Noname manuscript No.
(will be inserted by the editor)

A Learning Invader for the "Guarding a Territory" Game

A Reinforcement Learning Problem

Hashem Raslan · Howard Schwartz · Sidney Givigi

Received: date / Accepted: date

Abstract This paper explores the use of a learning algorithm in the "guarding a territory" game. The game occurs in continuous time, where a single learning invader tries to get as close as possible to a territory before being captured by a guard. Previous research has approached the problem by letting only the guard learn. We will examine the other possibility of the game, in which only the invader is going to learn. Furthermore, in our case the guard is superior (faster) to the invader. We will also consider using models with non-holonomic constraints. A control system is designed and optimized for the invader to play the game and reach Nash Equilibrium. The paper shows how the learning system is able to adapt itself. The system's performance is evaluated through different simulations and compared to the Nash Equilibrium. Experiments with real robots were conducted and verified our simulations in a real-life environment. Our results show that our learning invader behaved rationally in different circumstances.

Keywords Reinforcement Learning · Machine Intelligence · Adaptive Control · Continuous Time · non-holonomic · Fuzzy Q-Learning · Nash Equilibrium

1 Introduction

"Guarding a territory" is a game with two sets of players; Guards and Invaders. The guards try to protect a certain space, the territory, which the invaders are trying to reach. This game was first described by Isaacs[6]. Different applications for this game can be found in crucial real-life scenarios, such as protecting borders from smuggling or breaching a secure space. Some papers have been published that investigated "guarding the territory" game [11][8]; however, they mostly focused on the use of fixed algorithms. We investigate the use of a learning algorithm that can easily adapt itself to different strategies taken by the opponent.

The examined game occurs in a bounded space. It consists of a single invader and a single guard. The game runs in the continuous time domain. The invader's goal is to reach the territory, or at least get as close as possible to it. The guard's goal is to intercept the invader as far as possible from the territory.

All our developed efforts focus on constructing an adaptive learning invader, which can learn to perform optimally against the guard's different strategies. Different from previous work, our work uses a model with non-holonomic constraints. Also, we use a superior guard to play against our adapting invader.

If full knowledge of the environment (including the dynamic and kinematic model of the guard and evader) and the strategy of the guard is available, the optimal solution of the game from the invader's perspective can be achieved. In this paper we relaxed these assumptions. Therefore, this paper investigates the use of machine learning (namely a Reinforcement Learning) algorithm that can easily adapt itself and perform optimally to different strategies taken by the opponent. An invading system that can easily adapt itself to the current environment has an important use as it can be used as a testing mechanism to evaluate how successful and strong is a guarding system. Once the invader

H. Raslan
E-mail: HashemRaslan@sce.carleton.ca

H. Schwartz
E-mail: Schwartz@sce.carleton.ca

S. Givigi
E-mail: Sidney.Givigi@rmc.ca

learns the guard's weakness, the guard's weakness can be exploited.

The main contributions of this paper are: *(i)* the modeling of the "guarding a territory" game in terms of reinforcement learning; *(ii)* the proposal of reward functions that lead to the evaluation of the performance of the game; *(iii)* the proposal of an algorithm that solves the game so it achieves the Nash equilibrium; and *(iv)* experiments that demonstrate that the algorithm can be ported to real platforms and learn the optimal strategies on the fly.

This paper will provide a brief introduction to Reinforcement Learning in Sec. 2, then it will define the game problem with more details in Sec. 3. Sec. 4 will describe the control system developed. We will compare the simulation results from the learning (adaptive) control system to other fixed algorithms in Sec. 6 and show our experimental results in Sec. 7.

2 Reinforcement Learning

There are two types of learning in machine intelligence; supervised and unsupervised. Supervised learning depends on previous knowledge of the environment and action space. Supervised learning always require to know the correct answer. This means it has to know the environment prior to interaction. In this condition, Reinforcement Learning (RL), a type of unsupervised learning, is more useful.

Actions create events in the environment. For one to perform a certain task (goal), one has to take a set of actions. RL helps the learning agent to perform its goals throughout the agent's interactions with the environment, by focusing on increasing the agent's reward. The reward describes how well the agent is performing to reach its goals. The RL algorithm has to create a balance between exploration and exploitation; meaning the learning agent has to remember how successful were the actions previously taken and repeat them (exploitation). Also, the agent should occasionally take random actions (explorations). This is done to ensure trying different action scenarios and not getting stuck in a local-minima/maxima[12].

In the literature, RL has been used in many gaming scenarios to allow robots to learn a strategy [4]. The pursuer-evader game is one of the most common of these games, wherein a pursuer tries to capture an evader [6]. This game has been investigated in the context of RL where an algorithm based on hierarchical reinforcement learning and its learning efficiency is used [9], enabling the players to learn and "significantly reduce the complexity of the learning task". Givigi and Schwartz [5] explored a multiple pursuer-evader game. They represented the game as a Markov game and enabled each player to have its own decentralized learning, showing how their agents were able to learn and reached equilibrium points.

Researchers have also combined RL with fuzzy controllers to deal with large space and noisy environments. Schwartz and Desouky [2] proposed a technique called Q-learning based genetic fuzzy controller, where a fuzzy controller is used in order to reduce the number of states in a large continuous state space. The authors show how their players, in the pursuer-evader game, are able to converge their learning using this technique.

Other games and applications using RL have also been proposed as, for example, the patrolling of an area [7]. In this scenario, multiple agents learn an MDP using Q-Learning to patrol their environment. The method is shown to be robust to failures as well as efficient based on the number of agents used.

In RL, the agent takes a certain action (or a set of actions) then observes its new state from taking these actions. The rewards act as a feedback for the actions taken. They help the agent in knowing how useful were the actions taken. In our case, the learning agent is the invader and the environment is the "guarding a territory" game. In other words, the guard is part of the invader's environment.

Since the current game runs in continuous time with a bounded space, it is possible to discretize the entire game space. However, this will require a very large space of memory to be implemented on a simple device. The same goes for the action space. Moreover, this discretization will cause slow learning, since the agent will have a large amount of tables to fill and update. Discretization will also be computationally expensive, and hence discretizing the spaces is less favorable. Therefore, the use of a fuzzy logic controller is proposed to estimate the current system state. A fuzzy logic estimator with a reinforcement learning algorithm is used to produce our learning agent. Other than estimating the state, the fuzzy logic controller helps in dealing with noisy signals.

3 Problem Definition

Given a game with a single guard (G) and single invader (I) working around territory (T), the game terminates when the invader reaches the territory or the invader gets captured by the guard. For the sake of simplicity, the capturing action happens when the invader is at a specified proximity from the guard defined as $dist_{capture}$. The *payoff* (result) of the game can be determined as the distance between the invader and the territory’s centre at game termination time:

$$payoff = \sqrt{(x_I(t_f) - x_T)^2 + (y_I(t_f) - y_T)^2} \tag{1}$$

Where $x_I(t_f)$ and $y_I(t_f)$ represent the invader’s coordinates at terminal time. x_T and y_T represent the territory’s centre coordinates. This makes the goal for the invader is to minimize the *payoff* and the goal of the guard is to maximize the *payoff*.

Working in continuous time, any player j (invader or guard) will have the following equations of motion:

$$\begin{bmatrix} \dot{x}_j \\ \dot{y}_j \\ \dot{\varphi}_j \end{bmatrix} = \begin{bmatrix} \cos\varphi_j \\ \sin\varphi_j \\ 0 \end{bmatrix} v_j + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega_j \tag{2}$$

$$\text{such that} \begin{cases} x : x-\text{coordinates}, \\ y : y-\text{coordinates}, \\ \varphi : \text{robot orientation}, \\ v : \text{driving velocity}, \textit{and} \\ \omega : \text{angular speed} \end{cases}$$

These equations of motions describe the kinematics of a cart robot with non-holonomic constraints [13]. Both the guards and the invaders have to stay within the borders of the game-space. Hitting the border will make the player stay in the same position. This constraint is enforced such that the game will quickly come to completion. Both players will have the same action state for angular speed (i.e. $\omega_j = -0.5, 0, \text{or } 0.5$), but the linear speed of the guard is 20 percent faster than that of the invader.

Fig. 1 displays the game layout. Only the invader’s kinematics are labelled. Please note that the dimensions of the robot does not depict what is used for simulation, but it is only for illustrative purposes. The term L is the length of the game’s bounded space ($L = 30$ *unit space*).

3.1 Guard’s Optimal Strategy

Different guarding strategies were considered to play against the learning invader. The used guarding strategy will be hard coded into our guard. There will be no learning followed by the guard.

On the assumption that the invader can be captured, [6] describes the guard’s optimal policy as follows: if both the guard and the invader have equal kinematics, the perpendicular bisector line between the two players (at each time step) will indicate the region that the invader can reach before being captured by the guard. This is the region on the invader’s side of the line. Once the guard knows this line, it should move towards the point closest to the territory that lies on this line. This point is called closest interception point (closest to the territory). The guard has to calculate this point at each time step. The only optimal strategy against the guard’s optimal strategy is that the invader has to also aim at the closest interception point. This will allow the invader to get as close as possible to the territory, and hence achieve the minimum payoff possible. This is referred to as the Nash Equilibrium for both players.

Since in the investigated case the guard is 20 percent faster than the invader, the line that describes the region that can be reached by the invader is no longer the perpendicular bisector. The next steps illustrate the region that can be reached by the invader before being intercepted by the guard.

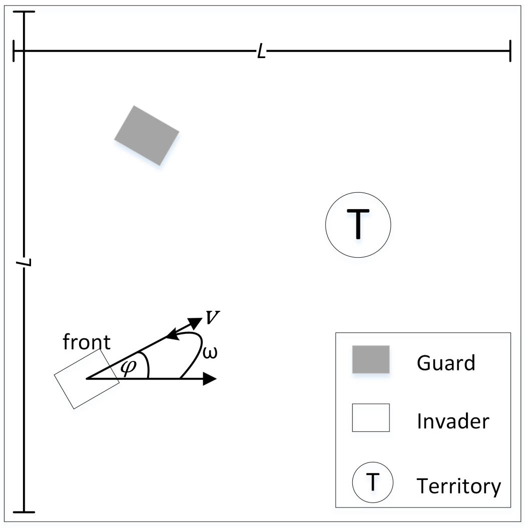


Fig. 1 Game Environment

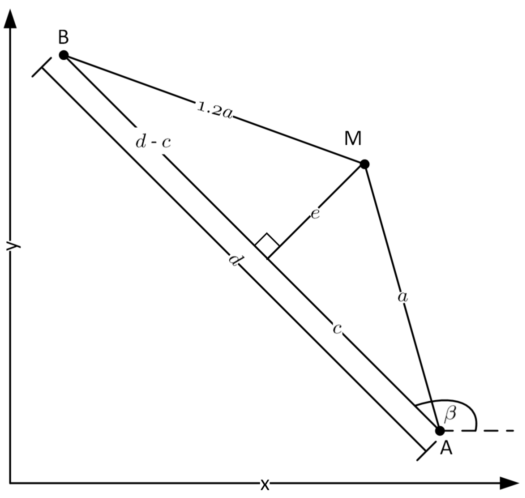


Fig. 2 Reachable region geometry

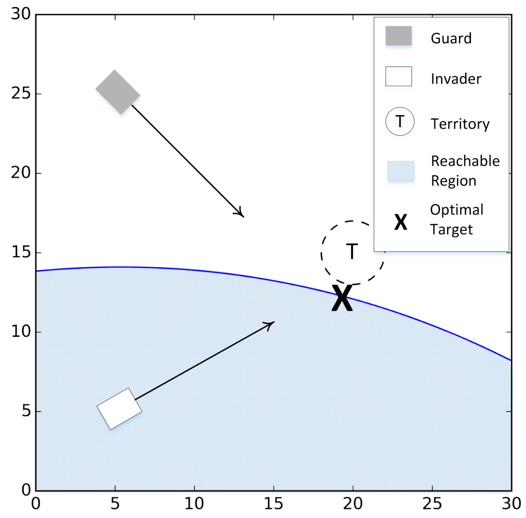


Fig. 3 Guard's Optimal Strategy - Nash Equilibrium

According to Fig. 2, assuming that the invader is at point 'A' and the guard is at point 'B' (separated by distance d) and that the speed of the guard is 20 percent faster than that of the invader, the invader can reach any point 'M' before being captured by the guard; such that the distance between 'A' and 'M' is 'a' and the distance between 'B' and 'M' is '$1.2a$'. The set of points that can satisfy the condition for 'M' are calculated through the following:

$$e^2 + c^2 = a^2$$
$$e^2 + (d - c)^2 = (1.2a)^2$$
$$0.44c^2 + 2dc + (0.44e^2 - d^2) = 0$$

For $e \in \mathbb{R}$, c can be found as:

$$c = \frac{-(2d)^2 \pm \sqrt{2d - 4 \times 0.44 \times (0.44e^2 - d^2)}}{2 \times 0.44} \tag{3}$$

Once the set of points is deduced and on the assumption that capturing is possible, the guard's optimal policy will be to target the closest interception point. If the invader targets the same point, then the invader is following the Nash Equilibrium strategy for an optimal guard. Fig. 3 illustrates the strategy.

3.2 Irrational Guard Strategy

Another guard algorithm used in our simulations is targeting the invader's current position. In the optimal strategy, we assume that the guard has full knowledge of the state of the environment. That is not always possible, hence, the irrational strategy is in several cases the best the guard can do. For example, if the guard has to observe the invader with a sensor, it may not know its orientation and the speed it is at.

This strategy performs worse than the optimal strategy, because it does not try to intercept the invader. We call this strategy the irrational guard approach.

In this strategy, the guard will ignore the position of the territory. It will aim at decreasing the distance between the guard and the invader. The invader's or guard's heading is also not considered. Thus, the guard will not be performing optimally and this will give a chance for the invader to reach the territory in some cases.

If it is impossible for the invader to reach the territory, a rational invader should get as close as possible to the territory (maximum reward). However, if it is possible to reach, then a rational invader should reach the territory.

4 System Configuration

As described previously, the system will combine a fuzzy logic controller with reinforcement learning to create our learning agent. This learning agent is called the Fuzzy Q-Learning algorithm (FQL)[12]. To build our fuzzy controller, rules-based fuzzy sets with constant consequent as described in [14] are used. It creates a basic fuzzy system configuration with If-Else rules [12][10]. The fuzzy logic controller was built to choose the invader's heading (steering angle) based on the game's current state. We aimed at creating a generic system that uses 3 different inputs as will be described in subsection 4.1.

4.1 The fuzzy logic controller

The first input was chosen to describe the territory position with respect to the invader. The angle (θ) between the global x-axis and the invader's line of sight towards the territory was used, as shown in Fig. 4, such that $\theta \in [0, 2\pi]$. The input θ is then divided into 9 fuzzy sets as follows:

- ZE: θ is near 0 (zero)
- SS: θ is near $\pi/4$ (small small)
- SM: θ is near $\pi/2$ (small medium)
- SL: θ is near $3\pi/4$ (small large)
- MS: θ is near π (medium small)
- MM: θ is near $5\pi/4$ (medium medium)
- ML: θ is near $6\pi/4$ (medium large)
- LS: θ is near $7\pi/4$ (large small)
- LM: θ is near 2π (large medium)

The second and third inputs were used to relate the current guard's position with respect to the invader. Those inputs were selected to be the Manhattan

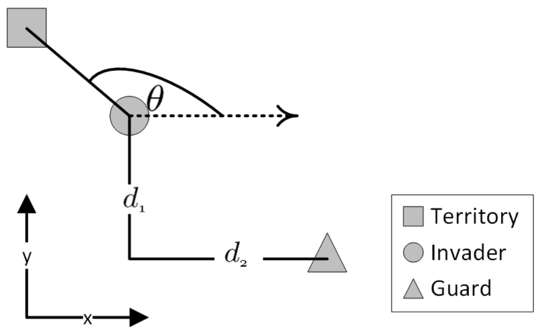


Fig. 4 Fuzzy Inputs: angle θ, d_1 and d_2

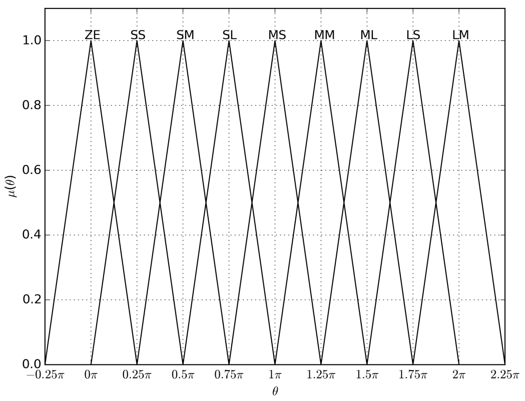


Fig. 5 Membership function for the angle input

distance between the invader and the guard positions. They are represented by two components d_1 and d_2 as shown in Fig. 4. Such that $d_1, d_2 \in [-L, L]$. Manhattan distance was used instead of the Euclidean distance because it contains more information about the relative position between the guard and the invader positions. Also, simulations showed a faster learning using Manhattan distance rather than using both Euclidean distance and the angle between the invader and guard.

For both d_1 and d_2, the following 5 fuzzy sets are created ($i \in 1, 2$):

– LN: d_i near -20 (large negative)
– SN: d_i near -10 (small negative)
– ZE: d_i near 0 (zero)
– SP: d_i near 10 (small positive)
– LP: d_i near 20 (large positive)

The membership degree function between the rules was chosen as a triangular function with no adaptation for all the inputs. Fig. 5 and Fig. 6 display the membership functions for the angle θ and the Manhattan distance inputs. Each of the fuzzy sets describe our *fuzzy-rules*.

In summary, using the previously mentioned inputs creates 225 different combinations, which are 225 rules for our fuzzy system. We also tested using other control

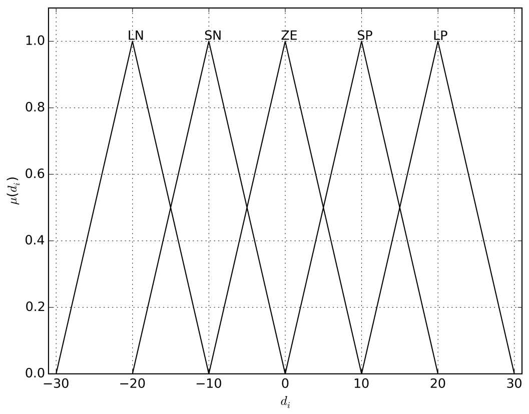


Fig. 6 Membership function for Manhattan distance input

system inputs, such as using the Manhattan distance between the invader and the territory. However, this would have required 625 rules and would be computationally complex. Additional input signals were also tested, but the previously mentioned selection describes our final fuzzy logic controller inputs and rules.

4.2 Fuzzy Q-Learning (FQL)

Using the product inference for fuzzy implication, t-norm, singleton fuzzifier and center average defuzzifier [15][10], the output of the fuzzy system U_t for the current state $\bar{x}_t$ becomes at time t,

$$U_t(\bar{x}_t) = \sum_{l=1}^{N} \phi_t^l a_t^l \tag{4}$$

$$\phi^l = \frac{\prod_{i=1}^{n} \mu_i^l(x_i)}{\sum_{l=1}^{N}(\prod_{k=1}^{n} \mu_k^l(x_k))} \tag{5}$$

Where μ is the membership function, x_i is the ith input; such that $\bar{x} = [x_1, x_2, x_3] = [\theta, d_1, d_2]$, n is the number of inputs ($n = 3$) and N is the number of rules ($N = 225$). The term a_t^l is the constant describing the centre of the fuzzy set for each rule, which in our case is the action selected at time t for rule l based on the Q-table (will be described afterwards)[12]. Also, action a belongs to an action set A. The action was selected to describe different possible steering angles. The action set was chosen as:

$$A = \{-3\pi/4, -\pi/2, -\pi/4, 0, \pi/4, \pi/2, 3\pi/4, \pi\}$$

After taking any action, the action is evaluated. Based on our evaluation a value is added to the Q-table. The Q-table contains a value for each action in the action set for each of the rules. For each rule, the action with a higher value is more favourable; however, a random selection factor was added to create exploration and exploitation. The values for each of the actions are updated and adapted based on the learning algorithm. The value for the current state is represented by capital Q, and the Q-table is represented by lower-case q. One can compute Q for the current state as,

$$Q(\bar{x}_t) = \sum_{l=1}^{N} \phi_t^l q_t(l, a_t^l) \tag{6}$$

The maximum possible value is represented as Q^* and computed as,

$$Q^*(\bar{x}_t) = \sum_{l=1}^{N} \phi_t^l \max_{a^l \in A} q_t(l, a_t^l) \tag{7}$$

Lastly, one computes the future temporal difference as [12],

$$E_{t+1} = r_{t+1} + \gamma Q^*(\bar{x}_{t+1}) - Q(\bar{x}_t) \tag{8}$$

Where γ is the forgetting factor and it focuses on the expected future rewards and r_{t+1} is the reward received by the agent after doing an action.

After calculating the temporal difference, the learning agent is ready to adapt its Q-table for each of the fuzzy rules $l \in$ *fuzzy rules*. The Q-table is adapted according to:

$$q_{t+1}(l, a) = q_t(l, a) + \alpha E_{t+1} \phi_t^l \tag{9}$$

Where α is the learning rate.

Exploration and exploitation is done by selecting a random action from action set A with probability ϵ (exploration rate). We call the forgetting factor, learning rate, and exploration rate as Learning Factors. The FQL algorithm is shown in Algorithm 1. This FQL was implemented in [3][1].

Algorithm 1 FQL algorithm

```
1: $q(l, a) \leftarrow 0,\ \forall a \in A \ and \ \forall l \in rules$
2: for each time step $t$ do
3:     For each rule, choose action $a_t^l$ based on:
4:     $\begin{cases} a_t^l = \underset{a \in A}{\text{argmax}}\ q^l(l, a), \text{with probability}(1-\epsilon) \\ \text{random action from } A, \text{ with probility } \epsilon \end{cases}$
5:     Calculate $\phi^l$ for each rule based on (5)
6:     Estimate current state value $Q(\bar{x}_t)$ using (6)
7:     Calculate system output for current state $U_t(\bar{x}_t)$ using (4)
8:     Take action $U_t$ (update the robot's kinematics using (2))
9:     Observe new state
10:    Obtain reward
11:    Calculate maximum possible future value $Q^*(\bar{x}_{t+1})$ based on (7)
12:    Calculate temporal difference $E_{t+1}$ using (8)
13:    Adapt the Q-table using (9)
14: end for
```

5 Reward Function Selection

Reinforcement Learning can have two different types of rewards. They are terminal rewards or instantaneous rewards. In our game, a terminal reward will be given only if the invader reaches the territory. However, this will make it difficult for the invader to learn when it is impossible to reach the territory. Also, it will cause slow learning. That is because the invader will have to take many actions before it can reach the territory. Thus, it will not know if it is doing the correct action for a long time. On the other hand, an instantaneous reward will always inform the agent how well the agent is doing at each time-step.

We propose using an instantaneous reward function. It is difficult to select the reward function. To construct our reward function, we divide our game into two games.

The first game is a single invader with no guards. The invader's goal is to reach the territory. In that case we can describe our instantaneous reward by how much the agent got closer to the territory. That is:

$$r_{t+1}^{p1} = dist_{IT}(t) - dist_{IT}(t+1) = \delta_{IT} \tag{10}$$

where $dist_{IT}(t)$ is the euclidean distance between the invader and the territory at time t. Thus, this constructs the instantaneous reward for the first part of the game.

We describe the second game as a pursuer evader game. Such that the invader is trying to run away from the guard and no territory is involved. In other words, the invader is trying to increase the distance between itself and the guard. Hence, we can describe the reward in this part of the game as how far the agent went from the guard. That is:

$$r_{t+1}^{p2} = dist_{IG}(t+1) - dist_{IG}(t) = -\delta_{IG} \tag{11}$$

such that $dist_{IG}(t)$ is the euclidean distance between the guard and the invader at time t.

Once we have constructed these two rewards, we propose combining both rewards with ratios K and J to produce the agent's full instantaneous reward.

$$r_{t+1} = K\delta_{IT} - J\delta_{IG} \tag{12}$$

We define K as a constant that describes the importance of getting closer to the territory. The larger

K is, the faster the agent will go to the territory. Similarly, J describes the importance of travelling furthest from the guard. Since the guard is always faster than the invader, $dist_{IG}$ will never increase. However, the invader's actions will choose how slowly the distance (between the guard and the invader) decreases. Thus, J describes the importance of controlling the distance decrease rate between the guard and the invader. Since an agent in our simulations and experiments is actually a robot, from now on we only refer to robots.

6 Simulation Setup

In the simulations, the game is run for several different episodes, which we call epochs, therefore, a simulation consists of a set of epochs.

We define an epoch as a run of the game until terminal time, meaning that the invader is captured or it reaches the target. Capturing happens when the invader and the guard are 2 units apart ($dist_{capture} = 2$).

Furthermore, an epoch can be a training or a test epoch. In any test epoch, all the learning factors are set to zero. As such, no learning or exploration will take place. Finally, the territory is represented by a circle with a radius of 2 units.

The training and testing technique are realized in Algorithm 2 after the learning factors are initialized. Equation (2) was solved in the simulation using Euler integration.

Algorithm 2 Train-Test algorithm

```
1: $q(l, a) \leftarrow 0,\ \forall a \in A$ and $\forall l \in rules$
2: $j \leftarrow 1$
3: for $j \leq 400$ do
4:     Run 1 training epoch, with robot initial coordinates assigned at random
5:     Set learning factors to zero
6:     Run 1 test epoch, with robot initial coordinates $P_I$ & $P_G$ ($P_I$ & $P_G$ are fixed coordinates for the invader and guard)
7:     Update learning factors
8:     $j$++
9: end for
```

The territory was placed at coordinates $(20, 15)$. The training region was arranged for the guard and the invader to randomly choose a starting position (x_{G0}, y_{G0}) and (x_{I0}, y_{I0}); such that $x_{G0} \in [0, 10], y_{G0} \in [20, 30]$ shown as the dashed area in Fig. 7 and $x_{I0} \in [6, 10], y_{I0} \in [6, 15]$ shown as the solid area in same figure. The testing starting positions were chosen such that: $P_I = (5, 5)$ and $P_G = (5, 25)$ and φ_I and φ_G are initialized to zero radians. The sampling time was selected as $T = 10mS$.

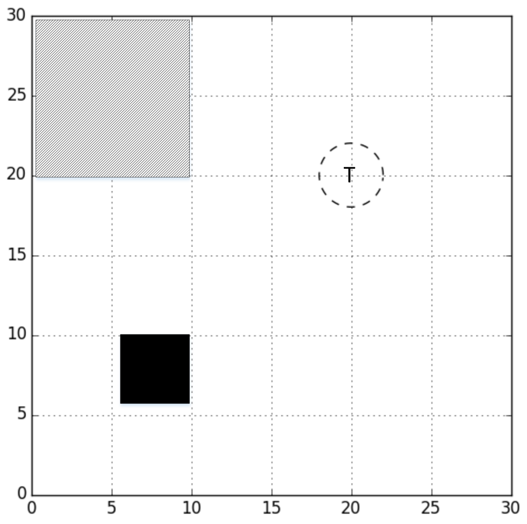


Fig. 7 Training region

The learning parameters were chosen to decay after every learning epoch. The decay was chosen as an exponential decay as follows (where j represents the epoch count):

$$\alpha = 0.99^j \alpha_0, \quad \gamma = 0.92^j \gamma_0, \text{and} \quad \epsilon = 0.95^j \epsilon_0 \qquad (13)$$

with $\alpha_0 = 0.1, \gamma_0 = 0.9$, and $\epsilon_0 = 0.3$. Even though this decay is fast, the robot will be able to update the same state multiple times within a single epoch due to the small sampling time and its non-holonomic constraints.

The instantaneous reward in (12) was used. The parameter K was set to a value of 2.0 and J was set to a value of 1.0. The K and J values were chosen based on simulation. Increasing the K value or decreasing the J value make the invader go in a straight line towards the territory, ignoring the guard. The results examined were based only on the test epochs results.

6.1 Optimal Guard Simulation Results

The first simulation was done using the learning invader against the optimal strategy guard (described in Sec. 3.1). This scenario demonstrates that the invader is able to reach the Nash Equilibrium (when possible) or to get as close as possible to it (when capture is inevitable). Fig. 8 shows the invader's and guard's trajectories for the first test epoch (in this case capture of the invader). Fig. 9 shows the guard's and invader's trajectories after the invader learns. Notice that even though the invader didn't reach the territory, it got the closest possible to the territory as desired.

Comparing Fig. 8 and Fig. 9, it can be seen that the invader was able to perform better and got closer to the territory, which implies that the learning procedure was

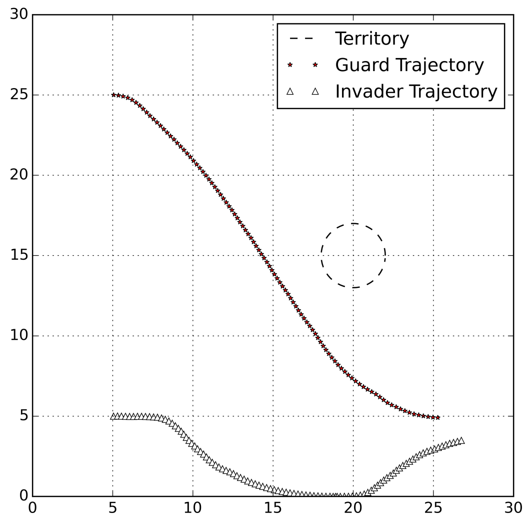


Fig. 8 Simulation: Learning Invader vs. Optimal Guard - first test epoch trajectories

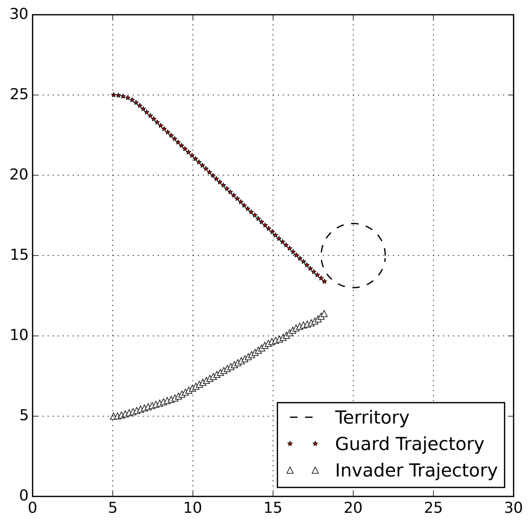


Fig. 9 Simulation: Learning Invader vs. Optimal Guard - final test epoch trajectories

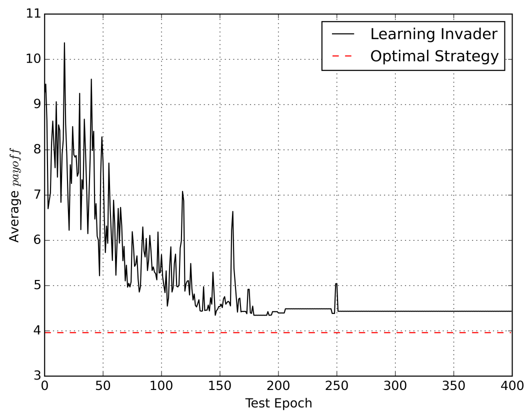


Fig. 10 Simulation: Invader's payoff against optimal guard

effective. Fig. 10 shows the average *payoff* of 10 simulations and one can observe that the learning robot's payoff converged to a specific *payoff* value, in this case 4.43. If the invader was following the Nash Equilibrium strategy, its payoff will be 3.96. The learning robot was able to get very close to the Nash Equilibrium target (≈ 0.5 units apart) and minimized the robot's *payoff*.

It should be emphasized that in the simulation shown in Fig. 9, the players' starting positions make it impossible for the invader to reach the territory (since the guard is doing the optimal strategy). Therefore, we further tested our invader by starting the game with positions that make it possible for the invader to reach the territory. As a consequence, our invader could only reach the territory if it is rational, i.e., it uses its optimal actions.

We set the invader's starting position to $(7, 7)$ and the guard's starting position to $(5, 25)$. We use the Q-table that produced the results in Fig. 9 in order to evaluate the learning. After running the test epoch, we found that the invader was able to reach the territory as seen in Fig. 11. The same simulation was repeated for the 10 different final Q-tables that produced Fig. 10 and the invader was always able to reach the territory for the different Q-tables.

These results imply the following. First, the invader adapted its parameters to maximize its future rewards. Second, maximizing our proposed reward function enables the invader to get the closest possible to the territory. Third, the control system designed was sufficient to describe different game states in continuous time. This demonstrates that the invader can effectively counteract the optimal strategy of the guard. However, one other question remains: what if the guard plays a sub-optimal strategy? Could the invader learn to take advantage of that situation? We pursue this question in the next section.

6.2 Irrational Guard Simulation Results

In our second simulation, the irrational guard algorithm, described in Sec. 3.2, is used instead of the optimal guard's strategy. We initialize the invader's Q-table to all zeros and start adapting. Fig. 12 shows the test results after the first test epoch.

In comparison, Fig. 13 demonstrates how the invader performed after learning (after 400 training epochs). The invader learned to decrease the distance to the territory, but not target it. Thus, forcing the guard to move towards the invader and to leave enough space for the invader to maneuver into the territory. Then once it is clear to go to the territory, the invader makes a sharp

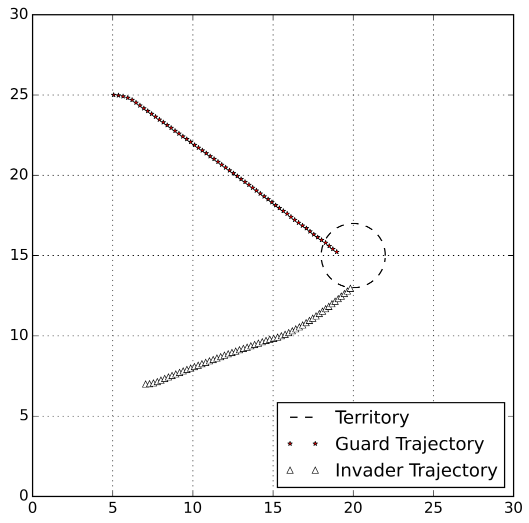


Fig. 11 Learning Invader vs. Optimal Guard - possible reaching test

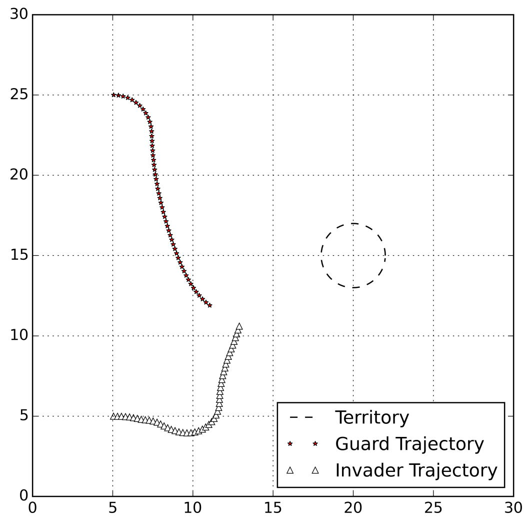


Fig. 12 irrational guard - first test epoch results

turn and goes directly to the territory. The same simulation was run 10 times and the simulations' *payoff* was averaged and plotted in Fig. 14.

It can be clearly seen that the robot converged to reaching the territory. To be able to compare our learning robot against the irrational guard strategy, the irrational guard strategy was run against the invader's Nash Equilibrium strategy for an optimal guard to create a benchmark as shown in Fig. 14. This shows that the invader was able to adapt and outperform the Nash Equilibrium strategy. The invader learned to take advantage of the irrational guard's poor strategy.

Again, the simulation results show that maximizing our proposed reward function enables the invader to get the closest possible to the territory (in this case, reaching it). In summary, the simulations described in sec-

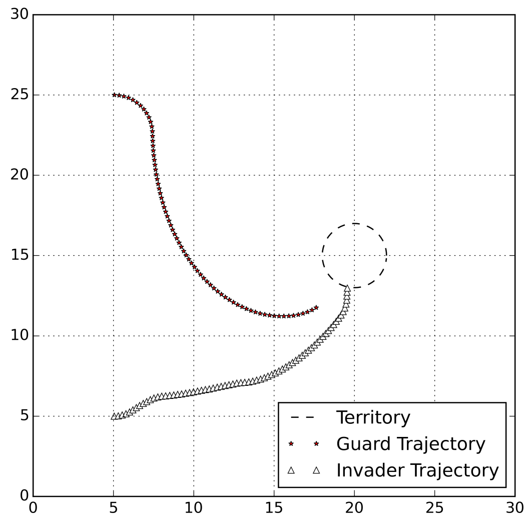


Fig. 13 Simulation: Learning Invader vs. Irrational Guard - final test epoch trajectories

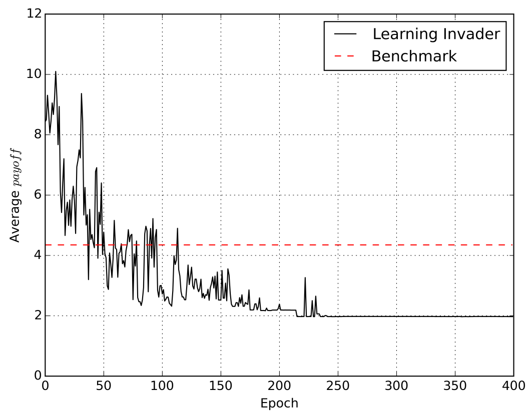


Fig. 14 Simulation: Invader's payoff against irrational guard

tions Sec. 6.1 and Sec. 6.2 match our expectations from literature. If the guard is employing the Nash Equilibrium strategy, the invader will adapt to the Nash Equilibrium strategy to maximize its rewards. If the guard is not employing the Nash Equilibrium, the invader should adapt to a new strategy to maximize its rewards.

7 Experimental Setup

Laboratory experiments were conducted at the Royal Military College of Canada to demonstrate how our learning algorithm performs in actual physical environments with noise and variations in models. CLEARPATH HUSKY robots were used as our players. These are 4 wheeled mobile robots with differential drives. The robots were programmed and configured using the *Robot Operating System* (ROS) and the Python scripting language [16]. Each robot ran its software independently in

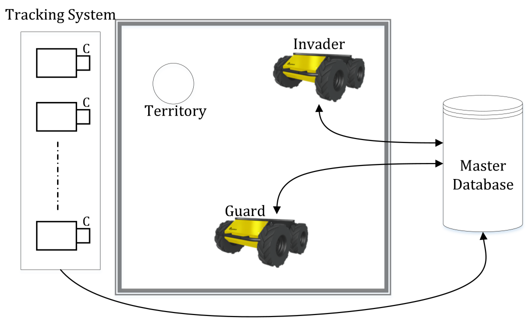


Fig. 15 Experiment Setup

a decentralized fashion. The game area was tracked using an OptiTrack system consisting of 24 synchronized infrared cameras. The tracking system sends the robot positions at a frequency $\approx 100Hz$ to a central database. The robots can request that tracking information. This mimics a GPS system. The experiment environment is illustrated in Fig. 15, the arrows display the possible flow of information.

Our experiments mapped the simulation size $30unit \times 30unit$ to $8.3meter \times 8.3meter$. Players' kinematics were set such that:

$$\text{Invader}: \; v_I = 0.35 meter/sec \tag{14}$$

$$\text{Guard}: \; v_G = 0.42 meter/sec \tag{15}$$

$$|\omega_j| = -0.5, 0, \text{ or } 0.5 rad/sec \tag{16}$$

The model that we used in (2) is not the same as the robot. However, it is very similar.

7.1 Experimental Evaluation of Simulation Results

In this section, we evaluate the invader's performance using the Q-table taken from the simulations. In other words, we are going to test our numerical results (from the simulations) in a real-life scenario. This is done to see how our simulation model compares to the laboratory environment. The invader uses the learned parameters from the simulations (the Q-table). There is no learning while the experiment is running. We use the Q-table from the final simulation epochs discussed in Sec. 6.

7.1.1 Learning Invader vs. Optimal Guard

In this experiment we use the Q-table used in the simulation shown in Fig. 10. The first experiment trajectory results can be seen in Fig. 16. The same experiment was run 10 times. The invader was able to produce an average *payoff* of 4.7 with 0.5 standard deviation. These results are very similar to the results discussed

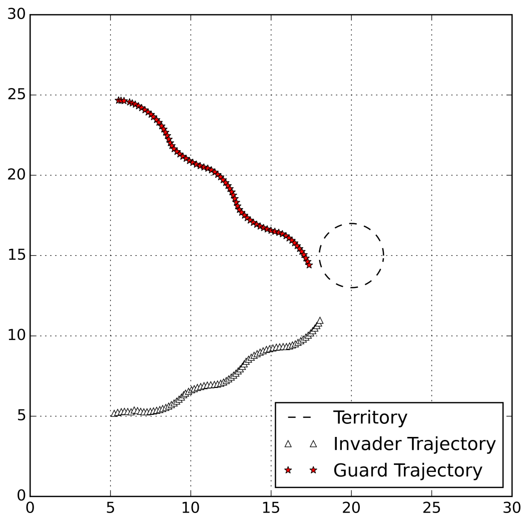


Fig. 16 Simulations' Experimental Evaluation: Invader vs. Optimal Guard

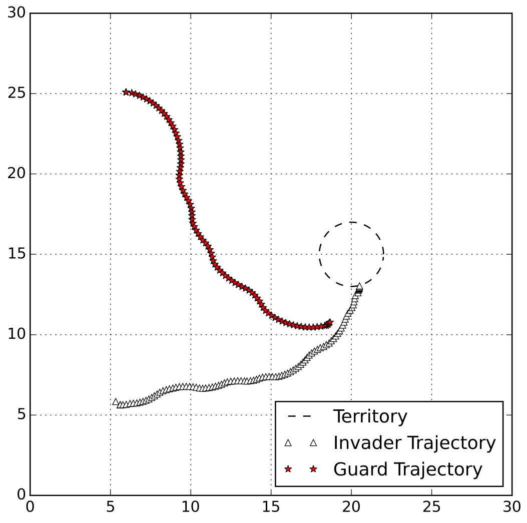


Fig. 17 Simulations' Experimental Evaluation: Invader vs. Irrational Guard

in Sec. 6.1 and demonstrate that the learning is robust to variations in the model (as the simulation model differs from the actual behaviour of the robot).

7.1.2 Learning Invader vs. Irrational Guard

Again, in this experiment we use the Q-table used in the simulation described in Fig. 13 in order to evaluate the quality of the learning. Trajectory results can be seen in Fig. 17. The same experiment was run 10 times and the invader was able to produce a *payoff* less than 2 *units*. Hence, it was always able to reach the territory (similar to the simulations). Again, these results demonstrate the robustness of the proposed solution.

7.2 Online Learning Experiments

Further to the evaluation of porting the simulation solutions to real robots, we implemented online learning experiments in the HUSKY robots. This was done in order to evaluate the robot’s ability to learn through real-life interactions, a very important issue if the model of either the guard or invader changes over time as well as changes in the environment (for example if traction changes due to water on the ground).

In online learning, the learning robot adapts its Q-table as it directly interacts with the environment (as it plays the game). The Q-table is first initialized to zero, meaning that the robot has no preference for any specific action and all actions would be randomly selected. Since the experiments are run sequentially, the Q-table is kept from one experiment to the next and the learning is cumulative. We always let the invader start the experiment at coordinate $(5, 5)$ and the guard at $(5, 25)$.

Each experiment represents a game run until termination time. The *payoff* is recorded after each experiment. We run two different sets of experiments to demonstrate how the invader learns online versus the different guard strategies.

7.2.1 Learning Invader vs. Optimal Guard

After programming the guard to follow the optimal strategy, we ran 100 experiments. As mentioned before, the invader started with a Q-table with all zeros and in each experiment thereafter, the invader used the parameters (Q-table) learned from the previous experiment. This is done to demonstrate the invader’s ability to learn with real interactions with the environment and adapt its Q-table.

Fig. 18 displays the players trajectories after learning. The payoff of each experiment was recorded and plotted in Fig. 19. This figure also demonstrates how our invader’s strategy progressed and minimized the robot’s *payoff*. In the last 20 experiments in online learning, the robot produced an average *payoff* of 3.82 with standard deviation of 0.23. This is better than the output from experiments that used the Q-table from the simulations illustrated in Fig. 16. This demonstrates that, in this case, the robot adapted to the physical realities of the experimental environment in a relatively low number of interactions.

7.2.2 Learning Invader vs. Irrational Guard

After programming the guard to follow the irrational strategy, we ran 50 experiments with the same conditions of the experiment described in section 7.2.1, i.e., the invader started with its Q-table initialized with zeros and the invader adapts the Q-table as it plays the game. This is done to demonstrate that the invader is able to learn directly from its interaction with a real environment and generate the Q-table again.

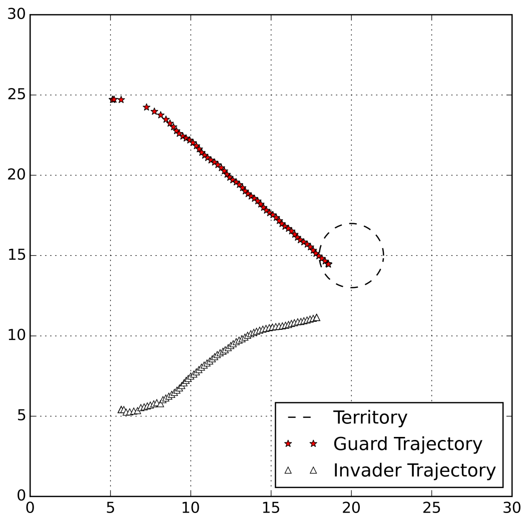


Fig. 18 Online Learning: Learning Invader vs. Optimal Guard - After Learning

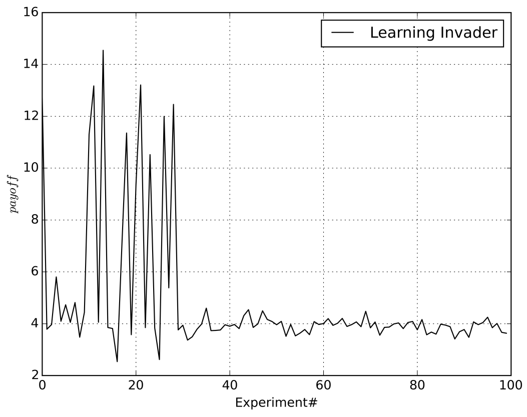


Fig. 19 Online Learning: Learning Invader vs. Optimal Guard - payoff

Fig. 20 displays the players’ trajectories after learning took place. Also, the payoff of each experiment was recorded and plotted in Fig. 21. We stopped at 50 experiments since the invader was able to reach the territory more than 10 times in a row. Fig. 21 demonstrates how our invader’s *payoff* converged into reaching the territory. Furthermore, the robot once again adapted to the physical environment and reached the territory in a shorter time than in Fig. 17.

These results and those of Sec. 7.2.1 are a good indication that online learning with the model and algorithm discussed in this paper is possible. This opens a new venue of investigation for the “guarding a terri-

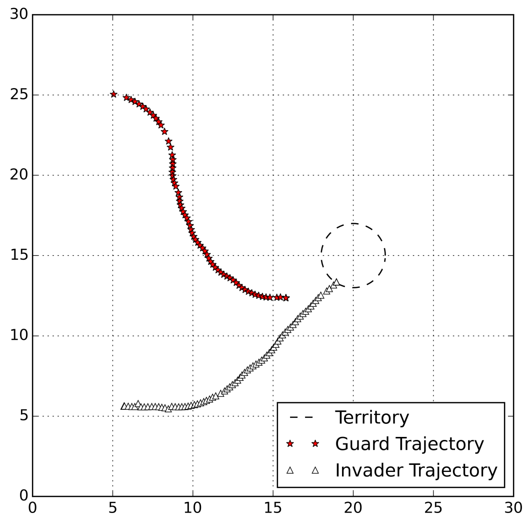


Fig. 20 Online Learning: Learning Invader vs. Irrational Guard - After Learning

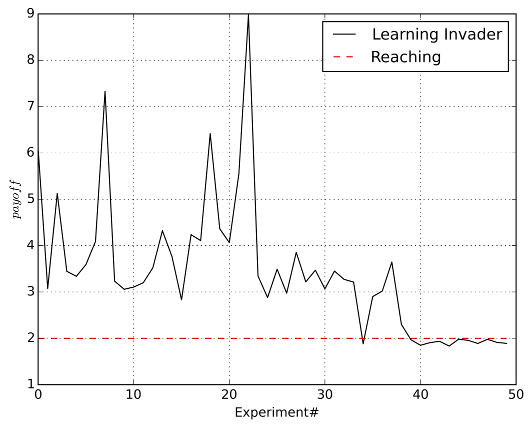


Fig. 21 Online Learning: Learning Invader vs. Irrational Guard - payoff

tory" game as one can move to fully experimental study of the machine learning algorithms.

8 Conclusion

A fuzzy logic controller was designed to estimate the invader's current state in the game. The controller inputs relate the invader's position to both the territory and the guard. The learning robot was then created by combining both the fuzzy logic controller with Q-Learning, which is called FQL. Different simulations were run to demonstrate the FQL performance.

Simulations were run having the guard follow different strategies. Against a guard employing the optimal strategy, the invader was able to get very close to the Nash Equilibrium strategy. In the second simulation, the invader competes against an irrational guard strategy. Following a Nash Equilibrium strategy while the guard is not acting optimally would produce a smaller payoff, hence, the invader has an incentive to change its strategy to exploit a non-optimal strategy by the guard. It was seen from the second simulation that the robot was able to adapt to the new guard strategy. It also performed better than the invader's Nash Equilibrium strategy against an optimal guard. The robot's strategy converged into reaching the territory. This shows that our learning robot was able to adapt and adjust its strategy to perform best with the current guard's strategy.

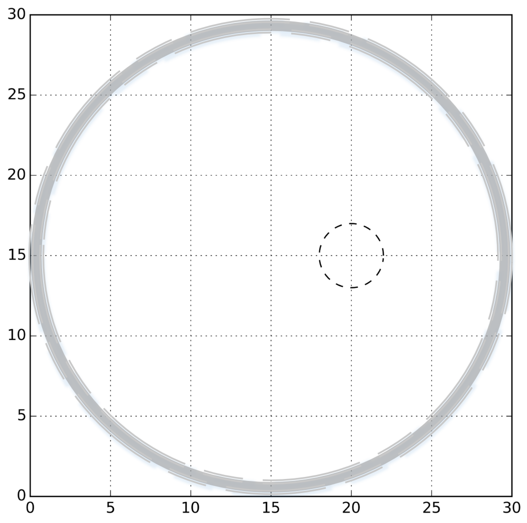


Fig. 22 Alternative Training Region

Our training method, focused on specific regions for both players. Other training methods can also be considered to optimize the robot's performance in any position in the game. This can be done by using a different training region as described in Fig. 22, such that the shaded circle describes the pool for the initial training positions for both players. The players can pick a random position from that region. However, training will take a longer time in this case and the learning rates need to be adjusted differently. The parameters will have to then decay slower to ensure learning in different configurations.

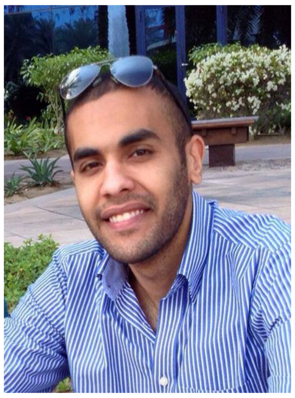
Furthermore, we ran experiments against the different guard strategies. This was done to verify that our simulation model was close to reality. Our simulations' experimental evaluation results were almost identical to our simulation results. Also, our online learning experimental results demonstrated our robot's ability to learn and perform well in a real-life environment. The results from online learning experiments were superior to the results from simulations' experimental evaluation results. The robot adapted to the physical realities of the experimental environment.

In conclusion, this paper examined the use of Reinforcement Learning in the “guarding a territory” game to adapt the invader. The invader was able to adapt itself against different opponent strategies and perform optimally and rationally.

In the future, we intend to implement this method for multiple invaders and guards. However, for this to be done, a different formulation of reward is necessary as well as a more efficient representation of the states of the game, as the number of possible states tend to grow exponentially with the number of players. However, if successful, such learning method would have applications in several different domains such as autonomous cars interactions and surveillance by autonomous robots.

References

1. Berenji, H.: Fuzzy q-learning: a new approach for fuzzy dynamic programming. In: Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on, pp. 486–491 vol.1 (1994). DOI 10.1109/FUZZY.1994.343737
2. Desouky, S., Schwartz, H.: A novel hybrid learning technique applied to a self-learning multi-robot system. In: Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, pp. 2616–2623 (2009). DOI 10.1109/ICSMC.2009.5346111
3. Er, M.J., San, L.: Automatic generation of fuzzy inference systems using incremental-topological-preserving-map-based fuzzy q-learning. In: Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence). IEEE International Conference on, pp. 467–474 (2008). DOI 10.1109/FUZZY.2008.4630410
4. Fang, M., Li, H., Zhang, X.: A heuristic reinforcement learning based on state backtracking method. In: Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on, vol. 1, pp. 673–678 (2012). DOI 10.1109/WI-IAT.2012.187
5. Givigi, S., Schwartz, H.M.: Decentralized strategy selection with learning automata for multiple pursuer-evader games. Adaptive Behavior **22**(4), 221–234 (2014). DOI 10.1177/1059712314526261. URL http://adb.sagepub.com/content/22/4/221.abstract
6. Isaacs, R.: Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization (1999)
7. Lauri, F., Koukam, A.: Robust multi-agent patrolling strategies using reinforcement learning. In: P. Siarry, L. Idoumghar, J. Lepagnot (eds.) Swarm Intelligence Based Optimization, *Lecture Notes in Computer Science*, vol. 8472, pp. 157–165. Springer International Publishing (2014)
8. Lee, Y.S., Hsia, K.H., Hsieh, J.G.: A problem of guarding a territory with two invaders and two defenders. In: Systems, Man, and Cybernetics, 1999. IEEE SMC ’99 Conference Proceedings. 1999 IEEE International Conference on, vol. 3, pp. 863–868 vol.3 (1999). DOI 10.1109/ICSMC.1999.823341
9. Liu, J., Liu, S., Wu, H., Zhang, Y.: A pursuit-evasion algorithm based on hierarchical reinforcement learning. In: Measuring Technology and Mechatronics Automation, 2009. ICMTMA ’09. International Conference on, vol. 2, pp. 482–486 (2009). DOI 10.1109/ICMTMA.2009.213
10. Nguyen, H.T., Walker, E.: A first course in fuzzy logic. Chapman and Hall, Boca Raton, FL (2006). URL www.summon.com
11. Rzymowski, W.: A problem of guarding line segment. In: Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, pp. 6444–6447 (2009). DOI 10.1109/CDC.2009.5400251
12. Schwartz, H.: Multi-Agent Machine Learning: A Reinforcement Approach. Wiley (2014)
13. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics Modelling, Planning and Control. Springer (2009)
14. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. Systems, Man and Cybernetics, IEEE Transactions on **SMC-15**(1), 116–132 (1985). DOI 10.1109/TSMC.1985.6313399
15. Wang, L.: A Course in Fuzzy Systems and Control. Prentice Hall PTR (1997)
16. Wang, S., Panzica, A., Padir, T.: Motion control for intelligent ground vehicles based on the selection of paths using fuzzy inference. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, pp. 1–6 (2013). DOI 10.1109/TePRA.2013.6556354



Hashem Raslanis a junior computer engineer. Hashem attained his BSc. degree in computer engineering from the American University in Dubai, UAE in April 2012. He then completed his MASc. degree in electrical and computer engineering at Carleton University, Canada in May 2015. Hashems main interests are artificial intelligence and robotics. Hashem Raslan lives in Ottawa, Canada.

Howard M. Schwartzreceived his B.Eng. degree from McGill University, Montreal, Canada, in June 1981 and his M.Sc. degree and Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, Ma, in 1982 and 1987, respectively. He is currently a Professor in the Department of Systems and Computer Engineering at Carleton University. His research interests include adaptive and intelligent con-

trol systems, robotics and process control, system modeling and system identification. His most recent research is in multi-agent learning with applications to teams of mobile robots with an emphasis on reinforcement learning.

Sidney N. Givigireceived a Ph.D. in Electrical and Computer Engineering from Carleton University, Ottawa, ON, Canada. He is now an Associate Professor with the Department of Electrical and Computer Engineering of the Royal Military College of Canada (RMCC). Sidney's research interests are mainly focused on autonomous systems and robotics with an emphasis on distributed control, machine learning and game theory.