Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning

Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, Thomas Funkhouser

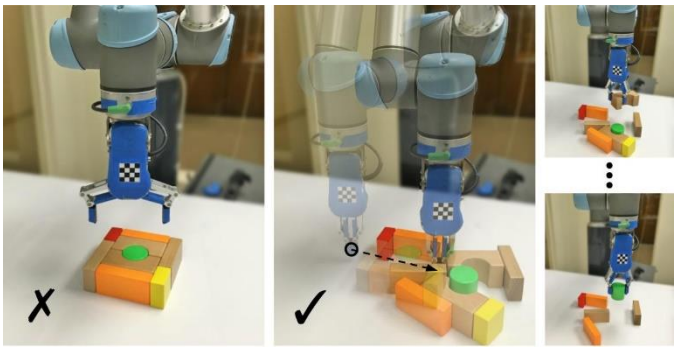
***Abstract*—In this work, we demonstrate that it is possible to discover and learn these synergies from scratch through model-free deep reinforcement learning. Our method involves training two fully convolutional networks that map from visual observations to actions: one infers the utility of pushes for a dense pixel-wise sampling of end effector orientations and locations, while the other does the same for grasping. Both networks are trained jointly in a Q-learning framework and are entirely self-supervised by trial and error, where rewards are provided from successful grasps. In this way, our policy learns pushing motions that enable future grasps, while learning grasps that can leverage past pushes. Qualitative results (videos), code, pre-trained models, and simulation environments reference.**

I. Introduction

Skilled manipulation benefits from the synergies between non-prehensile (e.g., pushing) and prehensile (e.g., grasping) actions: pushing can help rearrange cluttered objects to make space for arms and fingers (see figure); likewise, grasping can help displace objects to make pushing movements more precise and collision-free.

Although considerable research has been devoted to both push and grasp planning, they have been predominantly studied in isolation. Combining pushing and grasping policies for sequential manipulation is a relatively unexplored problem. Pushing is traditionally studied for the task of precisely controlling the pose of an object. However, in many of the synergies between pushing and grasping, pushing plays a loosely defined role, e.g., separating two objects, making space in a particular area, or breaking up a cluster of objects. These goals are difficult to define or reward for model-based or data-driven approaches.

Many recent successful approaches to learning grasping policies, maximize affordance metrics learned from experience or induced by grasp stability metrics.



In this work, we propose to discover and learn synergies between pushing and grasping from experience through model-free deep reinforcement learning (in particular, Q-learning). The key aspects of our system are:

- We learn joint pushing and grasping policies through self-supervised trial and error. Pushing actions are useful only if, in time, enable grasping. This contrasts with prior approaches that define heuristics or hard-coded objectives for pushing motions.
- We train our policies end-to-end with a deep network that takes in visual observations and outputs expected return (i.e. in the form of Q values) for potential pushing and grasping actions. The joint policy then chooses the action with the highest Q value – i.e., the one that maximizes the expected success of current/future grasps. This is in contrast to explicitly perceiving individual objects and planning actions on them based on hand-designed features.

This formulation enables our system to execute complex sequential manipulations (with pushing and grasping) of objects in unstructured picking scenarios and generalizes to novel objects (unseen in training).

II. Problemr Statement

Our problem is to formulate the pushing-for-grasping task as a Markov decision process. At any given state at a certain time, the robot chooses and executes an action according to a specific policy, then transitions to a new state and receives an immediate corresponding reward.

The goal of our robotic reinforcement learning problem is to optimize the policy that maximizes the expected sum of future rewards, from the given certain time to the infinite future.

We will investigate the use of off-policy Q-learning to train a greedy deterministic policy that chooses actions by maximizing the action-value function (i.e., Q-function), which measures the expected reward of acting at the certain time. Formally, our learning objective is to iteratively minimize the temporal difference error to a fixed target value.

$$\delta_t = |Q(s_t, a_t) - y_t|$$

$$y_t = R_{a_t}(s_t, s_{t+1}) + \gamma\, Q(s_{t+1}, \underset{a'}{\operatorname{argmax}}(Q(s_{t+1}, a')))$$

III. Literature Review

In the paper 'Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge'

Task:

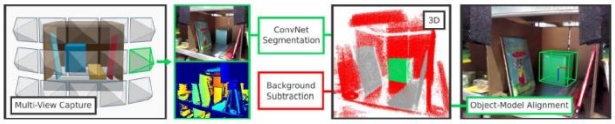
1. Select an instance of a given product ID from the filled shelf and place it in a handbag.
2. Place a handbag filled with products on a filled shelf.

Difficulty:

1. The environment is chaotic. Shelves and handbags may have multiple objects and may be arranged to deceive visual algorithms.
2. Self shielding. Due to the limited location of the camera, the system can only see a partial view of the object.
3. Sensor noise. Commercial depth sensors are not reliable in capturing reflective, transparent, or mesh surfaces.
4. There are many objects.

Algorithm description:

The algorithm first feed's multiple view images into a fully convolutional neural network, and then segments and labels multiple views. Then, the 3D model is fitted to a segmented point cloud to restore the 6D pose of the object.



Algorithm implementation:

Object segmentation

1) Multi-view target segmentation.
2) Remove point cloud noise.
3) Handle object duplicates.

Model fitting

1) Uneven density.
2) Initial posture.
3) From coarse to fine ICP.

Self-monitoring training

1) Place a single known object into a shelf or storage box in any posture, control the robot to move the camera, and obtain RGB D images from different perspectives.

2) To obtain a pixel level target segmentation label, create an object mask to separate the foreground from the background.

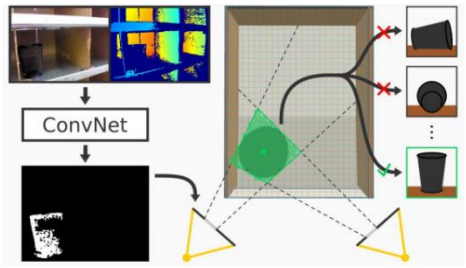
Automatically label object masks based on the results of two channels

Algorithm evaluation

Evaluate variations of methods in different scenarios on a benchmark dataset to understand two issues:

① How does segmentation perform under different input modes and training dataset sizes?

② How does the entire visual system perform?



1. Benchmark dataset.
2. Segmentation of the evaluation object.

Help for our research articles:

This article, as the predecessor of our article, has a good reference value. The methods used in this article can open our minds. This allows us to better understand the article we are studying.

IV. Technical Approach

We divide the whole project into six procedures:

1. Statement Representations

   This is the first step that we start. By taking photos, the heightmap was built in the computer, and is rotated for 22.5°for the next photo (Altogether 16 photos, 360°). The edges of the heightmaps are predefined with respect to the boundaries of the agent's workspace for picking.

2. Primitive Actions

   Since the robot arm is about pushing and grasping, the authors only focused on these two motions. The author parametrized actions:

   $$a = (\psi, q) \mid \psi \in \{\text{push}, \text{grasp}\}, q \twoheadrightarrow p \in s_t$$

   Ψ is the behavior, p is a pixel, and q means different ways of movement. For pushing, q is a horizontally 10cm push. For grasping, q is a vertically 3cm height-decrease. In both primitives, robot arm motion planning is automatically executed with stable, collision-free IK solves

3. Learning Fully-Convolutional Action-Value Functions

   Deep Q-networks are separated into two fully convolutional networks, for grasping and pushing. Both FCNs share the same network architecture. Pixel-wise parameterization of both state and action spaces enables the use of FCNs as Q-function

approximators. The action that maximizes the Q-function is the primitive and pixel with the highest Q value across all 32 pixel-wise maps.

4. Rewards

   Rewards are divided into two parts: one is for successful grasping R_g=1, and the other is for detectable pushing R_p=0.5. But the author mentions that a detectable pushing may not be able for further grasping, so in the experiment part, they make some differences.

5. Training Details

   This is how the Q-FCN functions will be trained, done by adapting the Huber Loss Function to the data.

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}(Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-})^2, \text{for } |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| < 1, \\ |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| - \frac{1}{2}, \text{ otherwise.} \end{cases}$$

   Besides this, the author also states the chosen value of some variables.

6. Testing Details

   The training policy is based on data, with the change of data it can iteratively become better. However, if there is no change of the outer environment, (like nothing is grasped or pushing), then it may fall into a drop-dead halt that repetitively does the same action. To avoid this, the author writes a small learning rate to the policy after each executed action, and at the time after training and before testing, network weights are set original. Furthermore, the author also counts actions' numbers so that the similar action will not exceed 10 times.

V. Intermediate / Preliminary Results

We have tried to run the demo of simulation the author provided on GitHub. During the process, we encountered lots of obstacles:

1. We found that CoppeliaSim no longer support the previous versions of V-REP 3.6.1, which the author used to do the simulation, so we have to install another system of Ubuntu 16.04 in order to support V-REP 3.5.0 (which is the oldest version of V-REP that we can find while the author used V-REP 3.0.4).)
2. There's problem in Pytorch and we have to change python version from 3.7.0 to 3.7.2 to support pytorch.
3. Still, the simulation cannot run for high version of pytorch, so we change to pytorch 0.3.1 and python 3.6 in order to support the version of pytorch. (The version of torchvision will also affect the version of pytorch, for torchvision later thatn 0.2.0, it will automatically set pytorch vision up to 1.1.0).
4. Finally we found that the author made a mistake on the demo, they might have wrote the demo in gramma of both pytorch 0.3.1 and in 0.4.0, while the two versions cannot fit each other, which made it impossible for the demo to be successfully run without any bugs. (Small mistake of the author made it a huge difficulty to us for we spent about a week on a demo of mistake!!!)

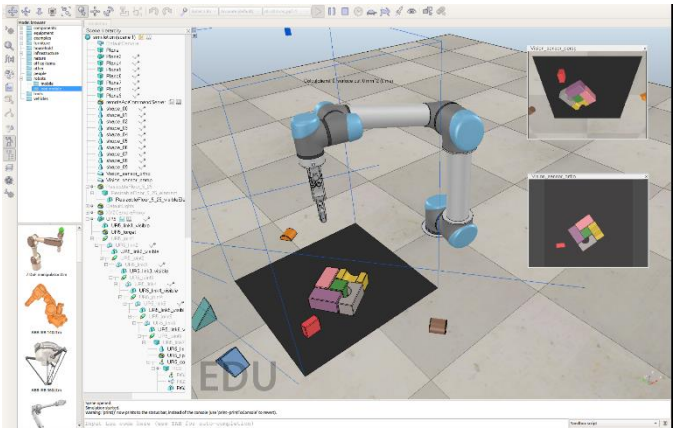


Fig1 the result of demo simulation (The robot arm cannot move because of bug).

This problem might not affect the rest of the part since they used pytorch 1.0+ for the training except the demo part. Nevertheless, it cannot be guaranteed there are not other mistakes in the rest of the part.

References

[1] Official website of amazon picking challenge. [Online]. Available: http://amazonpickingchallenge.org

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in CVPR, 2015, pp. 3431–3440.

[3] Website for code and data. [Online]. Available: http://apc.cs.princeton.edu/

[4] R. Jonschkowski, C. Eppner, S. Hofer, R. Mart ¨ ´ın-Mart´ın, and O. Brock, "Probabilistic multi-class segmentation for the amazon pick_x0002_ing challenge," http://dx.doi.org/10.14279/depositonce-5051, 2016.

[5] C. Eppner, S. Hofer, R. Jonschkowski, R. Martın-Martın, A. Sieverling, ¨

V. Wall, and O. Brock, "Lessons from the amazon picking challenge: Four aspects of building robotic systems," in RSS, 2016.