

Flow as the Cross-Domain Manipulation Interface

Anonymous Author(s)

Affiliation

Address

email

1 Data Collection

In Im2Flow2Act, we collect two types of data: a simulated robot play dataset and real-world human demonstration videos. Note that we do not collect any real-world robot data. Below, we detail the data collection process.

1.1 Simulated Play Data

We use MuJoCo as our simulation engine and have constructed three types of play environments featuring rigid, articulated, and deformable objects. An UR5e robot explores these environments using a set of predefined random heuristic actions.

Rigid: In the rigid play environment, the robot can interact with five different objects, including a toy car, a shoe, a mini-lamp, a frying pan, and a mug. At the beginning of each episode, one object is randomly selected and placed on the table. The robot first picks up the object and starts executing 6DoF random trajectories. A random trajectory is constructed as a 3D cubic Bézier curve with four key waypoints: the start point p_0 , two control points p_1 and p_2 , and the end point p_3 . The start point is where the robot first picks up the object, and the end point is randomly selected within the robot’s reachable world coordinates. The two control points are obtained by adding curvature to the line defined by p_0 and p_3 . Specifically, for the first control point p_1 , we first calculate the one-third point m_1 between p_0 and p_3 , i.e., $m_1 = p_0 + \frac{(p_3 - p_0)}{3}$. We then twist m_1 in 3D space to obtain the first control point by adding a random offset (curvature), i.e., $p_1 = m_1 + \epsilon$, where $\epsilon \in \mathbb{R}^3$. We set each dimension of ϵ be uniformly sampled from $(-0.05, 0.05)$. Similarly, we obtain the second control point by calculating the two-thirds point and adding some random offset. Once we obtain the points p_0, p_1, p_2 , and p_3 , we define the cubic Bézier curve by $(1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$, where $t \in [0, 1]$. We equally sample k ($k = 16$) waypoints along this curve, which defines the translation for the robot’s play trajectory. We add a target object orientation (random sample from $(-\frac{\pi}{8}, \frac{\pi}{8})$ for each dimension) when the object arrives at the final point p_3 . We interpolate between the initial orientation at p_0 and the target orientation at p_3 and attach them to the k waypoints. For each episode, we sample two consecutive 3D cubic Bézier curves, allowing the robot to interact with random objects and build the correspondence between flows and actions. The trajectory ends with a random rotation or place actions.

Articulated: We construct various types of drawers, none of which are the same as those tested in the real world. In each episode, one drawer is selected and placed on the table. The robot explores these articulated objects by opening the drawer to different extents, not necessarily fully open.

Deformable: At the beginning of the episode, a cloth is randomly placed on the table. The robot has the option to grasp either the left or right corner of the cloth. Once grasped, the robot begins random folding (not necessarily folding diagonally). We have defined nine different folding targets, and the robot randomly selects one to start folding. During this process, we also vary the folding trajectory by lifting the cloth to different heights.

In total, we collect 4800 random play trajectories. Once all data is collected, we use an iterative procedure to obtain object flows. For each collected episode, we begin by sampling uniform grid

(30x30) keypoints on the initial frame and track all points using Tapir [1], similar to the approach in ATM [2]. We then apply moving filters and SAM (Segment Anything Model) filters which we will explain in details at section 5 below to obtain keypoints on the object. However, as the initial sampling is uniform, there may be few points located on the objects. To address this, we sample additional points around the existing keypoints based on the SAM output to achieve a denser distribution of object keypoints. Specifically, for segments containing keypoints after filtering, we sample proportionally based on the number of filtered keypoints in that segment. In total, we sample 900 points on each object. Finally, we run the point tracking algorithm again on the newly sampled object keypoints to obtain dense object flow.

1.2 Real-world Human Demonstration Video:

We collect in-domain human demonstration videos for four tasks: pick & place, pouring, opening drawers, and folding cloth. We use a RealSense camera to record the demonstrations at 30 FPS. The descriptions below also serve as detailed task descriptions. In Im2Flow2Act, we use the human videos to provide the system with task information. We define the robot base as the origin of the world coordinate system and use the Right-Handed Coordinate System.

- **Pick & Place:** Pick up a green cup and place it into a red plate. The red plate is randomly placed on one side of the table, while the cup is placed randomly in the middle of the table.
- **Pouring:** Pick up a green cup and hover it over a red bowl, then rotate the cup towards the bowl. The initial placement of the cup and the bowl is the same as in the pick & place task.
- **Drawer Opening:** Fully open a drawer that is randomly placed on one side of the table. The pose of the drawer, including its position and orientation, can vary. The drawer is not rigidly attached to the table.
- **Cloth Folding:** Fold a 33 cm x 33 cm cloth from one corner (either lower left or lower right) to the diagonal. The cloth is randomly placed in the middle of the table.

Once the human demonstration is collected, for each episode, we use Grounding DINO to obtain the object bounding box at the initial frame and uniformly sample the keypoints inside, where we set $H = W = 32$ as sampling parameters. We then run point tracking algorithm to track the keypoints across frames in the episode. To construct the training dataset, we uniformly sample 32 frames from each episode to form the task flow.

2 Ablation study

We conducted an ablation study in simulation to evaluate the impact of using pretrained StableDiffusion (SD) versus training it from scratch on the flow generation model. In this ablation study, we still use pretrained AE (auto-encoder) from the SD but trained the U-Net from scratch instead of incorporating LoRA layers. To ensure a fair comparison, we deploy the same flow-conditioned

Table 1: **Ablation study Results (%)**

	Pick&Place	Pour	Drawer	Cloth
Pretrain U-Net	90	95	90	35
U-Net From Scratch	90	90	95	30

policy for both the pretrained SD and the training scratch as the manipulation policy. We collect data for four tasks in the simulation by substituting the UR5e robot with a sphere robot to create a cross-embodiment scenario. Both networks were trained for the same number of epochs and evaluated within the same initial state. Based on the results shown in Tab. 1, the choice of pretraining had a minor impact on Im2Flow2Act’s final performance. This suggests that: *i*. Although StableDiffusion was initially designed for image generation, directly using its pretrained weights for flow generation does not impact its performance. Utilizing LoRA can lead to better training efficiency compared to

80 training from scratch. *ii.* The latent space encoded by the AE from the pretrained SD might benefit
 81 the diffusion model’s learning process.

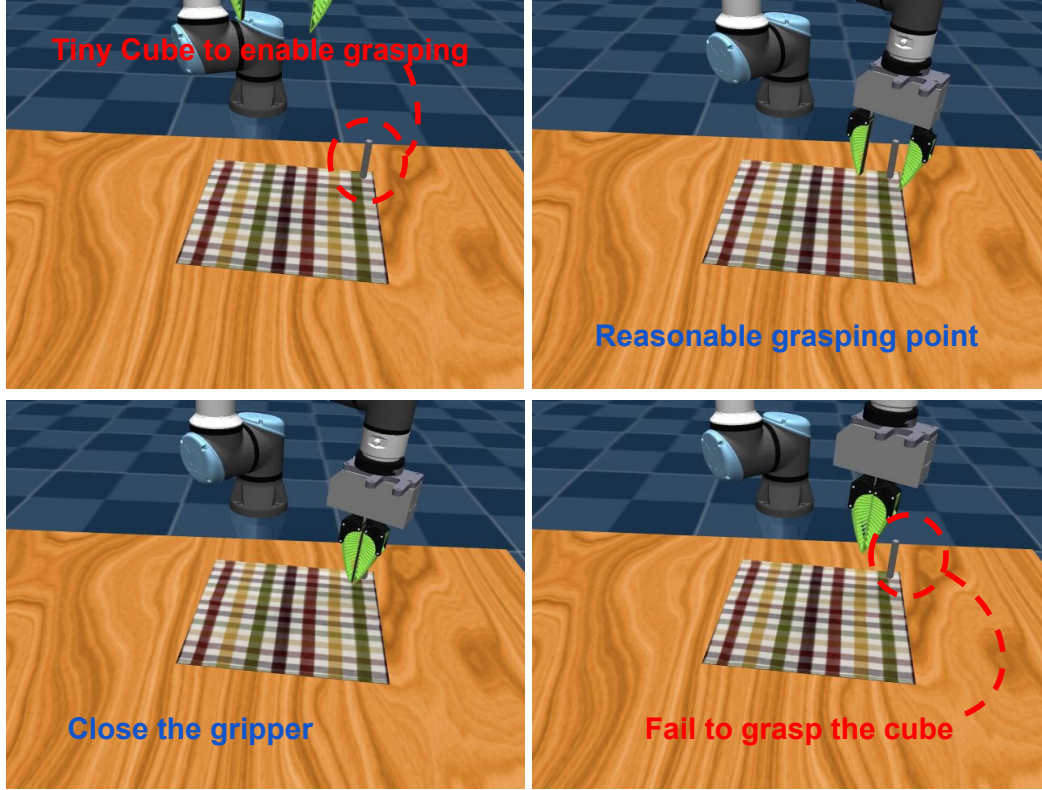


Figure 1: **Policy rollout in Deformable Environment.** We attach a tiny cube (1cm x 1cm x 9cm) to enable robot grasp deformable objects. The generated flow guides the policy toward an appropriate grasping point; however, it fails to grasp the tiny cube, resulting in task failure.

82 Compared to the results using ground truth flow in simulation, we observed that for tasks like pick
 83 & place, pouring, and drawer opening, the success rates by taking generated flow as input are very
 84 close. This further demonstrates flow generation network’s capabilities. However, the success rate
 85 for cloth folding drops significantly. We attribute this to the way we constructed the cloth folding
 86 simulation environment. Grasping deformable objects like cloth is challenging, so we attached a
 87 cube (See Fig. 1) at the corner of the cloth to help the robot to lift the corner by grasping the cube.
 88 During data generation and inference, we render only the cloth, not the cube, to mimic realistic cloth
 89 folding behavior. We made the cube’s dimensions on the xy-plane very small (1cm x 1cm) to emulate
 90 the width of a gripper grasping actual cloth. Since the output from a diffusion model typically
 91 exhibits multimodal distributions and our training dataset contains considerable randomness, the
 92 learned flow generation model produces reasonable flows for the folding task, but may not direct
 93 the policy to grasp the cube precisely. Moreover, the flows after motion filtering exacerbate the
 94 issue. We found that in most cases, the robot can reach the corner and attempt the grasp (See Fig.
 95 1), but it fails to accurately grasp the cube. In contrast, we observed a reasonable success rate in
 96 real-world experiments because, in practice, the robot can grasp the cloth anywhere along the corner
 97 to executing folding.

98 3 Experimental Details

99 3.1 Real-World Setup

100 We use a UR5e robot equipped with a WSG-50 gripper. During inference, the UR5e receives end-
 101 effector space positional commands from a 2.5 Hz policy. We limit the end-effector’s speed to less

than 0.2 m/s and restrict its position to at least 1 cm above the table for safe execution. A RealSense D415 depth camera is mounted at the table to capture policy observations, including the initial depth image for the initial point cloud x_0 and the real-time RGB images used for online point tracking. The RealSense camera records at 720p and 30 Hz. We downsample the resolution to 256x256 at 5 Hz for the online point tracking algorithm to process. Additionally, we use a smartphone positioned at a different angle to better record robot execution. A desktop with a 24 GB NVIDIA RTX 4090 runs the flow generation network and flow-conditioned imitation learning policy inference and online point tracking algorithm.

3.2 Real-World Evaluation Protocol

In this section, we describe the details of the evaluation protocol in the real world.

1) **Initial State:** When evaluating our system, we do not match the background scene setup to those recorded in the human demonstrations such that we can also test the generalization capability of flow generation network. For the initial position of objects, they are placed with roughly the same distribution as in the human video demonstrations.

2) **Success Metric:** Below are the details of the real-world success metric for all four tasks:

- **Pick & Place:** The robot needs to pick up the cup and place it into the red plate. We do not require the mug to be centered on the plate. We consider one episode successful if: *i.* the robot can steadily place the mug onto the red plate; *ii.* the plate does not move more than 5 cm on the table during the robot’s manipulation process.
- **Pouring:** The robot needs to pick up the cup, hover it near the bowl, and execute pouring actions. We consider one episode successful if: *i.* the robot demonstrates the complete pouring behavior by rotating the cup more than 30 degrees; *ii.* at least half of the cup overlaps with the red bowl on the x-axis in terms of world coordinates.
- **Drawer Opening:** The robot needs to fully open the drawer without colliding with its surface. Since the drawer is not rigidly attached to the table, it may slightly slide during the robot’s execution. An episode is considered a success if: *i.* the robot gripper does not heavily collide with the drawer surface and push the drawer back more than 5 cm; *ii.* the drawer is fully opened;
- **Cloth Folding:** The robot needs to fold the cloth from one corner to the diagonal opposite corner. We consider an episode successful if: *i.* the robot folds the corner as indicated by the generated flow; *ii.* the two corners are within a threshold once the robot completes the execution. Due to the large sim2real gap for deformable simulation, we set this threshold as 7 cm.

3.3 Evaluation Procedure

Im2Flow2Act with/without alignment: For each task, we first record the initial frame with different initial states for 20 evaluation episodes. We then query Grounding DINO on the object of interest to obtain the bounding boxes in all initial frames. Using the bounding box, initial frame, and the task description, we generate the object flow (task flow) for all episodes and store them as a buffer on the disk. With all ingredients for policy inference set, we start policy evaluation for each episode by manually matching the initial states to be close to pixel-perfect within the mounted RealSense camera and load the corresponding generated flow from the previously saved flow buffer.

Heuristic-based policy: To obtain ground truth future point clouds for the objects, we first record human demonstrations for 20 evaluation episodes. We store both RGB and depth images during this process. For each evaluation episode, we manually match the initial states to be close to pixel-perfect and obtain the open-loop action sequence by estimating object pose transformations between the initial frame and future frames for each time step in human demonstrations. We use the same motion filters in Im2Flow2Act to ensure fair evaluation. Furthermore, we provide the maximum available points (without downsampling) from the task flow. We manually check the transformed point cloud by overlapping the transformed initial frame point cloud and future frame point cloud to ensure the transformation is largely correct under the noisy conditions of the real-world depth camera.

151 4 Training Details

152 4.1 Flow Generation Network

153 For the rectangular flow image, we set the spatial resolution to $H = W = 32$ and $T = 32$,
154 generating flow for 1024 keypoints over 32 steps. We finetune the decoder from StableDiffusion
155 for 400 epochs with a learning rate of $5e - 5$. To obtain these keypoints, we uniformly sample
156 them from the bounding box provided by Grounding DINO. For training AnimateDiff, we insert the
157 LoRA (Low-Rank Adaptation) with a rank of 128 into the Unet from StableDiffusion and train the
158 motion module layer from scratch with learning rate of 1×10^{-4} for 4000 epochs using AdamW
159 [3] optimizer with weight decay 1×10^{-2} , betas (0.9, 0.999) and epsilon 1×10^{-8} . We load the
160 pretrained (openai/clip-vit-large-patch14) weights from CLIP [4] to process the initial frame and
161 freeze them during the entire training. Zero-initialized linear layers are used to process the patch
162 embedding and the initial keypoints embedding before passing the conditions into the cross-attention
163 layers.

164 4.2 Flow-Conditioned Imitation Learning Policy

165 **Training Data Format:** A training sample consists of $(\rho_t, f_t, \mathbf{a}_t, F_{0:T})$, where ρ_t is the propriocep-
166 tion data, \mathbf{a}_t is a sequence of actions a_t, \dots, a_{t+L} of length L , and f_t contains the locations (u, v)
167 of $N = 128$ object keypoints in the image space at time t . We set the object flow (i.e., task flow)
168 horizon $T = 32$, which matches the output of the flow generation network. The action sequence
169 length is set to 16. The N keypoints are randomly selected from all available keypoints for every
170 training sample during the training process. To construct the task flow $F_{0:T}$, we randomly select T
171 frames from the episode length T' to which the training samples belong. To ensure the task flows
172 are complete, we include both the first and last frames of the episode in the task flows.

173 **State Encoder:** We project the keypoints' initial 3D coordinates, X_0 , into a 192-dimension vector
174 using a linear layer. We also encode keypoints' locations (u, v) in image space into another 192-
175 dimension vector, using a fixed 2D sinusoidal positioning. These two vectors are concatenated to
176 form the descriptor ϵ , with a total dimension of 384. We then pass all keypoints' descriptors into the
177 state encoder ϕ , which is a transformer with 4 encoder layers. It outputs a state representation of
178 dimension 384 using a CLS token.

179 **Temporal Alignment:** As discussed in the main paper, during training, we first encode the remain-
180 ing task flow $f_{t:T'}$ into $z_t \in \mathcal{Z}$. This process involves encoding the keypoints at each time step $f_{t:T'}$
181 into $s_{t:T}$ via the state encoder. Next, we encode the future state representation $s_{t:T}$ into the latent
182 space through the encoder ξ , which is implemented as a transformer with 4 encoder layers. We use
183 fixed 1D sinusoidal positional encoding to preserve temporal information in the state representation
184 $s_{t:T}$ before feeding them into ξ . The Temporal Alignment model is implemented as a transformer
185 with 8 encoder layers. We also add fixed 1D sinusoidal positional encoding to all inputs and utilize
186 a CLS token for making predictions.

187 **Diffusion Action head:** We use the diffusion policy [5] as our action head. We use DDIM scheduler
188 with 50 training diffusion steps and 16 inference steps.

189 We train the policy for 500 epochs with learning rate $1e-4$ using AdamW with weight decay 1×10^{-2} ,
190 betas (0.9, 0.999) and epsilon 1×10^{-8} .

191 5 Inference Details

192 In this section, we describe the details of the inference process, which includes Grounding DINO,
193 motion filters, and online point tracking.

194 5.1 Grounding DINO

195 For each task, we begin by using Grounding DINO to identify the object of interest. We manually
196 provide the keyword to the model; however, this process could potentially be automated using a
197 large language model to find the desired object in the task description. Specifically, we employ the
198 grounding-dino-base model to extract the object’s bounding box. The keywords used for the pick &
199 place and pouring tasks are “green cup”. For drawer opening, the keyword is “yellow drawer”, and
200 for cloth folding, it is “checker cloth”. The input images are processed at a resolution of 480x640.

201 5.2 Motion Filters

202 We use motion filters to process the object flow (i.e., task flow) generated from the flow gener-
203 ation model. As explained in the main paper, the initial keypoints are constructed by uniformly
204 sampling within the bounding box. This approach inevitably yields keypoints that are not on the
205 object, specifically, keypoints that fall on the background. To address this, we deploy several filters
206 simultaneously to remove these background keypoints. Additionally, we implement depth filters to
207 eliminate keypoints that lack depth data from noisy real-world depth image.

208 **Moving Filter:** In the training set, keypoints sampled on the background remain static in the image
209 space, as only the object is moving. Therefore, we deploy a moving filter during inference time to
210 remove keypoints whose movement in the image space (256x256) is below a certain threshold. We
211 find that this filter effectively eliminates most background keypoints. In real-world experiments, we
212 set the threshold as 20 for pick & place, pouring, and drawer opening tasks, and as 10 for cloth
213 folding.

214 **SAM Filter:** To further remove points after applying the moving filter, we deploy the Segment
215 Anything Model (SAM) [6]. Specifically, we first resize the initial frame to 256x256 and pass it
216 through SAM to obtain the finest segmentation. We then iterate through the keypoints and filter out
217 those where the area of the located segment exceeds a threshold. We use a high threshold value of
218 10,000 for all tasks to prevent filtering out keypoints on objects with rich textures.

219 **Depth Filters:** Real-world depth images are often noisy and contain many “holes.” We filter out
220 keypoints where the depth value is missing (i.e., the value is zero).

221 We randomly select $N=128$ keypoints which is the same number we used for training after applying
222 motion filters as the policy input.

223 5.3 Online Point Tracking:

224 We utilize the online point tracking function from Tapir [1] to track the filtered keypoints during
225 inference. We resize the visual observations to 256x256 and run the online point tracking at 5Hz.

References

- [1] C. Doersch, Y. Yang, M. Vecerik, D. Gokay, A. Gupta, Y. Aytar, J. Carreira, and A. Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10061–10072, 2023.
- [2] C. Wen, X. Lin, J. So, K. Chen, Q. Dou, Y. Gao, and P. Abbeel. Any-point trajectory modeling for policy learning. *arXiv preprint arXiv:2401.00025*, 2023.
- [3] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.
- [4] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.
- [5] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. C. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.026.
- [6] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv:2304.02643*, 2023.