Bachelor thesis

Student: Anna Fedorchenko
Supervisor: M.Sc. Atanas Tanev
Advisor: Prof. Dr.-Ing. Rüdiger Dillmann

April 9, 2020

Current title of the thesis

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

Problem statement

The focus of the thesis is to develop an approach for the robot to learn how to grasp rigid objects using images from RGB-D 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 and reinforcement learning. The training and testing will be conducted in a simulated environment. In case of successful performance of the developed algorithm in the simulation, the next step will be to prepare the transition to the real-world environment with the help of domain randomization.
One of the advantages of such algorithm is the ability to generalize to new objects. Previous research shows that data-driven algorithms that are based on machine learning have successful rates of grasping objects that were never used in training. Another advantage is the fact that no human labeling will be required. This spares time and effort.
Training the robot in real world is time consuming and expensive, simulation makes it possible to avoid these costs. However, there is a so-called "reality gap" problem, when the algorithm trained in simulation does not perform well in the real world scenario. One of the possible solutions to this problem is using domain randomization and/or fine-tuning with training in reality. The sim-to-real transfer is an additional optional part of the thesis, that will take place only if the algorithm performs well in simulation.

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 some warehouse scenarios robots have to pick an object and then perform different actions on it: lift, shift, put on a specific position. Similar process takes place in the scenario of a kitchen robot, that is being developed for assisting people at cooking. A kitchen robot might be asked to bring a glass of water or pick a utensil. In order for a robot to pick an object it has to grasp it first. It is crucial for the grasping part to succeed in order to be able to complete the proceeding steps of the manipulation.
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. Even more difficult would grasping be for the kitchen robot, where it is impossible to always predefine positions of objects that must be grasped.

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 some cases, the object is singulated []. Objects can be already known in advance with existing 3D meshes of them (i.e.in the industry scenario) or never seen before. There are a some variations to the grasping problem such as: the type of the object to be grasped (rigid/non-rigid, transparent, having certain shape, color,..), the number of objects (cluttered environment, a couple of objects, a single one), whether the object is known, familiar or completely unknown, the type of the gripper (two-finger, five-finger,...), the type of the camera and so on.
The grasping problem has been profoundly researched throughout the last decades. Many different approaches have been developed. Since the beginning of the 21century, Machine Learning has been used in more and more of them. ——¿ Übergang zum Reinforcement learning 404 not found —— Reinforcement learning, one of the types of Maschine Learning, has shown promising results in business strategy planning(???), continuous control problems(?) and creating artificial intelligence for computer games. Reinforcement learning approach is similar to how humans learn. One of the simple everyday tasks that every human learns is catching a ball. Either we watch somebody else doing it or try it out ourselves. We automatically in the course of split seconds estimate the size and mass of the ball, what our position should be in order to catch it. The result of catching - positive or negative - influences our behavior for the next time when we have to perform the same task. (????)

Related work

Sahbani et al. [1] 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[2], [3]. "Computing 3-D Optimal Form-Closure Grasps" [4] 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.
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.
One of the strategies is to compare the candidate grasp to (human) labeled examples. Redmon et al. [6] use the Cornell grasping dataset [7] 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. [8] 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 [9]). For every object robust grasps covering the surface were sampled, using the antipodal grasp sampling approach developed in Dex-Net 1.0 [10]). 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 [8]. 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. [11] 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. [11] 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.
—– reinforcement learning
——

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) [12]:

- 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:
$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}\left[R(\tau)\,|s_0 = s, a_0 = a\right]$
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):
$Q^{*}(s,a) = \max_{\pi} \underset{\tau\sim\pi}{E}\left[R(\tau)\,|s_0 = s, a_0 = a\right]$
, where policy π is optimal.
There is a connection between value and action-value functions, that is helpful for some calculations:
$V^{\pi}(s) = \underset{a\sim\pi}{E}\left[Q^{\pi}(s,a)\right]$ - 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

1 Algorithms in reinforcement learning

There is a wide variety of reinforcement learning algorithms:

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. [13] 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

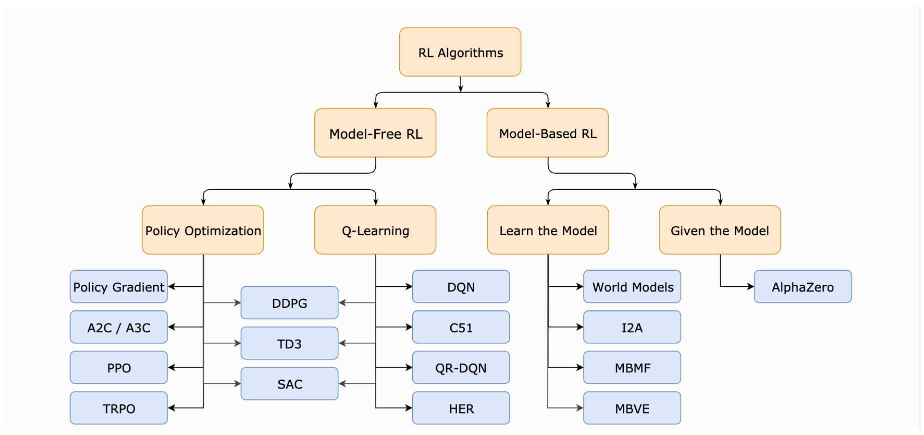

Figure 1: Reinforcement learning algorithms taxonomy Quelle: Spinning up Ich werde ein ähnliches Bild machen, aber nur mit Algorithmen, die ich benutzt habe.

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. [14] 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.

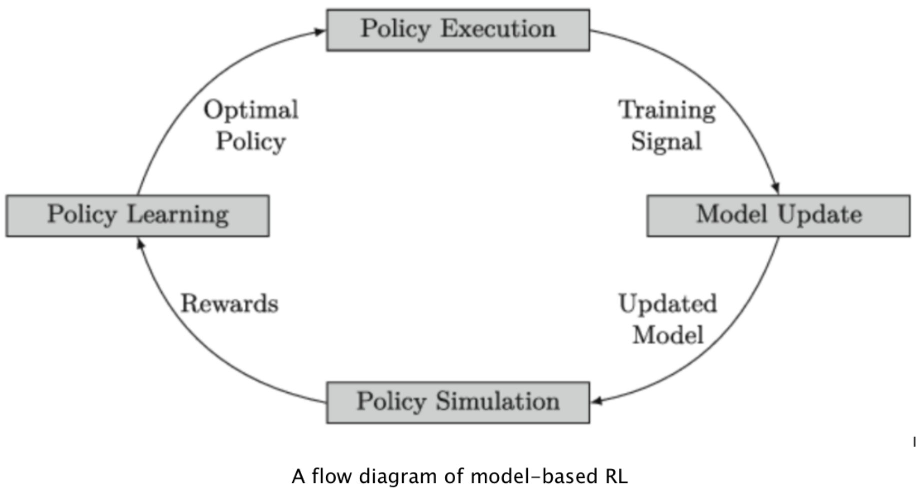

Figure 2: Model-based RL Quelle: s

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.

1.1 Q-Learning

Q-learning is a model-free algorithm that is based on approximating an objective function in order to find the optimal policy.
$Q^*(s,a) = R(s,a) + \gamma \sum\limits_{s' \subset S} P(s_{t+1}|a_t, s_t) * \max\limits_{a'} Q*(s',a')$
R(s,a) is the immediate reward in state s for executing action a.
Since $V^*(s) = \underset{a}{\max} Q^*(s,a)$, the optimal policy can be calculated as:
$arg\underset{a}{\max} Q^*(s,a)$
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 = $arg\underset{a}{\max} Q^*(s,a)$. 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:
$Q(s,a) = Q(s,a) + \alpha(r + \gamma\underset{a'}{\max} Q(s+1,a') - Q(s,a))$,
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.
The updating rule is analogic to the the standard Q-learning approach:
$T = R(s,a) + \gamma\underset{a'}{\max} Q_\theta(s',a')$ - 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

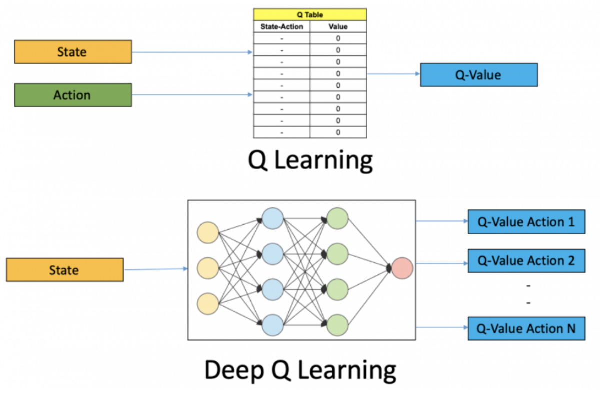

Figure 3: TODO: make my own scheme

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. [15] 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.

1.2 Temporal-Difference Learning

Temporal-Difference Learning (TD)
TODO

1.3 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy algorithm similar to Q-Learning - it learns an optimal action-value function $Q^*(s, a)$. The main difference between these two algorithms is that DDPG works with continuous action spaces and Q-Learning with discrete ones. For discrete action spaces there are finite number of actions, so Q-values for all possible actions can be calculated and compared, the biggest one would be the max value. However this approach would not work in continuous action spaces as there are endless number of possible actions. .....TODO

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

1.4 Actor Critic

The "Critic" estimates the value function (Q function = action-value function or V function = state-value) "Actor" "updates 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.
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 "actor". The critic estimates the value-function, the actor uses it to update its policy.

1.5 SAC

TODO

1.6 HER

TODO

Approach

Simulation

In data-driven approaches training data is required to learn for a successful grasp. Levine et al. [16] used 14 robotic arms that sampled 800 000 grasp attempts. Pinto and al. [11] 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. [17] created a grasp planner containing database of grasps for 3D models of different objects, that were generated using GraspIt! [18] simulation engine. Kasper et al. [9] introduced the system for digitizing objects. In year 2010 the OpenGRASP [19] simulation toolkit for grasping and dexterous manipulation was created.
James et al. [20] 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. [21] 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. [6] stated that "pretraining large convolutional neural networks greatly improves training time and helps avoid overfitting".

Results of the training

The movement of the robot consists of several steps: the robot rotates the gripper to the angle α, after that it goes to the target (x,y) coordinate. The z-coordinate of the gripper stays constant during the first two parts of the movement. After reaching the target positions: (x,y, α), the robot executes a planar grasp: it goes down to almost reach the height of the table with the max gripper opening, then it closes the jaws and goes up. In the final position, when the gripper is above the table, the width between the jaws of the gripper is calculated. If it is not null, then the object was successfully grasped.
In reality the simulation engine showed an unexpected behavior: when the gripper was down, closed the jaws and successfully grasped the object, the object did not stay through the whole way up, it slipped from the jaws. According to Mujoco documentation the type of numerical algorithm for solving convex optimization problems - solver - could be changed to reach the desirable behavior as well as some other parameters such as "$noslip_iterations$", "cone", "impratio". However tuning in those parameters did not help to solve the problem. The method that helped was to slightly close the gripper on the way up.
The observation input was an RGB 81x81 image, the action space continuous: $x \in [1.1, 1.45]$, $y \in [0.55, 0.95]$, $\alpha \in [0, 90]$.
In the beginning the algorithm was tested for the simplified task: the test object was a black cube with constant position on the table, within the reach of the robot arm.

The first algorithm tested was DDPG with 50 000 steps. The expected outcome was that the model will memorize the position of the cube and will always predict the correct grasping action.
After

References

[1] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

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

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

[4] Dan Ding, Yun-Hui Liu, and Shuguo 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.

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

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

[7]

[8] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken 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.

[9] Alexander Kasper, Zhixing Xue, and Rüdiger 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.

[10] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken 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.

[11] 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)*, pages 3406–3413. IEEE, 2016.

[12] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[13] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[14] Somil Bansal, Roberto Calandra, Kurtland Chua, Sergey Levine, and Claire Tomlin. Mbmf: Model-based priors for model-free reinforcement learning. *arXiv preprint arXiv:1709.03153*, 2017.

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

[16] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre 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.

[17] Corey Goldfeder and Peter K Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011.

[18] Andrew T Miller and Peter K Allen. Graspit! a versatile simulator for robotic grasping. 2004.

[19] Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, Sami Moisio, Jeannette Bohg, James 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.

[20] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos 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.

[21] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt 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.