ECSE 683 Assignment: Imitation learning in PyBullet and RoboSuite

Shiyao Ni Faculty of Enginneering

Department of Mechanical Engineering

McGill University

shiyao.ni@mail.mcgill.ca

Abstract

This assignment explores the application of imitation learning to two distinct robotic tasks: locomotion and manipulation. The locomotion task involves training a quadruped robot in PyBullet to mimic real dog walking patterns using motion capture data, while the manipulation task involves a robotic arm performing pick-and-place tasks in Robosuite using human teleoperated demonstrations. The locomotion task employs reinforcement learning-based imitation learning due to the challenges of simulating dynamic behaviors and the morphological discrepancy between real and simulated robots. Despite extensive efforts in domain adaptation, learned locomotion policies remain sensitive to perturbations and fail to guarantee long-term stability. In contrast, the manipulation task successfully applies behavior cloning with an LSTM network trained on human demonstrations, achieving a 94% success rate. However, it suffers from a lack of failure recovery, requiring extensive data collection from expert operators. A comparative analysis highlights the challenges of applying imitation learning to both tasks, emphasizing that locomotion is more constrained by simulator inaccuracies, whereas manipulation demands large datasets for generalization. The study suggests that integrating reinforcement learning with imitation learning for manipulation tasks could improve robustness by enabling policies to recover from failures. Overall, imitation learning is a powerful tool for robotic control but faces limitations in failure recovery and data efficiency.

For this assignment, I have used 2 examples on 2 different simulators to test different tasks: a quadruped robot (unitree Laikago) in pyBullet for locomotion tasks and a grasper (Franka Panda) in Robosuite for simple manipulation tasks (pick up a cube).

1 Imitation learning on locomotion tasks

1.1 Description of the tasks

From the start of this assignment, I selected the quadruped robot unitree A1 to mimic a real dog walking dataset. I extracted the real dog walking dataset from[1], and unitree A1 structure from [2]. The original work have used a 2-stage process. In the first stage, the referenced animal motion is retargeted onto the robot by mapping selected keypoints onto the robot's body via inverse kinematics. The keypoints in this work contains the feet and hips locations of the real dog, and inverse kinematics is used to compute the pose of the robot that tracks the keypoints. In this stage the "task space" is defined by these keypoints in 3D space, 4 feet and 4 hips positions, added to$8x3 = 24$ dimensional task space. In the second stage the policy is trained to track a sequence of target poses expressed in the robot's own configuration space. Here the configuration space is the set of all possible robot poses, which will be the robot's own configuration space. The laikago robot used in the tasks contains 18 DoF, including 12 actuated joint and 6(x,y,z coordinates and pitch raw roll) for the body. The task space is also the 18 DoF of the robot.

The state is composed of a short history of recent robot poses and actions. Specifically, it includes the poses q at three consecutive timesteps (t–2, t–1, and t). And each pose contains the IMU readings for the robot's root (roll, pitch, yaw) as well as the local rotations of all joints, which is the all 18 DoF sensor data of the last 3 consecutive timesetps. It also has the information regarding the actions taken at the *previous* three timesteps (from t–3 to t–1).

The action, on the other hand, is a vector of target joint rotations for each actuated joint. These targets are fed to PD controllers that compute the actual torque commands, ensuring that the robot's joints follow the desired trajectories.

1.2 Method of the work

The method of the original work includes following steps:

- Motion Retargeting: Reference motions from animal data are first retargeted onto the robot's morphology using inverse kinematics. In this step, keypoints (feet and hips) from the animal motion are mapped to corresponding points on the robot, ensuring that the target motion respects the robot's kinematic structure and physical constraints.
- Imitation Learning via Reinforcement Learning: The problem is formulated as an imitation learning task. The robot's controller is trained with reinforcement learning to track a sequence of target poses derived from the retargeted motion. The state includes a short history of recent poses and actions, and the action is a vector of target joint angles that are fed to PD controllers. The reward function is designed to penalize deviations in joint angles, velocities, end-effector positions, and base orientation, thereby guiding the policy to generate motions that are both faithful to the reference and physically feasible.
- Domain Adaptation: To account for discrepancies between simulation and real-world dynamics, domain randomization and latent space adaptation techniques are employed. These ensure that the learned policies are robust and can be fine-tuned with relatively few real-world trials, further enforcing that the motions remain within the robot's physical limits.

And in order to ensure that the motion stays within physical limits, the work incorporates:

- Kinematic Constraints: The inverse kinematics retargeting maps the reference motion into a robot-specific configuration that inherently respects the robot's joint limits and morphology.
- Reward Shaping: The reward function includes terms that directly penalize unrealistic joint positions, velocities, and end-effector deviations, ensuring adherence to physically plausible behaviors.
- Physics-Based Simulation: Training is performed in a simulation environment using PyBullet that enforces rigid-body dynamics, contact constraints, and actuator limits.
- PD Controllers: The actions are executed through PD controllers, which help in smoothing the trajectories and maintaining them within feasible limits.

- Joint torque constraints: The work did **NOT** incorporate fixed numerical torque limits for each joint. Instead, they model the actuator dynamics via parameters such as motor strength and friction (as shown in Table I), and use PD controllers that compute torques based on the difference between target and actual joint angles. In simulation (using PyBullet), these dynamic properties inherently impose limits on the torques that can be applied. This means that while there may not be an explicit "torque cap" specified, the motor model and controller design ensure that the generated torques remain within physically plausible ranges.

1.3 Discussion and limitations

Among testing the work, there exist several problems that prevented me from doing a full analysis of imitation learning within the scope of this assignment: 1. when I was re-training the full imitation learning algorithm from the paper[1], the training has run for 2 consecutive days without stopping and still no way near finishing. Therefore, I have planned to program a simpler imitation learning algorithm that use plain imitation learning behavior cloning algorithm. However, there problems while I am trying to extract the dataset. The dataset from the repo did not contain enough information regarding the data structure of the motion captured information. I tried to feed in the mocap data after retargeting onto the laikago robot and a1 robot(using only the joint angles) using a behavior cloning model but they all failed. Robot lies on the ground with bellies upward. I think that these problems are a result from feeding the wrong information into wrong limb control sequence. (e.g. mapping body xyz coordinates onto joint angles). Also, due to the high morphology discrepancy from the source of data and unitree robots in the simulator, I doubt if a simple behavior cloning model will work. I suspect the robot policy trained will be very fragile towards any disturbances (e.g. simulator numerical instability)

However, although the training took too long to reproduce the results[1], I am able to visualize the pre-trained model in the paper. The result seems very robust as well as mimicking the behavior from mocap well. However, if we visualize all the results 1, there still exist a pattern of error despite very heavy effort from the original work have been put on domain-adaptation and generalization.

Figure 1: Pre-trained visualization. First column: dog pace. Second column: dog trot. Third column: dog spin. First row: result from first few frames. Second row: result from last few frames

From Fig. 1, we observe that in the initial frames of the visualization, the error between the original dataset and the pre-trained model is relatively small. However, as the sequence progresses, the error accumulates, except in the case of the dog pace run. This suggests a gradual deviation from the reference trajectory over time. Given this trend, I suspect that the robot may eventually fall if the simulation continues running with the same policy. Upon analyzing the pre-trained model in action, I conclude that the dog pace motion is the most “quasi-static”, making it easier for the simulator to replicate. As a result, it experiences the least error post-training compared to other motions.

The learned controllers in the work are also not as stable as manually designed controllers, especially when executing more dynamic behaviors.

The work did not work 100% of the time since the robot often falls when executing highly dynamic motions such as running or large jumps, indicating that the learned policies still lack robustness. Moreover, due to the limitation of the simulator, the framework struggles with motions that require precise contact dynamics, such as motions involving sudden direction changes or aerial phases.

1.4 Addition notes

This work did not utilize standard imitation learning method but with reinforcement learning methods based of imitation learning. I did not successfully implement imitation learning (BC) models on the dataset as well. Locomotion tasks compare to manipulation tasks generally have less states to consider, and a less complicated environment to have feedback upon. The dataset it contains is also very simple, only one scene with one dog running on one flat terrain(for dog_pace scene) for each category. More data or generalization of the controller are produced by the use of reinforcement learning(which makes computation speed very slow). The framework struggles with motions that require precise contact dynamics, such as motions involving sudden direction changes. Locomotion tasks are more fragile towards dynamical interactions due to the nature of simulator handles friction. Therefore, since the topic of the assignment is to investigate "imitation learning", I have decided to go further and reproduce results from the robomimic paper [3], to get some more insights on the basic behavior cloning imitation learning algorithm.

2 Imitation learning on manipulation tasks

Ajay Mandlekar et al. [3] have created human-proficient data for pick tasks using the Franka Panda robot arm. In my work, I am only focusing on the human-proficient dataset for pick tasks, using the pre-converted low resolution hdf5 dataset.

2.1 Description of the tasks

The pick task studied involves robotic arms performing manipulation tasks such as lifting a cube, picking a can, or grasping and positioning objects with precision. These tasks are evaluated both in simulated and real-world environments, using various datasets that include human and machine-generated demonstrations. In my study, only the senario of lifting a cube is considered.

The configuration space describes all possible positions and orientations of the robotic arm. In this study, the Franka Emika Panda arm, has a 7-degree-of-freedom (DoF) configuration space, consisting of:

- Translation of the end effector: Movement along X, Y, Z axes. (3 DoF)
- Rotation of the end effector: Orientation in space using axis-angle representation. (3 DoF)
- Gripper control: Opening and closing of the robotic gripper. (1 DoF)

Unlike locomotion tasks, the manipulation tasks uses a different task space. The task space refers to the workspace where the manipulation task occurs, which includes:

- Object positions and orientations.
- Target locations where objects must be placed.
- Constraints on movement. The object's initial position and necessary precision for grasping and placing.

The state space of this work consists of:

- End-effector position (spatial 3D coordinates).
- End-effector quaternion.
- Gripper finger positions .
- Object states, including absolute position and quaternion (7D for some tasks).
- For some tasks, camera observations may be used too. But for pick tasks, camera input is not included.

The action space of the work includes:

- Translation: Desired movement in Cartesian space.
- Rotation: Change in orientation using axis-angle encoding.
- Gripper control: Opening/closing commands for object grasping.
- Actions are transformed into end-effector targets, and controlled through an operational space controller.

2.2 Method of the work

For the pick-a-cube task, the method involves imitation learning combined with motion planning to ensure smooth and efficient execution. The approach can be broken down into the following steps:

- Data Collection: They used human teleoperated demonstrations to collect high-quality grasping trajectories.
- Behavioral Cloning: A LSTM Network is trained to predict actions based on past state observations. The network learns from demonstrations, enabling the robot to mimic human-like grasping behavior.
- Motion Planning: Inverse kinematics is used to generate feasible joint angles for each step.

To ensure safe and feasible execution within the robot's physical limits, following constraints are enforced:

- Joint Limit Constraints: Each joint's motion is clamped within allowable angles to prevent self-collision (e.g. For a 7-DOF arm, each joint has a min/max rotation limit)/
- Velocity and Acceleration Limits: Motion is planned using velocity profiles to avoid excessive speeds.
- Gripper Force Control: A force threshold prevents excessive gripping pressure.
- Collision Avoidance: The robot's environment is mapped, and the motion planner ensures no collisions occur.
- End-Effector Position Constraints: The Z-axis constraint ensures the cube is only picked within a specific height range.

2.3 Discussion and limitations

The imitation learning algorithm demonstrates high computational efficiency, completing training for the PH-lift task in approximately one hour, significantly outperforming reinforcement learning in terms of speed. However, its limitations are evident, particularly in failure recovery. The model is trained only on successful demonstrations, meaning it has no exposure to failure scenarios. As a result, when an error occurs—such as a missed grasp—the robot arm fails to adapt, repeatedly executing incorrect actions instead of recovering. Furthermore, the algorithm requires a large dataset to generalize well, even for simple grasping tasks. These data must be gathered from proficient human operators, either in simulation or real-world environments, making the approach data-intensive and potentially impractical for complex or highly variable tasks. The trained model achieved a 94% successrate on both runs with different seeds.

3 Comparison of imitation learning between two tasks

In the end, I wish to comment based on imitation learning applied to both locomotion and manipulation. Locomotion tasks are normally more constrained by simulators, as many simulators struggles on modeling dynamic behaviors [4]. Therefore the authors have decided to use a reinforcement learning based imitation learning method. Pure imitation learning will suffer from instability from simulation as well as difference in morphology between real world dataset source and simulator robot model. After completing this assignment, I feel important to re-construct my project. Having done trainings in both environment, it is clear that manipulation tasks are easier to apply imitation learning algorithms upon, but it still suffers from data collection from expert human operators. Due to the complex environment graspers need to interact with, it is impossible to collect little data as if the locomotion tasks requires for the algorithm to be operational. However, it could be a neat idea to implement a similar imitation-reinforcement learning structure towards manipulation tasks such that the control policy could learn to recover from a failed state.

In conclusion, imitation learning is a powerful and efficient tool for modern robot learning. However, I personal consider its biggest problem underlies failure recover process apart from data collection.

References

[1] X. Bin Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *Robotics: Science and Systems (RSS), Virtual Event/Corvalis, July*, pp. 12–16, 2020.

[2] unitreerobotics, "unitree_pybullet," 2020.

[3] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín, "What matters in learning from offline human demonstrations for robot manipulation," in *5th Annual Conference on Robot Learning*, 2021.

[4] D. Blanco Mulero, O. Barbany, G. Alcan, A. Colome, C. Torras, and V. Kyrki, "Benchmarking the sim-to-real gap in cloth manipulation," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–8, 03 2024.