MINI PROJECT 4: LEARNING

Due: Monday, December 08 11:59pm EST

The objective of MP4 is to study **learning-based robot manipulation**, involving two tasks:
i) pushing a container to a target position using the robot gripper, and
ii) picking up a cube and placing it into the container.

For each task, we will provide a **robot teleoperation dataset** consisting of 30 teleoperation trajectories collected under slightly different initial object configurations (i.e., varying initial poses of the container or cube). Each trajectory includes:

1) **7-DoF Robot joint positions (6 arm joints, 1 gripper)**,

2) **RGB image stream** from one camera,

3) **RGB-D image stream** from another camera.

For each trajectory, the dataset has already been synchronized. An example of the synchronized image stream for each task is shown in Figure. 1.

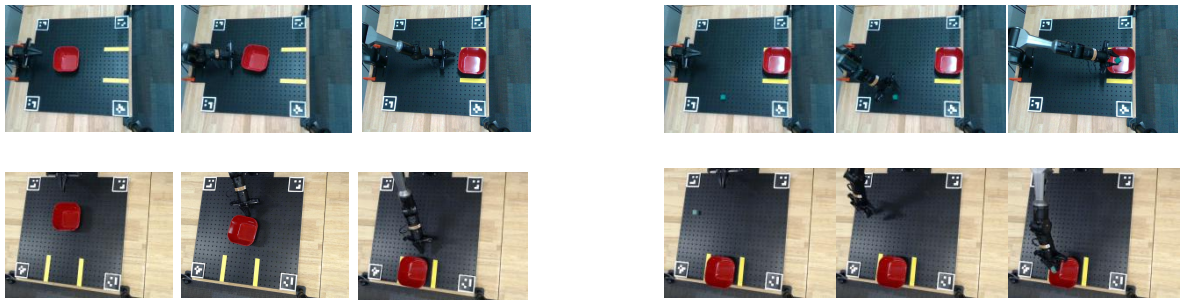

Figure 1. An example of synchronized RGB image streams from two cameras for a single trajectory in the container-push task (left), and the pick-and-place task (right). Note that the top camera streams are from the RGBD camera.

During evaluation, we will first assess performance on each task individually, and then evaluate **sequential manipulation** (push → pick-and-place). From the learning perspective, you may choose to i) train two **separate policies** for each task, or ii) train **one unified policy** using the combined dataset.

We will also provide an **equivalent Mujoco simulation dataset** in the same format. You can train your policies in simulation first and evaluate their performance there, allowing you to observe how the learning loss relates to actual manipulation performance. The purpose of providing a separate

simulation dataset is to avoid the real-to-sim gap that may arise from differences in robot dynamics, visual textures, and other factors. Since the real-world and simulation datasets share the same data format, your learning pipeline can be applied to both seamlessly.

For this assignment, you will also have full freedom in designing your learning pipeline. For example, you may choose the policy observation space — using a single RGB image, two RGB images, or incorporating depth. You may also select the perception module, such as a pre-trained ResNet (https://docs.pytorch.org/vision/main/models/resnet.html), a more advanced ViT model (https://docs.pytorch.org/vision/main/models/vision_transformer.html), or any architecture you prefer. Likewise, you can decide on the policy model, whether it is a state-of-the-art diffusion model (https://colab.research.google.com/drive/18GIHeOQ5DyjMN8iIRZL2EKZ0745NLIpg?usp=sharing), an ACT-based architecture (https://github.com/tonyzhaozh/act), or a simpler MLP/RNN/LSTM design (https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html).

A. Completion of the dataloader and submission of a system report explaining the design choices for each component (3 pts)

You are first required to complete the dataloader so that it can load all relevant information from the provided teleoperation dataset. In particular, it should read the robot joint positions and image streams, and organize them into a format suitable for policy learning. You can find useful guidance in the official PyTorch documentation:
https://docs.pytorch.org/tutorials/beginner/basics/data_tutorial.html.

You are also required to provide a short report or a block diagram of your designed learning system. This should describe: (i) what inputs are fed into the system, (ii) what model architecture you use, and (iii) what learning loss you optimize. Having a clear picture of this design will also make it easier to develop your code.

Grading criteria: 1 pt — a functional dataloader, 2 pt — a clear system report.

B. Training performance on the test trajectories (6 pts)

A key concept in learning-based systems is **generalization**—the model should perform well not only on the trajectories it was trained on, but also on **unseen (test) trajectories**. To achieve this, you should randomly split the 30 teleoperation trajectories into 25 trajectories for training, and remaining 5 ones for validation.

During training, if the implementation is correct and the model is learning meaningful behaviors, you should observe **two loss curves**: i) Training loss - evaluated on the training trajectories, and ii) Validation loss - evaluated on the reserved validation trajectories. If the validation loss begins to keep increasing while the training loss still goes down, it typically indicates model overfitting and you might consider terminating the training.

Typically, the model checkpoint with the **lowest validation loss** should be selected for the final evaluation, as it is expected to best encode behavior that generalizes. For evaluation, we will run your chosen checkpoint on **5 test trajectories** that are **not released** (held by the TAs). Since the test data are sampled from the **same range of initial object conditions** as train/validation, a model that generalizes well should show **similar prediction loss** on the test set as on the validation set.

Grading criteria: 2 pt — Code review for each learning component, 2 pts — Demonstration of both learning curves (training loss and validation loss), 2 pts — Reasonable prediction loss on the test set.

C. Policy evaluation on the manipulation tasks (6 pts)

You will be asked to evaluate the saved models (e.g., the checkpoint that achieves the lowest error on the validation dataset) **in both the real-world setting and the simulation setting (note that you will train separate models)**. Ideally, when rolling out the trained model, the robot should be able to successfully perform the two manipulation tasks.

During evaluation, **the model must receive the same type of observations used during training**. For example, if your policy was trained using a single RGB image stream, then during inference your model must also take the RGB image from the ***same camera*** as its input.

Grading criteria: 2 pt — Simulation performance, 4 pts — Real-world performance.

Further Tips:

I) Start early! This assignment provides very little starter code, so it will require **more effort** on the coding and debugging side compared to the previous assignments.

II) Along with this document, we provide access to **one sample real-world trajectory for each task** (download link here). You will be able to access and download the **full datasets** very soon.

III) We will also provide the **simulation dataset** and a **simulation inference code template** soon, which will help you evaluate your trained model's performance inside the simulator.

IV) Since this is the final assignment and the timeline has shifted from the original schedule, some students may leave early and be absent during the in-lab grading. **We will make reasonable accommodations for such cases.**