Imitation of Pushing from Video Demonstrations

Dingkun Guo*, Yulun Zhang*

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA

***Abstract*—Imitating actions from videos is a necessary ability for robots, as it allows robots to perform more complex skills without manually programming of manipulation tasks. However, this technique remains challenging due to the difficulty in inferring physics from the scene and object relationships. In this project, we let a robot in simulation to imitate the box pushing behavior in videos without knowing the friction coefficient or weight of the box. We propose a naive random search method to solve the box pushing as a black-box optimization problem. Our results shows the robot pushes the box following the box trajectory in a video with an average position error of 1.07 cm. This work can be used as a baseline for future data-driven methods of learning pushing from video demonstrations.**

***Index Terms*—robot pushing, imitation from video demonstration, simulation**

I. INTRODUCTION

Video demonstrations can provide skill guidance for robots to finish tasks. From videos, robots can understand and acquire many complex skills. Many works have explored the possibility of learning from demonstrations, and in this project, we simply want a robot in simulation to imitate the box pushing behavior in videos. One of the difficulties is that a video is the only input from the program. Without knowing the real friction coefficients or box weight, we cannot directly calculate the force needed to push the box. Also, we consider the sim-to-real error in this project, which means the box could end up with different locations even if being pushing with the same force.

To solve the problem, we first exacted poses of the box from a video using ApirlTag detector and we defined the box pushing as an optimization problem to find the optimized robot actions. With random search method, we achieved the robot pushing the box following desired trajectory with an average position error of 1.07 cm, which suppressed the error of 6.19 cm using our baseline method of pushing the box with constant force. While there are many limitations, this work produces a simple pushing primitive that allows robot to push box in any similar scenarios without measuring frictions and this work can be used as a baseline for future data-driven methods of learning pushing from video demonstrations.

This is the final project report of Course 16-741: Mechanics of Manipulation at Carnegie Mellon University, Pittsburgh, PA, USA in Fall 2022. The code and videos for this project can be found at https://github.com/dkguo/Pushing-Imitationhttps://github.com/dkguo/Pushing-Imitation

*All authors contributed equally to this project and are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA {dingkung, yulunz}@andrew.cmu.edu

(a) (b)

Fig. 1: (a) A screenshot of a box pushing video and (b) simulation environment. To let a robot in the simulation recover pushing behavior in the video, we extract a trajectory of the box and randomly search desired robot actions to push the box while optimizing position and orientation error.

II. PRELIMINARIES

In this section, we introduce some background knowledge of sim-to-real error that helps understandings of our methods.

As illustrated by red arrows in Figure 2, we can directly search a desired robot action u_i^* to push the box from the ground truth pose P_i to P_{i+1}. However, in reality, if the robot pushes the box at P_1^* with u_1^*, the box will end at P_2 instead of P_2^* because there always exists an error between the simulation and the real-world. In this case, if the robot keeps pushing with u_i^*, the box will get much further than ground truth poses as the sim-to-real error accumulates.

We demonstrated this sim-to-real error with two simulated environment with different noise. We used one environment for simulation to search desired robot actions, and the other for execution. The two environment will generate different box poses, even if the same robot actions and initial conditions are applied.

III. METHODS

To recover the pushing behavior in a recorded video, we first exacted positions and orientations of the box being pushed and put the box trajectory into simulation as the ground truth trajectory with which the robot want to push the box to follow. Secondly, we defined the pushing as an optimization problem to find desired robot actions.

A. Pose Estimation

To find poses of the box being pushed in the video, we stuck a AprilTag on the box and detected its pose using an existing

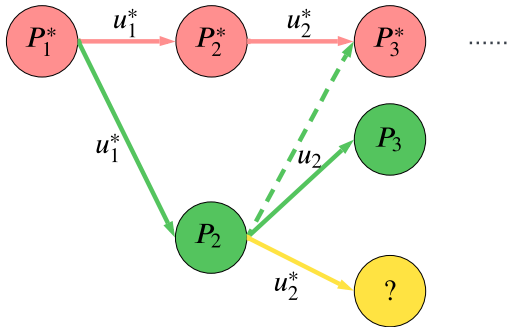


Fig. 2: State transition diagram. u_i^* is the desired robot action found by direct search to push the box from pose P_i^* to P_{i+1}^*. If the robot keeps pushing with u_i^*, the box will get much further than ground truth poses as the sim-to-real error accumulates. Therefore, we search desired robot action u_i step-by-step during execution.

detector [2]. The detector provided the pose of AprilTag in frame i, $T_c^{t_i}$, (here T is a 4x4 homogeneous transformation matrix) relative to the view of camera. Then we used Equation 1 to calculate the pose of box in each frame.

$$P_i^* = T_O^{t_0} \cdot (T_c^{t_0})^{-1} \cdot T_c^{t_i} \tag{1}$$

P_i^* is the pose of the box in frame i; $T_O^{t_0}$ is the pose of AprilTag in the first frame relative to world coordinate, defined by where we wanted to put the box. The detected poses are accurate as the detector provided small reprojection error, so we used these poses as ground truth trajectory in next steps.

B. Step-by-step Black-box Optimization

We find the desired robot action $\hat{u}_i$ at each timestep i by solving a black box optimization problem defined in euqation 2.

$$\hat{u}_i = \underset{f,\mathbf{n}}{\arg\min}\ l(g(IK(f,\mathbf{n})), P_i^*) \tag{2}$$

The variables f and $\mathbf{n}$ refers the force magnitude and contact normal of the force. The function *IK* is a inverse kinematics function that returns the low level robot control operations given a force magnitude and a contact normal. The function g is the simulator that returns the simulated object pose P_i after taking the given robot action. The objective function l then calculates the difference between object pose P_i and the corresponding ground-truth object pose P_i^*. The function l calculates the object pose difference. In particular, given simulated object pose $P_i = \begin{bmatrix} p_i & q_i \end{bmatrix}^T$ and ground-truth object pose $P_i^* = \begin{bmatrix} p_i^* & q_i^* \end{bmatrix}^T$, where p_i, p_i^* are the 3D position, and q_i, q_i^* are the normalized quaternions, the function l is defined in equation 3.

$$l(P_i, P_i*) = ||p_i - p_i^*||_2 + w_l * (1 - q_i \cdot q_i^*) \tag{3}$$

Essentially, the function l calculates a weighted sum between 1) the L2 norm of the difference in position of the given object poses and 2) the negative cosine similarity of the quaternions. The parameter w_l is a hyper-parameter that controls the weight between these two parts.

C. Random Search with Adapted Search Space

To solve the black-box optimization problem defined in equation 2 and 3, we apply a modified random search that automatically adapts the search space based on the object pose difference from the previous timestep. Defined in equation 4, the force magnitude f is sampled from an uniform distribution with lower 100 and upper bound 200. The contact normal is sampled from an unit cube around an automatically adapted center of search space.

$$f \sim uniform(100, 200) \tag{4}$$

To adapt the search space of the contact normal, we first find the simulated object pose $P_{i-1} = \begin{bmatrix} p_{i-1} & q_{i-1} \end{bmatrix}^T$ from the previous timestep. We then set the center of the search space to be a unit cube around the position $p_{i-1} - O + I * w_p * e_{i-1}$, where O is a fixed offset from the object pose to the default contact position, e_{i-1} is the object pose error, and w_p is a weight variable. The variable I is an indicator variable that encodes if the simulated object pose surpasses or falls behind the ground-truth. In particular, I is defined in equation 5.

$$I = \begin{cases} -1 & \text{if } p_{i-1\ x} \geq p_{i-1\ x}^* \\ 1 & \text{if } p_{i-1\ x} < p_{i-1\ x}^* \end{cases} \tag{5}$$

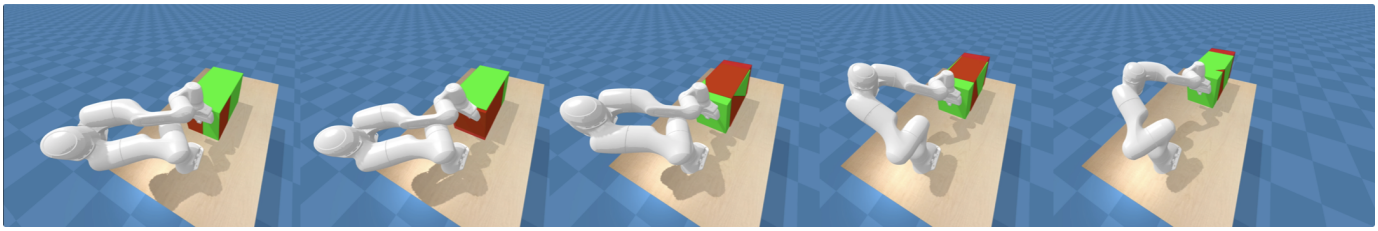
Intuitively, when $p_{i-1\ x} \geq p_{i-1\ x}^*$, meaning that the simulated object pose surpasses the ground-truth, we want the contact normal to move away from the object to slow down the pushing. Otherwise, when the simulated object pose falls behind, we want the contact normal to move closer to the object so that the robot push harder on the object to catch up in the next time step. The the object pose error controls how far the contact normal forward or backward from the default contact normal at $p_{i-1} - O$. If the error is large, the current reconstructed trajectory either surpasses or falls behind the ground-truth too much, then the next contact normal needs to be either move further away or closer to the object to slow down or speed up pushing. Therefore, sampling strategy of contact normal $\mathbf{n}$ is defined equation 6

$$\mathbf{n} \sim uniform(0, 1) + p_{i-1} - O + I * w_p * e_{i-1} \tag{6}$$

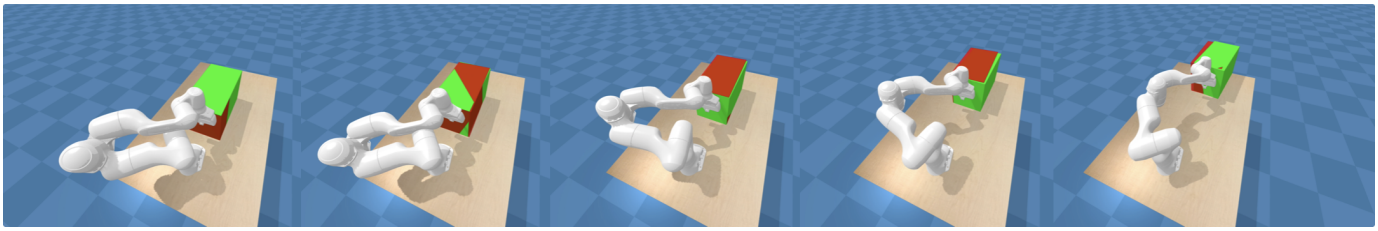
After getting the center of the search space, we can then find the best contact normal in the unit cube around the given center by sampling N random vectors and choosing the one that minimize the objective defined in equation 3.

IV. RESULTS

Figure 3 shows our reconstructed trajectories compared with a baseline trajectory from directly pushing the object with fixed force magnitude and contact normal. It is clear that our reconstructured trajectory follows the ground-truth trajectory closer than directly pushing with fixed force.



(a)



(b)

Fig. 3: **Resulting trajectories:** Our reconstructed trajectories (below) vs. baseline trajectory from directly pushing with fixed force. The green cube is pushed by the robot and red cube is the ground-truth trajectory.

TABLE I: Numerical difference between ours and baseline trajectory with the ground-truth trajectory

Trajectory	Average Position Difference	Average Quaternion Difference
Ours	1.07 cm	0.0001
Baseline	6.19 cm	0.0002

Table I shows the numerical difference between the reconstructed and baseline trajectory and the ground-truth. The position difference is calculated as the sum of euclidean distance between the corresponding positions of the trajectories and the quaternion difference is calculated as sum of 1 minus the cosine simularity of the corresponding quaternions. The quaternion difference encodes the translational difference between the trajectories. We observe that while our reconstructed trajectory has much lower position difference compared with baseline, the translational difference is almost identical. This is because we only test our algorithm with a very naive trajectory that is almost straight. We discuss this further in section V.

V. LIMITATION AND CONCLUSION

Our work is limited in many ways which induces many future directions. First, as mentioned in section IV, we only tested our algorithm with a very naive trajectory with a few assumptions: 1) we assume that the trajectory is almost a straight line on a 2D plane that involves almost no translations. Under this assumption, we can use adapt the center of contact normal search space based on if the object has ”surpassed” or ”falls behind” from the ground-truth trajectory. 2) We assume that the object being pushed is simple cube so that sampling from a unit cube around the adapted center of contact normal search space can give us a reasonably good contact normal. Therefore, in the future we can explore more advanced method that can reconstruct trajectories that 1) involves complicated translations which makes the trajectory ”curved”, and 2) involves fancier real-world objects than a simple cube.

In addition, we use a naive random search to solve the black-box optimization problem defined in 2, which is known to be sample inefficient with arbitrarily large search space. As a future work, we could try more advanced black-box optimizers such as evolutionary strategies.

ACKNOWLEDGMENT

This project could not be finished without the mengine simulator [1] developed by Prof. Zackory Erickson. Also thank him for the fascinating course materials and support that he provided this semester.

REFERENCES

[1] Z. Erickson, “Manipulation engine,” https://github.com/Zackory/mengine, 2022.

[2] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.