2019 4th Asia-Pacific Conference on Intelligent Robot Systems

Deep Reinforcement Learning for Mobile Robot Navigation

Martin Gromniak
Institute of Medical Technology
Technical University of Hamburg
Hamburg, Germany
e-mail: Martin.gromniak@tuhh.de

Jonas Stenzel
Department of Automation and Embedded Systems
Fraunhofer Institute for Material Flow and Logistics
Dortmund, Germany
e-mail: jonas.stenzel@iml.fraunhofer.de

***Abstract*—While navigation is arguable the most important aspect of mobile robotics, complex scenarios with dynamic environments or with teams of cooperative robots are still not satisfactory solved yet. Motivated by the recent successes in the reinforcement learning domain, the application of deep reinforcement learning to robot navigation was examined in this paper. In particular this required the development of a training procedure, a set of actions available to the robot, a suitable state representation and a reward function. The setup was evaluated using a simulated real-time environment. A reference setup, different goal-oriented exploration strategies and two different robot kinematics (holonomic, differential) were compared in the evaluation. In a challenging scenario with obstacles at changing locations in the environment the robot was able to reach the desired goal in 93% of the episodes.**

Keywords-reinforcement learning; end-to-end-learning; mobile robot navigation; robot learning

I. INTRODUCTION

Navigation is arguably the most crucial capability of any mobile robot. At the most fundamental level, it is the task of finding a path from one place to another without colliding with any obstacle in the environment. The exact problem formulation changes with its prerequisites: Does the robot know the environment in which it is moving? What kind of information can the robot perceive with its sensors and is available for decision making? Does the robot use legs, wheels or even wings to locomote? While navigation got a lot of attention from robotics researchers since the early days of robotics it is still not satisfactory solved yet. Remaining challenges are, for example, navigating in unknown terrain, navigation based on camera input, or navigating in dynamic environments.

The traditional approach to robot navigation usually employs a modular solution to the problem. The overall problem is decomposed into several subproblems which are independently solved. For example one module might perform global planning based on a static map and another module might perform local collision avoidance. While the modular approach is appealing from an engineering perspective, it also has drawbacks, e.g. it takes time to develop and integrate each individual module. Moreover, when facing complex tasks like multi-robot navigation it is hard to come up with an optimal problem decomposition. In the recent years, advantages in the field of deep reinforcement learning, enabled algorithms to solve several tasks for which is was hard to engineer direct solutions. Instead, these algorithms applied end-to-end learning. This refers to learning a black box model, like a neural network, which directly maps sensory inputs to desired outputs. The model is trained by performing gradient based optimization of a some score function.

Motivated by the recent successes in this domain, the application of deep reinforcement learning to robot navigation should be examined in this paper. The task scenario is inspired by an indoor logistics setting, where one robot is situated in an environment with multiple obstacles and should execute a transport order. The desired behavior of the robot is to travel to desired goal positions in the least amount of time, without colliding with the obstacles or with the walls of the environment. Additionally to the information about its own position and its goal, the positions of all obstacles and the global map of the environment are assumed to be known to the robot.

II. PROBLEM FORMULATION

In order to apply deep reinforcement learning to robot navigation, the problem setting must be specified and translated into a RL framework. In this work, the following scenario is considered: One robot, the decision making agent that will be described as ego robot from now on, is moving within a static map of the environment. In this environment, obstacles, whose current positions are also known, are present in changing locations. The ego robot can collide with the walls of the map or with the obstacles. A goal exists as a position within the map. The desired behavior of the ego robot is to reach the goal position without colliding with any obstacle.

The ego robot should be able to drive from arbitrary start positions to arbitrary goal positions and not collide with any obstacle regardless of their positions. In order to accomplish this, the agent must be exposed to all possible situations during training. Therefore, an episodic training scheme with randomized start settings is used. At the beginning of one episode, the ego robot's position, thgoal position and the positions of all obstacles are sampled from a uniform distribution over the free space of the map. One episode lasts for multiple time steps in which the agent receives states and rewards from the environment and moves around by executing velocity commands. The episode ends when the ego robot has reached its goal, it has crashed or when the number of time steps has exceeded a timeout value.

978-1-7281-2229-8/19/$31.00 ©2019 IEEE

The following sections will propose the design of a reinforcement learning framework that is supposed to achieve this behavior. The design includes an action space, a state representation, a reward function and a generic reinforcement learning algorithm, with its various hyperaparameters. While this chapter suggests multiple design possibilities, their experimental evaluation is postponed to the following chapter.

A. Action Space

The action space determines how an agent can affect the environment in which it is situated. For example, the action space of a chess playing agent could be the set of all possible moves, while in a robot control setting, it could be a combination of motor velocities.

Within this work, we used a discrete action space by assembling a set of motion primitives. In this case, the discrete actions must resemble useful velocity combinations, so that the ego robot is actually able to move around in a meaningful way. Here, the following three simple motion primitives have been used

$$a_1 = [k, 0,0], a_2 = [0,0, l], a_3 = [0,0, -l] \quad (1)$$

where k and l are scaling parameters. The action a_1 implements a step forward, while actions a_2 and a_3 implement a rotation to the right or left respectively. The values of the parameters have been empirically set to $k = 1$ m/s and $l = 4$ rad/s.

B. State Representation

The state representation of a reinforcement learning problem should in the optimal case include all information that are necessary to make decisions based on it. The DRL theory does not enforce any particular form of the state input. Both manually composed features [1], and raw images [2] have been used as inputs to neural networks. The overall goal when designing a DRL algorithm for a particular problem is to make the state information available to a neural network, in a form that the network can process it in the best way. We have used an image-like state representation where each channel of the image encodes a different information component.

When the map is discretized in $i\ x\ j$ cells, the state representation consist of the following four image planes, each with $i\ x\ j$ pixels (see Figure 1), which are concatenated to an image representation with four channels:

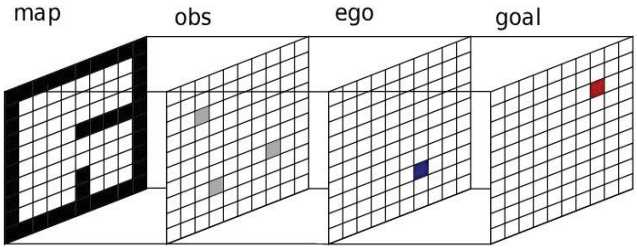

Figure 1. State Representation without Orientation; adapted from [3] .

1. In the map plane, a cell is set to 1, if it contains a wall segment or 0, otherwise.
2. In the obstacle plane, for every obstacles, the cell whose center is closest to the obstacles center position x_i, y_i is set to 1. All other cells are set to 0
3. In the ego position plane, the cell whose center is closest to the position of the ego robot x_{ego}, y_{ego} is set to 1. All other cells are set to 0
4. In the goal position plane, the cell whose center is closest to the position of the goal x_{goal}, y_{goal} is set to 1. All other cells are set to 0.

The described state representation is inspired from two recent publications. Silver et. al used a binary image representation for the state in the game of GO [4]. With one cell per stone position they used 49 feature planes to encode different information components, e.g. a recent history of stone positions, how many turns passed since the last stone was played at a specific position or how many opponent stones can be captured in a next move. Gupta et. al [3] used a similar encoding as the proposed one as an example for a multi-agent reinforcement learning problem. In a pursuerevader scenario a group of learning pursuer robots must catch multiple evader robots. All robots can execute discrete actions which let them move to adjacent cells. Instead of a goal channel, they used one channel for encoding the positions of the evader robots. The main difference to problem setting in this work is that the state spaces in GO and the pursuer-evader problem are discrete, while here, a discretized state space is constructed from an underlying continuous state space.

When the ego robot has a differential kinematic, the information about its orientation needs to be made available to the agent. For this purpose, the orientation can be represented as a point on a unit circle by $v = (\cos\emptyset,\ \sin\emptyset)$. A relatable orientation encoding was applied in [5] for the task of pose estimation from images. In their case, the orientation was the output of a CNN, in contrast to this work, where it is used as an input. The sine and cosine values are concatenated to the state representation of the four image planes using two additional planes that contain the sine and cosine of the orientation angle respectively in every entry. In doing so the orientational information is made available to a CNN at the very beginning, enabling the convolutional layers to beneficially use it.

C. Reward Function

In RL the reward function specifies what behavior of the agent is desirable. The desired behavior in robot navigation is to move from its start pose to the desired goal pose and at the same time not collide with any obstacles. The goal-oriented and the collision-avoiding behavior as we will call them from now on implement two objectives, that are at least to some point competing with each other. If the robot moves towards a goal, e.g. through narrow doorways it puts itself in more risk of crashing, than if it would simply stand still. In this sense the considered problem is a particularly challenging one. Most reinforcement learning problems in

the literature have a reward function which encourages a single objective, e.g. winning a game or travel as far as possible when learning a walking pattern.

Sparse rewards have been used for the considered navigation scenario. A reward is only returned at the end of every episode, while the reward is zero within an episode. The agent receives a reward of +1 when the robot reached its goal and −1 when the robot crashed. A reward of 0 is returned when a timeout occured after the length of the episode exceeded a certain length. Sparse reward are appealing because they directly specify the desired behavior, however they are also challenging for RL algorithms. Because the feedback from the environment is only given at the very end of an episode, but a sequence of actions have led to this result, it is hard to estimate the contribution of a single action. This is known as the credit assignment problem [6], which is one of the most challenging aspects of RL. Continuous reward functions, where the agent receives a reward at every time step do provide more feedback to the agent and can improve learning or even enable it in the first place. Therefore, continuous reward components, so called *shaping rewards*, in addition to the described sparse terminal rewards have been examined. Following the advice from [7], the continuous reward components were calculated as the difference in value of a potential function Φ applied to two successive states according to

$$r_{shape} = \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{2}$$

III. Neural Network Architecture

In this work the Proximal policy optimization (PPO) algorithm was chosen as the reference algorithm. This choice is motivated by the fact that PPO is a highly performant algorithm and is extendable to continuous action spaces [8]. In PPO there exist two functions. The policy is a mapping from states to action probabilities. The value function describes the mapping from states to expected returns. Both functions are approximated by (deep) Convolutional Neural Network (CNN) which only differ in the output layer. The used architecture design is oriented towards the neural networks which were used in AlphaGO Zero [9] for a similar state input. The image-like state input is first processed by multiple convolutional layers. For the convolutional layers, skip connections as introduced by [10] were adopted, which were shown to increase performance in deep CNNs. The skip connections were implemented with building blocks, called *residual blocks*. Their input is processed by two convolutional layers. The input is then added to the output of the second convolutional layer. In the architecture, a stack of multiple of such residual blocks is then followed by densely connected layers, which output depends on the policy or the value function. As proposed in [11] the value and policy network share most convolutional layers of the described architecture. This is reasonable as the learned features for both networks are likely to be similar.

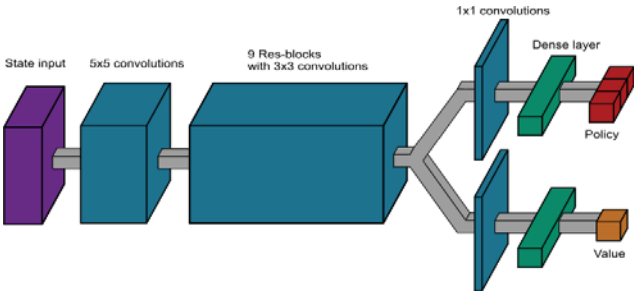

Figure 2. Neural Network Architecture

The layers of the used neural network are depicted in Figure 2. The image-like state input has the size $40 \times 30 \times 6$. The first convolutional layer zero-pads the input into a $44 \times 34 \times 6$ image, convolves it with k 5×5 filters and a stride of 1 and applies an ELU [12] nonlinearity. This is followed by a stack of 9 residual blocks, as described above. The convolutional layers of the residual blocks zero-pad the input to a size of $42 \times 32 \times k$, convolve it with k 3×3 filters and a stride of 1 and apply an ELU Nonlinearity. The number of res-blocks was chosen so that the last convolutional layers has an effective receptive field which spans over the full state input, i.e. it is able to learn global patterns. The number of feature maps k, was empirically chosen as 32. After the stack of residual blocks the information stream splits up in a policy stream and a value stream. In the value head first a 1×1 convolutional operation with stride of 1 is applied.

TABLE I. Parameters of the PPO Algorithm Used in This Work

Parameter	Value
Discount factor γ	0,99
Workers	16
n-step returns	4
Cliprange ϵ	0.2
Policy loss weight k_p	1
Value loss weight k_v	0.05
Entropy loss weight k_e	0.01
Batch size	64
Learning rate α	0.0001
Optimizer	RMSProp

It compresses k feature planes to an output with the size $40 \times 30 \times 1$ and therefore greatly reduces the number of parameters. It is followed by a densely connected layer with 256 ELU Units. The output of the policy stream is densely connected layer with a size equal to the number of actions. It uses a softmax activation in order to output a probability distribution over all discrete action. The value stream applies the same kind of 1×1 convolutions which are followed by a densly connected layer with 256 ELU Units. The value output is a fully connected layer with a single unit and no activation function.

IV. EXPERIMENTAL RESULTS

The theoretical considerations from the previous section are experimentally evaluated in this section. The scenario consists of a rectangular map with the size 4 3 meters. Six obstacles are present at changing locations, which do not move during an episode. For their geometry the same robot model as for the ego robot was chosen. It is a circular geometry with a diameter of 0.3 m. At the beginning of an episode the goal is placed randomly on the map within a distance of 3.5 m. The ego robot is considered at its goal when the distance of its center to the goal position is smaller than a threshold of 0.2 m. The maximum number of steps in an episode was set to 350, which results in 35 seconds of real-time.

The implemented image-like state representation which represents the described state has a special size of 40×30 pixels, so that its resolution is 0.1 m/pixel.

The performance of a training run is measured as the proportion of episodes which end with a successfully reached goal. It is also possible to compare the accumulated reward in an episode, as this is the measure which the DRL agent wants to maximize. However, the proportion of successful episodes offer a comparability across different reward schemes and actually represent the behavior that should be achieved from an engineering perspective. In the following, the best performing DRL for the defined problem setting is described. Afterwards, this reference setup is compared with alternative setups which were introduced in section 2.

A. *Reference Setting*

The parameters of the PPO algorithm in the reference setting are shown in Table I. A reward scheme with sparse terminal rewards and an additional continuous shaping reward is used (see section 2.3). The terminal rewards are +1, 0 or -1, depending on whether an episode ended with a reached goal, a timeout, or a collision respectively. The purpose of the shaping reward is to encourage exploration towards the goal. It is calculated from a potential function which has a maximum at the goal and is monotonically decreasing with distance from the goal. Therefore, the robot receives a positive reward if it moves towards the goal, while the reward is negative when the robot is moving away from it. The used potential function is $\Phi(s) = 1 - d(s)$, where $d(s)$ denotes the distance between the ego robot's position and its goal in state s. The total reward is the sum of terminal and shaping reward.

The performance of the reference setup is shown in Figure 3. On the left axes the graph shows the share of episode endings with each of the possible termination conditions. Either an episode ends with a reached goal, a timeout or a collision. These shares are plotted against the number of executed training steps. At the start of training, when the agent performs uninformed actions, it collides frequently with the obstacles and the walls of the map. Furthermore, by moving around randomly it is unlikely to encounter the goal which can be in far distance from the initial position of the ego robot. Shortly after the start of training it can be observed that the number of collisions is decreasing and the number of timeouts is increasing. The agent is able to learn from the large number of collision encounters and starts avoiding these situations. The beginning of a goal-oriented behavior can be observed around 0.5×10^5 training steps. From thereon the share of successful episodes rises to approx. 0.93 at step 2.5×10^5. The maximum number of shown training steps equals approx. 16×10^6 single experiences with a batch size of 64. The total runtime of this period are approx. 30 hours. On the right axis and as the dotted line, the total reward accumulated in an episode is included. The increasing performance over time is also reflected in the total reward. Additionally, it can be verified that the DRL algorithm tries to maximize this measure.

Qualitatively, the ego robot describes trajectories which not only reach the desired goal, but are also close to the shortest possible most of the times. However, there are also suboptimal behaviors that can be identified. They can be roughly classified into three groups. First, the ego robot sometimes collides with obstacles while navigating closely around those. This can be explained with a discretization error of the cell state representation. The ego robot can localize itself and other obstacle only to the extend of half a cell size, in this case 0.05 m. Second, it appears that the ego robot does not include information about obstacles on a global scale in an optimal way. While it does anticipate close obstacles, obstacles with a larger distance to the ego robot are only anticipated when the ego robot approaches them. This sometimes leads to a third problem where the ego robot gets stuck in an area and is not able to find a path out of it despite a path exists. This can potentially be a general problem of the purely reactive nature of model-free RL. In one time step one action is applied which is only dependent on the current state. No planning over multiple time steps and actions is performed. In contrast to this, classical navigation almost always includes a planning part (see Section 2.1). Possibly it can be the case that planning is a crucial necessity for certain environments like navigation where model-based RL methods are more appropriate.

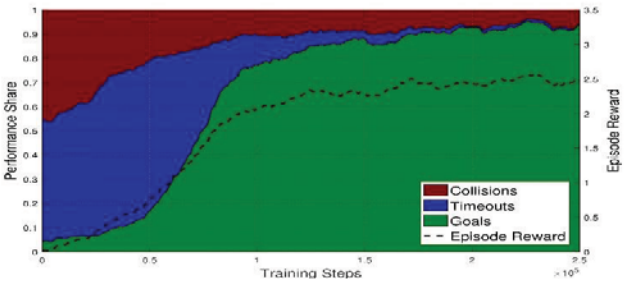

Figure 3. Reference Performance

B. *Goal-Oriented Exploration Strategies*

In this section two measures are compared, which tackle an issue that arises due to insufficient exploration in the navigation environment. If only a sparse reward scheme with terminal rewards is used, it can be observed that the agent fails to learn any goal-oriented behavior. This can be traced

back to the fact that the randomly initialized agent collides much more frequently than it reaches a goal. Starting from this situation, a better policy not to far to seek is one which simply does not apply any forward movements. Such a policy avoids all collisions, but does approach the goal.

From an optimization perspective this behavior can be seen as an undesired local minimum. In the reference setting, a shaping reward is used additionally to the terminal rewards in order to encourage a goal-oriented behavior. The continuous feedback when the ego robot is moving closer to the goal is a much richer feedback signal than the sparse reward at the end of a successful episode. It enables the learning of a goal-oriented behavior in the first place.

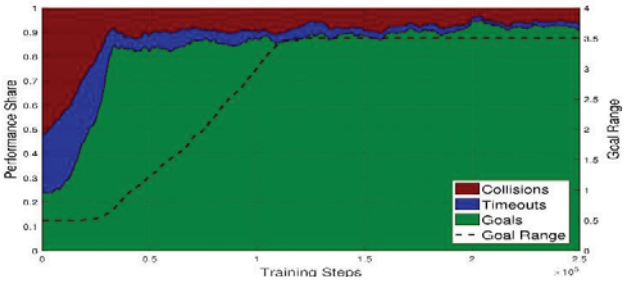

Figure 4. Curriculum Learning Approach with an Adaptive Goal Range

As an alternative to reward shaping, a form of curriculum learning was developed whose advantage is that it does not modify the reward function. Initially, goals for the ego robot are placed randomly within a distance of 0.5 m around the ego robot at the start of every episode. Placing goals close to the ego robot increases the chance of encountering a goal significantly and therefore makes the agent learn a goal-oriented behavior. Whenever the share of reached goals averaged over 100 episodes is greater than 0.8, the goal range is increased by 0.1 m until a maximum of 3.5 m is reached. Figure 4 shows the performance of the curriculum learning approach together with the adaptive goal range. It can be seen that at the start of training the share of reached goals is approx. 0.25 and increases soon after training. After the share of reached goals becomes greater than 0.8, the goal range starts to increase.

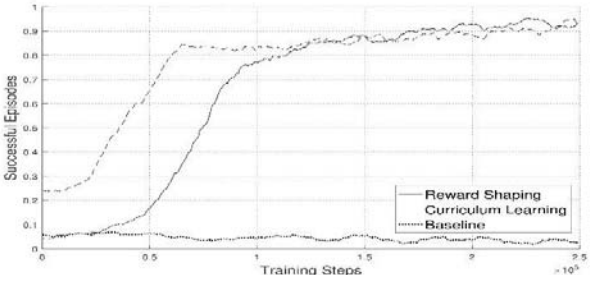

Figure 5. Comparison of Goal-Oriented Exploration Strategies

In the direct comparison, shown in Figure 5, it can be observed that both methods, reward shaping and curriculum learning, have a similar final performance after the maximum goal range of the curriculum learning approach is reached. As a conclusion, both methods, shaping reward or curriculum learning, can be rated equally suited. In contrast the baseline implementation without any of both methods, fails to learn completely. Therefore, either of both methods is necessary.

C. *Robot Kinematics*

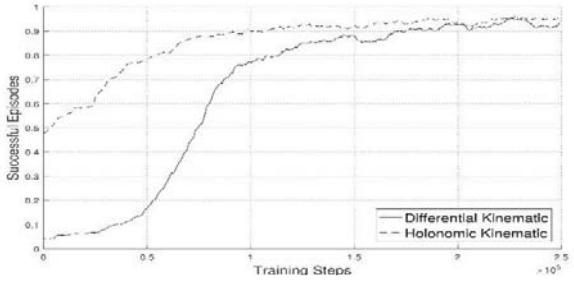

Figure 6. Comparison of Robot Kinematics

The comparison between the differential kinematic of the reference setup and a holonomic kinematic shows how much of the complexity of the navigation task can be ascribed to constraint movement of the ego robot and the necessary encoding of the current orientation of the robot for the differential kinematic case. Figure 6 shows a performance comparison of the two kinematics. As can be expected the performance of the holonomic kinematic is higher and learning is faster, because the overall problem is simpler. However, the difference in final performance is only approx. 2%.

V. Conclusion and Outlook

In this work, robot navigation was approached using end-to-end deep reinforcement learning. The experimental evaluation showed that DRL can be applied successfully to robot navigation. After one day of training in an exemplary navigation setting with multiple obstacles, the robot was able to reach its goal in approx. 93% of the episodes. By doing so the system showed that is was able to use all the information components encoded in the state representation and learn from those. Approx. 6% of the episodes resulted in collisions. Additionally, it could be observed that the neural network does not take into account information about obstacles which are far away from the ego robot.

During this work, parameter tuning has shown to be a laborious process. The implemented algorithm has a large number of major hyperparameters, which influence the training process and the resulting performance in a sensitive way. Due to the long training times of approx. one day, it was not possible to perform a reasonable parameter grid search.

Additionally, the neural network architecture can be seen as a hyperparameter with infinite possible values. Finding an architecture which is able to incorporate information about obstacles in the whole environment and in a parameter efficient way is a worthwhile next work step. The work can be also extended to a cooperative multi-robot navigation scenario, where multiple robots seek to move to individual

goal positions while obstructing each other as little as possible. In this case multiple decision making agents would move in one environment and collect experiences, instead of one agent per environment as in this work.

REFERENCES

[1] G. Tesauro. Temporal difference learning and td-gammon. Commun. ACM, 38(3):58–68, Mar. 1995. ISSN 0001-0782. doi:10.1145/203330.203343. URL http://doi.acm.org/10.1145/203330.203343.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. CoRR, 2013. URL http://arxiv.org/abs/1312.5602.

[3] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2017), 2017.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484– 489, jan 2016. doi:10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

[5] K. Hara, R. Vemulapalli, and R. Chellappa. Designing deep convolutional neural networks for continuous object orientation estimation. CoRR, 2017. URL http://arxiv.org/abs/ 1702.01499.

[6] R. S. Sutton. Temporal Credit Assignment in Reinforcement Learning. PhD thesis, 1984. AAI8410337.

[7] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML, volume 99, pages 278–287, 1999.

[8] J. Schulman, F.Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. CoRR, 2017. URL http://arxiv.org/abs/1707.06347.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. Nature, 550(7676):354–359, oct 2017. doi:10.1038/nature24270. URL https://doi.org/10. 1038/nature24270.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. CoRR, 2015. URL http://arxiv.org/abs/1512.03385.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning.

[12] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). CoRR, 2015. URL http://arxiv.org/abs/1511.07289.