Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control

Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, Peter Corke
ARC Centre of Excellence for Robotic Vision (ACRV)
Queensland University of Technology (QUT)
fangyi.zhang@hdr.qut.edu.au

Abstract

This paper introduces a machine learning based system for controlling a robotic manipulator with visual perception only. The capability to autonomously learn robot controllers solely from raw-pixel images and without any prior knowledge of configuration is shown for the first time. We build upon the success of recent deep reinforcement learning and develop a system for learning target reaching with a three-joint robot manipulator using external visual observation. A Deep Q Network (DQN) was demonstrated to perform target reaching after training in simulation. Transferring the network to real hardware and real observation in a naive approach failed, but experiments show that the network works when replacing camera images with synthetic images.

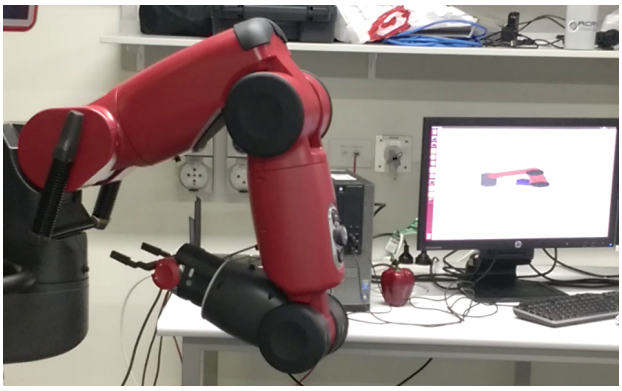


Figure 1: Baxter's arm being controlled by a trained deep Q Network (DQN). Synthetic images (on the right) are fed into the DQN to overcome some of the real-world issues encountered, i.e., the differences between training and testing settings.

1 Introduction

Robots are widely used to complete various manipulation tasks in industrial manufacturing factories where environments are relatively static and simple. However, these operations are still challenging for robots in highly dynamic and complex environments commonly encountered in everyday life. Nevertheless, humans are able to manipulate in such highly dynamic and complex environments. We seem to be able to learn manipulation skills by observing how others perform them (learning from observation), as well as, master new skills through trial and error (learning from exploration). Inspired by this, we want robots to learn and master manipulation skills in the same way.

To give robots the ability to learn from exploration, methods are required that are able to learn autonomously and which are flexible to a range of differing manipulation tasks. A promising candidate for autonomous learning in this regard is Deep Reinforcement Learning (DRL), which combines reinforcement learning and deep learning. One topical example of DRL is the Deep Q Network (DQN), which, after learning to play Atari 2600 games over 38 days, was able to match human performance when playing the game [Mnih *et al.*, 2013; Mnih *et al.*, 2015]. Despite their promise, applying DQNs to "perfect" and relatively simple computer game worlds is a far cry from deploying them in complex robotic manipulation tasks, especially when factors such as sensor noise and image offsets are considered.

This paper takes the first steps towards enabling DQNs to be used for learning robotic manipulation. We focus on learning these skills from visual observation of the manipulator, without any prior knowledge of configuration or joint state. Towards this end, as first steps, we assess the feasibility of using DQNs to perform a simple target reaching task, an important component of general manipulation tasks such as object picking. In particular, we make the following contributions:

- We present a DQN-based learning system for a target reaching task. The system consists of three components: a 2D robotic arm simulator for target

reaching, a DQN learner, and ROS-based interfaces to enable operation on a Baxter robot.

- We train agents in simulation and evaluate them in both simulation and real-world target reaching experiments. The experiments in simulation are conducted with varying levels of noise, image offsets, initial arm poses and link lengths, which are common concerns in robotic motion control and manipulation.
- We identify and discuss a number of issues and opportunities for future work towards enabling vision-based deep reinforcement learning in real-world robotic manipulation.

2 Related Work

2.1 Vision-based Robotic Manipulation

Vision-based robotic manipulation is the process by which robots use their manipulators (such as robotic arms) to rearrange environments [Mason, 2001], based on camera images. The early vision-based robotic manipulation was implemented using pose-based (position and orientation) closed-loop control, where vision was typically used to extract the pose of an object as an input for a manipulation controller at the beginning of a task [Kragic and Christensen, 2002].

Most current vision-based robotic manipulation methods are closed-loop based on visual perception. A vision-based manipulation system was implemented on a Johns Hopkins "Steady Hand Robot" for cooperative manipulation at millimeter to micrometer scales, using virtual fixtures [Bettini *et al.*, 2004]. With both monocular and binocular vision cues, various closed-loop visual strategies were applied to enable robots to manipulate both known and unknown objects [Kragic *et al.*, 2005].

Also, various learning methods have been applied to implement complex manipulation tasks in the real world. With continuous hidden Markov models (HMMs), a humanoid robot was able to learn dual-arm manipulation tasks from human demonstrations through vision [Asfour *et al.*, 2008]. However, most of these algorithms are for specific tasks and need much prior knowledge. They are not flexible for learning a range of different manipulation tasks.

2.2 Reinforcement Learning in Robotics

Reinforcement Learning (RL) [Sutton and Barto, 1998; Kormushev *et al.*, 2013] has been applied in robotics, as it promises a way to learn complex actions on complex robotic systems by just providing informing the robot whether its actions were successful (positive reward) or not (negative reward). [Peters *et al.*, 2003] reviewed some of the RL concepts in terms of applicability to control complex humanoid robots and highlighting some of the issues with greedy policy search and gradient based methods. How to generate the right reward is an active topic of research. Intrinsic motivation and curiosity have been shown to provide means to explore large state spaces, such as the ones found on complex humanoids, faster and more efficient [Frank *et al.*, 2014].

2.3 Deep Visuomotor Policies

To enable robots to learn manipulation skills with little prior knowledge, a convolutional neural network (CNN) based policy representation architecture (deep visuomotor policies) and its guided policy search method were introduced by Sergey et al. [Levine *et al.*, 2015a; Levine *et al.*, 2015b]. The deep visuomotor policies map joint angles and camera images directly to the joint torques. Robot configurations are the only necessary prior knowledge. The policy search method consists of two phases, i.e., optimal control phase and supervised learning phase. The training consists of three procedures, i.e., pose CNN training, trajectories pre-training, and end-to-end training.

The deep visuomotor policies did enable robots to learn manipulation skills with little prior knowledge through supervised learning, but pre-collected datasets were necessary. Human involvements in the datasets collection made this method less autonomous. Besides, the training method specifically designed to speed up the contact-rich manipulation learning made it less flexible for other manipulation tasks.

2.4 Deep Q Network

The DQN, a topical example of DRL, satisfies both the autonomy and flexibility requirements for learning from exploration. It successfully learnt to play 49 different Atari 2600 games, achieving a human-level of control [Mnih *et al.*, 2015]. The DQN used a deep convolutional neural network (CNN) [Krizhevsky *et al.*, 2012] to approximate a Q-value function. It maps raw pixel images directly to actions. No pre-input feature extraction is needed. The only one thing is to let the algorithm improve policies through playing games over and over again. It learnt playing 49 different games, using the same network architecture with no modification.

The DQN is defined by its inputs – raw pixels of game video frames and received rewards – and outputs, i.e., the number of available actions in a game [Mnih *et al.*, 2015]. This number of actions is the only prior knowledge, which means no robot configuration information is needed to the agent, when using the DQN for motion control. However, in the DQN training process, the Atari 2600 game engine worked as a reward function, but for robotic motion control, no such engine exists. To apply it in robotic motion control, a reward function is needed to assess trials. Besides, sensing noise and higher com-

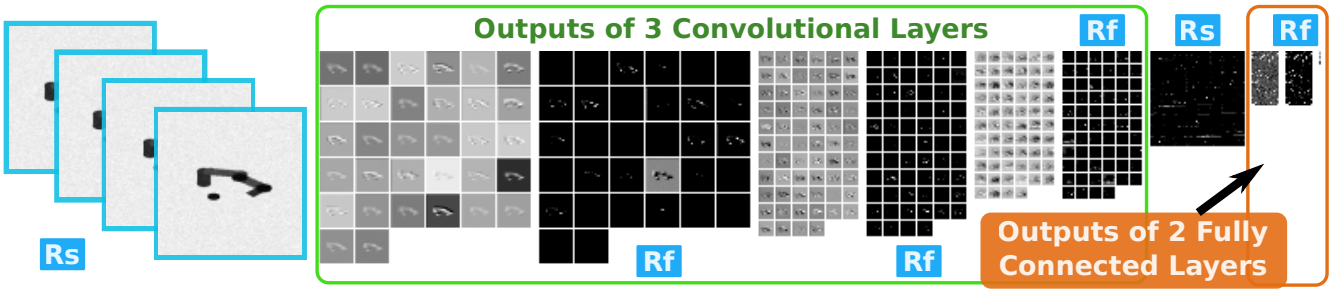


Figure 2: Schematic of the DQN layers for end-to-end learning and their respective outputs. Four input images are reshaped (Rs) and then fed into the DQN network as grey-scale images (converted from RGB). The DQN, consists of three convolutional layers with rectifier layers (Rf) after each, followed by a reshaping layer (Rs) and two fully connected layers (again with a rectifier layer in between). The normalized outputs of each layer are visualized. (Note: The outputs of the last four layers are shown as matrices instead of vectors.)

plexity and dynamics are inevitable issues for real-world applications.

3 Problem Definition and System Description

A common problem in robotic manipulation is to reach for the object to be interacted with. This target reaching task is defined as controlling a robot arm, such that its end-effector is reaching a specific target configuration. We are interested in the case in which a robot performs the target reaching with visual perception only. To learn such a task, we developed a system consisting of three parts:

- a 2D simulator for robotic target reaching, creating the visual inputs to the learner
- a deep reinforcement learning framework based on the DQN implementation by Google Deepmind [Mnih *et al.*, 2015], and
- a component of ROS-based interfaces to control a Baxter robot according to the DQN outputs.

3.1 DQN-based Learning System

The DQN adopted here has the same architecture with that for playing Atari games, which contains three convolutional layers and two fully connected layers [Mnih *et al.*, 2015]. Its implementation is based on the Google Deepmind DQN code[1] with minor modifications. Fig. 2 shows the architecture and examplary output of each layer. The inputs of the DQN include rewards and images. Its output is the index of the action to take. The DQN learns target reaching skills in the interactions with the target reaching simulator. An overview of the system framework for both the learning in simulation and testing on a real robot is shown in Fig. 3.

[1]https://sites.google.com/a/deepmind.com/dqn/

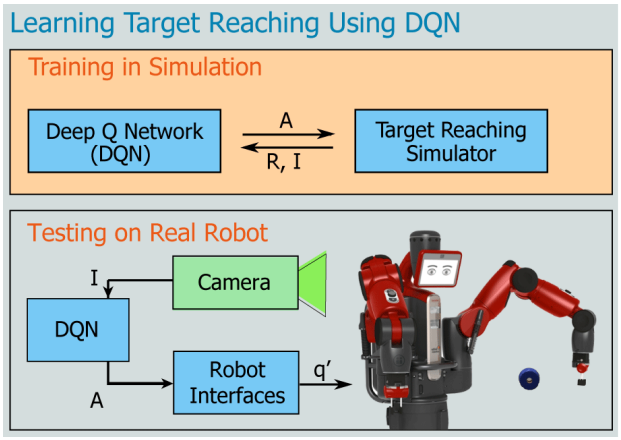


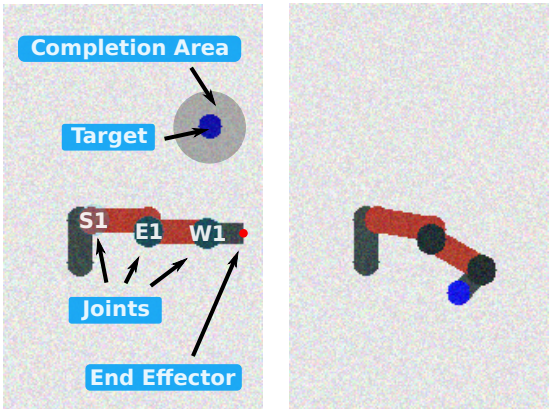
Figure 3: System overview

When training or testing in simulation, the target reaching simulator provides the reward value (R) and image (I). R is used for training the network. The action output (A) of the DQN is directly sent to the simulated robotic arm.

When testing on a Baxter robot using camera images, an external camera provides the input images (I). The action output (A) of the DQN is implemented on the robot controlled by ROS-based interfaces. The interfaces control the robot by sending updated robot's poses (q').

3.2 Target Reaching Simulator

We simulate the reaching task to control a three-joint robotic arm in 2D (Fig. 4). The simulator was implemented from scratch. In the implementation, no simulation platform was used. As shown in Fig. 4(a), the robotic arm consists of four links and three joints, whose configurations are consistent to the specifications of a Baxter arm, including joints constraints. The blue spot is the target to be reached. For a better visualization, the position of the end-effector is marked with a red spot.



(a) Schematic diagram (b) The robot simulator during a successful reach

Figure 4: The 2D target reaching simulator, providing visual inputs to the DQN learner. It was implemented from scratch, no simulation platform was used.

The simulator can be controlled by sending specific commands to the individual joints "S1", "E1" and "W1". The simulator screen resolution is 160×320.

The corresponding real scenario that the simulator simulates is: with appropriate constant joint angles of other joints on a Baxter arm, the arm moves in a vertical plane controlled by joints "S1", "E1" and "W1", and a controller (game player) observes the arm through an external camera placed directly aside it with a horizontal point of view. The three joints are in position control mode. The background is white.

In the system, the 2D simulator is used as a target reaching video game in connection with the DQN setup. It provides raw pixel inputs to the network and has nine options for action, i.e., three buttons for each joint: joint angle increasing, decreasing and hold. The joint angle increasing/decreasing step is constant at 0.02 rad. At the beginning of each round, joints "S1", "E1" and "W1" will be set to a certain initial pose, such as [0.0, 0.0, 0.0] rad; and the target will be randomly selected.

In the game playing, a reward value will be returned for each button press. The reward value is determined by a reward function introduced in Section 3.3. When satisfying some conditions, the game will terminate. The game terminal is determined by the reward function as well. For a player, the goal is to get an as high as possible accumulated reward before the game terminates. For clarity, we name an entire trial from the start of the game to its terminal as one round.

3.3 Reward Function

To keep consistent to the DQN setup, the reward function has two return values: one for the reward of each action; the other shows whether the target reaching game is terminal. Its algorithm is shown in Algorithm 1. The reward of each action is determined according to the distance change between the end-effector and the target. If the distance gets closer, the reward function returns 1; if gets further, returns -1; otherwise returns 0. If the sum of the latest three rewards is smaller than -1, the game terminates. This reward function was designed as a first step, more study is necessary to get an optimal reward function.

Algorithm 1: Reward Function

input : P_t: the target 2D coordinates;
P_e: the end-effector 2D coordinates.
output: R: the reward for current state;
T: whether the game is terminal.

$Dis = \texttt{ComputeDistance}(P_t,\ P_e)$;
$DisChange = Dis - PreviousDis$;
if $DisChange > 0$ **then**
  $R = -1$;
else if $DisChange < 0$ **then**
  $R = 1$;
else
  $R = 0$;
end
$R_{acc} = R_t + R_{t-1} + R_{t-2}$;
if $R_{acc} < -1$ **then**
  $T = True$;
else
  $T = False$;
end

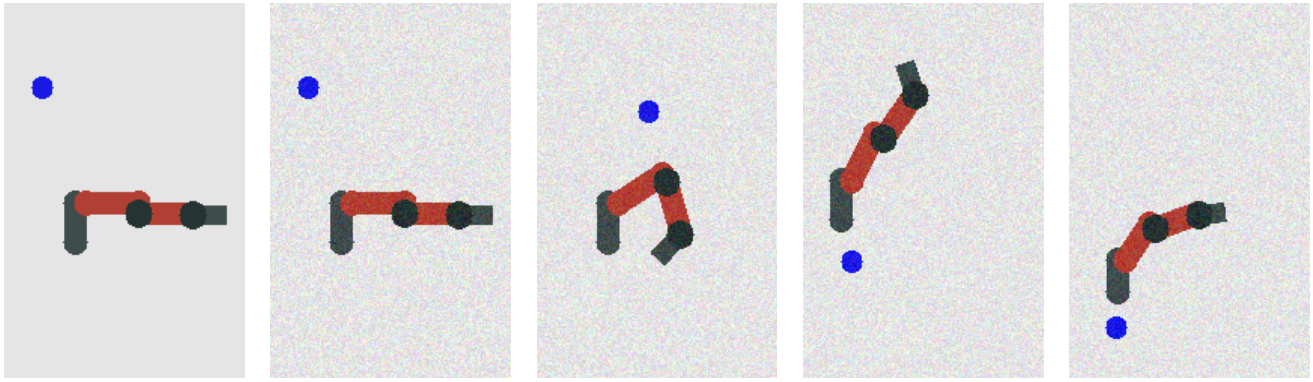
4 Experiments and Results

To evaluate the feasibility of the DQN-based system in learning performing target reaching, we did some experiments in both simulation and real-world scenarios. The experiments consist of three phases: training in simulation, testing in simulation, and testing in the real world.

4.1 Training in Simulation Scenarios

To evaluate the capability of the DQN to adapt to some noise commonly concerned in robotic manipulation, we trained several agents with different simulator settings. The different settings include sensing noise, image offsets, variations in initial arm pose and link length. The setting details for training the five agents are shown in Table 1. Their screenshots are shown in Fig. 5, respectively.

Agent A was trained in Setting A where the 2D robotic arm was initialized to the same pose ([0.0, 0.0, 0.0] rad) at the beginning of each round. There was no image noise in Setting A. To simulate camera sensing noise, random noise was added in Setting B, on the basis of



(a) Settings A: simulation images (b) Setting B: simulation images + noise (c) Setting C: Setting B + random initial pose (d) Setting D: Setting C + random image offset (e) Setting E: Setting D + random link length

Figure 5: Screenshots highlighting the different training scenarios for the agents.

Table 1: Agents and training settings

Agent	Simulator Settings
A	constant initial pose
B	Setting A + random image noise
C	Setting B + random initial pose
D	Setting C + random image offset
E	Setting D + random link length

Setting A. The random noise was with a uniform distribution with a scale between -0.1 and 0.1 (for float pixel values).

In Setting C, in addition to random image noise, the initial arm pose was randomly selected. In the training of Agent D, random image offsets were added on the basis of Setting C. The offset ranges in u and v directions were respectively [-23, 7] and [-40, 20] in pixel. Agent E was trained with dynamic arm link lengths. The link length variation ratio was [-4.2, 12.5]% with respect to the link length settings in the previous four settings. The image offsets and link lengths were randomly selected at the beginning of each round, and stayed unchanged in the entire round (not vary at each frame). All the parameters for noisy factors were empirically selected as a first step.

All the agents were trained using more than 4 million steps within 160 hours. Due to the difference in setting complexity, the time-cost for the simulator to update each game video frame varies in five different settings. Therefore, within 160 hours, the exact numbers of used training steps for the five agents are different. They are 6.475, 6.275, 5.225, 4.75 and 6.35 million, respectively.

The action Q-value converging curves are shown in Fig. 6. The Q-value curves are respect to training epochs. Each epoch contains 50,000 training steps. Fig. 6 shows the converging case before 80 epochs, i.e., 4 million training steps. The average maximum action Q-values are the average of the estimated maximum Q-values for all states in a validation set. The validation set was randomly selected at the beginning of each training.

Figure 6: Action Q-value converging curves. Each epoch contains 50,000 training steps. The average maximum action Q-values are the average of the estimated maximum Q-values for all states in a validation set. The validation set has 500 frames.

From Fig. 6, we can observe that all the five agents converge towards to a certain Q-value state, although their values are different. One thing we have to emphasize is this converging is just for average maximum action Q-values. A high value might but not necessarily indicate a high performance of an agent in performing target reaching, since this value cannot completely indicate the target reaching performance.

4.2 Testing in Simulation Scenarios

We tested the five agents in simulation scenarios with the 2D simulator. Each agent was tested in all those

five settings in Table 1. Each test took 200 rounds, i.e., terminated 200 times. More testing rounds can make the testing results closer to the ground truth, but need too much time.

In the testing, task success rates were evaluated. In the computation of success rates, it is regarded as a success when the end-effector gets into a completion area with a radius of 16 cm around a target, as shown in the grey circle in Fig. 4(a), which is twice size of the target circle. The radius of 16 cm is equivalent to 15 pixels in the simulator screen. However, for the DQN, this completion area is a ellipse (a=8 pixels, b=4 pixels), since the simulator screen will be resized from 160×320 to 84×84 before being input to the learning core.

Table 2 shows the success rates of different agents in different settings after 3 million training steps (60 epochs). The data in the diagonal (with a cell color of gray) shows the success rate of each agent tested with the same setting in which it trained, i.e., Agent A was tested in Setting A.

We also did some experiments for agents from different training steps. Table 3 shows the success rates of different agents after some certain training steps. The success rates of each agent were tested with the same simulator setting in which it was trained, i.e., the case in the diagonal of Table 2. In Table 3, "f" indicates the final number of steps used for training each agent in 160 hours, as mentioned in Section 4.1.

What we will discuss regarding the data in Table 2 and 3 is based on the assumption that some outliers of some conclusions appeared accidentally due to the limited number of testing rounds. Although 200 testing rounds are already able to extract data changing trends in success rates, they are insufficient to extract the ground truth. Some minor success rate distortions happen occasionally. To make the conclusions more convincing, more study is necessary.

From Table 2, we can find that Agent A and B can both adapt to Setting A and B, but can not adapt to the other three settings. This shows that these two agents are robust to random image noise, but not robust to dynamic arm initial pose and link length, and image offsets. The random image noise is not a key feature in these two agents.

In addition, other than the settings in which they are trained, Agent C, D and E can also achieve relatively high success rates in the settings with fewer noisy factors than their training settings. This indicates that agents trained with more noisy factors can adapt to settings with fewer noisy factors.

In Table 3, we can find that the success rate of each agent normally goes up after more training steps. This shows that, in the training process, all the five agents can learn to adapt to the noisy factors presented in their settings. However, some goes down after a certain training step, e.g., the success rate of Agent A goes down after 4 million training steps. Theoretically, with a appropriate reward function, the DQN should perform better and better, and the success rates should go up iteratively.

Table 2: Success rates (%) in different settings

Agent	Setting				
	A	**B**	**C**	**D**	**E**
A	51.0	53.0	14.0	8.5	8.5
B	50.5	49.5	11.0	8.0	10.0
C	32.0	34.5	36.0	22.5	14.0
D	13.5	16.5	22.0	19.5	15.0
E	13.0	16.5	20.0	16.5	19.0

Table 3: Success rates (%) after different training steps

Agent/Setting	Training Steps / million				
	1	**2**	**3**	**4**	**f**
A	36.0	43.0	51.0	36.0	36.5
B	58.0	55.5	49.5	51.5	13.5
C	30.5	33.0	36.0	48.0	13.5
D	16.5	17.5	19.5	26.5	14.0
E	13.0	18.5	19.0	23.0	27.0

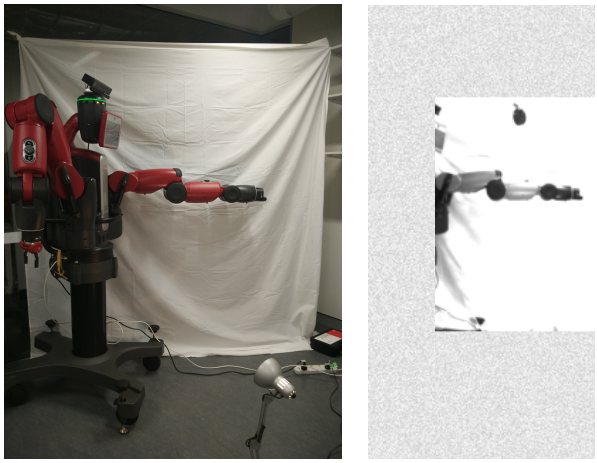
The going down case was quite possibly caused by the reward function, which has the possibility to guide the agent to a wrong direction. For the case in this paper, the evaluation is based on success rates, but the reward function is based on distance changes. The relation between success rates and distance changes is indirect. This indirect relation provides the incorrect guidance possibility. This should be considered carefully in future work.

Table 3 also shows that the success rate of the agent trained in a more complicated setting is normally smaller than that in a simpler setting, and needs more training time to get to a same level of success rate. For example, the success rate of Agent E is smaller than that of Agent D in each training episode, but is close to that of Agent D in a latter training episode.

In general, no matter whether the discussion assumption holds or not, the data in Table 2 and 3 at least shows that the DQN has the capability to adapt to these noisy factors, and is feasible to learn performing target reaching from exploration in simulation. However, more study is necessary to increase the success rates.

4.3 Real World Experiment Using Camera Images

To check the feasibility of trained agents in the real world, we did a target reaching experiment in real sce-



(a) Real world experiment using camera images (b) A sample input image

Figure 7: Testing scene and a sample input of the real world experiment using camera images. In the testing scene, a Baxter arm moved on a vertical plane with a white background. To guarantee that images input to the DQN have an as consistent as possible appearance to those in simulation scenarios, camera images were cropped and masked with a boundary. The boundary is from the background of a simulator screenshot.

narios using camera images, i.e., the second phase mentioned in Section 3.1. In this experiment, we used Agent B trained with 3 million steps, which has relatively high success rates for both Setting A and B in the testing in simulation.

The experiment settings were arranged to the case that the 2D simulator simulated, i.e., a Baxter arm moved on a vertical plane with a white background. A grey-scale camera was placed in front of the arm, observing the arm with a horizontal view of point (for the DQN, the grey-scale camera is the same with a color camera, since even the images from Atari games and the 2D target reaching simulator are RGB-color images, they are converted to grey-scale images prior to being input to the network). The background was a white sheet. The testing scene and a sample input to the DQN are shown in Fig. 7(a) and 7(b), respectively.

In the experiment, to make the agent work in the real world, we tried to match the arm position (in images) in real scenarios to that in simulation scenarios. The position adjustment was made through changing camera pose and image cropping parameters. However, no matter how we adjusted, it did not reach the target. The success rate is 0.

Other than the success rate, we also got a qualitative result: Agent B mapped specific input images to certain actions, but the mapping was ineffective for performing target reaching. There were some kind of mapping distortions between real and simulation scenarios. The distortions might be caused by the differences between real-scenario and simulation-scenario images.

4.4 Real World Experiment Using Synthetic Images

To verify the analysis regarding the reason why Agent B failed to perform target reaching, we did another real world experiment using synthetic images instead of camera images. In the experiment, the synthetic images were generated by the 2D simulator according to real-time joint angles ("S1", "E1" and "W1") on a Baxter robot. The real-time joint angles were provided by the ROS-based interfaces. In this case, there was no difference between real-scenario and simulation-scenario images. All other settings were the same with those in Section 4.3, as shown in Fig. 1.

In this experiment, we used the same agent that was used in Section 4.3, i.e., Agent B trained with 3 million steps. It achieved a consistent success rate with that in the simulation-scenario testing.

According to the results, we can conclude that the reason why Agent B failed in completing the target reaching task with camera images is the existence of input image differences. These differences might come from camera pose variations, color and shape distortions, or some other factors. More study is necessary to exactly figure out where the differences came from.

5 Conclusion and Discussion

The DQN-based system is feasible to learn performing target reaching from exploration in simulation, using only visual observation with no prior knowledge. However, the agent (Agent B) trained in simulation scenarios failed to perform target reaching in the real world experiment using camera images as inputs. Instead, in the real world experiment using synthetic images as inputs, the agent got a consistent success rate with that in simulation. These two different results show that the failure in the real world experiment with camera images was caused by the input image differences between real and simulation scenarios. To determine the causes of these more work is required.

In the future, we are looking at either decreasing the image differences or making agents robust to these differences. Decreasing the differences is a trade-off between making the simulator more consistent to real scenarios and preprocessing input images to make them more consistent to those in simulation scenarios. If choose to increase the fidelity of the simulator, it will most likely result in a slow-down of the simulation, increasing training time.

Regarding making agents robust to the differences, there are four possible methods: adding variations of the factors causing the image differences into simulation scenarios when training, adding a fine-tuning process in real scenarios after the training in simulation scenarios, training in real scenarios directly, and designing a new DRL architecture (still can be a DQN) which is robust to the image differences.

In addition to solving the problem of image differences, more study is necessary in the design of reward function. A good reward function is the key to get effective motion control or even manipulation skills and also speed up the learning process. The reward function used in this work is just a first step. It is far less than enough to be a good reward function. Other than the effectiveness and efficiency concerns, a good reward function needs also to be flexible to a range of general purpose motion control or even manipulation tasks.

Besides, the visual perception in this work is from an external monocular camera. An on-robot stereo camera or RGBD sensor can be a more effective and practical solution for applications in the 3D real world. The joint control mode in this work is position control, some other control modes like speed control and torque control are more common and appropriate for dynamic motion control and manipulation in real-world applications.

Acknowledgements

This research was conducted by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016). Computational resources and services used in this work were partially provided by the HPC and Research Support Group, Queensland University of Technology (QUT).

References

[Asfour *et al.*, 2008] Tamim Asfour, Pedram Azad, Florian Gyarfas, and Rüdiger Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. *International Journal of Humanoid Robotics*, 5(02):183–202, 2008.

[Bettini *et al.*, 2004] Alessandro Bettini, Panadda Marayong, Samuel Lang, Allison M Okamura, and Gregory D Hager. Vision-assisted control for manipulation using virtual fixtures. *IEEE Transactions on Robotics*, 20(6):953–966, 2004.

[Frank *et al.*, 2014] Mikhail Frank, Jürgen Leitner, Marijn Stollenga, Alexander Förster, and Jürgen Schmidhuber. Curiosity driven reinforcement learning for motion planning on humanoids. *Frontiers in Neurorobotics*, 7(25), 2014.

[Kormushev *et al.*, 2013] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[Kragic and Christensen, 2002] Danica Kragic and Henrik I Christensen. Survey on visual servoing for manipulation. Technical report, Computational Vision and Active Perception Laboratory, Royal Institute of Technology, Stockholm, Sweden, 2002.

[Kragic *et al.*, 2005] Danica Kragic, Mårten Björkman, Henrik I Christensen, and Jan-Olof Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and Autonomous Systems*, 52(1):85–100, 2005.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[Levine *et al.*, 2015a] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. Technical report, University of California, Berkeley, CA, USA, 2015.

[Levine *et al.*, 2015b] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 156–163, 2015.

[Mason, 2001] Matthew T Mason. *Mechanics of robotic manipulation*. MIT Press, 2001.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. Technical report, Google DeepMind, London, UK, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Peters *et al.*, 2003] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, pages 1–20, 2003.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.