Catching Dynamics

CS 275 Artificial Life Course Project

Griffin Galimi[1], Soham Patil[2], and Jerry Yao[1]

[1]Computer Science Department, University of California, Los Angeles
[2]Mathematics Department, University of California, Los Angeles

June 2025

Abstract

This paper presents a framework for simulating dynamic catching behaviors with a 2-dimensional robotic arm in the MuJoCo physics engine. We investigate the emergence of adaptive strategies for intercepting and securing a moving object, a task that requires a tight coupling of perception and action analogous to that found in biological systems. Through reinforcement learning (RL), using Ray RLlib with a PyTorch backend, our agent autonomously develops effective catching policies from environmental interaction, rather than from explicit programming. This approach allows complex, timed behaviors to arise from simple reward signals, demonstrating how an agent's control system can adapt to its physical embodiment and dynamic environment. By constraining the problem to a 2D space, we isolate and analyze fundamental principles of timing, trajectory prediction, and manipulator control. This work contributes to understanding how sophisticated, seemingly intelligent behaviors can emerge from the interplay of learning, embodiment, and environmental physics, offering insights relevant to both robotics and the broader study of autonomous agents.

Contents

1 Introduction

1.1 Motivation

The ability to dynamically interact with moving objects is a hallmark of intelligent biological systems, yet it remains a significant hurdle for robotic platforms. The seamless hand-eye coordination exhibited by humans in tasks as simple as catching a ball involves sophisticated processes of prediction, timing, and feedback control that are challenging to replicate artificially. Mastering these dynamic manipulation skills is crucial for the next generation of robots designed to operate in unstructured and unpredictable environments. The development of robust catching capabilities has far-reaching implications, with potential applications in diverse fields such as assistive robotics, where a robot might catch a falling object for an elderly person; advanced manufacturing, for high-speed sorting and assembly; and human-robot interaction, enabling more natural and collaborative partnerships.

1.2 Definitions

To maintain a clear and focused scope for our investigation, we define our simulation within a 2-dimensional

Cartesian plane. All dynamic interactions between the robotic arm and the object occur within the XZ coordinate system. The 'X' axis represents the horizontal dimension, while the 'Z' axis corresponds to the vertical dimension, simulating the effects of gravity. This simplification to a 2D physics space allows for a more tractable analysis of the core challenges in dynamic manipulation, by reducing the dimensionality of the state and action spaces.s

1.3 Related Work

The pursuit of dynamic robotic manipulation has been a long-standing area of research. Early approaches often relied on precise analytical models of object trajectories and robot kinematics. While effective in constrained settings, these methods often struggle with the uncertainties and variations inherent in real-world scenarios. With the advent of deep learning, there has been a significant shift towards data-driven techniques, particularly reinforcement learning (RL) [7]. RL-based methods have demonstrated remarkable success in learning complex control policies for a variety of robotic tasks, including grasping, pushing, and in-hand manipulation. Our work builds upon these advancements, applying RL to the specific challenge of catching, which requires a higher degree of temporal precision and predictive capability than many static manipulation tasks [2]. We draw inspiration from research in both robotic grasping and the broader field of deep reinforcement learning for continuous control.

2 Functional Overview

2.1 Technologies

Our simulation framework is built upon a selection of open-source tools and libraries, chosen for their performance and widespread adoption in the robotics and machine learning communities. The core of our project is developed in Python, a versatile language with extensive support for scientific computing. The simulated environment is constructed using the Gymnasium and Gymnasium-Robotics libraries, which provide a standardized interface for reinforcement learning tasks [1]. The underlying physics simulation is powered by MuJoCo, a high-fidelity physics engine renowned for its speed and accuracy in simulating complex contact dynamics [6]. For the implementation and training of our reinforcement learning agents, we utilize Ray RLlib, a scalable and flexible library for distributed reinforcement learning [8]. The neural network models that form the basis of our learning agents are implemented using PyTorch, which serves as the backend for RLlib [5].

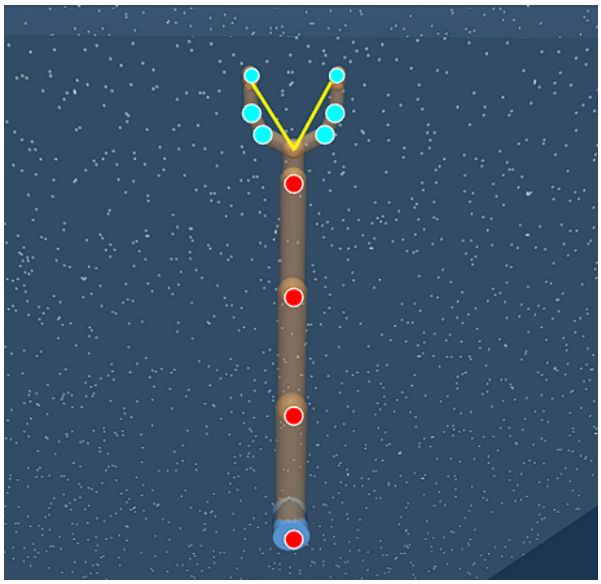

Figure 1: MuJoCo model of the arm. Red indicates motorized joints, blue denotes free joints, and yellow represents tendons.

2.2 Physical Model

The design of our simulated arm is based on the manipulator model from the *dm_control* suite, adapted for our catching task. The arm (Figure 1) possesses 5 degrees of freedom (DoF), providing a high level of dexterity within its planar workspace. Its movement is controlled by 4 motors, which actuate the root, shoulder, elbow, and wrist joints. This multi-joint setup allows for a wide range of motion and precise positioning of the end-effector. The grasping mechanism is implemented using a single tendon, which provides a simple yet effective means of securing the target object.

2.3 Environments

In our simulation, the robotic arm is equipped with a concave, bowl-like container affixed beneath its end-effector, serving as the primary tool for catching falling objects. This container is rigidly attached and integrated into the agent's morphology, enabling the arm to scoop or cradle the object mid-flight. The addition of this catching vessel not only enhances the realism of the task by more closely mirroring biological grasping mechanisms, but also increases the precision required in timing and control. Successful interception demands the coordination of the arm's movement with the object's trajectory, ensuring that the bowl is positioned to absorb the object's momentum upon impact. This physical structure also introduces new dynamics, such as rotational effects and angular constraints, which the policy must learn to account for during training. The full environment is seen in Figure 2.

Beyond the single-arm setup, we have designed a dual-

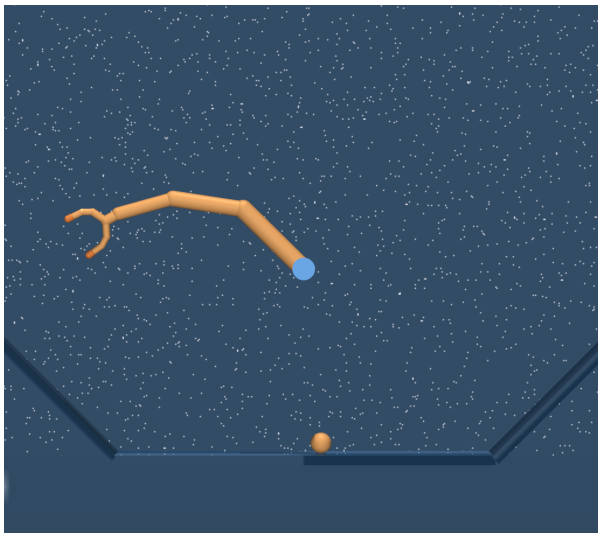

Figure 2: Full single arm environment illustrating arm, ball, and container.

arm extension of the environment to support future exploration of more complex interaction paradigms. In this configuration, each arm can be independently controlled, enabling scenarios in which one agent throws the object while the other attempts to catch it. This shift transforms the task from a purely reactive control problem into a multi-agent learning challenge, where timing, coordination, and possibly even prediction of another agent's behavior become crucial. Such a setup lays the groundwork for studying collaborative dynamics or adversarial strategies within a shared physical space, expanding the scope of learning from individual embodiment to multi-agent interaction.

3 Locomotion

3.1 Task Setup

In order to efficiently train two agents to play catch, we devised an incremental learning process involving an environment where one arm learns to catch a ball, **SingleCatchEnv**, and an environment where one arm learns to throw a ball, **SingleThrowEnv**. The motivation behind this is that if we have a competent policy in each respective environment, then in the game environment, we could train a simple master controller to determine whether an agent should follow the catch or the throw policy. We mainly focused on the reward formulations of these environments that would lead to our desired behaviour. To train the agents, we used PPO and a feedforward neural network of 2 layers. The inputs of the neural network include the arm's joint angles and velocities, and the ball's position and velocity.

The catching environment starts with a ball in the opposite bucket being "shot" randomly towards the bucket containing the arm. This emulates a throw from what will eventually be the opposite arm. Similarly, the throwing environment starts with the ball in the grasp of the arm, the goal being to throw the ball to the other bucket.

3.2 Catching

3.2.1 Reward Formulation

To incentivize the arm to catch a ball, we consider the arm's position and the wrist's angle with respect to the ball, and whether the palm is touching the ball.

At an arbitrary timestep, we denote $\vec{c}$ as the fixed center of the arm, $\vec{x}$ as the position of the arm's hand, $\vec{b}$ as the ball's position, and R as the maximum radius of the arm. With these variables, we can define where exactly the arm should be to minimize its distance from the ball:

$$\vec{x}_{target} = \vec{c} + \min(\|\vec{b} - \vec{c}\|, R) \cdot \frac{\vec{b} - \vec{c}}{\|\vec{b} - \vec{c}\|},$$

so that when the ball is outside the arm's reach, the target position is on the circumference, and when the ball is within the arm's reach, it is simply the ball. Then we define the distance reward as

$$r_{\text{dist}} = -\|\vec{x}_{target} - \vec{x}\|.$$

To reward the wrist alignment, denote θ_{ball} as the angle of the velocity of the ball, θ_{wrist} as the angle of the wrist, and θ_{h2b} as the angle from the hand to the ball. Then we separate the wrist reward into three parts: if the ball is already in the hand, we don't worry and set $r_{wrist} = 0$. If the ball is still in its trajectory, we want the wrist to face opposite of the ball's velocity so that the ball smoothly lands in the hand; so we set $r_{wrist} = -|\theta_{wrist} + \theta_{ball}|$. However, if the ball is below the arm and on the bucket, we want the wrist to directly face the ball, so that it can be picked up; then we set $r_{wrist} = -|\theta_{wrist} - \theta_{h2b}|$.

Finally, just to nudge the arm to more precisely catch the ball, we give an extra bonus of 10 whenever the palm is touching the ball.

3.2.2 Results

We initially started training with just the distance reward, and the arm quickly learned to move to the ball and track it as it rolls around the bucket. The wrist reward that we formulated seemed to be easy to learn for the arm, as it also quickly learned to face opposite the ball's velocity in its trajectory, and straight down at the ball when it fell. The hope was that this would make it easier to happen upon catching the ball. Unfortunately, we found that even when the arm got the ball to bounce on its palm many times during training, the catching process was too precise for the arm to learn in time.

3.3 Throwing

3.3.1 Reward Formulation

To incentivize the arm to learn a throwing behavior, we initially designed a sparse reward function that granted a large bonus if the ball achieved a sufficient forward velocity and landed near a target position. However, this approach proved ineffective: the agent struggled to discover meaningful strategies due to the lack of intermediate feedback or "stepping stones" during early exploration. We found that learning was greatly improved by introducing a denser, shaped reward that provided more frequent signals throughout the throw.

The final reward formulation was based on two main components: the ball's velocity, r_{vel}, and its horizontal position. The first term, rewards the agent for increasing the ball's velocity in the positive x direction:

$$r_{\text{vel}} = v_x$$

The second term, r_{dist}, rewards the horizontal displacement of the ball, i.e., how far the ball has traveled forward. This term is scaled more heavily if the ball has left the hand (i.e., is no longer in contact with any part of the gripper), rewarding letting go of the ball after what we hoped to be a sufficient amount of acceleration:

$$r_{\text{dist}} = \begin{cases} 1.5 \cdot x_b & \text{if } x_b > 0.5, \text{ ball not in contact w/ hand} \\ x_b & \text{if } 0.25 < x_b \leq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

3.3.2 Results

In the early stages of training, the arm learned to gently grasp and toss the ball forward, primarily driven by modest rewards for achieving positive xxx-velocity. This baseline behavior is demonstrated in the *Early Throwing Toss* video, where the arm consistently executes low, controlled throws. As training progressed, we modified the reward function to place greater emphasis on the ball's forward displacement rather than just its velocity. This adjustment led to the emergence of a novel behavior: the arm began to perform a **pinch launch**, using a sudden release of its fingers to catapult the ball forward with significantly greater range. This behavior is captured in the *Throwing Pinch* video. While these high-distance throws were not entirely consistent—being highly sensitive to the ball's exact position within the grasper—they represent a meaningful adaptation in strategy. Notably, the agent retained its ability to default to the original toss if a pinch launch failed and the ball remained in the grasper, suggesting a form of conditional fallback behavior. This progression highlights how relatively simple reward shaping can lead to diverse and situationally adaptive motor strategies. Footage of several runs of the final behaviour can be observed in the *Full Throwing Demo* video. Future work simulating more realistic throwing motions involving retraction and release, moving beyond the pinch-launch mechanism to better approximate human-like throwing dynamics.

4 Perception

A key challenge in natural catching lies in the tight coupling between perception and motor control. Just as biological systems must interpret noisy, partial sensory inputs to make real-time decisions, our agent must translate visual information into spatial predictions suitable for reactive behavior. Inspired by visual processing in animals, particularly the separation of sensory estimation and memory-based inference, we designed a hierarchical perceptual system that reflects this natural division. Our approach involves decomposing the perception task into spatial recognition, temporal inference, and a learned fusion mechanism that adaptively integrates the two.

To simulate the role of vision in a grounded yet abstract way, we employ a 1D ray-based sensing system from a fixed camera perspective. An example of these captured 1D images can be seen in Figure 4 over time. These ray-traces form the core of our visual input, analogous to retinal photoreceptors scanning a narrow horizontal band of the environment. This deliberately constrained setup enables controlled analysis of how much information an agent truly needs to make accurate spatial predictions, and whether it can generalize in the face of incomplete or degraded observations.

4.1 Ray-Based Vision

The agent perceives its environment through a fixed, forward-and-down-facing camera that emits a uniform array of $n = 107$ rays across its field of view. Each ray returns a scalar value representing the distance to the first object it intersects, effectively creating a 1D "image" that encodes spatial structure without relying on traditional RGB inputs. This ray representation can be thought of as an abstract and simplified analog of retinae, distilling vision into a minimal yet sufficient form.

This compact visual encoding serves two purposes. First, it reduces the complexity of the input space, focusing the learning problem on how to transform raw spatial measurements into usable control signals. Second, it allows us to examine perception as a learned process rather than one dictated by architectural priors—an important distinction when aiming to replicate adaptive, biological-like strategies. The ray-based observation model forces the network to implicitly learn properties like depth, object motion, and spatial continuity from raw geometric signals.

4.2 Neural Perception

To interpret the ray-based input, we employ a convolutional neural network (CNN) that serves as a spatial recognition module. The CNN maps the 1D array of

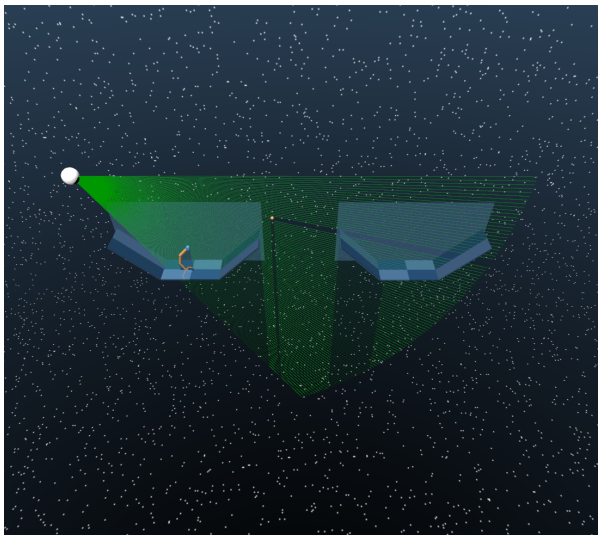

Figure 3: Ray traces pictured as green lines. The fixed camera is 'eyeball' in top left–the origin for the rays.

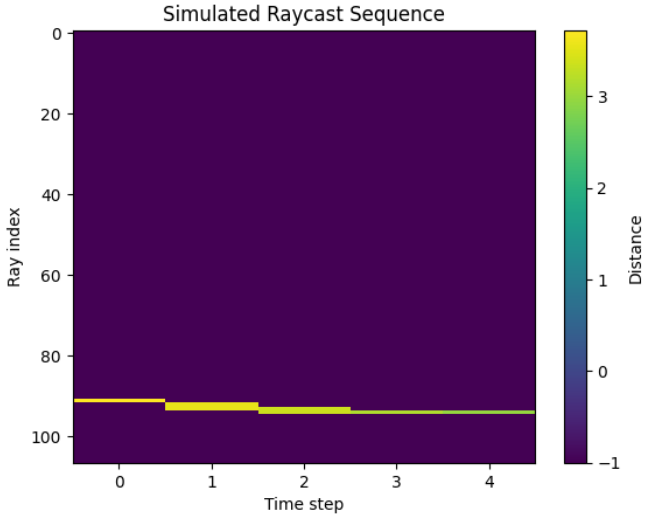

Figure 4: Each timestep represents the captured image. You can see the ball move through space as different rays capture the yellow color.

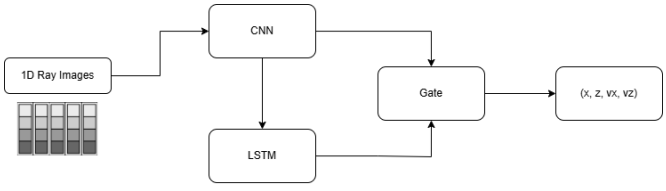

Figure 5: Network hierarchy

ray distances to estimated (x, z) coordinates of the target object, providing a spatial "snapshot" at each timestep. This output represents the instantaneous visual belief of the object's location, and is not informed by previous frames, thereby acting as a purely perceptual estimate.

In parallel, we deploy a long short-term memory (LSTM) network that models temporal inference. By consuming the sequential outputs of the CNN, the LSTM learns to predict not only the position (x, z) but also the velocity (v_x, v_z) of the object. This recurrent pathway captures patterns over time, such as ballistic motion, occlusions, or temporary visual dropouts. The LSTM thereby functions as a memory-driven inference system, approximating the internal simulation of object dynamics that humans rely on when visual input is compromised or absent. The full network hierarchy is seen in Figure **??**.

4.3 Gating Mechanism

To unify these two subsystems—perception and memory—we introduce a learned gating network inspired by attentional mechanisms and perceptual confidence estimation in biological systems. The gating network receives as input both the CNN's current output and a hidden state from the LSTM, and produces a scalar gating value $G \in [0, 1]$ via a sigmoid activation. This value is then used to fuse the spatial predictions according to the equation:

$$S_{\text{final}} = G \cdot V_{\text{spatial}} + (1 - G) \cdot L_{\text{spatial}}$$

where V_{spatial} is the CNN-derived position and L_{spatial} is the LSTM-derived position.

The intuition behind this formulation is to allow the system to fluidly transition between perceptual reliance and memory-driven inference, depending on the quality and clarity of the visual input. In situations where the object is fully visible and the ray-based signal is clear, the network can learn to favor the CNN's direct spatial estimate. Conversely, during occlusion or ambiguous visual input, the system can shift weight to the LSTM's internally modeled predictions. This gating mechanism is trained jointly with the rest of the model in an end-to-end manner, and allows for emergent behaviors akin to "closing the eyes" and internally simulating the motion of the object.

4.4 Theoretical Framing

This modular perception architecture reflects a broader ambition: to investigate how biologically inspired principles can be synthesized in artificial agents to produce adaptive, context-sensitive behavior. The separation between spatial recognition (CNN) and temporal inference (LSTM) draws a theoretical parallel to the division of labor in biological perceptual systems, such as the dorsal and ventral streams in primate vision, or the distinction between perception and mental simulation in humans.

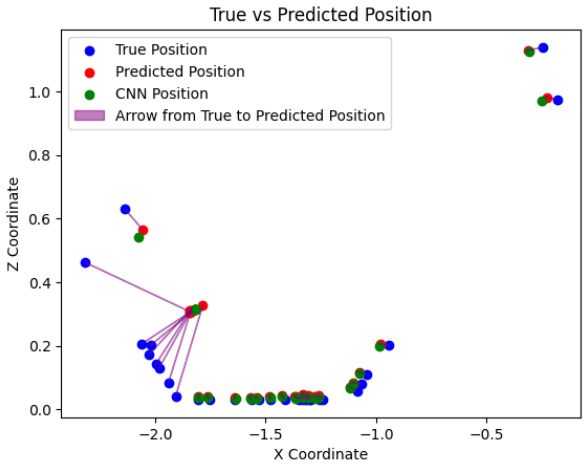

Figure 6: Example of the predicted positions of model run *without* context.

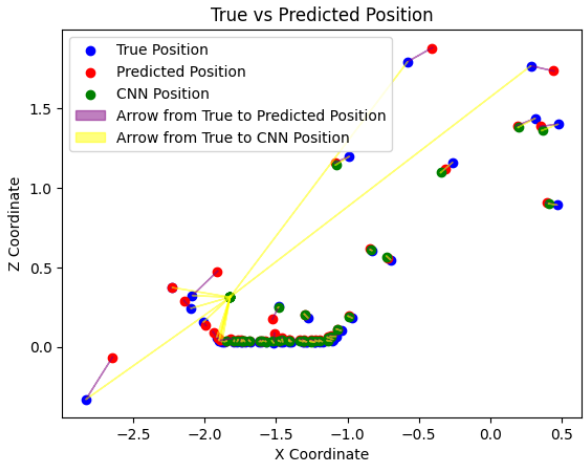

Figure 7: Example of predicted positions of model run *with* context

The gating mechanism adds an additional layer of cognitive flexibility, reminiscent of how animals modulate attention or confidence based on environmental reliability. From a theoretical standpoint, this approach views perception not as a monolithic input channel but as a dynamic integration process—a function of signal quality, task demands, and internal expectations. In doing so, it bridges the gap between low-level sensory encoding and high-level control, and enables the emergence of strategies that are robust to noise, ambiguity, and temporally extended uncertainty.

4.5 Results

As seen in Figure 6, in the bottom left, when the ball is not visible (outside of range of rays), the prediction is not super accurate. That said, when given the ball position context over time, calculated from the CNN when it is not blinded and from the LSTM when the ball is not visible, the predicted position is much closer to the true position as the model does give motional inference (this can be seen in the vision demo video). We also see this in Figure 7 where the yellow distance between the true and CNN positions are much greater than the difference in true and final predicted positions that use this context. There is definitely room for improvement in training–specifically altering how the model is trained when it is 'blind,' which could help the CNN-LSTM combination drift a little less when the ball is not visible. This drift, however, is expected as an equivalent happens in real life when we close our eyes; our estimated positions shift over time and may not be very accurate after a while.

5 Summary

This project presents a reinforcement learning framework for teaching a 2D robotic arm to catch and throw a moving object within the MuJoCo physics simulation environment. Emulating the sensorimotor coupling seen in biological systems, the agent learns to develop timing-sensitive control policies through interaction with its environment, rather than explicit programming. The setup leverages PPO in Ray RLlib with a PyTorch backend, and features a multi-joint manipulator equipped with a catching vessel, allowing for precise interception tasks.

Training is divided into two environments—one for catching and one for throwing—using reward shaping to encourage successful behavior. For perception, a biologically inspired architecture is introduced: a fixed camera provides 1D ray-based inputs, processed by a CNN for spatial localization and an LSTM for temporal inference. A learned gating mechanism fuses these outputs, dynamically weighting visual and memory-based predictions to enhance robustness. This perceptual-control system models principles like attentional flexibility and mental simulation, offering a compelling bridge between artificial and natural intelligence in dynamic manipulation tasks.

6 Future Work

While this project lays a foundation for dynamic catching and throwing in a 2D simulated environment, several promising directions remain open for exploration.

One major extension involves integrating the currently separate catching and throwing agents into a unified, end-to-end learning system capable of coordinated behavior. Training both behaviors together in a shared environment would enable the emergence of strategies such as self-correction, anticipatory positioning, and coordinated timing—key elements in games like catch or collaborative handoffs. This would also introduce a more complex control hierarchy, potentially involving a high-level decision module that switches between catching and throwing behaviors depending on context.

Expanding the visual system is another promising avenue. Our current setup uses a fixed, ray-based "eyeball" for 1D perception, which, while effective for basic inference, limits the richness of spatial understanding. Introducing a dynamic eye—one that can rotate or move independently of the body—would open up new challenges in active perception and sensorimotor coordination (building off similar work to [3]). Furthermore, generalizing the simulation to a 3D environment would better reflect real-world dynamics and significantly increase the complexity of both perception and control.

Another compelling direction is the incorporation of adversarial or competitive scenarios. For instance, two agents could be trained in opposition—one attempting to catch and the other trying to evade or misdirect. Such settings could foster more sophisticated strategic behaviors, encourage generalization, and provide insights into game-theoretic dynamics in embodied agents.

Finally, replacing the motorized joints with a muscle-based actuation system could add another layer of realism and biological fidelity. Muscle-based control would introduce non-linear force dynamics, energy constraints, and time delays, which would necessitate fundamentally different learning strategies, which builds off of much of the Artificial Life research such as [4]. This shift could provide a closer parallel to animal biomechanics and deepen the connection between artificial agents and their natural counterparts.

Collectively, these directions aim to move the project closer to embodied intelligence that is not only reactive but anticipatory, coordinated, and adaptive across increasingly realistic environments.

7 Appendix

1. Source Code: `https://github.com/griffing52/catching_dynamics`

References

[1] Rodrigo de Lazcano et al. *Gymnasium Robotics*. Version 1.3.1. 2024. URL: `http://github.com/Farama-Foundation/Gymnasium-Robotics`.

[2] Ngan Le et al. *Deep Reinforcement Learning in Computer Vision: A Comprehensive Survey*. 2021. arXiv: `2108.11510 [cs.CV]`. URL: `https://arxiv.org/abs/2108.11510`.

[3] M. Nakada et al. "Biomimetic Eye Modeling and Deep Neuromuscular Oculomotor Control". In: *ACM Transactions on Graphics* 38.6 (Nov. 2019). Proc. ACM SIGGRAPH Asia 2019 Conference, Brisbane, Australia, 221:1–221:14. DOI: `10.1145/3355089.3356538`.

[4] Masaki Nakada et al. "Deep learning of biomimetic sensorimotor control for biomechanical human animation". In: *ACM Transactions on Graphics* 37.4 (Aug. 2018). Proc. ACM SIGGRAPH 2018 Conference, Vancouver, BC, August 2018, 56:1–56:15. DOI: `10.1145/3197517.3201304`.

[5] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: `1912.01703 [cs.LG]`. URL: `https://arxiv.org/abs/1912.01703`.

[6] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: `10.1109/IROS.2012.6386109`.

[7] Saran Tunyasuvunakool et al. "`dm_control`: Software and tasks for continuous control". In: *Software Impacts* 6 (2020), p. 100022. ISSN: 2665-9638. DOI: `https://doi.org/10.1016/j.simpa.2020.100022`. URL: `https://www.sciencedirect.com/science/article/pii/S2665963820300099`.

[8] Zhanghao Wu et al. "RLlib Flow: Distributed Reinforcement Learning is a Dataflow Problem". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2021. URL: `https://proceedings.neurips.cc/paper/2021/file/2bce32ed409f5ebcee2a7b417ad9beed-Paper.pdf`.