Active Vision for Robotics

Siddharth Ancha
(sancha)
sancha@andrew.cmu.edu

Jagjeet Singh
(jagjeets)
jagjeets@andrew.cmu.edu

Abstract

Modern computer vision is all about learning from non-interactive data. But humans learn by actively interacting with their environment. If we want robots to do tasks in messy, cluttered environments, we need to let the robot control its own camera. To this end, we first design an OpenAI Gym simulation environment, using the PyBullet physics engine. We attach a camera at the end of the KUKA IIWA robotic arm. We train the robot to find and focus on target objects present in the environment. We use an FCN architecture to process visual images, and design low dimensional state and action spaces and dense rewards to learn a visuomotor policy using DQN in less than 100 episodes. The learned agent can explore its environment to find the target object, and maintain focus on the full visible extent of the object. Our learned agent is also robust to perturbations. Code will be made publicly available. Videos of our results are available here[1][2].

1 Introduction

In modern computer vision, there is a lot of emphasis on learning from data. However, learning is done largely from static images (eg. ImageNet) or videos (eg. Youtube 8M). In these scenarios, the learning agent has no control over which images/scenes it gets to look at.

However, this is not how humans learn. Humans can look around objects from different viewpoints, rotate and manipulate objects to get better views, do visual navigation in 3D for better perception etc. This suggests that we need to combine vision with control for optimal visual learning.

Robot manipulation is the task of robots perceiving, grasping and manipulating objects like humans do. However, for most manipulation tasks, the robot's perceptual device, i.e. the camera, is held in a static location and pose. The robot is expected to find objects and grasp them, using a video stream from a fixed viewpoint. This is equivalent to a person constrained to not move his/her head, or even the eyes, while asked to perform complex tasks. We believe that by providing the robot with the ability to control its own camera and "look around" the scene, it can actively perceive its environment and better carry out tasks like manipulation.

As a step in this direction of active robot vision, we propose the task of a robot (to whose arm a camera is attached) learning to control its camera arm and moving it around in order to efficiently locate objects and keep them in sight. We set up the following task, which we believe is a good representative of the aforementioned problem. We consider a variety of objects placed in front of a robot. The selection of objects and their positions are random. During test time, the robot is instructed to locate an object, which is identified by providing a single image of the object. We call this the *target image*. The object may not be directly observable from the initial camera position because the object may be occluded by other objects and/or may be present in a different viewpoint. The robot would need to look around its environment to find the object. It needs to learn a vision-driven policy such that, from the current camera image, it should take an action and move its arm (thereby

[1] https://youtu.be/eBbDKi_3KXI
[2] https://youtu.be/_dxg1hFzVn4

generating new camera images) in a way that it eventually reaches the target object in minimum possible time.

The paper is organized as follows. We briefly talk about related work. We then describe the simulation environment that we built for this task. We then describe our active vision system, which consists of two parts – 1) vision and 2) control. We discuss experiments that we conducted after merging the two components, and the results that we obtained. We conclude with future directions of research.

2 Related Work

There has been significant progress made in letting robots take actions based on their understanding of their visual environments. Lawson et. al. proposed an active search methodology to manipulate occluded objects in [17]. Some works incorporate active vision in belief space of probability distributions over configurations of robots and objects [8]. The robot uses active learning on the belief map to find regions of maximum uncertainty; these regions are selected to be explored in order to manipulate occluded objects.

Reinforcement Learning can be broadly categorized into 1) Q-learning algorithms and 2) policy gradient (PG) algorithms. Different problems vary in the complexity of their policies and action-value functions. When the policy is simpler and easier to approximate, then a PG-based method will typically learn faster and learn a superior policy. Volodymyr et. al. proposed the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning [16]. Lillicrap et. al. proposed an actor-critic, model-free algorithm based on deterministic policy gradients that can operate over continuous action spaces [9]. Getting sparse rewards is one of the main challenges faced by any RL task. Hindsight Experience Replay proposed by [1] allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering.

Semantic segmentation has been an important task in the vision community. Earlier approaches used Texton forest [6] and random forest based classifiers for semantic segmentation. In 2014, fully convolutional networks (FCN) by Long et al. [7] popularized CNN architectures for dense predictions without any fully connected layers. This allowed segmentation maps to be generated for images of arbitrary size and was also much faster compared to patch classification approaches. Since CNNs use pooling layers, they lose a lot of spatial information which is important for semantic segmentation. Two different classes of architectures are popularly used to tackle this issue. First is the *encoder-decoder* architecture. The encoder gradually reduces the spatial dimension with pooling layers and the decoder gradually recovers object details and spatial dimension. Shortcut connections from encoder to decoder are commonly used to help the decoder recover object details more reliably. U-Net [12] is a popular architecture from this class. The second class encompasses architectures which use dilated/atrous convolutions [5] and do away with pooling layers altogether. Recent state-of-the-art models like SegNet [2], PSPNet [19] and DeepLab [3] are all based upon an underlying FCN architecture. [15] presented one-shot learning to perform dense pixel-level prediction on a test image for a new semantic class.

3 Simulation environment

We simulate an environment in the PyBullet[4] simulator. We use a KUKA robot arm (in simulation). We mount a camera at the end of the robot's movable arm. We also generate a collection of objects on the floor at random positions.

We then create an OpenAI gym environment that uses PyBullet for backend physics and rendering. We define the necessary gym functions (step, reset etc.) appropriately for the active vision task.

Figure 1 shows a snapshot of our environment in the PyBullet physics simulator. The environment consists of a KUKA robot and multiple objects around it. The robot has seven moving parts which can be controlled by applying torques to each link of the robot. A camera is fitted on the end of the KUKA arm.

Three camera views are shown in the left pane. The first view is the RGB input to the robot (what it sees through its camera). This will be a representative of the current state. The second is the depth map, which is not used anywhere in the task. The third view is the ground truth segmentation mask

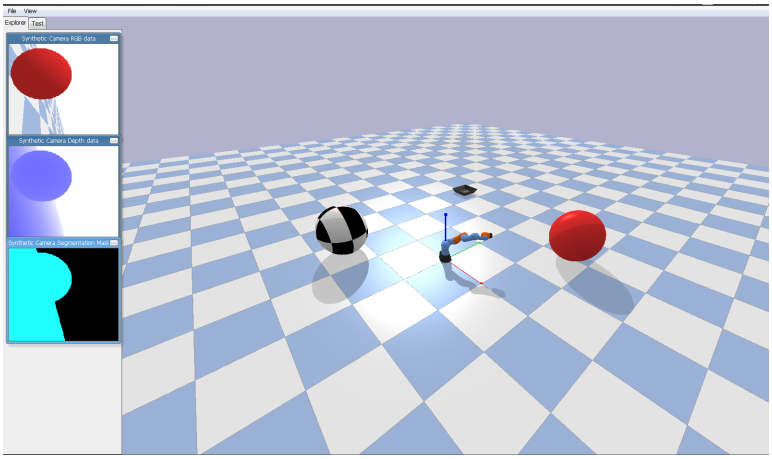

Figure 1: Our simulation environment in PyBullet [1]. KUKA robot looking at one of the objects. In left pane (a) RGB camera image (input to robot), (b) depth map, (c) segmentation mask of all objects, useful for defining loss.

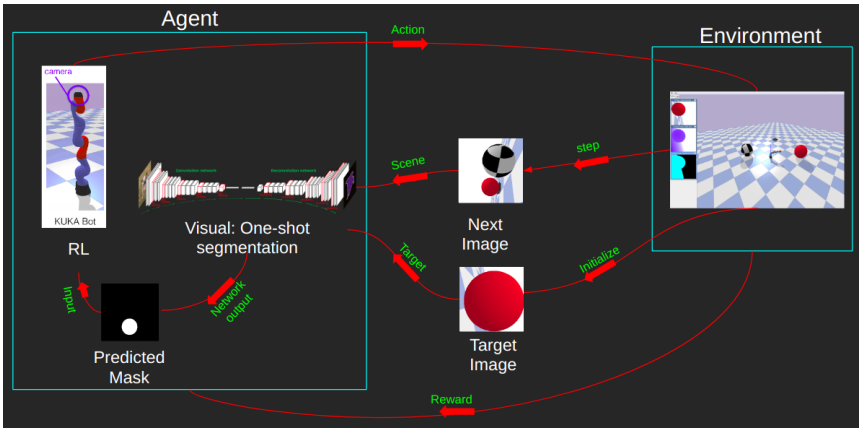

Figure 2: Flowchart of the Active Vision setup

of the scene (each object is shown with a different color). This map is used by the environment to compute rewards.

4 Methodology

Fig 2 shows a high-level flowchart of our system. It consists of the agent and the environment. The learning agent in our case has two components: 1) vision and 2) RL (control). The environment provides visual input to the vision component; this input consists of two images, the current scene image and target object image. The two images together form the current state of the environment. The vision component takes as input the two images and predicts a segmentation mask of the target object in the current scene image. The RL component then inputs this mask and maps it to actions. The action taken by the agent changes the state and the environment, which in turn gives a reward. We discuss the state space, action space and rewards in section 4.2.

4.1 Vision component: one-shot segmentation

The vision component receives two images from the environment. The first is the current scene RGB image as captured by the camera attached to the robot's end-effector. The second is an RGB image of the target object; it is an 'idealistic' image of the target, which is zoomed-in, centered, and is un-occluded. The goal of the vision component is to segment the target object in the current scene

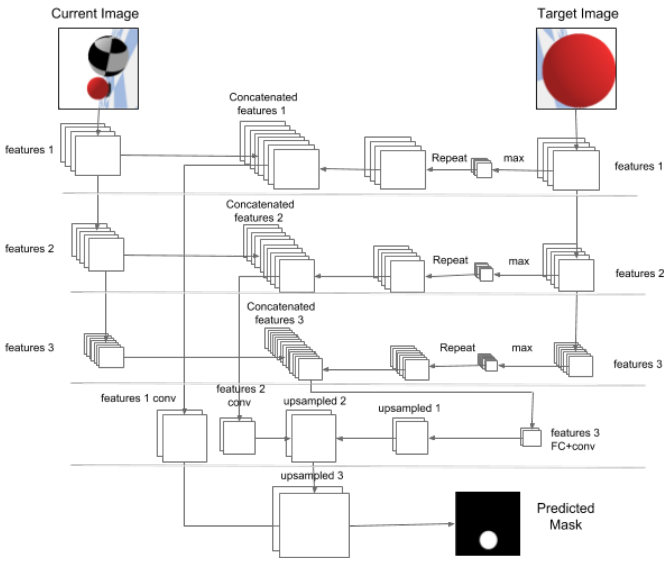

Figure 3: Proposed architecture for one-shot segmentation.

image. This is a one-shot formulation of the segmentation task wherein the target image is provided only at test time; the hope is that the one-shot segmentation network can segment novel, unseen objects using the *target image* at test time.

Fully Convolutional Networks (FCN) [7] by Long et. al. has become the most popular approach for dense prediction tasks. There have been many advancements which have built upon the FCN model and lead to improvements in performance. We, however, are dealing with images rendered in simulation; these tend to be rather simplistic because they lack the variation and diversity of natural images. Hence we found that the vanilla FCN architecture performed reasonably well, and we didn't have to use more advanced models like U-Net, SegNet and DeepLab.

Figure 3 shows the proposed architecture for one-shot segmentation. The network takes two images as input – the current scene image and an enlarged view of the target object. We use a pre-trained VGG-16 network to obtain convolutional features. The VGG features are used after the 17^{th}, 24^{th} and 30^{th} layers, which correspond to *features1*, *features2* and *features3* respectively in the figure. At each level, the features of the target image are first max-pooled, for every channel, such that the spacial dimension reduces to 1x1. We assume that these pooled features are able to store enough information about the target object and can be used to localize that object in any image. The target object features obtained after pooling are *tiled* over the corresponding features of the input image, and concatenated. The same process of pooling and concatenation is repeated at three different levels, so that we retain enough local and global information. The concatenated features after the final layer are sent through some more convolution and fully connected (using 1x1 convolutions) layers. Upsampling using transposed convolutions is then performed to gradually restore the spatial resolution. During upsampling, we also add skip connections to replenish the layer with low level features, which were captured by earlier layers. The final output has two channels which correspond to the target object and background classes. We use a simple pixelwise cross-entropy loss. We train for 30 epochs after which the model starts to overfit. Figure 4 shows the loss curves and evaluation metrics. The best accuracy achieved was 99.85% and the best mean IoU (intersection over union) was 98.7%. We attribute such high values to two reasons. First, the input images were simplistic because they were rendered by a simulator. Second, the objects used during validation were similar to the objects the network was trained. Furthermore, we also trained using AlexNet instead of VGG and found the performance to be satisfactory.

4.2 RL based control component

We use reinforcement learning to teach the robot to move its actuators such that the robot arm moves closer to the desired object. We use a multi-layer perceptron that maps the input space (and features/bounding box detections computed by the vision part) to actions.

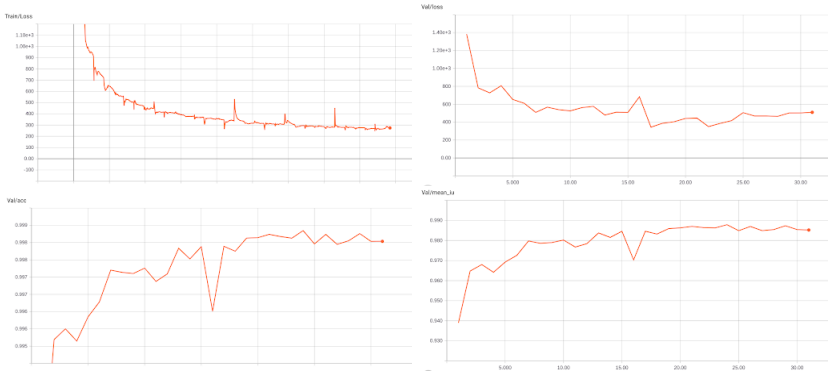

Figure 4: Top Left: training loss vs number of updates, Top Right: validation loss vs epoch, Bottom Left: validation accuracy vs epoch, Bottom Right: validation mean IoU vs epoch.

Reinforcement learning usually works well when the state and action spaces are low dimensional, and when the rewards are dense. So, we design the elements of our environment to satify the above properties.

4.2.1 State Space

We want our state space to be low dimensional. The two input images, i.e. the current scene image and the target image are processed by our vision component; the vision part segments out the target object in the scene image. Segmentation output, however, is high dimensional. For 256×256 images, the segmentation output is a binary vector of size $256 \times 256 = 65,536$. In order to reduce the dimension, we use a trick where we convert the segmentation mask to bounding boxes. A bounding box is a rectangle that just fits a given segment. It is represented by four values (x, y, w, h); the center (x, y), height h, and width w. Hence we reduce the input space to four dimensions i.e. $\mathbb{R}^4$. Note that for simplicity, we are not using any image features apart from the bounding box detection as input to the control part.

4.2.2 Action Space

The eventual actions are the torques that are to be applied to every motor of the robot. There are seven links of the KUKA robot, and hence this action space is $\mathbb{R}^7$.

Continuous action spaces are hard for RL to learn over, even if there exist RL algorithms that can be used with continuous actions spaces (ACKTR[18], DDPG[9], TRPO[13], PPO[14], HER[1]). We discretize our action space by asking the robot to only specify which direction it wants to move its camera end-effector i.e. whether it wants it to go up / down / left / right / top / bottom. This reduces the action space to a set of six discrete actions.

But how do we map directions of movement, which are high level actions, to low-level motor torques? To that end, we use *inverse kinematics* of the robot. Given a dynamics model of the robot (which is available by default in a simulator), inverse-kinematics [11] computes/plans the trajectory of link orientations that can lead the robot from the current state to a final state; the final state can be described in many ways, one such way being the final position of the end-effector. From the link orientations, we can deduce the the direction of torques to be applied; the magnitude is then capped by maximum torque limits that the robot can allow.

4.2.3 Reward

Since the objective of our agent is to locate and zoom into the target object in the scene, a natural reward is the amount of "visible extent" of the target object in the current camera input. More precisely, we use the proportion of the current camera image occupied by the target object as the reward signal. This will encourage the agent to move closer to the object in order to maximize the area occupied by the object. The reward signal can be generated from ground-truth segmentation/detection, which is available in simulation.

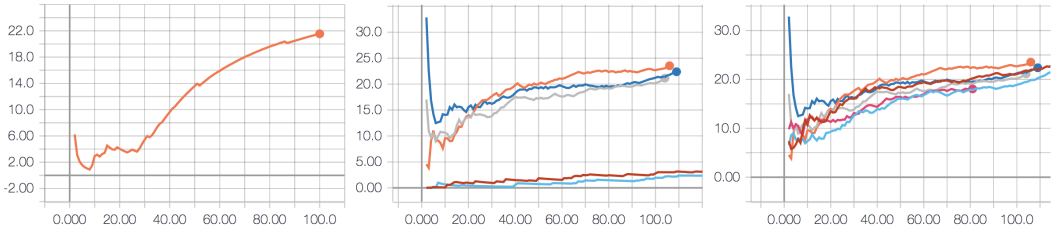

Figure 5: Mean rewards for (a) object in fixed position (b) object in random position, top: using exploration subroutine, bottom: not using subroutine (c) object in random position, using and not using link orientations as part of state.

Note that this signal is a *dense* reward. At every state of the simulation i.e. every camera input, the agent can precisely know how much of the target object is visible. This is especially true when the object is at least partially visible. After choosing one of the six directional actions (see section 4.2.2), the agent gets immediate feedback as to whether the action leads to an increase or decrease in visibility of the object. Dense rewards have the advantage of making reinforcement learning more efficient.

4.2.4 RL training algorithm

In the previous sub-sections, we described how we were able to reduce the dimensionality of the state and action spaces, make the action space discrete, and design a dense reward for the task. Because of these simplifications, we can use a relatively simplistic deep reinforcement learning algorithm for our active vision task.

We implement Deep Q-Learning [10] with the aforementioned state space, action space and reward specified in our custom gym environment. We use a simple multilayer perceptron with a single hidden layer with 20 hidden units, as the value function approximator for DQN. We use a learning rate of 0.001 and a buffer size of $50,000$. We train for only a 100 episodes. We decay the exploration parameter ϵ starting from 0.1 in the first episode to 0.02 by the final (100^{th}) episode.

5 Experiments & results

In the previous section, we described the two main components of our agent – 1. vision and 2. control. We now combine the two components into a single agent. The vision component receives RGB images of the scene and target from the environment, and maps them to bounding boxes. The control part take the bounding boxes as input and maps them to robot actions. The vision part was trained on static images generated from the environment (see section 4.1). We found the vision system to be quite accurate and reliable, over a range of objects and positions. So, after combining the two components, we fix the vision part, and only train the control/RL part. We perform experiments for the scenarios where the location of the object is fixed and where the location is randomized. We also show that our agent learns to be robust to perturbations.

5.1 Object placed at fixed position

As a first experiment, we place the object at a fixed position and try to learn a policy to locate the object using Deep Q-Learning. The arm always points to the sky in the beginning of every episode. In general, the agent has to learn to look around and locate the target object. But this is a simplified case where it only needs to learn to find the fixed position of the object. We find that this is easily achieved by the agent. The agent executes the task almost perfectly; it navigates towards the fixed position and then keeps the target object in focus while maintaining the object's entire extent in the camera view. The training curve for mean reward during this task is shown in figure 5(a).

5.2 Object placed at random position

We then consider the more realistic setting where the object is at a random position around the robot. Without any changes to the RL algorithm, the agent completely fails to locate the random object.

We found that there is a fundamental issue why the previous algorithm fails. Our policy is deterministic. Which means that the robot will explore the space via a deterministic trajectory (at least initially) to find the randomly placed object. A good exploration trajectory should maximize the area the robot looks at, while avoiding traversing the same region multiple times. This is a difficult policy to learn; however it is easy to specify. We program an exploration policy which brings the camera at an altitude of about half the height of the robot. Then we rotate the camera about the axis of the robot so that it gets to look at the entire space.

We run the exploration policy and let the robot learn what actions to take when the exploration policy finds the object (i.e. the vision component tells the robot that at least a small fraction of the target object is visible). This greatly simplifies the problem as the robot can efficiently find the randomly placed object before trying to focus in on it. A demonstration video is available[3]. The benefit of using the exploration policy is shown in figure 5(b). The top curves show the mean rewards when using the exploration policy, while the bottom curves are when the robot doesn't use any exploration policy. The gap between the curves is evidence that the exploration policy indeed helps in learning.

We run another experiment to verify whether adding the positions and orientations of the links of the robot to the state, helps in learning. Figure 5(c) shows three curves for which link information was added to the state space, and three curves for which it wasn't. Since no set of curves seems to be significantly better than the other, we conclude that adding link information doesn't really help. Because the robot is directly controlling the direction the camera should move, we hypothesize that the camera image is sufficient information to learn a good policy.

5.3 Robustness to perturbations

We observed an interesting property that our robot agent had learned while focusing on the target object – robustness to perturbations. When the robotic camera finds the object and focuses on it, we tried applying "perturbations", i.e. we tried to move the camera away from the object using externally applied force. After we stop applying this perturbation, the robot instantly 'snaps back' and starts focusing on the camera again. We tried this for small and large perturbations, and for different random directions. Every time this happened, the camera manages to move back to its original position and bring the entirety of the object back into view. A video of this is available here[4]. This tells us that the robot has learned to infer which direction to move to find the whole object, given only the partially visible object. Such robustness to perturbations is a useful property for any active vision system.

6 Conclusion

We set up an active vision task by creating an OpenAI gym environment using the PyBullet physics simulator. A camera is attached at the end of a robotic arm, and it is supposed to find and zoom into a target object present in its surroundings. We design a two-component agent where the first component i.e. the vision part, takes the scene and target images as input and segments the target object in the scene, using a fully-convolutional network architecture. The segmentation is used by the control component to map to robot actions which make the robotic arm move and locate the object. We reduce the dimensionality of the state and action spaces of the control agent, as well as define a dense reward function, to enable learning of a good policy in relatively few ($\approx$100) episodes. This is achieved even when the object is randomly placed in the environment. Additionally, we show that the learned agent is robust to external perturbations.

For future work, we want to train and test on a more diverse collection of objects. We also want to design environments with occluded objects, and learn a visuomotor policy that handles occlusions. In the longer term, we want to work on transfer of active-vision policies learned from simulation to real world robots. We also want to experiment with augmenting the agent with memory (sequence models) so that it can automatically learn good exploration policies.

[3] https://youtu.be/eBbDKi_3KXI
[4] https://youtu.be/_dxg1hFzVn4

References

[1] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5055–5065, 2017.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[4] Y. B. Erwin Coumans. PyBullet, a python module for physics simulation in robotics, games and machine learning. 2016-17. Available at `http://pybullet.org/`.

[5] Vladlen Koltun Fisher Yu. Multi-scale context aggregation by dilated convolutions. In *Computer Vision and Pattern Recognition*. IEEE, 2016.

[6] Matthew Johnson Jamie Shotton and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer Vision and Pattern Recognition*. IEEE, 2008.

[7] Trevor Darrell Jonathan Long, Evan Shelhamer. Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recognition*. IEEE, 2015.

[8] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[12] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. In *Computer Vision and Pattern Recognition*. IEEE, 2015.

[13] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] Amirreza Shaban, Shray Bansal, Zhen Liu, Irfan Essa, and Byron Boots. One-shot learning for semantic segmentation. *CoRR*, abs/1709.03410, 2017.

[16] David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Volodymyr Mnih, Koray Kavukcuoglu and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Conference on Neural Information Processing Systems*. IEEE, 2013.

[17] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2814–2819. IEEE, 2013.

[18] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.

[19] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network.