performs significantly worse at 19.53cm. Details of these results can be found in appendix chapter 7.4. From these results we can say that DR

Table 2: Success ratio of a policy trained only on a single environment (env1)

Test Environment	Trained on env1						Trained on Randomized env.					
	GT		GT + Noise		Vision		GT		GT + Noise		Vision	
	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}
Tested on env1	0.92	4.71	0.84	4.91	0.46	6.93	0.99	3.78	0.87	4.39	0.50	6.74
Tested on Randomized env.	0.50	4.47	0.52	7.13	0.0	-	0.95	4.51	0.78	5.18	0.48	6.15

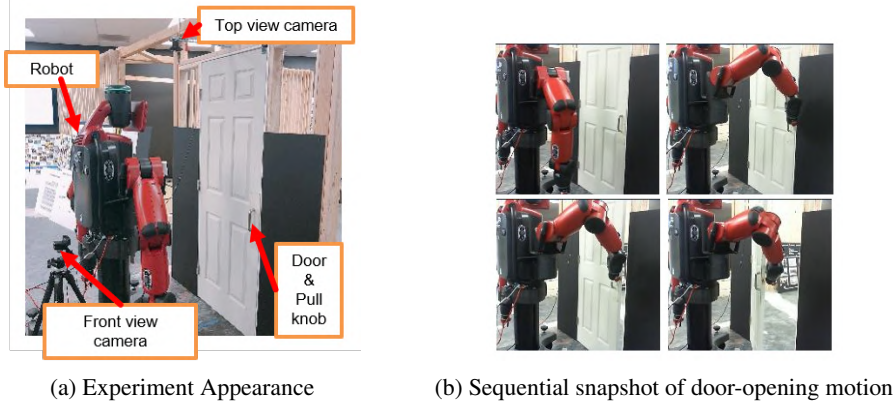


Figure 7: Real world experiment

is required to make both the vision network and policy networks robust enough to work in different domains.

5 Real World Transfer

In order to check the robustness of the trained policy, we performed zero-shot sim-to-real policy transfer. The experiment was executed using the Baxter robot and a pull knob equipped wooden door (Height: 2100mm, Width: 820mm, Weight: 11.3kg). The setup is shown in Figure 7a. The front view camera, top view camera, and robot are placed at the same coordinates as in the simulator. During the experiment, only the right arm of the robot is active since the doorknob is placed on the right side of the door, and the arm starts from the hanging position. The policy used in the experiment is trained purely in simulation using PPO with same hyperparameters as in section 4.4. The location of the doorknob is estimated by the vision network. The control loop frequency is 50Hz.

The experiments were evaluated 100 times, with an experiment episode length of 20.0s. If the robot opened the door more than 0.2 rad during the episode it is counted as a success. The results of using Baxter in both simulation and the real world are shown in Table 3. Result shows that even though the success rate decrease from 70% to 59%, the policy is able to open the door in the real world. However, the behavior of the robot looks unstable, and also the total time to open the door is significantly longer than in simulation.

Table 3: Results of Real World Transfer Experiment

	r_{ASR}	r_{AT}
Simulator	0.70	4.71
Real World	0.59	16.25

6 Conclusion

We presented DoorGym, a simulator environment that supports many degrees of domain randomization on a door opening task with varying degrees of difficulty. As a starting point, we presented a baseline agent based on PPO and SAC, which is capable of opening doors in novel environments. We evaluated the success of this agent by measuring the success rate in 100 attempts, with a success rate of up to 95%. Moreover, we were able to transfer a trained policy from simulation to the real world. Future work will involve expanding the baseline networks as well as incorporating more complicated tasks such as a broader range of doorknobs, locked doors, door knob generalization, and multi-agent scenarios.

References

- [1] E. Klingbeil, A. Saxena, and A. Y. Ng. Learning to open new doors. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2751–2757, 2010.
- [2] R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knodler, L. Li, C. Y. Liu, X. Long, T. Padi, F. Polido, G. G. Tighe, and X. Xinjilefu. What happened at the darpa robotics challenge , and why ? In *None*, 2015.
- [3] B. Nemec, L. Žlajpah, and A. Ude. Door opening by joining reinforcement learning and intelligent control. In *2017 18th International Conference on Advanced Robotics (ICAR)*, 2017.
- [4] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, May 2017. doi: 10.1109/ICRA.2017.7989385.
- [5] Y. Karayiannidis, C. Smith, F. E. Viña, P. Ogren, and D. Kragic. “open sesame!” adaptive force/velocity control for opening unknown doors. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.
- [6] F. Endres, J. Trinkle, and W. Burgard. Learning the dynamics of doors for robotic manipulation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3543–3549, 2013.
- [7] D. Anguelov, D. Koller, E. Parker, and S. Thrun. Detecting and modeling doors with mobile robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 4, pages 3777–3784 Vol.4, April 2004. doi: 10.1109/ROBOT.2004.1308857.
- [8] R. B. Rusu, W. Meeussen, S. Chitta, and M. Beetz. Laser-based perception for door and handle identification. In *2009 International Conference on Advanced Robotics*, pages 1–8, June 2009.
- [9] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644, Sep. 2011. doi: 10.1109/IROS.2011.6095096.
- [10] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *J. Mach. Learn. Res.*, 11:3137–3181, Dec. 2010. ISSN 1532-4435.
- [11] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR*, abs/1709.10087, 2017. URL <http://arxiv.org/abs/1709.10087>.
- [12] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran. Epopt: Learning robust neural network policies using model ensembles. *CoRR*, abs/1610.01283, 2016. URL <http://arxiv.org/abs/1610.01283>.
- [13] F. Sadeghi and S. Levine. (cad)\$^2\$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL <http://arxiv.org/abs/1611.04201>.
- [14] 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. *CoRR*, abs/1703.06907, 2017.
- [15] F. Sadeghi, A. Toshev, E. Jang, and S. Levine. Sim2real view invariant visual servoing by recurrent control. *CoRR*, abs/1712.07642, 2017. URL <http://arxiv.org/abs/1712.07642>.
- [16] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.
- [17] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *CoRR*, 2018. URL <http://arxiv.org/abs/1808.00177>.

- [18] ShadowRobot. Shadowrobot dexterous hand, 2005. URL <https://www.shadowrobot.com/products/dexterous-hand/>.
- [19] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL <http://arxiv.org/abs/1801.00690>.
- [20] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. *CoRR*, abs/1811.02790, 2018. URL [bit.ly/2XJsT9N](https://arxiv.org/abs/1811.02790).
- [21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.
- [22] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 767–782. PMLR, 29–31 Oct 2018. URL <http://proceedings.mlr.press/v87/fan18a.html>.
- [23] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.
- [24] OpenAI. Mujoco-py. <https://github.com/openai/mujoco-py>, 2018.
- [25] AutoDesk, Inc. Autodesk fusion360, 2013. URL <https://autode.sk/2XvQgiL>.
- [26] D. V. Gealy, S. McKinley, B. Yi, P. Wu, P. R. Downey, G. Balke, A. Zhao, M. Guo, R. Thomasson, A. Sinclair, P. Cuellar, Z. McCarthy, and P. Abbeel. Quasi-direct drive for low-cost compliant robotic manipulation. *CoRR*, abs/1904.03815, 2019. URL <http://arxiv.org/abs/1904.03815>.
- [27] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. *CoRR*, abs/1501.05611, 2015. URL <http://arxiv.org/abs/1501.05611>.
- [28] P. D. A. R. O. K. John Schulman, Filip Wolski. Proximal policy optimization algorithms. *CoRR*, 2017. URL <https://arxiv.org/abs/1707.06347>.
- [29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.
- [30] I. Kostrikov. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>, 2018.
- [31] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.
- [32] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [33] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL <http://arxiv.org/abs/1812.05905>.
- [34] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

7 Appendix

7.1 Proximal Policy Optimization (PPO)

Since the door opening task is a complex continuous control problem, a policy gradient method [31] (Williams et al, 1992) is one of a potent method to use. Usually, direct policy search suffers from instability due to high gradient variance. While this can be mitigated by using a larger batch size during training, policy gradient algorithms can be unstable. Proximal Policy Optimization (PPO) is a policy gradient method that deals with this instability by constraining the objective function,

$$\mathcal{L}_{\text{PPO}} = \left[\min \left(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t, \text{clip} \left(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \quad (4)$$

where π and π_{old} are the current and previous policies respectively, \hat{A}_t is the advantage and ϵ is a hyperparameter to clip the value function. Unlike other policy gradient methods, after collecting a new batch of rollout data, PPO continues to optimize the policy multiple times using Importance Sampling. The Importance Sampling ratio, $\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$, can be interpreted as the probability of taking the given action under the current policy π compared to the probability of taking the same action under the old policy that was used to generate the rollout data. The loss encourages the policy to take actions which have a positive advantage (better than average) while clipping discourages changes to the policy that overcompensate. This will effectively prevent the behavior of the current policy π_θ from deviating too far from the previous policy $\pi_{\theta_{\text{old}}}$. PPO trains a value function $V(s_t)$ simultaneously with the policy in a supervised manner to estimate \hat{V}_t . In order to get the target value \hat{V}_t , we use Generalized Advantage Estimation (GAE) [32](Schulman et al, 2016).

7.2 Soft Actor Critic (SAC)

Since PPO is an on-policy algorithm, it inherently suffers from a data inefficiency problem. Soft Actor Critic (SAC) is a family of data efficient off-policy algorithms which will allow for reuse of the already collected data from different scenarios. SAC is based on maximum entropy reinforcement learning, a method to simultaneously maximize both the expected reward and the entropy of the policy. Standard reinforcement algorithm aims to learn the parameters ϕ of some policy $\pi_\phi(a_t|s_t)$ such that the expected sum of rewards is maximized under the trajectory distribution ρ_π . However, the objective function of the maximum entropy algorithm also includes the entropy term $\mathcal{H}(\pi_\phi(\cdot|s_t))$ as a target to maximize. In short, it will converge to the policy that is the most random but still achieves a high reward. The objective function of maximum entropy reinforcement learning framework is shown as follows.

$$\phi^* = \arg \max_{\phi} \sum_{t=1}^T \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi_\phi(\cdot|s_t))] \quad (5)$$

where r is the reward function, and α is a hyperparameter for tuning the trade-off between optimizing for the reward and for the stochasticity of the policy. In our experiments, α will be actively tuned with an automatic entropy tuning function [33](Haarnoja et al, 2018). SAC is based on the soft policy iteration, a general algorithm for learning optimal maximum entropy policies that alternates between policy evaluation and policy improvement under the maximum entropy framework. Although original soft policy iteration only works on the discrete problems (i.e. tabular case), SAC extends the algorithm into continuous domain using function approximators for both the policy π_ϕ (actor) and the Q-function Q_ψ (critic), and optimize these both network alternately. The soft Q-function parameters ψ are optimized to minimize the soft Bellman residual,

$$J_Q(\psi) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\psi(\mathbf{s}_t, \mathbf{a}_t) - \left(r_t + \gamma V_\psi^-(\mathbf{s}_{t+1}) \right) \right)^2 \right] \quad (6)$$

$$V_\psi^-(\mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{a}_{t+1} \sim \pi_\phi} \left[Q_\psi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi_\phi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}) \right] \quad (7)$$

where, \mathcal{D} is the rollout memory buffer, γ is the discount factor, and $\bar{\psi}$ is an exponentially moving average of the value network weights, which is to add stability to the training. Finally, the policy is updated targeting the exponential of the soft Q-function,

$$J_{\pi}(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{a}_t \sim \pi_{\phi}} [\alpha \log (\pi_{\phi} (\mathbf{a}_t | \mathbf{s}_t)) - Q_{\psi} (\mathbf{s}_t, \mathbf{a}_t)] \right] \quad (8)$$

Benefit of the sample efficiency of the off-policy learning algorithm come from the fact that both value estimators and the policy can be trained entirely on off-policy data. Since SAC shows the better robustness and stability than other off-policy algorithm [29], we choose SAC as a second baseline algorithm for DoorGym.

7.3 Realistic Rendering and Post Process using Unity

In order to improve transfer-ability and to increase domain randomization, the vision network can be trained with images that are closer to the real world image distribution. However, MuJoCo’s built-in renderer is not designed to render photo-realistic images. OpenAI, 2018 [17] deal with this problem by using the Unity game engine as a rendering system. Official plugin for Unity parses MuJoCo world files and provides meshes, textures, etc., to Unity, and the simulation state is then synchronized between MuJoCo and Unity through TCP protocol. Our plugin extends the existing MuJoCo-Unity plugin to add additional control over the simulation, with the most noteworthy additions being support for loading worlds at run-time, randomization of material parameters, automatic UV-mapping of detail textures, and the output of semantic segmentation images. Image quality can be further improved with in-engine post-processing effects, such as ambient occlusion shaders which can approximate global lighting, lens distortion, depth of field, and image sensor noise. Post processing samples can be seen Figure 8.

7.4 Vision Network Results

To evaluate the necessity of both Unity’s realistic rendering, and domain randomization for transfer, we perform two sets of experiments. First we evaluate the performance of MuJoCo/Unity models trained with and without domain randomization, both in the current domain, and in unseen domains. Second we compare models trained with data from MuJoCo with data from Unity to assess the transferability of each model.

As can be seen in table 4 training the vision net with no domain randomization and trying to transfer to new domains gives vastly worse performance. MuJoCo gives an error of 0.29 cm in the same environment but explodes to anywhere between 19.89 and 35.82 cm when transferred to a new environment. This is contrasted by the model trained with DR which gives low out of error when tested in unseen MuJoCo environments. Transferring to completely different environments does cause error to go up substantially though, suggesting that DR is not enough alone for generalization. Unity shows a similar pattern of 1.71 cm in domain, with errors between 16.99 and 21.65 cm out of domain. DR once again helps, raising in domain error, but substantially reducing out of domain error.

For our second experiment with the vision network, we compare MuJoCo with DR to Unity with DR. As can be seen in table 4, each model has lower in domain error than when transferred, 0.43 cm as opposed to 3.49 cm when testing in unseen MuJoCo environments, and 10.33 cm compared to 3.41 cm when tested in Unity environments respectively. The gap between transferring MuJoCo to Unity is much larger (-9.90 cm) than Unity to MuJoCo (-0.08 cm). To fairly see the difference between using MuJoCo and Unity, we also transfer our results to real doorknobs, and see significantly lower error at 4.58 cm. We empirically noticed in our experiments that an error of 3 cm makes it difficult to open a door, but more than 5 cm made it impossible for the agent to locate the doorknob. Photos of the doorknobs that were used for real world transfer can be seen in Figure 9.

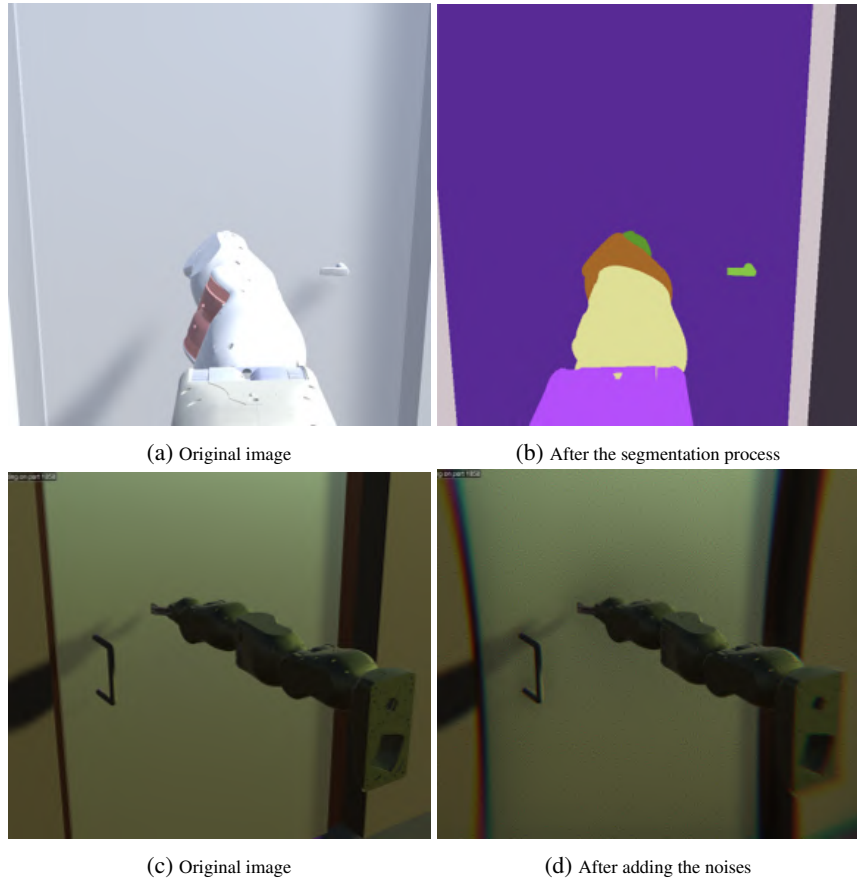


Figure 8: Examples of the post processing that can be apply by Unity. Top row shows the function to make semantic segmentation labels. Bottom row shows the image when the Gaussian noise, camera distortion, and chromatic aberration are added.



Figure 9: Sample knobs that used for sim2real transfer for the vision network.

Table 4: Results of training and then evaluating vision network on MuJoCo and Unity rendered images with and without Domain Randomization (DR). All measurements are in cm. Bolded results are lowest error rate in that domain. Italicized are unseen environments for in the same domain as training.

		In Domain	Transfer to MuJoCo	Transfer to Unity	Transfer to Real
MuJoCo	No DR	0.29	19.89	35.82	23.47
	DR	0.43	0.43	10.33	8.49
Unity	No DR	1.71	16.99	21.65	19.53
	DR	3.41	3.49	3.41	4.58

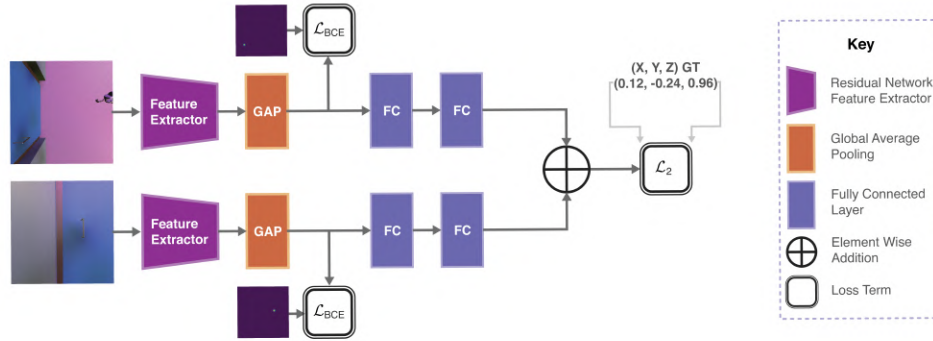


Figure 10: Vision Network Architecture

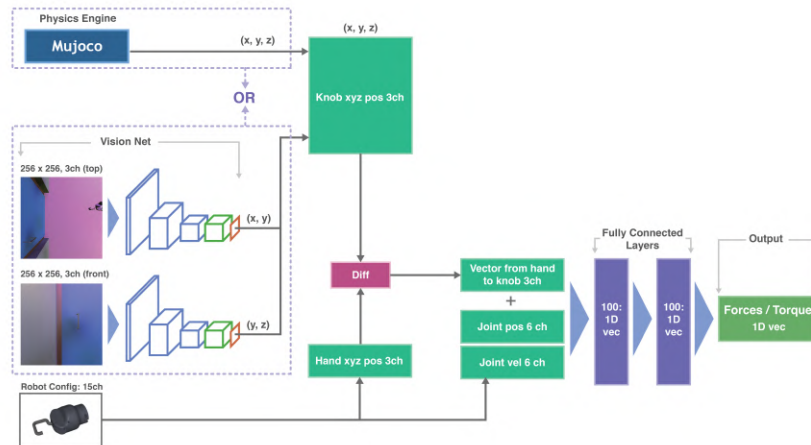


Figure 11: Policy Network Architecture

Table 5: List of Randomized Parameters in the Door World

Randomization Parameters	Scalling Factor	Range	Unit
Door Physical parameters			
Wall Property	wall location	y-axis: Uniform[-200, 200]	mm
Door Frame Joint Property	door frame damper	Uniform[0.1, 0.2]	-
	door frame spring	Uniform[0.1,0.2]	-
	door frame frictionloss	Uniform[0, 1]	-
Door Property	door height	Uniform[2000, 2500]	mm
	door width	Uniform[800, 1200]	mm
	door thickness	Uniform[20, 30]	mm
	knob height	Uniform[950, 1050]	mm
	knob horizontal		
	location ratio	Uniform[0.10, 0.20]	-
	door mass	Uniform[22.4, 76.5]	kg (Based on MDF density)
Knob Door Joint Property	hinge position	left hinge/right hinge	-
	opening direction	push/pull	-
	knob door damper	Uniform[0.1, 0.2]	-
	knob door spring	Uniform[0.1, 0.15]	-
	knob door frictionloss	Uniform[0, 1]	-
	knob rot range	Uniform[75, 80]	degree
	knob mass	Uniform[4, 7]	fg
Knob Property	knob surface friction	Uniform[0.50, 1.00]	-
Robot Physical Parameters			
Robot Property	arm joint damping	Uniform[0.1, 0.3]	-
Vision Parameters			
Lighting	light number	Uniform[2-6]	-
	light diffuse	Uniform[0.0, 1.0]	RGBA
	light position	x:Uniform[0.0, 5]	m
		y:Uniform[-5, 5]	
		z:Uniform[3, 7]	
Wall Material	light direction	x:Uniform[-0.5, 0.5]	rad
		y:Uniform[-0.5, 5.0]	
		z:Uniform[-0.5, -0.25]	
		wall shininess	Uniform[0.01, 0.50]
		wall specular	Uniform[0.01, 0.50]
Frame Material	wall rgb1	Uniform[0.0, 1.0]	RGBA
	frame shininess	Uniform[0.01, 0.70]	-
	frame specular	Uniform[0.01, 0.80]	-
Door Material	frame rgb1	Uniform[0.0, 1.0]	RGBA
	door shininess	Uniform[0.01, 0.30]	-
	door specular	Uniform[0.01, 0.80]	-
	door rgb1	Uniform[0.0, 1.0]	RGBA
	door rgb2	Uniform[0.0, 1.0]	RGBA
Doorknob Material	knob shininess	Uniform[0.50, 1.00]	-
	knob specular	Uniform[0.80, 1.00]	-
	knob rgb1	Uniform[0.0, 1.0]	RGBA
Robot Material	robot shininess	Uniform[0.01, 0.70]	-
	robot specular	Uniform[0.01, 0.80]	-
	robot rgb	Uniform[0.0, 1.0]	RGBA

Table 6: Hyperparameters for PPO

Hyperparameter	Value
Hardware Configuration	8 NVIDIA GeForce GTX TITAN-X GPU 8 CPU cores
Optimizer	Adam [34](Kingma et al, 2014)
Learning Rate	0.001
Multi-worker number	8
PPO mini-batch size	256
PPO clipping ratio	0.2
Max Grad Norm	0.5
Discount Ratio Gamma	0.99
GAE lambda	0.95
Entropy-coef	0.0
Action-loss-coef	1.0
Value-loss-coef	0.5

Table 7: Hyperparameters for SAC

Hyperparameter	Value
Hardware Configuration	NVIDIA GeForce GTX TITAN-X GPU 8 CPU cores
Optimizer	Adam [34]
Policy Learning Rate	0.001
Q-function Learning Rate	0.001
Discount Ratio Gamma	0.99
Rollout Memory Buffer Size	10^6
Target update Period	1
Target smoothing coef	0.005
Automatic Entorpy Tuning	True

Table 8: Result of experiments using PPO

Algorithm	Open Direction	Robot type	Knob type	GT		GT + Noise		Vision	
				r_{ASR}	r_{AT}	r_{ASR}	r_{AT}	r_{ASR}	r_{AT}
PPO	pull	hook	pull	0.95	3.47	0.99	3.20	0.79	3.73
			lever	0	-	0	-	0	-
			round	0.51	5.56	0.47	5.20	0.26	5.47
		gripper	pull	0.68	4.18	0.59	4.12	0.62	3.73
			lever	0	-	0	-	0	-
			round	0	-	0	-	0	-
		floating hook	pull	0.95	4.51	0.78	5.18	0.48	6.15
			lever	0.68	4.12	0.54	4.11	0.29	3.83
			round	0	-	0	-	0	-
		floating gripper	pull	0.93	4.81	0.86	5.49	0.55	5.79
			lever	0	-	0	-	0	-
			round	0	-	0	-	0	-
		mobile hook	pull	0.84	4.41	0.8	4.88	0.66	5.36
			lever	0.88	5.82	0.67	5.46	0.37	5.51
			round	0	-	0	-	0	-
		mobile gripper	pull	0.71	5.13	0.65	4.73	0.57	5.12
			lever	0	-	0	-	0	-
			round	0	-	0	-	0	-
	push	hook	pull	1.00	2.01	1.00	1.53	1.00	1.52
			lever	0.11	6.94	0.04	4.92	0	-
			round	0.11	7.45	0.14	6.65	0	-
		gripper	pull	1.00	1.20	1.00	1.01	1.00	1.11
			lever	0.10	5.04	0.21	5.71	0.21	4.29
			round	0.04	7.43	0	-	0	-
		floating hook	pull	0.97	2.02	1.00	2.22	1.00	2.14
			lever	0.37	5.38	0	-	0	-
			round	0.02	2.63	0.02	4.75	0.02	4.14
		floating gripper	pull	0.98	2.37	1.00	2.19	1.00	2.31
			lever	0.66	6.99	0.16	7.12	0.05	8.15
			round	0	-	0.01	2.2	0	-
		mobile hook	pull	1.00	2.71	1.00	1.84	1.00	1.64
			lever	0.44	5.22	0.45	4.62	0.39	4.11
			round	0.06	7.70	0.12	6.29	0.06	5.73
		mobile gripper	pull	0.98	1.68	1.00	1.38	1.00	1.42
			lever	0.56	5.67	0.63	5.15	0.41	5.23
			round	0.25	8.47	0.22	6.96	0.08	8.66

Table 9: Result of experiments using SAC

Algorithm	Open Direction	Robot type	Knob type	GT		Vision	
				r_{ASR}	r_{AT}	r_{ASR}	r_{AT}
SAC	pull	hook	pull	0.62	3.21	0.25	2.93
			lever	0	-	0	-
			round	0	-	0	-
		gripper	pull	0.58	3.42	0.07	2.36
			lever	0	-	0	-
			round	0.01	7.08	0	-
		floating hook	pull	0.23	4.29	0	-
			lever	0	-	0	-
			round	0.03	7.21	0	-
		floating gripper	pull	0.47	4.98	0	-
			lever	0.19	7.99	0	-
			round	0	-	0	-
		mobile hook	pull	0.33	5.02	0.12	3.72
			lever	0.05	5.93	0	-
			round	0.03	5.56	0	-
		mobile gripper	pull	0.47	4.03	0.05	5.06
			lever	0.13	6.39	0.01	6.94
			round	0	-	0	-
	push	hook	pull	1.00	1.24	0.99	0.76
			lever	0.09	5.71	0.01	3.12
			round	0.04	3.56	0.01	9.66
		gripper	pull	0.99	1.04	0.99	0.80
			lever	0	-	0	-
			round	0.01	7.46	0	-
		floating hook	pull	0.99	2.22	0.99	1.95
			lever	0.02	8.62	0	-
			round	0.02	7.2	0	-
		floating gripper	pull	0.98	2.04	0.98	2.15
			lever	0.43	6.68	0	-
			round	0	-	0	-
		mobile hook	pull	1.00	1.26	0.95	1.23
			lever	0.23	6.03	0.05	6.89
			round	0.03	2.92	0.03	2.23
		mobile gripper	pull	0.97	1.12	0.98	1.03
			lever	0.21	6.74	0.12	5.07
			round	0.01	9.98	0.02	9.25