Reproduction of Pushing And Grasping Robot Arm

Zican Jin 12011610, Kewei Zuo 11912128, Zihao Wang 11910907, Handi Liu 12011205, Yizhou Lu 12012209

***Abstract*—Skilled manipulation refers to the high-level skills and dexterity that robots possess when manipulating objects. Research in this field aims to enable robots to perform complex object manipulation tasks, such as grasping, moving, and pushing, similar to humans. Reinforcement learning is a common method used to achieve Skilled manipulation. In reinforcement learning, robots learn how to take actions to maximize rewards through interaction with the environment. In Skilled manipulation, robots need to learn how to perform complex object manipulation tasks, such as grasping and moving objects. These tasks require the robots to have high-level skills and dexterity, so reinforcement learning can train robots through trial and error to gradually master these skills. Therefore, reinforcement learning is an important method to achieve Skilled manipulation.**

Based on this understanding, we decided to reproduce the work of a robot arm that is able of pushing and grasping. In this report, we will explain how we reproduce the work — some previous related work and background introduction, reasons we chose to reproduce this work, difficulties we encountered during the process and how we fixed them, our expected results of the experiments and the actual final results.

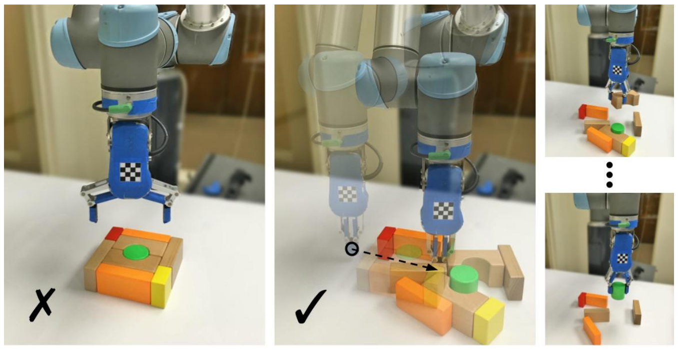


Fig. 1. **Example configuration of tightly packed blocks** reflecting the kind of clutter that commonly appears in real-world scenarios (e.g. with stacks of books, boxes, etc.), which remains challenging for grasp-only manipulation policies. The model-free system is able to plan pushing motions that can isolate these objects from each other, making them easier to grasp. By doing this, it improves the overall stability and efficiency of picking.

I. Background

A. Skill Manipulation

Skilled manipulation in robotics benefits from the intricate synergies between non-prehensile actions, such as pushing, and prehensile actions, such as grasping, see in Figure 1[12]. Pushing can aid in rearranging cluttered objects to create space for robotic arms and fingers, while grasping can assist in displacing objects to facilitate precise and collision-free pushing movements. The authors argue that skilled robotic manipulation requires complex synergies between non-prehensile (e.g., pushing) and prehensile (e.g., grasping) actions. They explain that pushing can help rearrange cluttered objects to make space for arms and fingers, while grasping can help displace objects to make pushing movements more precise and collision-free.

The limitations of previous approaches to robotic manipulation, as discussed in this paper, are that they have focused on either prehensile or non-prehensile manipulation, but not both together. This has resulted in limited robotic manipulation skills. For example, grasping can be limited by cluttered environments where objects are difficult to reach or grasp, while pushing can be limited by the inability to displace objects accurately and collision-free. By combining these two types of actions in a mutually supportive way, the authors argue that robots can greatly improve their manipulation skills.

B. Related Works

The related work discussed in this report is at the intersection of robotic manipulation, grasping, and pushing with grasping. We will briefly review the related work in these domains.

1) Robotic Manipulation: Previous works have focused on either prehensile or non-prehensile manipulation, but not both together. Some works have used grasping to manipulate objects, while others have used pushing to rearrange objects. However, there has been limited research on combining these two types of actions in a mutually supportive way. The literature on this topic is extensive, with classical solutions based on explicit modeling of pushing dynamics with frictional forces dating back several decades [5, 4]. However, many of these methods rely on modeling assumptions that do not hold up in practice [1, 11]. For instance, non-uniform friction distributions across object surfaces and variations in friction can lead to inaccurate predictions of friction-modeling pushing solutions in real-world settings. While some recent methods have explored data-driven algorithms for learning the dynamics of pushing [6, 14], many of these studies have focused mainly on executing stable pushes for one object at a time. Modeling the larger-scale consequences of pushing in settings with severe clutter and friction variation remains a complex problem. Moreover, effectively utilizing these models to discover optimal policies in real-world settings is even more challenging.

2) Grasping: Grasping has also been extensively studied in the domain of model-based reasoning, with approaches ranging from modeling contact forces to characterizing grasps by their ability to constrain object mobility[10]. However, these methods typically assume knowledge of object shapes, poses, dynamics, and contact points, which is rarely available for novel objects in unstructured environments. Recent data-driven methods have explored the possibility of training model-agnostic deep grasping policies that detect grasps by exploiting learned visual features[8, 13], without relying on object-specific information. These methods have shown promising results, including improved performance through pre-training on auxiliary tasks and the use of fully convolutional networks to model grasping policies with affordances. Obviously, many of these methods involves approaches in computer vision.

3) Pushing with grasping: The combination of non-prehensile and prehensile manipulation policies is an interesting area of research that has been less explored. Previous work has presented robust planning frameworks for push-grasping to reduce grasp uncertainty and to move around obstacles in clutter. However, these policies have largely been handcrafted. In contrast, the proposed method is data-driven and learned online through self-supervision. Other methods have explored the model-free planning of pushing motions to move objects to target positions that are more favorable for pre-designed grasping algorithms[7, 3]. However, defining similar goals for data-driven, model-agnostic grasping policies become less clear, as these policies are constantly changing and adapting behaviors over time with more data. More closely related to the proposed method is the work of Boularias et al.[2], which explores the use of reinforcement learning for training control policies to select among push and grasp proposals represented by hand-crafted features. However, their method models perception and control policies separately, relies on model-based simulation, and is tuned to work mainly for convex objects.

C. Using of VPG

1) Contributions: The main contribution of this paper is a new perspective to bridging data-driven prehensile and non-prehensile manipulation. The authors demonstrate that it is possible to train end-to-end deep networks to capture complementary pushing and grasping policies that benefit from each other through experience. In VPG, the authors compare their proposed method to two baseline methods: ”Pushing + Grasping Reactive” (P+G reactive) and ”Grasping Only” (G only). The main differences between VPG and these baseline methods are:

- VPG combines both pushing and grasping actions in a mutually supportive way, while P+G reactive only uses reactive grasping actions and G only only uses grasping actions.
- VPG uses deep reinforcement learning to learn pushing and grasping policies from experience, while P+G reactive uses a rule-based approach that reacts to the current state of the environment, and G only uses a grasp planner that selects grasps based on object geometry.
- VPG uses affordance-based manipulation to estimate the likelihood of successful pushing or grasping actions based on visual input, while P+G reactive and G only do not use visual input for action selection.
- In experiments with 30 objects randomly dropped onto a table, VPG outperforms both baseline methods across all metrics, including completion rates and action efficiency.

2) Methods: The authors formulate the task of pushing-for-grasping as a Markov decision process, where the robot chooses and executes an action based on a policy that maximizes the expected sum of future rewards. The goal is to find an optimal policy that maximizes the expected future reward, given by a discounted sum over an infinite-horizon of future returns. Overall, VPG combines deep reinforcement learning with affordance-based manipulation to learn pushing and grasping policies in a mutually supportive way from experience. By estimating affordances from visual input, VPG can learn to perform complex interactions with objects beyond what is possible with rule-based or geometry-based approaches

II. Questions about our choice on reproduction of this work

A. Why do we choose to reproduce this work?

We found this article on Github, and one of the most interesting things about it is that compared to normal robot arms, the one described in this paper has the ability to push objects. As undergraduate students, things that we can have access to are relatively few. This robot arm is a novel thing to us because, by pushing objects, it is able to change the outer shape of a set of things. By pushing irrelevant things away, it becomes easier for the robot arm to grab the main object, which is really useful in grasping things. So we decided to reproduce this work and figure out how it is able to push and grasp.

B. What will be reproduced?

The author of the paper provided not only experiments on computers, but also a platform for the robot arm to operative in real. However, due to some conditions, we cannot build a robot arm like that, and the limited time also prevents us from doing that, so we just decided to reproduce this paper in simulation environment, not in reality. We planned to reproduce the work in the following three aspects: single VPG, without push rewards, and short-sighted VPG.

C. What we need to do?

Thanks to the author, details have already been listed on Github, so it is easy for us to follow the instructions. We just mainly focused on the experiment part. The goals of the experiments are listed:

- To investigate whether the addition of pushing as a motion primitive can enlarge the set of scenarios in which objects can successfully be grasped (i.e. does pushing help grasping).
- To test whether it is feasible to train pushing policies with supervision mainly from the future expected success of another grasping policy trained simultaneously.
- To demonstrate that their formulation is capable of training effective, non-trivial pushing-for-grasping policies directly from visual observations on a real system.

D. The problems we encountered and how we overcame them

1) Version Problems: The files on Github were uploaded in 2018, five years ago. Therefore, some of the software version are incompatible. Copperlia Simulation, is called 'V-REP' at first, but later the version after 3.6.1 was updated and renamed Copperlia simulation. The project was launched in 2018, when v-rep3.1.9 was used. Our first trial was to go to the official website to check some of the old versions. Unfortunately, the version required by the paper is so old that the official website does not provide now. Also, the scene in the previous project will report an error missing the PID controller. However, the higher version can be compatible with the lower version and will run despite the error. Despite that, the phenomenon of skew of clamping jaws would occur during operation, so we are not sure whether it is a version problem or not. Anyway, we chose Copperlia Simulation to run V4.4 our codes finally, and it turned out to be fine.

2) Not Enough Memory: We first used our own laptops to run the training code. However, the memory was far from enough. After several times of testing, a graphics card with more than 8 GB of memory is required to run through training. Although CPU can be used for training, the final training time and training results are not as good as GPU. The training graphics card we used is NIVIDIA RTX 1080Ti, 12GB, but it still had trouble in the 'Reactive' type training learning. 'Non-reactive' learning consumes 7 to 8GB of video memory in training, and the time needed for training a relatively complete model would take about 5 to 7 hours. If a better performance graphics card was used, that time will be greatly reduced. We recommend that others who want to reproduce this work in the future to use at least NIVIDIA 3070, more than 16GB graphics card. In that case, compared to 1080ti, the training time can be reduced by more than half.

3) Incompatible Versions: Cuda and Pytorch. These two softwares need to be compatible for their versions, The final version adopted was CUDA 10.0 and pytorch 1.2.

III. Software Installation

We just install all the softwares.

As mentioned, we just reproduced the simulation part, not real part. The authors test VPG by executing a series of tests in which the system must pick and remove objects from a table with novel arrangements of objects, as we can see in Figure 2. Policies are trained in scenarios with random arrangements of 10 objects (left in Figure 2), then evaluated in scenarios with varying degrees of clutter (10 objects, 30 objects, or challenging object arrangements).

A. Settings

Their are some of our experiments settings:

- Reference code: https://github.com/andyzeng/visual-pushing-grasping.
- Hardware: Intel(R) Xeon(R) CPU E5-2640 v4@ 2.40GHz, GeForce RTX 3090, CUDA11.1, Ubuntu 20.04 LTS.

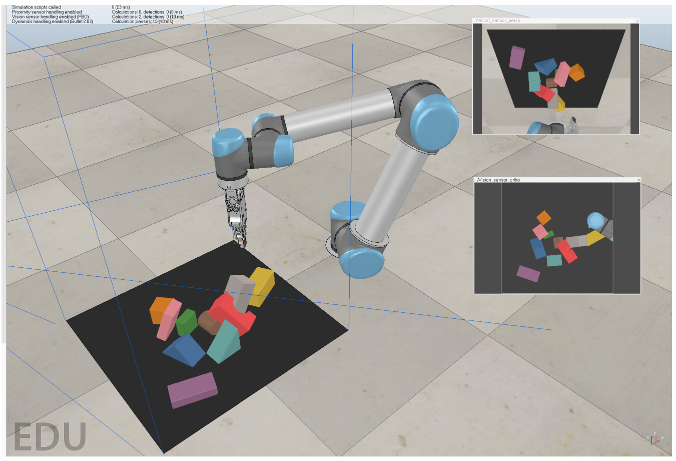


Fig. 2. **Simulation environment**

- Version of key packages: python3.8, pytorch 1.8.1, V-REP 4.3.0[9].
- Settings of robot: UR5 robot arm, RG2 gripper in V-REP, Bullet Physics 2.83.

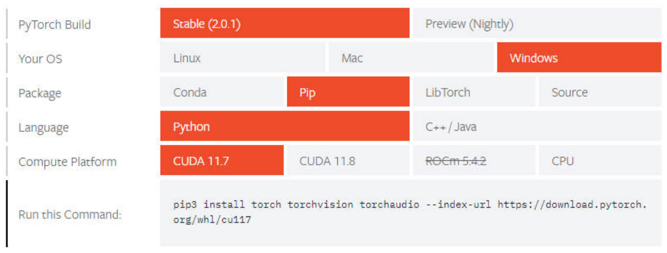
We execute n runs where $n \in [10, 30]$ and then evaluate the performance with 3 metrics:

- The average completion rate over the n test runs, which measures the ability of the policy to finish the task by picking up all objects without failing consecutively for more than 10 attempts.
- The average grasp success rate per completion.
- The action efficiency($\frac{\#objectsintest}{\#actionsbeforecompletion}$), which describes how succinctly the policy is capable of finishing the task.

For all of these metrics, higher is better.

B. Build Development Environment

a) Installation of PyTorch and CUDA: First, we install the PyTorch by the instructions on https://pytorch.org/get-started/locally/, see in Figure 3.



PyTorch Build	[✓ Stable (2.0.1)]	Preview (Nightly)		
Your OS	Linux	Mac	[✓ Windows]	
Package	Conda	[✓ Pip]	LibTorch	Source
Language	[✓ Python]	C++ / Java		
Compute Platform	[✓ CUDA 11.7]	CUDA 11.8	~~ROCm 5.4.2~~	CPU

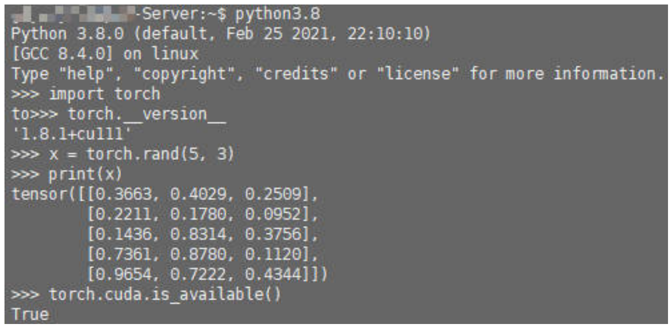
Run this Command:

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.
org/whl/cu117
```

Fig. 3. **Install PyTorch**

We also install CUDA follow the instructions in https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html. Then we verify that PyTorch and CUDA are installed correctly, see in Figure 4.

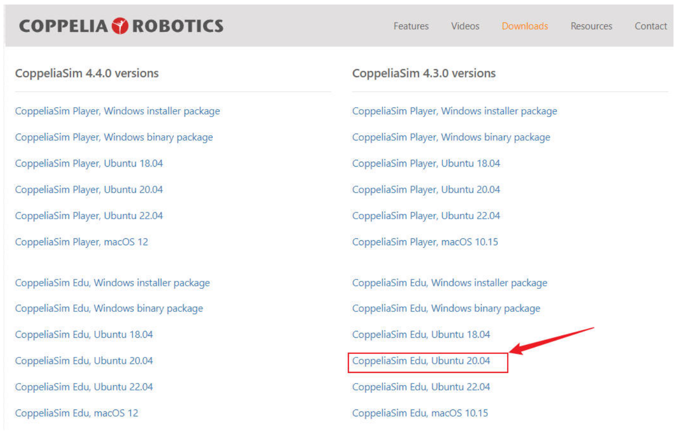
b) Installation of V-REP: V-REP is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin,



```
[redacted]-Server:~$ python3.8
Python 3.8.0 (default, Feb 25 2021, 22:10:10)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
to>>> torch.__version__
'1.8.1+cu111'
>>> x = torch.rand(5, 3)
>>> print(x)
tensor([[0.3663, 0.4029, 0.2509],
        [0.2211, 0.1780, 0.0952],
        [0.1436, 0.8314, 0.3756],
        [0.7361, 0.8780, 0.1120],
        [0.9654, 0.7222, 0.4344]])
>>> torch.cuda.is_available()
True
```

Fig. 4. **Verify Installation of PyTorch and CUDA**

a ROS node, a remote API client, or a custom solution. Controllers can be written in C/C++, Python, Java, Lua, Matlab or Octave. We download the install package from https://www.coppeliarobotics.com/previousVersions, see in Figure 5.



COPPELIA ROBOTICS

Features Videos Downloads Resources Contact

CoppeliaSim 4.4.0 versions	CoppeliaSim 4.3.0 versions
CoppeliaSim Player, Windows installer package	CoppeliaSim Player, Windows installer package
CoppeliaSim Player, Windows binary package	CoppeliaSim Player, Windows binary package
CoppeliaSim Player, Ubuntu 18.04	CoppeliaSim Player, Ubuntu 18.04
CoppeliaSim Player, Ubuntu 20.04	CoppeliaSim Player, Ubuntu 20.04
CoppeliaSim Player, Ubuntu 22.04	CoppeliaSim Player, Ubuntu 22.04
CoppeliaSim Player, macOS 12	CoppeliaSim Player, macOS 10.15
CoppeliaSim Edu, Windows installer package	CoppeliaSim Edu, Windows installer package
CoppeliaSim Edu, Windows binary package	CoppeliaSim Edu, Windows binary package
CoppeliaSim Edu, Ubuntu 18.04	CoppeliaSim Edu, Ubuntu 18.04
CoppeliaSim Edu, Ubuntu 20.04	CoppeliaSim Edu, Ubuntu 20.04
CoppeliaSim Edu, Ubuntu 22.04	CoppeliaSim Edu, Ubuntu 22.04
CoppeliaSim Edu, macOS 12	CoppeliaSim Edu, macOS 10.15

Fig. 5. **The Download page of V-REP**

After installed, open the software, we can see the UI of V-REP in Figure 6. We can see that there are some important parts in V-REP's UI: the views helps you adjust the view of the scene, you can choose a 3D model into the scene on the right, and you can choose all objects in the scene, finally you can control the action of objects in the scene.

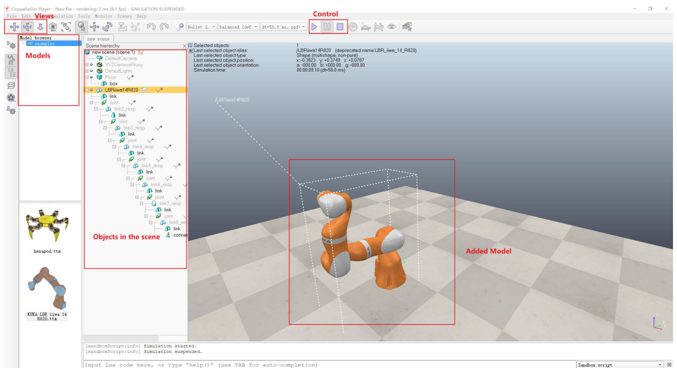
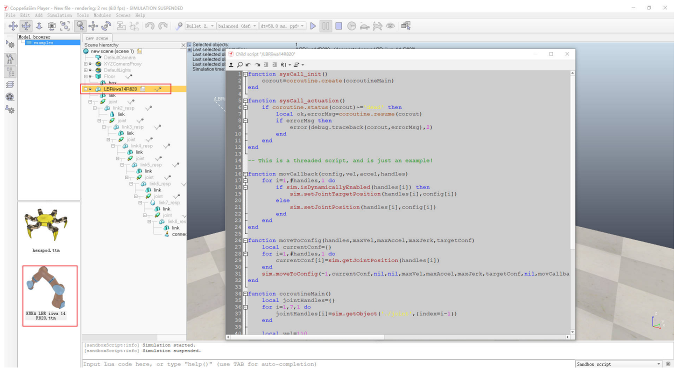


Fig. 6. **The User Interface of V-REP**

In V-REP, object's action is programmed by scripts, let's see the script of the added model, see in Figure 7, the default script language is Lua, and we can follow https://www.coppeliarobotics.com/helpFiles/index.html to use python to control objects.



```
function sysCall_init()
    corout=coroutine.create(coroutineMain)
end

function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end

-- This is a threaded script, and is just an example!

function movCallback(config,vel,accel,handles)
    for i=1,#handles,1 do
        if sim.isDynamicallyEnabled(handles[i]) then
            sim.setJointTargetPosition(handles[i],config[i])
        else
            sim.setJointPosition(handles[i],config[i])
        end
    end
end

function moveToConfig(handles,maxVel,maxAccel,maxJerk,targetConf)
    local currentConf={}
    for i=1,#handles,1 do
        currentConf[i]=sim.getJointPosition(handles[i])
    end
    sim.moveToConfig(-1,currentConf,nil,nil,maxVel,maxAccel,maxJerk,targetConf,nil,movCallba
end

function coroutineMain()
    local jointHandles={}
    for i=1,7,1 do
        jointHandles[i]=sim.getObject('./joint',{index=i-1})
    end

    local vel=110
```

Fig. 7. **The Action Script of model in V-REP**

c) Installation of Bullet Physics: Bullet Physics is a professional open source collision detection, rigid body and soft body dynamics library. Bullet Physics targets real-time and interactive use in games, visual effects in movies and robotics. We can use it by installing PyBullet. After installation, let's try an example of PyBullet by *python -m pybullet_robots.panda.loadpanda*, see in Figure 8.

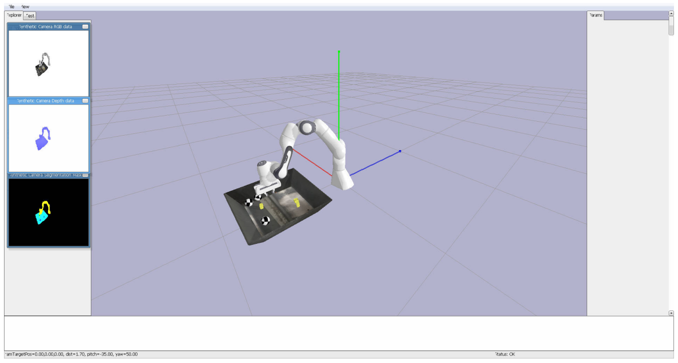


Fig. 8. **An Example of Bullet**

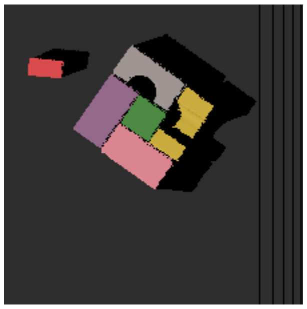
When V-REP and Bullet Physics are prepared, we can carry out the experiment.

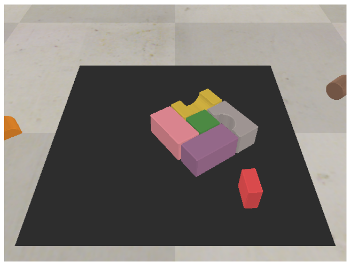
IV. Reproduction

A. Expected Results

We planned to reproduce the project in the context of simulation based on this paper. We attempted to reproduce all of the simulation configurations in challenging arrangements, and compare our testing results with the original results in paper, to see how well we have done with our reproduction work.

Because of limited computer memory, we cannot reproduce the 'Reactive' training part. So we just mainly focus on the following three configurations: Original VPG training,


(a) heightmap


(b) image

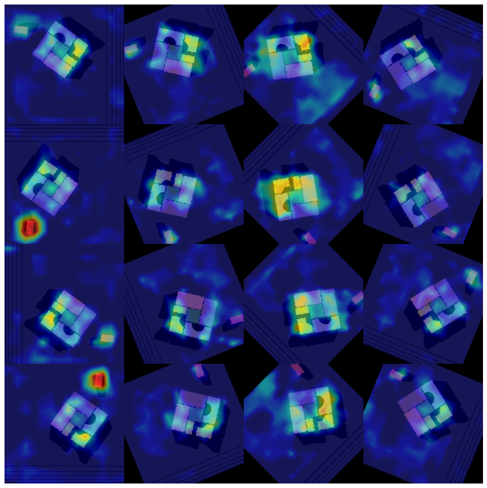

(c) visualization

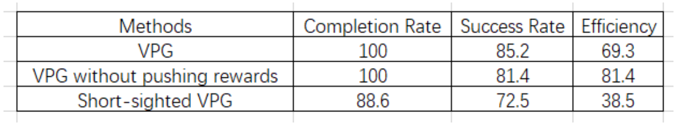
Fig. 9. **Operation interface**

VPG training without pushing rewards, and short-sighted VPG training.

For our reproduction work, we hope to have the same results as theirs. And, if possible, we hope that in some cases we can do better (Making some of the rates higher).

B. Actual results

Here we just briefly show the results. We will discuss about them in detail in the next part.



Methods	Completion Rate	Success Rate	Efficiency
VPG	100	85.2	69.3
VPG without pushing rewards	100	81.4	81.4
Short-sighted VPG	88.6	72.5	38.5

Fig. 10. **Results in one chart**

There are three columns in the chart. 'Completion Rate' means the ability of the policy to finish the task by successfully grasping all the objects. (Failures often happen during the grasping process, and are inevitable. So if the number of grasping failure times is less than 10, we still consider it as completed). 'Success Rate' means total grasp success rate per completion(As previously discussed, one completion does not ensure every grasp to be successful). 'Efficiency' means objects in test divided by actions before completion, which describes how succinctly the policy is capable of finishing the task. For all three rates, the higher means the better.

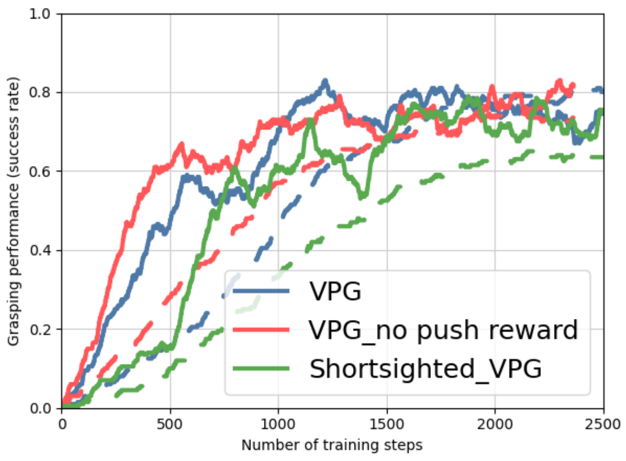
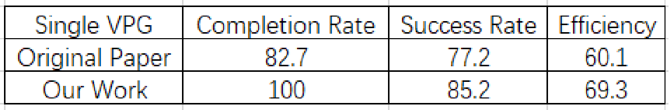


Fig. 11. **Results of three configurations in one image**

Legends are listed in the image. Totally we have 2500 training steps for each configuration. At every step, we obtain the success rate, then plot it on the image. The solid lines indicate grasp-only operations, the dash lines indicate push-then-grasp operations.

C. Results Analysis

1) Single VPG: Firstly we compare the results of single VPG configuration. We find that all of our three rates are higher than the original ones. We are happy with that because in the first configuration we manage to surpass the original paper.



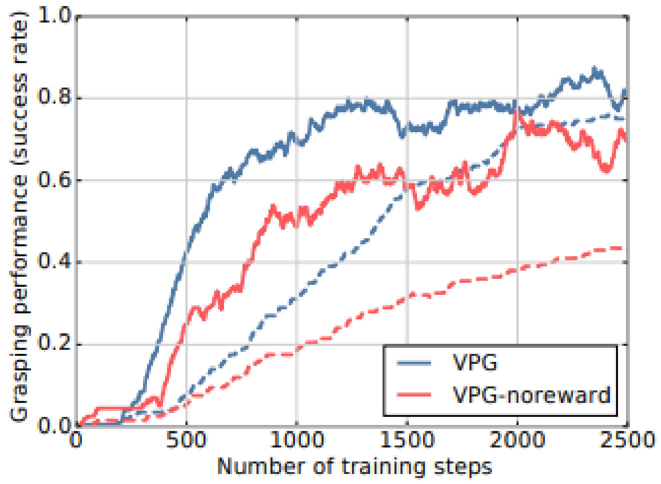
Single VPG	Completion Rate	Success Rate	Efficiency
Original Paper	82.7	77.2	60.1
Our Work	100	85.2	69.3

Fig. 12. **Comparison of single VPG results**

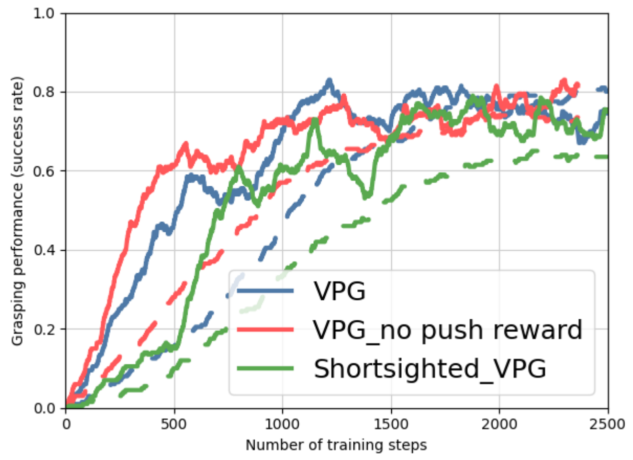
We think one of the reasons may be this — In the original paper, the author provides totally 11 test cases, but due to time limitation, we can only finish one of them. This can be reasonable according to the original paper, the author has a 100 completion rate for 5 of the 11 test cases. However, there may be some differences of the difficulty, so what we have chosen may be an easier one for the simulation to go around. In this case, a regular push is succinct and is helpful to the further process grasping. That may explain why our rates are much higher.

2) No pushing rewards: Secondly we compare the results of no pushing rewards. This is the configuration that the method has been modified to make sure that it learns synergistic pushing and grasping actions without any intrinsic rewards for pushing. In this more difficult setting, the pushing policy learns to effect change only through the reward provided by future grasps.

In the original paper, the author uses an image instead of chart data to present the result. Therefore, we choose the same way of drawing images.



(a) original results



(b) our results

Fig. 13. **Comparison of no pushing rewards results**

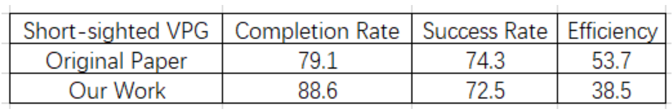
Since there is no virtual way to measure the quality of the pushing motions for how well they benefit a model-free grasping policy, this secondary metric serves as a good approximation.

Compared to the original results, the final results are similar, but our VPG without pushing rewards has higher grasping performance at first, and is even higher than single VPG. We consider that can be the same reason — a much easier test case.

Rewards can be regarded as a kind of feedback. This feedback, in the long run, can make the grasping process more accurate. However, for short times, more operations add in the time for training, resulting short increase at first of the success rate. But as long as the number of the steps is big enough (About 1000 times), the difference becomes smaller.

This goes the same for differences between grasp-only and push-then-grasp configurations. The dash lines indicate push-then-grasp operations. For grasp-only configuration, even if one grasp fails, it is still very likely to change the position of the block. Then, the second grasp may be successful. However, because the case is easier, pushing may be a negative action. It adds more computation, making the number of the steps more to reach same grasping performance.

3) Short-sighted VPG: Thirdly we compare the results of short-sighted VPG configuration. In the paper, the author proposed a 'long term strategy', that is, chaining multiple pushes to enable grasping, grasping to enable pushing, grasping to enable other grasps, etc. This method ensures that each step will have an effect on further steps. To test the value of this strategy, another short-sighted version of VPG is trained. The discount factor on future rewards is smaller at r = 0.2. In the right case, the VPG method should have higher rates than shorted-sighted VPG method.



Short-sighted VPG	Completion Rate	Success Rate	Efficiency
Original Paper	79.1	74.3	53.7
Our Work	88.6	72.5	38.5

Fig. 14. **Comparison of short-sighted VPG results**

This is what the author got in his paper, 'Interestingly, we see that VPG-myopic improves its grasping performance at a faster pace early in training (presumably optimizing for short-term grasping rewards), but ultimately achieves lower average performance (i.e. grasp success, action efficiency) across most hard test cases.' However, we seem to have opposite results. Compared with original results, ours seem to have an increase in the completion rate. But still, the reason has been discussed before. For other two rates, the results are not so good, especially for efficiency, the rate decreased towards 38.5. This is reasonable because we have few test cases, which means that the whole training process is somehow missing some parts.

And, compared with other two configurations, it is still the worst. This suggests that the ability to plan long-term strategies for sequential manipulation could benefit the overall stability and efficiency of pushing and grasping.

V. Conclusion

In this report, we reproduce the Visual Pushing for Grasping (VPG) proposed in [12], which combines pushing and grasping actions in a mutually supportive way using deep reinforcement learning and affordance-based manipulation. We reproduce for three situations including original VPG, VPG without pushing rewards and short-sighted VPG.

The original paper just discussed pushing and grasping motions for robot arms. But for further studies, we may focus on different motions (like rolling, toppling and squeezing) to make deeper reinforcement learning of robot arm. We may also adjust the shape and numbers of the objects to be grasped.

Currently they are just geometric blocks, but in real cases, their shape may be irregular, and that will be more difficult for robot arms to grasp them. In the cases we discussed, we use randomly arranged 10 blocks for testing. But in the original paper, the author finally has the testing over 30 blocks. Increase in the number of the blocks largely makes it more difficult for the robot arm to grasp them.

The installation of software costs us much time. Some of the version is out of date that even on official website we cannot find the version we want. And we have to look for other ways. If we were about to do this again, we may change another simulation environment(like Robosim or RobotStudio). If possible, we may use larger memory so that we can reproduce more different situations, including the 'Reactive' part.

References

[1] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, page 3008–3015. IEEE Press, 2017. doi: 10.1109/ICRA.2017.7989345. URL https://doi.org/10.1109/ICRA.2017.7989345.

[2] Abdeslam Boularias, J. Andrew Bagnell, and Anthony Stentz. Learning to manipulate unknown objects in clutter by reinforcement. In *AAAI Conference on Artificial Intelligence*, 2015.

[3] Ignasi Clavera, David Held, and Pieter Abbeel. Policy transfer via modularity and reward guiding. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 1537–1544. IEEE Press, 2017. doi: 10.1109/IROS.2017.8205959. URL https://doi.org/10.1109/IROS.2017.8205959.

[4] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, 143(2):307–330, 1991. ISSN 0043-1648. doi: https://doi.org/10.1016/0043-1648(91)90104-3. URL https://www.sciencedirect.com/science/article/pii/0043164891901043.

[5] M T Mason. Mechanics and planning of manipulator pushing operations. *Int. J. Rob. Res.*, 5(3):53–71, sep 1986. ISSN 0278-3649. doi: 10.1177/027836498600500303. URL https://doi.org/10.1177/027836498600500303.

[6] Tekin Meriçli, Manuela Veloso, and H. Levent Akın. Push-manipulation of complex passive mobile objects using experimentally acquired motion models. *Auton. Robots*, 38(3):317–329, mar 2015. ISSN 0929-5593. doi: 10.1007/s10514-014-9414-z. URL https://doi.org/10.1007/s10514-014-9414-z.

[7] Damir Omrčen, Christian Böge, Tamim Asfour, Aleš Ude, and Rüdiger Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 277–283, 2009. doi: 10.1109/ICHR.2009.5379566.

[8] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, page 3406–3413. IEEE Press, 2016. doi: 10.1109/ICRA.2016.7487517. URL https://doi.org/10.1109/ICRA.2016.7487517.

[9] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[10] Jonathan Weisz and Peter K. Allen. Pose error robust grasping from contact wrench space metrics. In *2012 IEEE International Conference on Robotics and Automation*, pages 557–562, 2012. doi: 10.1109/ICRA.2012.6224697.

[11] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 30–37. IEEE Press, 2016. doi: 10.1109/IROS.2016.7758091. URL https://doi.org/10.1109/IROS.2016.7758091.

[12] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 4238–4245. IEEE Press, 2018. doi: 10.1109/IROS.2018.8593986. URL https://doi.org/10.1109/IROS.2018.8593986.

[13] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R. Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *Int. J. Rob. Res.*, 41(7):690–705, jun 2018. ISSN 0278-3649. doi: 10.1177/0278364919868017. URL https://doi.org/10.1177/0278364919868017.

[14] Jiaji Zhou, Robert Paolini, J. Andrew Bagnell, and Matthew T. Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, page 372–377. IEEE Press, 2016. doi: 10.1109/ICRA.2016.7487155. URL https://doi.org/10.1109/ICRA.2016.7487155.