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

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

A MILES: Additional Details on the Method

A.1 Related Work

As follows, we ground our work relative to methods that can learn manipulation skills from a single demonstration, unlike most approaches that require large demonstration datasets [1, 18, 19].

Imitation learning from prior knowledge. An effective way to compensate for the lack of large demonstration datasets is to leverage prior task knowledge such as access to ground truth object poses [20, 21] or by meta-learning policies by first pretraining on large demonstration datasets [22, 23]. However, precise knowledge of the objects' poses is hard to obtain in practice and meta-learning methods are often limited to tasks similar to the ones seen in the demonstrations. Instead, MILES can learn a new task from just a single demonstration without any prior object or task knowledge.

Imitation learning via Reinforcement learning (RL). Inverse RL methods from a single demonstration learn to follow that demonstration by minimizing a similarity metric between the trajectories of the learned policy and the demonstration [3, 24, 4, 25]. Other RL methods that learn from demonstrations infer rewards through alternative means, like goal images [8]. Though effective, these methods are often inefficient as they rely on random exploration and repeated environment resets which require significant human effort. Instead, our self-supervised data collection makes MILES highly efficient and eliminates the need for repeated environment resetting.

Imitation learning via pose estimation and demonstration replay. Replay-based imitation learning methods first estimate and move the robot to a similar pose relative to the objects of interest as in the demonstration and then replay the demonstrated robot actions [7, 26, 6, 17, 27]. While these methods are the most efficient in terms of human time, small errors in pose estimation cause errors to compound during demonstration replay, leading to task failures [2]. And even under the assumption of perfect pose estimation, potential environment collisions may prevent the robot from reaching the desired pose or may perturb the objects such that replaying the demonstration fails to complete the task. Instead, MILES' self-supervised data collection procedure retains the human-time efficiency of pose estimation methods, while learning to avoid unnecessary collisions, and minimizing or completely eliminating open-loop replay errors depending on the task.

Imitation learning by demonstration augmentation. Demonstration augmentation approaches like DAgger [5] and DART [28] mitigate covariate shift by relying on laborious interactive expert queries to expand the known state distribution of a policy. And methods that do not require an interactive expert still rely on multiple demonstrations or task-specific optimizations [29, 30, 31, 32, 33] which limit their practical application. Instead, MILES is a fully autonomous method that uses self-supervision to augment a single demonstration and can learn a wide range of diverse tasks.

A.2 Method Pseudocode

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

A.3 Validity Conditions for Augmentation Trajectories

As follows, we introduce two conditions that determine whether an augmentation trajectory can be fused with the human demonstration. Consider an augmentation trajectory τ_k aimed at returning the robot to the k_{th} demonstration state, (w_k^ζ, o_k^ζ):

(1) **Condition 1, Reachability:** After executing the augmentation trajectory, the EE's pose must equal the pose of the demonstration waypoint w_k^ζ. This equality can be verified trivially using proprioception. In many scenarios, the environment's dynamics (e.g. collisions) or inevitable systematic inaccuracies in a robot's controller may prevent it from reaching its target waypoint w_k^ζ.

Thus, if $w_M^{\tau_k} \neq w_k^{\zeta}$, the augmentation trajectory cannot return to demonstration state k, rendering the augmentation trajectory invalid.

(2) **Condition 2, Environment Disturbance:** While collecting τ_k, the robot may disturb the environment, resulting in a final observation $o_M^{\tau_k}$ that no longer matches that of the demonstration (even if w_k^{ζ} is reached). For instance, during data collection if the robot's gripper pushes an object to a different pose than it had at timestep k of the demonstration, the final observation in the augmentation trajectory will differ from the demonstration's k_{th} observation. Therefore, if $o_M^{\tau_k} \neq o_k^{\zeta}$, the augmentation trajectory cannot be combined with the human demonstration to create a new, valid demonstration. To detect such disturbances, we compare the cosine similarity of the DINO features [34, 35] of the RGB image I_k^{ζ} from the demonstration's observation o_k^{ζ} and the image $I_M^{\tau_k}$ after executing the augmentation trajectory. If the similarity falls below a threshold θ, we assume the environment has been disturbed and stop data collection.

A.3.1 How do we check for the Reachability condition?

Reachability. To check for reachability, after executing an augmentation trajectory τ_k, we verify whether the final achieved pose matches the pose of the k_{th} demonstration waypoint using proprioception, as described in section A.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^{ζ} is unreachable during data collection, we cannot automatically determine how to reach w_k^{ζ} 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.2 (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.2 (a, right).

A.3.2 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 A.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.2 (b). The comparison between the two RGB images relies on the similarity of their DINO features [34]. Specifically, we use a pre-trained DINO ViT [34] to obtain the DINO features for different patches of each image similarly to [35]. By computing the cosine similarity between the DINO features of each corresponding image patch in I_k^{ζ} and $I_M^{\tau_k}$, we can calculate the average similarity between the two images [35]. If the similarity is below a threshold θ (to see how we automatically determine θ please see section B.3.3), 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.2 (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 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.

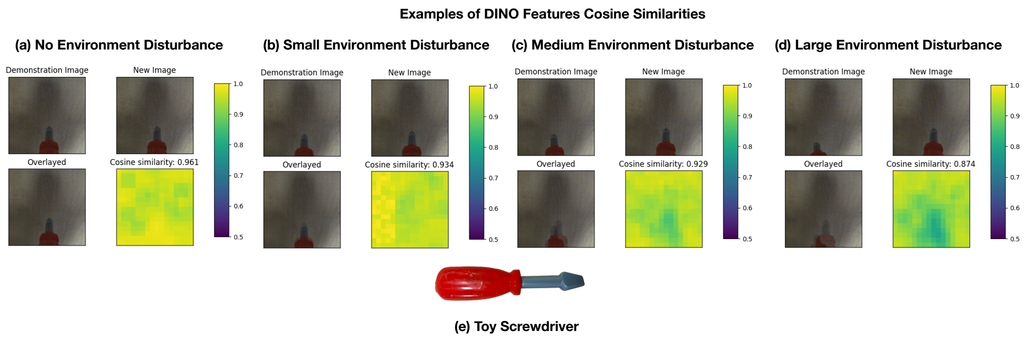

Figure A.1: The cosine similarity computed using the DINO features for the screwdriver task under varying environment disturbances.

A.4 Additional Results on Environment Disturbances and DINO Features

We demonstrate in this section several examples of possible environment disturbances and how we can detect them using the DINO features on the toy screwdriver used in our experiments. We use the screwdriver as an example as during data collection for the ”Twist screw” task, data collection was stopped due to an environment disturbance caused at the grasped screwdriver. Additionally, disturbances caused in the grapsed objects are often the most subtle, and as such make for the most interesting cases.

Figure A.1 (e) shows the screwdriver object (not grasped). All the other figures depict the screwdriver as it appears in the view of the wrist camera when grasped by the robot. Figure A.1 (a) shows a “Demonstration Image” and a “New Image” that depicts the DINO Cosine similarity (higher better) when no environment disturbance has occurred, i.e., the grasp has not changed. The heatmap demonstrates the similarity between each corresponding patch between the “Demonstration Image” and the “New Image” (the cosine similarity reported is the mean of these). As shown, the cosine similarity (0.961) is greater than our universal threshold θ of 0.94 (for more details please see experiments section 3). The reason it is not a perfect 1.0 is due to noise and light changes as the photos were captured at different moments in time. Figure A.1 (b), shows a detected environment disturbance based on the DINO features. As shown under the ”New Image” the screwdriver has moved by a small amount in the gripper and the cosine similarity falls slightly below our threshold θ. Then, Figure A.1 (c) shows a slightly bigger detected environment disturbance, and finally Figure A.1 (d) shows a rather large environment disturbance. Generally, as shown in Figure A.1, the DINO features are robust in detecting environment disturbances of different scales and as we move from smaller to larger disturbances in the grasped screwdriver the cosine similarity also decreases, as expected.

A.5 MILES’ Policy

A.6 Policy Overview

Training. We train a separate policy π for each task as an LSTM network with behavioral cloning that receives as input the RGB and force-torque observations in the dataset $\mathcal{D}_{new}$ and regresses the corresponding actions. Note that $\mathcal{D}_{new}$ does not contain proprioception data, allowing our policies to generalize to different object poses naturally due to the use of our wrist camera.

Inference. We deploy our policy π to solve a task up to the R_{th} demonstrated state. If no environment disturbance occurred during data collection for that task, then the R_{th} state is the final state in the demonstration and π solves the task completely in a closed-loop manner. Otherwise, after π completes the task up to the R_{th} state, the remaining demonstrated action segment $\zeta_{remaining}$ *is replayed*. We provide more details regarding how we deploy our policy, the network architecture, and how we detect that π has reached the R_{th} below.

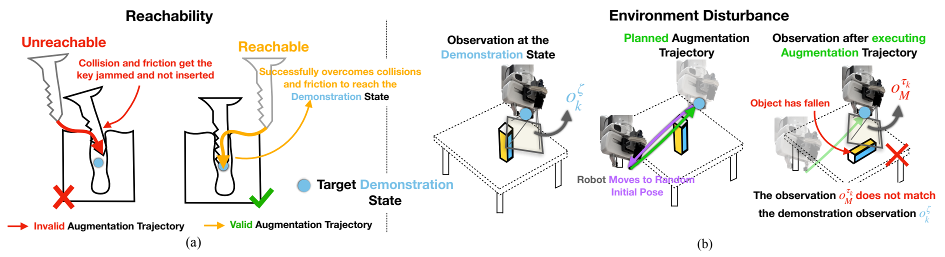
Reachability
Unreachable
Collision and friction get the key jammed and not inserted
Reachable
Successfully overcomes collisions and friction to reach the Demonstration State
Target Demonstration State
Invalid Augmentation Trajectory
Valid Augmentation Trajectory
(a)
Environment Disturbance
Observation at the Demonstration State
o_k^ζ
Planned Augmentation Trajectory
Robot Moves to Random Initial Pose
Observation after executing Augmentation Trajectory
Object has fallen
$o_M^{\tau_k}$
The observation $o_M^{\tau_k}$ does not match the demonstration observation o_k^ζ
(b)

Figure A.2: **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.

A.6.1 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 π that comprises a single neural network f_ψ, parameterized by ψ, that receives as input an RGB image captured from the wrist camera and force-torque feedback to predict 6-DoF actions: $f_\psi : \mathbb{R}^{H \times W \times 3} \times \mathbb{R}^6 \rightarrow \mathrm{SE}(3)$ 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_ψ and all actions are predicted relative to the EE's frame. f_ψ consists of a ResNet-18 backbone [36] 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 [37] network for action prediction. The network is trained using standard behavior cloning to maximize the likelihood of $\mathcal{D}_{new}$.

A.6.2 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 2.2). In this scenario, if our policy consists only of f_ψ, then during task execution the robot would be able to solve the task only up to the R_{th} state, but not complete it. As such, we define our policy π to consist of two components: (1) the first component is a neural network f_ψ identical to the above scenario, but trained up to the R_{th} state and (2) the second component corresponds simply to the sequence of the remaining demonstration actions from the R_{th} state onwards, for which no self-supervised data was collected, i.e., $\zeta_{remaining} = \{a_n^\zeta\}_{n=R}^N$.

A.6.3 How do we deploy MILES' policy?

Deployment: Our LSTM-based policy closely follows the implementation of BC-RNN [38]. 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_ψ is used to complete the task equivalently to policies trained using reinforcement learning or behavioral cloning.

If data collection was interrupted, first f_ψ is deployed to solve the task up to the R_{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.6.4.

During deployment we reset the hidden state of the LSTM at an interval equal to two times the number of timesteps (i.e., waypoints) in the demonstration for which augmentation trajectories were collected. For example, if for a task MILES collected augmentation trajectories for 40 demonstration waypoints before stopping due to an environment disturbance, then, during deployment the hidden state of the LSTM is reset every 80 timesteps. We did not find the frequency of resetting the hidden memory to have significant effects on the policy's performance. We would like to note that the only important observation we made was that the number of timesteps should not be very low (e.g., 5) as then the robot would end up progressing towards completing a task very slowly.

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

A.6.4 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_{th} state has been reached. Hence, we deploy f_ψ 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 Implementation Details

For our experiments, we use a FLIR camera mounted to the wrist of Franka Emika Robot. We sample $Z = 10$ augmentation trajectories for each demonstration waypoint ($\approx$ approximately 1 minute of data collection per waypoint). This number is set arbitrarily, but as we show later in our ablations, some tasks may require less data. We collect augmentation trajectories with initial poses near the demonstration in the range of 4cm and 4 degrees around each demonstration waypoint, a wider range compared to existing augmentation methods that learn from multiple demonstrations [29, 30]. As commonly done in the literature [3, 8, 7, 17], we provide our demonstrations starting near each object. At deployment, to reach the object from far away we first estimate the object's pose using pose estimation and approach it before switching to MILES. Finally, we set the environment disturbance threshold θ to 0.94 for all our tasks. Additional details on the pose estimation method we use and how to set each one of MILES' parameters can be found below.

B.2 Pose Estimation

In practice, as with most methods [3, 8, 7, 17], we naturally provide the demonstrations starting near the task-relevant object to focus self-supervised data collection at the part of the task that is the most important, that is the robot-object interaction part.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 is possible – as MILES makes no assumptions or restrictions on the length of the demonstration– it may be inconvenient. As such, inspired by [2, 26, 17] 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 [7] 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 webpage: https://sites.google.com/view/miles-imitation.

B.3 MILES Data Collection Hyperparameters

B.3.1 How do we set the data collection range around each demonstration waypoint?

As discussed in our experiments section 3, 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.2 which we obtained based on [7].

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 2: 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.

B.3.2 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 C.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 ”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.3.3 How do we determine the Environment Disturbance threshold θ ?

We determined θ 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.4 Task Descriptions

A detailed description of each task along with their Data Collection Times (**DCT**) can be found in Table 2.

B.4.1 How long is each demonstration?

The demonstration lengths varied across each task. As follows, we list for each task the number of demonstration waypoints comprising each human demonstration (each demonstration waypoint can be interpreted as a timestep): Lock with key: 32, USB task: 20, Plug into socket: 40, Insert power cable: 29, Twist screw: 47, Bread in Toaster: 70, Open lid: 80. All demonstrations were collected using teleoperation. Note that the number of demonstration waypoints is not necessarily equal to the number of waypoints for which MILES collected augmentation trajectories. This is because environment disturbances may have caused the data collection to stop earlier.

B.4.2 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

Methods	Insert Onto Square Peg	Lightbulb In	Pick Up Cup	Turn Tap	Lamp On	Mean
Demo Replay	0	0	5	5	0	2
Reset Free Residual RL	0	0	0	0	0	0
Reset Free FISH (Inverse Residual RL)	0	0	0	0	20	4
Pose Estimation + Demo Replay	70	65	90	**80**	95	80
MILES	**90**	**75**	**100**	75	**100**	**88**

Table 3: Task success rates (%) of each method on RLBench.

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.6.4. 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.5 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 [7], 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. This baseline estimates and moves the robot to a pose relative to the object of interest as depicted in the first state in the demonstration and replays the complete demonstration. We chose this baseline compared to alternatives, as it leverages task-specific data allowing it to achieve very precise pose estimation.

Reset-Free FISH [3]. 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.6 Details on the Evaluation Setup

For a fair evaluation, we carefully tuned each method's hyperparameters. Additionally, each learning-based baseline collected the same number of observations as MILES during data collection for each task. We evaluated each method's success rate across 20 trials. For each trial we randomized the relative starting pose of the robot and the task-relevant object equivalently across all methods within a sphere of 20cm around the object as long as the object was visible to the camera. Finally, we emphasize that for all evaluations both MILES and the baselines predict *6-DoF actions*.

C Additional Experiment Results

C.1 Simulation Results

To aid other researchers in reproducing our results, we conducted additional simulation experiments on the RLBench benchmark [39] on 5 tasks, specifically: 1) 'Insert Onto Square Peg', 2) 'Lightbulb In', 3) 'Pick Up Cup', 4) 'Turn Tap' and 5) 'Lamp On'. We performed an identical evaluation to our real-world experiments where we performed 20 evaluation trials for each method. Additionally, we used the images captured only from the wrist camera in RLBench. During training we allowed each method to collect the same amount of data and **we did not perform any environment resets during training/data collection for any methods**. The results can be seen in Table 3. As shown, MILES significantly outperforms the baselines, while the relative performance when comparing all methods remained relatively unchanged compared to our real-world results.

Similarly to our real-world experiments, the reinforcement learning baselines obtained poor performance for reasons in line with the ones discussed in our experiments section. Specifically, during

Method Ablations	Lock with key	Insert USB	Plug into socket	Insert power cable	Twist screw	Bread in toaster	Open lid	Mean
No Sequence	60	20	20	10	0	85	95	43
No Environment Disturbance	90	70	85	85	0	0	0	47
No Reachability	75	40	95	20	85	95	**100**	73
No Memory	50	65	**100**	75	35	90	**100**	74
MILES	**90**	**70**	85	**85**	**85**	95	**100**	**87**

Table 4: Task success rates (%) for 20 trials reported for each method ablation.

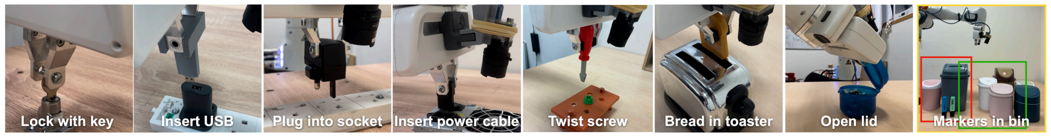

Figure C.3: The tasks used in our experiments. The "Markers in Bin" is used to evaluate MILES' ability to generalize (the bins marked green denote the training set, while the red denote the test set).

training we observed that for the tasks 'Insert Onto Square Peg' and 'Lightbulb In' a random gripper action drops the grasped object during exploration and the policy never manages to grasp it again during training without a reset in the given training time. For the 'Pick Up Cup' task, the reinforcement learning policy knocks the cup off the table during exploration, consequently never learning something useful. For the 'Turn Tap' task the RL policies never learned to properly grasp and rotate the handle and for the 'Lamp On' task, only Reset Free FISH managed to learn a policy that obtains 20% success rate in the given training time. As discussed in our real-world experiments, if instead we had allowed environment resets and more training time that would have resulted in significantly higher success rates for the RL baselines, compared to their current performance.

C.2 How does MILES perform under different method ablations?

This section studies MILES' performance by ablating 4 different components of the method: (1) **No Environment Disturbance:** we ablate the environment disturbance condition by not checking for that condition when collecting augmentation trajectories. (2) **No reachability:** we ablate the reachability condition by relabeling each observation's action (of the existing MILES data), to move the robot to the nearest waypoint in the demonstration based on their Euclidean distance. If the constraint for reachability is not important, then simply moving from each pose to the nearest waypoint in the demonstration in a straight line would be sufficient to solve a task. (3) **No sequence:** we recollect MILES' data but instead of collecting Z augmentation trajectories for the first demonstration state, then progressing to the second state and so on, we collect data *without* following the demonstration's waypoint sequence and instead follow a random one. (4) **No Memory:** For this ablation we retrain a network on the existing MILES data that does not account for history.

Results. Table 4 shows MILES performance after ablating each component. Collecting augmentation trajectories for each demonstration state in a random order (**No Sequence**), with an average success rate of 43%. Additionally, not checking for the environment disturbance condition (**No Environment Disturbance**) appears to cause significant performance degradation for the tasks where an environment disturbance occurred during data collection, corresponding mostly to the non-contact rich tasks. On the other hand, not checking for the reachability condition (**No Reachability**) also lowers performance, particularly for the precise, contact-rich tasks, indicating that the reachability condition is the most important when learning tasks requiring precise manipulation. Finally, the lower performance obtained by removing the LSTM (**No Memory**) demonstrates the performance benefits of training memory-based networks on datasets collected using MILES.

C.3 How important are vision and force modalities to the performance of MILES?

In this section, we ablate the use of vision and force feedback as policy inputs for the four contact-rich tasks from our earlier experiments. We retrain and evaluate two policies: one using only vision and one using only force. The results, shown in Figure C.4, indicate that the vision-based policy improves MILES' performance in the "Insert USB" and "Plug into socket" tasks but reduces performance in the other two tasks. This suggests that force feedback might not consistently benefit MILES, possibly due to its noisy signal which makes it hard to distinguish between different environment states.

The force-based policy, however, fails almost completely. This is expected as force feedback is zero in free space and can be ambiguous due to symmetries in object surfaces. Overall, while force feedback aids performance in some tasks, it is not always necessary. Vision remains the most crucial modality to MILES' high performance.

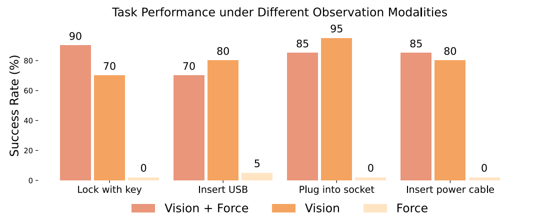

Figure C.4: MILES' performance when trained only on either vision or force feedback or both.

C.4 How does MILES perform under different sizes of self-supervised data?

In this section, we ablate the dataset size used to learn four tasks by splitting their original datasets into chunks containing 75%, 50%, and 25% of the original data. We evaluated the best and worst performing contact-rich tasks ("Lock with key" and "Insert USB") and non-contact-rich tasks ("Open lid" and "Twist screw"). Data collection times for each task can be found in the supplementary material. Figure C.5 shows that for high tolerance tasks like "Open lid," MILES achieves a 100% success rate even with 25% of the data, corresponding to *only* 8 minutes of data collection. However, for precise tasks, success rates decrease as dataset size is reduced. Notably, for "Lock with key" and "Twist screw," reducing the dataset to 50% results in a high failure rate. To summarize, we observe that high tolerance tasks are likely to require less data, and in practice only a few minutes of data collection time. Instead, for high-precision tasks, like inserting a USB, the dataset size appears to impact MILES' performance significantly.

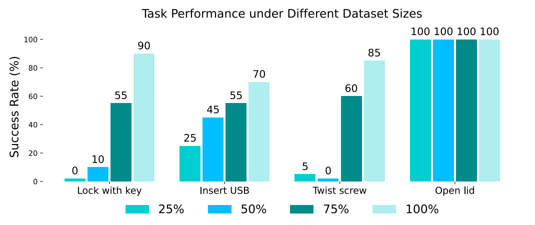

Figure C.5: MILES' performance when trained on different dataset sizes. 100% corresponds to the original dataset. 75%, 50%, and 25% correspond to splits of the original dataset.

C.5 Experiment Results on Generalization Performance

Since MILES uses BC to train policies, existing generalization results for BC [19, 1] also apply to MILES. For tasks that include demonstration replay following the closed-loop policy, MILES can generalize to new objects by retrieving the replay trajectory of the most similar object in the existing demonstrations, similar to prior work [6]. To test this, we tasked MILES with throwing markers of different colors into differently shaped and colored bins, shown in Figure C.3 (8). Trained on five bins (marked green) and tested on two new bins (marked red), MILES achieved an 80% success rate on the pink bin and 60% on the gray bin, over 10 trials each starting from poses where simple demonstration replay would fail. 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 [6]. Videos exhibiting MILES generalization on the two test case bins can be found on our webpage: https://sites.google.com/view/miles-imitation.

C.6 Experiment Results on Multi-stage Tasks

To evaluate MILES' ability to solve multi-stage tasks, we tasked MILES with picking up the plastic bread shown in Figure 1 (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.2 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.

C.7 MILES' 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 webpage: https://sites.google.com/view/miles-imitation.

C.8 What if MILES stops data collection early due to a detected environment disturbance?

There is no requirement as to how early MILES may stop data collection due to an environment disturbance, as long as it has collected sufficient augmentation trajectories for at least the first demonstration waypoint. During data collection, MILES can effectively learn a policy even if an environment disturbance occurs early. Unlike RL, MILES learns to solve the task closed-loop up to the demonstration waypoint where the disturbance was detected, after which it replays the demonstration. This is because MILES collects data progressively for each demonstration waypoint, rather than rolling out a policy all at once like RL. Consequently, during data collection, if a disturbance occurs as early as (for example) near the 2nd waypoint, MILES will still know how to get to the 1st waypoint during deployment, where it will replay the demonstration.

Overall, MILES can handle early environment resets during data collection. While as with the majority of learning-based methods, the more the data the better the performance, as such the later an environment disturbance occurs in the data collection process the better. However, MILES can still learn a robust policy as long as sufficient data has been collected at least for the 1st demonstration waypoint. This is typically trivial as most human demonstrations naturally begin by controlling the robot in free-space far from the object of interest, before interacting with it.

C.9 Discussion

Limitations. We now highlight some important limitations of our method. Firstly, MILES' reliance on a wrist camera enables MILES to obtain spatial generalization, however, simultaneously this limits its field of view and its applicability to larger task spaces. Future work could address this by incorporating an external camera to initially approach an object before switching to the wrist camera, similarly to [17]. Secondly, while MILES is robust to distractors at deployment before data collection begins it requires a human to set up the robot's workspace such that only the task-relevant object is in camera view for the policy to achieve spatial generalization. While this requires only a few seconds of human time, future work could address this by extending MILES to incorporate segmentation methods, similar to [17, 41], that segment the task-relevant object in the dataset. Similarly, to address any unwanted collisions that MILES could cause in the presence of multiple objects, future work could study incorporating an external camera during self-supervised data collection to plan and collect collision-free augmentation trajectories. Thirdly, our current implementation of MILES trains a separate policy for each task and hence it is unclear how well MILES would generalize to completely new tasks. In future work, we aim to study this by training a single monolithic policy on MILES' self-supervised data combined with replay-trajectory retrieval [42].

Conclusion. We introduced MILES, a framework that makes imitation learning easy. MILES requires only a single demonstration and collects self-supervised data that demonstrate to the robot how to return to and then follow that demonstration. Subsequently, this enabled us to obtain manipulation skills comprising either (1) a single end-to-end policy trained with behavioral cloning or (2) a combination of an end-to-end policy and demonstration replay. Our real-world experiments showed that self-supervised data enable the acquisition of manipulation skills that achieve considerably improved performance compared to several state-of-the-art baselines on many everyday tasks ranging from learning to open a lid to using a key to lock a lock or inserting a USB into a port, both of which require complex and precise contact-rich manipulation.

D 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: $\theta = 0.94$)
1:  $\mathcal{D} = \{\}$ // init empty dataset of augmentation trajectories
2:  Reachable = True // init variable that tracks reachability
3:  Disturbance = True // init variable that tracks environment disturbances
4:  $R = 1$ // init variable that stores the timestep when self-supervised data collection stops
5:  Move robot to the initial demonstration pose $w_1^\zeta$
6:  for iteration $k = 1$ to $N$ do
7:      $j = 1$ // init variable that tracks the number of collected augmentation trajectories per demo waypoint
8:      while $j \leq Z$ do
9:          $\tau_k$ ←SampleTrajectory($w_k^\zeta$) (Alg. 2)
10:         Reachable ←CheckReachability($w_k^\zeta$) (Alg. 3)
11:         if Reachable is False then
12:             ReturnToDemoWaypoint($k$, $\zeta$) (Alg. 4)
13:             Break // exit while loop
14:         end if
15:         $I_M^{\tau_k}$ ← Capture RGB wrist-cam image // $M$ is the $M_{th}$ (final) timestep of $\tau_k$
16:         Disturbance ← CheckEnvDisturbance($o_k^\zeta$, $I_M^{\tau_k}$, $\theta$) (Alg. 5)
17:         if Disturbance is True then
18:             $R = k$ // store timestep when data collection stops
19:             Break // exit while loop
20:         end if
21:         $\mathcal{D} = \mathcal{D} \cup \tau_k$ // add augmentation trajectory to dataset
22:         $j = j + 1$
23:     end while
24:     if Disturbance is True then
25:         Break // exit for loop
26:     end if
27:     Proceed to the next demonstration state by performing action $a_k^\zeta$ // follow the demonstration's progression
28: end for
29: $\mathcal{D}_{\text{new}}$ ←FuseAugmentationsWithDemo($\mathcal{D}$, $R$, $\zeta$)(Alg. 6)
30: $\pi$ ←TrainPolicy($\mathcal{D}_{\text{new}}$, $R$, $\zeta$)(Alg. 7)
31: Deploy($\pi$, $R$, $\zeta$)(Alg. 8)
Output: $\pi$
```

Algorithm 2: SampleTrajectory

```
Input: Demonstration waypoint $w_k^\zeta$
1: $\tau_k = \{\}$ // init empty augmentation trajectory
2: Sample initial pose $w_1^{\tau_k}$ and move robot (Optional: record trajectory poses)
3: 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).
4: $m = 1$ // observations, actions index
5: while moving to $w_k^\zeta$ do
6:     $\tau_k = \tau_k \cup (w_m^{\tau_k}, o_m^{\tau_k}, a_m^{\tau_k})$ // 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.
7:     ($o_m^{\tau_k}$ comprises wrist cam RGB images + force-torque readings)
8: end while
Output: Return augmentation trajectory $\tau_k$
```

Algorithm 3: CheckReachability

```
Input: Demonstration waypoint $w_k^\zeta$
1: Reachable ← True // init reachability variable
2: $w_M^{\tau_k}$ ← EE pose // achieved after executing the augmentation trajectory (comprising $M$ timesteps); read from proprioception
3: 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 $\zeta$
1: Move to initial demonstration waypoint $w_1^\zeta \in \zeta$
   // replay demonstration up to the $k_{\text{th}}$ timestep
2: for iteration $t = 1$ to $t = k$ do
3:     Perform action $a_t^\zeta \in \zeta$
4: end for
```

Algorithm 5: CheckEnvDisturbance

```
Input: Demonstration observation $o_k^\zeta$, captured live image $I_M^{\tau_k}$, similarity threshold $\theta$
1: Disturbance ← False // init environment disturbance variable
2: $I_k^\zeta \in o_k^\zeta$ // retrieve RGB image $I_k^\zeta$ from the demonstration's observations
3: $[f^1_{I_k^\zeta}, f^2_{I_k^\zeta}, ...]$ ←DINO-ViT($I_k^\zeta$) // compute DINO-ViT features [35, 34] for each image patch $f^x_{I_k^\zeta}$ for the demo waypoint image
4: $[f^1_{I_M^{\tau_k}}, f^2_{I_M^{\tau_k}}, ...]$ ←DINO-ViT($I_M^{\tau_k}$) // compute DINO-ViT features [35, 34] for each image patch $f^x_{I_M^{\tau_k}}$ from the current live environment image (captured after executing the augmentation trajectory).
5: $sim$ =AvgCosineSimilarity($[f^1_{I_k^\zeta}, f^2_{I_k^\zeta}, ...], [f^1_{I_M^{\tau_k}}, f^2_{I_M^{\tau_k}}, ...]$)
6: if $sim < \theta$ then
7:     Disturbance ← True
8: end if
Output: Disturbance
```

Algorithm 6: FuseAugmentationsWithDemo

```
Input: Dataset of augmentation trajectories $\mathcal{D}$, final data collection time step $R$, single demonstration $\zeta$
1: $\mathcal{D}_{\text{new}} = \{\}$// init empty dataset to store fused trajectories
2: for $\tau_k$ in $\mathcal{D}$ do
3:     $\zeta_{\text{segment}} = \underbrace{\{(w_n^\zeta, o_n^\zeta, a_n^\zeta)\}_{n=k}^R}_{\text{demonstration segment from } k_{\text{th}} \text{ demo waypoint to } R_{\text{th}}} \in \zeta$
4:     $\tau_{k_{\text{new}}} := \tau_k \cup \zeta_{\text{segment}}$
5:     $\mathcal{D}_{\text{new}} = \mathcal{D}_{\text{new}} \cup \tau_{k_{\text{new}}}$
6: 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 $\zeta$
1: 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 $f_\psi$
2: if $R <$ length($\zeta$) then
3:     $\pi = \{f_\psi, \{a_n^\zeta\}_{n=R}^N\}$ // policy consists of an end-to-end neural net + demo replay (if an environment disturbance stopped data collection before the last demo waypoint)
4: else
5:     $\pi = \{f_\psi\}$ // policy consists only of an end-to-end neural net
6: end if
Output: $\pi$
```

Algorithm 8: Deploy

```
Input: Policy $\pi$, final data collection timestep $R$, single demonstration $\zeta$
1: Capture observation $o$ // comprising RGB wrist cam image + force-torque feedback
2: Action $a = f_\psi(o)$
3: Perform action $a$
4: while $a$ is not the identity transformation do
5:     Capture observation $o$
6:     Action $a = f_\psi(o)$
7:     Perform action $a$
8: end while
   // if an environment disturbance stopped data collection before the last demo waypoint
9: if $R <$ length($\zeta$) then
10:    Replay remaining demo $\{a_n^\zeta\}_{n=R}^N$
11: 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. 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.

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

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] W. Sun, A. Vemula, B. Boots, and D. Bagnell. Provably efficient imitation learning from observation alone. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6036–6045. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/sun19b.html`.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

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

[38] A. Mandlekar et al. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, 2021.

[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`.

[41] D. Seita, Y. Wang, S. J. Shetty, E. Y. Li, Z. Erickson, and D. Held. Toolflownet: Robotic manipulation with tools via predicting tool flow from point clouds. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1038–1049. PMLR, 14–18 Dec 2023. URL `https://proceedings.mlr.press/v205/seita23a.html`.

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