Karlsruher Institut für Technologie

Pixel-based object grasping with Deep Reinforcement Learning in simulation

Bachelor thesis

Anna Fedorchenko

Fakultät für Informatik
Institut für Anthropomatik
und
FZI Forschungszentrum Informatik

Erstgutachter:	Prof. Dr.–Ing. R. Dillmann
Zweitgutachter:	-
Betreuender Mitarbeiter:	M. Sc. Atanas Tanev

Bearbeitungszeit: 01. January 2020 – 30. April 2020

Title of thesis - Second title line

von
Anna Fedorchenko

I will not use this image for my thesis!

Bachelor thesis
im April 2020

Bachelor thesis, FZI
Fakultät für Informatik, 2020
Gutachter: Prof. Dr.–Ing. R. Dillmann, -

Abteilung Technisch Kognitive Assistenzsysteme
FZI Forschungszentrum Informatik

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, *Anna Fedorchenko*
im April 2020

Inhaltsverzeichnis

1 Introduction **3**
- 1.1 Motivation . . . 3
- 1.2 Problem statement . . . 4

2 Related work **5**
- 2.1 Grasp Synthesis Algorithms . . . 5
- 2.2 Simulation . . . 8
  - 2.2.1 End-To-End Learning . . . 8
  - 2.2.2 Deep reinforcement learning for grasping problems . . . 9
- 2.3 Algorithms in reinforcement learning . . . 12
  - 2.3.1 Q-Learning . . . 13
  - 2.3.2 Actor Critic . . . 15
  - 2.3.3 Soft Actor-Critic . . . 16

3 Approach **19**
  - 3.0.1 System components . . . 19
  - 3.0.2 Mujoco . . . 19
  - 3.0.3 OpenAI Gym . . . 19
  - 3.0.4 Stable Baselines . . . 20

4 Implementation **21**
  - 4.0.1 Simulation Setup . . . 21
  - 4.0.2 CNN structure . . . 21
- 4.1 System Architecture . . . 21

5 Evaluation **23**
  - 5.0.1 Simulation Inaccuracies . . . 23
  - 5.0.2 Experiments with the Soft-Actor-Critic Algorithm . . . 24
  - 5.0.3 Experiments with the Proximal Policy Optimization Algorithm . . . 28

A Abbildungsverzeichnis **33**

B Tabellenverzeichnis **35**

C Literaturverzeichnis **37**

Data-driven robotic grasp synthesis in a simulated environment.
(Datengesteuerte Robotergreifenssynthese in einer Simulationsumgebung.)

1 Introduction

1.1 Motivation

Nowadays there is a big variety of use cases in the field of robotics in everyday life, starting from smart home devices to autonomous driving vehicles and space shift robots. Due to the significant development of soft- and hardware in the last decades it is becoming possible to create new robots that could considerably impact humans' lives.

In various scenarios robots have to grasp an object and then perform different actions on it: lift, shift, put on a specific position. It is crucial for the grasping part to succeed in order to be able to complete the proceeding steps of the manipulation.

The parameters of the grasping problem vary depending on the use case scenario. In some cases there are several objects on the surface, one of them (a specific one or a random) has to be grasped - this is a cluttered environment scenario. In other cases, the object is singulated. Objects can be already known in advance with existing 3D meshes of them (i.e.in the industry scenario), familiar or never seen before. They can have a specific shape (only cubes) or have a random form, be rigid or non-rigid, transparent or not etc. They type of gripper plays a role (i.e. a two- or a five-finger gripper) and whether there is a camera or several cameras to control the process.

The current state-of-the art approach for the grasping problem in the industry is based on knowing the exact positions of the robotic gripper and the object, what kind of the object that is, its size and shape. So the grasping motion is also already predefined. However, if the some characteristics of the objects slightly change, the robot might not be able to grasp it anymore.

The goal solution to the grasping problem would consist of the robot being able to grasp any object. It means that the robot should be able to adapt to novel objects that it never saw in the training - generalization. Another important aspect is minimizing human involvement in the training process: is it possible for the robot to learn how to grasp without a need for a human to spend hours or even days teaching him how to do so? Moreover, it would be preferable for the learning time of the robot to be realistic - if the training process lasts for years, the problem might change during that time and the training result might be not applicable anymore.

Data-driven approaches based on reinforcement-learning have shown promising results for solving the grasping problem. Self-supervised systems collect training data without any human help and learn from it. They are able to generalize to novel objects. Training the robot in simulation is helpful for fast collecting large quantities of data with little cost. Combining these approaches and then transferring to the real world set-up might be an effective way to solve the grasping problem.

1.2 Problem statement

The focus of the thesis is to develop an approach for the robot to learn how to grasp a rigid object using images from RGB camera and two-finger parallel gripper. The learning process should be self-supervised: any human labeling should be avoided. This will be achieved with the help of deep reinforcement learning. The training and testing is conducted in a simulated environment.

2 Related work

2.1 Grasp Synthesis Algorithms

Sahbani et al. [34] divided different approaches in analytical and empirical. Analytical approaches investigate the physics, kinematics and geometry of the object and the robot in order to determine contact points and gripper position[29], [28]. [35] defined a force-closure grasp as the one that guarantees that "the fingers are capable of resisting any arbitrary force and/or moment that may act on the object externally". Shimoga listed four independent properties that are crucial for a successful force-closure grasp : dexterity, equilibrium, stability, dynamic behavior. Analytical approaches concentrate on developing algorithms that satisfy these properties.
[11] of Ding et al. is an example of the analytical approach. Having a multi-fingered gripper with n fingers, an object and the position of m of n fingers that do not form a form-closure grasp, the position of the remaining m-n fingers have to be determined to satisfy the requirement of the form-closure grasp. There is a Coloumb friction between object and fingers. In order for fingers not to slip while executing the grasp, the finger force must have a certain direction (lie in a friction cone), which can be expressed as a set of non-linear constraints. Calculations consider the center of mass of the object, combination of grasping force and torque, center of the contact points. A sufficient and necessary condition for form-closure grasps was formulated.

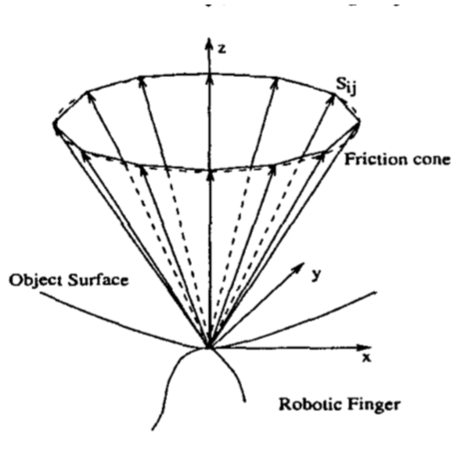


Figure 1: The friction cone at a grasping point.

Abb. 2.1: Outtake from "Computing 3-D Optimal Form-Closure Grasps"[11] of Ding et al.

As Bohg et al. [5] stated, analytical approaches usually require exact 3D models of the object, rely on the knowledge of the object's surface properties, its weight distribution and are not robust to noise.

[5] made a detailed overview of the data-driven grasp synthesis approaches. They define grasp synthesis as "the problem of finding a grasp configuration that satisfies a set of criteria relevant for the grasping task". Data-driven or also called empirical approaches sample various grasp candidates and then rank them using different strategies.

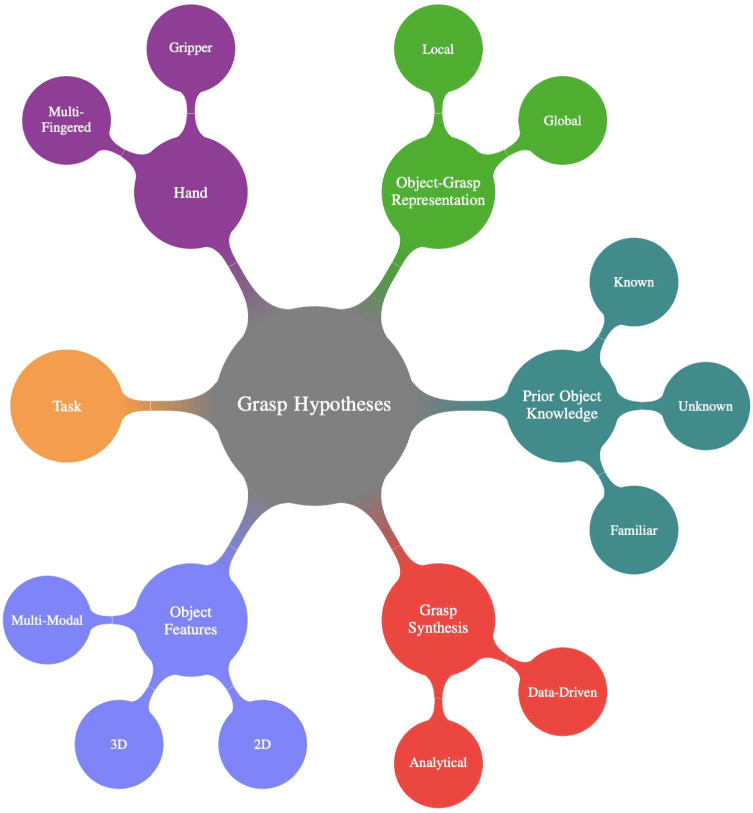

Gripper
Local
Multi-Fingered
Global
Hand
Object-Grasp Representation
Known
Task
Grasp Hypotheses
Prior Object Knowledge
Unknown
Familiar
Object Features
Grasp Synthesis
Multi-Modal
Data-Driven
3D
2D
Analytical

Abb. 2.2: Classification of different aspects that influence the problem of the grasping problem according to [5] of Bohg et al.

Human demonstrations can be used to generate data for learning. In [12] magnetic trackers were placed on the hand of the human who was grasping and moving objects on the table. The robot recognized which object was moved and which grasp type was used, it then reproduced the task using this information.

Another strategy is to compare the candidate grasp to (human) labeled examples. Redmon et al. [33] use the Cornell grasping dataset [1] to compare the sampled grasps to the "ground truth"grasps from the dataset. The rectangle metric is used for filtering good grasps. The metric includes two requirements: the grasp angle must be within 30° of the ground truth grasp and the Jaccard Index $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ of the predicted grasp and the ground truth is greater than 25 percent.

Empirical approaches can use analytical metrics for ranking grasp candidates or labeling robust grasps. Mahler et al. [24] introduced a synthetic dataset that includes 6.7 million point clouds with parallel-jaw grasps and analytic grasp quality metrics. Objects in the dataset come from the database containing 1500 3D object mesh models (129 of them come from KIT object database

[19]). For every object robust grasps covering the surface were sampled, using the antipodal grasp sampling approach developed in Dex-Net 1.0 [25]). For stable poses of each object planar grasps (grasps perpendicular to the table surface) are chosen, as well as corresponding rendered point clouds (depth images). The introduced dataset was used to train a neural network, that was also introduced in [24]. The network accepts a depth image of the scene and outputs the most robust grasp candidate.

Human labeling of possible grasps for objects has some significant disadvantages. First of all, it is time consuming. Secondly, there exist a number of possible grasps for every object, it is hard and even impossible to tag every one of them. Pinto et al. [30] also remarked the fact that human labeling is "biased by semantics": as an example they describe usual human labeling of handles of cups, because that is how a person would most likely to grasp a cup, although there are more possible configurations for successful grasp.

This reasoning led to development of self-supervised systems, where a robot learns from its own experience during the trial and error process. This approach is inspired by the way that people learn. Pinto et al. [30] used a CNN for assessing planar grasp candidates. The grasp candidate is represented as (x,y) coordinates - the center of a chosen part of the image(patch) - and as an angle q - the rotation of the gripper around the z-axis. The CNN estimates a 18-dimensional likelihood vector. Each component of the vector represents the probability whether the grasp will be successful at the center of the patch with one of the 18 possible rotation angles of the gripper. The grasp with the highest probability of the success is executed. Depending on the result of the grasp, the error loss is back-propagated through the network. This way, the system does not rely rather on analytical metrics nor on human labeled examples, - it learns from its own experience.

Following the classification of [5], the objects that have to be grasped can be divided into three categories: known, familiar and unknown objects.

For known objects the data-driven approaches for finding a successful grasp often consist of estimating the pose of the object and then choosing the suitable grasp candidate from a precalculated grasp database, ëxperience database". In [38] of Tremblay et al. the deep neural network takes as input only one RGB-image of the scene with known household objects and outputs belief maps of 2D keypoints of these objects. After that a standard perspective-n-point (PnP) [21] algorithm uses these belief maps to estimate the 6-DOF pose of each object. For grasping the robot moves its arm to a point above the object and then completes a top-down grasp.

Another category of objects to grasp is familiar objects. Objects can have similarities in various aspects(texture, shape, etc.). Familiar objects might be grasped in similar ways, the difficulty consists in detecting these similarities between objects and then applying grasping experience on them. [26] researched category-level object manipulation with the help of using semantic 3D keypoints as object representation. The two object categories examined were shoes and cups. The goal of robotic manipulation was not only grasping but also positioning the objects in a specific predefined way (place shoes on a shelf, hang cups on the hook by the handle). First the database of manually labelled keypoints on 3D reconstruction of the objects in different training scenes was created. Then a neural network was used to detect these keypoints from an RGB-D image of the scene. The detected keypoints together with the RGB-D image were used to calculate a suitable grasp using a

similar to the baseline learning-based algorithm demonstrated in [42].
In case of unknown objects, the object was never seen by the robot beforehand and never used in training. Many approaches are based on approximating the shape of the object and then choosing the grasp for it [4]. In recent years the approaches that have been most successful at solving this type of grasping problem through trial-and-error approach [30], [40], [42].

2.2 Simulation

In data-driven approaches training data is required to learn for a successful grasp. Levine et al. [23] used 14 robotic arms that sampled 800 000 grasp attempts. Pinto and al. [30] used one robot manipulator to conduct 50 000 grasp attempts in course of 700 hours. However, it is expensive to collect such amount of data and it is very time consuming, this is where the arguments for learning in simulation come to a point.
Goldfeld et al. [14] created a grasp planner containing database of grasps for 3D models of different objects, that were generated using GraspIt! [27] simulation engine. Kasper et al. [19] introduced the system for digitizing objects. In year 2010 the OpenGRASP [20] simulation toolkit for grasping and dexterous manipulation was created.
James et al. [17] developed an approach that used only a small amount of real-word training data in addition to simulation, which helped to reduce the real-world data by more than 99%. Bousmalis et al. [7] implemented a grasping method that helps to significantly reduce the amount of additional real-world grasp samples. In their experiment 50 times less real-world samples were required to achieve the same level of performance compared to their previous system.
Another advantage of using simulation is the ability to pretrain the network. Redmon et al. [33] stated that "pretraining large convolutional neural networks greatly improves training time and helps avoid overfitting". However there are some drawbacks to synthetic data. Most significant one is the reality gap: the network trained in a simulated environment shows much worse performance in real world. An effective way to trying to close the reality gap is to use domain randomization(TODO cite). [38] used photorealistic data in addition to domain randomization. It added variation and complexity to the training, which helped for the trained neural network to be able to successfully operate in the real world scenario without any additional fine-tuning.
(TODO domain adaptation) ——

2.2.1 End-To-End Learning

In robotics the classical control over the robot consists of several steps, each one of them usually represented by a separate module which are connected with each other and pass the data to each other. For example: after retrieving an observation of the scene using a calibrated camera another module detects objects in the image. The next module might create a physics model of the scene to plan an action that needs to be executed. The next step would calculate the positions of robot's joints and after that the actual action would be executed. If an error is made in one of these steps, it might become even more significant in the next steps resulting in incorrect output. End-to-end approach suggests making such connection between modules, that can be changed and adapted during the learning. In that way mistakes that would happen in one of the modules would be corrected in the following ones so that in the end the output would not be influenced by them. Levine et al. [22] discovered that training perception and control systems end-to-end shows better results and consistency than training each component separately.

2.2.2 Deep reinforcement learning for grasping problems

raw-pixels, image obs ([31])
One of the most important goals of learning-based approaches is for system to be able to perform effectively on previously unseen objects - generalization. Another essential aspect that is considered during development of the approach is the intention of minimizing human's participation in the training of the system. In recent years several reinforcement learning approaches have been successfully applied for solving the grasping problem. (TODO cite). The are based on the robot learning from it's trial and error experience. It is collecting the training data by itself, This lead to development of self-supervised systems.
[18] made following definition for the reinforcement learning problem: "Reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment."The trial-and-error interactions are the factor that is crucial for a self-supervised system: it supervises itself by committing an action, getting the result of this action and learning from it. In many approaches for solving the grasping problems self-supervised systems can be interpreted as the ones based on reinforcement learning, even if they do not operate on standard RL algorithms. (Atanas Im not sure about that these are just my thoughts but I think they sound logical xD)
In recent years several data-driven systems that take only images of the scene as input have been successfully applied for the grasping problem. Different types of convolutional neural netwroks: TODO: [30] [23] developed a grasping approach that is based on collecting large-scale data using multiple real robots, using no human involvement in the training process, and getting continuous feedback on visual features using only over-the-shoulder RGB camera. This end-to-end approach uses raw pixel images as input and directly outputs task-space gripper motion. The paper addresses the issue of the need of massive analysis and planning in robotic manipulation tasks. The suggested methods avoids it by providing the system with continuous feedback from the setup, fragmenting the grasping attempt in several timesteps and letting it decide what action to take at each timestep. This way the robot can correct its mistakes using previous experience, at the same time not requiring exact camera calibration.
The approach consists of two major parts: the prediction network $g(T_t, v_t)$ that accepts visual input I_t and a task-space motion command v_t and outputs the probability of success of grasping after completing v_t. The second part is the servoing function $f(I_t)$ that uses the output of the prediction network to control the gripper for executing a good grasp. The grasping attempt consists of T steps: The robot makes T steps before closing the gripper. It creates T training samples: $(I_t^i, p_T^i - p_t^i, l^i)$: image, collected every step, vector that is calculated through reached pose at the end and the pose

at the timestep t, and label l_i that is the same for the whole attempt i and which denotes the success of the attempted grasp. The whole self-supervised method can be interpreted as a type of reinforcement learning but without standard reinforcement learning algorithms.

Reinforcement learning has been successfully used for grasping tasks [6], (TODO cite).

[31] compared different reinforcement learning off-policy algorithms for vision-based robotic grasping. The authors state that on-policy algorithms struggle with diverse grasping scenarios as the robot has to go back to previously seen objects in order to avoid forgetting the gained experience. Therefore off-policy methods might be preferred for the grasping problems. The criteria for evaluating the RL were: overall performance, data-efficiency, robustness to off-policy data and hyperparameter sensitivity - these factors are important when applying the grasping algorithms to robotic systems in real life.

Boularias et al. [6] used reinforcement learning approach to grasp objects in dense clutter. The task was formulated as a Markov Decision Process (MDP). The state was represented as the RGB-D image of the scene. Action space consists of two types of actions: pushing and grasping, each action is an according vector. In a cluttered environment with a variety of objects sometimes the position of an object makes it hard to grasp it. Executing a pushing action on this or another object might help to gain better access to the object for grasping. The reward is 1 if the robot managed to successfully grasp an object, otherwise 0. The RGB-D image is primarily segmented into objects using spectral clustering [39].

The state-of-the art end-to-end self-supervised system for solving the grasping problem is presented in [41]. Zeng et al. used model-free deep reinforcement learning approach to modulate the system. In the MDP the RGB-D the state was represented as a RGB-D image of the scene. The actions are either a grasping or a pushing motion. The reward is 1 for successful grasps, 0.5 for pushing motions that make a considerable change to the environment, otherwise 0. The system consists of two DenseNet-121 neral networks (TODO cite) that take the image as input and output the probability that .

TODO: - tossing bot
- Andy Zeng(2)
pre-grasp manipulation: pushing: [6]

Reinforcement learning

TODO
- Introduction history applications
- Value vs policy based

Reinforcement learning is one of the types of Machine Learning. The core idea of reinforcement learning is having an agent that can perform actions in a specified environment. The agent gets observations from the environment as an input, the output is an action. At all times the agent is in one of the predefined states, the actions that are possible in the current state are a subset of all actions set. For every performed action the agent receives a feedback in form of a reward from the environment. According to the reward the agent can decide whether the chosen action was the right decision. The core idea of reinforcement learning is modeled as Markov decision process (MDP), which consists of 4-tuple (S, A, R, T) [18]:

- set of states S
- set of actions A
- a reward function r (R: SxAxS -> RR)
- state transition function T: S x A ->P(S), where P(S) is a probability distribution over the set S (i.e. it maps states to probabilities)

in some notation the the starting state distribution r0 is defined as the fifth element of the MDP. The name of the process comes from the concept of Markov chains: having a sequence of events the probability of each event depends only on the state that was achieved in the previous event, ignoring all events before. Markov state: $P(s_{t+1}|s_t) = P(s_{t+1}|s_1,...,s_t)$(TO DO: site).
The policy Pi determines which action must be taken in a each state. The action might be deterministic: (Source: spinning up)
$a_t = \mu(s_t)$
or stochastic:
$a_t \sim \pi(\cdot|s_t)$
Same for the state transition function: deterministic $s_{t+1} = f(s_t, a_t)$ or stochastic $s_{t+1} \sim P(\cdot|s_t, a_t)$.
Often there are possible ways for the agent to accomplish a task. For example, there might be a short and a long path to a destination point, both of them conform the requirements. However, the short path is better because it takes less time and less actions, so the agent should take this path. This can be expressed as following: goal of the optimal policy is to maximize the sum of rewards during action sequence that leads to completing the task. There are multiple ways to define the sum of the rewards:

- finite-horizon undiscounted return: R(t) = $\sum_{t=0}^{T} r(t)$, a straightforward sum of all rewards for all taken actions.
- infinite-horizon discounted return: R(t) = $\sum_{t=0}^{\infty} y^t r(t)$, where y is a discount factor ((0,1)). The discount factor makes sure the actions that are taken soon are more relevant that the ones that are taken many steps later. From mathematical point of view, the discount factor is one of the conditions that the infinite sum converges to a finite value.

Value function of the state s is an expected return that the agent will get if it starts in state s and act according to the policy. Action-Value Function adds dependency to an action a: expected return after starting in state s and taking action a, continuing by acting forever according to the policy π:
$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}[R(\tau)\,|s_0 = s, a_0 = a]$
the Optimal Action-Value Function, Q*(s,a), which gives the expected return if you start in state

s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:
Finally, the Optimal Action-Value Function, Q*(s,a):
$Q^*(s,a) = \max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau)\,|s_0 = s, a_0 = a]$
, where policy π is optimal.
There is a connection between value and action-value functions, that is helpful for some calculations:
$V^\pi(s) = \underset{a\sim\pi}{E}[Q^\pi(s,a)]$ - the value function of the state is an expected return of the Q-function in this state if the action a taken comes from the policy π.

and

$V^*(s) = \max_a Q^*(s,a).$
TODO: Bellman equations

2.3 Algorithms in reinforcement learning

There is a wide variety of reinforcement learning algorithms:

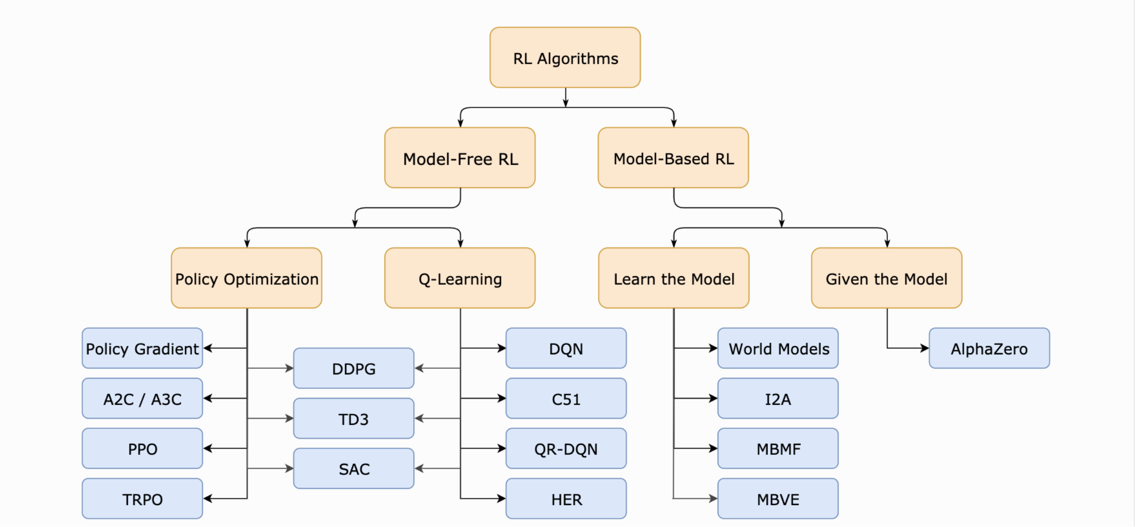

RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Abb. 2.3: Reinforcement learning algorithms taxonomy Quelle: Spinning up Ich werde ein ähnliches Bild machen, aber nur mit Algorithmen, die ich benutzt habe.

Reinforcement learning algorithms can be categorized in model-based and model-free. As the name suggests, model-based algorithms have a model of the environment and use it to find an optimal policy. There are some advantages and disadvantages of both approaches. Deisenroth et al. [9] talk about "robot control as a reinforcement learning problem": forming the trajectory of the robot, which is sequence of states and motor commands that lead to them. Model-free policy search methods usually use real robots to sample such trajectories, which in most cases requires human supervision and causes fast wear-and-tear of robots, especially the ones that are not industrial. What is more, it is time consuming. Model-based methods aim to develop efficiency by sampling some observation trajectories and building a model out of them. The model should decrease the number of real-robot manipulations and better adapt to new unseen environments or parameters. However, in practice, such models can be used not exact enough, which leads to learning a poor policy.

Bansal et al. [3] state that model-free reinforcement learning approaches are effective at finding complex policies, however they sometimes take very long time to converge. Model-based algorithms might be better at generalizing and reduce the number of steps to find an optimal policy, however without exact model the learned policy might be far from an optimal one. Also the model must be updated together with the policy.

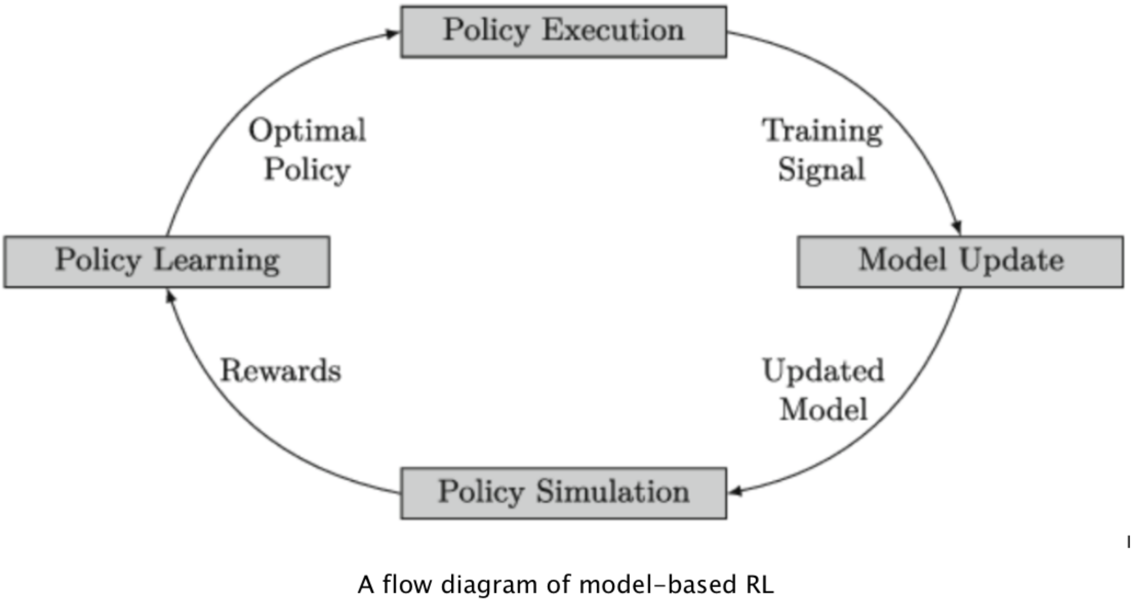


Abb. 2.4: Model-based RL Quelle: s

Model-free (?) reinforcement learning algorithms can also be divided in being on- and off-policy. Policy is used in reinforcement learning to decide which action to perform in the current state. While learning and building the policy up, the algorithm does not necessarily need to always chose the action that the latest version of the policy suggests. The chosen action might be for example the one that will maximize the value function for the state (????) as in Q-learning. In that case the algorithm is off-policy. In the opposite case, when algorithm strictly follows the policy while learning, it is an on-policy one.

2.3.1 Q-Learning

Q-learning is a model-free algorithm that is based on approximating an objective function in order to find the optimal policy.

$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \subset S} P(s_{t+1}|a_t, s_t) * \max \underset{a'}{Q} * (s', a')$

R(s,a) is the immediate reward in state s for executing action a.
Since $V^*(s) = \max_a Q^*(s,a)$, the optimal policy can be calculated as:
$arg\max_a Q^*(s,a)$
The strategy for choosing the action in the current state is ε-greedy: with the probability ε the chosen action will be calculated through the Q-function: a = $arg\max_a Q^*(s,a)$. With probability (1 - ε) the sampled action will be a random one from the action space: $a \in A(s)$.
The O-learning rule is:
$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s+1,a') - Q(s,a))$,
where α is the learning rate and $Q(s,a)$ is the old value.
Usually all state-action pairs are stored in a table, a so-called q-table. The agent refers to this table to select the best action - the action with the biggest q-value - for the state he is in.
An agent is exploiting if he uses the q-table and selects the action with the biggest Q-value to perform next. An agent is exploring, if he ignores the table values and takes a random action. Exploring is important as the agent finds other states with possibly better results, that would not be discovered if the agent strictly followed the table. Exploitation/exploration can be controlled by an ε-value - how often should the agent perform a random exploration step.
Reinforcement learning is often used for complex problems with a wide action and state space, which makes it impossible to store all Q-values in a table because of the a big amount of time for calculation of the values of the table and the amount of memory that would be required to save the table. Deep Q-Learning (DQN) is the variant of the Q-Learning which is one of the possibilities to deal with this problem with the help of a neural network as a function approximator.
The Q-values for all possible actions of the state are calculated, the one that maximizes the Q-Value is chosen as the next action.

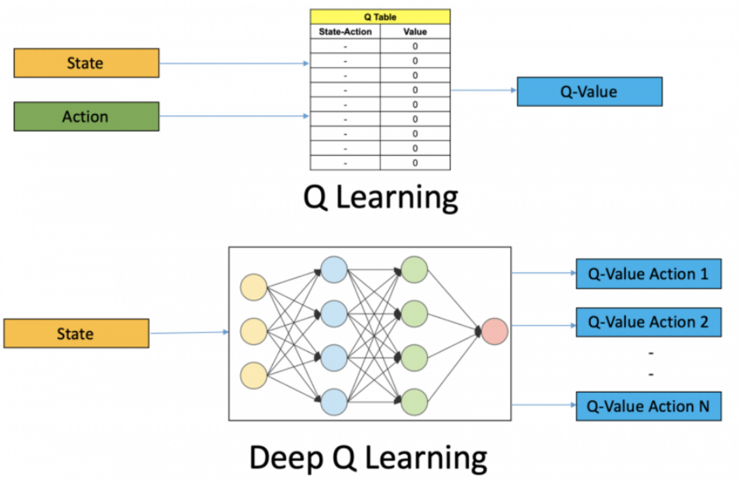


Abb. 2.5: TODO: make my own scheme

The updating rule is analogic to the the standard Q-learning approach:
$T = R(s,a) + \gamma \max_{a'} Q_\theta(s',a')$ - target value.
$\theta = \theta + \gamma \nabla_\theta E_{s' \sim P(s'|s,a)}[(T - Q_\theta(s,a))^2]$

$T - Q_\theta(s,a))^2$ is the temporal difference in form of the mean square error. The gradient of the error is used for calculating new weights for the network.
While the error is calculated only for one state, the gradient of the error impacts all weights in the network - all states. This makes the learning become unstable. In order to cope with this problem another network with the same architecture - target network - is used to compute the target value T. Target network is initialized with the same weights as the main function approximator network, and it is updated every defined number of steps.
Replay buffer is a technique that is also used in DQN. The agent's transitions (s_t, a_t, r_t, s_{t+1}) are saved to a replay memory D. After collecting some number of transitions, mini-batches containing random transitions from the replay buffer are sampled and then used for training the network with stochastic gradient descent(SGD). Due to the randomness of trajectories in mini-batches the learned knowledge does not tend to resemble only one type of trajectories. It is also more data-efficient as the experience data can be used multiple times for learning.
Replay buffer and target network make Q-learning stable and efficient. Q-learning is one of the most popular reinforcement learning methods as it is simple to implement. However, it has its disadvantages. [16] stated that because of the fact that it uses max operator to calculate the Q-value for the state, there are often significant overestimations of these values. Double Q-Learning is an off-policy value based reinforcement learning algorithm that uses two different Q-functions: one for selecting the next action and the other for value evaluation.

DDPG has the Actor&Critic architecture for policy network and Q-network.

2.3.2 Actor Critic

The "Criticëstimates the value function (Q function = action-value function or V function = state-value) Äctorupdates the policy distribution in the direction suggested by the Critic (such as with policy gradients)."
TODO Value-based methods are based on maximizing a Q-function that is usually given by Bellmann equation(TODO), the optimal policy can be then calculated as $\pi(s) = arg\max_a Q^*(s,a)$. Policy-based methods learn the policy directly without the value-function. The goal for policy-based approches is to maximize the cumulative reward of the trajectory(TODO Definition) $J(\theta) = E_{\tau\sim\pi_\theta[R_\tau]}$ as opposed to the value-based approach where the goal is to minimize the error function (which is normally in the form of the temporal difference error). Policy gradient searches for parameters that maximize the goal function by moving in the direction of the gradient - gradient accent: $\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\theta)$, where $\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta log \pi_\theta(a_t|s_t) R(\tau)]$, $R(\tau)$ is the cumulative reward of the trajectory. When the reward of the sampled trajectory is positive, the gradient of the goal function is also positive, the policy will be optimized in the direction of the gradient, this way it learns that the sampled trajectory was a good one and vice versa. Discrete stochastic policy gives as output success probabilities for all actions (Karams Folie). Continuous stochastic policy-based approach makes it possible to work with continuous action spaces. (stochastic policy gives probability distribution over all actions)
The idea of the actor-critic method is to use a Q-function instead of the (average??) cumulative

reward $R(\tau)$ - the Q-function of the state gives an expected future reward after completing action a. The approximated policy gradient will then be:
$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)Q_w(s,a)]$
where w are the weights of the neural network that calculates the Q-function. That is a separate network that is called "the critic". The network that calculates the goal function is the äctor". The critic estimates the value-function, the actor uses it to update its policy. TOTO actor-crtitic scheme from Sutton Barto find it it's good!!

TODO Policy, value iteration????

2.3.3 Soft Actor-Critic

Soft Actor-Critic(SAC) is an off-policy state-of-the art reinforcement learning algorithm, it outperformed other previous methods ([**?**].
As the name suggests, it is based on the actor-critic approach. The key idea of SAC is the actor trying to maximize not only the expected return but also the entropy. The algorithm is stable and also sample-efficient, it is capable of solving high-dimensional complex tasks.

- optimizes a stochastic policy
- central feature of SAC is entropy regularization
- entropy, a measure of randomness in the policy
- increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum.

Usually the main goal of reinforcement learning approaches is to maximize the expected sum of rewards:
$\sum_t E_(s_t,a_t) \sim \ \rho_\pi[r(s_t,a_t)]$

In SAC the goal of the algorithm is to find a policy that will maximize the objection:
$J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\sim\ \rho_\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))]$

α is the temperature parameter that controls the influence of the entropy term on the reward function which means it "controls the stochasticity of the optimal policy"??????
Entropy is a measure of randomness in the policy. The term 'entropy' comes from the area of information theory and means the amount of information or 'surprise' of the possible outcome of the variable. When the outcome of some source of data is rather unexpected because it has less probability, it's entropy is high.
Wikipedia:
Given a random variable X, with possible outcomes x_i, each with probability $P_X(x_i)$, the entropy H(X) of X:

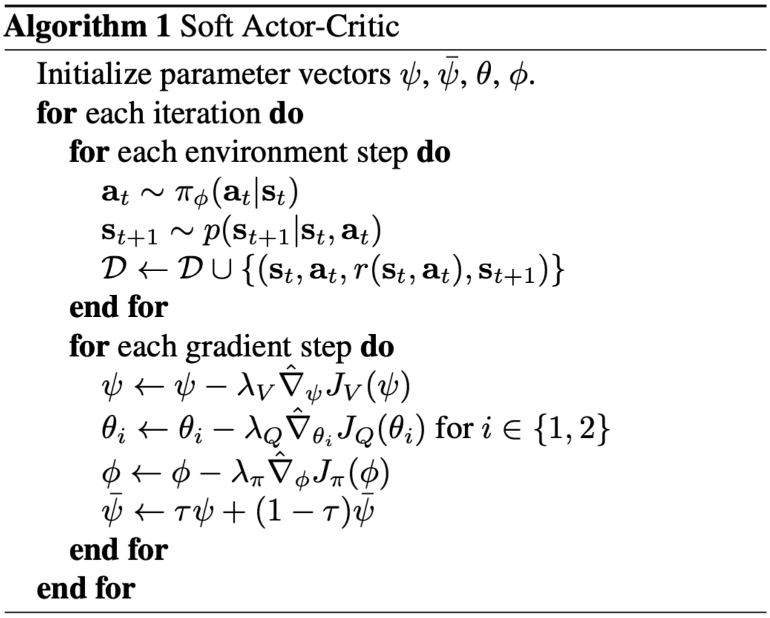
$H(x) = -\sum_i P_X(x_i)log_b P_X(x_i) = \sum_i P_X(x_i)I_X(x_i) = E[I_X]$

The higher the entropy value is, the more exploration the algorithm will perform. It can accelerate learning and avoid the premature(????) converging of the policy in a bad local minimum.

Entropy regularization.
Entropy of x from its distribution P:
$H(P) = -\underset{x\sim P}{\log} P(x)$

Quote openai: Ïn entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy at that timestep. This changes the RL problem"

Three functions to be learned:
Parameterized state value function $V_\psi(s_t)$, soft Q-function $Q_\theta(st,at)$, and a tractable policy $\pi_\phi(at|st)$. The parameters of these networks are ψ, θ, and ϕ.



Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
for each iteration do
    for each environment step do
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
        $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t,\mathbf{a}_t,r(\mathbf{s}_t,\mathbf{a}_t),\mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1,2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
    end for
end for
```

Abb. 2.6: Custom environment example for using stable baselines

Quote paper: D is the distribution of previously sampled states and actions, or a replay buffer.

Value network:
$V(s_t) = E_{a_t\sim\pi}[Q(s_t,a_t) - log\pi(a_t|s_t)]$

The soft value function is trained to minimize the squared residual error:
$J_V(\psi) = E_{s_t\sim D}[\frac{1}{2}(V_\psi(s_t) - E_{a_t\sim\pi_\phi}[Q_\theta(s_t,a_t) - log\pi_\phi(a_t|s_t)])^2]$
+ Target value network $V_{\bar{\psi}}$

The soft Q-function parameters can be trained to minimize the soft Bellman residual:
$Q(s_t,a_t) = r(s_t,a_t) + \gamma E_{s_{t+1}}[V_\psi(s_{t+1})] - targetQvalue$

$J_Q(\theta) = E_{(s_t,a_t)\sim D}[\frac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2]$

$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma E_{s_{t+1}\sim p}[V_{\hat{\psi}}(s_{t+1})]$

The Q-function is learned on-policy => Value function as well.

Two Q-functions with θ_i are trained independently to optimise $J_Q(\theta_i)$. The minimum of two functions is used in computing the value gradient $\hat{\nabla}_\psi J_V(\psi)$ and policy gradient $\hat{\nabla}_\phi J_\pi(\phi)$.

Quote paper: The policy parameters can be learned by directly minimizing the expected KL-divergence. $\pi_{new} = arg \min_{\pi' \in \Pi} D_{KL}(\pi'(\cdot|s_t)||\frac{exp(Q^{\pi_{old}}(s_t,\cdot))}{Z^{\pi_{old}(s_t)}})$

$J_\pi(\phi) = E_{s_t\sim D}[D_{KL}(\pi(\cdot|s_t)||\frac{exp(Q_\theta(s_t,\cdot))}{Z_\theta(s_t)})]$

Reparameterizatin trick:

Normally actions are sampled $a_t \sim \pi_\phi$

the policy is reparameterised using network transformation:

$a_t = f_\phi(\epsilon_t; s_t)$

for example $a_t = \mu_\phi(s_t) + \epsilon_t \sigma_\phi(s_t)$ where $\epsilon_t \sim N(0,1)$, $\mu_\phi(s_t)$ is the mean action, $\sigma_\phi(s_t)$ variance.

So instead of sampling $a_t \sim \pi_\phi(s_t)$ we can sample $\epsilon_t \sim N(0,1)$. Then:

$Q_\theta(s_t,a_t) = Q_\theta(s_t, \mu_\phi(s_t) + \epsilon_t \sigma_\phi(s_t))$ -smaller variance.

The network computes $\mu_\phi(s_t)$ and $\sigma_\phi(s_t)$

Due to the reparameterizatin trick:

$J_\pi(\phi) = E_{s_t\sim D, \epsilon_t \sim N}[log\pi_\phi(f_\phi(\epsilon_t;s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t;s_t))]$ where ϵ_t is the input noise vector , sampled from some fixed distribution, such as a spherical Gaussian.

polyak-averaging???

3 Approach

3.0.1 System components

Mujoco was chosen as simulation environment as it has shown faster and more accurate performance than other simulators. As a base for the simulation the gym "FetchPickAndPlace-v0ënvironment was used. Implementation of the reinforcement learning algorithms SAC and PPO were taken from the stable baseline library.

3.0.2 Mujoco

Mujoco (Multi-Joint dynamics with Contact) is a physics engine developed for model-based control ([37]). It was developed at "Movement control laboratory", University of Washington for research projects in the area of 'optimal control, state estimation and system identification", as none of already existing tools delivered a satisfying performance. Mujoco has shown an especially more stable, fast and accurate performance for robotic tasks in comparison to other physics simulators [13].

Mujoco is user-convenient as well as computation efficient. The model can be specified in a an XML-file format. The visualization is interactive and done by the rendering library OpenGL. Some of the key features of Mujoco include:

- Generalized coordinates combined with modern contact dynamics
- Soft, convex and analytically-invertible contact dynamics
- Separation of model and data

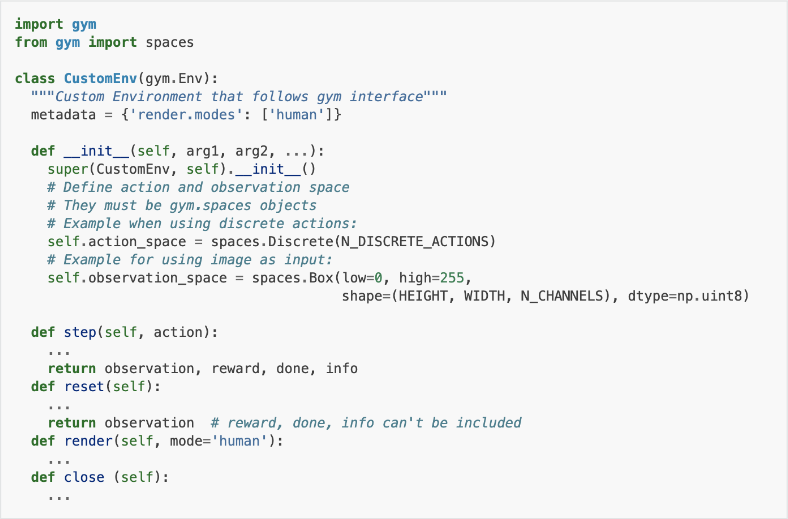
and many more. Further description of Mujoco and its documentation can be found on the official website [2].

3.0.3 OpenAI Gym

Gym is toolkit developed by OpenAI for researching in the area of reinforcement learning. Gym is an open-source library which includes collection of environments for different reinforcement learning problems. They include Atari (i.e. Breakout-v0) and Board games (Go), Robotics(HandManipulateBlock-v0) and many more [8]. The user can update the environment and construct own agent, which would only have to implement the specified interface to use the environment.

3.0.4 Stable Baselines

OpenAi baselines [10] is a library developed by OpenAI containing implementations of different reinforcement learning algorithms. Stable baselines [**?**] is a collection of improved RL algorithms based on OpeanAI baselines. The algorithms of stable baselines have unified structure, are better documented, there are more tests for them as well as some new new ones. They can be used for gym environments. Stable baselines also include a set already pretrained agents [32].
In order to train the agent using reinforcement learning algorithms from stable baselines the environment must follow the gym interface: it must implement methods __init__(), step(), reset(), render(), close() and inherit from gym.Env:



```
import gym
from gym import spaces

class CustomEnv(gym.Env):
  """Custom Environment that follows gym interface"""
  metadata = {'render.modes': ['human']}

  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()
    # Define action and observation space
    # They must be gym.spaces objects
    # Example when using discrete actions:
    self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)
    # Example for using image as input:
    self.observation_space = spaces.Box(low=0, high=255,
                                        shape=(HEIGHT, WIDTH, N_CHANNELS), dtype=np.uint8)

  def step(self, action):
    ...
    return observation, reward, done, info
  def reset(self):
    ...
    return observation  # reward, done, info can't be included
  def render(self, mode='human'):
    ...
  def close (self):
    ...
```

Abb. 3.1: Custom environment example for using stable baselines

4 Implementation

4.0.1 Simulation Setup

As a base for the simulation the gym "FetchPickAndPlace-v0ënvironment was used: "Fetch has to pick up a box from a table using its gripper and move it to a desired goal above the table". The step() function was adjusted in order to modulate following behavior: the movement starts with gripper having a constant height z above the table. The action is expressed as [x, y], where x and y are cartesian coordinates of the target gripper x and y positions. The gripper goes to (x,y) preserving its height z. Afterwards the planar grasp takes place: the gripper goes down, changing only its z-coordinate and having maximal jaw opening. Then the gripper closes and goes back up. The camera was adjusted to take an RGB-image of the part of the table where the object can be located:

Abb. 4.1: Observation from the camera

The 81x81 pixels RGB image represents the state of the environment. The agent receives reward = 1 iff the object is grasped successfully. The success of grasping is determined as following: if the gripper's jaws are not entirely closed after the grasp attempt, there is something preventing them from closing, which means that the object if located in the gripper above the table: it was grasped successfully.

max_episode_steps is a parameter that defines how many times the step() function can be called before the episode ends - how many times can robot attempt to grasp in the current episode.

If not explicitly stated, the parameters for training SAC are:

4.0.2 CNN structure

4.1 System Architecture

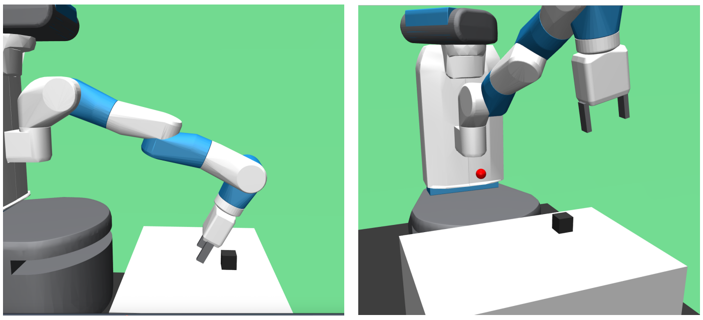
5 Evaluation

5.0.1 Simulation Inaccuracies

The Angle Of The Gripper

The action space consists of target (x,y) coordinates of the point that the gripper has to achieve before going down. The step() function computes the difference between target and current positions, the difference is then applied to the simulation using inverse kinematics. The inverse kinematics is calculated by solving the convex optimization problem. The optimization problem is defined combining description of bodies that are used in the simulation and physical constraints such as forces, friction, etc.

One of the challenges that was faced during the set-up is that there can be many solutions to the problem of gripper achieving the target (x,y) position. In some cases the gripper developed an angle in different plains, successfully reaching target position but the angle made it impossible to compute a successful grasp.



Setting the robot to the starting position for every step helped to correct that inaccuracy for the current task. However, this approach is time-consuming and might be not applicable for the similar problems in different set-ups. The possible better solution to this problem might be to set additional constraints for the position of joints of the robot that would effect the solution of the inverse kinematics problem.

Slipping Of The Object

The first version of grasping consisted of completely closing the gripper and going up with an object. However, the object did not stay in the gripper - it slipped down. This behavior was not expected as physical forces such as friction were set to the default values. The solution to the problem was to keep closing the gripper while going up - it helps to prevent the object from slipping.

Results of the training

5.0.2 Experiments with the Soft-Actor-Critic Algorithm

Several experiments using Soft-Actor-Critic(SAC) were conducted. The results were unstable: repeating the same experiment with same parameters delivered different results. In some cases the training was successful. However sometimes the network was showing good performance which dropped after some training steps, resulting in 0% success rate at the end. Saving the model several times during training helped to retrieve the model that was giving a good performance. In some trainings the agent never achieved good performance. There might be several reasons for such unstable behavior, they will be discussed later on.

The implementation of SAC in stable_baselines only works with continuous action space, so the action space for the robot in these experiments was a continuous, represented through gym Space Box: $self.action\_space = spaces.Box(np.array([1.1, 0.55]), np.array([1.45, 0.95]), dtype='float32')$.

1 Constant Positions Of The Object

Among successful experiment runs, where the position of the object was learned even under 10k training steps, there were some that that were not stable. As an example a training that lasted 30k steps. After 14k steps the agent was performing with 100% sucessrate. However shortly after the performance dropped and never recovered.

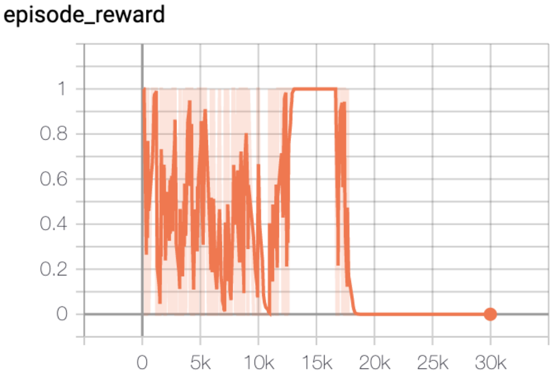


Abb. 5.2: An example of unstable behavior of the algorithm: after 14k training steps the agent learned correctly where the object is located. Then the performance dropped to 0 and did not recover after that.

20 Constant Positions Of The Object

A list of 20 (x,y) coordinates in the range of robot's action space were randomly generated. During each episode the object is located on the table with its (x,y) coordinates one random position from the list. The goal of the experiment was to determine whether the network is able to learn how to grasp the object.

The training statistics is:

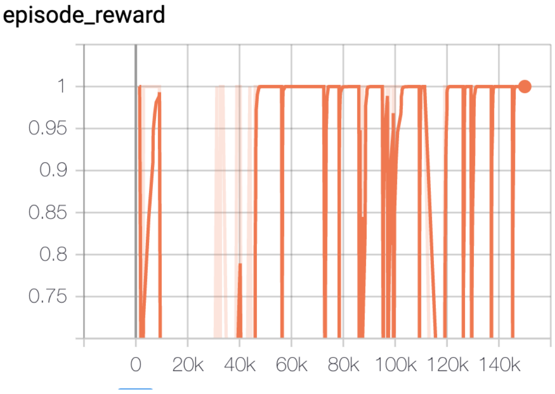


Abb. 5.3: After about 45k training steps the success rate was almost always 1. The reason it dropped in some episodes might be due to inaccuracy in actions or some object positions occurred for the first time - the agent never saw them before which lead to bad performance, they were learned after that

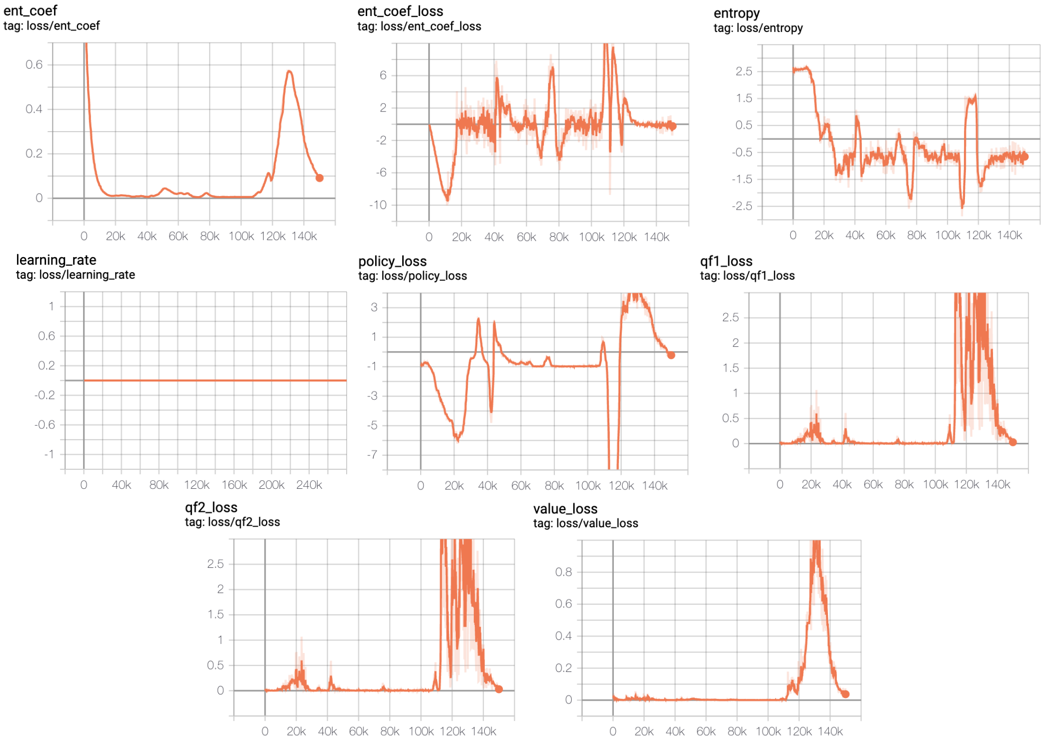


Abb. 5.4: Statistics of the training: object's position is one of 20 possible ones, learn to grasp the object in 150k steps. At the end of the training the entropy is becoming constant which means that the algorithm is sure which action to take. The entropy coefficient is going down as well because the agent does not need to do much exploration anymore. Policy loss is going to zero, as well as the losses from both Q-networks and the target value network, which means the weights of the network are adjusted in an optimal way to achieve success.

The evaluation consisted of 300 episodes, 100% success rate - the task was completed successfully.

The 150000 steps of the training resulted in 43998 episode steps.As max_episode_steps value was set to 50, in the beginning the majority of episodes lasted 50 steps, however at the end of the training the value shrank to 1, occasionally going up.

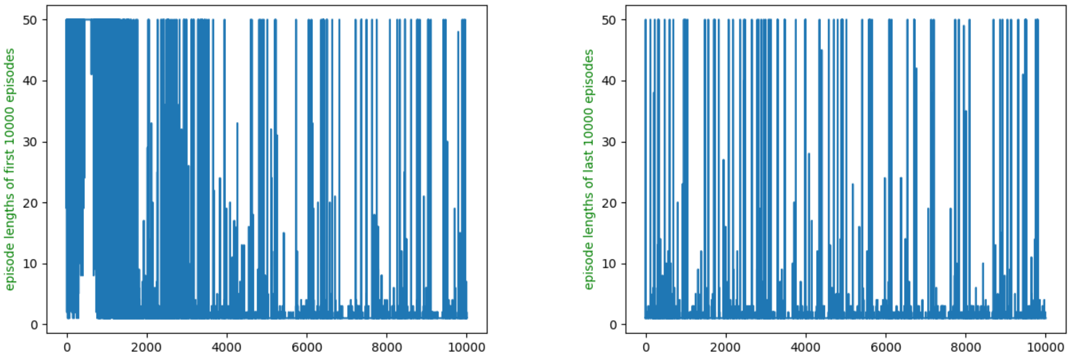


Abb. 5.5: The first image shows the statistics of the length of first 10k episodes of the training, the second one - of the last 10k. It is noticeable, that in the second case tha majority of episodes were short because the agent has already learned most of positions of the objects. In case of inaccuracy of actions or incorrect assumptions about the location of the object the episode lasted longer - up to 50 steps

50 Constant Positions Of The Object

The same experiment showed different results. In some cases the agent did not learn at all. Below there are two examples of training in which the agent reached relatively good performance. The first one is a successful learned model which achieved success rate 100%. The second one showed 90% success rate during evaluation with standard deviation approximately 30% - the agent was not always sure which action to take.

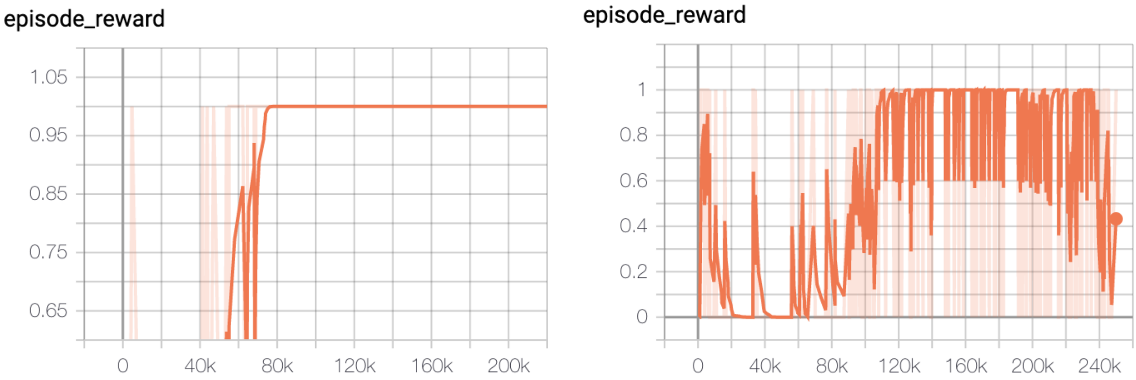


Abb. 5.6: The same experiment that differed only in number of training steps. The first image shows that in the first experiment the agent learned positions successfully after 80k steps, in the second experiment the agent did not manage to succeed even after 250k steps of training. The evaluation proved it: the first agent always graspes the object, the second one only in 90% of cases with a high uncertainty of 30%.

The training statistics show, that the entropy factor became constant in the first experiment, in the second one it variated:

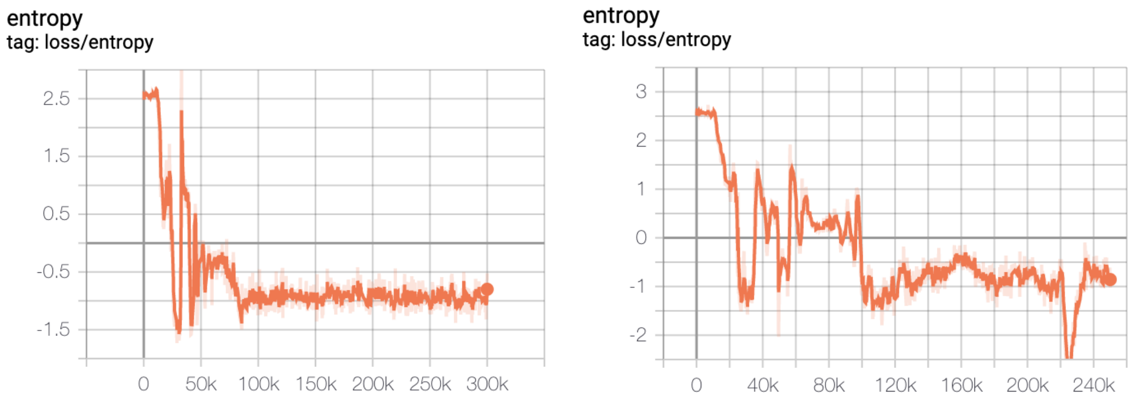


Abb. 5.7: Entropy loss of the successful agent became constant after about 80k steps - when the agent learned the task. In the second experiment the agent was not sure which action to take.

An unexpected behavior showed the statistics about the value loss. In the case of the successful agent the value loss was extremely high (from 1e+4 to 5e+5), going up and then down during training, which did not effect the agent's success rate, it stayed 100%. The not so successful agent's value loss was ranging from 0 to 1.

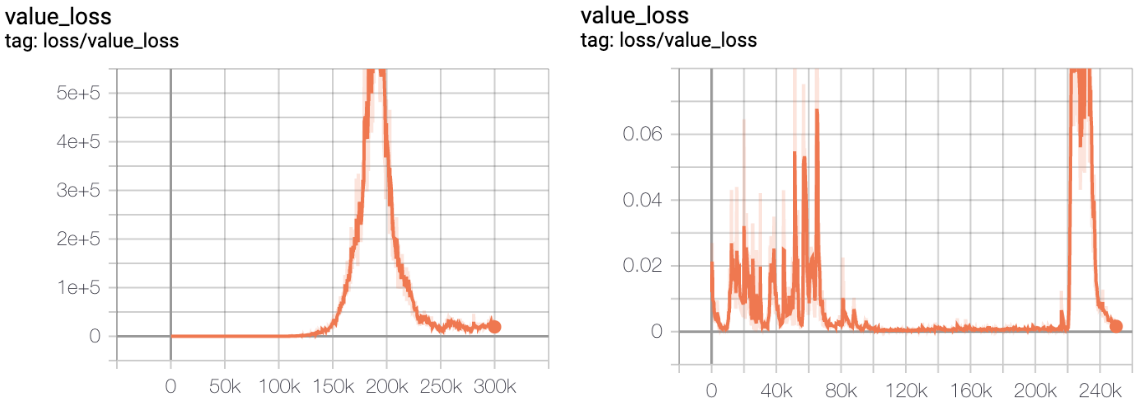


Abb. 5.8: Value loss statistics of two experiment.

Random Position Of The Object

In previous experiments the agent was expected to learn finite number of positions, which should have been an easy task because he could just learn them by heart. In case of a random position of the object, it is impossible to learn all combinations as there are endless.

After 400k steps of training during the evaluation the agent performed with 97% success rate and 17% variance.

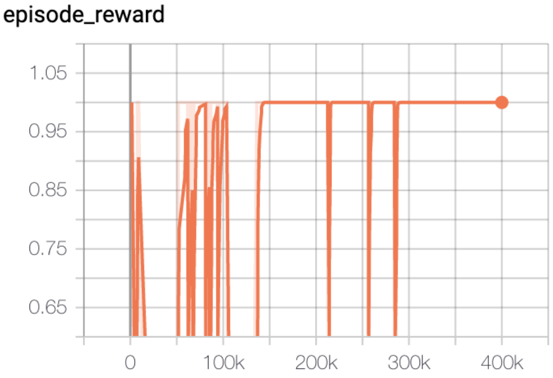


Abb. 5.9: 400k step training of SAC on random position of the object

Reasons for Unstable Training with the Soft-Actor-Critic Algorithm

Soft-Actor-Critic is a state-of-the-art reinforcement learning algorithm, it showed impressive results on such gym tasks as Humanoid-v2 and Ant-v2 [15]. The current grasping task is much more simple with a smaller action space, which is why the instable training results are very unlikely to be caused by the algorithm. Another reason could be the instablity of the environment. However, training with PPO was stable: repeating the experiments delivered same expected results, which means the task can be trained in the created environment. Another assumption about the cause of unstable behavior is the implementation of SAC by stable_baselines. The actual reason should be discovered in future work.

5.0.3 Experiments with the Proximal Policy Optimization Algorithm

Continuous action space

The algorithm performed very poorly in the continuous action space ($self.action\_space = spaces.Box(np.array([1.1, 0.55$
$float32')$.). It failed to learn even the simplest task where the object's position does not change.

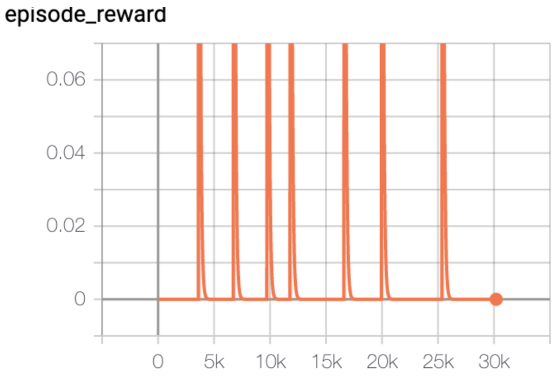


Abb. 5.10: PPO performed extremely bad on the task without being able to learn the simplest set-up where the position of the object is constant during the whole training. In the first graphics there are some cases of reward 1 - there are so rare that they are most likely accidental.

Discrete action space

Following the idea of Zeng et al. in [41], [40], the action space was discretized to consist of a 50x50 grid. The action (x,y) would mean the gripper would execute a planar grasp in the middle of the cell (x,y). The action space is represented by the gym.Space.MultiDiscrete:
$self.action\_space = spaces.MultiDiscrete([49,49])$
It then is translated to the coordinate system to accord to a grid on the table within the observation space.
The results of the experiments with 1, 20 and 50 constant positions of the object were successful: during evaluation for all these cases the agent performed with almost 100% success rate.

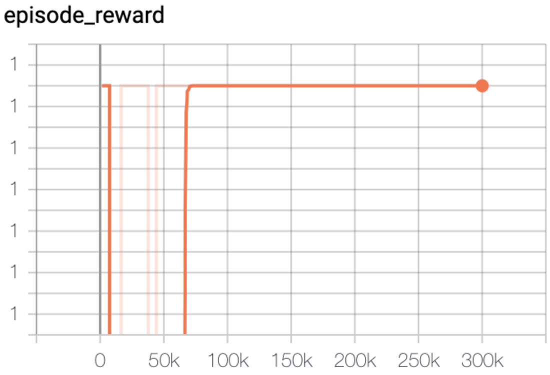


Abb. 5.11: 300k step training of PPO on random position of the object

For random object position after 400k training steps the success rate was 98% with 11% variance. The model created in another training with 500k steps showed 0.995% success rate with 0.07% variance during 1000 evaluation steps.

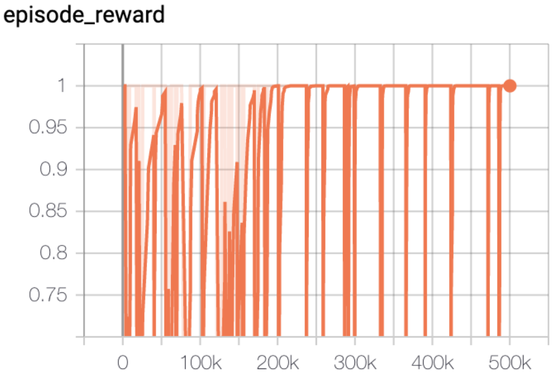


Abb. 5.12: 300k step training of PPO on random position of the object

The idea of discretizing the action space was formulated in [36]. Although the idea is simple, it can drastically improve the performance of baseline on-policy algorithms.

Liste der noch zu erledigenden Punkte

A Abbildungsverzeichnis

2.1 Outtake from "Computing 3-D Optimal Form-Closure Grasps"[11] of Ding et al. 5

2.2 Classification of different aspects that influence the problem of the grasping problem according to [5] of Bohg et al. 6

2.3 Reinforcement learning algorithms taxonomy Quelle: Spinning up Ich werde ein ähnliches Bild machen, aber nur mit Algorithmen, die ich benutzt habe. 12

2.4 Model-based RL Quelle: s 13

2.5 TODO: make my own scheme 14

2.6 Custom environment example for using stable baselines 17

3.1 Custom environment example for using stable baselines 20

4.1 Observation from the camera 21

5.2 An example of unstable behavior of the algorithm: after 14k training steps the agent learned correctly where the object is located. Then the performance dropped to 0 and did not recover after that. 24

5.3 After about 45k training steps the success rate was almost always 1. The reason it dropped in some episodes might be due to inaccuracy in actions or some object positions occurred for the first time - the agent never saw them before which lead to bad performance, they were learned after that 25

5.4 Statistics of the training: object's position is one of 20 possible ones, learn to grasp the object in 150k steps. At the end of the training the entropy is becoming constant which means that the algorithm is sure which action to take. The entropy coefficient is going down as well because the agent does not need to do much exploration anymore. Policy loss is going to zero, as well as the losses from both Q-networks and the target value network, which means the weights of the network are adjusted in an optimal way to achieve success. 25

5.5 The first image shows the statistics of the length of first 10k episodes of the training, the second one - of the last 10k. It is noticeable, that in the second case tha majority of episodes were short because the agent has already learned most of positions of the objects. In case of inaccuracy of actions or incorrect assumptions about the location of the object the episode lasted longer - up to 50 steps 26

5.6 The same experiment that differed only in number of training steps. The first image shows that in the first experiment the agent learned positions successfully after 80k steps, in the second experiment the agent did not manage to succeed even after 250k steps of training. The evaluation proved it: the first agent always graspes the object, the second one only in 90% of cases with a high uncertainty of 30%. . . . 26
5.7 Entropy loss of the successful agent became constant after about 80k steps - when the agent learned the task. In the second experiment the agent was not sure which action to take. 27
5.8 Value loss statistics of two experiment. 27
5.9 400k step training of SAC on random position of the object 28
5.10 PPO performed extremely bad on the task without being able to learn the simplest set-up where the position of the object is constant during the whole training. In the first graphics there are some cases of reward 1 - there are so rare that they are most likely accidental. 28
5.11 300k step training of PPO on random position of the object 29

B Tabellenverzeichnis

C Literaturverzeichnis

[1]

[2]

[3] S. Bansal, R. Calandra, K. Chua, S. Levine, and C. Tomlin. Mbmf: Model-based priors for model-free reinforcement learning. *arXiv preprint arXiv:1709.03153*, 2017.

[4] J. Bohg, M. Johnson-Roberson, B. León, J. Felip, X. Gratal, N. Bergström, D. Kragic, and A. Morales. Mind the gap-robotic grasping under incomplete observation. In *2011 IEEE International Conference on Robotics and Automation*, pages 686–693. IEEE, 2011.

[5] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2013.

[6] A. Boularias, J. A. Bagnell, and A. Stentz. Learning to manipulate unknown objects in clutter by reinforcement. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[9] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[10] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[11] D. Ding, Y.-H. Liu, and S. Wang. Computing 3-d optimal form-closure grasps. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 4, pages 3573–3578. IEEE, 2000.

[12] S. Ekvall and D. Kragic. Learning and evaluation of the approach vector for automatic grasp generation and planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4715–4720. IEEE, 2007.

[13] T. Erez, Y. Tassa, and E. Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[14] C. Goldfeder and P. K. Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011.

[15] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[16] H. V. Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[17] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[18] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[19] A. Kasper, Z. Xue, and R. Dillmann. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012.

[20] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, et al. Opengrasp: a toolkit for robot grasping simulation. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 109–120. Springer, 2010.

[21] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[22] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[23] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[24] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[25] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust

grasp planning using a multi-armed bandit model with correlated rewards. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1957–1964. IEEE, 2016.

[26] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[27] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. 2004.

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

[29] V.-D. Nguyen. Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, 1988.

[30] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[31] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE, 2018.

[32] A. Raffin. Rl baselines zoo. `https://github.com/araffin/rl-baselines-zoo`, 2018.

[33] J. Redmon and A. Angelova. Real-time grasp tection using convolutional neural networks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1316–1322. IEEE, 2015.

[34] A. Sahbani, S. El-Khoury, and P. Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

[35] K. B. Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[36] Y. Tang and S. Agrawal. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*, 2019.

[37] 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. IEEE, 2012.

[38] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*, 2018.

[39] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[40] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019.

[41] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. 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)*, pages 4238–4245. IEEE, 2018.

[42] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.