Machine Learning for Robotics RBE 577

Project 4: ACT of Insertion – Imitation Learning for Peg Placement

DUE: Tuesday April 29 at 11:59 pm.

All written components should be typeset using LaTeX. A template is provided on Overleaf. However, you are free to use your own format as long as it is in LaTeX.

Submit two files: **(a)** your code as a zip file and **(b)** the written report as a pdf file. If you work in pairs, only one submission is required; include your partner's name in Canvas.

1 Programming Component (100 Points)

In this project, you will implement the Action Chunking Transformer (ACT) architecture for a peg-insertion task using a Universal Robot (UR10) in the MuJoCo simulation environment. The implementation pipeline is divided into three core phases: generating demonstration data, training the ACT model, and evaluating its performance.

Problem Statement and Motivation

The goal of this project is to enable a UR10 robot to perform precise peg-insertion tasks using a transformer-based imitation learning model. Specifically, you will implement the Action Chunking Transformer (ACT), which learns to map multi-modal observations—such as 3-camera RGB feeds and robot end-effector positions—to a sequence of robot control actions.

Unlike traditional reinforcement learning (RL), which explores the environment and learns through trial and error, ACT follows the imitation learning paradigm. Here, the model is trained offline on expert demonstration data. During training, ACT does not interact with the environment—it learns from previously collected trajectories that contain:

- RGB images from three camera views,
- End-effector positions and joint states,

Once trained, the ACT model receives a stream of multi-view camera inputs at inference time and directly outputs robot control commands at a fixed frequency, without requiring any reward signal or exploration. This paradigm allows for safe and efficient policy learning in complex environments, especially when real-world data collection is expensive or risky.

You will build this pipeline from scratch, beginning with data generation and culminating in a trained model capable of solving the insertion task in a closed-loop manner.

1.1 Demonstration Generation (30 Points)

In this phase, you will design hard-coded policies to perform peg insertions for two object shapes—square and triangle—in the MuJoCo simulation environment.

- Use `make_sim_eps.py` to generate scripted episodes demonstrating successful insertions.
- Implement separate policies for both square and triangle peg-hole insertions.
- Each episode should be saved in HDF5 format, containing:
  - Time-series data of end-effector states
  - Visual observations (camera images)
- Generate approximately 50 demonstrations for each peg type.

- For each peg shape, randomize the initial position over 3 distinct placements, with the X and Y coordinates perturbed by ± 1 cm.
- Save all demonstration data under the directory: `data/dataset-ur10_mb/`.
- Simulation parameters, including shape configuration and randomization range, can be customized via `config/sim_data_collection.yaml`.
- You can change the views of the three cameras defined but you will need to use 3 cameras feeds for training this partciular model.

1.2 Training the ACT Model (60 Points)

In this phase, you will implement and train the Action Chunking Transformer (ACT) model using the demonstration data generated earlier. Your task includes completing core modules and running the training loop.

- The ACT architecture is based on the DETR-style transformer with VAE components for learning latent action representations. You are required to implement the following components:
  - `detr/models/transformers.py` — Implement the transformer architecture used for attention-based policy learning. This includes:
    - Transformer encoder and decoder
    - Attention mechanisms
    - Residual and normalization flows
  - `detr/models/detr_vae.py` — Implement the VAE wrapper around the DETR transformer for behavior cloning. This includes:
    - Encoding the action sequences to obtain latent variables $(\mu, \log \sigma^2)$
    - Reparameterization and decoding latent representations into predicted actions
    - Query embedding setup and integration with camera/state inputs
  - `detr/models/position_encoding.py` — Implement positional encoding schemes (e.g., sinusoidal and learned embeddings) to provide temporal and spatial information for the transformer inputs.
- Implement the training loops in the script `imitate_episodes.py` with the configuration in `config/imitation_train.yaml` to begin training.
- Ensure your model supports both camera-based and proprioceptive state inputs as defined in the configuration file. The model checkpoints should be saved under `weights/`.
- Include in your report:
  - Loss curves over training epochs (e.g., reconstruction loss, KL divergence if applicable)
  - Architecture summary and hyperparameters

Note: You are encouraged to read through the original DETR and ACT papers to understand the rationale behind these module designs. Proper implementation of these components is critical to achieving good performance during evaluation.

1.3 Evaluation and Analysis (10 Points)

In this section, evaluate the trained ACT model and assess its performance.

- Evaluate using `imitate_episodes.py` with `config/imitation_eval.yaml`.
- Quantitatively assess success rate across different initializations and peg types.
- Visualize episodes, compare trajectory fidelity, and analyze robustness.
- Include rollout videos, trajectory plots, accuracy metrics in the final report.
- Discuss limitations and propose improvements.

2 Directory Structure

Your repository should follow this structure:

```
ACT-Insertion-Project/
├── assets/
├── config/
│   ├── sim_scripts/
│   ├── common.yaml
│   ├── imitation_train.yaml
│   ├── imitation_eval.yaml
│   └── sim_data_collection.yaml
├── data/
│   └── dataset-ur10_mb/
├── detr/
├── helpers/
├── lib/
│   ├── policies/
│   ├── metrics/
│   ├── rewards/
│   └── sim/
├── logs/
├── scripts/
├── weights/
├── make_sim_eps.py
├── imitate_episodes.py
├── environment.yml
└── requirements.txt
```

3 Submission Instructions

- Follow the provided directory structure.
- Submit your code as a zip file and your report as a PDF.
- Only one submission is required per team; include both names on Canvas.

4 Deliverables

- A detailed PDF report with:
  - Training curves (e.g., loss vs. epochs).
  - Figures showing sample rollouts and success rates.
  - Training curves (e.g., loss vs. epochs).
- A zipped codebase that adheres to the specified structure.

5 Milestones

To help you pace your work and receive timely feedback, this project will be graded based on milestone submissions.

Milestone 1: Demonstration Generation

Due Date: April 15, 2025 (11:59 PM)

By this date, you are expected to:

- Complete the hardcoded scripted policies for both square and triangle peg insertion.
- Generate and save at least 50 high-quality demonstrations for each shape.
- Store data in the correct `.hdf5` format under `data/dataset-ur10_mb/`.
- Provide a brief write-up (1 page max) summarizing your approach.

Grading Weight: 30 Points – Based on the quality, completeness, and organization of your demonstration data.
Late Submission Penalty: 10 Points — Submissions received after the deadline will incur a deduction of 10 points.

Milestone 2: Final Submission

Due Date: April 29, 2025 (11:59 PM)

This includes the complete implementation:

- Training the ACT model on your dataset.
- Evaluating the learned policy and comparing it with the scripted baseline.
- Full report submission with figures and analysis.
- Code packaged and submitted according to the directory structure.

Grading Weight: 70 Points – Based on correctness, depth of analysis, clarity of report, and code quality.

6 Computational Resources

You may use the Turing cluster for training and evaluation. Please include the following SBATCH flags in your job script:

SBATCH Directives

```
#SBATCH -A rbe577
#SBATCH -p academic
```

7 Additional Tips

Here are some practical suggestions to help you succeed in this assignment:

- **Training Strategy:** After your ACT model converges, continue training for an additional 2-3 times longer than it took to stabilize, this helps your trajectories look smoother and improve generalization. You can configure the number of epochs in `config/imitation_train.yaml`.
- **Helper Utilities:** The `helpers/` folder contains pre-written utility functions for:
  - Loading and processing demonstration data
  - Writing demonstrations to disk
  - Logging and metric visualization

  Specifically, refer to `utils.py` for core I/O and logging operations.
- **Environment and Cameras:** The `data/` directory includes:
  - Example video files showing how environments and camera views look
  - Images and different graphs for reference

  You are allowed to modify camera positions, but your implementation must include **three camera streams** to match the input expectations of the model.
- **Offline Data Collection:** You may collect demonstration data locally on your machine and then upload the resulting HDF5 files to the Turing cluster for training. This will save cluster time and accelerate iteration.
- **Logging and Visualization:** Learn to use the logging tools provided in `helpers/logger.py`. These support real-time metric tracking via TensorBoard, which will help you monitor training progress, spot overfitting, and fine-tune hyperparameters.
- **Interactive Robot Control (Exploration Only):** You may use `robot_keyboard_control.py` to manually control the UR10 robot in the simulation. This script is useful for understanding joint movements and testing actions in real time.

  Important: *Do not use this script to collect demonstrations.* It is intended for exploration only. You should instead write scripted policies to automate the peg insertion demonstrations, which will ensure consistency and reproducibility. You can use different functions that are used here to move the robot, to create a script that does this automatically.

8 Additional Resources

For further reading and a deeper understanding of the underlying methodologies, consider the following resources:

- **Action Chunking Transformer (ACT):** Zhao, T., et al. ”Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware.” *arXiv preprint arXiv:2304.13705*, 2023. Available at: https://arxiv.org/abs/2304.13705
- **DEtection TRansformer (DETR):** Carion, N., et al. ”End-to-End Object Detection with Transformers.” *arXiv preprint arXiv:2005.12872*, 2020. Available at: https://arxiv.org/abs/2005.12872
- **Transformer Architecture:** Vaswani, A., et al. ”Attention Is All You Need.” *arXiv preprint arXiv:1706.03762*, 2017. Available at: https://arxiv.org/abs/1706.03762