Learning to accurately throw Paper Planes using Reinforcement Learning

Marcus Kornmann[1] and Qimeng He[2]

***Abstract*— This paper presents a novel approach to optimizing paper plane throwing using reinforcement learning and trajectory encoding. We introduce a method that combines a Variational Autoencoder (VAE) for encoding paper plane trajectories with a Soft Actor-Critic (SAC) algorithm to learn optimal throwing strategies. Our approach dynamically adapts to the unique aerodynamic properties of randomly generated paper plane designs. We explore two training scenarios: a one-step setting with a plane store and a multi-step setting with episode-based trajectory accumulation. Our experiments show that incorporating information from previous throws improves performance, particularly when generalizing to unseen plane designs.**

I. INTRODUCTION

Robotic manipulation tasks often involve complex dynamics and control challenges. One such intriguing task is for a robot to accurately throw paper planes at a target. Despite its apparent simplicity, this task requires accurate determination and execution of initial launch conditions, considering the aerodynamic properties and inertial parameters of the paper plane. The challenge of robotic paper plane manipulation has attracted recent attention in the field. For instance, Tanaka et al. [1] demonstrated a robots' ability to fold paper planes. However, the precise launching of these planes remains an open problem. This task encapsulates core difficulties in robotics: real-time adaptation to object properties and precise control of launch parameters. By addressing these challenges, we aim to advance techniques applicable to a wide range of dynamic object manipulation tasks. This work focuses specifically on learning the initial conditions required for successful throws. To address this challenge, we introduce an episodic reinforcement learning (RL) framework that aims at determining the optimal initial velocity and orientation for launching the plane. It learns from observed trajectories of past throws, utilizing a MuJoCo simulation environment [2] as illustrated in Figure 1. Key aspects of our approach include:

1) RL Environment: We design a RL environment in which the robot's actions, setting the plane's initial velocity and orientation, determine its trajectory.
2) Domain Randomization: To improve generalization and facilitate potential transfer to real-world robotic systems, we randomize the target's position and the plane's properties, ensuring robustness across different conditions and reducing the sim-to-real gap.

[1]Development of the learning environment and model training
[2]Randomization of paper planes

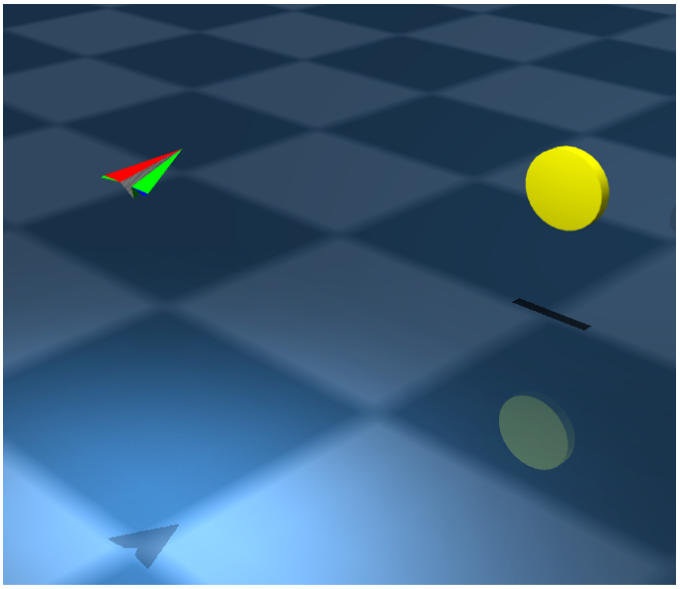

Fig. 1: In MuJoCo we generate a paper plane and set the relative position with target randomly within a certain range. Our goal is to throw the paper plane as accurately as possible onto the target (yellow disc).

3) Latent Trajectory Embeddings: A Variational Autoencoder (VAE) [3] generates latent embeddings of the paper plane's past trajectories. These embeddings capture patterns in the plane's flight behavior, which indirectly reflect its aerodynamic properties without explicitly modeling them. By learning from these trajectory patterns, the robot can adapt its throwing technique to handle each specific plane more effectively.
4) Soft Actor-Critic (SAC) Algorithm [4]: We employ the SAC algorithm to learn the optimal initial conditions for launching the paper plane. Specifically, this involves determining the ideal initial orientation and velocity of the plane based on the current configuration of the environment and the latent embeddings of past trajectories.

By leveraging knowledge from previous throws, we can train an agent capable of handling a variety of paper planes. This approach eliminates the need for plane-specific agents. Most importantly, it enables our agent to adapt to unseen paper planes, using prior throws with the same plane as references.

II. Background

Our approach to the paper plane throwing task integrates two key machine learning techniques: the Soft Actor-Critic (SAC) algorithm for reinforcement learning and Variational Autoencoders (VAEs) for efficient representation learning. This section provides context on how these methods contribute to our solution.

A. Soft Actor-Critic Algorithm

The Soft Actor-Critic (SAC) algorithm [4] is a reinforcement learning method designed for continuous action spaces. SAC is an off-policy algorithm, meaning it can learn from previously collected experiences, which often leads to improved sample efficiency compared to on-policy methods. It is built upon the maximum entropy reinforcement learning framework, which encourages the agent to behave as randomly as possible while still maximizing rewards. This approach helps in exploration and robustness to environmental changes. Empirically, SAC has been shown to outperform other state-of-the-art model-free deep RL methods, such as the off-policy Deep Deterministic Policy Gradient (DDPG) algorithm [5] and the on-policy Proximal Policy Optimization (PPO) algorithm [6], particularly in tasks requiring extensive exploration and fine control [4].

B. Variational Autoencoder

Variational Autoencoders (VAEs) [3] are a class of generative models that learn to encode high-dimensional data into a lower-dimensional latent space and then decode it back to the original space. VAEs combine ideas from variational inference and neural networks to create a powerful tool for unsupervised learning and representation learning. The key innovation of VAEs lies in their ability to learn a probabilistic mapping between the data space and a latent space. This probabilistic approach allows VAEs to capture uncertainty and variability in the data, making them particularly useful for tasks involving complex, high-dimensional data distributions. VAEs have found applications in various domains, including image and text generation [7, 8], fault detection [9], and representation learning for downstream tasks. Their ability to learn meaningful, compact representations of complex data has made them a valuable tool in many machine learning pipelines, especially when dealing with large, unlabeled datasets.

III. Related Work

Since its inception, MuJoCo has quickly become a powerful tool for simulating robotics and studying reinforcement learning algorithm [2]. It is convenient to creat or modify environment in MuJoCo and it performs well in exploring reinforcement learning techniques [10].

A. Simulation Environment and Domain Randomization in MuJoCo

MuJoCo is an open-source physics engine that generates fast and accurate simulations of articulated structures interacting with their environment [2]. It is widely used in various fields such as biomechanics, graphics, and especially robotics. Compared to other physics engines and simulation tools like DART, Bullet, and PhysX, MuJoCo is both the fastest and most accurate for constrained systems relevant to robotics [11]. Therefore, it is a suitable platform for simulating the task of paper airplane throwing with a robot and building reinforcement learning environments. After modeling the paper plane, it is also important to randomize the parameters of the model in order to generate a diverse dataset for the reinforcement learning algorithm. Standard domain randomization is widely used in simulation, but it often results in high variance data [12]. An alternative strategy is residual physics [13], which generates a dataset composed of two parts: analytical models and compensations for unknown physical phenomena. Similar to this approach, we first develop a base model in simulation and then incorporate physics residuals for specific parameters of interest.

B. Reinforcement Learning in Throwing

Reinforcement learning (RL) has emerged as a primary tool for tackling the challenge of enabling robots to throw objects accurately. There are numerous examples where reinforcement learning has been effectively applied to object throwing, such as Dart-throwing robot [14], Ball-throwing Robot [15] and TossingBot [13], which demonstrate its potential to improve the accuracy of robotic object-throwing through continuous interaction with the environment. For a dart-throwing robot, the environmental factors such as aerodynamics have relatively little impact on the dart motion. As for the TossingBot, it focuses on the joint learning of grasping and throwing arbitrary objects. Reinforcement learning is also widely employed to solve locomotion problem such as quadrupedal locomotion over uneven terrain [16, 17] and performs excellently in generating a feedback-based trajectory tracking system [18].

IV. METHODS

This section details our approach to training a reinforcement learning agent to accurately throw paper planes. We begin by describing our domain randomization technique for generating diverse paper plane models. We then explain how we handle different plane behaviors using a Variational Autoencoder (VAE) to encode trajectory information. Next, we formulate the problem as a Markov Decision Process (MDP) and describe our chosen reinforcement learning algorithm. Finally, we outline our training process and the two distinct scenarios we use to evaluate our approach.

A. Domain Randomization

Even if you fold two paper planes using the same procedure and paper, there will still be slight differences between them, and often, the two sides of the same paper plane are asymmetrical. Therefore, to simulate the act of throwing a paper plane more realistically, we randomize parameters of a symmetrical paper plane model to generate asymmetry. Here we choose some physical properties such as the position of center of mass (CoM), the angle and the size of the wings and

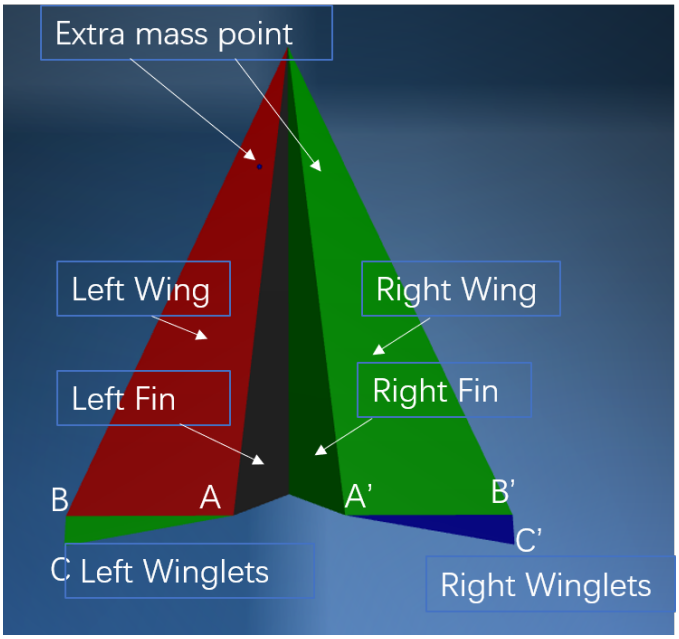

Fig. 2: Paper plane model generated in MuJoCo randomly based on a standard symmetric paper plane. We change the coordinates of vertices of paper plane to change the length and angle of wings and winglets asymmetrically, and change position of extra mass point to get asymmetrical inertia.

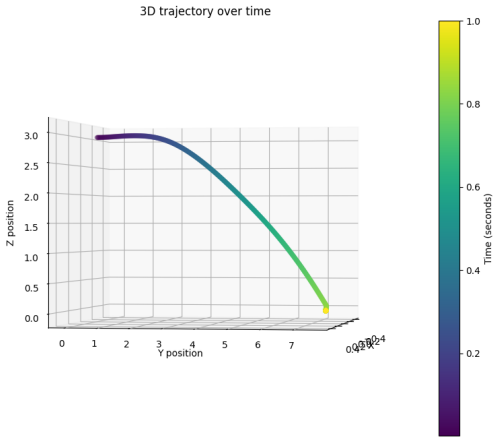

Fig. 3: The trajectory of throwing an symmetrical paper plane model, the initial location is (0,0,3m) and initial vilocity is (0,10m/s,0), model is rotated 40° around X axis which as its intial orientation.

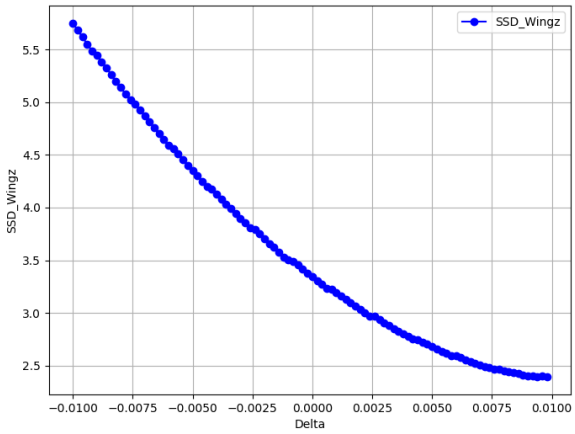

Fig. 4: Illustration of the sum of squared distances between the wing's changing trajectory and the base trajectory under the same initial conditions. The unit of delta is meter.

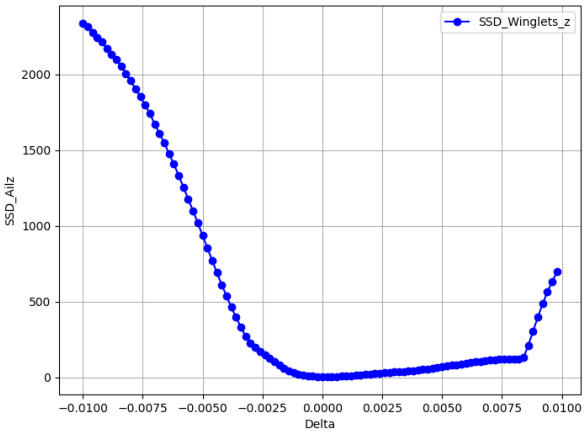

Fig. 5: Illustration of the sum of squared distances between the winglets' changing trajectory and the base trajectory under the same initial conditions. The unit of delta is meter.

winglets (small hinged sections on the outboard portion of a wing). The MuJoCo model of paper plane is shown in figure 2. We change the coordinates of extra mass point to change the intertia, change the coordinates of some vertices(A, A', B, B', C, C') of paper plane model to change the angle and size of wings and winglets. We aim to training the throw of as many different types of paper planes as possible without concern for the likelihood of these planes appearing in real life, so we use uniform distribution to randomize these parameters. Before randomizing the parameters we simulate the throwing with different model parameters in the same initial conditions, such as initial velocity and location, to see how these parameters influence the trajectory. In Figure 3 we can see the trajectory when throwing a symmetrical paper plaen model. In order to represent the data more compact, we

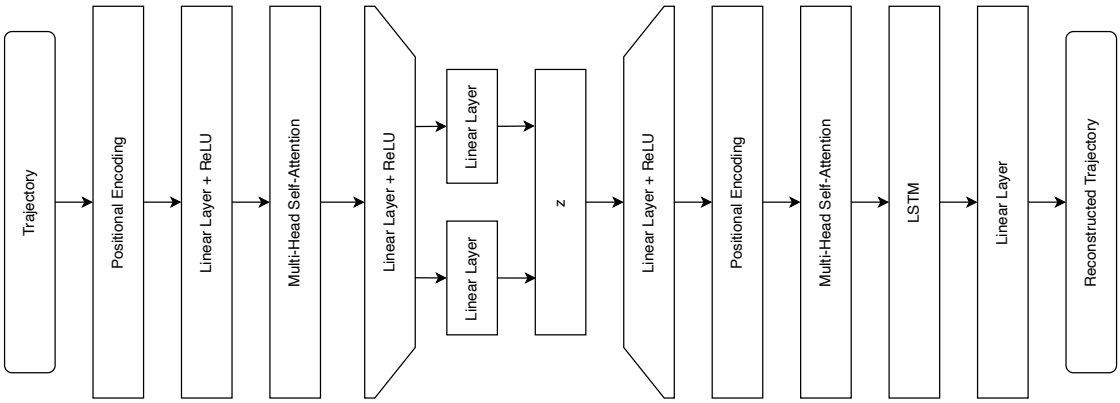

Fig. 6: Architecture of the Variational Autoencoder (VAE) used for learning latent representations of paper plane trajectories. The encoder (left) processes variable-length input sequences using sinusoidal positional encoding, multi-head attention, and fully connected layers to generate the latent distribution $\boldsymbol{z}$. The decoder (right) takes a sample $\boldsymbol{z}$ from the latent distribution and reconstructs the input sequence using positional encoding, multi-head attention, an LSTM layer, and a final linear layer. The VAE architecture enables the model to handle trajectories of different lengths and capture the essential aerodynamic behavior of the paper planes in the latent space.

compute the sum of squared distance(SSD) of coordinates of two different trajectory to observe the overall impact of parameter chaging on the flight trajectory. The initial Z axis of B and B' is 0.05m, when we change it in a range (0.04,0.06), we can observe that overall the values of SSD is small and the impact of the plane's wings bending downward is greater than that of bending upward. But when we change the height of winglets, value of Z axis of C and C', the impact is more remarkbale. Here the changing range is (0.043,0.063), in Fig 5 we can see that if we bend down or up the winglets out of a small range,especially the situation of bending down, the trajectory will change a lot.

For stable flight, we try to fold symmetrical paper planes as much as possible, but it is difficult to achieve perfectly. Therefore, our parameter randomization follows a similar approach: Based on a symmetric model we add a small amount of noise to parameters which obeys a uniform distribution. The position of the target is sampled from a variable circle around the origin defined by a 3-tuple $(r_{\text{target}}, \alpha_{\text{target}}, z_{\text{target}})$ where $r_{\text{target}} \sim \mathcal{U}(4, 10)$ [meters] is the radius of this circle, $\alpha_{\text{target}} \sim \mathcal{U}(-30, 30)$ [degrees] is the angle to the target relative to a reference orientation and $z_{\text{target}} \sim \mathcal{U}(0.5, 2.2)$ [meters] is the z-coordinate of the target. The interaction between these parameters can be seen in figure 7.

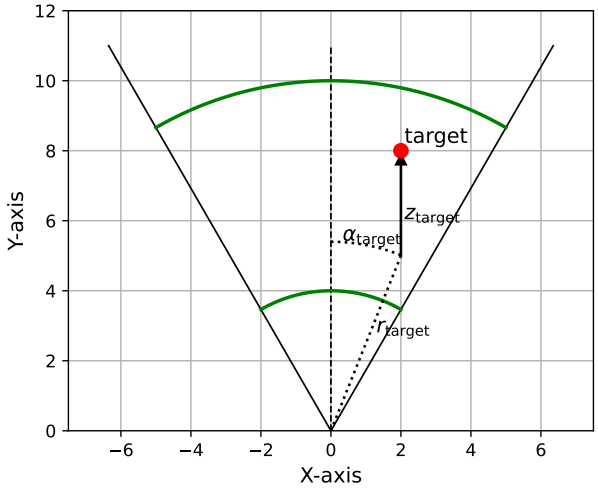

Fig. 7: Illustration of the constraints and parameters defining target positions. The target always lies within the area bounded by the angle range (depicted by black diagonal lines) and the radius limits (shown as green circular arcs). Each target point is uniquely characterized by three parameters: its angle from the y-axis (α), its radial distance on the x-y plane, and its height (z) above this plane.

B. Handling different plane behaviors

Due to the randomization of plane properties, each plane exhibits unique aerodynamic behaviors. To account for this variability, the agent is designed to learn about the plane's behavior from the trajectories of previous throws of the same plane. This approach avoids explicitly learning from the physical characteristics of the plane, which could exacerbate the sim-to-real gap. Such physical information is often difficult to obtain from real paper planes, making

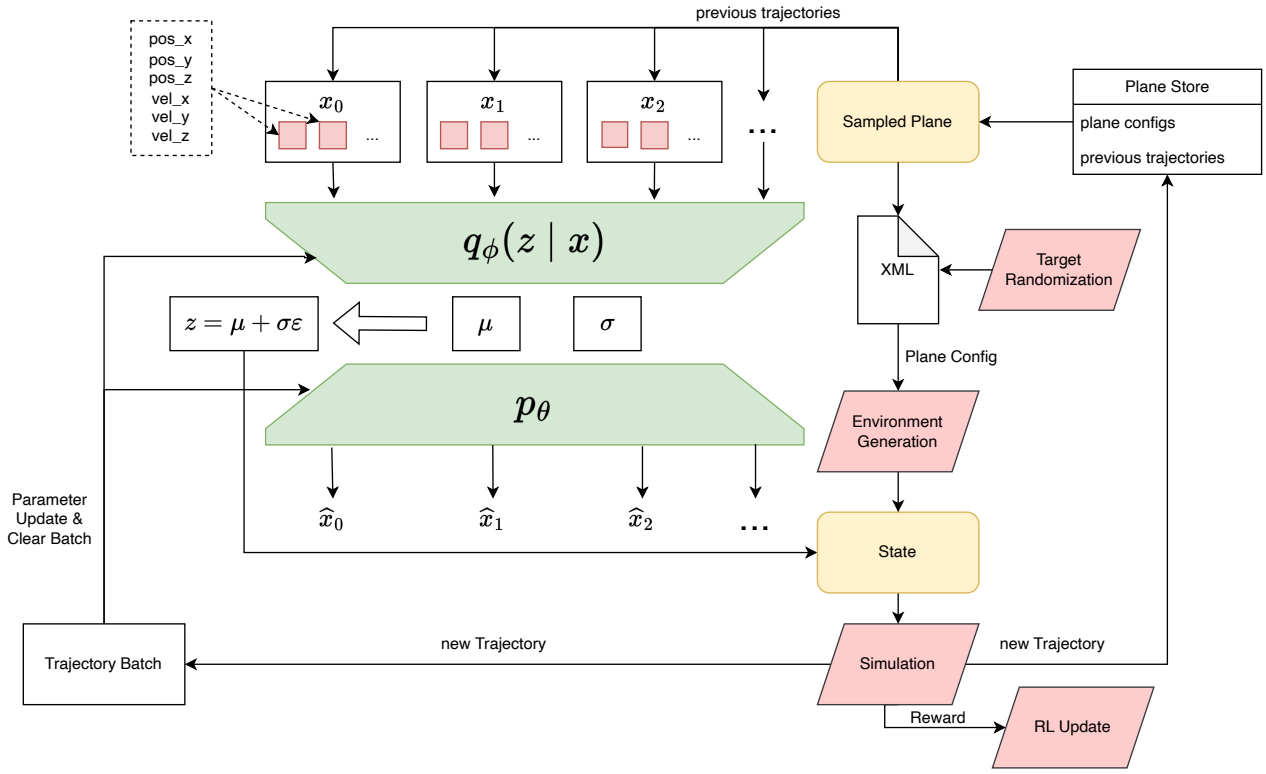

Fig. 8: Schematic representation of the paper plane throwing optimization approach. The system comprises a plane store containing diverse plane configurations and their previous throw trajectories, an environment generator, a Variational Autoencoder (VAE) for trajectory encoding, and a Reinforcement Learning (RL) agent. Each iteration involves: (1) Random plane selection from the store; (2) Environment XML generation based on plane configuration and target randomization; (3) State generation by combining environmental information with VAE-encoded previous trajectories; (4) RL agent update based on simulation reward; (5) Periodic batch updates of the VAE using collected trajectories (online training), followed by trajectory discard. The VAE is trained to reconstruct throw trajectories, while the RL agent learns to optimize throwing strategies.

our method more applicable to real-world scenarios. Also, different combinations of physics parameters can result in the same behavior. For us, only the differences in behavior are relevant. Each trajectory point consists of the plane's position $\boldsymbol{p}$, velocity $\boldsymbol{v}$, and orientation $\boldsymbol{o}$. Given that trajectories can vary in length, they are not passed directly to the model. Instead, a latent embedding of the trajectory is generated to provide a compact representation of the plane's behavior. To create latent embeddings of the trajectories, we employ a Variational Autoencoder (VAE). The VAE processes sequences of different lengths and generates a latent representation that encapsulates the plane's aerodynamic behavior. The architecture of the VAE is as follows, see figure 6 for more details:

- Encoder: The encoder uses sinusoidal positional encoding [19], followed by a linear layer and a ReLU activation function. This is followed by multi-head attention [19], another linear layer with ReLU, and finally two separate linear layers that output the mean ($\boldsymbol{\mu}$) and logarithm of the variance ($\log \boldsymbol{\sigma}^2$) of the latent distribution.
- Decoder: The decoder consists of a linear layer, sinusoidal positional encoding, multi-head attention, a LSTM layer, and a final linear layer. The input to the decoder is the latent representation $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma}^2 \cdot \epsilon$ with $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$.

We choose this architecture for our VAE model because it effectively addresses the challenges posed by variable-length trajectory data. The combination of Attention and LSTM enables the model to handle sequences of different lengths, capture long-range dependencies, and encode contextual information. The multi-head attention mechanism in the encoder allows the model to focus on relevant parts of the input trajectory, generating a rich and informative latent representation. The LSTM layer in the decoder helps maintain the temporal coherence of the generated trajectories by capturing and utilizing the relevant information from the entire sequence. Additionally, the sinusoidal positional encoding in both the encoder and decoder preserves the temporal structure and ordering of the points in the trajectory.

C. Markov Decision Process (MDP)

Our environment is modeled as a MDP $= (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$. Every state $s \in \mathcal{S}$ is represented as a tuple $(z_{\text{initial}}, z_{\text{target}}, d, \mathbf{v}, \mathbf{z}_1, \ldots, \mathbf{z}_n)$. The first components are a representation of the current setup, while the later components $(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ encode the trajectories of previous throws. z_{initial} and z_{target} are the z-coordinates of the initial and target position, d is the Euclidean distance between the plane and the target, and $\mathbf{v}$ is the normalized difference between the plane's initial position and the target's position. Specifically, $\mathbf{v}$ is calculated by taking the elementwise difference between the target position and the initial position and normalizing this difference vector so that its components sum to 1. This state representation was chosen to ensure spatial invariance, allowing the model to generalize across different throwing scenarios regardless of absolute position in 3D space. $\mathbf{z}_1, \ldots \mathbf{z}_n$ are the latent representations of the n previous throws with the same plane. If fewer than n throws have been recorded, the missing values are filled with zeros. An action $a \in \mathcal{A}$ is a 3-tuple, setting the initial orientation around the x- and z-axis and the initial speed in meters per second along this orientation. The reward $r \sim R(\cdot|s, a)$ is the negative minimal distance of the plane to the target during the whole trajectory. The specific implementation of the transition model $\mathcal{T}$ and the choice of discount factor γ depend on the training scenario, as detailed in the Training Scenarios subsection.

D. RL Algorithm

For the reinforcement learning algorithm, we utilize the Soft Actor-Critic (SAC) algorithm from the *stable_baselines3* library [20]. The agent learns to map the current state to an action that specifies the initial velocity and orientation of the paper plane.

E. Training

We use a Variational Autoencoder (VAE) with the following loss function:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}q_\phi(z|x)[\log p_\theta(x|z)] - D_{\text{KL}}(q_\phi(z|x)||p(z))$$

where $q_\phi(z|x)$ is the encoder and $p_\theta(x|z)$ is the decoder. The VAE is initially pre-trained on a set of collected trajectories from our simulated paper plane models. Subsequently, it is trained online alongside the SAC model, but with different update frequencies. The SAC model updates after every episode. In contrast, the VAE is updated every k throws (iterations). During each VAE update, we perform a training step on the trajectories collected over the past k throws. After this VAE training step, these trajectories are discarded, and new data is collected for the next update.

F. Training Scenarios

We explore two distinct training scenarios to evaluate our approach:

1) One-step setting with plane store: In this scenario, we maintain a plane store that contains the trajectories of previous throws along with their corresponding plane designs. When sampling a plane for training, we also fetch its associated trajectories. This approach ensures that we train with a full buffer of previous throws for all but the first 5 throws with each plane. The environment is modeled as a one-step setting, where each episode consists of a single throw, and the trajectories from previous throws serve as background knowledge for the agent. In this scenario, the transition model $\mathcal{T}$ is not explicitly used as each throw is treated independently. The discount factor γ also does not impact the learning in this scenario. Figure 8 shows the interactions between all components during training.

2) Multistep setting with episode-based trajectory accumulation: In this scenario, we model our environment as a multistep setting. Each episode involves throwing a plane $n + 1$ times, where n is the size of the buffer for previous trajectories. Every episode begins with an empty buffer, which is gradually filled throw by throw. In this setting, the previous trajectories are an integral part of the state, allowing the agent to learn and adapt its strategy within each episode. For this scenario, the transition model $\mathcal{T}$ resets the plane to its initial position after each throw within an episode. We set the discount factor γ to 0.99, giving higher importance to later throws which benefit from more accumulated trajectory information. The interaction between components remains consistent across scenarios, as illustrated in Figure 8 for the first training scenario. The key distinction lies in the handling of previous trajectories: rather than being stored separately in the plane store, they are incorporated directly into the state representation.

V. EXPERIMENTS

Our experiments are designed to achieve two primary goals:

1) Identify a suitable range of parameters for randomizing the paper planes.
2) Train a model on a set of planes using the parameters identified in step 1.

To establish a feasible range of paper plane designs for our study, we focus on optimizing parameters for the plane's winglet. Optimizing multiple parameters simultaneously can be challenging due to their complex interactions. By focusing on the winglet, we can more effectively manage the complexity of our experimental design while at the same time significantly influencing a plane's aerodynamic properties. The parameter selection process involves training our model on a diverse set of 130 randomly generated plane designs, with each design's agent undergoing 5000 iterations of training. This approach helps us identify a range of plane designs for which our algorithm can effectively learn to hit the target, while still allowing us to test our model's ability to generalize across different plane configurations within this established range. Next, we use planes generated from the identified parameter range to collect a set of trajectories, which are then used to pre-train our VAE. We set the dimension for

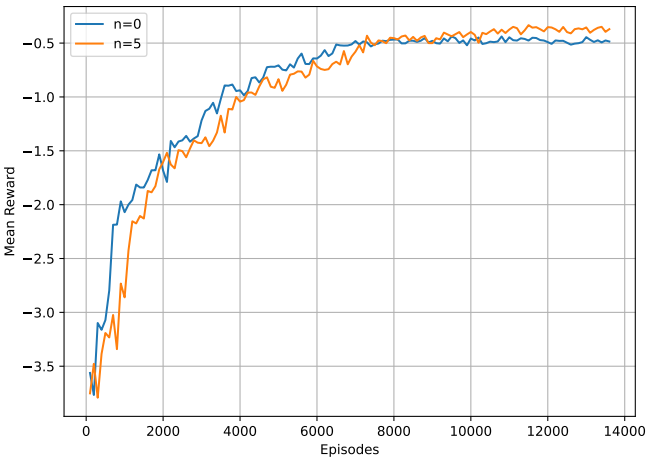

Fig. 9: Reward progression during training. Each point represents the mean reward over 100 episodes. The orange line indicates the reward for an agent utilizing up to five trajectories from previous throws, while the blue line shows the reward for an agent relying solely on current environmental information, without considering past throws.

the hidden linear layers to 16 and the dimension of the latent space to 8. As discussed in the methods chapter, each point in a trajectory consists of a 10-dimensional vector, so our input dimension is 10. We train the VAE for 15 epochs using the Adam [21] optimizer with a learning rate of 10^{-3}. The pre-trained VAE serves as the initialization for all subsequent experiments. Finally, we train the SAC model, varying the number n of previous throws considered. The SAC policy is initialized randomly, while the VAE is initialized using the pre-trained VAE. Both models are trained simultaneously, as described in the methods section.

A. Implementation of Training Scenarios

As described in the Methods section, we implement two distinct training scenarios: a one-step setting with a plane store and a multistep setting with episode-based trajectory accumulation. Here, we detail the specific parameters and implementation choices for each scenario.

1) One-step Setting Implementation: For the one-step setting we use a plane store containing 10 different plane designs. We compare a setting where each plane was associated with 5 previous throws to a setting where the agent has no knowledge about prior throws.

2) Multistep Setting Implementation: For the multistep setting we also use a set of 10 different plane designs. Each episode consists of $n+1$ throws, where $n = 5$. We train the agent for 15000 episodes. The buffer of previous trajectories starts empty at the beginning of each episode and is filled progressively.

VI. RESULTS

We present our findings from the two training scenarios: (1) a one-step setting with a plane store, and (2) a multistep setting with episode-based trajectory accumulation.

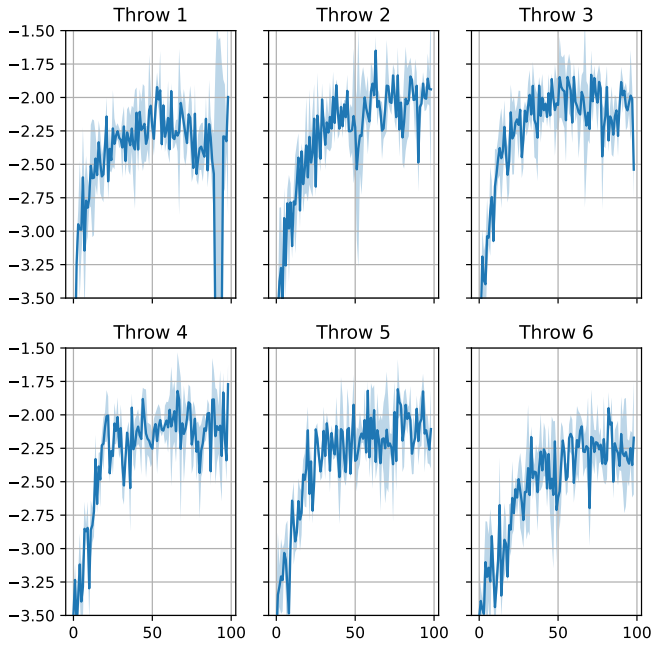

Fig. 10: Evolution of rewards across multiple throws within a single episode. Each subplot represents a sequential throw, with the x-axis showing training steps and the y-axis indicating the reward. Data points are aggregated over multiple iterations to improve readability. The darker line represents the mean reward, while the shaded area depicts the standard deviation. Throw 1 begins without prior information, while each subsequent throw incorporates data from all preceding throws in the episode.

A. Scenario 1: One-step Setting with Plane Store

We conduct a comparative analysis of two distinct models: one that exclusively considers the environmental configuration, and another that incorporates both environmental factors as well as the trajectories of up to five previous throws. As illustrated in Figure 9, the model focusing solely on environmental information initially demonstrates superior performance. However, as the simulation progresses, the model that utilizes individual plane behavior data achieves better performance. The benefits of incorporating information from previous throws become even more evident when evaluating the models' performance on a set of 20 previously unseen paper planes. In these novel scenarios, the model that utilizes prior throw information consistently outperforms the model that relies exclusively on environmental configuration. Table I presents a comprehensive comparison of rewards across different models when tested on these unseen planes.

B. Scenario 2: Multi-step Setting with Episode-based Trajectory Accumulation

Figure 10 illustrates the evolution of agent performance across multiple throws within a single episode, providing insights into how the accumulation of trajectory information impacts the agent's ability to optimize its throwing

Number of previous trajectories	Reward
0	-0.4913 ± 0.2461
5	-0.3676 ± 0.1827

TABLE I: Performance comparison between two models for predicting paper plane trajectories. The first model uses no reference trajectories, while the second model incorporates information from 5 previous throws of the same plane. Each model is evaluated on a test set of 20 unseen planes, and the average reward across these planes is reported.

Throw number	Reward
1	-2.1175 ± 1.2290
2	-1.9544 ± 0.8115
3	-2.0931 ± 0.8646
4	-2.1432 ± 0.5443
5	-2.1525 ± 0.9796
6	-2.3030 ± 1.0560

TABLE II: Mean rewards and standard deviations for each throw across 300 episodes with 20 unseen plane designs.

strategy. The performance of the first throw, which occurs without any prior trajectory information, initially shows an upward trend. However, we observe a subsequent decline in performance as training progresses. In contrast to the first throw, the second throw, which incorporates information from the trajectory of the first throw, demonstrates consistent improvement throughout the training process. Interestingly, the performance of throws that utilize information from multiple previous trajectories (throws 3-6) exhibits oscillatory behavior. Moreover, we observe that as the number of previous trajectories used increases, there is a trend towards decreased performance. Table II presents the mean rewards and standard deviations for each throw across 300 episodes, using 20 unseen paper plane designs. The data is aggregated from three different models trained with the same parameters but different random seeds. These results corroborate our earlier findings from figure 10.

VII. DISCUSSION

Our study revealed distinct patterns in agent performance across two training scenarios, each offering insights into the role of historical trajectory information in optimizing paper plane throws. In the first training scenario, we observe an initial advantage for the agent not utilizing previous throw trajectories. This early success can be attributed to the effectiveness of a consistent throwing strategy in the initial stages, which facilitates rapid progress. However, after approximately 8000 iterations, the model leveraging past throws as reference demonstrates superior performance. This improvement likely stems from its ability to utilize information about each paper plane's specific flight characteristics, enabling more precise and tailored throws.

In the second training scenario, during training, we observe an initial improvement in the first throw's performance, followed by a decline. This pattern suggests an initial learning phase where the agent develops a baseline strategy based on environmental conditions alone. However, the subsequent drop in performance might indicate overfitting to certain plane designs or environmental conditions, potentially at the expense of generalization. The high variability observed in the first throw with unseen planes further supports this interpretation, highlighting the challenge of generalizing to new designs without prior trajectory information. The consistent improvement in the second throw's performance, both during training and with unseen planes, underscores the value of even a single previous trajectory in enhancing the agent's decision-making. This suggests that the agent can effectively leverage this limited historical data to refine its throwing strategy and achieve better outcomes. While information from a single previous throw proves beneficial, our results show a decline in performance for later throws. This unexpected trend could be attributed to several factors: (1) Insufficient capacity or complexity to effectively process multiple trajectories, (2) current representations of trajectories are not covering the information good enough, (3) overspecialization to specific throw sequences. To disentangle these factors and gain a deeper understanding of which aspects of the trajectories are most informative, ablation studies would be valuable in future research. Qualitative observations of throw recordings suggest a potential change in the agent's strategy: the first throw often exhibited high velocity aimed directly in the direction of the target, while subsequent throws, including the second, appeared to use lower velocities. This change in approach coincides with the availability of previous trajectory information. However, it's important to note that this observation is based on qualitative review and not quantitative analysis.

VIII. FUTURE WORK

The findings of this work highlight a crucial area for immediate investigation: understanding why information from the first throw improves the second throw's performance, while additional trajectory information appears to be less beneficial or even detrimental. This phenomenon warrants a detailed analysis as it could provide valuable insights into the nature of the information being captured and how it's being utilized by the current model. Ablation studies should be performed to isolate the impact of different components of the trajectory information. This could help identify which aspects of the previous throws are most informative for subsequent throws, and why this information might become less useful over time. Only after gaining a deeper understanding of these dynamics should scaling up the model size be considered, particularly the VAE component. If the analysis suggests that the current model is indeed capacity-limited, increasing its size could potentially address the observed performance decline when processing multiple trajectories. An important direction for future work is increasing the complexity of the paper plane model. While the current work focuses on optimizing parameters for the plane's winglet to manage complexity, future iterations should explore randomizing a broader range of parameters.

IX. CONCLUSIONS

Our study demonstrates the effectiveness of combining reinforcement learning with trajectory encoding for optimizing paper plane throwing strategies. The proposed method, integrating a VAE for encoding throw trajectories with an SAC algorithm for action selection, shows improvements over approaches relying solely on environmental information. By incorporating data from previous throws, our model successfully adapts to the unique flight characteristics of individual paper planes, resulting in more accurate throws, especially for unseen designs. However, we observed diminishing returns when using multiple previous trajectories, highlighting areas for future research. These findings not only advance paper plane throwing optimization but also offer insights for broader applications in robotics and dynamic object manipulation tasks.

REFERENCES

[1] Ruoshi Liu et al. "PaperBot: Learning to Design Real-World Tools Using Paper". In: *arXiv preprint arXiv:2403.09566* (2024).

[2] 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*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[3] Diederik P Kingma. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[4] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[5] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[6] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017).

[7] Mohamed El-Kaddoury, Abdelhak Mahmoudi, and Mohammed Majid Himmi. "Deep Generative Models for Image Generation: A Practical Comparison Between Variational Autoencoders and Generative Adversarial Networks". In: *Mobile, Secure, and Programmable Networking*. Cham: Springer International Publishing, 2019, pp. 1–8. ISBN: 978-3-030-22885-9.

[8] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. "A hybrid convolutional variational autoencoder for text generation". In: *arXiv preprint arXiv:1702.02390* (2017).

[9] Xiaoan Yan, Daoming She, and Yadong Xu. "Deep order-wavelet convolutional variational autoencoder for fault identification of rolling bearing under fluctuating speed conditions". In: *Expert Systems with Applications* 216 (2023), p. 119479.

[10] Vaddadi Sai Rahul and Debajyoti Chakraborty. "Exploring reinforcement learning techniques for discrete and continuous control tasks in the MuJoCo environment". In: *arXiv preprint arXiv:2307.11166* (2023).

[11] Tom Erez, Yuval Tassa, and Emanuel Todorov. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 4397–4404.

[12] Reda Bahi Slaoui et al. "Robust domain randomization for reinforcement learning". In: (2019).

[13] Andy Zeng et al. "TossingBot: Learning to Throw Arbitrary Objects With Residual Physics". In: *IEEE Transactions on Robotics* 36.4 (2020), pp. 1307–1319. DOI: 10.1109/TRO.2020.2988642.

[14] Chihiro Obayashi, Tomoya Tamei, and Tomohiro Shibata. "Assist-as-needed robotic trainer based on reinforcement learning and its application to dart-throwing". In: *Neural Networks* 53 (2014), pp. 52–60.

[15] Yeong-Gyun Kang and Cheol-Soo Lee. "Deep Reinforcement Learning of Ball Throwing Robot's Policy Prediction". In: *The Journal of Korea Robotics Society* 15.4 (2020), pp. 398–403.

[16] Vassilios Tsounis et al. "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3699–3706. DOI: 10.1109/LRA.2020.2979660.

[17] Joonho Lee et al. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47 (2020), eabc5986.

[18] Siddhant Gangapurwala et al. "Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 5973–5979.

[19] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[20] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[21] P Kingma Diederik. "Adam: A method for stochastic optimization". In: *(No Title)* (2014).