Optimal Demonstrations for Behavior Cloning

Ria Sonecha
MIT EECS
Cambridge, MA
rsonecha@mit.edu

Shao Yuan Chew Chia
Harvard CS
Cambridge, MA
shaoyuan_chewchia@college.harvard.edu

Nicholas Pfaff
MIT EECS
Cambridge, MA
nepfaff@mit.edu

***Abstract*—Behavior cloning for training visuomotor policies has become a popular framework in robotics. Yet, behavior cloning heavily depends on the quality of human demonstrations that tend to be both sup-optimal and time-consuming to collect. We propose to use Graphs of Convex Sets (GCS) to automatically create optimal demonstrations and use these demonstrations to train a Diffusion Policy. We show that the action trajectories executed by the Diffusion Policy are close to the optimal ones that GCS would have run for the same initial conditions while not requiring any human demonstrations. In doing so, we reveal this novel paradigm's potential to overcome the many downsides of human-generated demonstrations.**

I. Introduction

Our project is about using Graphs of Convex Sets (GCS) [6] to automatically generate demonstrations for Diffusion Policy [1]. In particular, we use GCS to create motion plans for a point finger for pushing an object into a target state. We record images of open loop execution of these motion plans and use them for training a Diffusion Policy. This allow us to learn a policy based on images rather than full-state feedback.

Diffusion Policy is a novel approach that works well and robustly in a behavior cloning setting. However, it is currently dependent on human demonstrations that are time-consuming to collect and potentially sub-optimal. With our project, we want to remove the need for human demonstrations by using GCS to generate good demonstrations automatically. Using full-state information, GCS has the potential to generate better demonstrations than a human can. Diffusion Policy allows us to use these demonstrations and convert them into an image-based policy without access to the full state. Such an image-based policy could then be executed on a real robot.

II. Related Work

Denoising Diffusion Probabilistic Models (DDPMs), also called diffusion models, are a class of generative models [4]. The diffusion process starts with random noise, and the diffusion model learns to gradually reduce the noise and generate a high quality output. DDPMs have been applied to a large variety of areas such as image synthesis, super-resolution, and more recently, in robotics for representing visuomotor control policies.

The recent work [1], introduces the idea of Diffusion Policies which generate action sequences via a "conditional denoising diffusion process on robot action space." In their experiments doing behavior cloning from human demonstrations, they find that diffusion policies are able to express multimodal action distributions, handle a high-dimensional output space, and are stable to train. These advantages help the authors achieve state-of-the-art results on a large variety of robotic manipulation benchmarks. In general, however, the quality of policies trained via behavior cloning is heavily dependent on the expert demonstrations, and human generated demonstrations are not guaranteed to be optimal. Additionally, collecting human generated demonstrations is time consuming.

In [5], the authors propose using graphs of convex sets [6] to perform optimal motion planning around obstacles. This work provides an alternative to the standard sampling-based planners which are commonly used for motion planning in complex configuration spaces and cluttered environments. By representing free space as a graph of convex regions, the authors are able to use the framework proposed in [6] for finding shortest paths in graphs of convex sets by solving a compact mixed-integer optimization. In this project we build off of an extension of these works which uses the idea of graphs of convex sets for optimal planning through contact [2] to automatically generate optimal demonstrations for training diffusion policies.

III. Method

A. Overview

In this project we focus on the task of a point finger pushing an axis-aligned box on the ground from an initial position to a goal location. Successfully achieving this goal requires the point finger to plan and execute a trajectory which makes and breaks contact. We start by formulating this task as a GCS problem and generate optimal trajectories. These trajectories are then used as expert demonstrations for training a diffusion policy via behavior cloning.

B. Problem Formulation

We will apply these techniques to a planar pushing task with an actuated spherical robot finger and an unactuated box. We constrain the box to not rotate in order to maintain convexity of the GCS problem. The state of our system is defined as

$$x = \begin{bmatrix} b_x & b_y & s_x & s_y \end{bmatrix}^\top \tag{1}$$

Control inputs u are absolute position commands for the finger

$$u = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{2}$$

We will assume that our system is quasistatic, meaning the velocities and accelerations of the system are 0 at the end of each time step. This corresponds to having a high amount of damping and is a reasonable assumption in the 2D planar pushing setup where we restrict pushing velocities to be low and there is a large amount of friction between the object and table surface. As a result, the state of our system only consists of positions.

C. System Architecture

Our simulator is composed of three modular parts, the environment, the high level controller and the low level controller. The environment consists of a *MultibodyPlant* with a top down camera, unactuated box and sphere finger robot. The GCS and Diffusion Policy controllers are instances of the high level controller. They take in finger state and RGB image data from the environment and calculate the desired position of the finger. This desired finger position is passed to the low level controller. The low level controller uses the *StateInterpolaterWithDiscreteDerivatives* leaf system to transform the position commands into desired positions and velocities and then feeds the desired state into an *InverseDynamicsController* with PID gains of 100, 1 and 10 respectively for error based tracking of the desired position.

D. GCS for Planning Through Contact

We start by defining two "contact pairs" – the finger and box, and the box and ground. Each contact pair has a set of allowable "position modes" and "contact modes," and in order to achieve the goal of moving the box from an initial position the contact pairs must be able to transition between the various position modes and contact modes.

For the box and the ground the only allowed position mode is the box being on top of the ground (i.e. the bottom of the box cannot sink below the ground). For the finger-box contact pair we manually decompose the free space around the box into 12 different position modes as shown in Figure 1. The position modes are defined according to the relative position of the finger with respect to the box. In the diagram the green finger is shown in the "bottom left" position mode. The narrow transition regions on each side of the box enforce that a) the finger only contacts the box in the center of each side so there is no rotation and b) there is no contact between the finger and the box as the finger moves around to different sides of the box.

Within each position mode, there are six possible contact modes which depend on the magnitude of the force between the bodies that make up the contact pair: "no contact", "sticking" (the bodies are in contact but there is no relative motion), "sliding left", "sliding right", "sliding up", and "sliding down".

Position modes are defined using constraints on the relative position of the finger with respect to the box $({}^Bx^F, {}^By^F)$. For

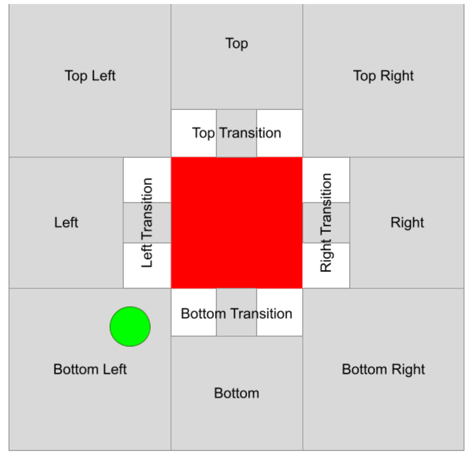

Fig. 1. Possible position modes for the finger-box contact pair.

example, for the "left" position mode, we require the following constraints to be true:

$$\begin{aligned} {}^Bx^F &<= -\frac{l_b}{2} - b_t \\ {}^By^F &<= \frac{l_b}{2} \\ {}^By^F &>= -\frac{l_b}{2}, \end{aligned} \tag{3}$$

where l_b is the width of the box and b_t is the transition buffer that we want to stay away from the box. The constraints are computed similarly for the 11 remaining position modes.

Contact modes are defined with constraints on the forces acting at the contact point c between bodies. We call the horizontal friction force acting at point c, $(f^c)_x$ and the vertical friction force $(f^c)_y$. The normal force acting at point c is called n^c. We call the relative velocity between the objects v_{rel}, and the friction coefficient μ. To define the "no contact" mode we have the following constraints:

$$\begin{aligned} &n^c = (f^c)_x = (f^c)_y = 0, \\ &v_{rel} = 0. \end{aligned} \tag{4}$$

In the case of "sticking" contact the constraints are:

$$\begin{aligned} &n^c \geq 0, \\ &|(f^c)_x| \leq \mu n^c, \\ &|(f^c)_y| \leq \mu n^c, \\ &v_{rel} = 0. \end{aligned} \tag{5}$$

And, for "sliding right" we have the following constraints:

$$\begin{aligned} &n^c \geq 0, \\ &|(f^c)_x| \geq \mu n^c, \\ &|(f^c)_y| \leq \mu n^c, \\ &(v_{rel})_x \geq 0, \\ &(v_{rel})_y = 0. \end{aligned} \tag{6}$$

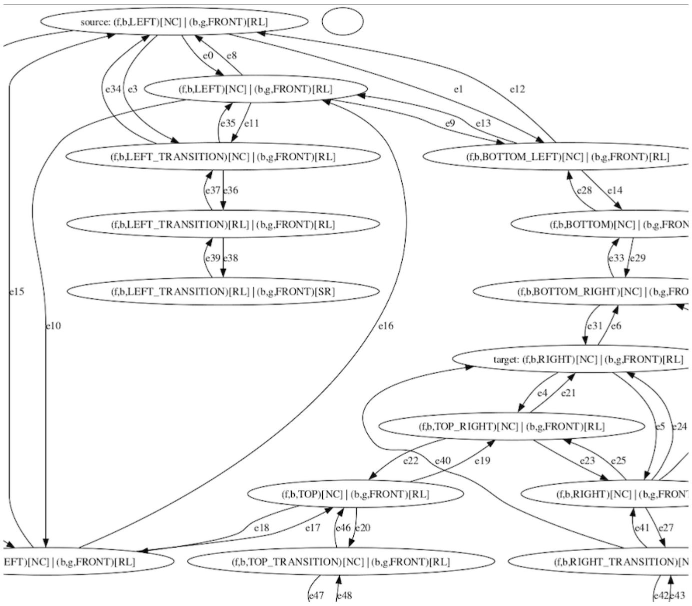

Fig. 2. GCS graph for planning through contact for box pushing.

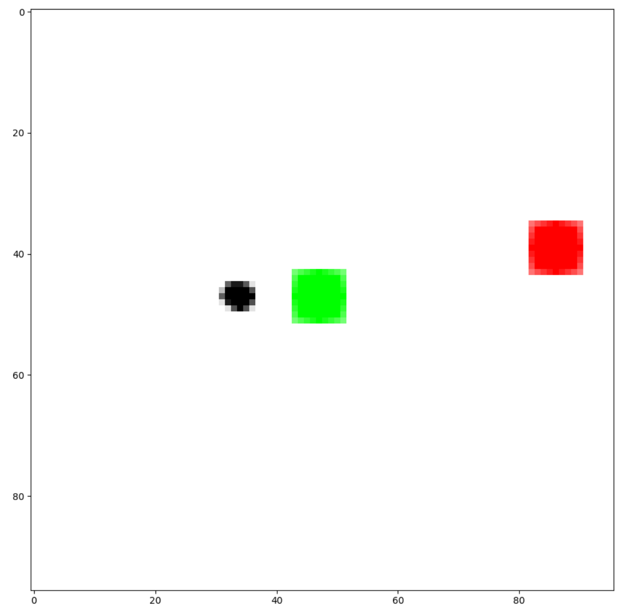

Fig. 3. One of the images that we generate while executing our GCS plans. We use these images to train the diffusion policy. The box that needs to be pushed is shown in red, the actuated point finger in black, and the box's goal location in green. The numbers indicate the image size and are not part of the image.

The remaining three ”sliding” modes are defined similarly, with different friction and relative velocity constraints depending on the direction of motion.

The vertices for the GCS graph are a combination of position mode and contact mode for each contact pair in the environment. For example, the combination of the ”left” position mode and the ”no contact” contact mode for the finger-box contact pair represents one vertex in the graph. The convex set corresponding to that vertex is the intersection of the position mode and contact mode constraints. In our setup, all of these sets are naturally convex. Edges exist between vertices with adjacent position modes and contact modes. For example, any vertex which contains the finger and box in the left position mode is connected to all vertices where the finger and box are in the left transition, top left, or bottom left transition mode. Additionally, when constructing the graph only physically feasible vertices and edges are added. For example, while we technically allow the finger-box contact pair to be “sticking” when they are in the “left” position mode, this is not physically possible given the geometry of the position modes. Thus, the corresponding convex set will be empty and no vertices with this combination of position and contact mode will be added. We also manually prune some modes to limit the size of the GCS graph. For example, we require that the box is always in contact with the ground so we do not allow the box-ground pair to ever be in the no contact mode. A small section of the full GCS graph is shown in Figure 2.

Once the GCS graph is constructed we can use the same method as [5] to solve the mixed integer optimization problem for finding a shortest path from the source node to the target. The output is a finger trajectory parameterized by Bezier curves which moves the box from its initial position to the goal position.

E. Data Generation

Both the training and validation of the policy requires generating a set of random initial conditions of the system. To do this we define the bounds of the workspace and sample the initial positions of the finger and box from a uniform distribution within those bounds. We then use Drake's collision checker to determine whether the finger and box are in collision by evaluating the contact results output port of the MultibodyPlant and reject the sample if they are in collision.

To generate GCS demonstrations for the diffusion policy, we generate 100 valid initial conditions, use GCS to plan the optimal open-loop position trajectory and then execute the trajectory in Drake. Every 1000 simulation time steps we log the RGB image output of the camera positioned above the workspace, the open-loop desired position command, and the actual position of the finger. An example image is shown in Fig. 3.

F. Behavior Cloning with Diffusion Policies

Like [1], we use Denoising Diffusion Probabilistic Models (DDPMs) to represent a visuomotor policy for pushing the box. In their most general form DDPMs starts with a sample of random noise x^k and performs K denoising steps, resulting in a sequence of samples $x^{k-1}, \ldots, x^0$, where x^0 is noise-free. The process of denoising is done by a trained de-noising network $\epsilon_\theta(x^k, k)$, which predicts noise given a sample and iteration step.

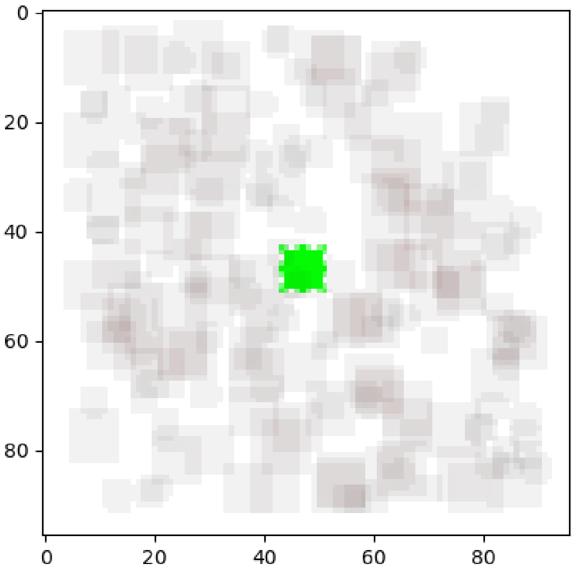

Fig. 4. Images of the 100 initial conditions of the training dataset averaged.

In order to train the noise prediction network ϵ_θ, random samples x^0 are drawn from the data and a random iteration number k is chosen. From this, a noisy sample x^k is constructed by adding k iterations of random noise ϵ^k to x^0. Then, the following loss function is minimized

$$\mathcal{L} = \mathrm{MSE}(\epsilon^k, \epsilon_\theta(x^0 + \epsilon^k, k)).$$

To use DDPMs to represent visuomotor policies, we use the method proposed by [1] which makes two fundamental changes to the standard DDPM. First, the output is modified to represent a trajectory of robot actions. In our experiments the model predicts a series of 16 actions and the first 8 are executed before predicting a new action sequence. By doing inference in this way we essentially execute the policy in a closed-loop Model-Predictive-Control scheme.

Secondly, the de-noising process is conditioned on the current observation O^t using FiLM conditioning [7]. In our experiments the observations are top down images of the environment which show the finger, box, and goal location, in addition to the known finger position (proprioception). The noise prediction network now predicts an action sequence A^0 given the observation sequence O^t and the new loss function is

$$\mathcal{L} = \mathrm{MSE}(\epsilon^k, \epsilon_\theta(O^t, A^0 + \epsilon^k, k)).$$

During policy execution, the first m actions of the action prediction horizon are executed before replanning.

Additionally, to represent the diffusion policy we use the same CNN-based U-Net that is used in [1]. Similarly to [1] we use a ResNet-18 [3] as the image encoder, replacing the global average pooling with a spatial softmax pooling to maintain spatial information.

In practice, we use an action prediction horizon A^0 of 16, an observation horizon O^t of one, and an action execution horizon of eight. We found an observation horizon of one to be sufficient due to our quasi-static setup and the top down, unoccluded camera view. Our actions have dimension two and represent the desired finger positions. We create a Drake *LeafSystem* for our diffusion policy that takes in an image observation, predicts an action sequence, and commands the next action in the action cache until that action is reached. This process is repeated once the action cache becomes empty. We execute these high-level actions using the same low-level PID controller as we used for executing our GCS plans. We train our policy using 100 demonstrations that were generated as described in section III-E. A schematic of the training pipeline is shown in Fig. 5.

IV. Results & Discussion

A. Experiment Setup

We compare our GCS open-loop policy that consists of planning with GCS and executing the plan in an open-loop fashion with a diffusion policy trained on GCS demonstrations, and the same diffusion policy but with three random force disturbances per simulation. For the diffusion policy with disturbances, we apply a random force to the box with a probability of 0.5% at every timestep for a maximum of three disturbances per simulation. For each disturbance, we pick at random one of the four box faces and apply a force f on that face. Fe sample the force magnitude at random, $f \in [50, 300]$ newtons, and apply it for five consequitive timesteps, where each timestep has a duration of 0.01 seconds.

B. Evaluation Metrics

For evaluation, we generate 20 random initial conditions from the same distribution as the inital conditions that were used during training (see section III-E). We run each of our three policies five times for each of these initial conditions. For each of the 100 runs per policy, we log whether the policy succeeded, the run time, the simulation time, and the planning time.

We determine success based on the box's final distance from the goal position. Particularly, we consider a run to be successful if it terminates with the box having an L2 distance of less than 0.1 meters from the goal position. We consider it to be a failure if the box is more than that distance from the goal position after a simulation timeout of 500 seconds. The simulation time refers to the real time required for executing the final policy and consequently is an indicator of the policy's optimality.

The diffusion policy is an online feedback policy and hence plans multiple times during execution. The GCS is an open-loop policy that creates the entire plan at the beginning. We define the run time as the time taken to generate the simulation. This includes the diffusion policy planning time as the policy plans during execution but excludes the GCS open-loop planing time as planning is performed before execution. We take the planning time as the GCS planning time that is required before the open-loop execution. The total time is the sum of both run and plan time, representing the total time taken to

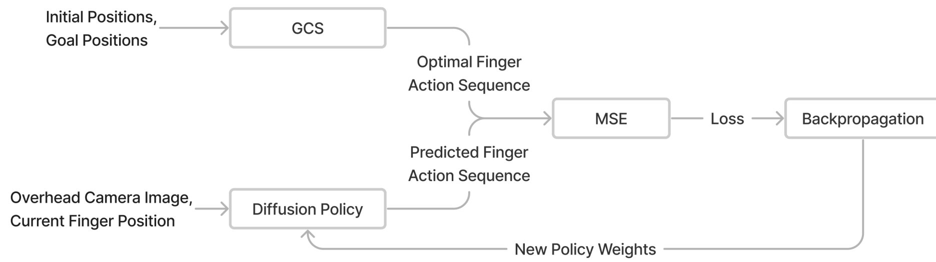

Fig. 5. Pipeline for training diffusion policies.

generate a policy. We only report timings for the successful runs as the unsuccessful runs end after the 500s timeout and hence do not carry informative timing information.

All results were produced with on a Dell Precision 7560 laptop with an A5000 GPU and run one at a time without use of the laptop by other applications. All computations were performed on the CPU apart from the diffusion policy action trajectory predictions that were performed on the GPU. Performing the trajectory predictions on the GPU lead to a five times speedup compared to performing them on the CPU.

C. Experimental Results

We report the success rate and simulation time results in Table I. Notice that the disturbances do not significantly affect the policy's success rate, demonstrating robustness to such disturbances. The GCS open-loop policy has access to the true state of the box and consequently achieves success 100% of the time. The diffusion policy only takes in visual information and as a result achieves a lower success rate. Most diffusion policy failure cases are caused by initial scene states that lie outside of the training distribution. Table II shows the average success rate for each of the 20 initial conditions. It can be noticed that increasing the success threshold L2 distance from 0.1m to 0.2 leads to an increase in success rate for most of the initial conditions. Such an increase also leads to the overall success rate increasing to 0.775 for both the diffusion policy and diffusion policy with disturbances. The initial conditions are additionally visualized with their success rate in Fig. 6. There are four initial conditions that achieve a zero percent success rate, as highlighted in bold in Table II and shown in dark in Fig. 6. By comparing the location of these four initial conditions, as shown in Fig. 6, with the initial box positions in the training data, as shown in Fig. 4, it can be observed that they lie outside the training distribution. Two of the positions are at the top right corner and the other two positions at the bottom left corner of the box position state space. The training data contains none or few initial conditions in these state space regions. Consequently, these four positions represent an expected failure case as behavior cloning is known to not generalize well outside of the training distribution. Looking at the average success simulation times, it can be noticed that the diffusion policy requires longer to reach the goal than the GCS open-loop policy. This is expected as GCS finds optimal paths and is therefore able to reach the goal in the shortest amount of time possible. The diffusion policy has no such guarantees.

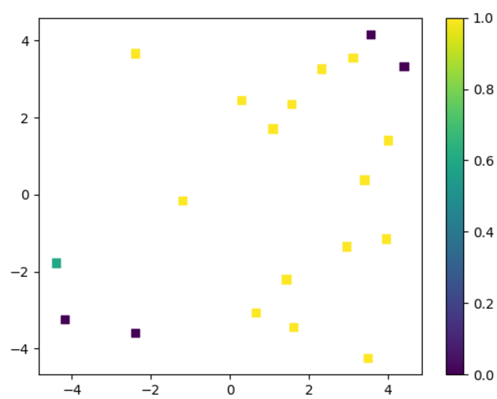

Fig. 6. Initial box positions in the validation dataset colored by success rate (0.2m success threshold).

The average run time, planning time, and total time for successful executions are shown in Table III. It can be seen that due to GCS's long planning time, the diffusion policy is able to compute a plan to the goal quicker than GCS. However, as discussed above, the resulting diffusion policy still takes longer to reach the goal than GCS.

V. Conclusion & Future Work

In this project, we explored using GCS to automatically create optimal demonstrations for Diffusion Policy. We found that the action trajectories executed by the diffusion policy are close to the ones that GCS would have executed for the same initial scene state. Moreover, we show that our Diffusion Policy can achieve a high success rate on a planar

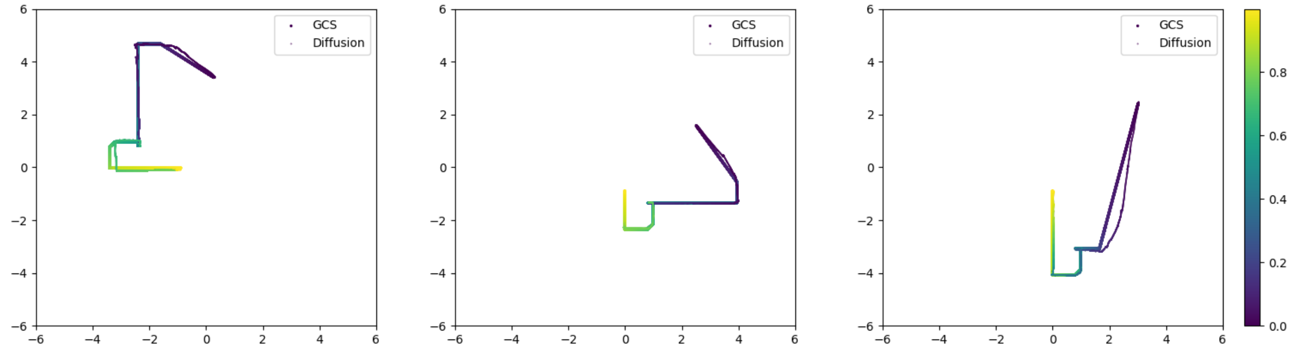

Fig. 7. The trajectories executed by the GCS open-loop policy compared to the ones executed by the Diffusion Policy. The GCS trajectory is shown with a big line and the Diffusion Policy one with a thin line. The trajectories get brighter with simulation time. Each image represents a different initial condition in the evaluation set. Notice that for the first two initial conditions, the Diffusion Policy trajectory is basically identical to the GCS one. For the third initial condition, the Diffusion Policy trajectory is slightly less optimal than the GCS one (curved vs straight line at the beginning).

	GCS Open Loop	Diffusion Policy	Diffusion Policy (Disturbances)
Mean Success Rate	1.00	0.67	0.65
Mean Success Simulation Time (s)	111.97	259.36	259.58

TABLE I

MEAN SUCCESS RATE AND SIMULATION TIME OF THE GCS OPEN-LOOP POLICY, A DIFFUSION POLICY TRAINED ON GCS DEMONSTRATIONS, AND THE SAME DIFFUSION POLICY BUT WITH THREE RANDOM FORCE DISTURBANCES PER SIMULATION.

Initial Condition	0.1m	0.2m	0.1m (Disturbances)	0.2m (Disturbances)
0	1.00	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00
2	1.00	1.00	0.60	1.00
3	1.00	1.00	1.00	1.00
4	0.60	1.00	0.60	1.00
5	1.00	1.00	0.60	1.00
6	1.00	1.00	1.00	1.00
7	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00
9	0.00	1.00	1.00	1.00
10	0.80	1.00	1.00	1.00
11	1.00	1.00	1.00	1.00
12	0.60	1.00	1.00	0.60
13	0.00	0.00	0.00	0.00
14	1.00	0.60	1.00	1.00
15	1.00	1.00	1.00	1.00
16	0.00	0.00	0.00	0.00
17	1.00	1.00	1.00	1.00
18	0.60	1.00	1.00	1.00
19	0.80	1.00	0.60	1.00

TABLE II

SUCCESS RATES FOR THE DIFFUSION POLICY AND DIFFUSION POLICY WITH DISTURBANCES. 0.1M INDICATES THE SUCCESS RATE WHEN SUCCESS IS CONSIDERED AS THE BOX BEING LESS THAN 0.1M FROM THE TARGET AND SIMILARLY FOR 0.2M. THE INITIAL CONDITIONS THAT HAVE A ZERO PERCENT SUCCESS RATE ARE HIGHLIGHTED IN BOLD.

	GCS Open Loop	Diffusion Policy	Diffusion Policy (Disturbances)
Mean Success Run Time (s)	5.50	63.28	80.91
Mean Plan Time (s)	126.07	0.00	0.00
Mean Total Success Time (s)	131.57	63.28	80.91

TABLE III

MEAN RUN TIME, PLAN TIME, AND TOTAL TIME FOR THE GCS OPEN-LOOP POLICY, A DIFFUSION POLICY TRAINED ON GCS DEMONSTRATIONS, AND THE SAME DIFFUSION POLICY BUT WITH THREE RANDOM FORCE DISTURBANCES PER SIMULATION. THE MEANING OF THESE METRICS IS EXPLAINED IN SECTION IV-B.

pushing task from visual observations without the need for human demonstrations. We see great potential in this paradigm as human demonstrations are both sub-optimal and time-consuming to generate.

An obvious next step would be to extend our system to one that allows rotations rather than being axis-aligned. This would then enable planning planar pushing tasks for arbitrary objects such as the T from Diffusion Policy. Doing so should now be possible due to breakthroughs in this are that were made by Bernhard Graesdal while we were working on our project.

We observed that our Diffusion Policy struggles with initial conditions that lie outside the training distribution as is the norm for behavior cloning approaches. A fruitful direction of future work would be to explore a system in which GCS is used to automatically create new demonstrations for such failure modes. This is possible, as GCS can be used to create demonstrations from arbitrary initial positions, something that can be hard to achieve with human demonstrations. Such a system could then alternate between creating demonstrations and finetuning the diffusion policy with the new demonstrations, creating a continuous learning system.

VI. Contributions

We worked jointly on the initial GCS planning through contact formulation. After which Ria took the lead on implementing the additional transition modes such that the finger maintains a buffer around the edge of the box, Shao focused on the infrastructure required to generate training data with GCS and Nicholas worked on training and executing the Diffusion Policy.

References

[1] Cheng Chi et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. 2023. arXiv: 2303.04137 [cs.RO].

[2] Bernhard Paus Graesdal. *planning-through-contact*. https://github.com/bernhardpg/planning-through-contact. 2023.

[3] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In: *CoRR* abs/2006.11239 (2020). arXiv: 2006.11239. URL: https://arxiv.org/abs/2006.11239.

[5] Tobia Marcucci et al. *Motion Planning around Obstacles with Convex Optimization*. 2022. arXiv: 2205.04422 [cs.RO].

[6] Tobia Marcucci et al. “Shortest Paths in Graphs of Convex Sets”. In: *CoRR* abs/2101.11565 (2021). arXiv: 2101.11565. URL: https://arxiv.org/abs/2101.11565.

[7] Ethan Perez et al. “FiLM: Visual Reasoning with a General Conditioning Layer”. In: *AAAI*. 2018.