# Supplementary Material
# MILES: Making Imitation Learning Easy with Self-Supervision

For videos demonstrating MILES' performance and code implementation please see our anonymous webpage: https://sites.google.com/view/miles-imitation.

## A MILES: Additional Details on the Method

### A.1 Method Pseudocode

We provide a detailed **pseudocode** describing MILES, in Algorithms 1- 8.

### A.2 Validity Conditions for Augmentation Trajectories

As described in section 3.3, after collecting data for an augmentation trajectory, we check for two conditions: (i) **reachability** and (ii) **environment disturbance**, to determine whether an augmentation trajectory is valid and eligible to fuse with the demonstration. Figure A.1 shows examples of these two conditions.

#### A.2.1 How do we check for the Reachability condition?

**Reachability.** To check for reachability, after executing an augmentation trajectory $\tau_k$, we verify whether the final achieved pose matches the pose of the $k_{th}$ demonstration waypoint using proprioception, as described in section 3.3. Pseudocode describing how we check for reachability is also provided in Algorithm 3. It is crucial to check for reachability because an augmentation trajectory that does not meet this condition cannot be fused with the demonstration, as it cannot return to the demonstration state. If the waypoint $w_k^\zeta$ is unreachable during data collection, we cannot automatically determine how to reach $w_k^\zeta$ from $w_M^{\tau_k}$, without collecting observations that do so during self-supervised data collection. Consequently, we cannot automatically determine what actions to take to return back to the demonstration from $w_M^{\tau_k}$, as we can with valid augmentation trajectories. Figure A.1 (a, left) shows an example where the reachability condition is not met due to environmental dynamics, such as a key getting "jammed" and failing to reach the target waypoint due to collision and friction in the lock. A similar example where the reachability condition is met is shown in Figure A.1 (a, right).

#### A.2.2 How do we check for the Environment Disturbance condition?

**Environment Disturbance.** To determine whether an environment disturbance occurred, we compare the RGB image captured at the $k_{th}$ demonstration timestep with the RGB image captured at the final timestep of the augmentation trajectory, as described in section 3.3. A detailed pseudocode describing how we determine whether an environment disturbance occurred can be found in Algorithm 5, and a visual example can be seen in Figure A.1 (b). The comparison between the two RGB images relies on the similarity of their DINO features [26]. Specifically, we use a pre-trained DINO ViT [26] to obtain the DINO features for different patches of each image similarly to [27]. By computing the cosine similarity between the DINO features of each corresponding image patch in $I_k^\zeta$ and $I_M^{\tau_k}$, we can calculate the average similarity between the two images [27]. If the similarity is below a threshold $\theta$ (to see how we automatically determine $\theta$ please see section B.2.3), we assume the robot has disturbed the environment, and data collection is stopped. Our experiments showed that DINO ViT features are necessary because they are robust to lighting changes and noise in the RGB image. Other methods we tried, such as template matching or computing the per-pixel Euclidean distance, proved brittle and sensitive to lighting variations or noise in the captured images.

Understanding why checking for an environment disturbance is important is straightforward. Consider the rectangular object shown in Figure A.1 (b), and assume the task is to learn how to pick up that object. If the robot pushes the rectangular object, causing it to fall over during data collection, the image observed after returning to the demonstration state will no longer match that state's observation from when the demonstration was provided. Consequently, from the point where the disturbance occurred onward, we have no way of knowing how to reach any of the remaining

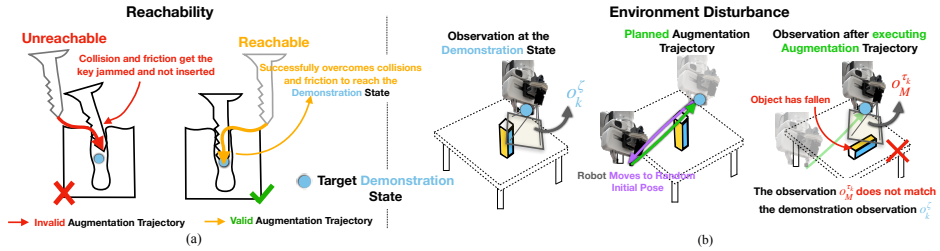


Figure A.1: **Reachability:** Two examples of possible augmentation trajectories for a locking task are shown; an invalid trajectory (left) that fails to reach the target demonstration waypoint due to collisions, friction, and potentially inevitable systematic controller errors and a valid one (right) that successfully reaches the target waypoint. **Environment Disturbance:** As the robot collects an augmentation trajectory, it perturbs the environment such that after returning to the demonstration's waypoint the live observation and the demonstrated one no longer match, indicating that data collection should stop.

demonstration states and as a result how to solve the task. This is because we only know how to solve a task by learning how to follow the demonstration after returning to it. But if an environment disturbance has occurred (e.g., the rectangular object has fallen), following the demonstration's actions no longer leads to task completion. Hence, if data collection continued, all future augmentation trajectories would contain invalid observations and actions, as they would demonstrate behavior that does not solve the task that the human demonstrated. This is why we stop data collection after detecting an environment disturbance.

## A.3 MILES' Policy

**Training:** To train our manipulation policy we leverage the dataset $\mathcal{D}_{new}$ comprising the fused augmentation trajectories with the demonstration as described in section 3.4. MILES' policy $\pi$ comprises either (1) an end-to-end network trained with behavioral cloning (BC) or (2) an end-to-end network trained with BC combined with demo replay, which is utilized when data collection was interrupted due to a detected environment disturbance.

### A.3.1 How is our policy defined when No Environment Disturbance occurred during data collection?

**No Environment Disturbance.** When no disturbance occurred our dataset $\mathcal{D}_{new}$ contains augmentation trajectories that can return to and then follow the demonstration from every state. In that case, we leverage $\mathcal{D}_{new}$ to train an end-to-end behavioral cloning policy $\pi$ that comprises a single neural network $f_\psi$, parameterized by $\psi$, that receives as input an RGB image captured from the wrist camera and force-torque feedback to predict 6-DoF actions: $f_\psi : \mathbb{R}^{H\times W\times 3} \times \mathbb{R}^6 \to \mathrm{SE}(3)$ as well as an additional binary value indicating the gripper action ($\mathbb{R}^{H\times W\times 3}$ refers to the RGB images where $H$: height, $W$: width and $\mathbb{R}^6$ to measured forces and torques). The force-torque feedback is captured directly using Franka Emika Panda's joint force sensors. For our policy to generalize spatially, *no* proprioception input is passed to $f_\psi$ and all actions are predicted relative to the EE's frame. $f_\psi$ consists of a ResNet-18 backbone [37] for processing RGB images, and a small MLP embeds force feedback into a 100-dimensional space. The output of the force MLP and ResNet-18 are concatenated and fed into an LSTM [38] network for action prediction. The network is trained using standard behavior cloning to maximize the likelihood of $\mathcal{D}_{new}$.

### A.3.2 How is our policy defined when an Environment Disturbance occurred during data collection?

**Environment Disturbance.** When self-supervised data collection was stopped due to an environment disturbance, our dataset $\mathcal{D}_{new}$ contains augmentation trajectories that can return the robot to any state from the initial demonstration state up to the demonstration state at timestep $R$, where $R < N$ (see section 3.2). In this scenario, if our policy consists only of $f_\psi$, then during task execution the robot would be able to solve the task only up to the $R_{\text{th}}$ state, but not complete it. As such, we define our policy $\pi$ to consist of two components: (1) the first component is a neural network $f_\psi$ identical to the above scenario, but trained up to the $R_{\text{th}}$ state and (2) the second component corre-

sponds simply to the sequence of the remaining demonstration actions from the $R_{\text{th}}$ state onwards, for which no self-supervised data was collected, i.e., $\zeta_{remaining} = \{a_n^\zeta\}_{n=R}^N$.

#### A.3.3 How do we deploy MILES' policy?

**Deployment:** Deploying the policy is straightforward and depends on whether data collection was interrupted due to an environment disturbance. If uninterrupted, then only the neural network $f_\psi$ is used to complete the task equivalently to policies trained using reinforcement learning or behavioral cloning.

If data collection was interrupted, first $f_\psi$ is deployed to solve the task up to the $R_{\text{th}}$ state in an identical way as the scenario of "no environment disturbance". After the robot reaches the $R_{th}$ state then $\zeta_{remaining}$ is executed. We determine whether the closed-loop policy has completed the task up to the $R_{th}$ in a very simple way as described in section A.3.4.

Pseudocode describing MILES' policy deployment can be found in Algorithm 8.

#### A.3.4 How do we determine when to switch from closed-loop control to demonstration replay?

Switching from closed-loop to demonstration replay is straightforward. As the objects and the robot can be at different poses during deployment from the ones during data collection, we cannot just use the robot's proprioception to know when the $R_{\text{th}}$ state has been reached. Hence, we deploy $f_\psi$ until it predicts continuously the identity transformation, indicating no robot movement. Then, we switch to demonstration replay, where we replay the rest of the demonstration $\zeta_{remaining}$.

## B More details on the Experimental Setup

### B.1 Pose Estimation

In practice, as with most methods [4, 14, 15, 18], we naturally provide the demonstrations starting near the task-relevant object. As such, we need a way to ensure that MILES can still solve any task regardless of how far the robot is from an object. An apparent solution to this is to provide the demonstration starting from a pose far away from the object and deploy MILES' data collection. While this would work well, it may be inconvenient. As such, inspired by [2, 16, 18] we use a simple pose estimator at deployment to estimate the relative pose between the robot at the initial state of the demonstration (for which MILES collected data) and the task-relevant object. As we do not assume any 3D object models, we use the method deployed in [15] although any other model-free pose estimator can be used. This allows us to first coarsely estimate the pose and move near the task-relevant object from any robot starting pose before deploying MILES. Uncut videos demonstrating this behavior can be found on our anonymous webpage: https://sites.google.com/view/miles-imitation.

### B.2 MILES Data Collection Hyperparameters

#### B.2.1 How do we set the data collection range around each demonstration waypoint?

As discussed in our experiments section 4, we collect data in a range of 4cm and 4 degrees around each demonstration waypoint. However, this range is *not limiting* and can be set to *any desirable range* like any other robot learning method. In our case, we set this range to be the average pose estimation error to reach the initial pose of the demonstration relative to the task-relevant object using the pose estimation method described in section B.1 which we obtained based on [15].

#### B.2.2 How do we determine the number of augmentation trajectories to collect for each demonstration waypoint?

For all of our experiments, we set the number of augmentation trajectories per demonstration waypoint, $Z = 10$. In our case, we set this arbitrarily, but as we showed in our method's data collection ablation in section 4.4 different tasks require different numbers of augmentation trajectories. As such, we provide two guidelines for setting the value for $Z$. Firstly, high tolerance tasks, like the

| Task: | Description | DCT | Task: | Description | DCT |
|---|---|---|---|---|---|
| **Lock with key** | Insert a key into a lock and rotate 90 degrees to lock it. | 24' | **Twist screw** | Insert a toy screwdriver into a screw and twist by 90°. | 22' |
| **Insert USB** | Insert a USB stick into a USB port ($<$ 1mm tolerance) | 21'. | **Bread in toaster** | Put a plastic bread inside a toaster. | 40' |
| **Plug into socket** | Plug a UK plug (3-pin) to a socket. | 37' | **Open lid** | Lift the lid of a blue box. | 31' |
| **Insert power cable** | Plug the power cable into the power port of a PC. | 28' | | | |

Table 3: Task descriptions of the 7 tasks used in our experiments. **DCT** stands for Data Collection Time and corresponds to the time spent collecting self-supervised data.

"Open lid" task reported in our experiments usually require a small number of augmentation trajectories. On the other hand, precise tasks, like the "USB insertion" task reported in our experiments require more augmentation trajectories. Secondly, as the data collection range around each demonstration waypoint increases, the number of augmentation trajectories collected should also increase with an approximately linear relationship, i.e., if the range is doubled, then the number of augmentation trajectories should be doubled as well. We recommend as a starting point, for a data collection range similar to our experimental setting of 4cm and 4 degrees, to collect 10 augmentation trajectories for precise, low-tolerance tasks, and 4 augmentation trajectories for high-tolerance tasks.

#### B.2.3 How do we determine the Environment Disturbance threshold $\theta$ ?

We determined $\theta$ simply by spawning several random RLBench [39] tasks in CoppeliaSim and running MILES. By setting up custom heuristics that determine environment resets in the simulation we found that for the DINO model we use, a similarity of $\theta < 0.94$ appeared to detect environment disturbances across all tasks successfully. Consequently, we used that in our real-world experiments too.

### B.3 Task Descriptions

A detailed description of each task along with their Data Collection Times (**DCT**) can be found in Table 3.

#### B.3.1 For which tasks was an Environment Disturbance detected?

An environment disturbance was detected for the following tasks: `Twist screw`, `Bread in Toaster` and `Open lid`. As such for these tasks the policies comprise a closed-loop and a demonstration replay component.

We also note that for the lock with key task, we stopped data-collection "half-way" through the 90 degrees twisting rotation for hardware safety. This is because the forces exerted on the robot as it was collecting self-supervised data were too high. In this case, we treated this identically to an environment disturbance. At deployment, the learned policy completes most of the task closed-loop, apart from a small twisting motion done with demo replay, after the closed-loop policy converges to predicting the identity transformation as discussed in section A.3.4. This is similar to adding force limits to reinforcement learning algorithms and was done to protect our robotic hardware; however, doing so is not a requirement.

### B.4 Baselines

Here, we provide further implementation details on two of the baselines we used in our paper.

**Pose Estimation + Demo Replay.** For this baseline, we follow the same problem formulation as in [15], but improve upon that baseline in two key ways: (1) the data on which it is trained on is the same data collected for MILES, as such it contains only valid trajectories that cover a larger part of

the task space and (2) instead, of replaying recorded velocities, we also replayed the recorded forces which is particularly important for the contact rich tasks.

**Reset-Free FISH** [4]. For Reset-Free FISH we use the implementation provided by the authors as it can be found in: https://github.com/siddhanthaldar/FISH. We only changed the implementation such that the policy always predicts 6-DOF actions instead of constraining the output to specific DOFs, as doing so assumes access to prior task knowledge. To learn residual actions on top of the demonstration we tested both using demo replay as the base policy, as well as VINN [40] but found that demo replay led to better performance.

### B.5 Generalization Performance

We reported results on generalization in section 4.5 on the "Markers in Bin" task shown in Figure 3 (8). The goal of this task was to throw one of the two shown markers (blue, green) in the bin that is available at deployment time. MILES was trained on the 5 bins marked as green in Figure 3 (8) and the generalization test was done in the two bin shown on the left, marked as red. The data collection time for this task was on average 34 minutes for each bin and an environment disturbance was detected for each bin. To determine which remaining actions to replay for the previously unseen bins, we selected the remaining actions from the bin in the training set whose RGB image in the demonstration has the highest similarity in terms of DINO features with the bin during deployment, inspired by prior work [17]. Videos exhibiting MILES generalization on the two test case bins can be found on our webpage: https://sites.google.com/view/miles-imitation.

### B.6 Multi-stage Tasks

To evaluate MILES' ability to solve multi-stage tasks, we tasked MILES with picking up the plastic bread shown in Figure 3 (6) (as part of the "Bread in Toaster" task) and inserting it into the toaster. To achieve this we broke the task down into two stages: first, we provided a demonstration showing how to pick up the bread and trained MILES. Then, we used the policy already trained on the "Bread in Toaster" task to finish the task. To link the two stages together, first the policy to pick up the bread is deployed. After, the execution ends, the robot returns to its default position. Then, the pose estimation method described in section B.1 is deployed to approach the toaster, and then the policy trained with MILES is deployed to insert the bread into the toaster. Videos exhibiting MILES' multi-stage task performance on picking up and inserting the bread into the toaster can be found on our webpage: https://sites.google.com/view/miles-imitation.

### B.7 Performance with distractors

We found that performing standard image augmentation techniques, including changing the brightness, contrast, noise, cropping random image parts, etc... allowed MILES to be robust to distractor objects, as shown in the videos provided on our anonymous webpage: https://sites.google.com/view/miles-imitation.

## C Detailed Pseudocode

**Algorithm 1: MILES Overview** (Simplified)

**Input:** Single Task Demonstration: $\zeta = \{(w_n^\zeta, o_n^\zeta, a_n^\zeta)\}_{n=1}^N$, Number of augmentation trajectories per demonstration waypoint $Z$, environment disturbance threshold $\theta$ (Default: $\theta = 0.94$)

$\mathcal{D} = \{\}$ `// init empty dataset of augmentation trajectories`
Reachable = **True** `// init variable that tracks reachability`
Disturbance = **True** `// init variable that tracks environment disturbances`
$R = 1$ `// init variable that stores the timestep when self-supervised data collection stops`
Move robot to the initial demonstration pose $w_1^\zeta$
**for** iteration $k = 1$ to $N$ **do**
  $j = 1$ `// init variable that tracks the number of collected augmentation trajectories per demo waypoint`
  **while** $j \leq Z$ **do**
    $\tau_k \leftarrow$ *SampleTrajectory*($w_k^\zeta$) (Alg. 2)
    Reachable $\leftarrow$ *CheckReachability*($w_k^\zeta$) (Alg. 3)
    **if** Reachable is **False then**
      *ReturnToDemoWaypoint*($k$, $\zeta$) (Alg. 4)
      Break `// exit while loop`
    **end if**
    $I_M^{\tau_k} \leftarrow$ Capture RGB wrist-cam image `//` $M$ `is the` $M_{th}$ `(final) timestep of` $\tau_k$
    Disturbance $\leftarrow$ *CheckEnvDisturbance*($o_k^\zeta$, $I_M^{\tau_k}$, $\theta$) (Alg. 5)
    **if** Disturbance is **True then**
      $R = k$ `// store timestep when data collection stops`
      Break `// exit while loop`
    **end if**
    $\mathcal{D} = \mathcal{D} \cup \tau_k$ `// add augmentation trajectory to dataset`
    $j = j + 1$
  **end while**
  **if** Disturbance is **True then**
    Break `// exit for loop`
  **end if**
  Proceed to the next demonstration state by performing action $a_k^\zeta$ `// follow the demonstration's progression`
**end for**
$\mathcal{D}_{\text{new}} \leftarrow$ *FuseAugmentationsWithDemo*($\mathcal{D}$, $R$, $\zeta$)(Alg. 6)
$\pi \leftarrow$ *TrainPolicy*($\mathcal{D}_{\text{new}}$, $R$, $\zeta$)(Alg. 7)
*Deploy*($\pi$, $R$, $\zeta$)(Alg. 8)

**Output:** $\pi$

**Algorithm 2:** SampleTrajectory

**Input:** Demonstration waypoint $w_k^\zeta$

$\tau_k = \{\}$ // init empty augmentation trajectory

Sample initial pose $w_1^{\tau_k}$ and move robot (Optional: record trajectory poses)

Move back to $w_k^\zeta$ // either by tracking the recorded trajectory poses backward or by re-planning a new, straight-line trajectory (equal performance, the former often leads to faster data collection).

$m = 1$ // observations, actions index

**while** moving to $w_k^\zeta$ **do**

  $\tau_k = \tau_k \cup (w_m^{\tau_k}, o_m^{\tau_k}, a_m^{\tau_k})$ // add waypoints, observations and actions to augmentation trajectory; actions are automatically inferred as the relative EE poses between consecutive timesteps; gripper actions are automatically copied from the demonstration.

  ($o_m^{\tau_k}$ comprises wrist cam RGB images + force-torque readings)

**end while**

**Output:** Return augmentation trajectory $\tau_k$

**Algorithm 3:** CheckReachability

**Input:** Demonstration waypoint $w_k^\zeta$

Reachable $\leftarrow$ **True** // init reachability variable

$w_M^{\tau_k} \leftarrow$ EE pose // achieved after executing the augmentation trajectory (comprising $M$ timesteps); read from proprioception

Reachable $= (w_M^{\tau_k} == w_k^\zeta)$ // check whether poses are equal (within the controller's feasible precision)

**Output:** Reachable

**Algorithm 4:** ReturnToDemoWaypoint

**Input:** Demonstration timestep $k$, single demonstration $\zeta$

Move to initial demonstration waypoint $w_1^\zeta \in \zeta$ // replay demonstration up to the $k_{\text{th}}$ timestep

**for** iteration $t = 1$ to $t = k$ **do**

  Perform action $a_t^\zeta \in \zeta$

**end for**

**Algorithm 5:** CheckEnvDisturbance

**Input:** Demonstration observation $o_k^\zeta$, captured live image $I_M^{\tau_k}$, similarity threshold $\theta$

Disturbance $\leftarrow$ **False** // init environment disturbance variable

$I_k^\zeta \in o_k^\zeta$ // retrieve RGB image $I_k^\zeta$ from the demonstration's observations

$[f^1_{I_k^\zeta}, f^2_{I_k^\zeta}, ...] \leftarrow$DINO-ViT($I_k^\zeta$) // compute DINO-ViT features [27, 26] for **each** image patch $f^x_{I_k^\zeta}$ for the demo waypoint image

$[f^1_{I_M^{\tau_k}}, f^2_{I_M^{\tau_k}}, ...] \leftarrow$DINO-ViT($I_M^{\tau_k}$) // compute DINO-ViT features [27, 26] for **each** image patch $f^x_{I_M^{\tau_k}}$ from the current live environment image (captured after executing the augmentation trajectory).

$sim =$AvgCosineSimilarity($[f^1_{I_k^\zeta}, f^2_{I_k^\zeta}, ...], [f^1_{I_M^{\tau_k}}, f^2_{I_M^{\tau_k}}, ...]$)

**if** $sim < \theta$ **then**

  Disturbance $\leftarrow$ **True**

**end if**

**Output:** Disturbance

**Algorithm 6:** FuseAugmentationsWithDemo

**Input:** Dataset of augmentation trajectories $\mathcal{D}$, final data collection time step $R$, single demonstration $\zeta$

$\mathcal{D}_{\text{new}} = \{\}$// init empty dataset to store fused trajectories

**for** $\tau_k$ in $\mathcal{D}$ **do**

  $\zeta_{\text{segment}} = \underbrace{\{(w_n^\zeta, o_n^\zeta, a_n^\zeta)\}_{n=k}^{R}}_{\substack{\text{demonstration segment from } k_{\text{th}} \\ \text{demo waypoint to } R_{\text{th}}}} \in \zeta$

  $\tau_{k_{\text{new}}} := \tau_k \cup \zeta_{\text{segment}}$

  $\mathcal{D}_{\text{new}} = \mathcal{D}_{\text{new}} \cup \tau_{k_{\text{new}}}$

**end for**

**Output:** $\mathcal{D}_{\text{new}}$

**Algorithm 7:** TrainPolicy

**Input:** Dataset of augmentation trajectories + demo $\mathcal{D}_{\text{new}}$, final data collection timestep $R$, single demonstration $\zeta$

Train neural network $f_\psi$ on $\mathcal{D}_{\text{new}}$ using standard behavioral cloning// **Discard** proprioception waypoints ($w_m^{\tau_k}$ and $w_n^\zeta$), only observation inputs are used for $f_\psi$

**if** $R <$ length($\zeta$) **then**

  $\pi = \{f_\psi, \{a_n^\zeta\}_{n=R}^N\}$ // policy consists of an end-to-end neural net + demo replay (if an environment disturbance stopped data collection before the last demo waypoint)

**else**

  $\pi = \{f_\psi\}$ // policy consists only of an end-to-end neural net

**end if**

**Output:** $\pi$

**Algorithm 8:** Deploy

**Input:** Policy $\pi$, final data collection timestep $R$, single demonstration $\zeta$

Capture observation $o$ // comprising RGB wrist cam image + force-torque feedback

Action $a = f_\psi(o)$

Perform action $a$

**while** $a$ is not the identity transformation **do**

  Capture observation $o$

  Action $a = f_\psi(o)$

  Perform action $a$

**end while**

// if an environment disturbance stopped data collection before the last demo waypoint

**if** $R <$ length($\zeta$) **then**

  Replay remaining demo $\{a_n^\zeta\}_{n=R}^N$

**end if**

# References


[1] A. Brohan et al. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

[2] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, 2023.

[3] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[4] S. Haldar, J. Pari, A. Rai, and L. Pinto. Teach a robot to fish: Versatile imitation from one minute of demonstrations. *arXiv preprint arXiv:2303.01497*, 2023.

[5] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Conference on Neural Information Processing Systems*, page 4572–4580, 2016.

[6] V. Mnih et al. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[7] O. X.-E. Collaboration et al. Open X-Embodiment: Robotic learning datasets and RT-X models, 2023.

[8] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, 2021.

[9] Y. Hu, M. Cui, J. Duan, W. Liu, D. Huang, A. Knoll, and G. Chen. Model predictive optimization for imitation learning from demonstrations. *Robotics and Autonomous Systems*, 163, 2023.

[10] Y. Huang, J. Silvério, L. Rozo, and D. G. Caldwell. Generalized task-parameterized skill learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5667–5474, 2018. doi: 10.1109/ICRA.2018.8461079.

[11] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *ArXiv*, abs/1709.04905, 2017.

[12] Z. Mandi, F. Liu, K. Lee, and P. Abbeel. Towards more generalizable one-shot visual imitation learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2434–2444, 2022. doi: 10.1109/ICRA46639.2022.9812450.

[13] G. Papagiannis and Y. Li. Imitation learning with sinkhorn distances. In *European Conference in Machine Learning and Knowledge Discovery in Databases 2022*, 2022.

[14] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. Aparicio Ojea, E. Solowjow, and S. Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[15] E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[16] E. Valassakis et al. Demonstrate once, imitate immediately (dome): Learning visual servoing for one-shot imitation learning. 2022.

[17] N. D. Palo and E. Johns. On the effectiveness of retrieval, alignment, and replay in manipulation. *RA-Letters*, 2024.

[18] P. Vitiello, K. Dreczkowski, and E. Johns. One-shot imitation learning: A pose estimation perspective. In *Conference on Robot Learning*, 2023.

[19] B. Wen, W. Lian, K. E. Bekris, and S. Schaal. You only demonstrate once: Category-level manipulation from single visual demonstration. *ArXiv*, abs/2201.12716, 2022.

[20] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, 2017.

[21] L. Ke, J. Wang, T. Bhattacharjee, B. Boots, and S. Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. In *International Conference on Robotics and Automation (ICRA)*, 2021.

[22] A. Zhou, M. J. Kim, L. Wang, P. Florence, and C. Finn. Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis, 2023.

[23] M. Jia, D. Wang, G. Su, D. Klee, X. Zhu, R. Walters, and R. Platt. Seil: Simulation-augmented equivariant imitation learning. In *International Conference on Robotics and Automation (ICRA)*, pages 1845–1851, 2023. doi:10.1109/ICRA48891.2023.10161252.

[24] L. Ke, Y. Zhang, A. Deshpande, S. Srinivasa, and A. Gupta. CCIL: Continuity-based data augmentation for corrective imitation learning. In *First Workshop on Out-of-Distribution Generalization in Robotics at CoRL 2023*, 2023.

[25] G. Cideron, B. Tabanpour, S. Curi, S. Girgin, L. Hussenot, G. Dulac-Arnold, M. Geist, O. Pietquin, and R. Dadashi. Get back here: Robust imitation by return-to-distribution planning, 2023.

[26] M. Caron, H. Touvron, I. Misra, H. J'egou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. *International Conference on Computer Vision (ICCV)*, 2021.

[27] S. Amir et al. Deep vit features as dense visual descriptors. *ECCVW What is Motion For?*, 2022.

[28] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[29] J. Luo, O. O. Sushkov, R. Pevceviciute, W. Lian, C. Su, M. Vecerík, N. Ye, S. Schaal, and J. Scholz. Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study. *ArXiv*, abs/2103.11512, 2021.

[30] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, 2024.

[31] T. Z. Zhao, J. Luo, O. Sushkov, R. Pevceviciute, N. Heess, J. Scholz, S. Schaal, and S. Levine. Offline meta-reinforcement learning for industrial insertion. In *International Conference on Robotics and Automation (ICRA)*, pages 6386–6393, 2022.

[32] A. Nair, B. Zhu, G. Narayanan, E. Solowjow, and S. Levine. Learning on the job: Self-rewarding offline-to-online finetuning for industrial insertion of novel connectors from vision. In *International Conference on Robotics and Automation (ICRA)*, pages 7154–7161, 2023.

[33] K. Kimble, K. Van Wyk, J. Falco, E. Messina, Y. Sun, M. Shibata, W. Uemura, and Y. Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE Robotics and Automation Letters*, 5(2):883–889, 2020.

[34] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ArXiv*, abs/1509.02971, 2015.

[36] N. Di Palo and E. Johns. Learning multi-stage tasks with one demonstration via self-replay. In *Conference on Robot Learning (CoRL)*, 2021.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pages 770–778. IEEE, June 2016.

[38] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

[39] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *CoRR*, abs/1909.12271, 2019. URL http://arxiv.org/abs/1909.12271.

[40] J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto. The surprising effectiveness of representation learning for visual imitation. *CoRR*, abs/2112.01511, 2021. URL https://arxiv.org/abs/2112.01511.