Attraction-Based Reinforcement Learning: A Real-Time Approach Using Techniques Based on the Animal Behavior

Marcos A. M. Laia, Edimilson B. Santos, Wesley S. Guimarães
Department of Computer Science
Federal University of São João del Rei (UFSJ)
São João del Rei, Minas Gerais - Brazil
e-mail: marcoslaia@ufsj.edu.br,
edimilson.santos@ufsj.edu.br

Márcio Mendonça, Rodrigo H. C. Palácios, Janaína F. S. Gonçalves
PPG of Mechanical Engineering - CP/PG
Technological Federal University of Paraná (UTFPR)
Cornélio Procópio, Paraná - Brazil
e-mail: mendonca@utfpr.edu.br,
rodrigopalacios@utfpr.edu.br, janainaf@utfpr.edu.br

***Abstract*—Building a virtual world with simulated physical phenomena based on attraction and repulsion rules offers a unique opportunity to move agents according to these rules and plan actions mimicking real-world animal behavior. This paper presents an attraction-based algorithm using the Unscented Kalman Filter (UKF) to learn and predict opponent behavior in real time. The algorithm leverages attraction and repulsion forces to simulate physical interactions, facilitating robust predictions and learning accurately. Agents can optimize their strategies through reinforcement learning by adjusting attraction and repulsion parameters. Our results demonstrate the algorithm's effectiveness in dynamic environments, compared with traditional Q-learning methods, especially in low-frame conditions.**

Keywords: Unscented Kalman Filter; Animal Behavior; Machine Learning; Autonomous Systems.

I. INTRODUCTION

Creating a virtual environment governed by attraction and repulsion principles enables the simulation of agent movements and decision-making processes that closely resemble real-world animal behavior. Animal movement is a dynamic spatio-temporal process where trajectory data reflect the instantaneous animal position in space and time, and other factors influence movement decisions between these observed positions [1]. This framework allows for accurate simulation of physical interactions and provides a robust foundation for improving predictions and facilitating learning.

By leveraging attraction and repulsion rules in the simulation, we can achieve coherent and realistic modeling of agent behavior. These rules can predict future states of the system by simulating natural interactions and dynamics observed in real-world scenarios. For instance, when an agent (e.g., a robot) moves towards a target (e.g., a ball), the attraction force guides its path, while repulsion forces from obstacles ensure collision avoidance. This combination can be fine-tuned to predict the agent's trajectory accurately. Each agent is assumed to move following a first-order Newtonian law, distinguishing speed and orientation, which results from the balance of behavioral stimuli defined by direction and weight [2].

These rules can also be utilized for learning purposes. By simulating various scenarios and observing the outcomes, the system can iteratively adjust the parameters governing the attraction and repulsion forces to optimize agents' performance. This approach is particularly effective in reinforcement learning, where agents learn optimal strategies through trial and error within the simulated environment. Interactive multi-agent simulation algorithms compute the trajectories and behaviors of different entities in virtual reality scenarios. However, current methods involve considerable parameter tweaking to generate plausible behaviors [3].

Integrating learning algorithms, such as Artificial Neural Networks (ANN) or Kalman filters (KF), with these physical rules can enhance the system's predictive and adaptive capabilities. For example, a KF can estimate the state of the system (e.g., the positions and velocities of agents) and update predictions based on observed data. This iterative process refines the model's accuracy over time, accommodating both linear and nonlinear dynamics present in the simulation. The KF is commonly applied with ANN for chaotic systems identification [4].

Constructing a virtual world governed by attraction and repulsion rules not only allows for realistic simulation of physical phenomena but also provides a powerful tool for improving predictions and facilitating learning. By continuously refining the parameters and incorporating learning algorithms, the system can evolve to exhibit increasingly sophisticated and accurate behaviors akin to those observed in the natural world [5].

For concrete cases, consider Agre and Chapman's approach, which addresses the complexity, uncertainty, and immediacy of real-world situations. This approach was applied using agents to play Pengo [6]. Agre describes that Pengi, the implementation for playing Pengo, is part of a cognitive architecture theory derived from theories by Drescher, Minsky, and others. Mackworth proposed an architecture aimed at providing solutions for intelligent embedded systems using AI and robotics [7]. Sahota built on

the work of Agre, Chapman, and Mackworth, proposing a simulator with two robots and a ball [8].

Robots can cooperate to perform specific tasks, such as moving an object from one place to another [9]. For more complex tasks, studying the natural behavior of animals performing such tasks can be effective. Anderson and Donath question how a robot can automatically act across various tasks and environments, displaying diverse behaviors [10]. Observing animal behavior in nature for specific tasks without excluding others is beneficial.

In their work, Anderson and Donath identify behaviors of animals with evasion (repulsion) and attraction. Animals may avoid certain locations through repulsion behavior to prevent collisions with moving objects, evade predators, and avoid unsuitable environments. Repulsion can be passive (stopping to avoid collision, like freezing) or active (direct commands to avoid approaching objects). Attraction behavior is fundamental for a robot's movement toward a goal [9].

Using the basic idea of animal instinct, where something that captures attention exerts an attractive force, directs the predator's actions similar to the brain's command to grasp an object on a table. Rejection is also instinctual, where an unwanted object causes repulsion. This behavior can be expressed by attractive and repulsive potential fields [9].

Section II describes the KF and its variations used in this work, while Section III discusses Q-learning and its variation with ANN. Section IV presents the adopted methodology, Section V shows the obtained results, and finally, the conclusions are presented in Section VI.

II. KALMAN FILTER

The Kalman filter is a powerful tool for estimating the state of a dynamic system from noisy measurements. Designed for on-the-fly correction, it obtains precise measurements through sample observations [11]. In its basic form, it is an optimal linear estimator with constraints. The necessary linear functions are:

$$\begin{cases} x_k = Fx_{k-1} + q_k \\ z_k = Hx_k + r_k \end{cases} \quad (1)$$

Here, x_k is the current state, q_k is process noise (zero mean, covariance Q_k, and r_k is observation noise (zero mean, covariance R_k). F and H are transfer matrices. The state and noise distributions are:

$$\begin{cases} x_k \sim N(\bar{x}, P_k) \\ q_k \sim N(0, Q_k) \\ r_k \sim N(0, R_k) \end{cases} \quad (2)$$

The discrete KF involves prediction and update steps:

$$\begin{cases} \hat{x}_k^- = Fx_{k-1} \\ \hat{P}_k^- = FP_{k-1}F^T + Q \\ K_k = \frac{\hat{P}_k^- C^T}{C\hat{P}_k^- C^T + R} \\ \hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ \hat{P}_k = (I - K_k H)\hat{P}_k^- \end{cases} \quad (3)$$

The EKF handles nonlinearities by approximating models with linear functions around the current state [12]:

$$\begin{cases} x_k = f(x_{k-1}) + q_k \\ z_k = h(x_k) + r_k \end{cases} \quad (4)$$

$$F_{k-1} = \frac{\partial f}{\partial x} \quad (5)$$

$$H_k = \frac{\partial h}{\partial x} \quad (6)$$

$$\begin{cases} \hat{x}_k^- = f(x_{k-1}) \\ \hat{P}_k^- = F_{k-1}P_{k-1}F_{k-1}^T + Q \\ K_k = \frac{\hat{P}_k^- H^T}{H\hat{P}_k^- H^T + R} \\ \hat{x}_k = \hat{x}_k^- + K_k[z_k - h(\hat{x}_k^-)] \\ \hat{P}_k = (I - K_k H_k)\hat{P}_k^- \end{cases} \quad (7)$$

The EKF can train neural networks by treating weights as states to be estimated. The nonlinear mapping g is parameterized by the weight vector W:

$$y_k = g(x_k, W) \quad (8)$$

The error is defined by:

$$e_k = d_k - g(x_k, W) \quad (9)$$

The state-space representation is:

$$\begin{cases} W_k = W_{k-1} + v_k \\ y_k = g(x_k, W_k) + e_k \end{cases} \quad (10)$$

The UKF improves estimation for highly nonlinear systems using the unscented transformation [13]. It represents the state distribution with sigma points:

$$\begin{cases} X_i = \bar{x} \\ X_i = \bar{x} + (\sqrt{(L+\lambda)P_x})_i, \text{para } i = 1, \dots, L \\ X_\rangle = \bar{x} - (\sqrt{(L+\lambda)P_x})_i, \text{para } i = L+1, \dots, 2L \\ W_o^{(m)} = \frac{\lambda}{L+\lambda} \\ W_o^{(c)} = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \\ W_i^{(m)} = W_i^{(c)} = 1/\{2(L+\lambda)\}, \text{para } i = 1, \dots, 2L \end{cases} \quad (11)$$

where λ is a scalar parameter defined by the equation:

$$\lambda = \alpha^2(L + \kappa) - L \quad (12)$$

where L is the state vector dimensionality. α determines the spread of sigma points, κ influences the spread, and β incorporates prior distribution knowledge.

KFs are used in navigation, tracking, signal processing, and control systems. They provide robust state estimation and improve accuracy in predictions and measurements. The UKF's ability to handle nonlinearities without Jacobian computations makes it suitable for complex dynamic systems [14].

III. Q-Learning for Reinforcement Learning

Reinforcement Learning (RL) is a computational approach where an agent learns to make decisions by performing actions and receiving feedback from the environment. One of the widely used algorithms in RL is Q-learning, introduced by Watkins and Dayan [15]. Q-learning is a model-free RL algorithm that seeks to learn the value of the optimal policy, which guides the agent's actions to maximize the cumulative reward.

Q-learning is an off-policy RL algorithm that learns the value of an action in a particular state without requiring a model of the environment. The core component of Q-learning is the Q-table, which stores the value (*Q-value*) of each state-action pair. The Q-value update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \quad (13)$$

where $Q(s,a)$ is the Q-value for state s and action a. α is the learning rate *($0 < \alpha \leq 1$)*. r is the immediate reward received after performing action a in state s. γ is the discount factor *($0 \leq \gamma < 1$)*. s' is the next state after taking action a. $\gamma max_{a'} Q(s',a')$ represents the maximum Q-value for the next state s' across all possible actions a'.

To approximate the Q-value function, we use a feedforward neural network. The network takes the current state as input and outputs the Q-values for all possible actions. The network is trained to minimize the Temporal Difference (TD) error:

$$\delta = r + \gamma\, max_{a'}\, \mathrm{Q(s', a')} \; - \; \mathrm{Q(s, a)} \quad (14)$$

The ANN architecture consists of an input layer with a size equal to the state space dimension, one or more hidden layers. An output layer with a number of neurons equal to the number of actions.

IV. Methodology

A. Integrating the Atari Emulator for Reinforcement Learning Algorithms

The first step was to find an emulator that could provide the necessary structure to test the algorithms. We used the Atari emulator available within the OpenAI framework. OpenAI offers a comprehensive structure for reinforcement learning within the Python environment; however, there is a higher complexity associated with working with matrices and equations in this environment. To simplify and facilitate the visualization of states and images, we used the Octave environment. To call functions and procedures from Python, we used the pythonic library. The emulator is invoked with a simple call specifying the name of the game, which is an emulated ROM. The game chosen for testing was Boxing, a boxing game between two players with an overhead view where the player must press a button to punch and move to dodge or hit the opponent's nose to score points. The players are confined to the ring area. The first player to score 100 points before the time runs out wins by knockout.

B. Understanding Emulator Inputs and Outputs

The second step was to study what can be sent to the emulator and what can be received from it. The emulator can accept button inputs from the controller to move the player, i.e., up, down, left, right, and an action button, which in this case is to punch the opponent. It accepts binary values such as 1 for pressed and 0 for released. By selecting the button values, they can be sent to the emulator via the `set_button_mask` function. Each iteration of the emulator can be performed using the `step()` function. This function runs one cycle of the game. Since a game runs at 60 frames per second, calling the `step()` function advances the game by one frame.

The emulator provides various outputs, such as memory, which can be utilized using the `get_state()` function. This function returns the current state of the game, allowing it to be saved and later restored using the `set_state(state)` function. In this study, we used the `get_screen()` function to capture the screen at the current game state. This screen capture was used to visualize the current state of the game.

C. Applying Computer Vision Techniques

The third step involved applying computer vision techniques to extract important data for the algorithms. The targets were four objects: the two player sprites and the two score sprites. The player sprites were divided into three points on a two-dimensional Cartesian plane, corresponding to the fighter's head and the two boxing gloves. For the scoring sprites, we used a change detector on the scoreboard to count hits. The hits can be worth one or two points, depending on the proximity of the strike. We used only the hit count as the parameter.

D. Implementation of the Proposed Algorithm

The fourth step was the implementation of the algorithm. For the first algorithm, which we propose in this work, we developed an on-the-fly learning system that observes the moves of the opponent. To achieve this, we mapped the points of the players. Each player has three points, and we created two additional points representing the position of the gloves before throwing a punch. This is a good way to determine if a fighter is approaching the head of the opponent, regardless of whether they have thrown a punch.

Using this visualization, we applied a Discrete Kalman Filter (DKF) with three states to estimate the acceleration of the points and thus predict future states, which are the future positions of the players.

$$a^-_k = a_k \quad (14)$$
$$v^-_k = v_k + a_k \quad (15)$$
$$p^-_k = p_k + v^-_k \quad (16)$$
$$\begin{bmatrix} p_k \\ v_k \\ a_k \end{bmatrix} = \begin{bmatrix} 1 & \delta t & 0 \\ 0 & 1 & \delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ v_{k-1} \\ a_{k-1} \end{bmatrix} \quad (17)$$

where *a* is acceleration, *v* is velocity and *p* is position.

A second nonlinear filter was used to estimate the intention of the fighters. This algorithm utilizes the UKF to determine the value of the attraction coefficient *k* (repulsion if the value is negative) between the points of player 1 and player 2.

By observing the five points of each fighter, we obtain *k* values for each point on each axis. This is because, to land a precise hit, the fighter must align the glove with the opponent's head and then deliver the punch in a straight line. The algorithm was configured to run in real-time. The UKF can map the attraction coefficient function:

$$O'_a = Attraction(O_a, O_b, k) \quad (18)$$

where O_a is the object attracted by object O_b and k is the attraction coefficient. This is a global attraction defined at any point on the field. In this work, the objects are the points of each player. The attraction is process and observation function is given by:

$$\begin{cases} k_k = (k_{k-1}) + q_k \\ p'_1 = Attraction(p_1, p_2, k_k) + r_k \end{cases} \quad (19)$$

where k_k is the actual attraction coefficient and the p'_1 is the future observed state estimated by the DKF. p_1 e p_2 are the states for actual position.

E. Comparative Techniques

To compare the results, we used two other machine learning techniques commonly employed to control players. Typically, these techniques are executed in offline systems, where the player makes a move, and the reward is calculated. The player can avoid low or negative rewards by retracing steps and navigating through the best rewards. In our work, these algorithms had to be adapted for real-time use.

For example, the Q-learning algorithm was modified to remove the exploration rate and was fixed at 6 states, corresponding to one button being pressed at a time or no button pressed, representing a state of no action.

To test Q-learning with artificial neural networks in an on-the-fly system, we used the Extended Kalman Filter (EKF). The EKF was chosen because it requires less memory and processing power compared to the UKF. With many states, the nonlinear filter becomes heavy and slow due to the computational cost of the sigma points.

V. RESULTS

To generate the results, we used the longest processing time of the algorithms to set a standard for real-time performance. The algorithm with the longest processing time was the EKF with artificial neural networks used to train Q-learning, followed by the UKF, and then Q-learning with tables.

Another factor to note is that the Pythonic library does not have an implementation to transfer a variable or matrix directly from Python to Octave. Transferring matrices proved to be very slow. Transferring an emulator image took about 1.3 seconds, involving a long and complex process. A much faster alternative was to save the image within Python using library calls and then read it in Octave. Using an SSD, the task took about 0.003 seconds.

To simulate a real system, we assumed that out of 60 frames per second (fps) (Figure 1), only 4 frames are received, meaning one frame is received every 15 frames (Figure 2), or 4 fps. For this type of task, not all systems are suitable for tracking, and many details are lost, limiting the results.

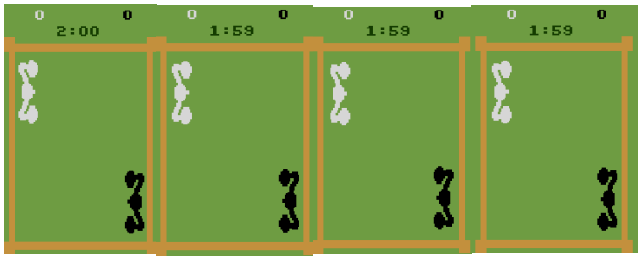

Figure 1. Sprite movement in 60 fps.

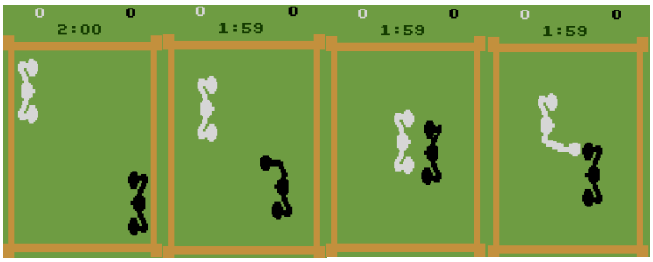

Figure 2. Sprites movement in 4 fps.

For the machine learning approach proposed in this work, we first used the DKF with the variance values x=[0;0;0], Q=10, R=1 and P=100 with Δt set to 4. We then applied the value *x* twice using y=F·(F·x) to obtain a prediction given the anterior states. This was done with the 5 points of each fighter. The DKF is used to provide a prediction of the movement in order to calculate the degree of attraction of the adversary to a position. Afterward, we applied the UKF with the values x=[0;0], Q=10, R=1 and P=100.

The algorithm for Q-learning was tested in a simulated environment with six predefined states. The Q-table was initialized, and the agent's performance was evaluated over multiple episodes. The learning parameters were set as follows: learning rate α=0.1, discount factor γ=0.9, and exploration rate ϵ=0.

The Q-learning algorithm with EKF and ANN was tested using the following parameters: Q=0.01, R=0.001, and P=1, with pre-defined, randomly initialized weights. The neural network architecture consisted of Input Layer: 12 neurons representing the 3 points of each fighter, Hidden Layer: 13 neurons, Output Layer: 5 neurons for the buttons. The transfer functions used were sigmoid for the hidden layer and linear for the output layer. For the Q-learning prediction values, we used a learning rate α of 0.1 and the following prediction equation:

$$o = p + \alpha(r - p) \quad (20)$$

where *o* is the observed state for EKF, *p* is the predicted state (the buttons for controller), *r* is the reward. The agent successfully learned to navigate the environment and maximize cumulative rewards. The Q-values converged to stable values, indicating the agent had effectively learned the optimal policy.

The reward calculation in the Q-learning algorithm is critical for guiding the agent towards the optimal policy. In this work, the reward function is defined to balance the time spent in the game, the effectiveness of the player's punches, and the actions taken by the player. The reward function is given by:

$$r = -1t_g + hit_1 hit_{1acc} - hit_2 hit_{acc2} + 5 * B \quad (21)$$

$$hit_{1acc} = hit_{1acc} + 7 \quad (22)$$

$$hit_{2acc} = hit_{2acc} + 3 \quad (23)$$

where t_g is the time counter of the game, hit_1 and hit_2 are the hit detection for player 1 and 2, hit_{1acc} and hit_{2acc} are the accumulative hit (each time, the hit value increase) and B is equal a 1 is for players' movement.

To test the inputs and the capability of the emulator, an agent was created with three actions: Approach, Retreat, and Attack. The player approaches the opponent's head, and when within a certain distance from either the left glove or the right glove, the attack button is pressed. If the opponent's glove is close before the player makes an attack, the player retreats to the upper or lower diagonal, depending on which of the opponent's gloves is closer.

In TABLE I, the results of the interactions are presented. The proposed algorithm can generate both actions and possible new positions for other algorithms. In this work, the actions were combined with Q-learning while the positions were passed to the EKF as states for the input of the Neural Network.

With the programmed player, the game time functions well because decisions are made at 60 frames per second, providing ample time to make decisions. The implementation was kept simple to test the use of buttons and the integration of the emulator into the Octave environment.

TABLE I. RESULTS OF THE ALGORITHMS IMPLEMENTED

Algorithms	*Score*	*Remaining Time*	*Total Rewards*
Programmed	KO x 74	34 s	45,422
UKF	79 x 78	0 s	27,670
Q-learning	KO x 97	3 s	33,361
EKF	24 x 58	0 s	-18,241
UKF + Q-learning	95 x 96	0 s	30,871
UKF + EKF	13 x 59	0 s	-42,262

By integrating these filtering techniques, we improved the prediction accuracy of the fighters' movements, which subsequently enhanced the performance of our reinforcement learning algorithm.

The implementation with the proposed algorithm of learning from intentions based on the attraction force between various points of the player proved to be a robust technique. This approach allowed the agent to learn intentions and use them to predict the opponent's attacks. Another positive aspect is that the agent did not have a predefined objective; it inferred that its objective was the same as the opponent's.

This capability highlights the algorithm's effectiveness in understanding and adapting to the game's dynamics, enhancing its predictive power and decision-making process in real-time scenarios.

The combination of the DKF and the UKF allowed us to effectively model the dynamic behavior of the fighters in real-time. The DKF provided a robust initial prediction, while the UKF refined these predictions by estimating the nonlinear effects, such as the attraction coefficient k between the fighters' points.

The defined reward function successfully guided the agent towards learning effective strategies in the game. By incorporating both the immediate rewards from actions and the long-term impact of hits landed by the players, the agent learned to balance between offensive and defensive strategies.

The use of EKF provided a robust method for state estimation, enhancing the performance of the Q-learning algorithm. The neural network effectively approximated the Q-values, allowing the agent to make informed decisions in real-time. The EKF algorithm exhibited issues with delayed positions, which theoretically should be better handled. Testing at 60 frames per second (fps) showed better results compared to other Q-learning techniques. However, when reduced to 4 fps, the EKF algorithm demonstrated weaknesses.

This indicates that while the EKF algorithm performs well under high frame rate conditions, its performance degrades with lower frame rates due to the increased latency in position updates. This latency impacts the algorithm's

ability to accurately predict and respond to the dynamic changes in the environment.

The limitation of using Q-learning with 6 states resulted in less natural movement of the player. However, with fewer directions, the algorithm performed better compared to using 9 states (8 directions plus attack). The issue with using Q-learning is that the reward must be known beforehand, and a function must be produced to achieve the optimal value.

By incorporating movement into the reward function, the player is discouraged from standing still and continuously punching. The idea of earning more points through successful attacks encourages the player to be more aggressive and proactive rather than merely avoiding the opponent.

The higher reward for attacking actions motivates the player to engage with the opponent actively. By rewarding movement, the player is incentivized to maneuver strategically rather than remain stationary. The penalty for being hit encourages the player to avoid attacks while planning their own.

VI. CONCLUSIONS

Both implementations have their strengths and limitations. The EKF with neural networks showed potential in understanding and predicting dynamic game scenarios, while Q-learning with fewer states proved to be more efficient in specific conditions. Future work should focus on improving the robustness of these algorithms, particularly in handling lower frame rates and refining the reward structures for better performance.

The reward function was integral in shaping the player's behavior, promoting a balance between offensive and defensive strategies while maintaining an active and engaging playstyle. The careful design of the reward structure ensured that the player optimized both movement and attack to achieve the best results.

This work presents an attraction-based algorithm as a more intuitive solution utilizing the UKF with attraction functions. The initial idea behind this algorithm was to learn the behavior of opponents to predict future positions and actions. However, it proved capable of learning in real-time within a limited sampling state, which would be impractical for humans.

The algorithm's ability to learn and predict in real-time, even with limited sampling data, demonstrates its potential for practical applications in dynamic environments. By modeling attraction forces between key points of the players, the algorithm effectively learns and anticipates opponent behavior, enabling strategic decision-making.

In future work, we plan to extend the application of this algorithm by analyzing video footage, applying the algorithm to analyze videos of players to learn their tactics. This would involve extracting key movement patterns and strategies from recorded gameplay. Testing with different players evaluating the learned strategies against both the implemented player and other players to assess the robustness and adaptability of the algorithm, enhancing the model improving the attraction model to incorporate more complex behaviors and interactions, potentially including environmental factors and varying opponent skill levels.

The proposed algorithm has potential applications beyond gaming, such as in sports analytics, where understanding and predicting player movements can provide significant strategic advantages. Additionally, it can be applied in robotics for real-time path planning and obstacle avoidance by learning dynamic environments.

REFERENCES

[1] R. W. Loraamm, “Incorporating Behavior Into Animal Movement Modeling: A Constrained Agent-Based Model For Estimating Visit Probabilities In Space-Time Prisms”, International Journal Of Geographical, 2019.

[2] S. Bernardi and M. Scianna, “An agent-based approach for modelling collective dynamics in animal groups distinguishing individual speed and orientation”. Phil. Trans. R. Soc. B 375: doi.org/10.1098/rstb.2019.0383, 2020.

[3] J. Ren, W. Xiang, Y. Xiao, R. Yang, D. Manocha and X. Jin, “Heter-Sim: Heterogeneous Multi-Agent Systems Simulation by Interactive Data-Driven Optimization”. IEEE Transactions on Visualization and Computer Graphics. doi:10.1109/tvcg.2019.2946769, 2019.

[4] R. J. de Jesus, “Stable Kalman filter and neural network for the chaotic systems identification”, Journal of the Franklin Institute, doi: 0.1016/j.jfranklin.2017.08.038, 2017.

[5] Y. Tanaka, “Machine Learning That Reproduces Physical Phenomena from Data”, NTT Technical Review, vol. 21, no. 10, pp. 15-19, Oct., 2023.

[6] P. Agre and D. Chapman, “Pengi: An implementation of a theory of activity”. In Proceedings of AAAI-87, pp. 196-201, 1987.

[7] A. K. Mackworth, “On seeing robots”, in: Computer Vision: Systems, Theory, and Applications. Singapore World Scientific, pp. 1–13, 1993.

[8] M. K. Sahota and A. K. Mackworth, “Can situated robots play soccer?”, in Artificial Intelligence 94, pp. 249–254. Banff, Alberta, 1994.

[9] J. T. Weigel, S. Gutmann, M. Dietl, A. Kleiner, and B. Nebel, “CS Freiburg: Coordinating Robots for Successful Soccer Playing”. IEEE Transactions on Robotics and Automation, vol. 18, no. 5, pp. 685-699, 2002.

[10] T. L. Anderson and M. Donath, “Animal behavior as a paradigm for developing robot autonomy”. Robotics and Autonomous Systems, vol. 6(1-2), pp. 145–168. doi:10.1016/s0921-8890(05)80033-8, 1990.

[11] R. E. Kalman, "A new approach to linear filtering and prediction problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35-45,1960.

[12] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems", Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, 1997.

[13] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation", Proceedings of the IEEE, vol. 92, no. 3, pp. 401-422, 2004.

[14] R. van der Merwe and E. Wan, "Sigma-point Kalman filters for probabilistic inference in dynamic state-space models", Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2001.

[15] C. J. C. H. Watkins and P. Dayan, Q-learning. Mach Learn 8, pp. 279–292. Doi: 10.1007/BF00992698, 1992.