We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

154
Countries delivered to

Our authors are among the

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS

BOOK CITATION INDEX

INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter

A Q-Learning-Based Approach for Simple and Multi-Agent Systems

Ümit Ulusoy, Mehmet Serdar Güzel and Erkan Bostanci

Abstract

This study proposes different machine learning-based solutions to both single and multi-agent systems, took place on a 2-D simulation platform, namely, Robocode. This dynamic and programmable platform allows agents to interact with the environment and each other by employing a variety of battling strategies. Q-Learning is one of the leading and popular machine learning-based solutions to be applied to such a problem. However, especially for continued spaces, the control problem gets deeper. Essentially, one of the main drawbacks of reinforcement learning (RL) is to design an appropriate reward function that the function can be described by only employing few parameters for simple tasks, whereas estimating the goal of the reward function may be a challenging problem. Recent studies prove that neural network-based approaches can handle these challenges and achieve to learn control strategies from 2-D or 1-D data. Besides those problems of RL algorithms for single robots, once the number of robots increases and the systems need to behave as multi-agent systems, the overall design requirements become more complex. Accordingly, the proposed system is validated by considering different battle scenarios. The performance of the Q-Learning-based system and the supervised learning techniques are compared by employing different scenarios for this problem. Results reveal the superiority of the ANN-based approach over other methods.

Keywords: multi-agent systems, Q-Learning, Robocode, auto-encoder, neural network, battling strategy

1. Introduction

Swarm intelligence is a scientific field that integrates the fields of swarm intelligence and cooperative robotics to establish and coordinate robots to achieve challenging tasks within a reasonable time [1, 2]. Multi-agent systems, on the other hand, are considered to be coordination of autonomous agents so as to complete tasks by exchanging or sharing information over a network. This resembles the swarm intelligence discipline in similar ways [3, 4], as it has been previously noted that multi-agent systems mainly deal with the coordination of multiple interacting agents so as to complete different tasks. The key objective of those systems is to coordinate rather simple agents instead of using a complex agent [5]. The coordination ability of agents gains different skills to these systems that individual agents may not be allowed to achieve, which are, namely, robustness, scalability, and flexibility [5, 6]. The Robocode platform on the other hand is a game developing

platform and allows developers to design robot battle tanks to battle against other tanks [7]. The battles are running in real time, and the game is played on a two-dimensional simulation environment by employing single or multiple robots, as shown in **Figure 1**. These robots can be defined as single robot, or some of them can be marked as team robots. Each of them is possessed with battling behaviors which allows them to decide movement, fire, and targeting in order to keep their energies high and destroy their opponents. The time is measured with ticks, and each robot is allowed only one movement for each tick. At the end of each round (game), the total score for each attendee is calculated by their "fire damage," "ram damage," and "survival status." This lets a team to obtain the highest score even if their robots did not survive.

The flexibility, scalability, and robustness of the Robocode platform encourage authors to employ machine learning-based approaches for multi-agent system problems. Despite their advantages, the platform also offers some challenges that should be handled in an appropriate manner. The critical issues are detailed as follows:

- Opponent rounds are not visible and the environment is not fully observable.

- Sensors used by robots are limited.

- The number of action is quite high which makes learning harder.

- The speed of robots slows down during firing and turning behaviors.

- Once the gun of robot's temperature is high, firing behavior does not work, which forces users to consider all parameters.

Robocode gathers great deal and attention from a big community including researcher, students, and engineers, in which design concepts and source codes are

Figure 1.
An example screenshot from the Robocode environment [7].

shared. Tournaments and leagues are arranged via websites. Hence, the rankings of customized robots are continuously updated [7]. Therefore, game strategies are very critical and continuously evolve by utilizing different approaches. Robocode game strategies are characterized as trees of atomic elements agreeing to actions and observations in a battle.

Machine learning has been widely used in single [8, 9] and multi-agent systems [2, 3]. This also encourages researchers to apply machine learning or meta-heuristic-based methods to train and prepare robotic teams for this battling process of Robocode environment. For instance, there exist studies employing genetic algorithm in order to generate various and evolving behaviors using genetic algorithm [8, 10]. Besides, decision tree and neural network-based solutions have been employed to estimate a strategy to obtain a higher rank in the league [11, 12]. Those studies prove that machine learning is an efficient way of designing and implementing strategies for such an environment. Accordingly, this study is inspired from those previous studies and introduces three different machine learning-based approaches to train and prepare robots for the battle. The first approach mainly employs reinforcement learning to train a single and team robot separately so as to allow them to survive in a tank battle. It is proven that despite its discrete structure, Q-Learning can be adapted for such a complex game. In addition, a neural network-based design is also implemented in order to compare the results, which has been previously employed in a similar study [13]. Finally, an auto-encoder-based model is designed to train a number of robots, allowing them to battle to the death in an arena. Similar studies can be seen in [14–16]. Next section mainly introduces the proposed methods separately. The experiments are defined, and results are evaluated in a detailed manner at the experimental section. Lastly, the study is concluded at the conclusion section.

2. Methodology

This study proposes three different machine learning-based solutions to the multi-agent battling game. The first of them employs reinforcement learning approach, aiming extracting the maximum award from the network used in learning procedure. The second approach, on the other hand, relies on a supervised learning algorithm based on an artificial neural network architecture. Finally, an auto-encoder-based model has been designed and implemented to train the robots for the challenge. Each of those solutions will be detailed, respectively.

2.1 Q-Learning robot for Robocode

Reinforcement learning (RL) aims to take suitable action to maximize reward in a specific situation. It is employed by various software and machines to find the best possible behavior in particular situation. Reinforcement learning differs from the supervised learning that the agent agrees what to do to achieve with the given task. Instead of employing a training dataset, the agent learns from its experiences. Q-Learning is the most popular RL algorithm and preferred for this study due to its efficiency and popularity [17]. The agent mainly observes the environment and performs the action by employing the previously defined action. The agent then obtains the action consequence or award from the environment. This state and action pair is kept for future usage since it gives clues about the reward. The algorithm mainly aims to generate a Q-Table which illustrates maximum expected future rewards for action at each state. Q-Learning update rule is designed based on

the Bellman equation so as to estimate the optimal Q-Value, and the Q-Learning update rule is given as Eq. (1):

$$Q(s_t, a_t)^{new} = (1 - \alpha)\mathbf{x}Q(s_t, a_t)^{old} + \alpha\mathbf{x}(r_t + \gamma\mathbf{x} \max Q(s_{t+1}, a_t)) \qquad (1)$$

where, at each time "t," the agent selects an action "a_t," observes a reward "r_t," and enters a newer state "s_{t+1}." Besides, α refers to learning rate, whereas γ illustrates discount factor. Within the given algorithm and approach, a Q-Learning robot is designed according to the rule and environment of Robocode. Accordingly, any robot knows enemy position, bearing angle, and distance to enemy by employing its radar; then at each step, a robot selects an action that maximizes the upcoming reward for the current state. These state and action pairs are stored in table that is updated during the game, and robot collects rewards at each "thick" to update the table (Q-Table) as a result of applied actions. The lookup table contains following states and actions for the proposed Q-Learning robot for the Robocode problem (see **Table 1**). The flow chart of the Q-Learning robot is given in **Figure 2**.

The pseudocode of the Q-Learning algorithm is given as follows:

Q-Learning algorithm:
Requires *States s:1 to n;*
   Actions a:1 to m;
   Reward Function;
α (learning rate), γ : (discount factor)
Ensures *Updated Q-Table for action state coordination*
Procedure *Q-Learning*
Initialize state-actions table *Q(s,a)*
Current state "s" should be selected
   While(A final state or threshold value is obtained)
   Basing on the action selection policies select and action **a**
   Obtain reward **r** for selected action alongside with the next state.
   Update Q value for current state **s** and for following state according to (1) and parameters
   EndWhile
EndProcedure

2.2 Artificial neural network robot for Robocode to approximate Q-Values

Artificial neural network is considered as universal approximators that can be adapted in many different and challenging problems. Several studies have already been applied to Robocode environment; some of those references can be seen in Section 1. Accordingly, a multilayer perceptron inspired from those studies has been adapted and designed for this study. It mainly aims to search the best output for

States	Actions
Robot location	Run away from the enemy
Enemy location	Move toward the enemy
Bearing angle with the enemy	Hold the current position
Energy level	Spin clockwise or anticlockwise

Table 1.
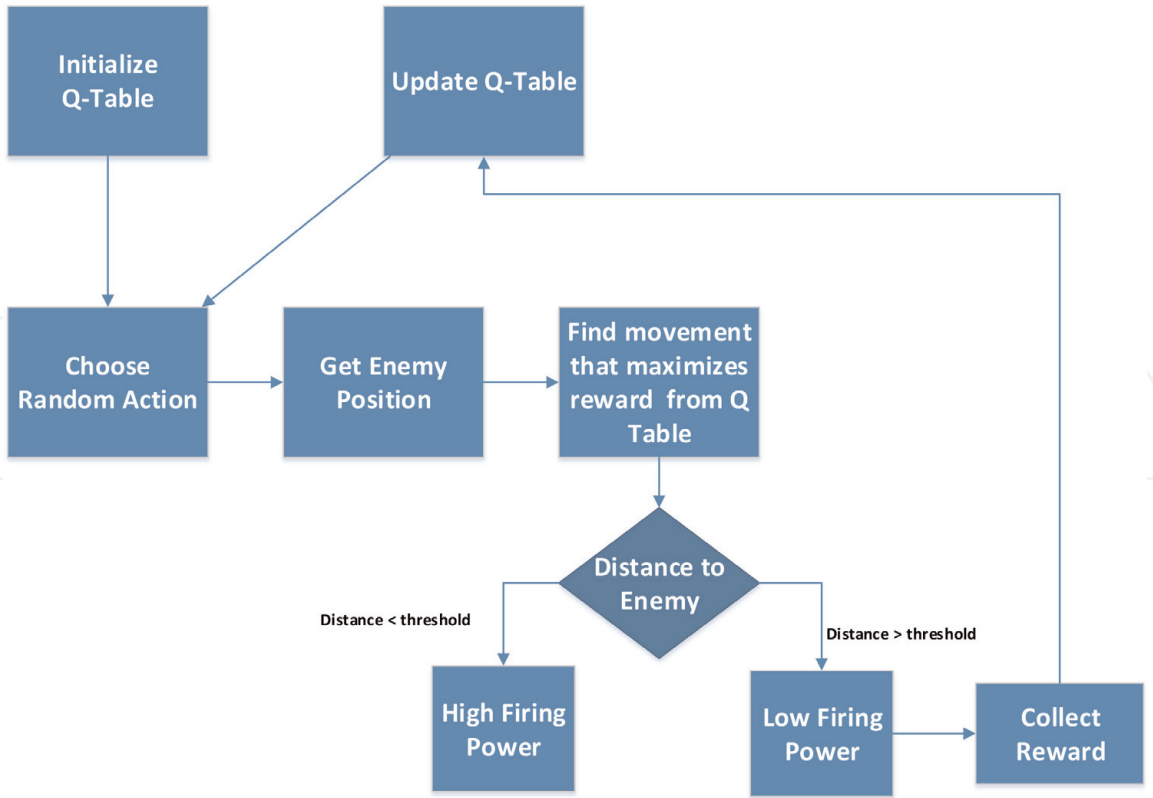*List of states and actions for Q-Learning robot of Robocode.*

Figure 2.
Q-Learning robot training flow chart.

each Robocode thick based on actions and states that mainly allows us to approximate maximum Q-Values.

The proposed neural network has a very simple structure which consists of two layers. The first layer represents inputs, namely, X position, Y position, the distance between the robot and the opponent, the bearing angle, action, and bias values. The second layer on the other hand is a fully connected layer. The final layer represents the Q-Values, as seen in **Figure 3**. Sigmoid function is employed as the activation function, and also, node numbers at the hidden layer are estimated by trial and error method that results in higher learning accuracy.

2.3 Deep auto-encoders applied Robocode to approximate Q-Values

Stacked sparse auto-encoder is a type of deep neural network involving stacking sparse auto-encoders, and a classifier is regularly used as the final layer for mainly classification or regression problems [18]. This model has not been applied in such a problem which encourages authors to employ the technique into the current problem. Consequently, an example model is designed and given for this problem shown in **Figure 4**.

Accordingly, the first auto-encoders are trained by utilizing an unsupervised training method [18]. Fundamentally, the output of the first sparse auto-encoder is considered as an input to the second one, and the output of second auto-encoders becomes an input to the classifier as shown in the corresponding figure. The auto-encoders and the classifier "SoftMax" are stacked and qualified in a supervised manner by employing the backpropagation algorithm for estimating the optimum Q-Value. Each auto-encoder is trained by employing using the cost function illustrated in Eq. (1). E_r value is regulated by employing mean square error (MSE) approach:

$$E_r = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} (x_{tm} - \hat{x}_{tm})^2 + \lambda * \Omega \, weights + \beta * \Omega \, sparsity \qquad (2)$$

where *Er* is the error rate, *x* is the input and $\hat{x}$ is the restored data, "*λ*" coefficient is used by L2 "Weight Regularization" and "*β*" coefficient is used for the "Sparsity Regularization," *m* is the number of observations, and *t* illustrates the training data label number.

The Ω *weights* illustrates "Weight Regularization" and is defined as flows:

$$\Omega \, weights = \frac{1}{2} \sum_{x}^{X} \sum_{j}^{J} \sum_{k}^{K} w_{jk}^{(x)^2} \qquad (3)$$

Here X indicates the number of hidden layers, n signifies observation numbers, and k shows hidden layers [18].

Sparsity Regularization is on the other hand can be defined as follows:

$$\Omega \, sparsity = \sum_{i=1}^{D} KL(\rho \| \hat{\rho}_i) \qquad (4)$$
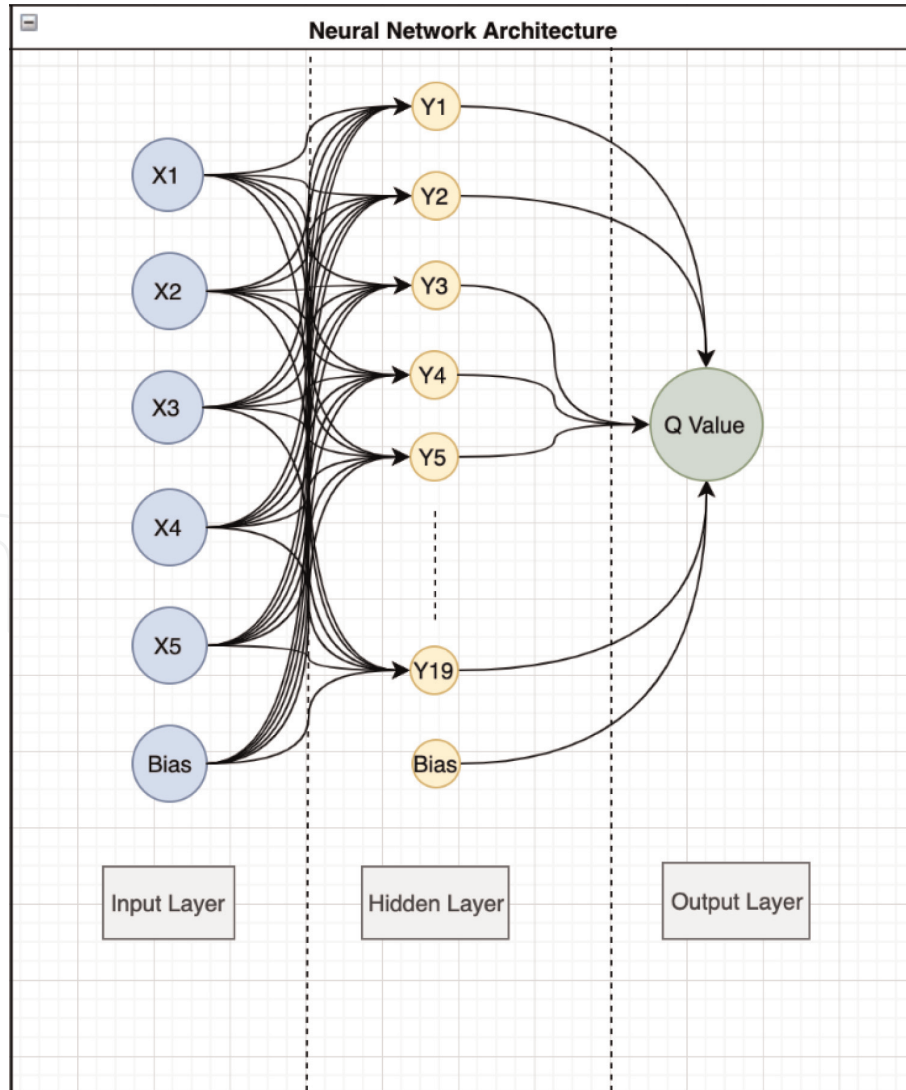
Figure 3.
ANN architecture to approximate Q-Values.

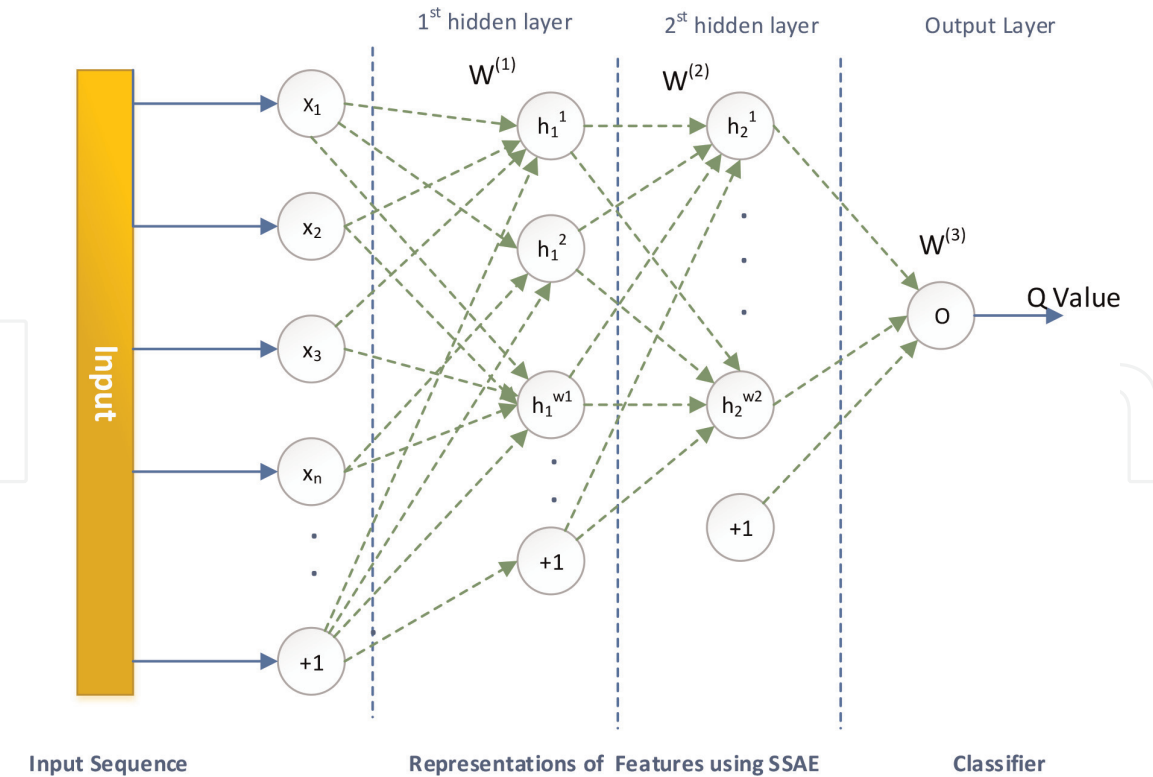**Figure 4.**
A stacked sparse auto-encoder (SSAE) having two hidden layers and softmax classifier.

where anticipated value is represented by ρ, $\hat{\rho}_i$ denotes the average output activation of each neuron "i," and "KL" is the function that evaluates the variance between two probabilities distribution over the same data. The details of those equations can be seen in [18].

3. Experimental results

Aforementioned Robocode is a tank-combat emulator developed by IBM alphaWorks [7]. Basically the tank or teams must navigate the environment to avoid being shot by its rivals. Three different machine learning-based approaches are employed to train the single and multi-agent systems to win the battle against their opponents in an autonomous manner. A desktop computer having Intel Core i7-6700 CPU @ 2.60-GHz and 16-GB RAM is employed to conduct experiments. Each method and results are illustrated separately by defining scenarios.

3.1 Scenario 1

This scenario illustrates a single robot battle, in which a Q-Learned customized robot (AUQRobot) fights against the Spin Robot. An example screenshot is illustrated in **Figure 5**. Within the scenario up to 12,000 round took place to train the robot. **Figure 6** illustrates the change of the winning percentage along the rounds. Regarding to the graph, it is very clear that winning percentage is up to 87% with the power of reinforcement learning that provided greedy method. Collaterally, collected cumulative reward change along the rounds gives a same curve. This result is obtained under Robocode maximum data storing constraints. Results are experienced with a Q-Table includes 9216 elements, and it is also noted that increasing the table size will probably increase the overall winning performance.
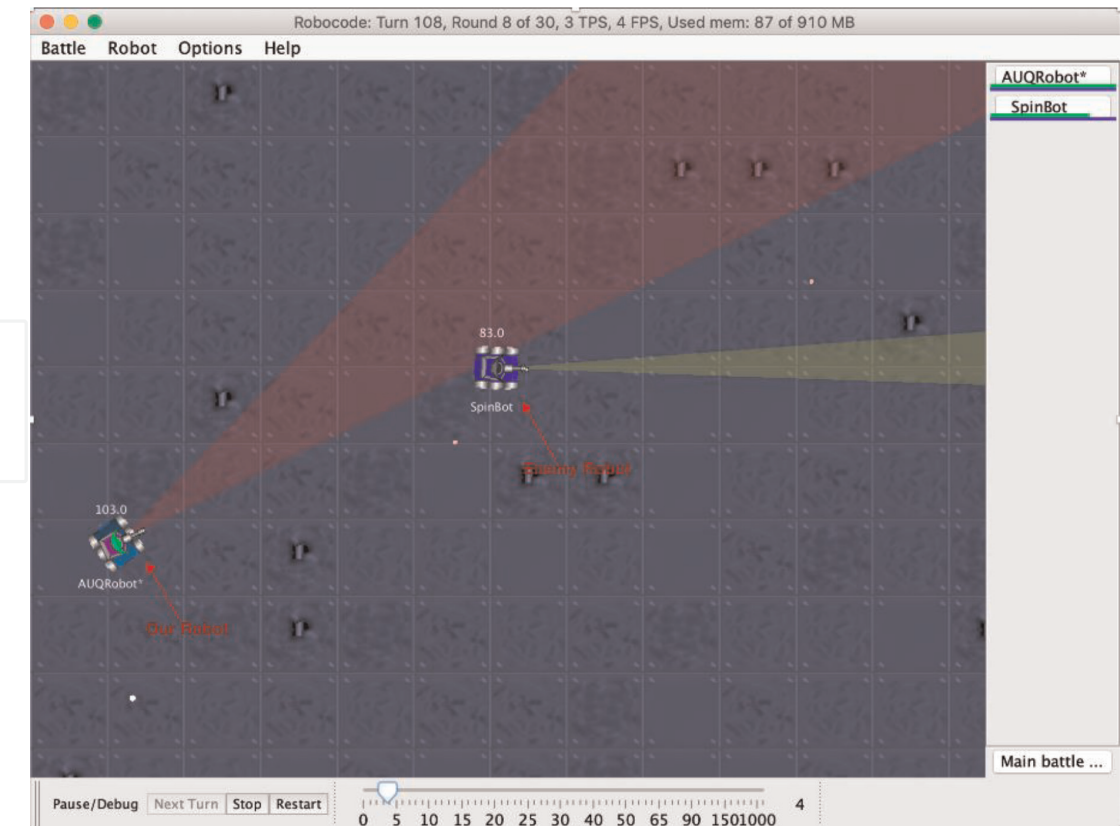
7

Figure 5.
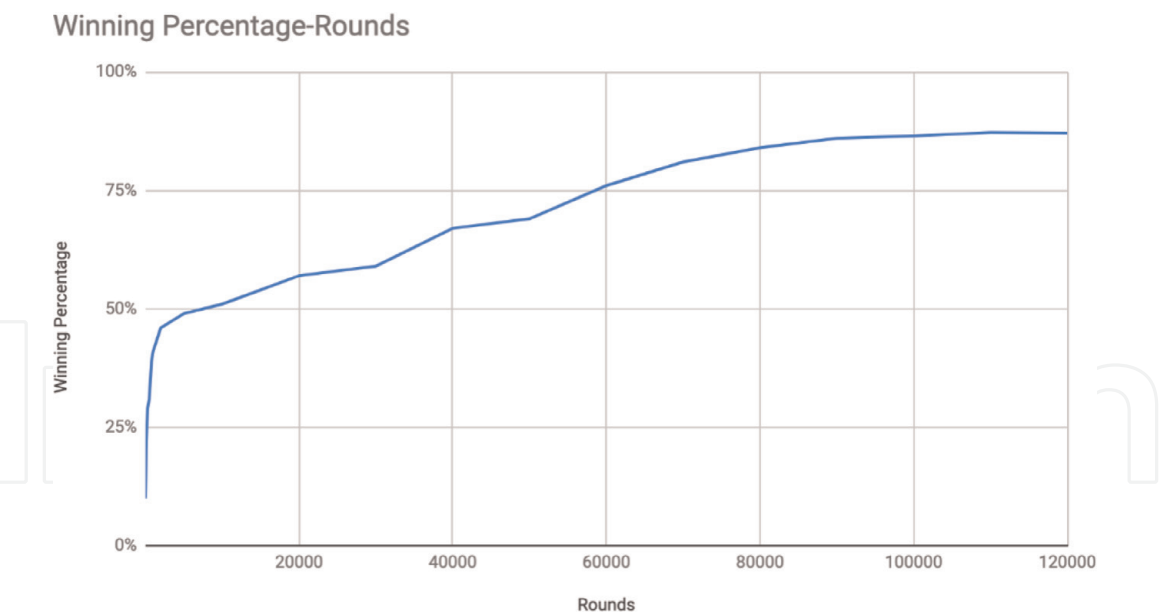*A screenshot obtained from Scenario 1 (SpinBot Robot).*

Figure 6.
Winning percentage of AUQRobot within rounds.

Within this scenario, Q-Learned customized robot (AUQRobot) fights against the TrackerBot Robot (see **Figure 7**), which is also a popular robot used in Robocode. For this scenario, the same training configuration is also applied, and 92% winning rate is also obtained. An example screenshot obtained from the Robocode platform is shown in **Figure 8**. Regarding to the results, Tracker Robot never survived during a 20-round battle. AUQRobot has lost 8% against to the opponent.

3.2 Scenario 2

This scenario illustrates a single robot battle, in which a customized robot (AUNNRobot) fights against the Spin Robot. The robot, which was implemented according to the artificial neural network architecture described above, was trained against SpinBot within 200 and 50,000 iterations. **Table 2** illustrates the configuration of ANN-based system.

The neural network performing linear regression and Q-Value, obtained from Q-Learning algorithm, was employed to train the system. **Figure 9** illustrates the
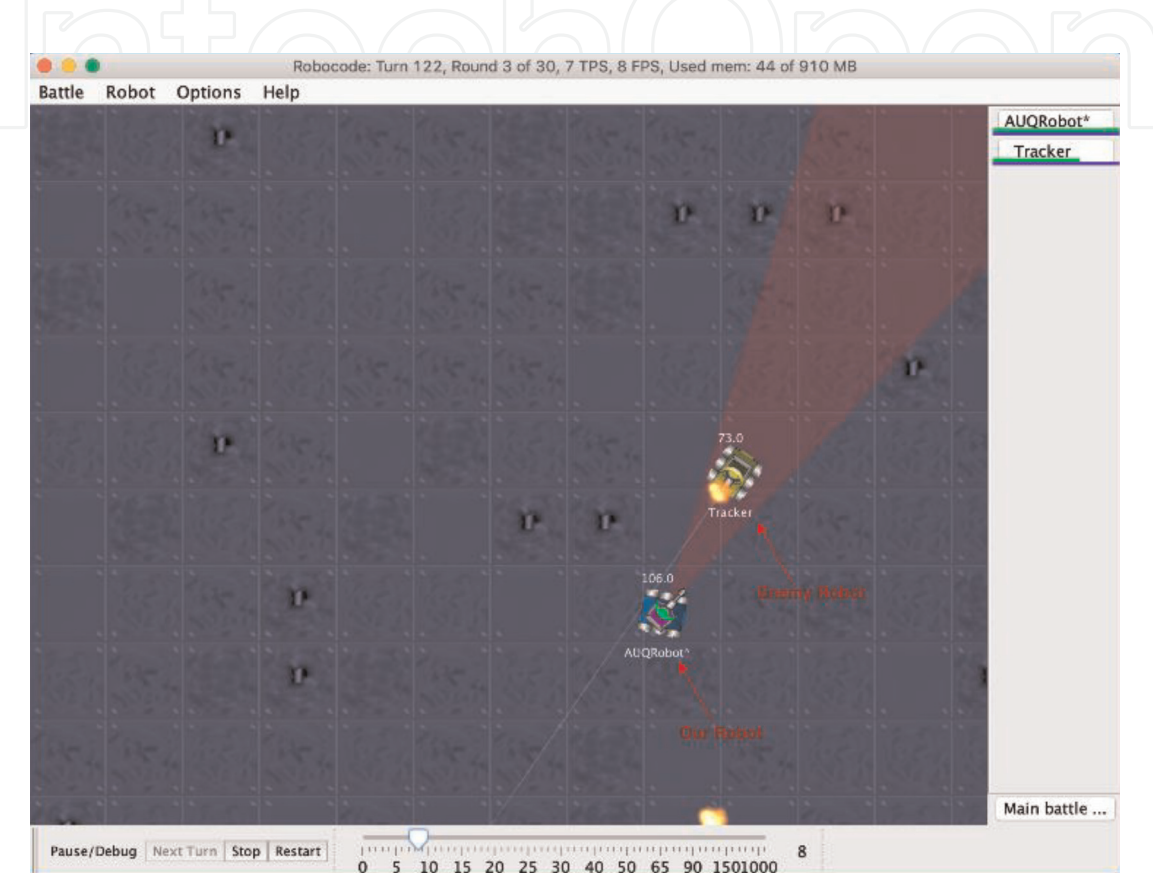
Figure 7.
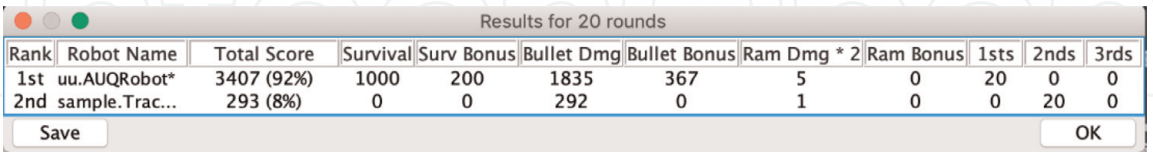*A screenshot obtained from Scenario 1 (Tracker Robot).*

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.AUQRobot*	3407 (92%)	1000	200	1835	367	5	0	20	0	0
2nd	sample.Trac...	293 (8%)	0	0	292	0	1	0	0	20	0

Figure 8.
A screenshot illustrating Total score for Scenario 1 (Tracker Robot).

Input name	Parameter range	NN input range
Position X	0–800px	{0.0, 0.1 … 7.9, 8.0}
Position Y	0–600px	{0.0, 0.1 … 7.9, 8.0}
Distance to enemy	0–1000px	{0.0, 0.1 … 5.9, 6.0}
Bearing angle between robot and enemy	0°–360°	{0.0, 0.1 … 5.9, 6.0}

Table 2.
Configuration of ANN-based system.

9

winning percentage of the ANN-based robot within training procedure. Results reveal that the ANN-based method starts learning rapidly but converge lately when compared with reinforcement-based approach.

Accordingly, it has been considered that a battle between AUQRobot and AUNNRobot, both have already been trained for same robot class, may compare both systems performance appropriately (see **Figure 10**). In general, none of the participants are able to outperform the opponent clearly, but AUNNRobot has an advance as 54–46% over the AUQRobot based on 50 rounds as can be seen in **Figure 11**.

The results of deep auto-encoder-based method have also been trained that the winning percentage of the network with respect to the training data is also illustrated in **Figure 12**, namely, AUAERobot (see **Figure 13**) that, however, provides less wining rate compared with AUNNRobot. It should be noted that if raw image data is employed as input instead of giving position values to the network, the deep
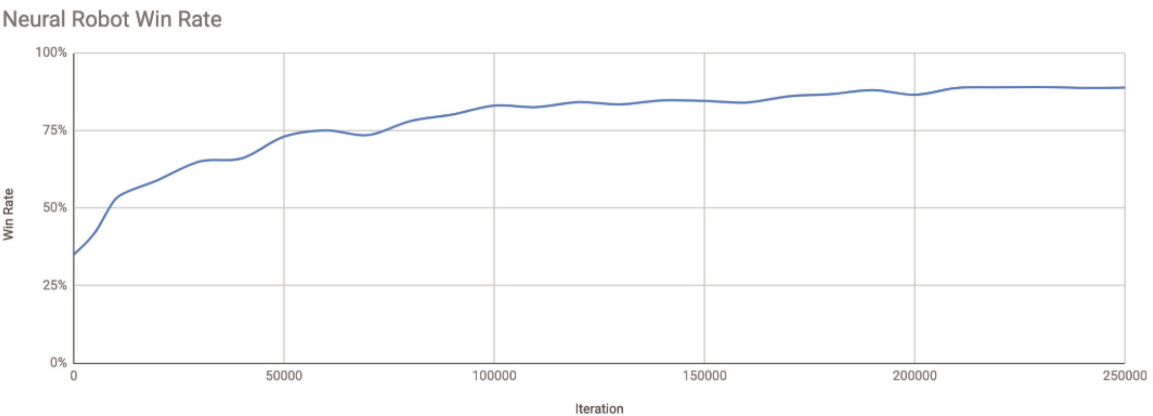

Figure 9.
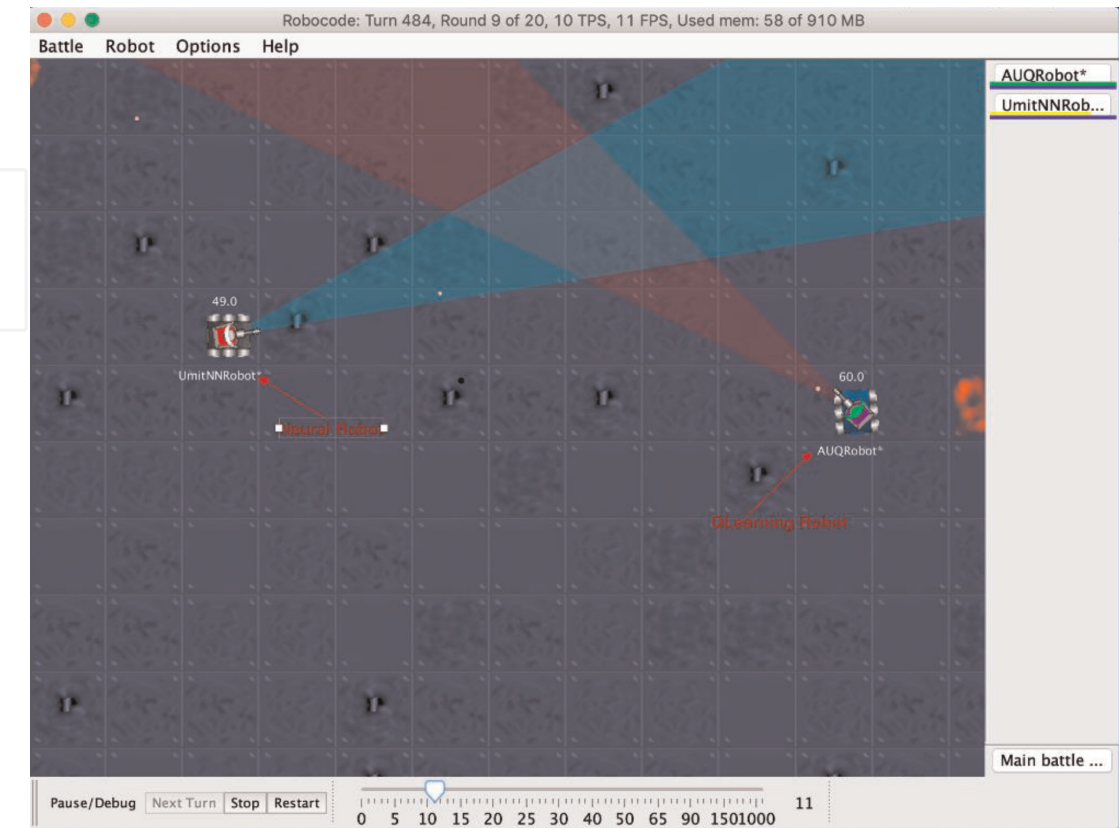*Winning percentage of AUNNRobot within rounds.*

Figure 10.
A screenshot obtained from Scenario 2 for AUQRobot vs. AUNNRobot.

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.UmitNNRobot*	3771 (54%)	900	180	2474	215	1	0	20	30	0
2nd	uu.AUQRobot*	3149 (46%)	1500	300	1186	158	6	0	32	18	0

Figure 11.
Results for 50 rounds for Scenario 2.
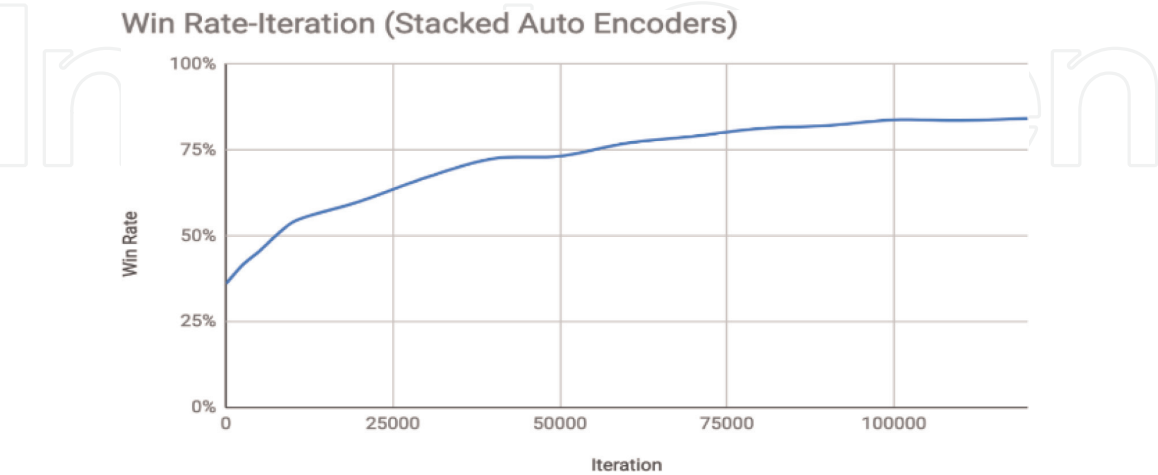
Figure 12.
Winning percentage of AUAERobot within iterations.
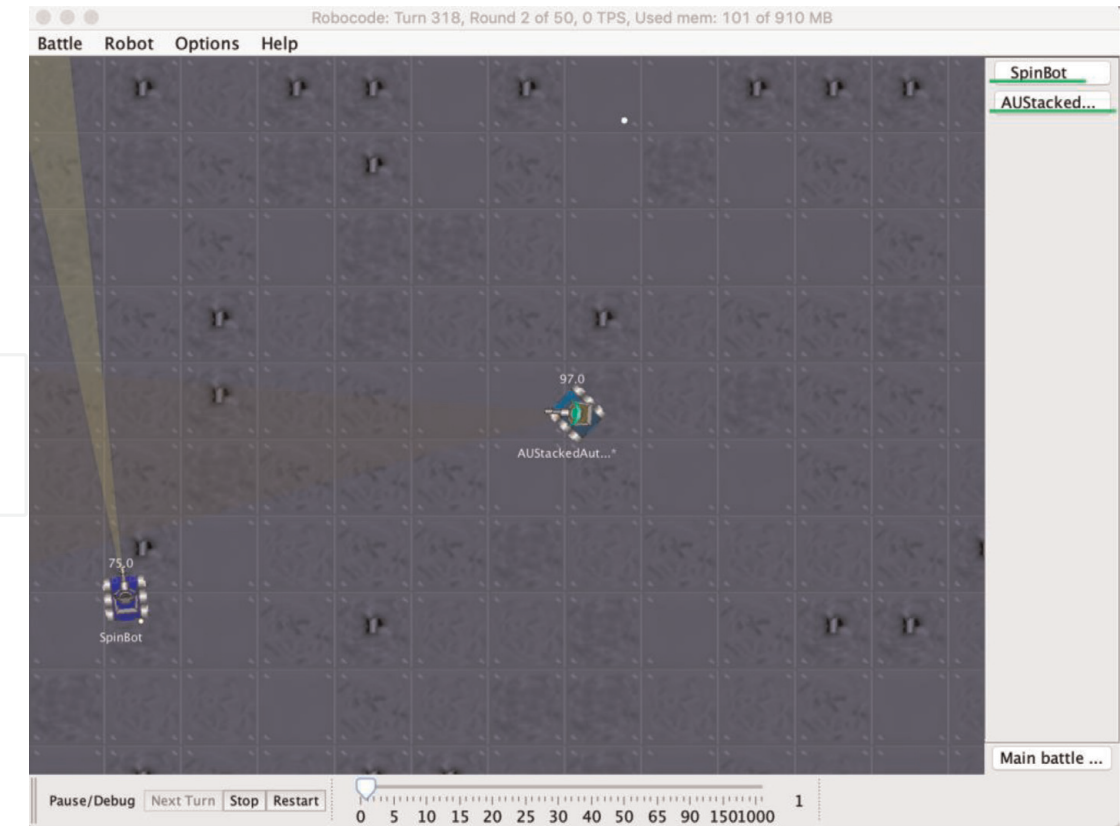
Figure 13.
A screenshot obtained from Robocode for Scenario 2 (AUAERobot).

auto-encoder and also CNN-based architectures may outperform the AUNNRobot architecture, which is, on the other hand, costly in terms of obtaining training data as well as performing training process compared with aforementioned approaches. Despite given challenges, it is planned to apply those architectures to the given

problem as future works. Consequently, instead of AUAERobot, AUNNRobot is preferred to compete with AUQRobot.

3.3 Scenario 3

This scenario illustrates a multi-agent robot battle, in which a customized robot (AUQRobot) fights against with SpinBot Team. Since one of the starting points of this study aims to perform multi-agent team battles, first the trained robot forms a team against a single robot class. Afterward, the members of this team are programmed not to strike each other. Finally, the robot class, where the training is achieved, is defined as a robot team of AUQRobot. **Figure 14** illustrates a screenshot from the battlefield. Results reveal that customized decentralized AUQRobot teams outperform "SpinBot Team" with an average of 65–35% as shown in **Figure 15**.

3.4 Scenario 4

This scenario illustrates a battle of multi-agent systems consisting of five robot tanks. According to which, the first team is inherited from AUQRobot, whereas the

Figure 14.
A screenshot obtained from Scenario 3 (multi-agent battle).

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	uu.AUQTeamRobot* ...	5897 (17%)	3450	450	1833	96	67	0	5	0	1
2nd	uu.AUQTeamRobot* ...	5251 (15%)	3250	0	1806	104	90	0	1	3	3
3rd	uu.AUQTeamRobot* ...	4836 (14%)	2750	180	1748	80	67	11	3	0	0
4th	uu.AUQTeamRobot* ...	4401 (13%)	2550	0	1687	77	88	0	0	2	2
5th	uu.TeamSpinRobot* (1)	3171 (9%)	2500	90	383	9	178	12	1	1	2
6th	uu.TeamSpinRobot* (4)	2978 (9%)	2250	90	415	10	210	3	1	0	0
7th	uu.TeamSpinRobot* (2)	2487 (7%)	2000	0	350	3	128	5	0	2	0
8th	uu.AUQTeamRobot* ...	2052 (6%)	950	0	1014	33	55	0	0	0	0
9th	uu.TeamSpinRobot* (5)	1939 (6%)	1450	0	348	1	140	0	0	1	2
10th	uu.TeamSpinRobot* (3)	1655 (5%)	1300	0	249	3	103	0	0	0	0

Figure 15.
A screenshot obtained from Robocode to illustrate results for Scenario 3.
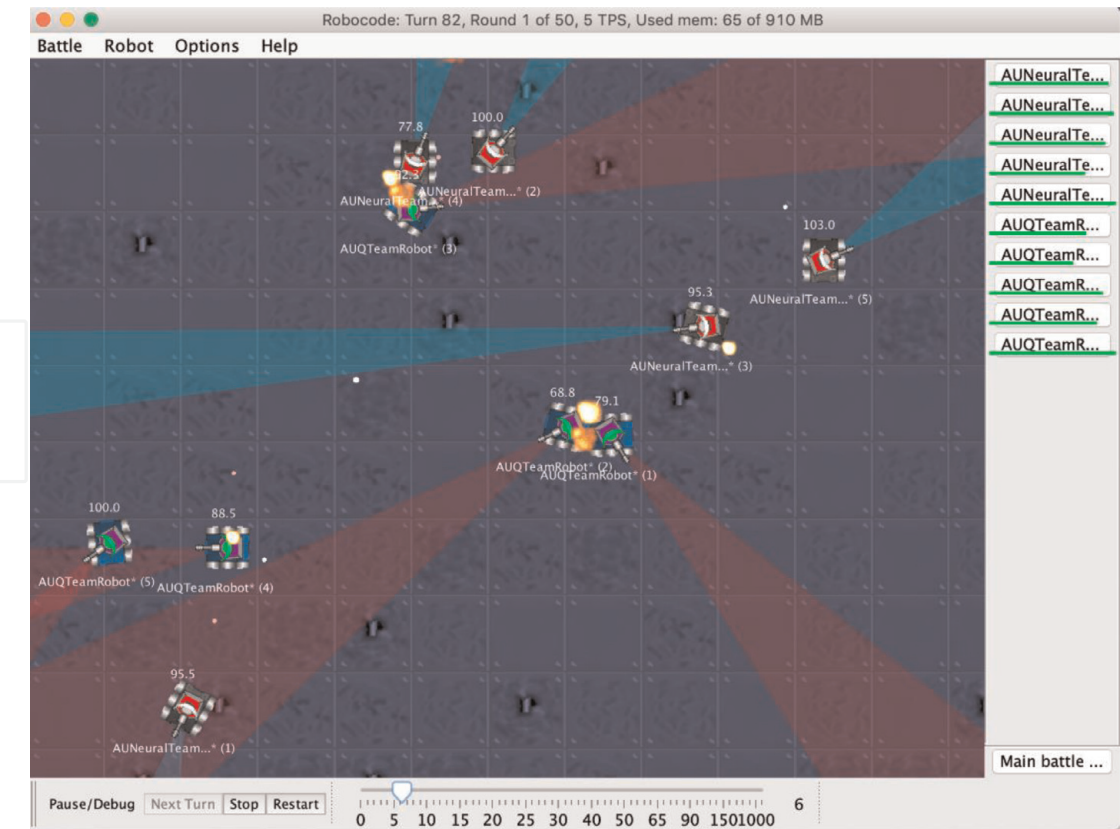
Figure 16.
A screenshot obtained from Robocode for Scenario 4.

Figure 17.
A screenshot obtained from Robocode to illustrate results for Scenario 4.

second one is inherited from the AUNNRobot model. An example screenshot obtained from this scenario is shown in **Figure 16**. Several different competitions (experiments) were conducted between those robot teams, and results reveal that the AUNNRobot team outperforms its opponent from 67% (minimum) to 74% (maximum) winning rate. **Figure 17**, generated from the Robocode platform during the experimental procedure, includes 50-round battle and illustrates the team performance of AUNNRobot against its opponent.

4. Conclusion

This paper introduces and compares some of the popular machine learning-based approaches for the single and multi-agent systems by employing a popular 2-D game simulator, namely, Robocode. This platform essentially allows researchers to design customized robot teams so as to join the competition and perform tank battle players and designers all around the world. Despite the challenges of continued space problem with respect to the characteristics of the games, a

Q-Learning-based model is introduced for the problem. Besides, an ANN-based model is designed to approximate Q-Values instead of constructing a huge Q-Table, which in essence is not a realistic approach. In addition, previous experiences prove that stacked auto-encoders (SAEs) may offer an alternative supervised learning approach once the labeled data is obtained. However, as position data is employed instead of a raw image, SAEs do not provide any advances such as denoising on images or reducing the input size. Within these results, it should be noted that raw images, illustrating game states, should better be employed by the deep architectures, as input to design a stronger architecture than an ANN architecture. However, within the given input, the ANN model outperforms both machine learning approaches on both single and multi-agent systems. The experimental results and evaluation of those results encourage authors to design a SAE- or CNN-based model using raw images as future works. Those models will only need raw image data to train models, which will probably outperform both RL- and ANN-based models but may need larger amount of training data, and also require excessive training time to form a suitable model. Despite the given explanation, it is not clear to estimate the performance of those algorithms on different multi-agent systems except Robocode without implementing and evaluating their overall performance.

Author details

Ümit Ulusoy, Mehmet Serdar Güzel* and Erkan Bostanci
Department of Computer Engineering, Ankara University, Ankara, Turkey

*Address all correspondence to: mguzel@ankara.edu.tr

IntechOpen

References

[1] Şahin E. Swarm robotics: From sources of inspiration to domains of application. In: Swarm Robotics Workshop: State-of-the-Art Survey; Şahin E, Spears W, editors, Lecture Notes in Computer Science; no. 3342; Berlin, Germany; 2005. pp. 10-20

[2] Brambilla M, Ferrante E, Birattari M, Dorigo M. Swarm robotics: A review from the swarm engineering. Swarm Intelligence. 2013;**7**(1):1-41

[3] Guzel MS et al. A novel framework for multi-agent systems using a decentralized strategy. Robotica. 2019;**37**(4):691-707

[4] Rodriguez-Angeles A, Vazquez CL. Bio-inspired decentralized autonomous robot mobile navigation control for multi agent systems. Kybernetika. 2018;**54**(1):135-154

[5] Jiménez AC, García-Díaz V, Bolaños S. A decentralized framework for multi-agent robotic systems. Sensors. 2018;**18**(2):417

[6] Dorri A, Kanhere S, Jurdak R. Multi-agent systems: A survey. IEEE Access. 2018;**6**:28573-28593. DOI: 10.1109/ACCESS.2018.2831228

[7] Jin-Hyuk H, Sung-Bae C. Evolution of emergent behaviors for shooting game characters in Robocode. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753). Vol. 1; Portland, OR, USA. 2004. pp. 634-638

[8] Sanjay MS, Chirag ST, Dharm S. Multimedia based fitness function optimization through evolutionary game learning. In: 2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC). 2011. pp. 164-168

[9] Nattharith P, Güzel MS. Machine vision and fuzzy logic-based navigation control of a goal-oriented mobile robot. Adaptive Behavior. 2016;**24**(3):168-180

[10] Harper R. Evolving Robocode tanks for Evo Robocode. Genetic Programming and Evolvable Machines. 2014;**16**(4):403-431. DOI: 10.1007/s10710-014-9224-2

[11] Deep Q-Learning for Robocode, Baptiste DEGRYSE [thesis]. Ecole polytechnique de Louvain; 2017

[12] Gade M, Knudsen M, et al. Applying machine learning to Robocode. Technical Report. 2003

[13] Shichel Y, Ziserman E, Sipper M. GP-Robocode: Using genetic programming to evolve robocode players. In: Proceedings of 8th European Conference on Genetic Programming; Lecture Notes in Computer Science. Vol. 3447. Germany: Heidelberg: Springer-Verlag; 2005. pp. 143-154

[14] Abdellatif AJ, McCollum B, McMullan P. Serious games quality characteristics evaluation: The case study of optimizing Robocode. In: 2018 International Symposium on Computers in Education (SIIE); Jerez; 2018. pp. 1-4

[15] Alaiba V, Rotaru A. Agent architecture for building Robocode players with SWI-Prolog. In: 2008 International Multiconference on Computer Science and Information Technology; Wisia; 2008. pp. 3-7

[16] Safak AB, Bostanci E, Soylucicek AE. Automated maze generation for Ms. Pac-Man using genetic algorithms. International Journal of Machine Learning and Computing. 2016;**6**(4):226-240

[17] Sharma A, Gupta K, Kumar A, Sharma A, Kumar R. Model based path

planning using Q-Learning. In: 2017
IEEE International Conference on
Industrial Technology (ICIT); Toronto,
ON. 2017. pp. 837-842. DOI: 10.1109/
ICIT.2017.7915468

[18] Karim AM, Güzel MS, Tolun MR,
Kaya H, Çelebi FV. A new generalized
deep learning framework combining
sparse autoencoder and Taguchi method
for novel data classification and
processing. Mathematical Problems in
Engineering. 2018:1-13