Deep Reinforcement Learning Applied to a Robotic Pick-and-Place Application

Natanael Magno Gomes[1][0000−0002−2444−375X], Felipe N. Martins[1][0000−0003−1032−6162], José Lima[2][0000−0001−7902−1207], and Heinrich Wörtche[1,3][0000−0003−2263−0495]

[1] Sensors and Smart Systems group, Institute of Engineering, Hanze University of Applied Sciences, The Netherlands

[2] The Research Centre in Digitalization and Intelligent Robotics (CeDRI), Polytechnic Institute of Bragança, Portugal
Centre for Robotics in Industry and Intelligent Systems — INESC TEC, Portugal

[3] Dep. Electrical Engineering, Eindhoven University of Technology, The Netherlands

Abstract. Industrial robot manipulators are widely used for repetitive applications that require high precision, like pick-and-place. In many cases, the movements of industrial robot manipulators are hard-coded or manually defined, and need to be adjusted if the objects being manipulated change position. To increase flexibility, an industrial robot should be able to adjust its configuration in order to grasp objects in variable/unknown positions. This can be achieved by off-the-shelf vision-based solutions, but most require prior knowledge about each object to be manipulated. To address this issue, this work presents a ROS-based deep reinforcement learning solution to robotic grasping for a Collaborative Robot (Cobot) using a depth camera. The solution uses deep Q-learning to process the color and depth images and generate a ϵ-greedy policy used to define the robot action. The Q-values are estimated using Convolutional Neural Network (CNN) based on pre-trained models for feature extraction. Experiments were carried out in a simulated environment to compare the performance of four different pre-trained CNN models (RexNext, MobileNet, MNASNet and DenseNet). Results show that the best performance in our application was reached by MobileNet, with an average of 84 % accuracy after training in simulated environment.

Keywords: Cobots · Reinforcement Learning · Computer Vision · Pick-and-Place · Grasping

1 Introduction

The usage of robots has been increasing in the industry for the past 50 years [1], specially in repetitive tasks. Recently, industrial robots are being deployed in applications in which they share (part of) their working environment with people. Those type of robots are often referred to as Cobots, and are equipped with safety systems according to ISO/TS 15066:2016 [2]. Although Cobots are easy to setup and program, their programs are usually written manually. If there is a change in the position of objects in their workspace, which is common when humans also interact with the scene, their program needs to be adjusted. Therefore, to increase flexibility and to facilitate the implementation of robotic automation, the robot should be able to adjust its configuration in order to interact with objects in variable positions.

A Robot manipulator consists of a series of joints and links forming the arm, at the far end are placed the end-effectors. The purpose of an end-effector is to act on the environment, for example by manipulating objects in the scene. The most common end-effector for grasping is the simple parallel gripper, consisting of two-jaw design.

Grasping is a difficult task when different objects are not always in the same position. To obtain a grasping position of the object, several techniques have been applied. In [3] a vision technique is used to define candidate points in the object and then triangulate one point where the object can be grasped.

With the evolution of the processing power, Computer Vision (CV) has also played an important role in industrial automation for the last 30 years, including depth images processing [4]. CV has been applied from food inspection [5] [6] to smartphone parts inspection [7]. Red Green Blue Depth (RGBD) cameras are composed of a sensor capable of acquiring color and depth information and have been used in robotics to increase the flexibility and bring new possibilities. There are several models available e.g. Asus Xtion, Stereolabs ZED, Intel RealSense and the well-known Microsoft Kinect. One approach to grasping different types of objects using RBGD cameras is to create 3D templates of the objects and a database of possible grasping positions. The authors in [8] used dual Machine Learning (ML) approach, one to identify familiar objects with spin-image and the second to recognize an appropriate grasping pose. This work also used interactive object labelling and kinesthetic grasp teaching. The success rate varies according to the number of known objects and goes from 45% up to 79% [8].

Deep Convolutional Neural Networks (DCNNs) have been used to identify robotic grasp positions in [9]. It uses RGBD image as input and gives a five-dimensional grasp representation, with position (x, y), a grasp rectangle (h, w) and orientation θ of the grasp rectangle with respect to horizontal axis. Two DCNNs Residual Neural Networks (ResNets) with 50 layers each are used to analyse the image and generate the features to be used on a shallow CNN to estimate the grasp position. The networks are trained against a large dataset of known objects and their grasp position.

Generative Grasping Convolutional Neural Network (GG-CNN) is proposed in [10], a solution fast to compute, capable of running real-time 50Hz. It uses DCNN with just 10 to 20 layers to analyse the images and depth information to control the robot in real time to grasp objects, even when they change position on the scene.

In this paper we investigate the use of Reinforcement Learning (RL) to train an Artificial Intelligence (AI) agent to control a Cobot to perform a given pick-and-place task, estimating the grasping position without previous knowledge about the objects. To enable the agent to execute the task, an RGBD camera is used to generate the inputs for the system. An adaptive learning system was implemented to adapt to new situations such as new configurations of robot manipulators and unexpected changes in the environment.

2 Theoretical Background

In this section we present a summary of relevant concepts used in the development of our system.

2.1 Convolutional Neural Networks

CNN is a class of algorithms which use the Artificial Neural Network in combination with convolutional kernels to extract information from a dataset. The convolutional kernel scans the feature space and the result is stored in an array to be used in the next step of the CNN.

CNN have been applied in different solutions in machine learning, such as object detection algorithms, natural language processing, anomaly detection, deep reinforcement learning among others. The majority of the CNN application is in the computer vision field with a highlight to object detection and classification algorithms. The next section explores some of these algorithms.

2.2 Object Detection and Classification Algorithms

In the field of artificial intelligence, image processing for object detection and recognition is highly advanced. The increase of Central Processing Unit (CPU) processing power and the increased use of Graphics Processing Unit (GPU) have an important role in the progress of image processing [11].

The problems of object detection are to detect if there are objects in the image, to estimate the position of the object in the image and predict the class of the object. In robotics the orientation of the object can also be very important to determine the correct grasp position. A set of object detection and recognition algorithms are investigated in this section.

Several features arrays are extracted from the image and form the base for the next layer of convolution and so on to refine and reduce dimensionality of the features, the last step is a classification Artificial Neural Network (ANN) which is giving the output in a form of certainty to a number of classes. See figure 1 where a complete CNN is shown.

The learning process of a CNN is to determine the value of the kernels to be used during the multiple convolution steps. The learning process can take up to hours of processing a labeled data set to estimate the best weights for the specific object. The advantage is once the model weights have been determined they can be stored for future applications.

In [13] a Regions with Convolutional Neural Networks (R-CNN) algorithm is proposed to solve the problem of object detection. The principle is to propose

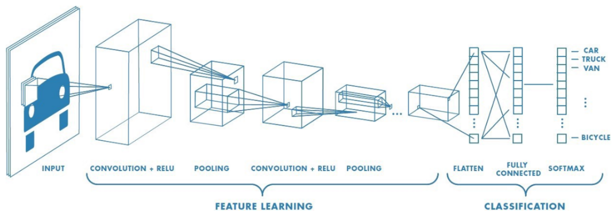

Fig. 1: CNN complete process, several convolutional layers alternate with pooling and in the final classification step a fully connected ANN [12].

around 2000 areas on the image with possible objects and for each one of these extract features and analyze with a CNN in order to classify the objects in the image.

The problem of R-CNN is the high processing power needed to perform this task. A modern laptop is able to analyze a high definition image using this technique in about 40 seconds, making it impossible to execute real time video analysis. But still capable of being used in some applications where time is not important or where it is possible to use multiple processors to perform the task, since each processor can analyze one proposed region.

An alternative to R-CNN is called Fast R-CNN [14] where the features are extracted before the region proposition is done, so it saves processing time but loses some abilities to parallel processing. The main difference to R-CNN is the unique convolutional feature map from the image.

The Fast R-CNN is capable of near real time video analysis in a modern laptop. For real time application there is a variation of this algorithm proposed in [15] called Faster R-CNN. It uses the synergy of between steps to reduce the number of proposed objects, resulting in an algorithm capable of analyzing an image in 198 ms, sufficient for video analysis. Faster R-CNN has an average result of over 70% of correct identifications.

Extending Faster R-CNN the Mask R-CNN [16] [17] creates a pixel segmentation around the object, giving more information about the orientation of the object, and in the case of robotics a first hint to where to pick the object.

There are efforts to use depth images with object detection and recognition algorithms as shown in [18], where the positioning accuracy of the object is higher than RGB images.

2.3 Deep Reinforcement Learning

Together with Supervised Learning and Unsupervised Learning, RL forms the base of ML algorithms. RL is the area of ML based on rewards and the learning process occurs via interaction with the environment. The basic setup includes the agent being trained, the environment, the possible actions the agent can take and the reward the agent receives [19]. The reward can be associated with the action taken or with the new state.

Some problems in RL can be too large to have exact solutions and demand approximate solutions. The use of deep learning to tackle this problem in combination with RL is called Deep Reinforcement Learning (deep RL). Some problems can require more memory than available, i.e., a Q-table to store all possible solutions for an input color image of 250x250 pixels would require $250 \times 250 \times 255 \times 255 \times 255 = 1.036.335.937.500$ bytes, or 1 TB. For such large problems the complete solution can be prohibitive by the required memory and processing time.

2.4 Deep Q learning

For large problems, the Q-table can be approximated using ANN and CNN to estimate the Q values. Deep Q Learning Network (DQN) was proposed by [20] to play Atari games on a high level, later this technique was also used in robotics [21] [22]. A self balanced robot was controlled using DQN in a simulated environment with performance better than Linear–quadratic regulator (LQR) and Fuzzy controllers [23]. Several DQNs have been tested for ultrasound-guided robotic navigation in the human spine to locate the sacrum with [24].

3 Proposed System

The proposed system consists of a collaborative robot equipped with a two-finger gripper and a fixed RGBD camera pointing to the working area. The control architecture was designed considering the use of DQN to estimate the Q-values in the Q-Estimator. RL demands multiple episodes to obtain the necessary experience. Acquiring experience can be accelerated in a simulated environment, which can also be enriched with data not available in the real world. The proposed architecture shown in Figure 2 was designed to work in both simulated and real environments to allow experimentation on a real robot in the future.

The proposed architecture uses Robot Operating System (ROS) *topics* and *services* to transmit data between the learning side and the execution side. The boxes shown in blue in Figure 2 are the ROS drivers, necessary to bring the functionalities of the hardware to the ROS environment. The execution side can be simulated, to easily collect data, or real hardware for fine tuning and evaluation. As in [22], the action space is defined as motor control and the Q-values correspond to probability of grasp success.

The chosen policy for the RL algorithm is a ε-greedy, i.e., pursue the maximum reward with ε probability to take a random action. R-Estimator estimates the reward based on the success of the grasp and the distance reached to the objects, following equation 1.

$$\mathcal{R}_t = \begin{cases} \dfrac{1}{d_t + 1}, & \text{if } 0 \leq d_t \leq 0.02 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where d_t is in meters.

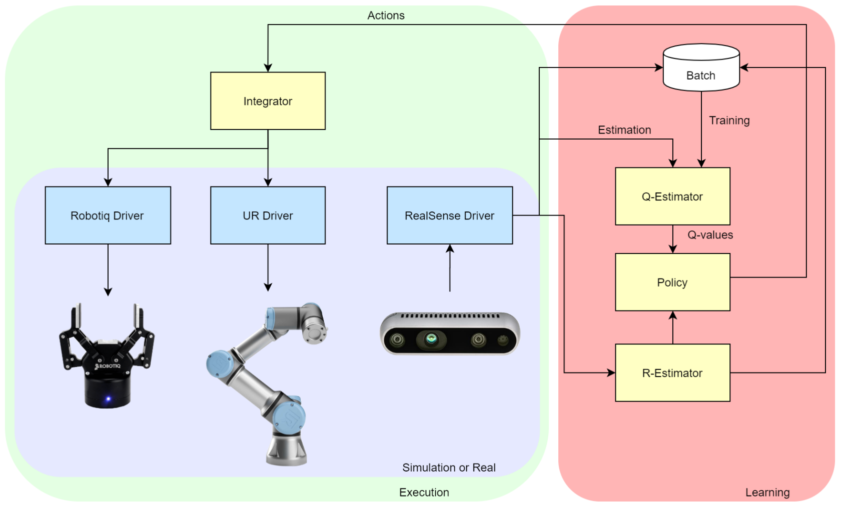

Fig. 2: Proposed architecture for grasp learning, divided in execution side (left) and learning sides (right). The modules in blue are ROS Drivers and the modules in yellow are Python scripts.

3.1 Action Space

The RL gives freedom to choose the possible actions the agent can choose, in this work actions are defined as the possible positions to attempt to grasp an object inside the work area, defined as:

$$\mathcal{S}_a = \{v, w\}, \tag{2}$$

where $\{v\}$ is the proportional position inside the working area in the x axis and $\{w\}$ is the proportional position inside the working area in the y axis. The values are discretized by the output of the CNN.

3.2 Convolutional Neural Network

To estimate the Q-values a CNN is used. For the action space $\mathcal{S}_a$ the network consists of two blocks to extract features from the images, a concatenation of the features and another CNN to reach the Q-values. The feature extraction blocks are pre-trained Pytorch models where the final classification network is removed. The layer to be removed is different for each model and, in general, the fully connected layers are removed. Four models were selected to compose the network, DenseNet, MobileNet, ResNext and MNASNet. The criteria considered the feature space and the performance of the models.

The use of pre-trained PyTorch models reduces the overall training time. However it brings limitations to the system, the size of the input image must be 224 by 224 pixels and the image must be normalized following the original

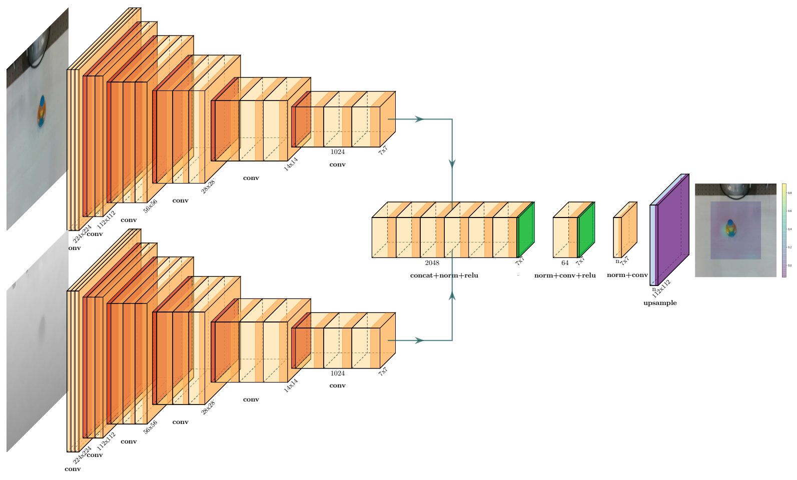

Fig. 3: The CNN architecture for the action space $\mathcal{S}_a$, the two main blocks are a simplified representation of pre-trained Densenet model [25], only the feature size is represented. The features from the Densenet model are concatenated and used to feed the next CNN, the result is an array with Q-values used to determine the action.

dataset mean and standard deviation [26]. In general this limits the working area of the algorithm to an approximately square area.

3.3 Simulation environment

The simulation environment was built on Webots, an open-source robotics simulator [27]. The choice has been made considering the usability of the software and use of computational resources [28]. To enclose the simulation in the ROS environment some modules were implemented: Gripper Control, Camera Control and a Supervisor to control the simulation. The simulated UR3e robot is connected to ROS using the ROS driver provided by the manufacturer and controlled with the Kinematics module. Figure 4 shows the the simulation environment, in which the camera is located in front of the robot, pointing to the working area. A feature of the simulated environment is to have control over all objects positions and colors. The positions were used as information for the reward and the color of the table was changed randomly at each episode to increase robustness during training. For each attempt the table color, the number of objects and the position of the objects were randomly changed.

Webots Gripper Control The Gripper Control is responsible to read and control the position of the joints of the simulated gripper. It controls all joints,

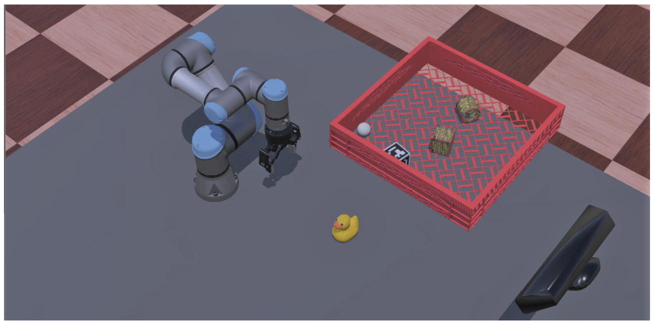
Fig. 4: The virtual environment built on Webots: it consists of a table, a UR3e collaborative robot, a camera and the objects used in the training.

motors and sensors of the simulated gripper. Touch sensors were also added at the tip of the finger to emulate the feedback signal when an object is grasped.

The Robotiq 2F-85 is the gripper we are going to use in future experiments with the real robot. It consists of 6 rotational joints intertwined to form the 2 fingers. During tests, the simulation of the closed kinematic chain of this gripper in Webots was not stable. To regain stability in simulation we used a gripper with simpler mechanical structure but with similar dimensions of the Robotiq 2F-85. The gripper used in simulation is shown in detail in Figure 5.

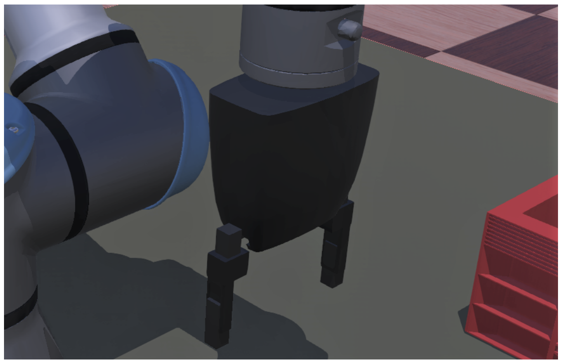
Fig. 5: Detail of the gripper used in the simulation: its appearance is based on the Kuka Youbot gripper and its bounding objects are simplified to blocks.

Webots Supervisor The Supervisor is responsible for resetting the simulation, preparing the position of the objects at the beginning of the episode, changing color of the table and publishing the position of objects to the reward estimator. To estimate the distance between the center of the end-effector and the objects, a

GPS position sensor is placed in the gripper's center to inform its position to the supervisor. The position of the objects is used to shape the reward proportional to the distance between the end-effector and the object. Although this information is not available in the real world they are used to speed up the simulation training sessions.

Webots Camera The camera simulated in Webots has the same resolution of the Intel RealSense camera. To avoid the need of calibration of the depth camera, both RGB and depth cameras had coincident position and field of view in simulation. The field of view is the same as the Intel RealSense RGB camera: 69.4 degrees or 1.21 radians.

3.4 Integrator

The Integrator is responsible for connecting all modules, simulated or real. It controls the Webots simulation using the Supervisor API and feed the RGBD images to the neural network.

Kinematics module The kinematics module controls the UR3e robot, simulated or real. It contains several methods to execute the calculations needed for the movement of the Cobot.

Although RL has been used to solve the kinematics in other works [29] [22], this is not the case in our system. Instead, we make use of analytical solution of the forward and inverse kinematics of the UR3e [30]. The Denavit–Hartenberg parameters are used to calculate forward and inverse kinematics of the robot [31]. Considering the UR3e has 6 joints, the combination of 3 of these can give $2^3 = 8$ different configurations which can give the same pose of the end-effector (elbow up and down, wrist up and down, shoulder forward and back). On top of that, the movement of the UR3e joints have a range from -2π to $+2\pi$ rad, increasing the possible solution space to $2^6 = 64$ different configurations to the same pose of the end-effector. To reduce the problem, the range of the joints is limited via software to $-\pi$ to $+\pi$ rad, but still giving 8 possible solutions from where the nearest solution to the current position is selected.

The kinematics module is capable of moving the robot to any position in the work space avoiding unreachable positions. To increase the usability of the module functions with the same behavior of the original Universal Robots "MOVEL" and "MOVEJ" have been implemented.

To estimate the cobot joints angles in order to position the end-effector in space the Tool Center Point (TCP) must be considered in the model. TCP is the position of the end-effector in relation to the robot flange. The real robot that will be used for future experiments has a Robotiq wrist camera and a 2F-85 gripper, which means that the TCP is 175.5 mm from the robot flange in the z axis [32].

4 Results and Discussion

This section shows the results and discussion of two training sections with different methods. The tests were performed on a laptop with a i7-9750H CPU, 32 GB RAM and a GTX 1650 4GB GPU, running Ubuntu 18.04. Although the GPU was not used in the CNN training, the simulation environment made use of it.

4.1 Modules

All modules were tested individually to ensure proper functioning. The ROS communication was tested using the builtin tool `rqt`, to check the connection between *nodes* via *topics* or *services*. The UR3e joints positions are always published in a topic and controlled via action client. In the simulation environment, the camera images, the gripper control and the supervisor commands are made available via ROS *services*. Differently from ROS *topics*, ROS *services* only transmit data when queried, decreasing the processing demanded by Webots. Figure 6 shows the *nodes* via *topics* in the simulated environment, *services* are not represented in this diagram. The diagram was created with `rqt`.

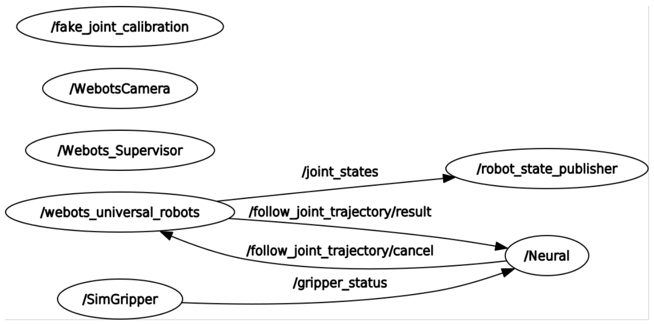

Fig. 6: Diagram of the *nodes* running during the testing phase. In the simulated environment most of the data is transmitted via ROS *services*. In the picture is shown the / follow_joint_trajectory *topic*, responsible for the robot movement, and the /gripper_status *topic*.

CNN From the four models tested, DenseNet and ResNext demanded more memory than the available GPU while MobileNet and MNASNet were capable of running on the GPU. To keep the fairness of the evaluation all timing tests were performed on the CPU.

4.2 Training

For training the CNN it was used a Huber loss error function [33] and an Adam optimizer [34] with weight decay regularization [35], the hyperparameters used for RL and CNN training are shown in the table 1.

Parameter name	Symbol	Value
Learning rate for CNN	α_{CNN}	1×10^{-3}
Weight decay for CNN	λ_w	8×10^{-4}
Learning rate for RL	α_{RL}	0.7
Discount factor	γ	0.90
Initial exploration factor	ε_0	0.90
Final exploration factor	ε_f	5×10^{-2}
Exploration factor decay	λ_ε	200

Table 1: Hyperparameters used in training.

To avoid color bias of the algorithm the color of the simulated table was changed for every episode.

Each training section was divided in four parts: collecting data, deciding the action to take based on the estimated Q-values, taking the action receiving a reward and training the CNN. Several sections of training were performed and the experience of the previous rounds were used to improve the training process.

The training cycle times are shown in table 2. Forward is the process following the direction from input to output of the CNN, backward is the process to evaluate the gradient from the difference in the output back to the input. In the backward process the weights of the network are updated with the learning rate α_{CNN}.

Base model name	Forward time (s)	Backward time (s)
DenseNet	0.408 ± 0.113	0.676 ± 0.193
ResNext	0.366 ± 0.097	0.760 ± 0.173
MobileNet	0.141 ± 0.036	0.217 ± 0.053
MNASNet	0.156 ± 0.044	0.257 ± 0.074

Table 2: Mean time and standard deviation of forward and backward time during training.

First training section In the first training round no previous experience is used and the algorithm learns from scratch. The main target is to get information of the training process about cycle time and acquire experience to be used in future training sections. The algorithm was training according to the most recent experience with batch size of 1.

In the training sections the accuracy was estimated based on 10 attempts every 10 epochs to verify how good the algorithm was performing at the time. The results are shown in figure 7. The training section took from 1:43 to 2:05 hours to complete.

In figure 7 is observed a training problem where the loss reaches zero and there is no gradient for learning. The algorithm cannot learn and the accuracy shows the q-values estimated are poor. There are several causes that can explain this case including the weights of the CNN are too small and the experience accumulated has most errors. The solutions for this are complex including fine-tuning hyperparameters and selecting best experiences for the algorithm as shown in [36]. Another solution is to use demonstration through shaping [37], where the reward function is used to generate training data based on demonstrations of the correct action to take.

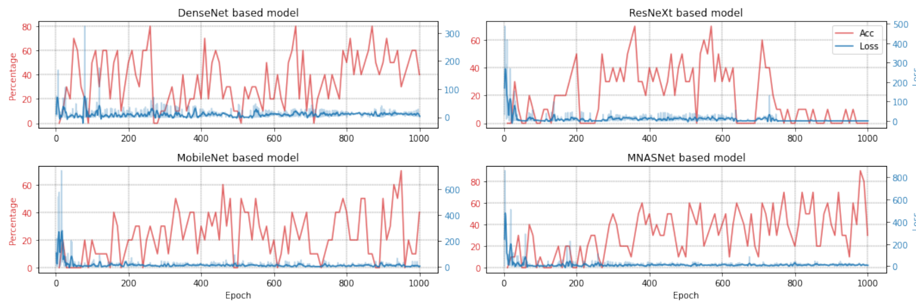

Fig. 7: The loss and accuracy of 1000 epochs training section, loss data were smoothed using a third order filter, raw data is shown in light colors.

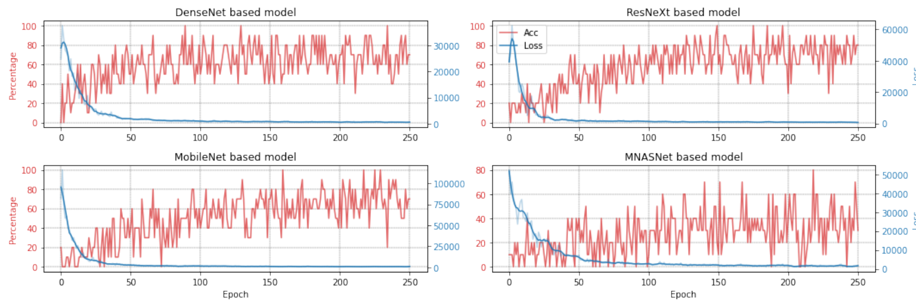

Fig. 8: The loss and accuracy during 250 epochs training section, data were smoothed using a third order filter, raw data is shown in light colors.

Second training section The second training section used the demonstration through shaping. It was possible because in the simulation environment the information of the position of the objects is available. The training process received experiences generated from the simulation, these experiences have the best action possible for each episode.

The batch size used on this training section was 10. The increase of batch size combined with the new experience replay caused a larger loss at the beginning of the training section as seen on the figure 8. The training section took from 3:43 to 4:18 hours to complete. The accuracy as estimated for every epoch based on 10 attempts.

5 Conclusion

This paper presented the use of RL to train an AI agent to control a Collaborative Robot to perform a pick-and-place task while estimating the grasping position without previous knowledge about the object. It was used an RGBD camera to generate the inputs for the system. An adaptive learning system was implemented to adapt to new situations such as new configurations of robot manipulators and unexpected changes in the environment. The results implemented on simulation validated the proposed approach. As future work, an implementation with a real manipulator will be addressed.

Acknowledgements

This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: UIDB/05757/2020 and by the Innovation Cluster Dracten (ICD), project Collaborative Connected Robots (Cobots) 2.0. The authors also thank the support from the Research Centre Biobased Economy from the Hanze University of Applied Sciences.

References

[1] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer International Publishing, Jan. 2016, pp. 1–2227, ISBN: 9783319325521. DOI: 10.1007/978-3-319-32552-1.

[2] "ISO/TS 15066 Robots and robotic devices - Collaborative robots," International Organization for Standardization, Geneva, CH, Standard, Feb. 2016.

[3] A. Saxena, J. Driemeyer, J. Kearns, and A. Y. Ng, "Robotic grasping of novel objects," in *Advances in Neural Information Processing Systems*, 2007, pp. 1209–1216, ISBN: 9780262195683. DOI: 10.7551/mitpress/7503.003.0156.

[4] C. Torras, *Computer Vision: Theory and Industrial Applications*. Springer Berlin Heidelberg, 1992, p. 455, ISBN: 3642486754.

[5] J. F. S. Gomes and F. R. Leta, *Applications of computer vision techniques in the agriculture and food industry: A review*, Dec. 2012. DOI: 10.1007/s00217-012-1844-2. [Online]. Available: http://link.springer.com/10.1007/s00217-012-1844-2.

[6] M. P. Arakeri and Lakshmana, "Computer Vision Based Fruit Grading System for Quality Evaluation of Tomato in Agriculture industry," in *Procedia Computer Science*, vol. 79, Elsevier B.V., Jan. 2016, pp. 426–433. DOI: 10.1016/j.procs.2016.03.055.

[7] M. U. M. Bhutta, S. Aslam, P. Yun, J. Jiao, and M. Liu, "Smart-Inspect: Micro Scale Localization and Classification of Smartphone Glass Defects for Industrial Automation," Oct. 2020. arXiv: 2010.00741. [Online]. Available: http://arxiv.org/abs/2010.00741.

[8] N. Shafii, S. H. Kasaei, and L. S. Lopes, "Learning to grasp familiar objects using object view recognition and template matching," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 2895–2900, ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759448.

[9] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 769–776, ISBN: 9781538626825. DOI: 10.1109/IROS.2017.8202237. arXiv: 1611.08036. [Online]. Available: http://arxiv.org/abs/1611.08036.

[10] D. Morrison, P. Corke, and J. Leitner, "Learning robust, real-time, reactive robotic grasping," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 183–201, Mar. 2020, ISSN: 0278-3649. DOI: 10.1177/0278364919859066. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364919859066.

[11] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on gpus," *Journal of Systems Architecture*, vol. 99, p. 101 635, 2019, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.101635. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762119302656.

[12] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way — by Sumit Saha — Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 06/20/2020).

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587, ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[14] R. Girshick, "Fast R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 1440–1448, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.169. arXiv: 1504.08083.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/results.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 386–397, 2020, ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2844175. arXiv: 1703.06870.

[17] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, *Detectron*, https://github.com/facebookresearch/detectron, 2018.

[18] D. Debkowski, *SuperBadCode/Depth-Mask-RCNN: Using Kinect2 Depth Sensors To Train Neural Network For Object Detection & Interaction.* [Online]. Available: https://github.com/SuperBadCode/Depth-Mask-RCNN (visited on 06/20/2020).

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, second edi. Cambridge, MA: The MIT Press, 2018, p. 552, ISBN: 978-0-262-03924-6.

[20] P. Zanuttigh, C. D. Mutto, L. Minto, G. Marin, F. Dominio, and G. M. Cortelazzo, *Time-of-flight and structured light depth cameras: Technology and applications.* Springer International Publishing, Jan. 2016, pp. 1–355, ISBN: 9783319309736. DOI: 10.1007/978-3-319-30973-6.

[21] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control," Nov. 2015. arXiv: 1511.03791. [Online]. Available: http://arxiv.org/abs/1511.03791.

[22] S. Joshi, S. Kumra, and F. Sahin, "Robotic Grasping using Deep Reinforcement Learning," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2020, pp. 1461–1466, ISBN: 978-1-7281-6904-0. DOI: 10.1109/CASE48305.2020.9216986. [Online]. Available: https://ieeexplore.ieee.org/document/9216986/.

[23] M. M. Rahman, S. M. H. Rashid, and M. M. Hossain, "Implementation of Q learning and deep Q network for controlling a self balancing robot model," *Robotics and Biomimetics*, vol. 5, no. 1, pp. 4–9, 2018, ISSN: 2197-3768. DOI: 10.1186/s40638-018-0091-9. arXiv: 1807.08272. [Online]. Available: https://doi.org/10.1186/s40638-018-0091-9.

[24] H. Hase, M. F. Azampour, M. Tirindelli, M. Paschali, W. Simson, E. Fatemizadeh, and N. Navab, "Ultrasound-Guided Robotic Navigation with Deep Reinforcement Learning," Mar. 2020. arXiv: 2003.13321. [Online]. Available: http://arxiv.org/abs/2003.13321.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," Tech. Rep. arXiv: 1608.06993v5. [Online]. Available: https://github.com/liuzhuang13/DenseNet.

[26] *Torchvision.models*, 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html (visited on 01/17/2021).

[27] Webots, *Http://www.cyberbotics.com*, C. Ltd., Ed., Commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com.

[28] A. Ayala, F. Cruz, D. Campos, R. Rubio, B. Fernandes, and R. Dazeley, "A Comparison of Humanoid Robot Simulators: A Quantitative Approach," *arXiv*, pp. 1–10, 2020. arXiv: 2008.04627.

[29] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," Tech. Rep. arXiv: 1709.10087v2. [Online]. Available: http://sites.google.com/view/deeprl-dexterous-manipulation.

[30] K. P. Hawkins, "Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms," Tech. Rep., Dec. 2013. [Online]. Available: https://smartech.gatech.edu/handle/1853/50782.

[31] *Universal Robots - Parameters for calculations of kinematics and dynamics*. [Online]. Available: https://www.universal-robots.com/articles/ur/parameters-for-calculations-of-kinematics-and-dynamics/ (visited on 12/31/2020).

[32] *Manual robotiq 2f-85 & 2f-140 for e-series universal robots*, Robotic, 145 pp., November 07, 2018.

[33] *SmoothL1Loss — PyTorch 1.7.0 documentation.* [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html (visited on 01/15/2021).

[34] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR, Dec. 2015. arXiv: 1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980v9.

[35] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," *arXiv*, Nov. 2017. arXiv: 1711.05101. [Online]. Available: http://arxiv.org/abs/1711.05101.

[36] T. De Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *Journal of Machine Learning Research*, vol. 19, pp. 1–56, 2018, ISSN: 15337928. DOI: 10.5555/3291125.3291134. [Online]. Available: http://jmlr.org/papers/v19/17-131.html..

[37] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2015-Janua, 2015, pp. 3352–3358, ISBN: 9781577357384.