Learning Primitive Skills for Mobile Robots

Yifeng Zhu[1], Devin Schwab[1], Manuela Veloso[1]

***Abstract*— Achieving effective task performance on real mobile robots is a great challenge when hand-coding algorithms, both due to the amount of effort involved and manually tuned parameters required for each skill. Learning algorithms instead have the potential to lighten up this challenge by using one single set of training parameters for learning different skills, but the question of the feasibility of such learning in real robots remains a research pursuit. We focus on a kind of mobile robot system - the robot soccer "small-size" domain, in which tactical and high-level team strategies build upon individual robot ball-based skills. In this paper, we present our work using a Deep Reinforcement Learning algorithm to learn three real robot primitive skills in continuous action space: `go-to-ball`, `turn-and-shoot` and `shoot-goalie`, for which there is a clear success metric to reach a destination or score a goal. We introduce the state and action representation, as well as the reward and network architecture. We describe our training and testing using a simulator of high physical and hardware fidelity. Then we test the policies trained from simulation on real robots. Our results show that the learned skills achieve an overall better success rate at the expense of taking 0.29 seconds slower on average for all three skills. In the end, we show that our policies trained in simulation have good performance on real robots by directly transferring the policy.**

I. Introduction

It has been a great challenge to achieve effective task performance on real mobile robots due to a great amount of effort involved. When a new skill is designed, the code needs to be rewritten and new parameters need to be tuned. Using learning algorithms shows the potential of lightening up the challenge above by using a single set of training parameters, but the feasibility of learning algorithms using real robots still remains a research pursuit. In this work, we focus on one of the mobile robot systems - robot soccer "small-size" domain. We use Deep Reinforcement Learning (DRL) algorithm to learn primitive skills instead of end-to-end skills for complex tasks so that primitive skills can be learned easily, and they can be easier to reuse, compose to complex skills.

Robot soccer "small-size" domain is a real-time, centralized multi-agent robot system [1]. Robots used in this system are omni-directional mobile robots which possess the ability to dribble and kick the ball. The system has overhead cameras to capture the vision information of the field. The vision information is streamed to a computer for processing. The processing of images is merely based on color features. Each robot has a "butterfly pattern" which is used for identifying a robot's team and id number, and the ball is in pure orange [2]. After applying image processing and state estimation methods like Kalman filter, the physical state features are obtained. Based on these state features, each team sends commands via radio to control robots.

In this robot soccer "small-size" domain, tactical and high-level team strategies are used within a successful framework of STP (Skills, Tactics, Plays) [3]. STP builds upon robot-ball-based primitive skills, which are referred to as "skills" in the rest of the paper.

In this paper, we learn three primitive skills using DDPG (Deep Deterministic Policy Gradient) [4]: `go-to-ball`, `turn-and-shoot` and `shoot-goalie`. We describe our training using a simulator with high physical, hardware fidelity. Previously, Schwab et al. presented the work of learning `go-to-ball` and `turn-and-shoot` in robot soccer "small-size" simulator [5]. Although this work shows the potential of learning skills in continuous action space for mobile robots, some improvements remain to be done. First of all, reward functions can be improved for better skill performance, and network structure was different for each skill. Secondly, while we were able to acquire a reasonable `turn-and-shoot` policy, the training diverged. Thirdly, there was no real robot evaluation.

To address these issues in previous work, we present the following contributions. Firstly, We present concise, improved reward functions for better performance, as well as using a single network structure for learning all skills. Specifically, `go-to-ball` and `turn-and-shoot` in this paper are similar to the ones in previous work, but with improved reward shaping functions and better learning performance as we will discuss in section III. Secondly, we train a new skill of shooting at a static goalie which enlightens the first step of using DRL in adversarial condition for "small-size" skill learning. At last and most importantly, we test our trained policies from simulation directly on real robots.

As our results show, the learned skills achieve an overall higher success rate at the expense of taking 0.29 seconds slower on average for all three skills. Our work also shows that the learned policies from simulation have equally good performance on **real robots** in **continuous action space** by directly transferring the policy.

The rest of the paper is organized as follows. Section II introduces related works on learning for robotics. In section III, we introduce the skills and the problem formulation. Section IV presents all the evaluation of learned skills both in simulation and using real robots. Section V gives a conclusion to this work and discusses future work.

[1]Yifeng Zhu, Devin Schwab and Manuela Veloso are with Carnegie Mellon University, Pittsburgh, PA, USA `yifengz2@andrew.cmu.edu` `dschwab@anderw.cmu.edu` `mmv@cs.cmu.edu`

II. Related Work

In recent years, Deep Reinforcement Learning (DRL) algorithms such as Deep Q-Networks (DQN), Deep Deterministic Policy Gradient DDPG have demonstrated great success for agents to learn policies in Markov Decision Process (MDP) environments [6] [7] [4]. DQN shows human level control in Atari games, and DDPG shows that it can learn low-level control tasks as well as pixel-based control in continuous action space. Other recent algorithms such as A3C, TRPO, PPO all enjoy success in learning policies of control tasks for agents [8] [9] [10]. DRL is also known for playing a master level of the Go game [11] [12].

Robot soccer is a term referring to a wide range of existing, general robot systems which play soccer. It may range from omnidirectional robots to humanoid walking robots. Though all these robot systems are categorized to robot soccer, it is important to notice that all these systems have many differences from one another in coordination strategy, real-time requirement, perception setup, etc. For example, CMDragons 2015 Champion uses a selectively reactive coordination which is different from other systems [13]. Therefore, different systems should be considered separately instead of identical in terms of conducting research.

Much learning research has been conducted based on the robot soccer related domain. Stone et al. has studied machine learning techniques in the subgame of robot soccer – Keepaway [14]. The goal of this task is to keep the ball as long as possible from the opponents. Both genetic programming methods and reinforcement learning methods have been used for solving this problem [15] [16] [17].

An evolutionary strategy approach has been applied to learn robot soccer humanoid walking. MacAlpine et al. used Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) approach to optimize parameters for humanoid walking [18] [19]. In this paper, we also compare our DRL evaluation with CMA-ES approach, which shows the overall advantage of DRL approach.

Reinforcement learning has also been used to learn skills on real robots for robot soccer "middle-size" [20]. However, since it requires deliberately discretizing action space, it might make the controllability of tasks impossible if discretizing is improperly done or discretizing is too coarse. Ours directly deal with continuous action space.

Recently, DRL is applied to skill learning in robot soccer "small-size" simulation. Hausknecht et al. propose the parameterized action space which combines the discrete and continuous action components [21] [22]. They learn a policy of navigating to a ball and shoot on an empty goal from demonstrations, and they make robot learn to manipulate ball from scratch. In their work, they show the success of applying DRL in the robot soccer domain, but an issue remains. Since they learn skills from end-to-end, it is impossible to decompose basic components like navigating to a ball from a learned policy and use it in other tasks such as navigating to ball and pass, which greatly limits the applicability of learned policies. There are also some previous works on sim-to-real testing such as using Progressive networks for manipulation tasks or learning robust policies for agile locomotion tasks [23] [24].

Previous work has explored the first step of using DDPG for skill learning in robot soccer "small-size" domain simulator [5]. The difference and contribution of this work compared to the previous one has been discussed in section I.

III. Problem Statement and Method

In this section, we describe the environmental setup for learning skills. First of all, we will give a quick review of two skills learned in previous work and introduce the new skill. Then we briefly introduce the DRL algorithm DDPG that we use, along with techniques for speeding up the convergence. At last, we describe the environmental setup for training three skills. We will briefly review the state and action space mentioned for `go-to-ball` and `turn-and-shoot`, along with simpler reward functions presented in this paper. Then we will introduce the state, action and reward function for new skill `shoot-goalie`.

A. Skill Description

In a robot soccer or human soccer game, ball possession and scoring are essential for a team to win. By hand coding, it takes a lot of human effort to create new skills for ball possession or scoring as different parameters need to be tuned for each skill. Therefore, we want to create skills using a learning algorithm. We focus on three skills that are related to ball possession and scoring: `go-to-ball`, `turn-and-shoot` and `shoot-goalie`. These threes skills have a clear success metric so that we can easily evaluate the learned policy. In the previous work, we have presented two skills `go-to-ball` and `turn-and-shoot`, which we will review briefly here. And we have an introduction of another skill, `shoot-goalie`.

We give a brief description of two mechanical structures to explain how the robot manages to dribble and kick which are essential for `turn-and-shoot` and `shoot-goalie` skills. Dribbler is a cylinder a robot has in the front. When the cylinder rotates fast enough, it can hold the ball and manipulate the ball while it is moving. It is basically a robot's ability to manipulate ball by meaning dribbling. Kicker is a solenoid which is reset by an elastic band and when the robot kicks, the solenoid hits the ball hard so that the ball is bounced away. The overview of the robot is shown in Figure 1.

`go-to-ball` skill: `go-to-ball` skill is a fundamental skill for robot navigating to the ball. By using this skill, a robot should learn to move to the ball. In the `go-to-ball` skill, a robot is spawned at an arbitrary position with arbitrary orientation, and the ball is randomly placed away from the robot in a certain range of distance. The goal of this skill is for robots to learn how to navigate towards the ball and touch the ball with its dribbler.

`turn-and-shoot` skill: This skill requires that the robot turns itself while dribbling the ball and shoot at the goal when facing it. For this skill, the robot is spawned on the field with arbitrary orientation, and the ball is spawned on the robot's dribbler. The robot needs to learn the proper control of dribbling strength and rotation velocity, as well as proper kick strength for shooting at goal.

`shoot-goalie` skill: For the `shoot-goalie` skill, we learn to turn and shoot on a goal while having a static goalie sitting in the middle of the goal. A stationary, attacker robot is spawned facing towards the goalie, and the ball is placed on the robot's dribbler. The attacker robot should learn to shoot avoiding the goalie, meaning that it needs to rotate and kick with proper strength so that the ball is not blocked by the static goalie while the robot shoots the ball within the range of goal. Moreover, if kick strength is not large enough, the ball will not roll far enough. Because the attacking robot is arbitrarily positioned on the field, it increases the adversary of this task.

In all skills, if the robot completes an episode within the maximum time steps limit, it is considered to be a success. If it exceeds maximal time steps limit before it succeeds, the episode is considered as a failure.

Fig. 1: Robot and ball.

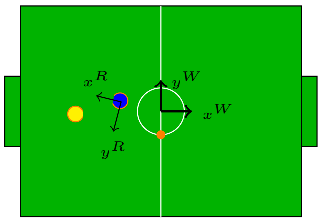


Fig. 2: Simulation rendering.

Figure 2 shows the example of simulation rendering. The green background is the soccer field. The blue circle is the robot we control, and the cutting surface is the robot's front face whose normal outward is the robot's orientation. This figure also shows the definition of the world frame of coordination and the robot's frame of coordination. (x^W, y^W) is the world frame, while (x^R, y^R) is the robot's frame. The yellow circle is an opponent's robot. The orange circle is the ball. There are two goals on each side. The simulation uses a physically realistic engine to model the robot's motion, collisions between objects, etc.

B. Algorithm

For learning three skills mentioned above, we use Deep Deterministic Policy Gradient (DDPG) algorithm [4] for learning policies. DDPG is an actor-critic, policy gradient algorithm which can deal with continuous action space. It has been shown to work in complex control tasks. It also uses a target network in the actor-critic framework, and it has a replay memory which stores samples during training and used for updating the policy. Since our states and actions are also continuous, DDPG is a suitable algorithm for training our policies.

When the exploration for action space is large, DDPG sometimes has a hard time to converge. We use two approaches to speed up the convergence. One is to leverage demonstrations for DDPG algorithm [25]. Before updating network, we fill in the replay memory with some demonstration transitions. These demonstrations are non-optimal, but can make robot receive some good rewards. By having some good demonstration examples in the replay memory, the policy will have a better idea of updating direction, and it can prevent cases such as an agent is exploring all the time but never has a chance of meeting a successful episode. The other approach which helps DDPG to converge faster is reward shaping. Usually, the reward is too sparse for the task that reward shaping is used for faster learning.

C. Environmental Setup

In this part, we present state representations, actions and reward functions for learning skills. For the state representation, we use state vectors instead of image inputs due to the computation efficiency that a robot soccer system requires.

Notice that in this part, all variables are transformed into **robot's frame** (relative coordinates). There are two benefits to represent in robot's frame. First of all, the robot's low-level velocity command is in robot's frame, which makes the state input more relevant to relative coordinates. Secondly, the representation is not strictly related to the actual size of the environment. When specifying the angle difference relative to the robot, we refer to the angle difference between the direction of an object relative to the robot and robot's orientation.

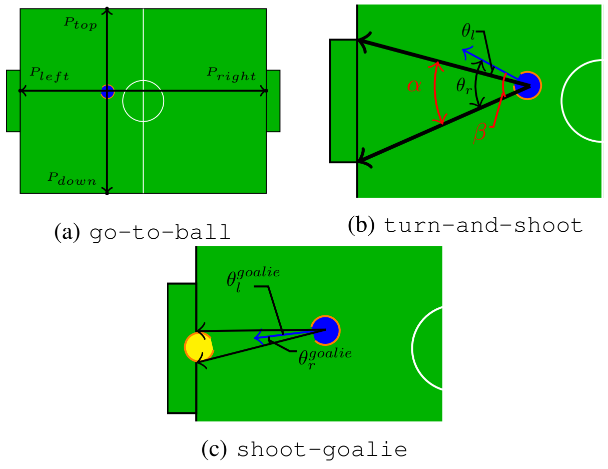


Fig. 3: Parts of the state representations for three skills.

1) **`go-to-ball` skill**: The state features for this skill follow from [5] as follows:

$$s = (P^B, V^R, \omega^R, d_{r-b}, P^R_{top}, P^R_{bottom}, P^R_{left}, P^R_{right})$$

where P^B is the x-y location of the robot, V^R is the robot's linear velocity, ω^R is the robot's angular velocity, d_{r-b} is the distance from the robot to the ball and $P^R_{top}, P^R_{bottom}, P^R_{left}, P^R_{right}$ are relative coordinates of the closest points on the top, bottom, left, right edges of the field to the robot. The robot's action space includes its linear velocity, and its angular velocity, which are noted as (v^R_x, v^R_y, ω^R), which is the same in previous work [5].

Because the field is relatively large compared to the ball, it has a very low probability for the robot to merely explore in a sparse reward setting to exactly touch the ball using its dribbler. In previous work [5], reward functions are considered separately for distance and angle differences. In this work, we simplify reward function by treating x, y and θ equally as follow:

$$r_{total} = r_{contact} + r_{distance}$$

where,

$$r_{contact} = \begin{cases} 100 & \text{ball on the dribbler} \\ 0 & \text{ball not on the dribbler} \end{cases}$$

$$r_{distance} = \frac{5}{\sqrt{2\pi}} \exp\left(-\frac{d_{r-b}^2 + \theta_{r-b}^2}{2}\right) - 2$$

Here θ_{r-b} is the direction of the ball relative to robot. $r_{contact}$ is the sparse reward regarding success. $r_{distance}$ drives the robot going near the ball and adjusting its orientation towards the ball in order to obtain more reward. Here we choose meter as the unit of d_{r-b} so that we can have the distance and angle value in the similar value range. This reward function prevents the redundancy of designing a new reward function for angle variable only, which also reduces the confusion to the training caused by having two independent reward functions.

We also apply the technique of DDPG from demonstration. During the demonstration process, the ball is always spawned right in front of the robot and the given action is simply driving straight forward. By doing this, the replay memory is filled with simple and good demonstrations of navigating towards a ball.

2) **`turn-and-shoot` skill**: As presented in previous work [5], state features include:

$$s = (P^B, V^B, \omega^R, d_{r-g}, f(\theta_l), f(\theta_r))$$

where P^B is the ball position, V^B is the ball velocity, ω^R is the robot's angular velocity, d_{r-g} is the distance between the robot and the center point of the goal line, θ_l and θ_r are angle differences between the robot and left, right goalposts respectively. Here $f(\theta)$ is defined as a vector function which is $(\sin(\theta), \cos(\theta))$. θ_l and θ_r are also shown in the Figure 3b. And the associated action space is $(\omega^R, dribble, kick)$ for rotating, holding the ball, shooting at correct timing as described in [5].

Notice that we use trigonometric functions of angles for states instead of angles directly. The reason is that two numerical values might represent the same angle because of its periodicity, and this would cause confusions. Therefore, by using trigonometric functions, we can eliminate the confusion introduced by the periodicity of angles.

In the previous work [5], reward shaping for this skill resulted in diverged training. Here we present a reward function which results in converged traninng. The reward function is:

$$r = \begin{cases} e^{\frac{(\alpha-\beta-0.05)*|V^B|}{5}} - 1 & \text{ball is kicked} \\ -0.5 & \text{ball not on dribbler before kick} \\ 0 & \text{otherwise} \end{cases}$$

Here α is the angle between two goal posts relative to robot's position, and β is the larger one of angle differences between the kicking direction and two goal posts (we calculate inferior angles). These two angles are also shown in Figure 3b. So the reward shaping would penalize the robot if it kicks hard in a wrong direction, or reward the robot if it kicks as hard as possible towards the goal and score.

In this skill, we also use DDPG from demonstration. During the demonstration, the robot is spawned facing towards the center of the goal, and is given actions of shooting towards without rotating and dribbling.

3) **`shoot-goalie` skill**: `shoot-goalie` skill is a skill that a robot shoots against a stationary goalie. This is an adversarial scenario because it needs to avoid shooting at the goalie. The state features we choose are:

$$\begin{aligned} s = &(P^B, V^B, \omega^R, d_{r-g}, f(\theta_l), f(\theta_r), \\ &f(\theta_c^{goalie}), f(\theta_l^{goalie}), f(\theta_r^{goalie})) \end{aligned}$$

where all the variables share the same meaning as ones in the `turn-and-shoot` skill except for θ_c^{goalie}, θ_l^{goalie} and θ_r^{goalie}. θ_c^{goalie} is angle of the direction of the goalie relative to the robot, while θ_l^{goalie} and θ_r^{goalie} denote the angle differences from the robot to the left, right point of the goalie. Figure 3c shows θ_l^{goalie} and θ_r^{goalie}.

In this skill, we assume that the robot's dribbler is on, and it just needs to learn to rotate to proper shooting angle and kick. In this skill, the robot dribbles all the time so that we focus more on learning the timing to shoot. Thus the action space is consisted of $(\omega^R, kick)$. The robot needs to kick when it is at the orientation where the ball can avoid the goalie.

Here we can just use a sparse reward function so that we don't shift complexity into reward engineering:

$$r = \begin{cases} 1 & \text{the episode succeeds} \\ -1 & \text{the episode fails} \\ 0 & \text{otherwise} \end{cases}$$

The episode is considered to be successful when the ball is in the goal and when it crossing the goal line, it is $100mm$ away from both sides of the goalie. This setting prevents the robot from learning to shoot at the robot and bounce back into the goal, which is a risky and unwanted action.

For `shoot-goalie` skill, we do not fill in any demonstration. We show in the next session that `shoot-goalie` skill learned well even in the sparse reward setting.

As mentioned in section II, we compare the performance of DRL method with CMA-ES. CMA-ES approach belongs to the idea of evolutionary strategy. For our purpose, it samples several candidates of parameters for parametrized policies from a Gaussian distribution in each iteration and updates the distribution from the performance of candidate policies. The details of CMA-ES can be found in [19]. For the comparison of CMA-ES, we use the same state, action and reward settings.

IV. Experimental Results

In this section, we show the experimental results of skill learning. We first present the result of training curves, and then evaluate policies by comparing with hand-coded skills from perspectives of success rate and time performance. We also evaluate skills on real robots, showing that the policies achieve good performance by directly transferring the policy from simulation.

A. *Training*

Here we present the network structure for skill learning, and all the hyperparameters for training. The actor network has 2 hidden layers with 300, 400 units respectively. After each layer, we apply 'relu' activation functions and layer normalization [26]. The critic network also has 2 hidden layers with 300, 400 units respectively, and action from the actor is included before the second hidden layer of the critic. During the training, we also choose Ornstein-Urlenbeck process as our exploration noise. All the policies are trained in a simulator with high physical and hardware fidelity. Table I shows the hyperparameters we use for DDPG to train policies. For the comparison approach CMA-ES, we set the initial value of variance to 1, 0.1, 0.1 respectively for `go-to-ball`, `turn-and-shoot` and `shoot-goalie` skills. We also choose 16 candidate policies from each sampling for every generation and we train the policy for 1000 generations.

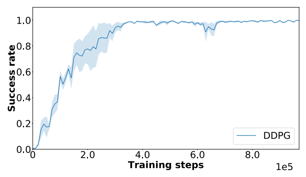
TABLE I: Hyperparameters for learning skills

Hyperparameters	**Value**
critic learning rate	1×10^{-3}
actor learning rate	1×10^{-4}
`go-to-ball` replay mem size	1,000,000
`go-to-ball` noise parameters	$\theta = 0.15, \mu = 0, \sigma = 0.3$
`turn-and-shoot` replay mem size	450,000
`turn-and-shoot` noise parameters	$\theta = 0.15, \mu = 0, \sigma = 0.1$
`shoot-goalie` replay mem size	100,000
`shoot-goalie` noise parameters	$\theta = 0.15, \mu = 0, \sigma = 0.1$

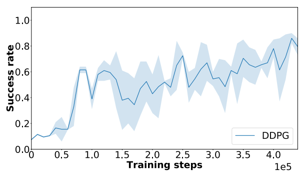
As we mention in section III, we utilize DDPG from demonstration for `go-to-ball` and `turn-and-shoot` skills. We have 10,000 demonstration transitions for both skills. For generating the training curve, we save the network weights with a certain frequency and test saved policies after training is done. During the testing, we run 100 episodes and calculate the percentage of successful episodes on each saved policy. We train three times for each skill, and the training curves are shown in the Figure 4.

B. *Skill Evaluation*

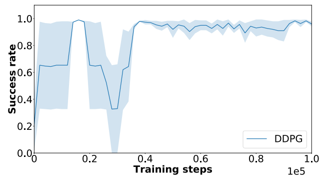
In this part, we evaluate the policies trained for three skills. We empirically demonstrate that our trained policy has reasonable performance by comparing the learned skills with existing hand-coded skills. We support our argument by showing the statistics collected for trained skills and hand-coded skills.



(a) `go-to-ball` skill



(b) `turn-and-shoot` skill



(c) `shoot-goalie` skill

Fig. 4: Training curves for all three skills.

TABLE II: Comparison of hand-coded policy and trained policy

	Hand-coded Policy	**Trained Policy**	**CMA-ES Policy**
`go-to-ball` skill	**1.0**	0.999	0.02
`turn-and-shoot` skill	0.71	**0.878**	0.186
`turn-and-shoot` skill (fine-tuned)	**0.948**	0.878	0.186
`shoot-goalie` skill	0.978	0.99	**1.0**

TABLE III: Comparison of time taken between hand-coded policy and trained policy (unit: s)

	Hand-coded Policy	**Trained Policy**
`go-to-ball` skill	1.46 ± 0.47	2.11 ± 1.19
`turn-and-shoot` skill	2.11 ± 0.99	2.35 ± 2.01
`shoot-goalie` skill	1.59 ± 0.36	1.56 ± 0.76

In table II, we show the success rate of all three skills using three approaches - hand-coded, reinforcement learning and CMA-ES. We evaluate final policies over 500 runs. We can see that CMA-ES only learns the `shoot-goalie` skill perfectly, but hardly learns reasonable policies for the other two skills. Also for `turn-and-shoot`skill, the hand coded policy without fine-tuning performs significantly worse than the trained policy. Only after taking a lot of effort fine-tuning some threshold in the code does the performance go up, which is also presented in the table. This table shows the overall higher success rate of DRL approach compared to original hand-coded policy (without fine-tuned deliberately) and CMA-ES.

We further compare the time duration it takes between hand-coded skills and learned skills. In table III, we show the mean and standard deviation of time taken by skills to complete tasks. For this statistics, we evaluate final policies over 500 runs. This table shows that learned skills take slightly slower time (0.29 seconds on average of all skills) when they have the overall better success rate than other approaches.

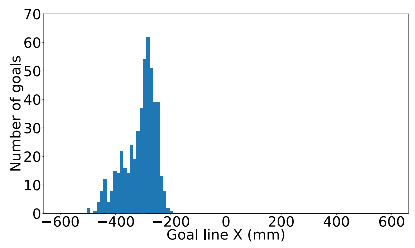


Fig. 5: Distribution of where ball crosses the goal line.

For shoot-goalie skill, we also evaluate the distribution of the locations when the ball crosses the goal line, as Figure 5. Over 500 runs of the policy, the ball goes into the goal for 498 times. Notice that the x axis is showing the goal line, which is from $-600mm$ to $600mm$. The distribution figure shows that the converged policy always chooses one side to shoot. Ideally, we would like the policy to learn the idea of "open angle" in an adversarial scenario, which means that the robot learns to shoot on both sides with equal probability. The observed result is due to the deterministic property of DDPG and a simple sparse reward function without any additional information. As a result, it is more likely to shoot at one side of the goal.

C. Real Robot Testing

We test go-to-ball, turn-and-shoot and shoot-goalie skills on real robots as well. We put the robot in different initial states to test final policies. We test 7 runs for go-to-ball, 36 runs for turn-and-shoot and 12 runs for shoot-goalie. We test these three skills with different numbers of test runs because we only need to test the performance of robot starting in different initial state, so we only choose a reasonable number of initial states for each run. The result is that go-to-ball succeeds 6 out of 7 runs, turn-and-shoot succeeds 33 out of 36 runs, and shoot-goalie succeeds 11 out of 12 runs. So the success rate for each skill on the robot is 0.857, 0.916 and 0.916 respectively. The overall performance is slightly lower than the result from simulation, but the performance remains good. In the following part, we show the snapshots of executing skills in Figure 6, 7 and 8. We also have a supplementary video[1] of showing how policies perform on real robots before, during and after training to illustrate how the policies improve over time. Notice that all the policies are trained in simulation.

V. Conclusion

In this paper, we learn primitive skills for robot soccer using DDPG algorithm. These skills are fundamental for ball possession, scoring and playing in an adversarial case. Based on these learned skills, we can compose these fundamental skills for a more complex skill.

In this paper, we have the following contributions: 1) We demonstrate that the learned skills have better performance compared to the previous work [5] by using improved reward

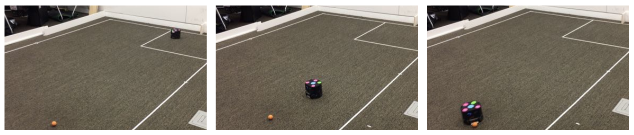


Fig. 6: `go-to-ball` skill on real robot

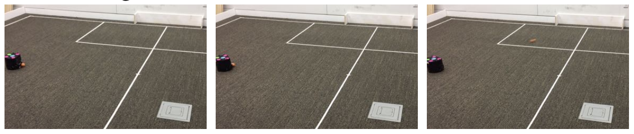


Fig. 7: `turn-and-shoot` skill on real robot

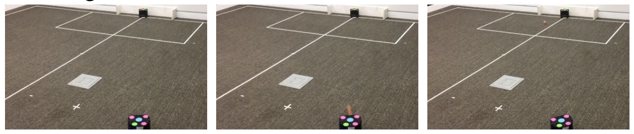


Fig. 8: `shoot-goalie` skill on real robot

functions. We also use single network structure for learning all three skills. 2) We learn a new skill for shooting at a goalie with a sparse reward, which is a first step to learn skills in adversarial cases. 3) Most importantly, we test the trained skills in **continuous action space** and **real world** for robot soccer "small-size" domain by using DRL algorithm. This provides an alternative to achieve effective task performance for other real mobile robot domains using reinforcement learning. For example, following go-to-ball skill, we can learn navigating to a position for other mobile robots. We can also learn skills of interacting with other objects as we do in turn-and-shoot and shoot-goalie.

Our work can be extended to learn pixel-based skills, but due to real-time requirement for computation, we did not use images as input in this work. Future work includes using image input for skill learning and learning more high-level policies in a multiple-robot system such as robot soccer "small-size" [27].

We also observe the drawback of using deterministic policy for adversarial cases. If the opponent learns how to block the shot, the learned policy would eventually fail. Therefore, another future work is to try stochastic DRL algorithms such as Soft Q Learning for training stochastic policies in adversarial environments [28].

The evaluation on real robot shows that the learned policies from simulation work well in the real world. We believe that if we further fine-tune the policies on real robots, the skill would have much better performance. Using domain randomization will be a potential way to increase the robustness of policies from the simulation to real world.

ACKNOWLEDGMENT

This research is partially sponsored by DARPA under agreements FA87501620042 and FA87501720152. The views and conclusions contained in this document are those of the authors only.

[1]A video of better quality can be viewed in the link attached: https://youtu.be/mE8xfZdE3gE

References

[1] The RoboCup Federation, "RoboCup," 2017. [Online]. Available: http://www.robocup.org/

[2] J. Bruce and M. Veloso, "Fast and accurate vision-based pattern detection and identification," in *International Conference on Robotics and Automation*, 2003.

[3] B. Browning, J. Bruce, M. Bowling, and M. Veloso, "STP: Skills, tactics and plays for multi-robot control in adversarial environments," *Journal of Systems and Control Engineering*, vol. 219, pp. 33–52, 2005.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *Internal Conference on Learning Representations*, 2016.

[5] D. Schwab, Y. Zhu, and M. Veloso, "Learning skills for robocup small size leagues," in *RoboCup International Symposium*, 2018.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR, abs/1502.05477*, 2015.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] 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*, 2016.

[12] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, 2017.

[13] J. P. Mendoza, J. Biswas, P. Cooksey, R. Wang, S. D. Klee, D. Zhu, and M. M. Veloso, "Selectively reactive coordination for a team of robot soccer champions." in *AAAI*, 2016.

[14] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu, *Keepaway Soccer: From Machine Learning Testbed to Benchmark*, 2006.

[15] W. H. Hsu and S. M. Gustafson, "Genetic programming and multi-agent layered learning by reinforcements." in *GECCO*, 2002.

[16] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler, "Co-evolving soccer softbot team coordination with genetic programming," in *RoboCup-97: Robot soccer world cup I*, 1998.

[17] D. Andre and A. Teller, "Evolving team darwin united," in *Robot Soccer World Cup*, 1998.

[18] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone, "Design and optimization of an omnidirectional humanoid walk: A winning approach at the robocup 2011 3d simulation competition." in *AAAI*, 2012.

[19] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[20] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, 2009.

[21] M. Hausknecht, Y. Chen, and P. Stone, "Deep imitation learning for parameterized action spaces," in *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.

[22] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proceedings of the International Conference on Learning Representations (ICLR)*, May 2016.

[23] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[24] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[25] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *CoRR*, 2017.

[26] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[27] D. Schwab, Y. Zhu, and M. Veloso, "Zero shot transfer learning for robot soccer," *AAMAS*, 2018. Extended Abstract.

[28] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *arXiv preprint arXiv:1702.08165*, 2017.