# Generalizable Dynamic Handover

**Xuanyi Xie**
IIIS, Tsinghua University
xie-xy23@mails.tsinghua.edu.cn

**Yingxi Lu**
IIIS, Tsinghua University
lu-yx23@mails.tsinghua.edu.cn

**Zhuo Cao**
IIIS, Tsinghua University
caozhuo23@mails.tsinghua.edu.cn

**Abstract:** In this project, we reproduce a previous research work Dynamic Handover[1], and try to improve it to have better generalizability on object's physical properties by adding physical property estimatior networks onto the base policy. We do experiments on different physical properties and test the accuracy of the estimator. We find that our design can help the policy generalize to a broader physical property range in some cases, and discuss the reason why it succeeds or fails.

**Keywords:** Robot Learning, Generalizability

## 1 Introduction

Dynamic dexterous manipulation remains challenging due to fast contact dynamics, underactuation, and severe sim-to-real gaps. Among such tasks, dynamic handover (throwing and catching) is particularly demanding: the thrower must generate a precise release state, while the catcher must react to a rapidly moving object under partial observability and model mismatch. Recent work Dynamic Handover [1] demonstrates that with multi-agent reinforcement learning and a learned trajectory/goal estimator, two dexterous hands can achieve agile throw-and-catch behaviours with promising sim-to-real transfer.

Despite these advances, a practical deployment challenge is *generalization across object physical properties*. Humans can throw objects with significantly different mass, scale and elastoplasticity. But for robots, while domain randomization improves robustness, we find that a policy trained within a limited physical-parameter range can still degrade significantly when tested out-of-distribution (OOD).

In this project, we reproduce Dynamic Handover [1] in IsaacGym and study how to improve physical-property generalization with minimal changes to the overall pipeline. Inspired by the trajectory estimator module in the original paper and by system identification / online adaptation literature, we introduce **Physical Property Estimator**: a lightweight neural network that infers time-invariant object properties from history of object positions and robot proprioception. The predicted physical parameters are then appended to the observation of the base MARL policy, enabling the policy to adapt its actions conditioned on estimated dynamics.

Our experiments focus on three parameters supported reliably by IsaacGym randomization: **mass**, **friction**, and **scale**. We train policies on a narrower randomization range and evaluate them on a wider OOD range. We find that conditioning on estimated mass significantly improves OOD success rates, while friction has limited influence in this task and scale generalization remains difficult. We further analyze the behavior patterns and discuss when estimator-conditioned policies are expected to help.

In summary, our contributions are:

1. a faithful reproduction of Dynamic Handover in IsaacGym with some training observations
2. a simple estimator-conditioned extension for physical-property generalization
3. an empirical study showing strong gains on mass OOD generalization but failure cases on scale, together with an interpretation based on continuous vs. discontinuous behavior changes.

## 2 Related Work

### 2.1 Dexterous Manipulation

In recent years, the robotics community has increasingly focused on dexterous manipulation due to its great flexibility and human-like dexterity. For example, in-hand object spinning[2][3] is a broadly studied topic and its great difficulty lies in both dexterity and sim-to-real gap. However, most dexterous hand research nowadays still focuses on quasi-static tasks. The operation speed of dexterous hand is an order of magnitude slower than humans, making them very inefficient to be deployed for real world tasks. While the humanoid and quadrupedal community has seen rapid advancements in agile locomotion[4][5], very few work in dexterous hand aim to complete high-speed and high-dynamic tasks.

Dynamic Handover[1] is among the few work that uses dexterous hands and robotic arms to perform highly dynamic tasks. It focuses on enabling robots to perform the complex task of throwing and catching objects using two dexterous hands attached to robot arms. The goal is to achieve high-speed, dynamic, and precise object handovers that mimic human-like collaboration. Our project is mainly based on their work, so we will go into some more details of their work.

**Setup:** Two Allegro Hands, each individually mounted on separate XArm-6 robots, are arranged in a face-to-face configuration. On initialization, an object is placed in the thrower's palm. The thrower needs to throw the object to the pre-defined goal, and the catcher needs to catch the object at that goal, and hold it tightly. A RealSense D435 camera is used for real-time object position tracking, which is oriented towards the working space. The whole policy is a state-based policy.

**Key contributions:** The authors leverage Reinforcement Learning to train a policy in simulation and then perform sim-to-real transfer on the real robots. Specifically, they introduce three key technical contributions

1. **Multi-Agent Reinforcement Learning (MARL).** The authors propose to tackle this problem as a multi-agent problem, with each hand being one agent. The two agents use different observation space. This helps improve the coordination of two agents during manipulation, which allows better sim-to-real transfer.
2. **Dynamic dexterity.** The authors use RL with randomization to allow the network to learn to handle diverse dynamics. They apply various physical property randomization during training. This is a very common trick nowadays.
3. **Object trajectory prediction.** To further increase the robustness for catching, the authors introduce a model to predict the object's future trajectory and destination ahead in real time. Instead of pre-defining the object destination and asking both hands to follow, the catch policy will take the predicted object's destination position as input in a close-loop manner. This allows flexible real-time adjustment of the catch policy.

**Training pipeline:** The training follows a three-stage pipeline:

1. Train a base policy using Multi-Agent Reinforcement Learning (MARL), where the catcher knows a pre-defined throwing goal.
2. Freeze the base policy, then train a goal estimator using supervised learning.
3. Replace the catcher's pre-defined goal with the estimated goal, and fine-tune both components together.

### 2.2 System Identification

System identification in robot manipulation enables accurate modeling of the robot's and object's dynamic properties, which significantly improves control performance and robustness to real-world variations. It facilitates the transfer of simulation-trained policies to real hardware, enhances safety, and allows adaptive strategies for interacting with new or changing environments. Some cutting-edge research in this area include, [6], which leverages differentiable simulation to calibrate object properties by using information from the robot, without relying on data from the object itself; [7], which adapts the simulation environment dynamics to real world dynamics without requiring gradient updates by leveraging past interaction histories as context; [8], which proposes an end-to-end self-supervised learning framework for physical property identification through active tactile exploration, which is then directly applied to dynamic manipulation tasks like swinging and tossing objects.

Beyond solely identify system properties, to cope with more complex real-world environments, online dynamics adaptation methods are needed. Rapid Motor Adaptation (RMA) [9] learns to encode recent state histories into a latent representation of environment dynamics by distilling knowledge from a privileged teacher encoder. [10] designs a dual-history architecture to overcome the sim2real dynamics gap with history information over varying horizons. DWL [11] utilizes a denoising world

model to predict environment configuration and privileged robot states simultaneously from history information. Any2Track [12] we introduce dynamics-aware world model prediction as a proxy task to learn better dynamics representations from the history buffer. As shown in Figure 1, it is verified that a relatively small, well-generalized network can serve as an effective guide to the base model.

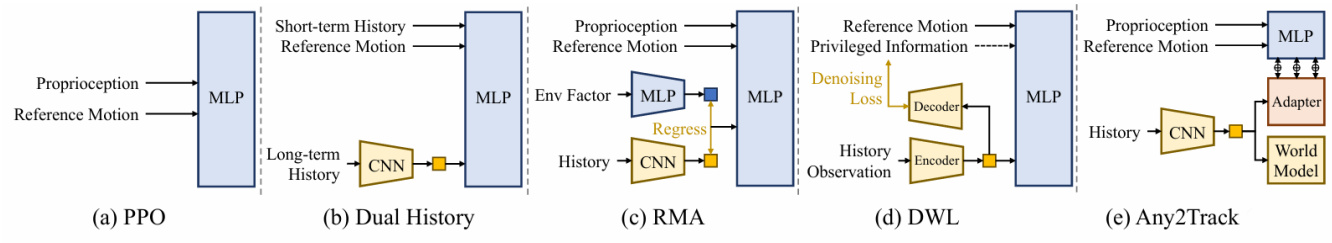



Figure 1: Comparison of different online adaptation methods. [12]

In our project, we leverage object's position and robots' proprioception information in history to do system identification on the object's physical properties including scale, mass and friction using neural networks.

## 3 Our Method

Our goal is to build a system that achieves better generalizability in object's physical properties, so that the policy can handle heavier, larger, or smoother objects that are not seen during training.

In Dynamic Handover[1], the policy input is restricted to joint positions, target goal, predicted goal (given by the goal estimator) and object's position. The policy is ignorant to the physical properties of object, like geometry, scale, mass, e.t.c.. To humans' instinct, these properties should be crucial to the policy's success. For example, humans will apply a larger force when the object is heavier to hit the desired target, and we will use different hand poses to grasp objects of different sizes.

We propose that a small, well-generalized network can serve as an effective guide to the base model if the base model cannot generalize well. So our idea is similar to the design of 'trajectory estimator module' in the original paper. We can estimate object properties by a neural network $\phi_\theta$, taking in past frames of object's position and robots' proprioception as input. Then we add these physical information into the observation space of the base policy. When explicitly informed of object's physical properties, we propose that the RL base policy will learn more behaviour patterns that help it adapt to different physical properties, like throwing lighter object with smaller force.

The training process is the same as in original paper. We use Adam to optimize the L2 distance between the predicted physical parameters and ground truth physical parameters until convergence in stage 2:

$$L(\theta) = \|\phi_\theta(h_{t-k:t}) - G\|^2,$$

where $h_{t-k:t}$ is object's position and robots' proprioception from $t-k$ to $t$ frames. $G$ is the ground truth physical property (time invariant).

## 4 Experiments

In this section, we provide our experimental results based on official codebase. Because we do not have access to real dexterous hands, we do our whole project in IsaacGym simulation. We aim at addressing the following three questions:

**Q1:** Can the method in original paper solve the dynamic handover problem well?

**Q2:** Can our method help the policy better generalize on object's physical properties? Does the physical property estimator give accurate results?

**Q3:** Can our method help the policy learn behaviour patterns to adapt to different physical properties?

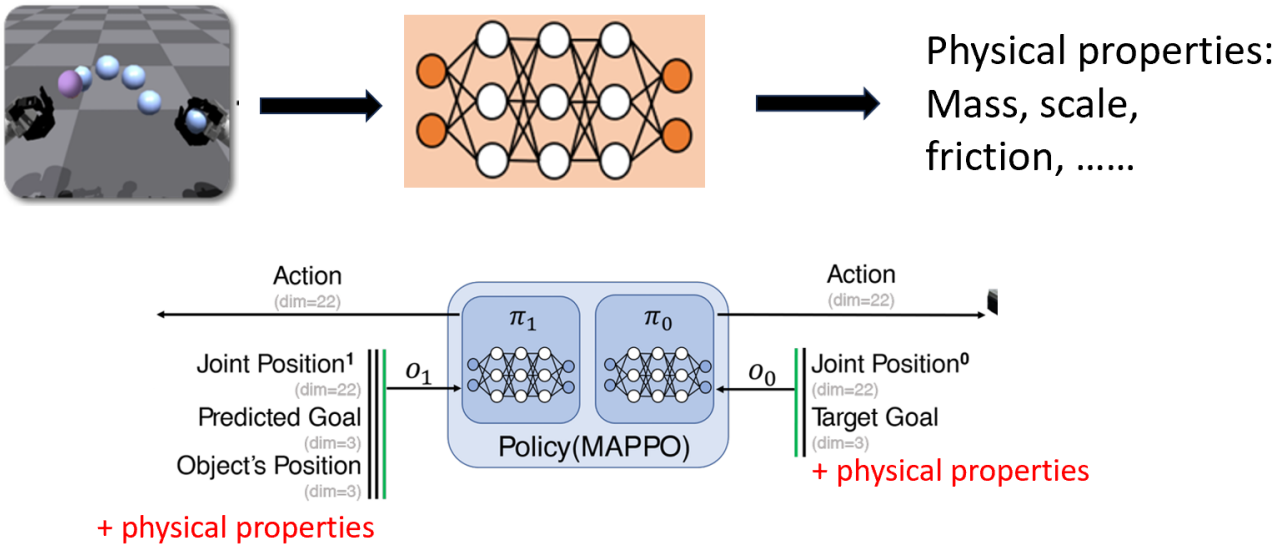



Figure 2: **Our method:** We train a physical property estimator network to derive object's physical properties from history, and add physical information to the observation space of base policy.

### 4.1 Answer to Q1

We run the training process of Dynamic Handover[1] on our device and validate that their method can work. We have two interesting findings: First, the MARL process is unstable. It sometimes gets stuck in local optima of low rewards and fail to explore high-reward regions of the state space, or crashes after it has already hits a high reward before. Second, the official codebase set a very small episode length (30 steps), and there is only around 10 steps for the thrower to 'prepare' before the object leaves its hand. We thought it is unreasonable because intuitively preparing longer should help it hit the target more accurately. However, after we set the episode length to 60 steps, the policy fails to learn to throw the object. The thrower just stretches its arm to move the object closer to the catcher, and the catcher does not learn anything. We guess this can be solved through reward engineering, but as this is not the focus of our project, we just leave it as it was. (Episode length set to 30 steps.)

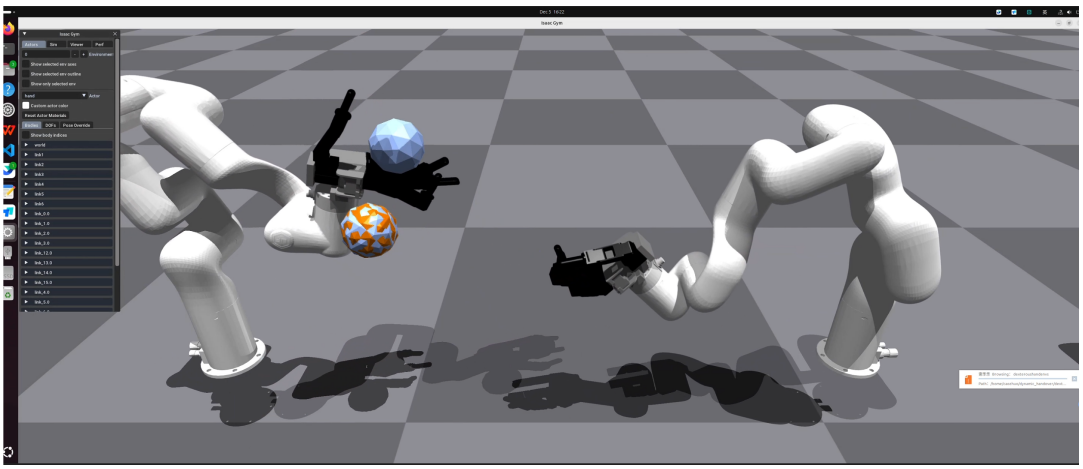


Figure 3: **Our reproduction:** The right arm is thrower and the left arm is catcher. The orange ball is the estimated goal, the blue ball that overlaps with it is the target, and the other blue ball is the actual object.

### 4.2 Answer to Q2

Common physical parameters that can affect the policy's performance include: scale, mass, friction, center of mass and inertia. We find that randomization of center of mass is not supported by IsaacGym, and randomization of inertia sometimes causes simulation bugs. Therefore, we focus on three physical parameters: **Mass**, **Friction** and **Scale**. To test generalizability, we train policies with a smaller randomization range, and test on a larger range, shown in table 1. For training, we do uniform and continuous randomization on **Mass** and **Friction**; uniform and discrete randomization on **Scale** (because continuous randomization causes bugs). For test, we sample discrete values from the test range and for every value, we test 20 times. We also train a base policy on the test range. (**BaseInDomain**)

**Implementation details of physical property estimator:** We adopt a Multi-Layer Perceptron (MLP) architecture with three hidden layers: an input layer projecting to 512 dimensions, followed by successive layers of 256 and 128 neurons, and an output layer, all activated by Exponential Linear Unit (ELU) functions. We set history length $k = 20$ steps, and try two settings: **w/ torque**, input includes object's position (dim=3)and two robots' dof position (dim=44); **w/o torque**, input includes object's position (dim=3), two robots' dof position (dim=44), and dof torque (dim=44).

The generalizability of the **Base** policy, **Teacher** policy (receives ground truth physical parameters in its observation space), and policy with physical parameter estimator (**PPPE**) on the 3 physical properties are shown in table 2. We plot the success rate in-domain and out-of-domain of all 5 policies on the 3 physical parameters in figure 4.

The result is interesting. Firstly, all policies perform well for in-domain test, showing that the original method has good robustness. As to generalizability, by checking out-of-domain test results, we can see that:

For **Mass**, our method works. Both **PPPE w/ torque** and **PPPE w/o torque** show great improvement over BASE in generalizability, and naturally, they do not perform as well as **Teacher** and **BaseInDomain**.

For **Friction**, all the policies perform well both in-domain and out-of-domain. This may indicate that **Friction** is not a crucial factor in this task.

For **Scale**, our method fails. Except **BaseInDomain**, the other four policies all perform poorly in out-of-domain test.

| Parameter | Base Value | Training Range | Test Range |
|---|---|---|---|
| **Mass** | 0.228670 | $[0.5, 1.5]$ | $[0.2, 2.2]$ |
| **Friction** | 1.00 | $[0.7, 1.3]$ | $[0.01, 10.0]$ |
| **Scale** | 0.05 | $[0.8, 1.2]$ | $[0.5, 1.5]$ |

Table 1: Physical property randomization ranges in training and test. The randomize factor is multiplied to the base value during training and test.

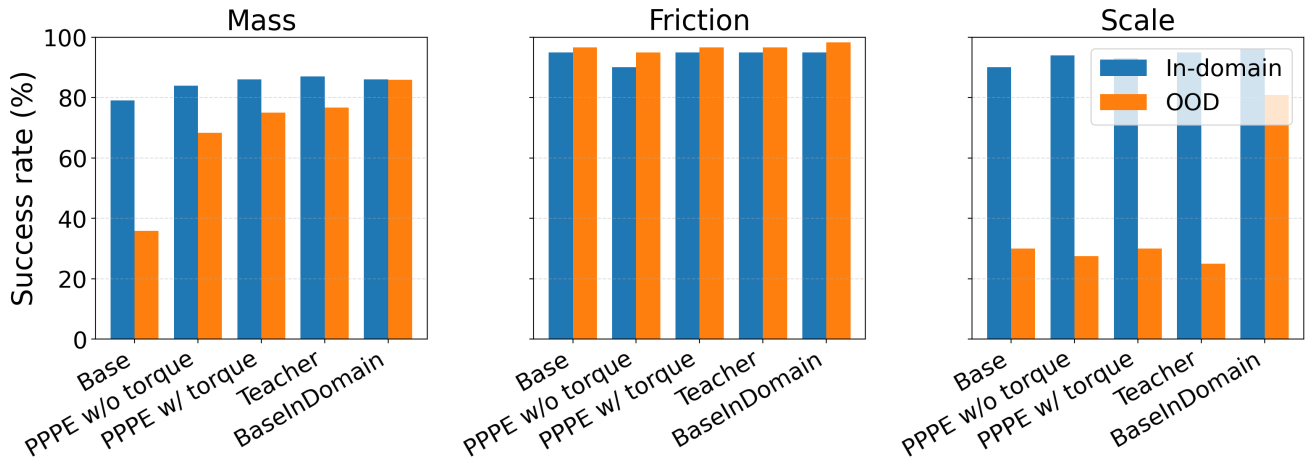



Figure 4: Success rate in-domain and out-of-domain of all 5 policies on **Mass**, **Friction** and **Scale**.

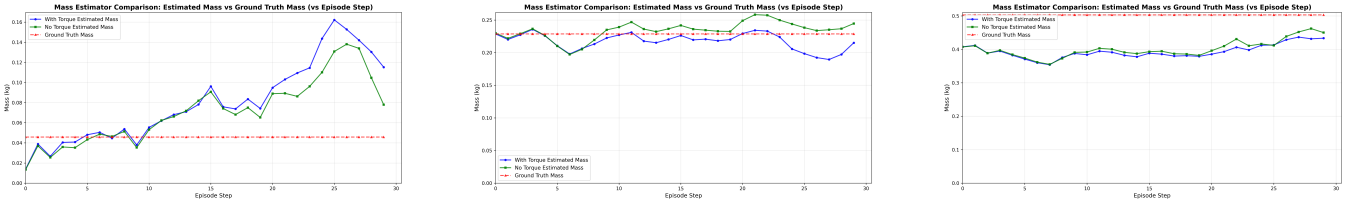
| Mass randomize factor | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 | 1.2 | 1.4 | 1.6 | 1.8 | 2.0 | 2.2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Base** | 6/20 | 2/20 | 14/20 | 15/20 | 16/20 | 18/20 | 16/20 | 13/20 | 7/20 | 11/20 | 4/20 |
| **PPPE w/o torque** | 15/20 | 15/20 | 16/20 | 18/20 | 17/20 | 16/20 | 17/20 | 15/20 | 17/20 | 10/20 | 10/20 |
| **PPPE w/ torque** | 14/20 | 17/20 | 19/20 | 16/20 | 20/20 | 14/20 | 17/20 | 19/20 | 20/20 | 13/20 | 7/20 |
| **Teacher** | 20/20 | 15/20 | 14/20 | 20/20 | 18/20 | 19/20 | 16/20 | 16/20 | 18/20 | 13/20 | 10/20 |
| **BaseInDomain** | 18/20 | 16/20 | 17/20 | 16/20 | 19/20 | 18/20 | 16/20 | 17/20 | 16/20 | 17/20 | 19/20 |

| Friction randomize factor | 0.01 | 0.1 | 1.0 | 10 |
|---|---|---|---|---|
| **Base** | 18/20 | 20/20 | 19/20 | 20/20 |
| **PPPE w/o torque** | 17/20 | 20/20 | 18/20 | 20/20 |
| **PPPE w/ torque** | 20/20 | 20/20 | 19/20 | 18/20 |
| **Teacher** | 19/20 | 19/20 | 19/20 | 20/20 |
| **BaseInDomain** | 19/20 | 20/20 | 19/20 | 20/20 |

| Scale randomize factor | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Base** | 0/20 | 2/20 | 6/20 | 18/20 | 20/20 | 15/20 | 20/20 | 17/20 | 14/20 | 12/20 | 2/20 |
| **PPPE w/o torque** | 0/20 | 0/20 | 5/20 | 16/20 | 20/20 | 20/20 | 19/20 | 19/20 | 15/20 | 13/20 | 0/20 |
| **PPPE w/ torque** | 0/20 | 1/20 | 8/20 | 18/20 | 20/20 | 18/20 | 19/20 | 18/20 | 15/20 | 12/20 | 0/20 |
| **Teacher** | 0/20 | 1/20 | 5/20 | 19/20 | 20/20 | 18/20 | 19/20 | 19/20 | 12/20 | 12/20 | 0/20 |
| **BaseInDomain** | 9/20 | 15/20 | 20/20 | 19/20 | 18/20 | 20/20 | 19/20 | 20/20 | 18/20 | 18/20 | 17/20 |

Table 2: The generalizability of different policies on object mass, friction and scale. Cells highlighted in green are beyond training scope.

We also test the accuracy of the **Mass** estimator (w/ and w/o torque). We roll out 30 successful trajectories for every value in test range, and calculate the mean relative error(MRE) of estimated value and ground truth. We also plot the mean estimated mass at every step, and show one mid-range value and two extreme value results in figure 5. More results are in appendix A. The results of **PPPE w/ torque** and **PPPE w/o torque** are similar. Both estimator implementations perform well in in-domain tests. They also have the sense that heavier objects are heavier. However, they still over-estimate too-light cases and under-estimate too-heavy cases. It is worth noticing that, while MRE on too-light objects is quite high, both implementations perform well on the first 10 steps, when the thrower is still in touch with the object. This means the catcher can use a near accurate mass estimation to facilitate its throwing, which is crucial to task success.




(a) Mass randomize factor=0.2 (b) Mass randomize factor=1.0 (c) Mass randomize factor=2.2

Figure 5: Mass estimator's accuracy on different mass randomization factors. The red line is ground truth, blue line is **PPPE w/ torque**, and green line is **PPPE w/o torque**. Notice that the label of y axis is mass, which is base value (0.228670) multiplied by the randomize factor.

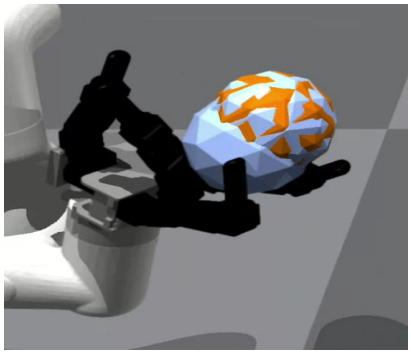
| Mass randomize factor | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 | 1.2 | 1.4 | 1.6 | 1.8 | 2.0 | 2.2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **PPPE w/o torque MRE** | 69.5% | 28.4% | 9.2% | 3.6% | 5.8% | 5.1% | 4.8% | 4.2% | 13.4% | 16.0% | 21.7% |
| **PPPE w/ torque MRE** | 86.2% | 21.1% | 7.0% | 4.4% | 5.5% | 5.5% | 10.7% | 5.4% | 11.9% | 16.4% | 20.3% |

Table 3: the accuracy of the **Mass** estimator on different randomize factors.

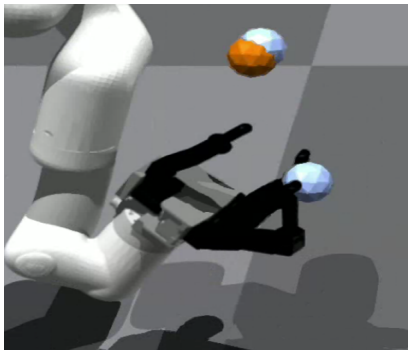
### 4.3 Answer to Q3

To answer whether our method can help the policy learn behaviour patterns to adapt to different physical properties, we show some visualization results in this section.

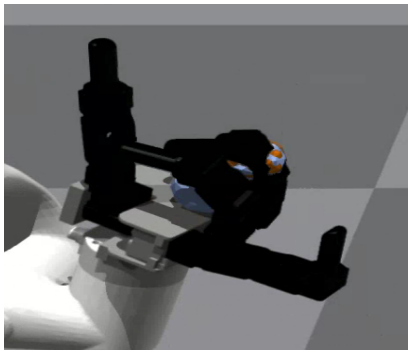
First we want to explain why adding the scale estimator does not help the policy to generalize. In figure 6, we can see that the **Teacher**'s thrower has the behevior pattern of extending the index and ring finger and raise the middle finger, clamping the object in between. This behaviour pattern cannot succeed for too-small objects because it will slip through the gap between the index and ring fingers. So even if the policy is informed of the object's accurate scale in **Teacher**, it cannot learn a correct behaviour pattern for too-small objects in training because all test cases can be solved by another pattern.



(a) **Teacher**, scale randomization factor=1.0



(b) **Teacher**, scale randomization factor=0.5

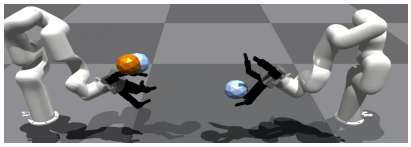


(c) **BaseInDomain**, scale randomization factor=0.5

Figure 6

Then we want to explain how adding the mass estimator helps the policy to generalize. In figure 7, we can see that the **BASE**'s thrower applies a too-large force to the light object. The object flies too far and fast that the catcher cannot grasp it tightly. In figure 8, we can see that after informed of the mass of the object, the thrower learns the pattern to adapt its force to different mass, so it applies a smaller force to the object, and now the catcher can catch it easily.

We suggest that, the core reason for the difference is that mass induces a largely continuous change in control, whereas scale can trigger a discontinuous change in the required interaction mode. Physical property estimators can help when the behaviour pattern varies smoothly with the estimated property. But it may fail when the policy needs to find new completely different behaviour patterns for out-of-domain tests. We need other methods like using a larger randomization range, or teacher-student method for generalization in this case.

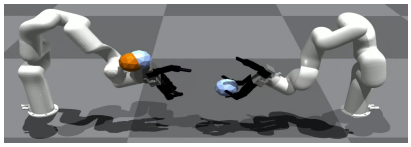


(a) **Base**, mass randomize factor=0.2, step 10

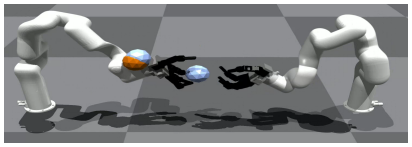
(b) **Base**, mass randomize factor=0.2, step 15

(c) **Base**, mass randomize factor=0.2, step 20

Figure 7



(a) **PPPE w/o torque**, mass randomize factor=0.2, step 10



(b) **PPPE w/o torque**, mass randomize factor=0.2, step 15

(c) **PPPE w/o torque**, mass randomize factor=0.2, step 20

Figure 8

## 5 Conclusion and Limitation

### 5.1 Limitation

Due to time limit, we have not conducted experiments on the accuracy of the scale estimator. We only focus on generalizability on the object's low-dimensional physical properties, but there are many other factors like object geometry, distance between the two hands, e.t.c. that can also impact policy's performance and may be harder to generalize on. The whole project is done in simulation and may encounter sim-to-real gap.

### 5.2 Conclusion

We represent Generalizable Dynamic Handover, a system that augments Dynamic Handover with a physical property estimator that infers object physical properties from history and feeds them back to the policy. In IsaacGym experiments with randomized mass, friction, and scale, we find that explicit conditioning on estimated properties can significantly improve out-of-distribution performance on mass, while friction has limited impact in this benchmark, and scale generalization remains challenging. We hope these results gives a guide on when lightweight system identification can effectively guide dynamic dexterous policies.

We make some other comments in appendix B.

### Acknowledgments

We would like to express our sincere thanks to Professor Li Yi for his computational resources and insightful instructions in the past year. Without him, it is unlikely that we will choose this topic. He did not give direct instructions on this project.

## References

[1] B. Huang, Y. Chen, T. Wang, Y. Qin, Y. Yang, N. Atanasov, and X. Wang. Dynamic handover: Throw and catch with bimanual hands. *arXiv preprint arXiv:2309.05655*, 2023.

[2] H. Qi, B. Yi, S. Suresh, M. Lambeta, Y. Ma, R. Calandra, and J. Malik. General in-hand object rotation with vision and touch. In *Conference on Robot Learning*, pages 2549–2564. PMLR, 2023.

[3] X. Liu, H. Wang, and L. Yi. Dexndm: Closing the reality gap for dexterous in-hand rotation via joint-wise neural dynamics model. *arXiv preprint arXiv:2510.08556*, 2025.

[4] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi. Agile but safe: Learning collision-free high-speed legged locomotion. *arXiv preprint arXiv:2401.17583*, 2024.

[5] T. He, J. Gao, W. Xiao, Y. Zhang, Z. Wang, J. Wang, Z. Luo, G. He, N. Sobanbab, C. Pan, et al. Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills. *arXiv preprint arXiv:2502.01143*, 2025.

[6] P. Y. Chen, C. Liu, P. Ma, J. Eastman, D. Rus, D. Randle, Y. Ivanov, and W. Matusik. Learning object properties using robot proprioception via differentiable robot-object interaction. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5997–6004. IEEE, 2025.

[7] X. Zhang, S. Liu, P. Huang, W. J. Han, Y. Lyu, M. Xu, and D. Zhao. Dynamics as prompts: In-context learning for sim-to-real system identifications. *IEEE Robotics and Automation Letters*, 2025.

[8] C. Wang, S. Wang, B. Romero, F. Veiga, and E. Adelson. Swingbot: Learning physical features from in-hand tactile exploration for dynamic swing-up manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5633–5640. IEEE, 2020.

[9] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots, 2021. URL https://arxiv.org/abs/2107.04034.

[10] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath. Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control, 2024. URL https://arxiv.org/abs/2401.16889.

[11] X. Gu, Y.-J. Wang, X. Zhu, C. Shi, Y. Guo, Y. Liu, and J. Chen. Advancing humanoid locomotion: Mastering challenging terrains with denoising world model learning, 2024. URL https://arxiv.org/abs/2408.14472.

[12] Z. Zhang, J. Guo, C. Chen, J. Wang, C. Lin, Y. Lian, H. Xue, Z. Wang, M. Liu, J. Lyu, H. Liu, H. Wang, and L. Yi. Track any motions under any disturbances, 2025. URL https://arxiv.org/abs/2509.13833.

# A More Results on Estimator Accuracy Test

More results on mass estimator accuracy test are shown in this section.

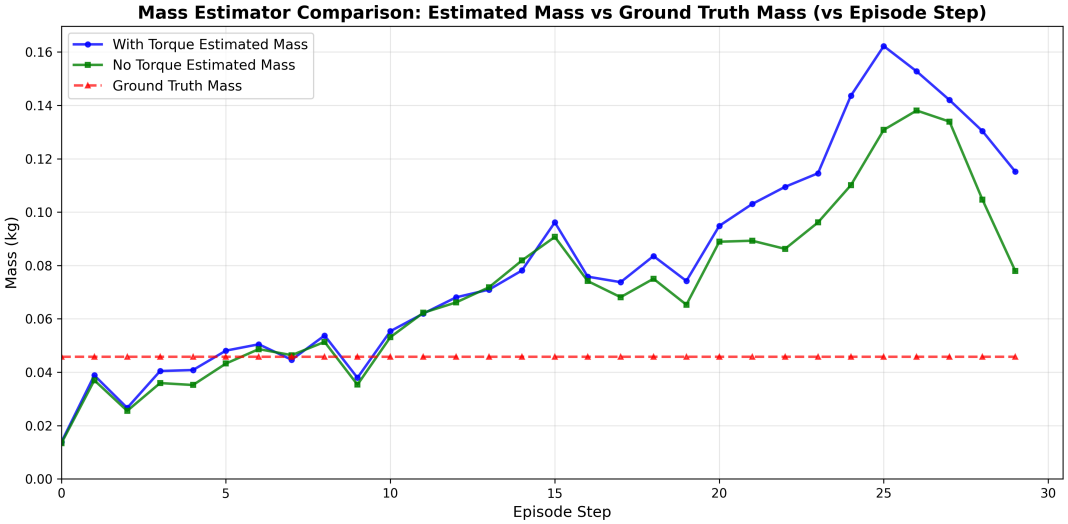



Figure 9: Mass randomize factor=0.2

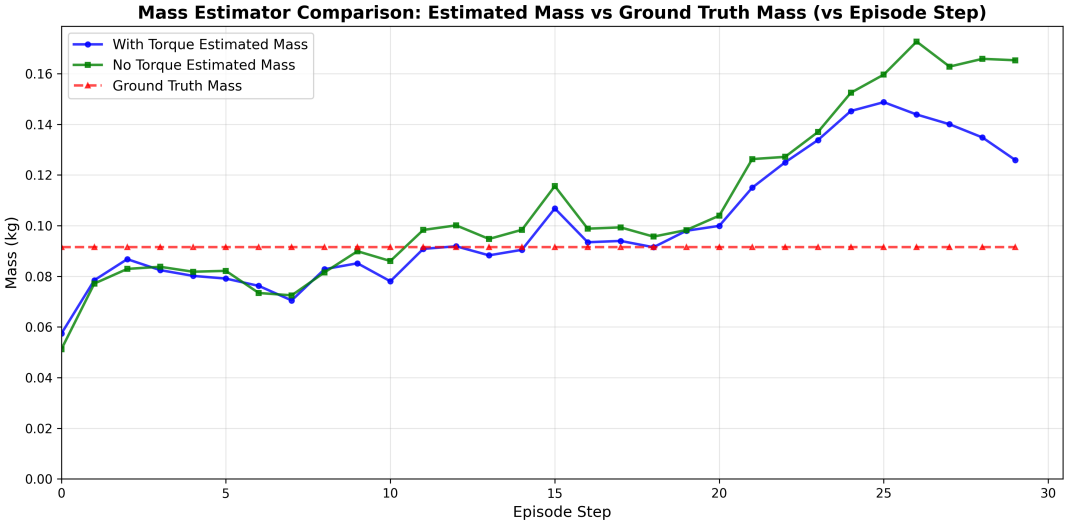



Figure 10: Mass randomize factor=0.4

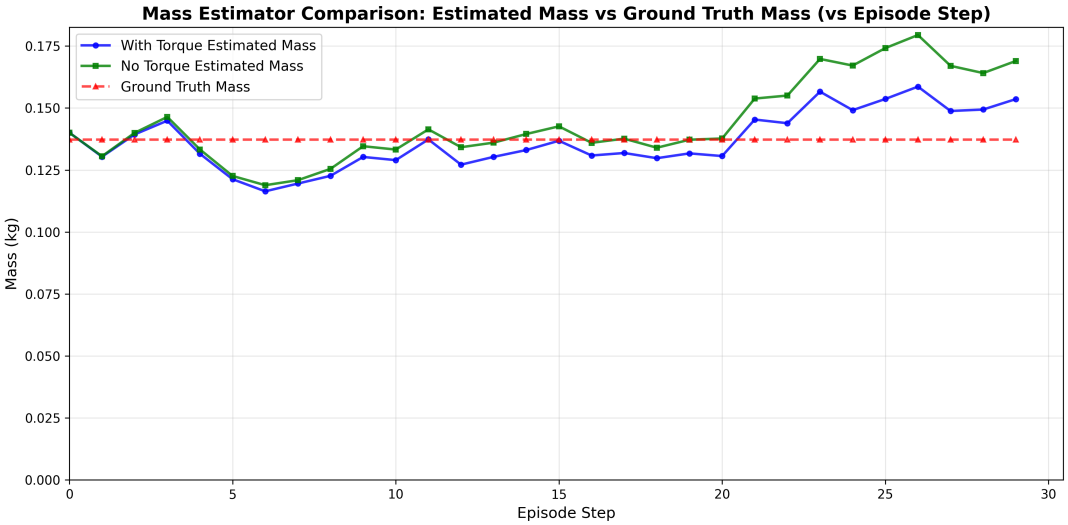



Figure 11: Mass randomize factor=0.6

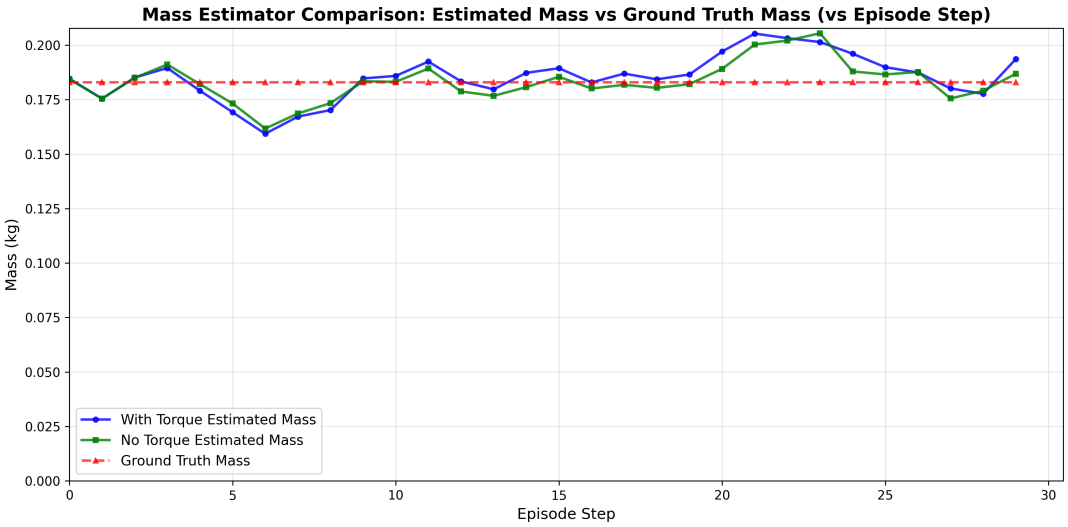



Figure 12: Mass randomize factor=0.8

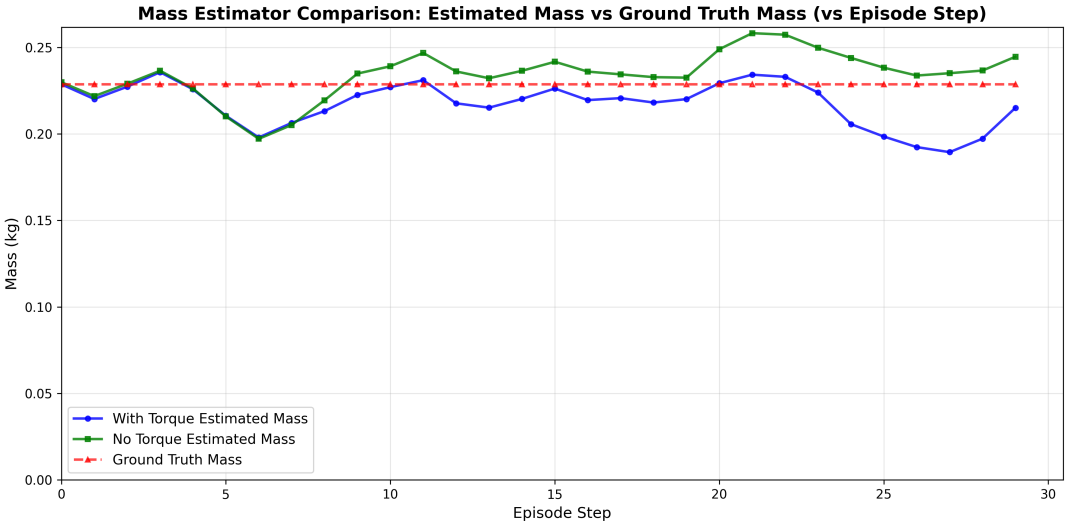



Figure 13: Mass randomize factor=1.0

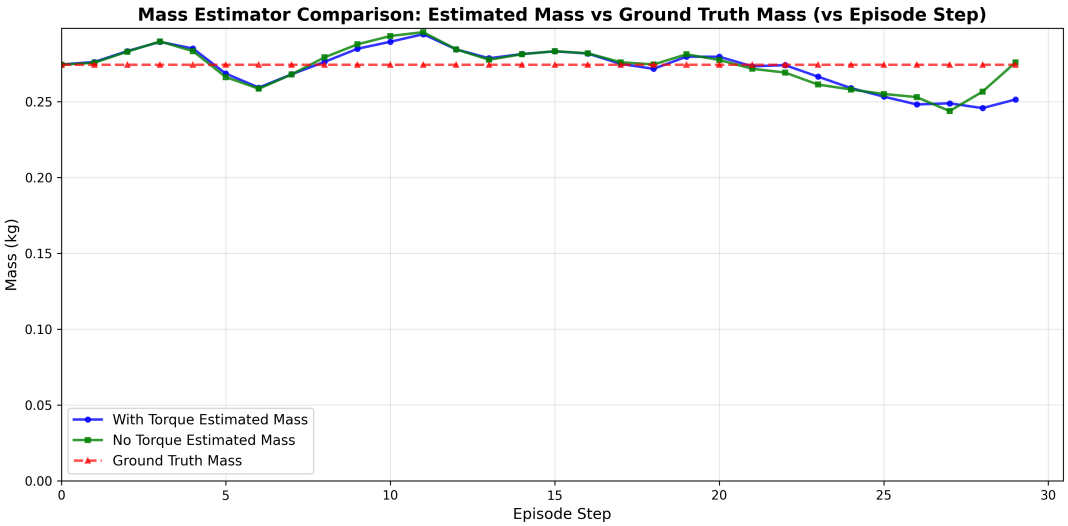



Figure 14: Mass randomize factor=1.2

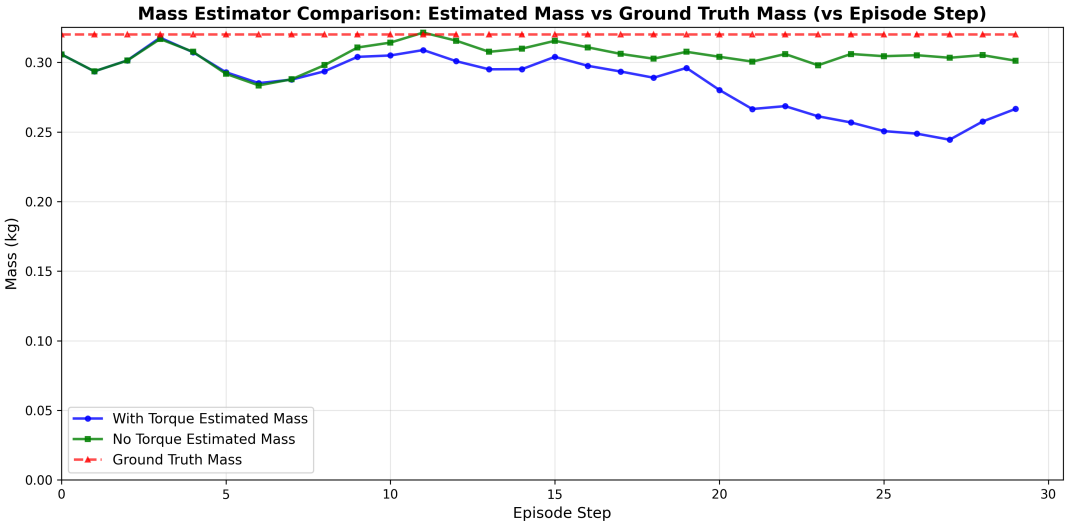



Figure 15: Mass randomize factor=1.4

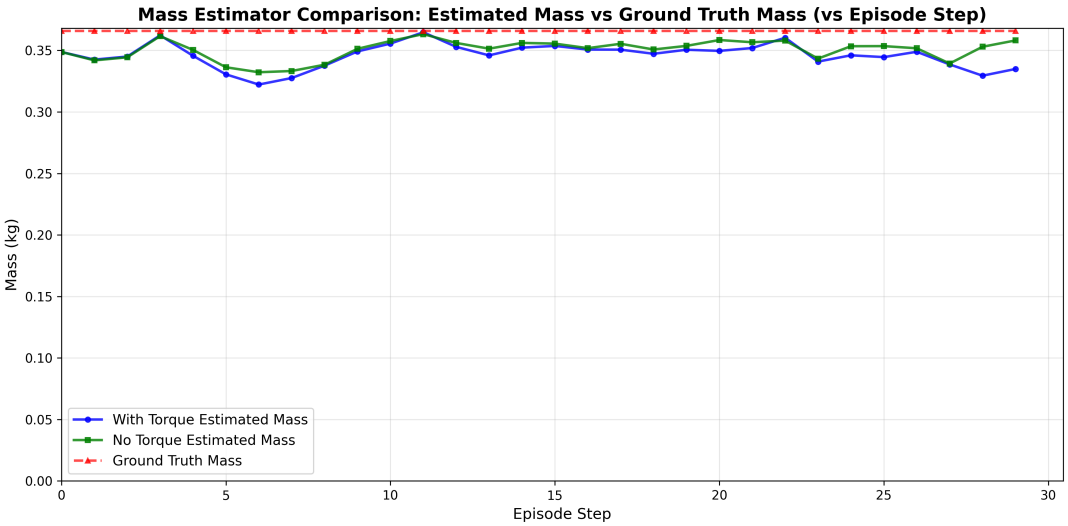



Figure 16: Mass randomize factor=1.6

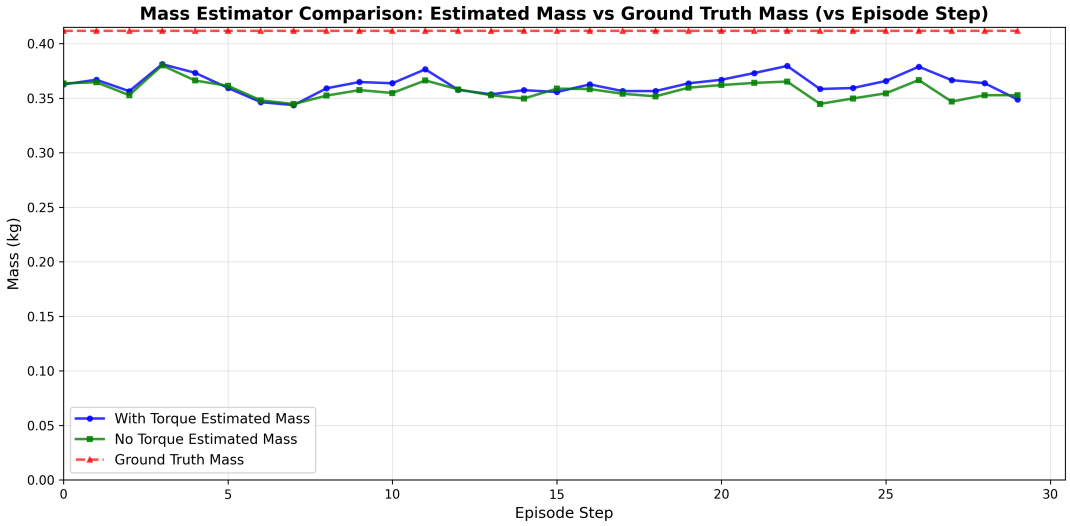



Figure 17: Mass randomize factor=1.8

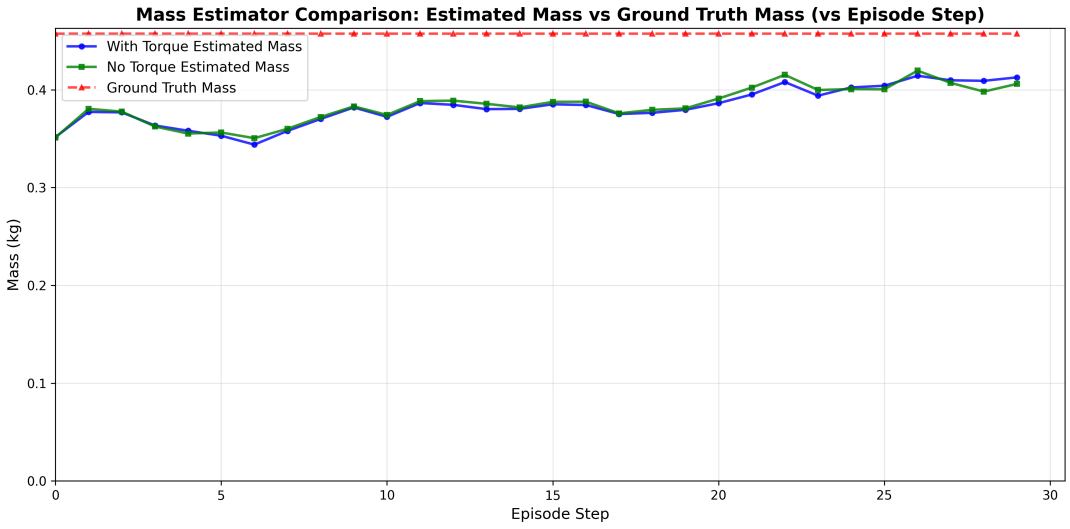



Figure 18: Mass randomize factor=2.0

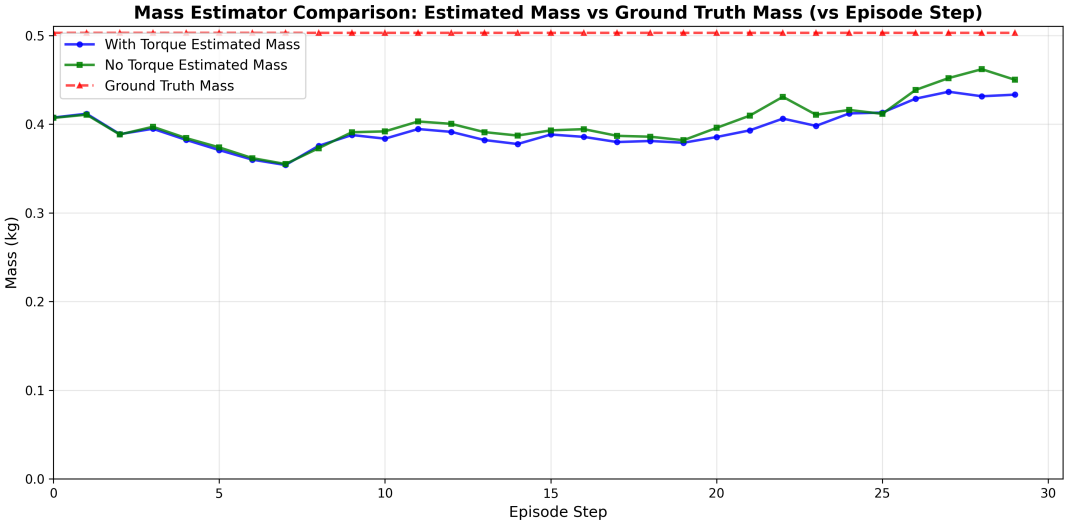



Figure 19: Mass randomize factor=2.2

## B Other Comments

The original codebase contains bugs in args parsing and simulation process. It took us some time to make it work correctly.

The only visualization device we have is a 5090 host with monitor. However, PyTorch versions compatible with RTX 50-series GPUs strictly require Python $\geq$ 3.9, which means legacy codebases that rely on Python versions below 3.9 cannot be used. Unfortunately, since NVIDIA has discontinued maintenance of IsaacGym, it only supports up to Python 3.8, creating a hard incompatibility with PyTorch. To solve this problem we had a hard time building PyTorch wheel from source code according to https://blog.csdn.net/m0_56706433/article/details/148902144.

Once we changed the initialization method of policy networks, and the training will completely fail. This bug also confused us for a long time.