Reinforcement Learning for Ball Striking in a Golf-like Setting: A Markov Decision Process Approach

Bamberger, Assaf
assaf.bam@campus.technion.ac.il
CS Department
Technion - Israel Institute of Technology

Ephraim, Asaf
asafephraim@campus.technion.ac.il
CS Department
Technion - Israel Institute of Technology

September 3, 2023

Abstract

The ability to strike a ball with the right velocity and angle is crucial in various sports and robotic applications. In this paper, we propose a reinforcement learning approach to teach a robot to strike a ball in a golf-like setting. We model the problem as a Markov Decision Process (MDP) and develop a transition model to simulate the ball's rolling motion after being struck. We formulate a reward function that encourages the ball to roll in a way that after given number of seconds t it will be as close to a target location as possible. Through extensive training in a simulated environment, the robot learns an optimal policy for striking the ball. We evaluate the acquired policy's performance using optimal world formulas and then test it in a simulated Gazebo environment. The Gazebo environment closely replicates our lab conditions, showcasing the effectiveness of our approach in achieving accurate ball striking across diverse testing scenarios. In future work, we plan to extend our experiments to the physical world to further validate the effectiveness of our approach in achieving accurate ball striking.

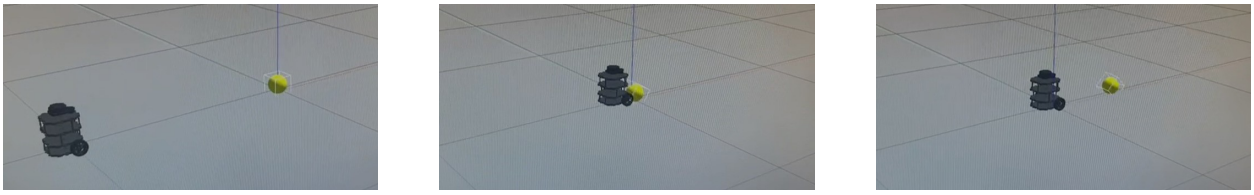
1 Introduction

Ball striking, which involves hitting a ball with the appropriate velocity and angle, is a fundamental skill in various sports such as golf, baseball, and tennis. Additionally, robotic systems often require precise ball-striking abilities for tasks such as object manipulation and coordination. Developing a robot's ability to strike a ball accurately is a challenging problem due to the complex dynamics involved in ball motion.

Reinforcement learning (RL) has shown promising results in training agents to perform complex tasks by learning from interactions with the environment. In this work, we apply RL techniques to teach a robot how to strike a ball in a golf-like setting. By formulating the problem as an MDP, we can learn an optimal policy for the robot to determine the velocity and angle of its strikes.

In this paper, we present a detailed methodology for modeling the ball-striking problem as an MDP. We define the state space, action space, and reward function, which capture the relevant variables and goals of the task. We also developed a transition model that incorporates the robot's movements and the ball's rolling motion after being struck.

To train the robot, we propose a training procedure involving updating the agent's value function and policy based on the observed rewards. Our training process was based on using optimal world data as a starting point. Essentially, we ensured that the training process began with a solid theoretical basis before moving into practical simulation experiments. These experiments were conducted in a highly realistic Gazebo simulation environment closely aligned with our lab setting. While we didn't perform real-world testing in this study, our comprehensive evaluation process was focused on the simulated environment, allowing us to assess the performance and generalization capabilities of the learned policy in this controlled context.



(a) initial position (b) the robot strikes the ball (c) the ball is rolling

Figure 1: Our setup

The rest of this paper is organized as follows: Section 2 presents the methodology for modeling the ball striking problem as an MDP, including the state space, action space, reward function, and transition model. Section 3 outlines the experimental protocol for training the RL agent using optimal world formulas and testing it in the Gazebo simulated environment. Section 4 discusses the results of our experiments and provides a detailed analysis. Finally, Section 5 concludes the paper and highlights future research directions.

2 Methodology

In this section, we present the methodology for modeling the ball-striking problem as a Markov Decision Process (MDP) and describe the components of the MDP.

2.1 State Space

The state space consists of the relevant variables that define the current state of the system. In our problem, the state space includes:

- Robot's x-coordinate x_r
- Robot's y-coordinate y_r
- Ball's x-coordinate x_b
- Ball's y-coordinate y_b
- Ball striking flag $f_{striking}$
- Ball's time of free movement t_{roll}

2.2 Action Space

The action space defines the available actions that the robot can take. In our problem, the action space includes:

- The velocity of the robot's strike v_r
- The angle of the robot's strike θ_r

2.3 Reward Function

The reward function defines the immediate rewards obtained by the agent for transitioning from one state to another. In our problem, we define the reward function as follows:

- We have opted for a continuous reward function, defined as follows: $R(s) = -\log\left(\gamma \cdot d\left(\left(x_{target}, y_{target}\right), \left(x_{current\_ball\_location}, y_{current\_ball\_location}\right)\right)\right)$, where d represents the distance metric between the current ball location $(x_{current\_ball\_location}, y_{current\_ball\_location})$ and the target location (x_{target}, y_{target}), and γ represents some coefficient for regularisation. In our work we choose $\gamma = 10$ in order to increase the sensitivity of the reward function for small differences near the target.

2.4 Value Function and Policy

The value function represents the expected cumulative rewards obtained by following a specific policy in a given state. The policy determines the mapping from states to actions. In our problem, we aim to learn the optimal value function and policy using reinforcement learning algorithms.

2.5 Transition Model

The transition model defines how the state evolves over time, incorporating the robot's movement towards the ball, the strike action, and the ball's subsequent rolling motion. The time factor is considered, and the transition from one state to the next is checked every 1 second.

2.5.1 Ball Not Struck ($f_{striking} = 0$):

For each state s and action a, the transition model determines the resulting state s' based on the robot's movement towards the ball.

- Robot's position is updated based on its movement toward the ball:
  - x-coordinate of the new position:

$$x_r' = x_r + (v_r \cdot \cos(\theta_r))$$

  - y-coordinate of the new position:

$$y_r' = y_r + (v_r \cdot sin(\theta_r))$$

- Ball's position remains unchanged: $(x'_b, y'_b) = (x_b, y_b)$
- The ball striking flag remains 0: $f'_{striking} = 0$
- The Ball's time of free movement remains 0: $t'_{roll} = 0$
- The resulting state s' is defined by the updated positions of the robot and the ball: $\left(x'_r, y'_r, x'_b, y'_b, f'_{striking}, t'_{roll}\right)$

2.5.2 Ball Struck ($f_{striking} = 1$):

For each state s and action a, the transition model determines the resulting state s' based on the strike action and the subsequent rolling motion of the ball.

- Robot's position is updated based on the strike action:
  - X-coordinate of the new position: $x'_r = x_r$
  - Y-coordinate of the new position: $y'_r = y_r$
- Ball's position is updated based on rolling motion:
  - We assume perfect transfer of energy and no heat deceptions.
    so: $v_b = \sqrt{\frac{m_r}{m_b}} \cdot v_r$
  - X-coordinate of the new position:
    * The ball is barely affected by friction since at the point of contact with the floor the radial velocity is 0
    * The velocity of the ball is $v_{x\_ball} = v_b \cdot \cos(\theta_r)$
    * Hence, $x'_b = x_b + v_{x\_ball}$
  - Y-coordinate of the new position:
    * In the same manner as for the x-coordinate or the ball, $v_{y\_ball} = v_b \cdot \sin(\theta_r)$
    * So, $y'_b = y_b + v_{y\_ball}$
- The time that the ball is in free motion is one second larger than the last state (since the states are one second apart): $t'_{roll} = t_{rool} + 1$
- The resulting state s' is defined by the updated positions of the robot and the ball: $\left(x'_r, y'_r, x'_b, y'_b, f'_{striking}, t'_{roll}\right)$

The transition model explicitly defines the resulting state s' after performing an action "a" in state s. The positions of the robot and the ball are updated based on their respective movements, considering the time factor. The ball striking flag indicates whether the ball has been struck or not.

3 Experimental Protocol

In this section, we outline the experimental protocol for evaluating the proposed RL approach.

3.1 Ideal world Environment

The initial phase of the experiment will take place in a simulation of the ideal world, i.e. no energy lost to heat, the ball is a perfect sphere and no air resistance. The simulated environment will be set up to replicate the golf-like setting with the robot and the ball, based on mathematical formulas. The RL agent will interact with this simulated environment and learn an optimal policy through training.

3.2 Training Procedure

The training procedure is designed to train the RL agent to learn the optimal policy for ball striking. It consists of the following steps:

1. Initialize the RL agent with an initial policy
2. Initialize a simulation environment with the robot and the ball at their starting positions
3. Repeat the following steps for a specified number of episodes or until convergence:
   (a) Reset the state of the environment to the initial state
   (b) Execute the RL agent's policy to select actions based on the current state
   (c) Update the positions of the robot and the ball based on the selected actions and the transition model
   (d) Calculate the reward based on the new state and update the RL agent's value function
   (e) Update the RL agent's policy based on the updated value function
4. Evaluate the learned policy by testing it in the simulated environment

3.3 Real-World Simulated Environment Testing

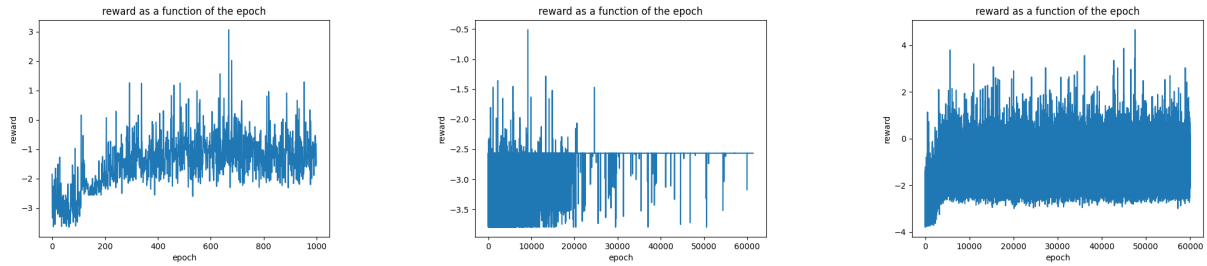
After training the RL agent in a simulated ideal world environment, the learned policy will be tested in a *Gazebo* simulation, mimicking the real world. The robot and the ball will be set up in an environment resembling the simulated setting. The RL agent will execute actions based on the learned policy, and the resulting ball-striking performance will be evaluated based on its ability to reach the desired position accurately.

4 Results and Discussion

In our experimental setup, we assessed the effectiveness of three reinforcement learning algorithms: *DDPG*, *SAC*, and *PPO*, in their efforts to optimize the solution to our problem within a Lab-like Gazebo simulated environment. The primary focus was on training an RL agent to learn an optimal values for θ, the angle at which the robot approaches the ball, and the velocity that the TurtleBot3 needs to strike the ball at in order to make the ball roll as close as possible to a target point, in a time frame t.

4.1 DDPG's Superior Performance

In our comprehensive empirical study, a notable finding emerged: DDPG proved to be the standout performer, particularly excelling in its ability to control the ball's angle accurately. Moreover, it exhibited a remarkable advantage in terms of convergence when compared to both PPO and SAC. Furthermore, DDPG not only outshone its counterparts but also showcased its effectiveness in learning the precise control policies required to achieve accurate angular adjustments for ball hitting. It's worth emphasizing that while SAC demonstrated respectable performance, it demanded a significantly greater number of training iterations to reach convergence. Conversely, PPO fell behind, yielding less accurate results when contrasted with DDPG and SAC in this research.



(a) DDPG, $1,000$ epochs (b) PPO, $60,000$ epochs (c) SAC, $60,000$ epochs

Figure 2: Algorithms results after n epochs of training

<u>*note:*</u> results for each of the algorithms can be found in the Appendix.

- DDPG: figure 3
- PPO: figure 4
- SAC: figure 5

4.2 Challenges with Velocity

DDPG demonstrated remarkable convergence during its training phase, following precise mathematical formulas. However, when we transitioned from this idealized training setting to a real-world-simulated environment, things became more challenging. In this lifelike setting, we introduced elements like wind and other variables that exist in the real world.
This transition revealed an interesting gap. While DDPG consistently excelled in controlling the ball's angle with high precision, its performance in controlling the ball's speed was not as consistent, even though the angle was spot-on.

4.3 Explanation of Velocity Challenges

Several factors may explain the challenges encountered with velocity control when using DDPG in a more realistic simulation environment:

- Idealized Training Environment: In the more realistic simulation, the ball is subject to physical forces, such as air resistance and friction, which may not have been fully considered during DDPG training with our optimal world formulas. These real-world physics factors can cause deviations from the desired velocity. The DDPG algorithm relies on a model of the environment to make predictions. If the model does not perfectly match the dynamics of the simulated environment, it can lead to discrepancies between predicted and actual velocities.

- Complexity of Control: Accurate velocity control can be more challenging than angle control, as it requires precise adjustments in force and timing. The interactions between these factors may not have been fully learned or generalized by the DDPG agent.

- Noise and Uncertainty: Real-world environments are subject to noise and uncertainty, which may not be fully accounted for during training. These uncertainties can lead to variations in velocity.

4.4 Potential Solutions and Future Work

To address the velocity challenges observed in the more realistic simulation environment, several approaches can be considered:

- Improved Training and Environment Modeling: Enhance the accuracy of the environment model used during DDPG training to better reflect the dynamics of the realistic simulation.

- Advanced Reinforcement Learning Algorithms: Explore advanced reinforcement learning algorithms that are designed to handle complex and uncertain environments, such as Trust Region Policy Optimization (TRPO) or Model Predictive Control (MPC).

- Reward Function Modification: Refine the reward function used in training to emphasize velocity control and encourage the agent to prioritize precise velocity adjustments.

5 Conclusion

In this study, we embarked on a journey to teach a robot the art of ball striking in a golf-like setting, employing reinforcement learning techniques within a Markov Decision Process (MDP) framework. Our objective was to enable the robot to strike the ball with the right velocity and angle, ultimately achieving accurate ball rolling towards a desired target position. While our research yielded valuable insights and promising results, it also uncovered challenges that warrant further investigation. We explored three prominent reinforcement learning algorithms: DDPG (Deep Deterministic Policy Gradient), SAC (Soft Actor-Critic), and PPO (Proximal Policy Optimization). Among these, DDPG emerged as the most promising candidate, offering a high degree of accuracy in determining the optimal striking angle for the ball, while requiring the least amount of training.
However, our journey also revealed a significant hurdle. The velocity calculated by our agents, while accurate within an optimal world defined by "clean" physics formulas, proved less reliable when applied in a more complex and unpredictable real-world-like environment, such as the Gazebo simulated environment. This discrepancy between the idealized world and the real-world simulation can be attributed to various physical parameters, including wind and other environmental factors,

which are difficult to model accurately by simple formulas.
In essence, we achieved commendable results in calculating the optimal striking angle (θ) for the ball, demonstrating the potential of reinforcement learning in mastering complex tasks. Nevertheless, the accuracy of the calculated velocity fell short when confronted with the intricacies of the real-world-like simulation, highlighting the challenges of bridging the gap between the ideal and simulated worlds.
Our findings underscore the importance of advancing our research. Future work might focus on training our agents within the simulated environment to better align with real-world conditions. By addressing the discrepancies in velocity calculations and accounting for the myriad physical factors that influence ball motion, we aim to enhance the realism and practicality of our learned policies. Ultimately, our goal is to test and refine our approach in a physical laboratory environment, further investigating the disparities between the Gazebo simulation and the physical world.
In conclusion, this study has provided valuable insights into the intricacies of reinforcement learning for ball striking in a golf-like setting. It has illuminated the challenges of transitioning from an idealized world to a simulation environment and, ultimately, to the real world. As we continue our journey, we look forward to narrowing these gaps and advancing the capabilities of robotics in the realm of precise ball striking, thereby extending the boundaries of what machines can achieve.

6 References

- Our Team GitHub Repository: assafBam/Affordance-RL

7 Appendix

```
(air) assaf@assaf-comp-unix:~/Desktop/AIR-project$ ./test 1000 ddpg
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (0, -1.3)        action: [0.09046108 1.582157  ]
target: (0, 1.3)         action: [0.09088472 4.712387  ]
target: (1.3, 0)         action: [0.09213719 3.1460989 ]
```

Figure 3: DDPG's results, $1,000$ epochs

```
(air) assaf@assaf-comp-unix:~/Desktop/AIR-project$ ./test 60000 ppo
target: (0, 1.3)          action: [0.        0.172476]
target: (0, -1.3)         action: [0.          0.45962256]
target: (0, 1.3)          action: [0. 0.]
target: (0, 1.3)          action: [0. 0.]
target: (0, -1.3)         action: [0.          0.90403956]
target: (1.3, 0)          action: [0. 0.]
target: (1.3, 0)          action: [0. 0.]
target: (1.3, 0)          action: [0. 0.]
target: (1.3, 0)          action: [0. 0.]
target: (0, 1.3)          action: [0. 0.]
target: (0, -1.3)         action: [0.        2.120098]
target: (0, 1.3)          action: [0.         0.6939698]
target: (0, -1.3)         action: [0. 0.]
target: (1.3, 0)          action: [0. 0.]
target: (0, -1.3)         action: [0.          0.10865293]
target: (0, -1.3)         action: [0.         0.7840185]
target: (0, -1.3)         action: [0.         2.4004717]
target: (0, 1.3)          action: [0. 0.]
target: (0, 1.3)          action: [0. 0.]
target: (1.3, 0)          action: [0.         0.1611561]
```

Figure 4: PPO's results, $60,000$ epochs

```
(air) assaf@assaf-comp-unix:~/Desktop/AIR-project$ ./test 60000 sac
target: (0, 1.3)          action: [0.10081013 4.677064  ]
target: (1.3, 0)          action: [0.10137131 2.9878    ]
target: (0, 1.3)          action: [0.07251389 4.8318205 ]
target: (0, -1.3)         action: [0.10173906 1.2648453 ]
target: (0, 1.3)          action: [0.10620108 4.7317247 ]
target: (0, 1.3)          action: [0.08639874 4.651271  ]
target: (1.3, 0)          action: [0.10203255 3.1781895 ]
target: (0, 1.3)          action: [0.08639418 4.972346  ]
target: (1.3, 0)          action: [0.0836476 3.0134897]
target: (1.3, 0)          action: [0.08908661 3.089475  ]
target: (0, -1.3)         action: [0.09684896 1.7811538 ]
target: (0, -1.3)         action: [0.08536867 1.492084  ]
target: (0, 1.3)          action: [0.10450923 4.8348365 ]
target: (1.3, 0)          action: [0.08942605 3.1510906 ]
target: (1.3, 0)          action: [0.07671729 3.1958213 ]
target: (0, 1.3)          action: [0.07913095 4.3348055 ]
target: (0, -1.3)         action: [0.0776986 1.629631 ]
target: (0, 1.3)          action: [0.11145891 4.5230513 ]
target: (0, 1.3)          action: [0.11631073 4.673488  ]
target: (0, 1.3)          action: [0.08188479 4.7941656 ]
```

Figure 5: SAC's results, $60,000$ epochs