Extended Abstract

Motivation To the casual observer, batting may seem straightforward, but it is widely considered one of the most difficult tasks in sports. In fact, even the most talented batters in MLB history manage to secure a hit in only about 30% of their plate appearances. A pitched ball reaches the batter in under half a second, demanding precise perception, quick decision-making, and rapid motor execution. For a robotic agent, this presents a short-episode, long-horizon challenge where early posture and motion are critical for achieving sufficient bat velocity at impact. The reward space is also extremely sparse, as traditional rewards occur only when the bat contacts the ball. Our goal is to teach a robotic arm to consistently hit a baseball using various methods—Behavioral Cloning (BC), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO)—and compare which approach produces the most effective and realistic swing.

Method For all of the following experiments we utilized the same environment setup: a single Panda robotic arm elevated on a platform to simulate the torso of the batter, with a bat placed in the end effector of the robotic arm and a ball placed in the environment for the agent to target. For behavior cloning (BC), we derived an expert policy from a dataset of swing trajectories collected from minor league and collegiate level players, using the robots prior k states to predict its desired state at the next timestep, and applying inverse kinematics to compute the joint torques required for this state change. For Soft Actor-Critic (SAC), we used an off-policy reinforcement learning algorithm to stabily learn stochastic policies and help the agent avoid local minima. For Proximal Policy Optimization (PPO), we employed an on-policy reinforcement learning method in which the agent continually generates new data from its current policy and refines the policy through clipped surrogate objective optimization, balancing exploration and policy stability.

Implementation We used PyBullet as the physics engine to simulate collisions between the ball and the bat. To prepare the BC dataset, we parsed baseball swing data from the OpenBiomechanics dataset and extracted the pose of the bat's sweet spot at each timestep. We then used a custom inverse kinematics solver to compute the corresponding joint-space configurations for the Franka Emika Panda robot, creating joint-annotated trajectories for training. We trained a multi-layer perceptron (MLP) policy to predict the next bat state based on the past k observations. For SAC and PPO, we used the same environment as in BC and trained policies using Stable-Baselines3 with MLP-based SAC and PPO models. The reward function was defined based on the bat's speed, its distance to the ball, and whether contact with the ball was achieved.

Results Behavior Cloning produced functional swing trajectories but failed to consistently emulate a realistic swing and align the bat with the ball, resulting in degraded reward performance over time. Soft Actor-Critic learned to exploit the reward function by generating high bat velocity in the ball's general vicinity, but the resulting swings lacked biomechanical realism and precision. However, Proximal Policy Optimization learned the most fluid and controlled swing motion, achieving stable and consistent reward improvement during training and evaluation. Visual inspection confirmed that PPO produced realistic swing trajectories with promising potential for high-quality contact.

Discussion Our experiments highlight the trade-offs between imitation-based and reward-driven learning strategies for robotic baseball hitting. BC provides a strong foundation for generating realistic motion but lacks adaptability to task-specific feedback. SAC's stochastic nature and tendency to exploit the reward function made it poorly suited for this precision task. PPO's clipped objective enabled it to balance realism with task performance, producing more consistent and purposeful swings. Nonetheless, all methods leave room for improvement, particularly in refining swing timing and increasing hitting consistency.

Conclusion This project shows that a robotic arm can learn to perform realistic baseball swings through a combination of reinforcement and imitation learning. PPO produced the most effective swings, while BC and SAC revealed clear limitations. Future work should explore model-based and hierarchical RL to further improve performance. These findings offer insights into applying reinforcement learning to fast, precision tasks with long-horizon dynamics.

Swing For the Fences: Deep Reinforcement Learning for Batting Practice

Chase Joyner
Department of Computer Science
Stanford University
`chasezj@stanford.edu`

Mack Smith
Department of Computer Science
Stanford University
`macks26@stanford.edu`

Abstract

Hitting a baseball is one of the most difficult feats in sports, so much so that major league players succeed in only about 30% of their at-bats. Replicating this task with a robotic agent presents a challenging reinforcement learning problem involving precise motor control, long-horizon dependencies, and sparse rewards. Our goal is to teach a Franka Emika Panda robot, equipped with a baseball bat, to consistently hit a pitched ball with high-quality contact. We compare three approaches: Behavior Cloning (BC), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO).

1 Introduction

To the casual observer, hitting a baseball may seem straightforward, but it is one of the most difficult tasks among all sports. In a professional setting, a pitched ball reaches the batter in under half a second, demanding precise perception, quick decision-making, and rapid motor execution. For a robotic agent, this means integrating fast sensory input, motion prediction, and a one-time, high-precision swing, posing a difficult challenge for deep reinforcement learning.

Our goal in this project is to design an agent capable of hitting a baseball in a simulated environment. The agent must learn both when and how to swing a virtual bat based on observations of the baseball. The problem is especially difficult due to the sparsity of rewards, as positive feedback is only received when the agent makes solid contact with the ball, achieving sufficient exit velocity and an optimal launch angle. Moreover, the task itself is inherently long-horizon despite having a short episode, as early elements of the swing strongly influence downstream outcomes.

To address these challenges, we explored a range of learning-based approaches. We began by leveraging the open-source biomechanical baseball dataset published by Driveline Research [1], which provides detailed 3D joint kinematics and kinetics for motion-captured baseball swings from professional baseball players. Using this data, we trained a Franka Emika Panda robot in simulation to replicate the core mechanics of expert human swings through behavior cloning (BC) in PyBullet [2]. This imitation learning phase focused on capturing realistic swing motion in isolation, without requiring the agent to reason about ball contact.

In the next phase, we shifted our focus from imitating expert motion to optimizing the swing for task performance. While the ball was present throughout all phases, behavior cloning emphasized reproducing motion patterns regardless of outcome. In contrast, our model-free reinforcement learning approaches—Soft Actor-Critic (SAC) [3] and Proximal Policy Optimization (PPO) [4]—used continuous feedback from the ball-to-bat interaction to refine swing execution. The reward function encouraged not only making contact, but also maximizing bat speed and producing favorable ball exit trajectories. This phase allowed the agent to adapt its timing and control to consistently deliver realistic and powerful swings.

To evaluate the effectiveness of each method, we compared BC, SAC, and PPO within the same simulation environment. Models were assessed based on their ability to make consistent and accurate contact with the ball, generate sufficient swing speed and angular momentum, and produce desirable ball trajectories. This comparison enabled us to examine the trade-offs between imitation-based and goal-directed learning strategies, and to identify which approach best balances biomechanical realism with task success.

By combining imitation learning and deep reinforcement learning, our project offers a platform for studying fast, one-shot decision-making and motor control in complex environments.

2 Related Work

Our project builds on recent advances in deep reinforcement learning, motion retargeting, and dynamic robotic control for fast, one-shot tasks. A central foundation for our approach is the DeepMimic framework introduced by Peng et al. [5], which demonstrated that deep RL can be combined with example-guided motion imitation to enable agents to learn highly dynamic, physics-based skills. DeepMimic showed that simulated characters can not only replicate complex motion patterns from reference data, but also optimize these motions for new objectives and adapt them to diverse environments. This approach provides a strong foundation for our method, which combines imitation learning from expert baseball swings with RL-based optimization to achieve effective hitting.

Motion retargeting across morphologies is also highly relevant to our work. For instance, Peng et al. [6] developed a system for learning agile robotic locomotion by imitating animal motion. Their modular architecture enables motion capture data to be adapted to robots with different kinematics and dynamics while preserving key motion characteristics. We adopt a similar strategy in retargeting human baseball swing data to a robotic arm with a distinct physical embodiment.

Finally, Dambrosio et al. [7] provided a compelling proof of concept that deep RL architectures can succeed in fast, precision one-shot striking tasks. Their competitive robot table tennis system combined hierarchical RL with domain randomization and modular skill learning to train a robot capable of rallying with human players. The ability of this system to integrate long-horizon planning with rapid execution directly motivates our application of similar techniques to baseball hitting, which presents comparable challenges in timing, accuracy, and sparse rewards.

3 Method

In this section, we present the methodology used to enable a robot to learn and execute realistic baseball batting motions. First, we apply Behavior Cloning (BC), an imitation learning technique, to learn a policy from expert human demonstrations. This provides an initial policy that is capable of producing plausible swing trajectories. We then direct our attention towards using deep reinforcement learning methods, specifically Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), which allow the agent to refine its behavior through interaction with the simulated environment. All methods share the same action space, which is defined as the vector of joint angle deltas for the seven controllable joints of the Panda robot:

$$a_t = \Delta\theta_t \in \mathbb{R}^7 \tag{1}$$

where $\Delta\theta_t$represents the vector of changes in the seven joint angles of the Franka Emika Panda robot at time step t. These joint deltas are directly applied to update the robot's configuration in the simulator.

3.1 Behavior Cloning

The first stage of our pipeline focuses on leveraging expert demonstrations to learn an initial batting policy via Behavior Cloning. For this purpose, we utilize a publicly available dataset of motion capture recordings collected from 496 right-handed professional baseball batters. These recordings contain detailed three-dimensional joint trajectories throughout each swing. Since the Franka Emika Panda robot possesses only a single 7-DoF arm, we focus our retargeting efforts exclusively on the lead arm of each batter (the left arm in the case of right-handed batters), which is primarily responsible for guiding the bat through the swing.

Adapting human motion capture data to the morphology and kinematic constraints of the Franka Emika Panda robot presents a significant challenge. The human arm possesses a slightly different joint structure and range of motion compared to the robot, and direct mapping from joint positions in 3D for physics engines like PyBullet is infeasible. To address this, we implement a custom motion retargeting pipeline based on inverse kinematics optimization. For each frame of the recorded swing, we solve the following optimization problem:

$$\theta^* = \arg\min_{\theta} \sum_{j \in \mathcal{J}} \left\| p_j^{\text{robot}}(\theta) - p_j^{\text{human}} \right\|^2 \tag{2}$$

In this formulation, θ denotes the vector of the robot's joint angles. The term $p_j^{\text{robot}}(\theta)$ represents the forward kinematics mapping from the robot's joint angles to the Cartesian position of joint j on the robot, while p_j^{human} denotes the corresponding target joint position from the human motion capture data. The set $\mathcal{J}$ contains the joints selected for retargeting, which include key anatomical points of the lead arm elbow and wrist.

The output of this optimization is a set of temporally consistent joint angle trajectories that approximate the lead arm motion of the expert batters, adapted to the Panda robot's kinematic model. These trajectories are then paired with corresponding environmental states to construct expert demonstration data suitable for training a policy.

To capture the relevant context for batting behavior, we define the robot's state at each time step t as follows:

$$s_t = [p_{\text{ball}}, p_{\text{bat}}, h_{\text{ball}}, \theta_t, s_{t-1}, s_{t-2}, \ldots, s_{t-k}] \tag{3}$$

In this state formulation, p_{ball} represents the current three-dimensional position of the ball in the global coordinate frame, and $p_{\text{bat}}$is the three-dimensional position of the bat's center or sweet spot. The term $h_{\text{ball}}$is a binary indicator that signals whether the ball has been successfully hit at any point during the current swing. Once again, θ_t represents the robot's current joint angles. Additionally, we append the previous k state observations $s_{t-1}, s_{t-2}, \ldots, s_{t-k}$ to provide temporal context, enabling the policy to reason over recent swing dynamics and anticipate ball motion.

Given this processed dataset of state-action pairs, we proceed to train a neural network policy $\pi_\theta(a_t \mid s_t)$ to imitate the expert behavior. We employ a standard Behavior Cloning objective, minimizing the mean squared error (MSE) between the actions predicted by the policy and the corresponding expert actions observed in the demonstration trajectories:

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| a_t^{(i)} - \pi_\theta(s_t^{(i)}) \right\|^2 \tag{4}$$

The resulting policy is capable of mimicking swing motions from expert baseball players when deployed in the simulated environment, providing a strong baseline for further reinforcement learning.

3.2 Soft Actor-Critic and Proximal Policy Optimization

For our model-free reinforcement learning experiments, we implemented SAC and PPO using the Stable-Baselines3 library. We used their standard SAC and PPO wrappers for continuous action spaces and trained policies directly within our PyBullet simulation environment.

The state space for both SAC and PPO was similar to that used in behavior cloning, consisting of the current joint angles of the robot arm, the positions of the ball and bat tip, and a binary indicator of whether the ball had been hit. Unlike behavior cloning, no history of prior states was included, as each observation reflected only the current state of the environment.

To train both SAC and PPO learn a realistic baseball swing, we designed a task-specific reward function to encourage accurate and powerful swings. The reward reflected several key objectives:

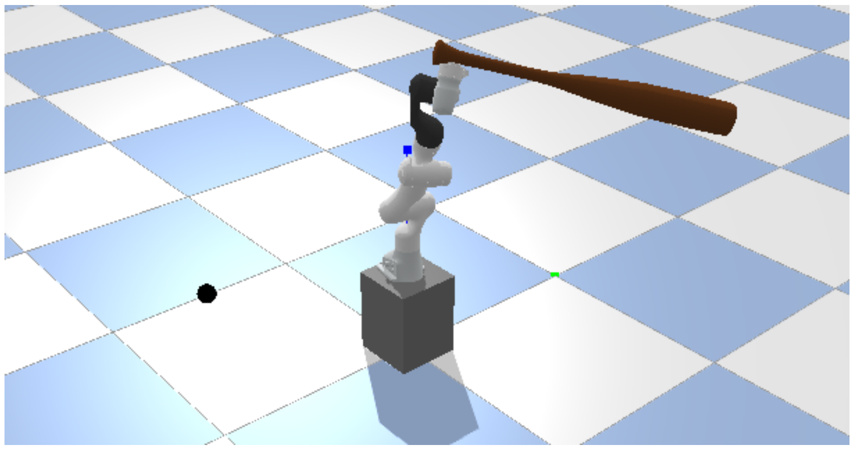


Figure 1: PyBullet Environment

making contact with the ball, maximizing bat speed and angular velocity at impact, and driving the ball to achieve desirable exit velocity and distance. The reward at each timestep was computed according to the following structure:

$$r_t = \begin{cases} w_{\text{hit}} + w_{\text{ball\_vel}} \cdot \|\vec{v}_{\text{ball}}\| + w_{\text{bat\_speed}} \cdot \|\vec{v}_{\text{bat\_tip}}\| + w_{\text{rot}} \cdot \omega_{\text{bat},z}, & \text{if ball is hit at } t \\ w_{\text{dist}} \cdot d_{\text{bat, ball}} + w_{\text{bat\_speed}} \cdot \|\vec{v}_{\text{bat\_tip}}\| + w_{\text{rot}} \cdot \omega_{\text{bat},z}, & \text{if ball not yet hit} \\ w_{\text{ball\_vel}} \cdot \|\vec{v}_{\text{ball}}\| + w_{\text{ball\_x}} \cdot |x_{\text{ball}}| + w_{\text{rot}} \cdot \omega_{\text{bat},z}, & \text{after ball has been hit} \end{cases} \tag{5}$$

In this expression, $d_{\text{bat, ball}}$ is the Euclidean distance between the bat tip and the ball, $\vec{v}_{\text{bat\_tip}}$ is the linear velocity of the bat tip, $\omega_{\text{bat},z}$ is the angular velocity of the bat around the vertical (Z) axis, $\vec{v}_{\text{ball}}$ is the linear velocity of the ball, and x_{ball} is the horizontal X-axis distance the ball has traveled.

Both SAC and PPO agents were trained using identical environment setups as the BC baseline but with this reward structure. This phase shifted the learning objective from purely imitating expert swing trajectories to discovering optimized swing strategies that maximize hitting performance and swing realism.

4 Experimental Setup

We conducted all experiments in a custom PyBullet simulation environment designed to model the baseball hitting task. The environment consists of a Franka Emika Panda robotic arm fixed to a raised platform to simulate the batter's torso, with a baseball introduced into the scene at a fixed initial position for each trial. The bat is attached to the robot's end effector, allowing the agent to control swing dynamics through its joint angles. The simulation operates at 100 Hz to emulate the sampling rate of most motion capture systems and capture the rapid motion involved in the swing and ball-to-bat interaction.

We used the reward function defined in Equation (5) as a unified evaluation metric across all agents, enabling consistent comparison of behavior cloning, PPO, and SAC performance. This allowed us to directly compare how well each agent learned to produce effective hitting behavior under a consistent evaluation framework.

5 Results

5.1 Behavior Cloning

Upon initial observation, we can see that the agent begins to degenerate over each iteration, as the reward steadily decreases (Figure 2). While the agent initially achieves a good reward by closely emulating the expert swing, performance degrades over time. This is likely because, once a bad action is taken, there is no corrective signal from an expert policy to steer the agent back onto a successful trajectory. As a result, deviations compound, especially in the later stages of the swing.

Visual inspections confirmed that the BC agent produced a functional swings within the earlier iterations of the training, but alignment errors emerged as the swing progressed, leading to a restricted swing inconsistent contact with the ball. Thus, although BC policy serves as a strong baseline for realistic motion generation, its inability to recover from compounding errors limits its robustness and generality.

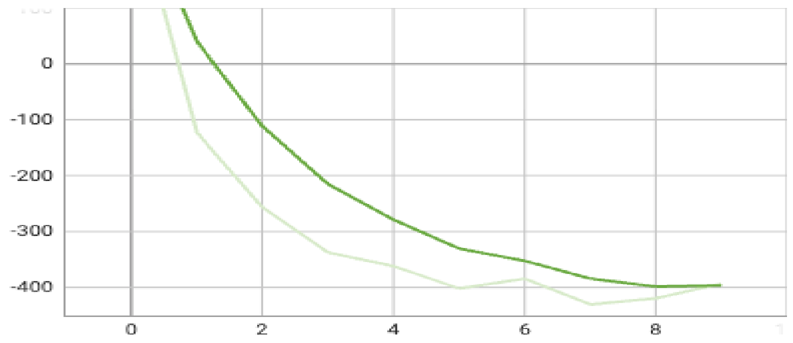
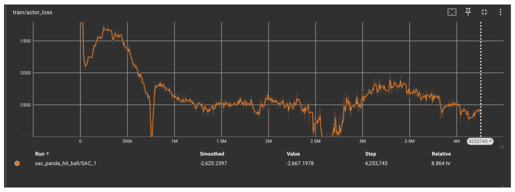


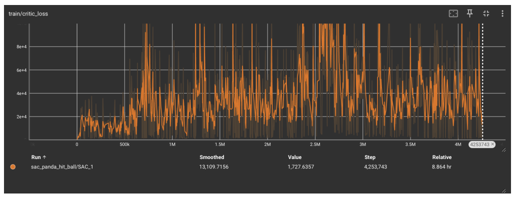
Figure 2: Behavior Cloning Average Reward (Evaluation)

5.2 Soft Actor-Critic (SAC)



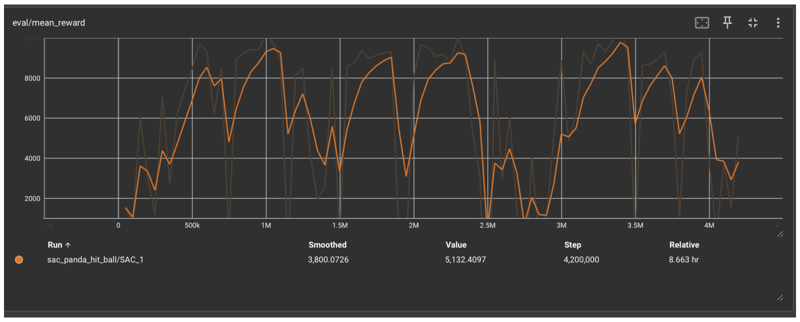
Run ↑	Smoothed	Value	Step	Relative
● sac_panda_hit_ball/SAC_1	-2,625.2397	-2,667.1978	4,253,743	8.864 hr

(a) SAC Actor Loss



Run ↑	Smoothed	Value	Step	Relative
● sac_panda_hit_ball/SAC_1	13,109.7156	1,727.6357	4,253,743	8.864 hr

(b) SAC Critic Loss



Run ↑	Smoothed	Value	Step	Relative
● sac_panda_hit_ball/SAC_1	3,800.0726	5,132.4097	4,200,000	8.663 hr

Figure 3: SAC Actor and Critic Losses (Top) and Evaluation Mean Reward (Bottom)

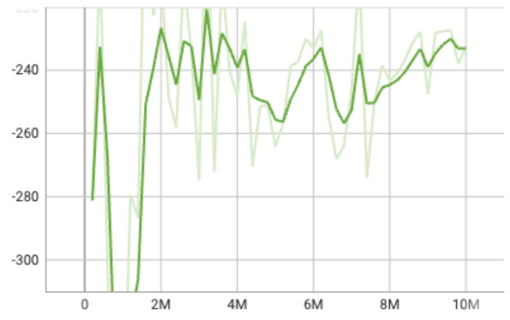
The random policy encouraged by SAC is innately sub-optimal as intuitively professional batters have a very similar swing trajectory each at bat just offset by a few inches. Additionally, stochasticity is generally not a suitable for a baseball swing where the slightest adjustment of a single inch can ultimately cause the bat to miss its target. Essentially, we would rather greedily iterate towards the trajectory than explore stochastically in the correct general direction because minor errors can quickly diminish major reward potential. Secondly, because our definition of the reward space includes euclidean distance from the bat to the ball, as well as the bat velocity, high reward generation is possible from any state, this fast random movement in the general direction of the ball is rewarded and enforced by SAC. This is because the policy can be perform very randomly while still receiving high reward, all the agent has to do while acting randomly is move as fast as possible and in the correct general direction. This behavior is observed when we rolled the policy out on a testing environment. The SAC agent achieved a high average episode reward of 10580.76, however physically inspecting the trajectories revealed that the agent was simply swinging the bat with high velocity in close vicinity to the ball. Although the agent was able to receive high rewards in the environment, it cannot be deemed as truly successful because the model learned how to swing quickly, just not at the correct target.

5.3 Proximal Policy Optimization (PPO)

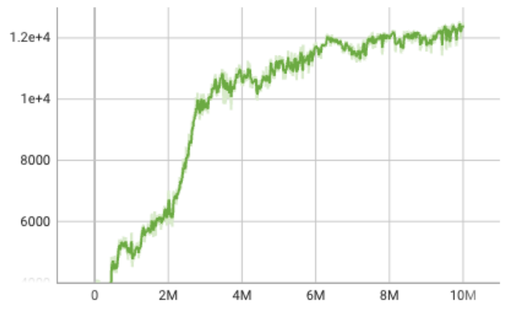
As shown in Figure 4a, the PPO agent's average training reward increased consistently and began to flatten smoothly, with stable increasing performance also observed during evaluation (Figure 4b). In contrast to the more erratic behavior observed with SAC, PPO exhibited smoother convergence dynamics, with fewer abrupt reward fluctuations across training iterations. Furthermore, PPO demonstrated steady learning progress even under our sparse and shaped reward structure, suggesting that its conservative update mechanism of clipping helped maintain robustness throughout training.

Furthermore, when visually inspecting the PPO agent's rollouts, we observed that it produced the most realistic, fluid, and fast swing motion of any method tested. The overall swing trajectory appeared biomechanically plausible and closely resembled a human baseball swing. Although the agent did not always achieve perfect contact with the ball, the motion itself exhibited strong potential, with controlled acceleration through the swing arc.

Ultimately, compared to SAC, PPO demonstrated greater consistency and more purposeful swing execution, with less random exploratory behavior during the swing. The PPO agent was able to generate visually convincing swings that could likely be further refined with additional training or reward shaping to improve hitting consistency.



(a) PPO Average Reward (Training)



(b) PPO Average Reward (Evaluation)

Figure 4: PPO Average Rewards for Training and Evaluation

6 Discussion

6.1 Behavioral Cloning

Interestingly enough, BC produces a policy that generates swing trajectories that resemble human batters far better than SAC. This is likely because humans swing baseball bat's in extremely greedy ways, so much so that new players do not learn how to swing by stochastically exploring which policies work the best, but by mimicking existing swing trajectories. Even in the MLB all players generally have similar swing mechanics.

However, we have two main limiting factors within BC. One is that observation's context, in the reprocessing phase of the project we batched all state data with the five previous observations to give the agent more context towards what the bat is doing at any given timestep. However, we left this a configurable parameter k so we could increase context in future experimentation, however due to computational limitations we were not able to raise it past $k = 5$. However, 5 time-steps is only .05 seconds real time which may not enough difference for the BC agent to extrapolate what part of the swing it is in. This is also why we do not see the bat move far from the prepared stance it takes initially, it does not know when to start swinging the bat when context is so limited. To fix this experimentation with higher k values is needed.

Additionally, we have a small dataset because we are limited to a small sample size of players, and data is only recorded when the player strikes the ball. To get a larger volume of data we should record all swings, not just when a ball is successfully hit. Then, we can represent the ball's location as the location of the bat's sweet-spot when the bat crossed into the strike zone. Increasing the dataset from 500 and increasing the agent's context will likely improve the model's robustness.

6.2 Soft Actor-Critic

As discussed briefly in the results section, SAC is not a good heuristic to learn a baseball swing. A baseball swing never benefits from stochasticity as it should always greedily use a trajectory that has been known to historically be effective. As mentioned above, SAC emphasizes a stochastic policy, and because high reward can be achieved by cheating the reward function (i.e. swinging as fast as possible) from any state, the policy does not actually get a chance to focus on the desired behavior of contacting the ball. This is evident from the actor and critic loss curves. Note that in 3a the actor is able to steadily decrease its loss from relatively high value. This shows that the actor is adequately learning how to find more rewards: albeit through a heavily exploitive strategy. However, note that 3b critic loss is extremely stochastic and does not meaningfully change throughout training. This is plausible because there is nothing for the critic to learn because no state—other than when the ball is contacted—is meaningfully different for future rewards than another. At all states the agent can choose to "cheat" the reward function which leads to infinite future rewards.

If we have intentions to continue to use SAC in the future, we must engineer a smarter reward function so that rewards are not sparse, but it is impossible to cheat the reward function to gain large rewards from any state. An example modification would be to give only the velocity in the direction of the ball a positive reward, and velocities in all other directions a negative reward.

6.3 Proximal Policy Optimization

When considering the quantitative and qualitative performance for our three agents, PPO outperforms the rest significantly. As mentioned when discussing Behavior Cloning above, while it is capable of generating functional swing mechanics, it ultimately suffers from a key limitation: the agent simply imitates expert motions without regard to whether the swing will successfully hit the ball in the current environment. This makes it the agent sensitive to small variations in ball position or timing, since the agent has no mechanism for adjusting its policy based on reward feedback. On the other hand, SAC explicitly optimizes for the reward function but introduces a level of exploration that is not suited for a task demanding fine motor precision with little room for error. The entropy term in SAC encourages the agent to maintain randomness in its policy, which works against the goal of producing a repeatable, high-accuracy swing.

In contrast to both, PPO's training dynamics align much more naturally with the needs of the baseball hitting task. Its clipped objective ensures that the policy improves in a stable and controlled manner, allowing it to refine swing motions without introducing large, destabilizing shifts from one update to the next. Thus, over time, the policy becomes a little more deterministic and more consistent, which is precisely what is required to achieve high-quality contact with the ball. Moreover, PPO's on-policy learning framework ensures that the agent is always adapting to its most recent behavior, which is particularly beneficial in a task where precise timing and coordination must be continually fine-tuned and adjusted. When visually inspecting PPO's rollouts, this difference was clear and immediate, for the agent developed a smooth, fluid, and powerful swing motion that, while not always achieving perfect contact, showed a level of control and realism unmatched by either Behavior Cloning or SAC. These qualities suggest that PPO provides a robust foundation for further refinement in the baseball hitting domain. However, there still lives room for improvement when considering other learning algorithms.

6.4 Future Work & Improvements

For future work to improve the realism of the learned baseball swing, an important direction is exploring model-based reinforcement learning (MBRL) methods. In our current approach, model-free algorithms such as PPO and SAC required large numbers of environment interactions to learn effective swing policies. Model-based approaches, which learn an explicit dynamics model of the environment, could enable more efficient learning by allowing the agent to plan and reason about future states. In particular, MBRL could help the agent better coordinate early swing posture and timing decisions with downstream impact outcomes by incorporating the learned dynamics of the environment. Additionally, incorporating hierarchical RL techniques, as demonstrated in the DeepMind table tennis system [7], could further improve performance by decomposing the task into modular low-level swing skills and a high-level controller to manage sequencing and timing. Such a hierarchy could be well-suited to the structure of the baseball swing, where early movements and reactive adjustments must be coordinated precisely to achieve optimal contact. Together, model-based and hierarchical methods offer a promising path toward building a more sample-efficient, generalizable, and biomechanically realistic robotic batter.

7 Conclusion

For this project, we explored the complex and challenging task of teaching a robotic arm to mimic a realistic and effective baseball swing using imitation learning and deep reinforcement learning techniques. We experimented with three different approaches—Behavior Cloning, Soft Actor-Critic, and Proximal Policy Optimization—and analyzed their quantitative performance and qualitative behavior. Although Behavior Cloning did imitate a generally functional swing trajectory, it struggled to adapt to the dynamic batting task in the absence of reward-based feedback. Additionally, it failed to emulate a swing realistically. Throughout training, Soft Actor-Critic exhibited instability and ultimately exploited the reward function in ways that did not produce fluid or meaning swing mechanics. However, PPO emerged as the most promising approach, as it learned to produce fluid and plausible swings with improved consistency and control despite the agent missing the ball several times. Therefore, our results suggest that the PPO agent offers a strong foundation for further refinement and improvement. For instance, incorporating model-based and hierachical RL methods and architectures may aid in improving sample efficiency, the adaptability of the agent, and the swing's realism. In conclusion, this project highlights and demonstrates the potential of utilizing imitation learning and deep reinforcement learning to tackle a short-episode long-horizon task such as hitting a baseball.

8 Team Contributions

- **Chase Joyner:** Environment Setup, Data Preprocessing and Analysis, and SAC
- **Mack Smith:** Environment Setup, Inverse Kinematics Optimization, Behavior Cloning, and PPO

Changes from Proposal We replaced the humanoid model with a simpler Franka Emika Panda robot and used public motion capture data of professional baseball swings instead of collecting our own with OptiTrack. We also shifted to working entirely in simulation, dropping original hopes for a perception model.

References

[1] Kyle W Wasserberger, Besky David M Jones BR Brady, Anthony C, and Boddy Kyle J. The openbiomechanics project: The open source initiative for anonymized, elite-level athletic motion capture data.

[2] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[5] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):1–14, July 2018.

[6] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals, 2020.

[7] David B. D'Ambrosio, Saminda Abeyruwan, Laura Graesser, Atil Iscen, Heni Ben Amor, Alex Bewley, Barney J. Reed, Krista Reymann, Leila Takayama, Yuval Tassa, Krzysztof Choromanski, Erwin Coumans, Deepali Jain, Navdeep Jaitly, Natasha Jaques, Satoshi Kataoka, Yuheng Kuang, Nevena Lazic, Reza Mahjourian, Sherry Moore, Kenneth Oslund, Anish Shankar, Vikas Sindhwani, Vincent Vanhoucke, Grace Vesom, Peng Xu, and Pannag R. Sanketi. Achieving human level competitive robot table tennis, 2025.