Learning Skills for Small Size League RoboCup

Devin Schwab(✉), Yifeng Zhu, and Manuela Veloso

Carnegie Mellon University, Pittsburgh, PA 15217, USA
digidevin@gmail.com

Abstract. In this work, we show how modern deep reinforcement learning (RL) techniques can be incorporated into an existing Skills, Tactics, and Plays (STP) architecture. STP divides the robot behavior into a hand-coded hierarchy of plays, which coordinate multiple robots, tactics, which encode high level behavior of individual robots, and skills, which encode low-level control of pieces of a tactic. The CMDragons successfully used an STP architecture to win the 2015 RoboCup competition. The skills in their code were a combination of classical robotics algorithms and human designed policies. In this work, we use modern deep RL, specifically the Deep Deterministic Policy Gradient (DDPG) algorithm, to learn skills. We compare learned skills to existing skills in the CMDragons' architecture using a physically realistic simulator. We then show how RL can be leveraged to learn simple skills that can be combined by humans into high level tactics that allow an agent to navigate to a ball, aim and shoot on a goal.

Keywords: Reinforcement learning · Robot software architecture · Autonomous robots

1 Introduction

RoboCup soccer is an international competition where teams of researchers compete to create the best team of autonomous soccer playing robots [22]. Multiple leagues from simulation, to full humanoid leagues compete each year. In this work we focus on the Small Size League (SSL) RoboCup, which is a challenging, fast paced, multi-agent league.

The Skills, Tactics, and Plays (STP) [3] software architecture has been used by the 2015 winning champions, CMDragons. STP is a hierarchical architecture consisting of three levels. Skills are coded policies that represent low-level tasks, used repeatedly in a game of soccer. These are tasks such as: dribbling the ball, navigating to a point, etc. Tactics combine skills into behaviors for a single

This research is partially sponsored by DARPA under agreements FA87501620042 and FA87501720152 and NSF grant number IIS1637927. The views and conclusions contained in this document are those of the authors only.

D. Holz et al. (Eds.): RoboCup 2018, LNAI 11374, pp. 83–95, 2019.
https://doi.org/10.1007/978-3-030-27544-0_7

robot. Typical tactics are roles such as: attacker, goalie, defender, etc. They are typically coded as state machines, where specific skills are called in each state. Plays are how multiple tactics are coordinated. Each robot is assigned a tactic based on cost functions, and then the robots execute these tactics independently.

In prior years, all levels of the STP architecture have been written by hand using classical robotics algorithms. Low-level policies such as navigation can use algorithms such as an RRT [9], while the tactic state machines have been written by intuition and improvement through extensive testing. Writing new skills is a large investment of human time and coding. Ideally, these low-level skills could be learned automatically, and then reused by the human-coded tactics in order to save time and man-power.

Recently, deep reinforcement learning (Deep RL) techniques have made major breakthroughs in performance. Deep Q-Networks (DQN) [13,14] have been used to learn policies from pixels in Atari that exceed human performance. More recently, Deep RL has been used to beat human performance in the game of Go [17,18]. Outside of games, Deep RL has been used to learn complex continuous control tasks such as locomotion [7]. It is therefore an attractive idea to use Deep RL for SSL RoboCup. However, it is unclear how best to learn policies in such a complex, multi-agent, adversarial game.

In the rest of this paper, we explore how Deep RL can be used to automatically learn skills in an STP architecture. While it would also be useful to learn tactics and plays, skills are small enough problems to be effectively learned using Deep RL in a short amount of time. Tactics are much more complicated, and plays would require multi-agent RL algorithms, therefore, in this work we focus on learning skills. A learning algorithm can be implemented once and then applied to learn many different skills. Whereas, each hand-coded skill will require it's own implementation, testing and tweaking. By learning skills, human coders can spend time working on the more complicated tactics and plays. We show that after skills are learned using Deep RL they can be effectively combined by humans into useful tactics.

2 Small Size League (SSL) RoboCup

Figure 1 shows an example of the CMDragons SSL robot. The robots are approximately 18 cm in diameter and move omni-directionally. The robots can kick at various speeds both flat along the ground and through the air. The robots also have spinning dribbler bars, which apply backspin to the ball.

Figure 1 shows an overview of the field setup. Overhead cameras send images to a central vision server. The vision server uses colored patterns to determine ball position and robot field positions, orientations, ID and team. This information is sent to each team at 60 Hz. Teams typically use a single computer that sends radio commands to all robots on the field. Therefore, centralized planning and coordination can be used.

The full game of SSL has many different parts: kickoff, free kicks, penalty kicks, main game play, etc. In this work we focus on skills that can be useful

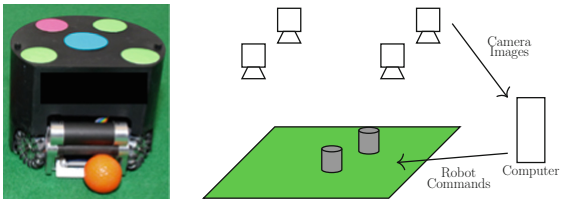


Fig. 1. (Left) A typical Small Size League (SSL) robot and ball. (Right) SSL field setup. Overhead cameras capture dot patterns on the tops of robots to determine positions, orientations, team and robot ID. Ball position is also determined from color. A central computer publishes this position information to each team. Teams then send radio commands to their robots on the field. (Color figure online)

in many different parts of the game: namely capturing the ball on the dribbler, and aiming and shooting at the goal.

3 Related Work

Deep RL algorithms, such as Deep Q-Networks (DQN), Asynchronous Advantage Actor (A3C) and others, have been shown to work in complex Markov Decision Processes (MDPs) such as Atari games and the game of Go [12–14,17]. There have also been some initial successes in applying Deep RL to continuous control tasks using policy gradient methods such as Deep Deterministic Policy Gradients (DDPG), Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) [10,15,16]. Unlike traditional RL, Deep RL algorithms can often work from low-level features such as joint angles or even pixels. This can alleviate the need for domain specific feature engineering.

Our work is not the first attempt to apply learning to RoboCup domains. Many techniques have been developed and applied for the game of Keepaway [20]. In Keep-away, teams or robots must learn to coordinate by moving and passing the ball so that an opponent team cannot steal the ball. Many techniques have been applied including genetic algorithms [1,8,11] and reinforcement learning [21,23].

Multiple groups have applied genetic programming techniques to learn team robot soccer policies [1,8,11]. Genetic programming uses genetic algorithm optimization techniques with a set of composable functions in order to "evolve" a program capable of completing some objective. Prior attempts have had limited success, either relying heavily on carefully hand-coded sub-policies or failing to learn cooperative team behaviors.

Reinforcement learning (RL) based approaches have also been popular in this domain [4,21,23]. Stone et al. [21], utilized an options [19] like framework. The low-level controls were hand-coded. The agent learned to call these sub-policies at the appropriate times. They were able to successfully train in up to a 4 teammates vs 3 opponents scenario. While their approach was successful, a significant amount of effort went into determining the proper low level actions

and the proper state-features to use. There is also no guarantee that the higher level features chosen are the best possible features. The features chosen can have a large impact on the final policy performance, and are in practice difficult to choose.

Most of these previous works have focused on learning multi-agent policies in sub-games, such as Keep-away. In this work, we are focused on learning small single agent skills that a human coder can combine into higher level behaviors in the STP architecture.

There has been more recent work on applying Deep RL to single agent skills with parameterized action spaces [5,6]. In Hausknecht, Chen and Stone [5], existing Deep RL techniques were extended to work in an action space that combines discrete and continuous components. They then learned a policy from demonstrations, that allowed the agent to navigate to the ball and score on an empty goal. Hausknecht and Stone [6] later extended this work to learn ball manipulation in this setting completely from scratch. Both of these approaches demonstrated the applicability of Deep RL based algorithms to robot soccer, however, they learned an end-to-end policy to go to the ball and score on the goal. It would be difficult to divide up this policy after training and use the different parts in other contexts. In this work, we aim to learn small reusable skills that can be combined by human written tactics.

4 Approach

4.1 Algorithm

In this work we use the Deep Deterministic Policy Gradient (DDPG) algorithm to train our skills [10]. DDPG is an actor critic, policy gradient algorithm that has been shown to work for continuous action spaces in complex control tasks. Like DQN, DDPG uses a target network for the actor-critic along with a replay memory. Samples are collected and stored in a replay memory. Batches of the samples are used to optimize a critic network which estimates the Q-value of a state-action input. Then the actor network, which takes in a state and returns a continuous action, is optimized to maximize the critic's estimate. We use DDPG in this work, because it is well studied and has been shown to work on a variety of interesting continuous control tasks [10].

4.2 Simulation

We train our skills in the CMDragons simulator. This simulator is physically realistic including configurable amounts of radio latency, observation noise and different robot dynamics models. This simulator has been used to develop the existing STP. The API of the simulator is similar to the real robots, so that a network trained on the simulator can then be run directly on a real robot.

To train the skills we setup different initial conditions in the simulator and applied the DDPG algorithm. When training the simulation is set in "step

mode", meaning that the simulation will only advance when a new command is sent. This guarantees that when training, the extra computation time for updating the network does not cause the agent to miss observation steps. However, when we evaluate the skills we set the simulator to real-time mode. In this mode, the physics runs in real time, and if the agent takes more than 16ms to send a command, then it will miss control time steps.

4.3 Skills

All of the skills use a vector of relevant state features as the input. While Deep RL algorithms such as DDPG can work with pixel based inputs, the state vector allows us to directly include information about velocities. The state-vector representation also requires less computation than an image based representation.

Go to Ball Skill. `go-to-ball` is a skill where the robot learns to navigate to the ball and get the ball on it's dribbler. The `go-to-ball` environment uses positions and velocities in the robot's own frame. By making the coordinates relative to the robot, the learned policy should generalize to different positions on the field better.

The state input for `go-to-ball` skill is as follows:

$$s = (P_x^B, P_y^B, V_x^R, V_y^R, \omega^R, \\ d_{r-b}, x_{top}, y_{top}, x_{bottom}, y_{bottom}, x_{left}, y_{left}, x_{right}, y_{right})$$

where P_x^B and P_y^B are the ball position, V_x^R and V_y^R are the robot's translational velocity, ω^R is the robot's angular velocity, d_{r-b} is the distance from robot to ball, x_{top} and y_{top} are the closest point on the top edge of the field to the robot, x_{bottom} and y_{bottom} are the closest point on the bottom edge of the field to the robot, x_{right} and y_{right} are the closest point on the right edge of the field to the robot, and x_{left} and y_{left} are the closest point on the left edge of the field to the robot.

The action space of this skill is robot's linear velocity, and angular velocity, which are: (v_x^R, v_y^R, ω^R).

The terminal condition for training `go-to-ball` skill is that if the robot has the ball on its dribbler, the episode ends and is marked as a success. If the robot fails to get the ball on its dribbler in 10 s, the episode ends and is considered a failure.

Aim and Shoot Skill. `aim-to-shoot` is a skill where the robot learns to aim towards goal and take a shot. In this skill, we assume that the robot already has a ball on its dribbler.

The state input is as follow:

$$s = (P_x^B, P_y^B, V_x^B, V_y^B, \omega^R, d_{r-g}, \sin(\theta_l), \cos(\theta_l), \sin(\theta_r), \cos(\theta_r))$$

where V_x^B and V_y^B are the x and y translational velocity of the ball, d_{r-g} is the distance from the robot to the goal, $\sin(\theta_l)$ and $\cos(\theta_l)$ are the sine and cosine of the angle of the left goal post with respect to the robot's orientation, $\sin(\theta_r)$ and $\cos(\theta_r)$ are the sine and cosine of the angle of the left goal post with respect to the robot's orientation, and the remaining state components match the `go-to-ball` skill. We use the sine and cosine of the angle, so that there is not a discontinuity in the input state when the angle wraps around from $-\pi$ to π.

The action space of `aim-to-shoot` skill contains robot's angular velocity, dribbling strength and kick strength: $(\omega^R, dribble, kick)$.

The terminal condition for training `aim-to-shoot` skill is that if the robot has kicked and scored, the episode ends and is considered as a success. Otherwise, the episode ends with the following failure conditions: the ball is kicked but does not go into the goal, ball is not kicked yet but the ball has rolled away from the dribbler, or the episode reaches the maximum episode length of 1.6 s.

Reward Functions. We use reward shaping to help speed up the learning. In the `go-to-ball` , our reward function is:

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

where,

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

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

$$r_{orientation} = \frac{1}{\sqrt{2\pi}} \exp\left(-2\frac{\theta_{r-b}}{\pi^2}\right)$$

where θ_{r-b} is minimum angle between the robot's dribbler and the ball.

For the `aim-to-shoot` skill, the agent gets positive reward when it kicks towards the goal and negative when it kicks away from the goal. We also want the robot to shoot as fast as possible on the goal, so we scale the reward by the ball velocity. Kicking fast towards the goal gives higher reward. The reward function for `aim-to-shoot` skill is as follows:

$$r = \begin{cases} 0.05(\alpha - \beta)|V^B| & \alpha > \beta \\ (\alpha - \beta)|V^B| & \alpha < \beta \end{cases}$$

where, α is the angle between left goal post and right goal post, β is the larger angle of one of the goal posts relative to robot's orientation.

4.4 Go to Ball and Shoot Tactic

We combined the `go-to-ball` skill and the `aim-to-shoot` skill into a tactic that can go to the ball, get the ball on the dribbler, turn towards the goal and shoot. Figure 2 shows a flow-chart for the tactic state machine. The robot starts out using the trained `go-to-ball` skill. Once the ball is near the dribbler (defined by $d_{d-b} \leq 35\,\text{mm}$, where d_{d-b} is the distance from the dribbler to the ball), the robot transitions to a fixed "drive forward" skill. The drive forward skill just turns on the dribbler and moves the robot forward for 1 s. After the skill completes, if the ball is no longer near the dribbler (i.e. $d_{d-b} > 35\,\text{mm}$), then the robot starts the `go-to-ball` skill again. Otherwise, the robot starts the learned `aim-to-shoot` skill. If the ball is lost during the `aim-to-shoot` , then the robot transitions back to go to ball and tries again.

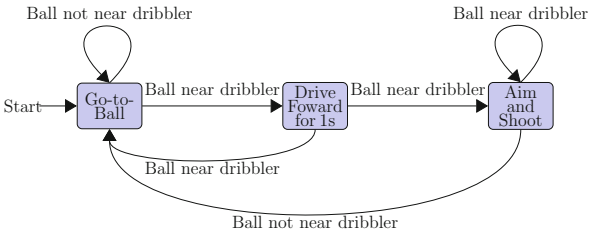


Fig. 2. Flowchart of Go to ball and shoot tactic.

5 Empirical Results

In this section we evaluate the learning performance and the final policy performance of our skills. We also evaluate the performance of our tactic composed of the two learned skills and one hard-coded skill.

5.1 Skill Learning

Tables 1 and 2 shows the hyperparameters used while training both skills. We used an Ornstein-Uhlenbeck noise process [10]. The table also shows the layer sizes for the actor and critic networks. Each layer is a fully connected layer. The hidden layers use ReLU activations, the final layer of the actor uses a tanh activation. The final layer of the critic uses a linear activation. Layer-norm layers were inserted after each hidden layer [2]. When training each skill we initialize the replay memory with examples of successful trajectories. This has been shown in the past to improve the convergence speed of DDPG [24].

During the training of `go-to-ball` skill, we start by collecting 50,000 steps of demonstrations of a good policy as part of the replay memory "burn-in". These samples initialize the replay memory before collecting samples according to the

current actor and noise policy. To get these good demonstrations, we spawn the robot at an arbitrary position on the field so that is facing the ball. We then drive the robot forward until it touches the ball. During the actual training, the robot must learn to reach the ball from arbitrary positions and orientations.

For the training of `aim-to-shoot` skill, we initialize the replay memory with 10,000 samples from a good policy. To get these initial samples we spawn the robot near the goal, facing the goal, and then kick the ball at full strength. During the actual training, the robot must learn to orient itself and the ball towards the goal and then kick.

Figure 3a shows the learning curve from training the go-to-ball skill. The initial part of the curve shows the initial policy demonstrations used to seed the replay. While there is variance in the final policy performance, we see that the agent takes about 500,000 training samples before it has converged to an acceptable policy. Figure 3b shows the average episode length while training. There is a large difference in the maximum number of steps taken to successfully complete an episode between the initial policy and the final policy.

We tested the learned go-to-ball skill against the existing go-to-ball skill. The existing skill moves the robot in a straight line while turning to face the ball. We spawn the ball in a random location of the field. Then the robot is also spawned at a random position and orientation. We then run the skill until either 10 s has elapsed or the ball has been touched on the dribbler.

Figure 4 shows a histogram of the times taken for the go-to-ball skill to get to the ball from 1000 different runs. We can see that while the learned skill has more variance in the times, the max time is still within approximately 2 s of the max time taken by the baseline skill. While the learned skill may take slightly longer, it does reach the ball as intended. The discrepancy in time is likely due to an inability of the DDPG algorithm to perfectly optimize the policy given the training samples. Table 3 shows the success rate of both policies. We see that the baseline is always successful, and the trained policy only failed to get the ball in a single run.

Qualitatively, the path the learned skill takes is very different from the baseline. The baseline is very consistent, moving in a straight line and slowly turning to face the ball. The learned policy's path curves slightly as it adjusts it's orientation to face the ball. Sometimes there are also overshoots in the learned policy. Figure 7 shows an example of a sequence of frames which includes part of the learned `go-to-ball` skill.

Figure 6 shows the training curve for the `aim-to-shoot` skill. This skill's learning curve is more unstable than the `go-to-ball` skill. However, we were still able to utilize the learned policy to aim and score on the goal.

Figure 5a shows the time taken to shoot by the baseline and Fig. 5b shows the time taken to score by the learned policy. Both the baseline and the learned policy were tested on 1000 different runs with different initial conditions. Each run, the robot is spawned at some position, with some orientation, with the ball on the dribbler. We then run the policy and measure the time taken to score. From the figures, we see that again, learned policy takes about 2 s longer to

Table 1. Hyperparameters used for `go-to-ball` skill

Name	Value
Critic learning rate	1×10^{-3}
Actor learning rate	1×10^{-4}
Critic size	$300, 400$
Actor sizes	$300, 400$
Replay mem size	1,000,000
Noise parameters	$\theta = 0.15, \mu = 0,$ $\sigma = 0.3$

Table 2. Hyperparameters used for `aim-to-shoot` skill

Name	Value
Critic learning rate	1×10^{-4}
Actor learning rate	1×10^{-4}
Critic size	$200, 300, 300, 300$
Actor sizes	200—$300, 300, 300$
Replay mem size	600,000
Noise parameters	$\theta = 0.15, \mu = 0,$ $\sigma = 0.3$

Table 3. Success rate for `go-to-ball` skill

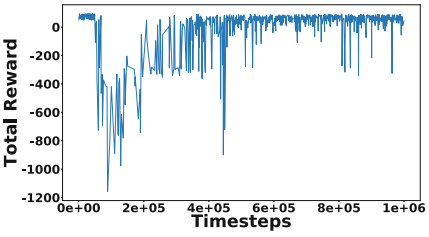
Name	Value
Baseline	1.0
Trained policy	0.999

Table 4. Success rate for `aim-to-shoot` skill

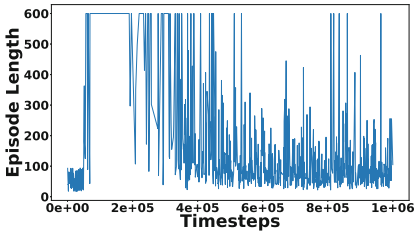
Name	Value
Baseline	0.71
Trained policy	0.772

score the goal on average. We believe the learned policy takes longer because the reward function prioritizes accuracy over time, whereas the hand-coded policy was designed to shoot at the first available opportunity.

Table 4 shows the success rate of the baseline `aim-to-shoot` skill vs the success rate of the learned `aim-to-shoot` skill. While the baseline takes shots on goal faster, we see that the learned policy is actually more accurate by approximately 6%. This makes sense, as our reward function gives negative rewards for failures, so the agent will be incentivized to prioritize good aim over time taken to score.

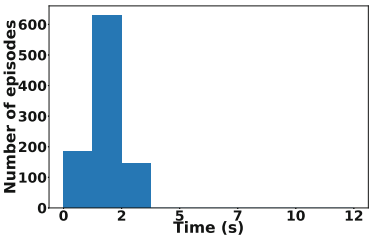


(a) Total reward vs Number of Samples while training `go-to-ball` skill. Higher is better. Initial part of the curve shows replay memory burn-in.

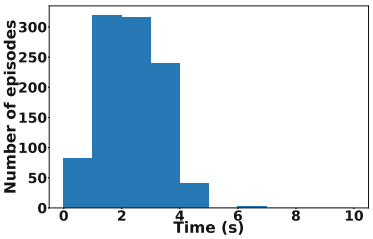


(b) Number of time-steps vs Number of samples while training `go-to-ball` skill. Each time-step is equal to 0.16ms. Lower is better. Initial part of the curve shows replay memory burn-in.

Fig. 3. Training curves for `go-to-ball` skill.

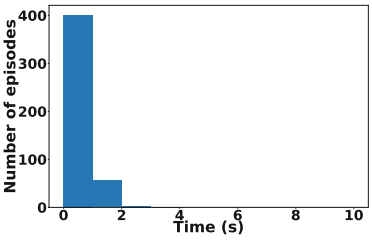


(a) Time taken by existing `go-to-ball` skill.

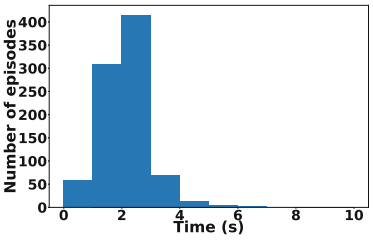


(b) Time taken by neural network `go-to-ball` skill.

Fig. 4. Comparison of existing `go-to-ball` skill vs learned `go-to-ball` skill.



(a) Time taken by existing `aim-to-shoot` skill.



(b) Time taken by neural network `aim-to-shoot` skill.

Fig. 5. Comparison of existing `aim-to-shoot` skill vs learned aim-to-shoot skill.

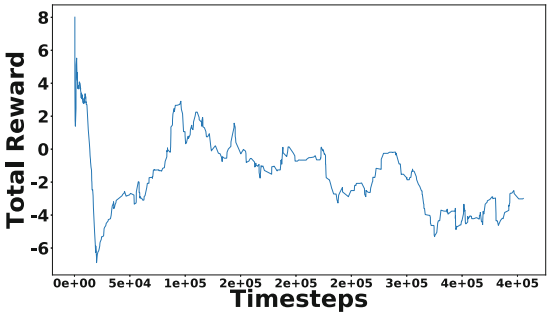


Fig. 6. Total reward vs number of samples while training `aim-to-shoot` skill. Higher is better.

5.2 Tactics Evaluation

In order to be useful in an STP hierarchy, the learned skills must be easily composable by human coders. The tactic state machine from Fig. 2 was implemented using the learned `go-to-ball` and `aim-to-shoot` skills. To evaluate the performance, we executed the state machine across 500 different runs. Each run, the robot was spawned at a random location on the field with a random orientation.

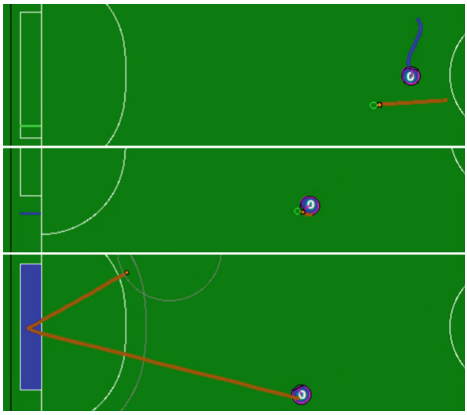


Fig. 7. Sequence of key-frames from execution of go to ball and shoot tactic using learned skills. The blue circle shows the robot. The blue line shows the history of the robot's trajectory. The orange circle is the ball and the orange line shows the trajectory of the ball. The following link contains videos of the simulated policy: https://goo.gl/xB7VAE (Color figure online)

The ball was also spawned at a random location on the field. We then run the tactic until either (1) a goal is scored or (2) the maximum time of 15 s elapses. The tactic was able to succeed 75.5% of the time. On average the tactic took 7.49 s with a standard deviation of 3.87 s.

6 Conclusion

In this work we have shown that Deep RL can be used to learn skills that plug into an existing STP architecture using a physically realistic simulator. We have demonstrated learning on two different skills: navigating to a ball and aiming and shooting. We showed that these learned skills, while not perfect, are close in performance to the hand-coded baseline skills. These skills can be used by humans to create new tactics, much like how hand-coded skills are used. We show that using a simple state machine, the two skills can be combined to create a tactic that navigates to the ball, aims, and shoots on a goal. Given these results, we believe that reinforcement learning will become an important part in future competitions. Future work will address how well the learned policies transfer from the simulation to the real robots.

References

1. Andre, D., Teller, A.: Evolving team Darwin united. In: Asada, M., Kitano, H. (eds.) RoboCup 1998. LNCS, vol. 1604, pp. 346–351. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48422-1_28
2. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
3. Browning, B., Bruce, J., Bowling, M., Veloso, M.: STP: skills, tactics and plays for multi-robot control in adversarial environments. J. Syst. Control Eng. **219**, 33–52 (2005). The 2005 Professional Engineering Publishing Award
4. Fernandez, F., Garcia, J., Veloso, M.: Probabilistic policy reuse for inter-task transfer learning. Robot. Auton. Syst. **58**, 866–871 (2009). Special Issue on Advances in Autonomous Robots for Service and Entertainment
5. Hausknecht, M., Chen, Y., Stone, P.: Deep imitation learning for parameterized action spaces. In: AAMAS Adaptive Learning Agents (ALA) Workshop, May 2016
6. Hausknecht, M., Stone, P.: Deep reinforcement learning in parameterized action space. In: Proceedings of the International Conference on Learning Representations (ICLR), May 2016
7. Heess, N., et al.: Emergence of locomotion behaviours in rich environments. CoRR (2017). http://arxiv.org/abs/1707.02286v2
8. Hsu, W.H., Gustafson, S.M.: Genetic programming and multi-agent layered learning by reinforcements. In: GECCO, pp. 764–771 (2002)
9. LaValle, S.M., Kuffner Jr., J.J.: Randomized kinodynamic planning. Int. J. Robot. Res. **20**(5), 378–400 (2001)
10. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. In: Internal Conference on Learning Representations (2016). http://arxiv.org/abs/1509.02971v5
11. Luke, S., Hohn, C., Farris, J., Jackson, G., Hendler, J.: Co-evolving soccer softbot team coordination with genetic programming. In: Kitano, H. (ed.) RoboCup 1997. LNCS, vol. 1395, pp. 398–411. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-64473-3_76
12. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning (2016). http://arxiv.org/abs/1602.01783v2
13. Mnih, V., et al.: Playing atari with deep reinforcement learning (2013). http://arxiv.org/abs/1312.5602v1
14. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
15. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. CoRR, abs/1502.05477 (2015)
16. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
17. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961
18. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
19. Stolle, M., Precup, D.: Learning options in reinforcement learning. In: Koenig, S., Holte, R.C. (eds.) SARA 2002. LNCS, vol. 2371, pp. 212–223. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45622-8_16

20. Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway soccer: from machine learning testbed to benchmark. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS, vol. 4020, pp. 93–105. Springer, Heidelberg (2006). https://doi.org/10.1007/11780519_9
21. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup soccer keepaway. Adapt. Behav. **13**(3), 165–188 (2005). https://doi.org/10.1177/105971230501300301
22. The RoboCup Federation: RoboCup (2017). http://www.robocup.org/
23. Uchibe, E.: Cooperative behavior acquisition by learning and evolution in a multi-agent environment for mobile robots. Ph.D. thesis. Osaka University (1999)
24. Vecerik, M., et al.: Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. CoRR (2017). http://arxiv.org/abs/1707.08817