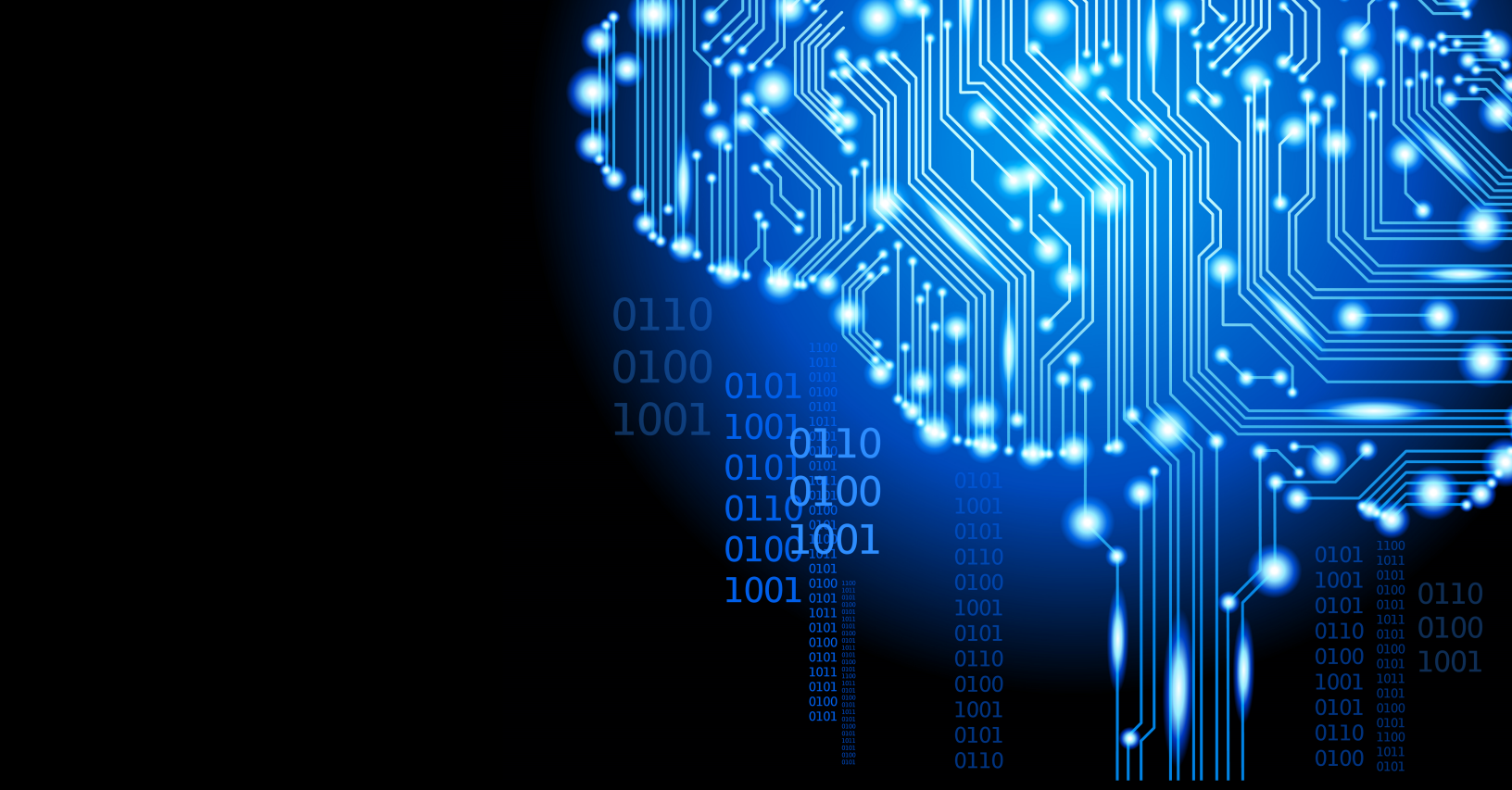


DDPG for Robotics Reaching and Grasping

Perception and Manipulation project

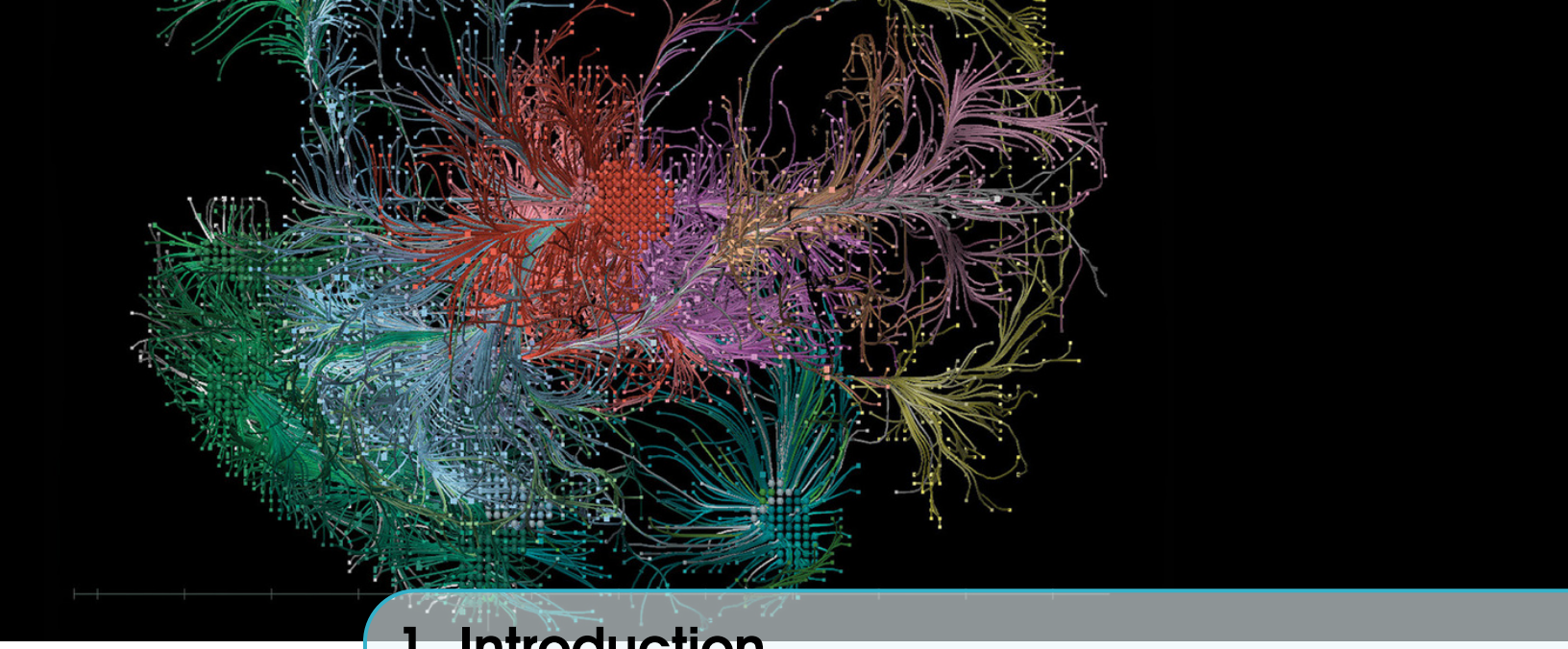
Eloïse DALIN

PERCEPTION AND MANIPULATION PROJECT

HTTPS://GITHUB.COM/DALINEL/DDPGBAXTER

First release, January 2017

Contents



1. Introduction

1.1 Motivation

Most of the robots are nowadays designed to serve a special purpose in a well controlled environment. Indeed, the usual optimal control techniques behave well in such environment but as soon as an unexpected event occurs, the robots lack robustness, adaptability and versatility. Reinforcement learning might be the key to solve the problem.

Reinforcement learning is used since the 1990s and techniques such as Q-learning[13] or SARSA are widespread. Nevertheless, applying reinforcement learning to robotics isn't a straight-forward problem. There are two types of reinforcement learning algorithms, those which can be applied to discrete actions and those which can be applied to continuous actions.

The robot control problem is continuous and discretization can lead to important errors. Besides, the significant number of possible states and actions can cause an exponential explosion also called "The curse of dimensionality"[6]. The use of deep learning has allowed tremendous advances in reinforcement learning. It has started with the DQN (Deep Q-Network) [9] in 2015 and techniques for the continuous case are appearing [2, 3, 4, 5, 8, 11]. Reinforcement learning can be applied in numerous fields of robotics. Some example are: robot soccer cooperation [12], helicopter flight [1] and humanoid robot control [10]. To simplify the learning task, reinforcement learning can also be coupled with imitation learning [7].

1.2 Presentation of the project

This project aims at using the novel reinforcement learning techniques for the reaching and grasping of a simple object by a manipulator arm. This project uses the robot Baxter simulated under gazebo and aims at grasping a cube has on the image below :

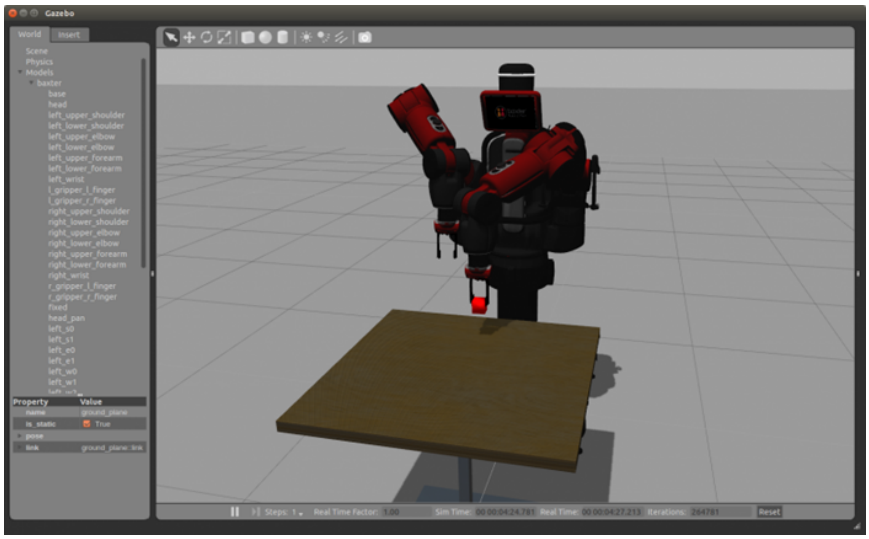


Figure 1.1: Baxter grasping environment

There are two main tasks to solve here :

- The reaching task
- The grasping task

In the part II of this report we are going to see the learning algorithm which has been chosen for this project and how it has been applied to the reaching and grasping problem. After this, we will see in the part III how those concepts have been implemented, the experiments which have been done and the associated results.

2. DDPG applied to reaching and grasping

The first step of this projected has been to select a reinforcement learning algorithm. The DQN algorithm [9] has revolutionized the reinforcement learning field but has explained in the introduction, the curse of dimensionality prevent the efficient use of this algorithm for continuous control tasks. The continuous version of the DQN algorithm is the DDPG algorithm (Deep Deterministic Policy Gradient). It has first been released in [8]. Some other improved versions have been released more recently, for example [11] proposes a distributed version of the DDPG algorithm. Nevertheless, for this project the original algorithm from [8] will be used. Indeed, in order to get familiar with the DDPG algorithm it is better to use the original algorithm. Moreover, the distributed version is using a lot of resources. It needs several computers.

2.1 DDPG algorithm

The Deep Deterministic Policy Gradient is an Actor Critic method. It means that it is composed of two parts. The actor part is choosing an action depending on the current state. The critic part is evaluating the action that the critic has chosen and give a positive or a negative feedback to the actor accordingly. The actor is the policy function and the critic is the value function.

Notice that the DDPG algorithm estimates a deterministic policy function $a = P(s)$ where a is the chosen action and s is the current state. In the case of a stochastic agent such as a chess player we would need a stochastic policy.

The critic is here the continuous Q function from the SARSA algorithm. The Q function evaluates the quality of every possible action according to the state.

The DDPG algorithm uses two Deep Neural Networks in order to estimate the policy and the value function. It was previously impossible to do so because training such functions was highly instable. Indeed, a minor change during training can drastically influence the weights of those functions. Nevertheless, thanks to the use of target networks firstly used with DQN [9] it is now

possible. During the training of each function a second Deep Neural Network called target network is created. Its weights are updated more smoothly and more gradually than the initial Deep Neural Network which prevents this instability.
The original paper presenting the DDPG algorithm [8] is using different kinds of Deep Neural Networks according to the state inputs. I recall that those state inputs are given by the environment to the DDPG algorithm after each executed actions. If the state inputs is a simple vector, such as for example joint positions plus a target to reach, two fully connected layers have been used. Nevertheless, if images are a part of the state input, the Deep Neural Network will be composed of a convolutional part followed by the two fully connected layers. For more details please refer to [8]. For intuitive explanations of the DDPG algorithm you can also refer to `http://pemami4911.github.io/blog/2016/08/21/ddpg-rl.html` and `https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html`

2.2 Reaching and Grasping task

Now that a reinforcement learning method has been chosen, it is important to know how to use it to solve the problem of the project. In order to solve this problem we need to identify the followings:

- which actions are returned by the policy function
- the state returned by the environment and transmitted the DDPG algorithm
- the reward function

2.2.1 Actions returned by the policy function

In [8] the actions returned by the policy function, for the reaching and grasping task, are the joint velocities of the manipulator arm. In this project, the same approach has been followed, the actor is returning the 7 joint velocities of the manipulator arm plus the gripper position (opened-closed) in percent. For the gripper, the position has been used instead of the speed because the Baxter simulation API doesn't allow the user to use the speed has an input.

2.2.2 State returned by the environment

For the reaching task it is obvious that the state returned by the environment has to contain at least the joints and gripper positions and the position of the target to reach. For the grasping task, the force detected by the fingers of gripper has also been added. In order to be able to generalize quickly to the task of moving the object toward a destination, the object position to reach has also been added.

2.2.3 Reward function

The reward function design is of tremendous importance. It is what will allow the DDPG to converge quickly or not. By trial and error I have noticed that a continuous reward function guiding the manipulator to reach it's goal continuously allows a faster convergence. Here are the rewards that have been used:

- $rewardReach = -distanceToObject$
- $rewardReachGrasp = -distanceToObject - \frac{0.0001}{distanceToObject} * (forceLeftFinger.x * forceRightFinger.x)$

$rewardReach = -distanceToObject$ is pretty straight forward, in order to maximize the reward, the

gripper will have to get closer to the object. For the grasping part we want to keep this influence, we want the gripper to get closer to the object. Nevertheless, we also want to grasp the object. An idea is to add the term $-(forceLeftFinger.x * forceRightFinger.x)$. Indeed Baxter has a parallel gripper, when grasping an object such as the cube, this formula will be positive. You can see it on the figure below:

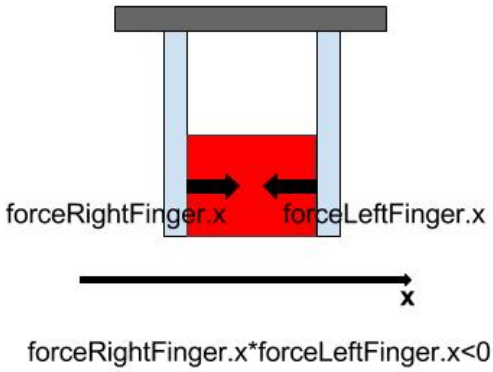


Figure 2.1: Forces in Baxter's parallel gripper while grasping

When Baxter will grasp an object we have $-(forceLeftFinger.x * forceRightFinger.x) > 0$ which will be added to the reward. Nevertheless, it happened during training that Baxter tried to grasp the border of the table for a brief moment. This unwanted action has added a positive reward. In order to prevent that, it is necessary to add the following coefficient to the grasping reward term :

$$\frac{0.0001}{distanceToObject}$$

With this coefficient the grasping reward term will only become important around the object. To summarize, for the grasping task, the reward is the sum of the two formulas below:

$$-distanceToObject$$

$$-\frac{0.0001}{distanceToObject} * (forceLeftFinger.x * forceRightFinger.x)$$

The first formula will make the gripper get closer to the object. The second formula will add a positive reward when the desired object has been grasped. The 0.0001 coefficient has been chosen in order to balance the two previous formulas and not have a grasping reward which would erase the effects of the distance reward.

3. Experimental setup

3.1 Computer setup

In order to realize this project the following have been installed:

- CUDA 8
- cuDNN 6
- Keras 1.2.2
- ROS kinetic
- Gazebo 7

The Baxter simulator for ROS Kinetic has been installed with this link `http://sdk.rethinkrobotics.com/wiki/Simulator_Installation#Baxter_Simulator_Installation`
You can check the final project files at `https://github.com/dalinel/ddpgBaxter`

3.2 Software architecture

For this project a separation between the learning algorithm and the environment will be made. The learning algorithm which is here DDPG will pick an action and send it to the environment. The environment which is here a ros node, will execute the action. When the action has finished to be executed, the environment will return to the learning algorithm a state vector and an associated reward. The goal is to maximize the total accumulated reward. You can have a look at the software architecture below:

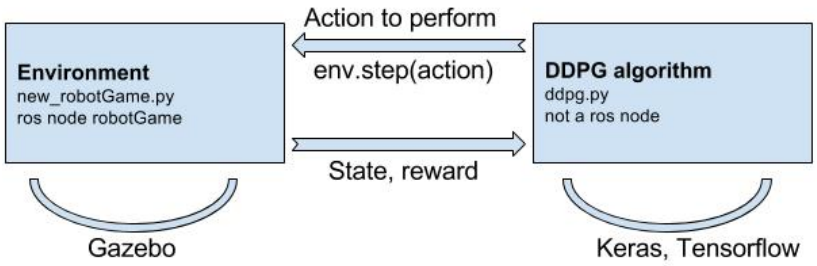


Figure 3.1: Software architecture

The actor and the critic are trained for a number of episodes defined by the user in ddpg.py. Here is the structure of an episode:

- ask *new_robotGame* to reset the simulated environment and execute 50 steps has followed:
  - recover an action from the actor (joints velocities and gripper position)
  - ask the *new_robotGame* to execute the action through the function step()
  - recover the state and reward observed from the step output
  - train the actor and critic accordingly

3.3 Experiments setup

For the both tasks, the environment reset each time in the following set up :

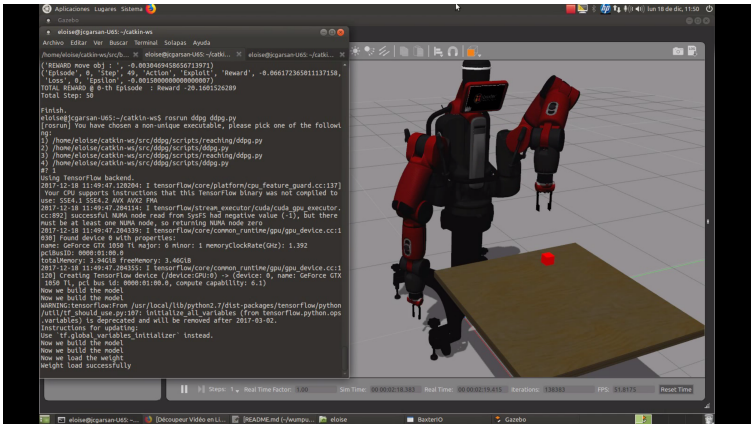


Figure 3.2: Baxter initial reaching position

The number of step per episode is 50. For the reaching task, I recall that the reward used is the following :

$$rewardReach = -distanceToObject$$

For the grasping task, the reward used is the following :

$$rewardReachGrasp = -distanceToObject - \frac{0.0001}{distanceToObject} * (forceLeftFinger.x * forceRightFinger.x)$$

4. Results

4.1 Reaching task

The reaching task has converged after a few hundred episodes and you can see the result below:

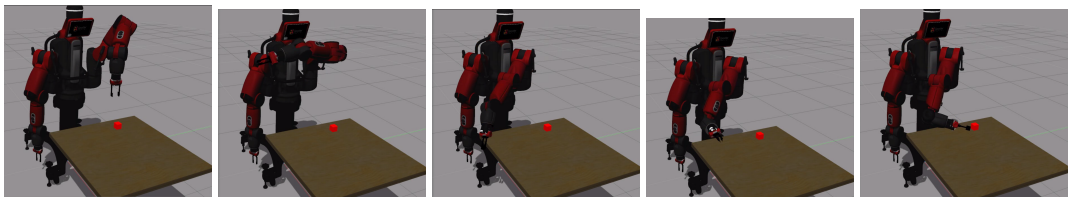


Figure 4.1: Baxter reaching the cube

At the end Baxter reaches the cube with a distance from the base of the gripper to the center of the cube of 6,7cm. You can notice here that the goal is achieved but the way to achieve this goal contains a random part. The reward here only specifies that the gripper needs to get closer to the cube. The arm joints positions are unconstrained by this reward. This means that the way to get closer to the cube is random. Indeed, when the actor and critic are trained some actions are picked randomly. This is a part of the exploration versus exploitation problem which is to decide whether it is better to exploit what has already been learned or decide to explore, picking a new random action, to see if the accumulated reward can be even more maximized.

4.2 Grasping task

The grasping task hasn't converged even for more than 1000 episodes. Several reasons might explain this. The first reason is that during the training sometimes Baxter picks up the object and then throws it away. The object, which is initially on a table can fall on the floor. At this moment the reward is becoming highly negative because the distance between the gripper and the cube increases a great deal. This negative reward will penalize the picking action. This problem is coming from the constant number of steps which is 50. A solution could be to define the end of the episode when

Baxter picks the object and to limit the number of steps to 50. Apart from this, a lot of different ideas could be put in place in order to make the Neural Networks converge:

- Tune the internal parameters of the DDPG (number of hidden units, exploration ratio, ...)
- Change the state input (add video feedback)
- Use two (actor,critic) pairs, one for reaching, one for grasping
- Change the initial position of the manipulator

Another problem might be the reward. For the reaching task, the reward is continuous, the algorithm can learn at each step. For the grasping reward it is different, it will only become relevant when Baxter grasp the desired object. In the mean time the algorithm do not learn anything about grasping. It is then normal that the training takes a lot longer. I think the next step for this project would have been to train a grasping (actor,critic) pair starting from an initial position close to the object and to define the end of the episode when the grasp has been done. It would then also be good to let the algorithm train for more episodes, some reinforcement learning tasks converge after millions of episodes.

5. Conclusion

To conclude, this project has been about using novel reinforcement techniques in order to solve the reaching and grasping problem. When it is possible to define a continuous reward function such as for the reaching task, the DDPG algorithm is very powerful and converges quickly. It would have been interesting to try to solve the global reaching problem from any starting position with this algorithm. Nevertheless, even if the gripper reaches its target, the other joints trajectories are not fully constrained by the reward. In order to solve this global reaching task it would have then been needed to integrate a collision checking part between the command sent by the actor and the execution made by the environment. If a command leading to a collision is sent this part could put the speed of the concerned joints to zero and send to the DDPG part a highly negative reward. For the graping task, because the reward is not continuous the convergence is harder. After what has been said earlier and in order to improve the learning speed, it would also be interesting to use imitation learning. Imitation learning would be here a great way to initialize the weights of the networks.

Bibliography

[1] Pieter Abbeel et al. “An Application of Reinforcement Learning to Aerobatic Helicopter Flight”. In: *Advances in Neural Information Processing Systems 19*. Edited by P. B. Schölkopf, J. C. Platt, and T. Hoffman. MIT Press, 2007, pages 1–8. URL: http://papers.nips.cc/paper/3151-an-application-of-reinforcement-learning-to-aerobatic-helicopter-flight.pdf (cited on page 5).

[2] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. “Gaussian Processes for Data-Efficient Learning in Robotics and Control”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (Feb. 2015), pages 408–423. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.218 (cited on page 5).

[3] Kenji Doya. “Reinforcement Learning in Continuous Time and Space”. In: *Neural Computation* 12.1 (2000), pages 219–245. DOI: 10.1162/089976600300015961. eprint: https://doi.org/10.1162/089976600300015961. URL: https://doi.org/10.1162/089976600300015961 (cited on page 5).

[4] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, July 2016, pages 1329–1338. URL: http://dl.acm.org/citation.cfm?id=3045390.3045531 (cited on page 5).

[5] Shixiang Gu* et al. “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates”. In: *IEEE International Conference on Robotics and Automation (ICRA 2017)*. *equal contribution. 2017 (cited on page 5).

[6] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pages 1238–1274. DOI: 10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721 (cited on page 5).

[7] Jens Kober and Jan Peters. “Learning motor primitives for robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. May 2009 (cited on page 5).

[8] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2015). arXiv: 1509.02971. URL: http://arxiv.org/abs/1509.02971 (cited on pages 5, 7, 8).

[9] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pages 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236 (cited on pages 5, 7).

[10] J. Peters, S. Vijayakumar, and S. Schaal. “Reinforcement learning for humanoid robotics”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids2003)*. clmc. Karlsruhe, Germany, Sept. 2003. URL: http://www-clmc.usc.edu/publications/p/peters-ICHR2003.pdf (cited on page 5).

[11] Ivaylo Popov et al. “Data-efficient Deep Reinforcement Learning for Dexterous Manipulation”. In: *CoRR* abs/1704.03073 (2017). arXiv: 1704.03073. URL: http://arxiv.org/abs/1704.03073 (cited on pages 5, 7).

[12] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. “Reinforcement Learning for RoboCup Soccer Keepaway”. In: *Adaptive Behavior* 13.3 (2005), pages 165–188. DOI: 10.1177/105971230501300301. eprint: https://doi.org/10.1177/105971230501300301. URL: https://doi.org/10.1177/105971230501300301 (cited on page 5).

[13] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pages 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698 (cited on page 5).