Learning to grasp objects with deep learning and convolution neural networks

Rik Timmers - S2245809

October 10, 2016

Master Project Proposal

Artificial Intelligence
University of Groningen, The Netherlands

First supervisor:
Dr. Marco Wiering (Artificial Intelligence, University of Groningen)
Second supervisor:
MSc. Amirhosein Shantia (Artificial Intelligence, University of Groningen)

university of groningen / faculty of mathematics and natural sciences

1 Introduction

Service robots are slowly becoming more popular, mostly small robots with limited functionality like vacuum cleaner robots. Bigger service robots that could help in a household environment are still much in development. These service robots need to be able to perform many complex tasks, in all kinds of environments, while doing them safely by not colliding with objects, humans and itself. Tasks like navigating, speech recognition, following/recognizing humans, object detection/recognition and manipulation are all important parts for a service robot, and these tasks are tested in competitions like the RoboCup [1] and RoCKin [2]. The difficulty in programming these robots is that they need to operate in all kinds of different environments, no household is the same, and this makes it hard to make sure that the robot can operate safely and correctly in situations that is has never seen before. This means that the robot needs to have intelligent behaviour so it can cope with any environment it encounters.
In this project we will focus on the manipulation and perception part of a service robot by making use of reinforcement learning so that the robot has learned how to grasp an object that it has seen with its camera. Common used approaches for grasping objects with service robots are currently by using simple cartesian control, or by making use of a planner that creates a path for the arm to traverse so it can grasp an object, like MoveIt [1] which uses the Open Motion Planning Library [3] that has many sampling based motion planning algorithms. The downside of using a planner is that it requires to make new calculations every time it needs to grasp an object, where sometimes it can not find a valid plan, or it takes too long to find a valid plan for the arm to move by.
Since the real world is complex a robot needs to be able to perform grasping tasks in an always changing world. It will have to work in an environment that is has not seen before. Instead of having an algorithm that calculates the best way to grasp an object in an ever changing environment, it is better to learn how to grasp, and be able to apply this learned behaviour in a complex environment. Although for this project we will focus on learning how to grasp an object, it will not focus on a complex environment. In order to learn grasping we need to start simple first and show that it works in the real world. We will limit our grasping to simple shaped objects that are easy to grasp, taking into account that we only have a simple gripper, a two finger gripper without sensors, available for real world testing. We will use Reinforcement Learning with an actor-critic network for the learning of the control network, and make use of Convolution Neural Networks (CNN) [4] for the perception of the environment, locating the object.

2 Theoretical framework

Reinforcement Learning is an area in machine learning in which an agent needs to take actions in an environment which provides the highest reward for that agent. The environment of a Reinforcement Learning problem is described as a Markov Decision Process (MDP) [5]. An MDP consists out of States, Actions, Rewards, and a Transition that maps how from a current State, by performing an Action, an agent gets into a new State. In this project we will use continuous

[1] MoveIt: `http://moveit.ros.org`

actions instead of discrete actions. The goal is to find a policy that will provide the best action based on the current state. For this purpose we will use the Continuous Actor Critic Learning Automaton (CACLA) algorithm [6]. CACLA is created to handle the continuous action space, which is needed to control a robotic arm. An action in this case is the velocity of a single actuator (joint) of the robotic arm, which can move with different speeds. CACLA is a model-free algorithm, meaning that the agent does not need knowledge about the environment beforehand, it can therefore be used in all sorts of problems, like controlling a robotic arm. An important part of Reinforcement Learning is exploration, this enables the algorithm to explore different policies, and from this the best policy can be found. The CACLA paper suggests two exploration methods. The first one is the ϵ-greedy method, it will select a random action with ϵ probability. The second method is Gaussian exploration, this method will always explore, but takes the actual action output as a mean and will add some Gaussian noise to that action.
To improve learning, extra parameters will be added to the CACLA algorithm to test their performance. For example we can look at the insights from the Deep Q Network (DQN) [7]. DQN shows for example that it is possible for large, non-linear networks to become robust and stable by training networks off-policy by using samples from a replay buffer to minimize the correlations between samples. In the 'Continuous control with deep reinforcement learning' paper [8], that also uses a actor-critic network for learning continuous control, they make use of batch normalization [9], a recent advance in deep learning. Also methods like dropout [10] to reduce overfitting will be tested.
For the perception part, object localization, we will make use of CNNs. CNNs do not learn features directly from the input image but they learn kernels that each can represent a feature. CNN are successfully used in object recognition and speech recognition [11]. In this project we want to use CNNs to determine the location of an object that the arm can grasp. The authors of [12] show it is possible to perform object localization, alignment and classification in one deep neural network, where the authors of [13] show that they can determine the 3D pose of a hand by using a depth image and CNNs, we can use this information to possibly determine the 3D location and classification of an object.

3 Research question

The goal of this project is to create a neural network that can control a robotic arm so that it is able to grasp an object that it has seen by using a 3D camera. The main algorithm for learning to control the arm will be the CACLA algorithm, we will use a CNN for the perception of objects that the arm needs to grasp. The project will be divided into two aspects, control and perception. For the control part the focus will be on creating a neural network that can grasp an object given by an operator manually. The perception part will focus on the detection of an object and extracting the location and orientation needed to grasp the object. The final stage is to make the control part depending on the perception part so it can autonomously grasp objects that are seen by the camera. Another aspect in this research project is to look at different architectures. Instead of having one neural network that controls the arm, perhaps a better result or faster training can be achieved by having multiple networks

where each network controls a single joint or smaller groups of joints, a network used for approaching the object and one network used for the actual grasping part, separate the CNN and the control neural network or combine both CNN and control neural network into one big neural network.
We will compare this method to an existing inverse kinematic controller that is being used by the MoveIt package. We will look at the speed, the time it takes the inverse kinematics to calculate and execute a grasp versus the immediate execution of the neural network controller. We will look at accuracy, is the controller able to grasp the object and how good are the grasps of each controller, whether it is holding the object correctly or just barely grasped it. Also we will look at the robustness, how well do the controllers perform in different situations.
We will first focus on just grasping a single object from a scene so we don't require an advanced object recognizer to separate objects. If these methods are working correctly and there is time left we can focus on muliple objects in a scene and an object recognizer to determine locations for each object.

The main research question:

- How does deep reinforcement learning compare to a much simpler (inverse) kinematics controller for grasping objects? Is there a gain in speed, accuracy or robustness?

The sub-questions:

- Which architecture results in the best performance?
- Does training in simulation result in correct behaviour on the real robot?

4 Methods

To control the arm we will make use of the CACLA algorithm. Where the goal is to be able to grasp an object, first we will start with getting the end-effector in a correct position and orientation. This will provide a good insight in the parameters like neural network size, learning rates, exploration, and rewards. In this stage we can already look at different architectures for controlling the arm. Instead of having one network to control all the joints in a robotic arm, we could have N networks to either control one joint per network, or multiple joints per network. The idea behind this is that the first joints on a robotic arm will greatly influence the position of the end-effector, where the last joints will have less influence on the final position, but will be mostly used for orientation and small adjustments of the end-effector. With a multi network architecture we can also define different reward functions which might help to decrease the training time and or accuracy.
The next step is to actually grasp an object, this includes using the fingers of the end-effector. Again we can look at different architectures, can one network do all tasks; move to the correct position and grasp, or should there be two networks; one for bringing the end-effector close to the object, and one for the actual grasping part.
The next stage is to look at the perception part, using CNNs to locate objects. CNNs are mostly used for object recognition, but because CNNs can find many

different features, specially in deep CNNs, it should be possible to extract 3D positional information from them. We will look at different input features like RGB, RGBD (D stands for depth), and Pointcloud data. To extract the ground truth for training the position we can make use of the simulation, we can set the position of an object using code, so we know the exact 3D pose of the object, but we can also use the Pointcloud data to get the 3D points that belong to an object by using segmentation and clustering, from there we can get either the min and max values of the x,y,z points, or get the center point of the object in 3D space. We can make use of transformations to transform the camera's coordinates system to the arm's coordinates system. In this case we can assume that there is only one object in the scene that can be grasped, since the network has a fixed number of outputs and is not able to give more than one location. There could be more objects in the scene but these are objects that the arm should avoid and are assumed to be static.
The next step is to combine the controller network and the perception network so that it is possible to grasp an object that is being seen by a camera. Two different architectures will be tested, in one architecture the CNN will be directly connected to the control network, in this case the CNN will recognize a single object that it needs to grasp and also take into account the rest of the environment for obstacle avoidance. The second architecture will be the same as the previous, a CNN connected to the control network, but the control network will have extra inputs namely the x, y, z coordinate of the object that needs to be grasped. This coordinate can either come from the current CNN that is connected to the control network, or can come from an external object recognizer, which will allow the network to grasp objects from a scene with multiple objects. In the last case the CNN doesn't need to perform object localization, but it still needs to know about the objects that it can grasp, and which objects it should completely avoid.
For the inverse kinematic controller we will make use of MoveIt, this package can calculate a path and grasping motions for a robotic arm, it also can take into account collision avoidance based on camera input. For this project we will mostly make use of simulation, but with a final goal that it should work on a real robotic arm. The real robotic arm is a Kinova Mico arm [2], with 6 Degrees of Freedom (DoF), and a 2 finger gripper. For the simulation environment we will make use of V-REP [14], this simulator already contains the Mico arm, but it also has other robotic arms available. The possible problem with the Mico arm is that the last 3 joints are in a weirdly bended structure, which may create difficulties for the control network to train correctly. We will therefore also look at different robotic arms like for example the UR10 [3], and KUKA LBR iiwa 14 R820 [4] to see if the design of a robotic arm has any influence on the training of the control network.
To create the programs we will mostly make use of Python, and a bit of C++. For creating the neural networks we make use of Tensorflow [15], Tensorflow is a framework that makes it easy to create different kind of networks, and automatically use the GPU to train them. Since the framework works best in Python, the rest of the agent will also be programmed in Python. The connection to

[2]Mico arm: `http://www.kinovarobotics.com/service-robotics/products/robot-arms/`
[3]UR10: `http://www.universal-robots.com/nl/producten/ur10-robot/`
[4]KUKA: `http://www.kuka-robotics.com/en/products/industrial_robots/sensitiv/lbr_iiwa_14_r820/`

V-REP is also possible in Python. For the connection to the real robotic arm C++ is required. Although most software for controlling the arm is already written. For the perception of the real world a Kinect V2 camera will be used. This camera provides 1080p images, and also 3D information. To receive images from the camera and to control the arm via Python the Robotic Operating System (ROS) [16] will be used.

5 Planning

September (starting at the end of September)

- Finalize research proposal
- Extended literate research (Batch normalization, drop-out, soft-target updates)
- Setup for real world testing (Arm and Camera positions), and simulation setup.
- Setup computers for running experiments and scripts to run remotely

October

- Framework for control network, including different architecture setups (without optimizations like batch normalization, etc)
- Framework for testing the control network, in both simulation and on the real arm (Mico arm only)
- Get MoveIt to work with Mico arm in simulation and on the real arm.
- Thesis writing

November

- Start with experiments for control network (Learn to go to position)
- Extend Framework with optimizations (Batch normalization, drop-out, etc)
- Experiments with optimizations (Learn to go to position), different architectures, different arms, grasping
- Thesis writing

December

- Finalize control network fine-tuning, should have a working network for grasping objects based on manual input.
- Experiments comparing inverse kinematic controller versus control network.
- Thesis writing

January

- Extend framework to work with CNN for perception part
- Experiments with getting 3D position of objects, test if simulation training works in real world
- Create large dataset of objects (script for creating objects in simulation, or create real world dataset)
- Experiments with RGB, RGBD, and Pointcloud as input
- Thesis writing

February

- Combine Perception network with Control network
- Experiment with different architectures, different arms
- Finalize experiments with complete working system so that real arm can grasp objects
- Experiments comparing inverse kinematic controller versus complete network
- Thesis writing

March

- Finalize experiments with different arms grasping objects in simulation
- Extra time for experiments in case needed
- Start writing final version Thesis

April

- Finalize Thesis

References

[1] T. Wisspeintner, T. Van Der Zant, L. Iocchi, and S. Schiffer, "Robocup@ home: Scientific competition and benchmarking for domestic service robots," *Interaction Studies*, vol. 10, no. 3, pp. 392–426, 2009.

[2] S. Schneider, F. Hegger, A. Ahmad, I. Awaad, F. Amigoni, J. Berghofer, R. Bischoff, A. Bonarini, R. Dwiputra, G. Fontana, *et al.*, "The rockin@ home challenge," in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pp. 1–7, VDE, 2014.

[3] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. http://ompl.kavrakilab.org.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[6] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 272–279, IEEE, 2007.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[10] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[11] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015.

[12] D. Lin, X. Shen, C. Lu, and J. Jia, "Deep lac: Deep localization, alignment and classification for fine-grained recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1666–1674, 2015.

[13] M. Oberweger, P. Wohlhart, and V. Lepetit, "Hands deep in deep learning for hand pose estimation," *arXiv preprint arXiv:1502.06807*, 2015.

[14] M. F. E. Rohmer, S. P. N. Singh, "V-rep: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.