# SawyerRoll: A Mechanical and Learning System for Robotic Ball Rolling

Rafael Torres, Cougar Seale, Anikait Gupta

***Abstract*—Rolling behavior in robot manipulation sits between pushing and throwing, requiring continuous-contact control like pushing, while maintaining directionality and release sensitivity similar to throwing. However, accurate robotic rolling remains underexplored in comparison to common manipulation tasks such as pick-and-place or grasping. In this project, we design and propose SawyerRoll, a novel system for enabling the Sawyer robot to roll a ball to a target using a custom-made mechanical tool for an end-effector, a physics-validated simulation environment, and three distinct training imitation-learning policies. Using 25 successful human teleoperated demonstrations, we evaluate two behavioral cloning models, as well as one Diffusion model. Through 30 episodes, we determined the performance. Our collected data indicated that models that have the higher amount of state information has the highest accuracy. We also demonstrated that adding a custom-made end-effector allows for a larger rolling reach, which proves that certain mechanical tools can elongate a robot's reach. Our limitation and conclusion outline key insights toward building a repeatable rolling benchmark for Robosuite.**

## I. INTRODUCTION

Robotic manipulation research has focused heavily on prehensile tasks such as grasping or placing, and on dynamic tasks such as throwing. Rolling, in contrast, has received limited study despite being relevant for tasks requiring gentle, directional, or continuous-contact object motion. Rolling involves nonlinear frictional effects, sensitivity to release angle, and complex contact dynamics, making it difficult for standard grippers or pre-trained policies.

We focus on two main difficulties that are crucial when making a system that can roll a ball. Firstly, uncomfortable and unfit end-effectors that control the movement of the ball. Different robot arms have different default end-effector, which makes creating a standardized benchmark for rolling complicated, as robot arms wouldn't react the same way. Therefore, we propose the idea of using a custom-made end effector that would be used instead of each robot arm's end-effector. This could allow different robot arms to use the same benchmark without needing to have the same default end-effector, as they would use the custom-made one. This enables the creation of the standardized benchmark for rolling a ball to a predetermined area.

Secondly, nonlinear forces created by friction and other contact forces when a ball rolls in a surface. Tasks like throwing, lifting, pick-and-place do not have to consider the physics of their surroundings, the only force acting on the object is air resistance, as the object flies through the air. In the case of rolling, we interact with many more forces which are capable of having great effects on the ball trajectory. Friction from the table as the ball rolls, even friction from the end-effector are types of forces which, if not controlled properly, will result in significant changes to the end position of the ball, affecting the overall task performance.

In this project, we propose SawyerRoll, a replication-ready system that allows a Sawyer robot arm to roll a ball to a predetermined area. Currently, SawyeraRoll performs in simulations within the Robosuite simulation framework built over the MuJoCo physic engine. SawyerRoll's goal is to construct a physics-accurate simulation where human demonstrations through teleoperations can be collected, as well as provide the ability to train different policies that support different performance metrics. In all, this would allow for evaluation of the performance. Most importantly, SawyerRoll proposes the idea of not utilizing distinct grippers, but one common custom end-effector that would facilitate the task of rolling a ball.

Through 25 demonstrations and 3 distinct training model, 2 Behavioral Cloning each with different amount of state information and 1 Diffusion with the maximum amount of state information, SawyerRoll will evaluate the performance and accuracy of the task.

We found that after running 30 episodes, BC - Rich (which had joint, ball and end-effector state information) performed significantly better than BC - Min (Which had ball and end-effector state information) in the whole duration of the episode, and in the actual tasking of making the ball land within the predetermined area. The evaluation discusses the meaning of low loss functions, total reward, and final reward, concluding that with more state information, the more accurate your system will be. Finally, we will discuss the limitations brought by the system, as well as proposed ideas for future improvement of SawyerRoll.

## II. BACKGROUND

Our study build upon existing research in robotic manipulation, object handling and physics-environments.

Below, we summarize key contributions that inform our methodology.

**ActivePusher: Active Learning and Planning with Residual Physics for Nonprehensile Manipulation**: ActivePusher [1] ActivePusher is a framework that combines residual physics, active learning, and uncertainty-aware modeling. This system uses these methods to examine non-prehensile manipulation, such as pushing or rolling objects. Their uncertainty model allows them to target their learning through confidence score, targeting those predictions where the score was lower, improving in the areas needing improvement. This also allows the robot to carry out the specific task denoted as not too "risky" by their score system. This is really helpful for small data scenarios. This framework is effective for our policy because we also share the context of a small data scenario, and it performs non-prehensile tasks, like we want to carry out.

**Throwing Objects into A Moving Basket While Avoiding Obstacles**[2]: This policy proposes a planning-and-control pipeline for throwing objects into a moving target while simultaneously avoiding obstacles. The paper focuses on trajectory generation combining a model-based throw and collision avoidance and demonstrates good performance in experiments, showing target tracking and obstacle-aware throws. The contribution is the integration of dynamic target prediction with obstacle avoidance in the throwing task. TossingBot tries to fix the issue of targets being out of reach for robot arms, which is why this policy is helpful. SawyerRoll also want to tackle reach by customizing a gripper which allows for the ball to be rolled a larger distance.

**Data efficient Robotic Object Throwing with Model-Based Reinforcement Learning**: Monte Carlo - Probabilistic Inference for Learning Object-Throwing (MC-PILOT) [3] Used data to simulate the throwing system and significantly reduce the time spent for training. They also used the data collected to create stochastic models that account for object dynamics. Using a model that accounts for the system, better release error distributions were also achieved. By implementing this approach, the new policy attained better accuracy and also required fewer trials to be conducted. This policy is helpful for us because it helps tackle are the largest problem, doing the most with the little amount of data that we have.

## III. METHOD

SawyerRoll integrates 2 subsystems:

### A. *Mechanical rolling tool*

For this project, we introduced one novel mechanical feature to assist our task of rolling a ball:

**3D Modelled End-Effector.** The Sawyer robot presents many physical and mechanical limitations which could hinder its ability to make a ball roll to a predetermined area. Being able to alter the speed and direction is something crucial for our project. Moreover, we predicted that using the Sawyer's gripper, a two-prong design, would add unnecessary complexity to our project. Issues such as needing to control these prongs mid-action to release the ball, and measuring the pressure of the prongs would require additional end-effector monitoring, diverting of from the focus of our study. A simplified end-effector would eliminate these issues, enabling a standardized rolling benchmark across other robotic arms.

For the end-effector design, we wanted a cup-like shape that would hold the ball through the duration of the task, providing stability. We thought of a spoon carrying an egg, as a visual representation of what we wanted to achieve, easy to carry around, and you can rotate the spoon on its own axis, allowing the egg to roll.

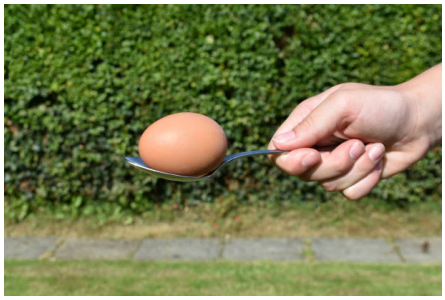

Fig. 1: Image of a spoon carrying an egg. Visual representation of our intended end-effector's functionality.

However, there are issues with this design. The ball can easily roll off when the spoon has a high speed, or a sudden change in direction, which as mentioned previously, are crucial parameters for SawyerRoll. To avoid this issue, we needed to increase the curvature of the cup in our design. We chose 'Onshape' [4] to carry out our designing, due to previous experience with the program.

Fig. 2: Image of end-effector design.

After a few design rounds, we chose Fig 2. The curvature of the cup was made considering the radius of the ball used for our rolling task. The ball sat perfectly inside the end-effector, allowing stability and free movement of the end-effector without rolling the ball off. Moreover, when needing to roll the ball, we could easily rotate the end-effector, enabling the task to carry out efficiently.

### *B. Robosuite simulation environment*

Given our project objectives, we chose Robosuite as the platform where we would build our environment, it was the optimal choice for a few reasons.

**MuJoCo Physics Engine.** Due to the nature of our experiment, when searching for a simulation framework software, we were looking for one which took into account the physics of a rolling ball. We looked at Robosuite [5] which is a simulation framework built on MuJoCo (Multi-Joint dynamics with Contact) [6], an open-source physics engine. MuJoCo has very accurate contact dynamics, meaning that it is more than suitable for tasks that have distinct physical parameters, such as friction. This is something crucial for our experiment, as our task is a ball rolling to a predetermined area. This physics engine enables the ability to change and vary these parameters, allowing us the freedom to change and test different settings, as well as better approximating the physical behavior of the ball to a real-world ball.

**Creation and Customization of Environments.** Robosuite allows for the creation and customization of frameworks, meaning that we would have the ability to customize our task and environment in necessary ways to allow us to reach the desired simulation setup. This also fits our goal of constructing a standardized benchmark for distinct environments. Due to its built-in models and tasks, we knew we could construct an arena with a Sawyer arm, ball, and a table, all crucial for our setup.

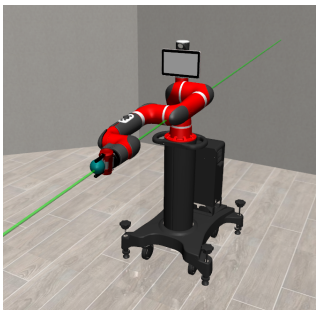

Fig. 3: Image of first environment created in Robosuite.

Fig 3 shows our first attempt at creating our environment. Using Robosuite's built-in robot models, we placed a Sawyer arm in an arena with a ball in-between its prongs.

After understanding MuJoCo better, we created an environment consisting of the Sawyer arm, a ball, and a table.

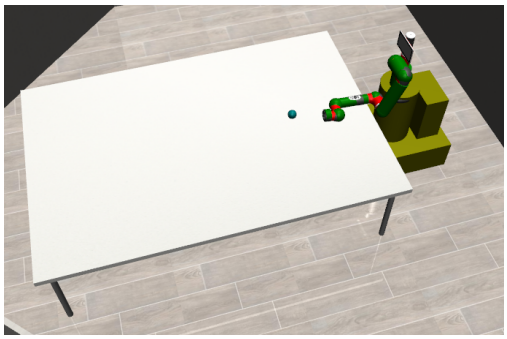

Fig. 4: Image of second environment created in Robosuite.

**XML Modification.** Robosuite enables the modifying of the XML file constructing the environment, as well as the robot's own XML file. This would allow us to implement our own end-effector design. Moreover, as seen in Fig 4, the Sawyer arm has no gripper, as this was our first attempt at modifying the XML of the Sawyer arm, and it worked successfully.

When implementing the end-effector from Fig 2 into our environment through an XML file, we ran into issues, our MuJoCo environment would not recognize our concave cup due to its convex-only collision mesh limitation. This meant we had to break down our design into multiple convex shapes. Through an Onshape functionality, we managed to decompose our end-effector into 16 convex parts.

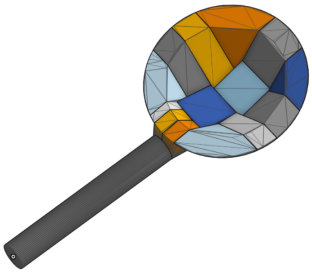

Fig. 5: Image of end-effector broken-down into 16 convex shapes.

Fig 5 is the final design that we implemented into our environment.

After adding the end-effector, we constructed an area within the table, to visualize the goal of the task, to roll the ball to this area.

**Teleoperated Human Demonstrations.** Robosuite also provide built-in scripts for data collection and data gathering of human demonstrations. We could use the

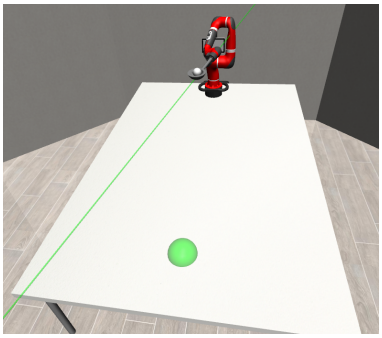

Fig. 6: Image of SawyerRoll simulation environment.

keyboard to carry out demonstrations and use the data as input when training our models. This is crucial for our objective of using imitation learning to train our models.

**Robomimic Integration.** To carry out the training of our models, we wanted to use Robomimic, a framework for robot learning from demonstration, [7] which fits our project goal really well. Robomimic has built-in functions to transfer data collected from Robosuite and use it as a dataset to train. Moreover, Robomimic has written scripts for training models, making the training section of the experiment much simpler.

## IV. EXPERIMENTS

To test our task and environment, we diced to carry out imitation learning through data collection. Using the teleoperation script provided by Robosuite, we managed to connect the keyboard to the environment, allowing us to control the Sawyer arm how we see fit, and therefore providing successful trials. For simplicity, we decided that when carrying out human demonstrations, the ball would be manually pre-placed in the end-effector. This would allow our system to only focus on the core task, rolling the ball. For our human demonstrations, we set variables to ensure an accurate data collection. For the environment, we set the following variables:

- Table height (0.75 m)
- Ball radius (0.035 m)
- Ball friction (3000)
- Ball density (4)

For the table height, we had seen a table next to a Sawyer arm and determined that table suitable for a real-life trial, so we copied the height of the table. For the radius, we used the radius of an average tennis ball. The friction and density were harder to get. When setting up the environment initially, we saw that every demonstration was a failure, it would always roll off. We had to play around with the number until we had values that would allow the ball to stay within the target for 5 seconds, and those are the values we ended up selecting.

For environment horizon, we set on 1300. After many trials, we found that 1300 allowed the demonstration enough time to reach a successful state, but also to determine whether a trial had not been successful. We also determined a check success function, which would terminate the demonstration if achieved. Due to MuJoCo not being a perfect physics environment, there were some issues when performing the demonstration. As rolling is one of the most complicated tasks you could perform in MuJoCo, as it is the one that has most contact with other objects, the behavior of the ball was never perfect. We attempted to perform a demonstration where the ball would lay still inside the area, but it never happened. We tried changing the physical parameters of the ball and arena, but it still never worked. To counter this inaccuracy, we decided that it was best to consider a successful trial when the ball would remain inside the area for at least 5 seconds. This would allow evaluating our system for successful trials while taking into account that MuJoCo is not perfect. The demonstration were coded so that whenever the state of the ball was considered successful, it would terminate the trial. This was our check success function:check success = ball inside area for at least 5 seconds.

A reward function was also constructed for our environment, which took into account various aspects of the demonstration:

$$\text{r} = \begin{cases} -1 & \|e - t\| < 0.3 \\ \frac{1}{1+d} - \frac{v}{10}\mathbb{I}(d < r_a) & \text{else} \end{cases}$$

Where d = distance from ball to target, v = sphere velocity magnitude, e = end-effector position, t = target area, and r_a is the acceptable radius of the area. In case where distance is inside the acceptable area then d smaller than r_a = 1, if not it equals 0.

Fig 7 is a screenshot of a successful demonstration, which has been taken during a demonstration. As we can see, in this case, the ball was inside the area for 5 seconds (we took a screenshot when it was exactly inside).

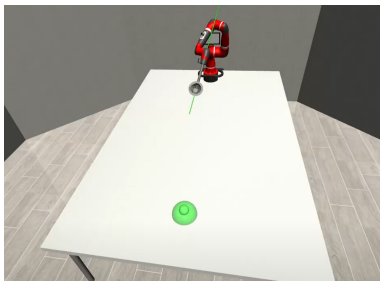

Fig. 7: Image of successful demonstration.

On the other hand, Fig 8 is a screenshot of an unsuccessful demonstration, which has been taken during a demonstration. As we can see, in this case the ball

did not last for 5 seconds inside the area, and thus it continued to roll, and rolled out of the table.

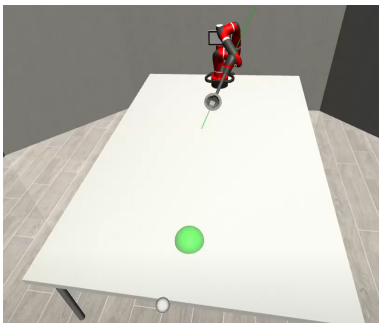

Fig. 8: Image of unsuccessful demonstration.

For our data collection, we decided to collect 25 similar successful demonstrations due to the policies we are going to use in the training section of the experiment. Using Robosuite's built-in gather function, we placed our 25 demonstrations within an HDF5 file, which is the format that Robomimic uses for training data. This HDF5 file contained the following for each demonstration:

- Joint positions, velocities, and angles
- End-effector position and orientation
- Ball position
- Timestamps
- Input actions
- Reward

### A. *Policy learning*

We selected three distinct policies to train our model, which would give us distinct insight on the trained model's behavior. These three policies also test our reason for carrying out 25 demos. Some of the policies, on average, require more demos to start outputting successful roll-outs, when others might not need as much. The amount of demos we have time to perform is also a limitation in our experiment. Due to our low amount of time to carry out the training of our model, we had to prioritize results over more demos.

*a) Behavioral Cloning - Minimal vs Rich*

Behavioral cloning works as the agent tries to replicate the actions you provide it with. In this case, it will try to replicate the demonstration of rolling the ball. We wanted to use BC due to its simple use with similar demos. BC works best when given multiple demos, as it records the states in each of them and uses the information to calculate its next action. If all demos are similar, then it will take less amount of demos until the model trains to perform successfully. BC is simple and fast to train, many times requiring less data to perform to the same level as other policies.

However, BC has negative aspects. The compounding error is a significant aspect to take into account. This term suggests the cases where the model commits an error, and then all actions afterward have different outputs, which are also erroneous, and in towards the end of the roll-out, the error snowballs and it becomes large. This means that if within a roll-out something occurs not according to plan, the BC model will not know how to react to this sudden change, as it only know what we did in every instance.

To see the effect of different objects and states in BC, we split BC into two different models. In the case of Behavioral Cloning - Minimal (BC-Min), we provided the model with the following data from the HDF5 file:

- End-effector position and orientation
- Ball position

This suggests that the model could only 'copy' these aspects of the demonstrations provided.

BC-Rich gives in data about the joint states, the position, velocity, and angles, which suggests that it will have more knowledge on how to move the arm as a whole, rather than just moving the end effector over to a position. We predict that this extra info on the joint states, will be reason for better results, as it provides more stability throughout the roll-out of the task. To be specific, the MuJoCo environment takes into account physics, which means that the arm will try to copy the action, but if there is a significant physical change then the BC model will act upon that, which will not give the same result. For example, in BC-Min, if the arm is copying the end-effector pose, but performs an action which causes gravity to pull the arm differently than expect, the end-effector will change its trajectory, and compounding error will begin. Therefore, having exact knowledge of how the arm moves, in case of BC-Rich, then it will not encounter this error as much.

Something important to note, is that in both cases, BC does not take reward functions, as they react solely based on copying the states the model is given.

For Behavioral Cloning - Rich (BC-Rich), we provided the model with the following data from the HDF5 file:

- Joint positions, velocities, and angles
- End-effector position and orientation
- Ball position

*b) Diffusion*

The other policy being tested is diffusion. Diffusion promises a strong multimodality, meaning that it will be better at understanding that there are more than one correct way to reach the success point. Moreover, it is able to decide which of these options is the best and more coherent to chose out of.

However, diffusion is known to be slower and more needy in training, meaning that it needs more objects

and data to be able to perform accurately. Moreover, it was hard for us to collect diffusion data due to the amount of space training this model takes. Diffusion is a heavy (memory wise) model to run, and in our case make the process of testing and training much slower. For example, for all policies we planned 100 epochs, however, the computer quit the training at epoch 50 due to limited space. This is definitely something to take into account when analyzing the result of the diffusion model, as it may need more epochs to begin performing correctly.

In the case of the diffusion policy, we provided the model with the following data from the HDF5 file:

- Joint positions, velocities, and angles
- End-effector position and orientation
- Ball position
- Timestamps
- Input actions
- Reward

## V. RESULTS

We evaluated each policy over 30 episodes using 3 metrics:

- Total reward
- Final reward
- Loss vs Epoch

Total reward refers to the cumulative reward received in the whole episode (roll-out), the sum of total rewards over the episode. Higher total reward suggests the mode is performing well in the whole episode, meaning it is accurately following the states and data it is given. Final reward is purely based on achieving success or not. The model will perform, and we analyze whether it did a successful episode, judging by the position of the ball at the end of the episode. Higher final reward suggests the model is performing successfully, and is managing to perform the task we have created. Loss vs Epoch is a metric to determine how well the model is learning. Each epoch we calculate the loss of the model, and by the last epoch we can create a graph to plot this change. Lower loss suggests that the model is learning correctly. For BC, a lower loss suggests the model is improving in its ability to copy the demonstration provided.

### A. BC-Min Performance

Total reward:

- Mean: 103.6
- Standard deviation: 6.404
- Min: 92.633 (episode 2)
- Max: 128.087 (episode 8)

Final reward:

- Mean: 0.234
- Standard deviation: 0.022
- Min: 0.190 (episode 2)
- Max: 0.280 (episode 20)

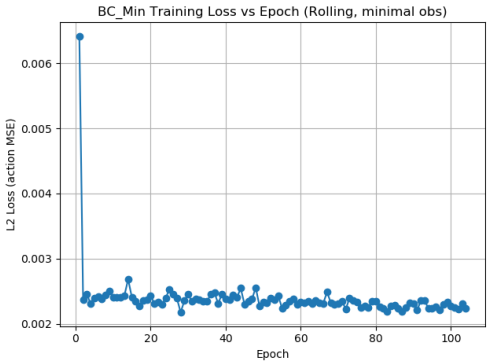


Fig. 9: Loss vs Epoch graph (BC - Min).

### B. BC-Rich Performance

Total reward:

- Mean: 133.508
- Standard deviation: 4.580
- Min: 131.720 (episode 28)
- Max: 157.979 (episode 29)

Final reward:

- Mean: 0.342
- Standard deviation: 0.027
- Min: 0.329 (episode 28)
- Max: 0.484 (episode 29)

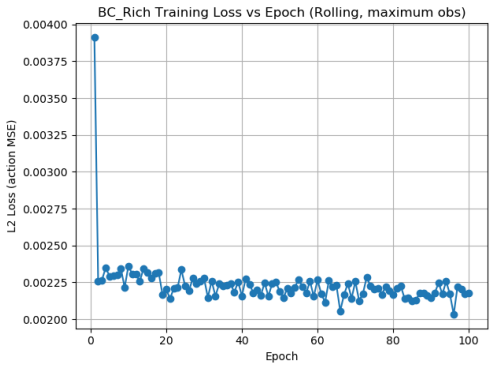


Fig. 10: Loss vs Epoch graph (BC - Rich).

### C. Diffusion Policy Performance

## VI. CONCLUSION

**Behavioral Cloning**. When analyzing the graphs, we can easily determine that the loss in both policies decreased to near 0 by the first few training epochs, it converges, they plateau rapidly and then have visible noise throughout the rest of epochs. This demonstrates that overall, both BC models are learning quickly how to

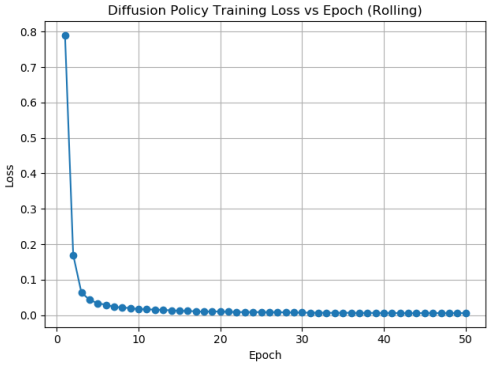


Fig. 11: Loss vs Epoch graph (Diffusion).

copy the demonstrations provided, considering the data each model is given. We also see that what the model learns each epoch gets lower, perhaps 100 epochs is not as necessary.

However, we see that the loss of the BC - Rich model is lower than the BC - Min loss. This tells us that the Rich model is more accurate when predicting the action performed by the demonstration, it's better at copying.

Through the graphs, we can determine that BC - Rich does a better job at making predictions, which is most likely due to the fact that we provide it with more data, so it can infer the action wanting to be carried out better. In the case of BC - Min, the loss suggests that the model won't be able to perfectly infer the demonstration's action, due to its limited amount of data.

This is conclusion is also proven correct through the quantitative results determined. Compared to the loss graphs, the performance metrics show a significant difference between both BC models. For total reward, BC - Rich performed almost 29% better than BC - Min, which is 5 times the standard deviation of BC - Min, suggesting that there is a clear difference caused by lack of data. The same happens for final reward, as BC - Rich's performance is almost 50% better than BC - Min. Therefore, the extra dimensions added to the BC - Rich model are indeed the reason of a more accurate model, which gives the ball more stability and a better episode performance.

When analyzing the metrics on their own, we can determine some insights. Firstly, there was no episode that was a success for either BC model. Although we gave both models 25 correct human demonstrations, we didn't see a reward of 1. This can be due to a few reasons. Firstly, compounding error. As mentioned previously, this suggests many small errors send the trajectory of track and return a final reward below one. These errors accumulate and in the end the model won't be performing accurately. Secondly, small variation of demonstrations. Because we are training the model to predict actions depending on what state it is, if it goes off track it won't be able to correct itself, as we have given it 25 similar demonstrations, it doesn't know how to do anything else. This means the model won't react accurately in different situations. Finally, a high total reward does not prove a high final reward. The BC - Rich model is having accurate total roll-outs, but not a successful trial, which can be proof of compounding error taking place, as these small changes wouldn't affect the total reward as much, but would heavily hinder the final reward.

**Diffusion.** The diffusion loss graph also shows loss decreasing after the first few epochs and plateauing near 0. However, the magnitude of this decrease is much greater for this model. Before the first 5 epochs, the model decrease from 0.8 to around 0.16 loss, and we also see how the loss has almost no noise throughout the epochs. Both these aspects suggest that the model learns the task rapidly, it becomes accurate at predicting the actions needed. Because we don't have the metrics of performance of the diffusion model, we can't determine whether the low loss causes a high performance, as it wasn't the case for the BC models.

An issue with diffusion policies is that if you have a small dataset, then the model con overfit and almost memorize your data, so it may be positive performance, but in reality when tested for somewhat distinct tasks then it wouldn't be as accurate.

## VII. LIMITATIONS & FUTURE WORK

Due to the context in which our project was made, we have identified several limitations throughout the project.

### *A. Demonstrations.*

Perhaps our biggest limitation in our project was the small set of demonstrations. This small set of all similar and correct demonstrations are useful enough to give an initial understanding of the problem and behaviors of the models we trained. However, if we wanted to further our experiment, we would need a larger dataset with more demonstrations, as well as a different dataset with demonstrations that are not all similar but still reach the success we are looking for. These would allow our models to be truly tested. Moreover, perhaps the reason why our loss scores are so low is because the models are overfitting to the demonstrations, which is another reason why having more variable data could be a stronger critic of the model we trained.

For a future step, we would focus on getting 2 datasets. The first one would include 250 similar successful demonstrations, while the second one would include 250 different successful demonstration. Comparing them would demonstrate, prove whether understanding what to do in different positions would help our models perform with higher accuracy.

### B. Time to finish script.

Another big limitation was the actual scripts. We found many issues throughout the project when interacting with Robosuite and Robomimic. Many of the issues were challenges that took weeks to solve and overcome. We ideally wanted to train an offline Reinforcement Learning model, but we had issues in our data collection wrapper, it had issues registering the next observation properly, which is something needed for offline RL models. Moreover, we wanted to access the performance metrics of the diffusion policy, but we had issues with the script that trained the diffusion policy and called for the metrics. These would have given us further insight on the models we tested.

For future steps, we would work on a few things in our scripts. Firstly, we would work on collecting the performance metrics of the diffusion policy, allowing us to compare it quantitatively to the BC models. Secondly, making necessary changes in data collection script to allow for an offline RL model to be trained. Finally, improve the physics of our environment. We could add an uncertainty-aware planning [1] as well as a residual physics model to increase accuracy of our trained models, and further consolidate SawyerRoll as a benchmark for rolling object to predetermined areas.

## References


[1] Zhuoyun Zhong, Seyedali Golestaneh, and Constantinos Chamzas. Activepusher: Active learning and planning with residual physics for nonprehensile manipulation, 2025. URL https://arxiv.org/abs/2506.04646.

[2] Hamidreza Kasaei and Mohammadreza Kasaei. Throwing objects into a moving basket while avoiding obstacles, 2022. URL https://arxiv.org/abs/2210.00609.

[3] Niccolò Turcato, Giulio Giacomuzzo, Matteo Terreran, Davide Allegro, Ruggero Carli, and Alberto Dalla Libera. Data efficient robotic object throwing with model-based reinforcement learning, 2025. URL https://arxiv.org/abs/2502.05595.

[4] Onshape, Inc. Onshape: Cloud-native cad and product development platform. https://www.onshape.com, 2024. Accessed 2025-03.

[5] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. robosuite: A modular simulation framework and benchmark for robot learning, 2020. URL https://arxiv.org/abs/2009.12293.

[6] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5032, 2012. doi: 10.1109/IROS.2012.6386109.

[7] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Robomimic: A framework for robot learning from demonstration, 2021. URL https://arxiv.org/abs/2108.03298.