2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning Skills to Patch Plans Based on Inaccurate Models

Alex Lagrassa[1], Steven Lee[1], and Oliver Kroemer[1]

***Abstract*— Planners using accurate models can be effective for accomplishing manipulation tasks in the real world, but are typically highly specialized and require significant fine-tuning to be reliable. Meanwhile, learning is useful for adaptation, but can require a substantial amount of data collection. In this paper, we propose a method that improves the efficiency of sub-optimal planners with approximate but simple and fast models by switching to a model-free policy when unexpected transitions are observed. Unlike previous work, our method specifically addresses when the planner fails due to transition model error by patching with a local policy only where needed. First, we use a sub-optimal model-based planner to perform a task until model failure is detected. Next, we learn a local model-free policy from expert demonstrations to complete the task in regions where the model failed. To show the efficacy of our method, we perform experiments with a shape insertion puzzle and compare our results to both pure planning and imitation learning approaches. We then apply our method to a door opening task. Our experiments demonstrate that our patch-enhanced planner performs more reliably than pure planning and with lower overall sample complexity than pure imitation learning.**

I. Introduction

The ability for robots to adapt to changing needs and conditions in human environments is necessary for expanding their utility into new application domains. A robot can be pre-programmed with general models and reasoning capabilities before deployment, but some amount of adaptation is necessary to capture the wide range of conditions a robot may encounter.

Motion planners with accurate models are often used to accomplish tasks, but are often highly specialized and require significant fine-tuning to be reliable [1], [2]. Inaccuracies or deviations in a system's model can increase the complexity of the controller, and the potential for failed task executions. Contact-rich manipulation tasks are difficult to model because of the intricacies of changing contact modes [3]. Nonetheless, planners are still useful when the robot maintains contact, as modeling some phenomena, such as friction on a sliding surface, can be sufficiently accurate to provide the planner with useful information [4], [5]. However, more complex interactions may lead to the model, and hence the planner, failing at execution time.

Learning-based approaches allow robots to adjust executions through experience, whether through supervised demonstrations or reinforcement learning. However, the amount of data required to acquire learned skills that generalize can be both large and often non-trivial to collect [6], [7], [8].

[1]Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA {alagrass, stevenl3, okroemer}@cs.cmu.edu

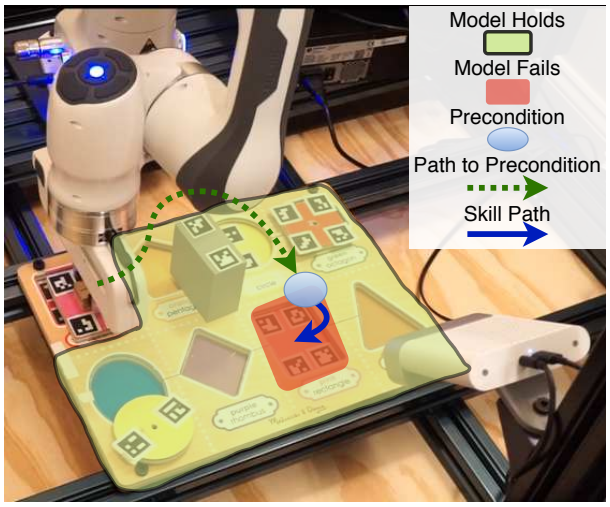

Fig. 1. Our method follows a trajectory to a precondition and then executes a learned skill to complete the shape insertion task in the region where the system model fails. A plastic gray box obstructs the path to the goal. An overhead (not shown in image) and angled camera are aimed at the center of the board.

In this work, we propose a framework that facilitates motion planning with sub-optimal planners, limited training data, and inaccurate system models. Our method combines model-based planning with model-free learning by first identifying states where the transition model provides a poor estimate and then learning a local policy, e.g., a skill, to patch the task execution. The robot can then reliably perform the task by predicting if the original plan will lead to model failure, then adapt the plan and patch it with a learned skill if needed. Learning skills that only need to generalize across a small part of the state space can require fewer samples than a more general learned skill would need.

Integrating planning and learning in this manner enables the robot to acquire new skills only when needed. The new skill is data-efficient to learn as it only generalizes across the relatively narrow range of conditions where the model fails. This hybrid approach maintains the benefit from the broad generalization afforded by the model-based planning in regions where the model is accurate.

We evaluate our method on a shape insertion task and a door opening task, comparing our results to the performance of pure planning and imitation learning approaches.

II. Related Work

Combining Model-Free & Model-Based Approaches: The most closely related work in combining model-based planning with model-free learning to ours is Lee et al.'s Guided Uncertainty-Aware Policy Optimization [9]. Lee et

al. use model-based planning to reach an uncertain region, which is defined by uncertainty in the observation model, and then switch to a model-free policy for a real-robot peg insertion task. Lee's work focuses on perception uncertainty, while our work directly addresses planning failure due to transition model errors. Other approaches use model-based planning as an exploration policy to more efficiently collect meaningful data samples that can be used for learning a model-free policy [10], [8], [11]. Adaptive Online Planning also combines model-based planning with model-free learning for a highly accurate, but computationally expensive, planner that can obtain trajectories [12]. In contrast, our approach is intended for planners that have have approximate models but are relatively inexpensive to query. Hoppe et al. use active learning as part of the trajectory optimization so the planner can select more informative samples for model-free policies [13].

Policy Composition: Combining and chaining policies is commonly formalized using the *options* framework [14]. Each option has an associated policy, precondition, and set of termination conditions. Konidaris et al. chain policies to form more complex behaviours and examine *option discovery* [15]. Option discovery is a problem where multiple options are learned and added to a system where needed. Most importantly, options can be combined hierarchically to execute complex behaviors [16].

Anomaly Detection: Our method uses anomaly detection to determine when a model deviation has occurred during execution. Multimodal monitoring has been shown to be more reliable than single mode monitoring of unexpected observations [17]. Park et al. use this concept as a flag to terminate executions after an anomaly has been detected [18]. Vemula et al. use a history of where the model has failed to avoid those states while planning [19]. In contrast to previous works, our approach uses anomaly detection with a model-based planning method to determine which states require a local policy using model-free learning.

Exploiting Contacts for Manipulation Tasks: Guan et el. leverage contacts to reduce the number of states considered during planning to scale with state complexity. Guan et al. also model the problem as a composite MDP (Markov Decision Process) in *SE*(2) with added domain-specific structures to enable a solution using dynamic programming [20]. Many approaches use contact modes as a variable during optimization for complex contact-rich locomotion and manipulation tasks [21], [22], [23]. However, the resulting trajectories can be difficult to execute reliably in the presence of modeling errors. Páll et al. use the Contingent, Contact-Exploiting RRT (ConCERRT) framework to find a path that accounts for uncertainty in the transition model, attempting to find a contingency plan for every possible belief state [24].

Model-Free Skill Learning: Contact rich manipulation has also seen advances through the use of model-free reinforcement learning [8], [25], [7]. However, deep reinforcement learning often requires large amounts of training data. Imposing structure on problems can help improve sample efficiency when performing model-free optimization. Constraint Optimization and Reinforcement Learning (CORL) employs a user-specified low-dimensional projection that provides structure to make reinforcement learning more efficient [26]. Englert et al. later use CORL to train a cabinet opening skill from a single demonstration [27]. Additionally, sample efficiency may be improved through defining the policy using fewer parameters [28]. Our method describes how these advances in model-free learning can be integrated into systems that plan using models.

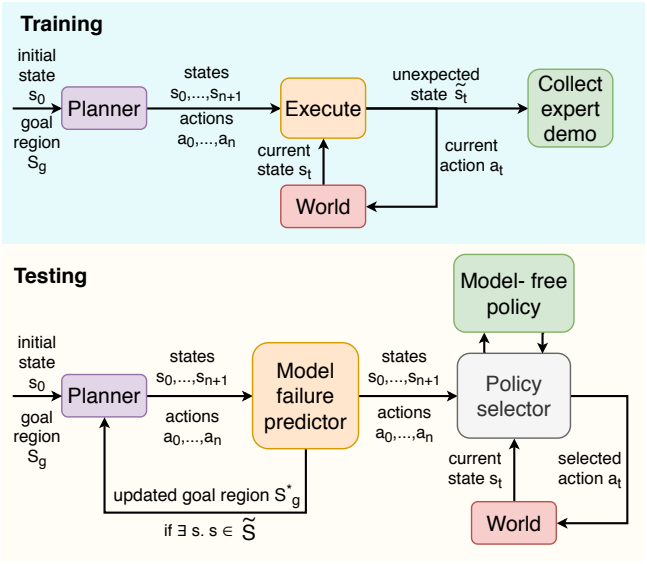

Fig. 2. System block diagram. We use the training phase to collect data and fit models for use in the testing phase. The planner gives a sequence of expected states and actions to the execution module, which checks for anomalous state observations and requests expert demonstrations after such observations occur. We refer to an unexpected state at time t as $\tilde{s}_t$ in this diagram for clarity.

III. Problem Statement

We formulate the problem as a planning problem with a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a starting state $s_0 \in \mathcal{S}$. The robot must compute a finite list of actions $[a_0, a_1, \ldots a_n]$ executed in states $[s_0, s_1, \ldots s_n]$, such that the final state, s_{n+1}, is in a goal set, $\mathcal{S}_g \subseteq \mathcal{S}$. π_{PLAN} maps a state, s_t, deterministically to the next action, a_t, according to the plan. We have access to an approximate transition model, but not the underlying dynamics. The transition model we use is $\widehat{\mathcal{T}} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, which is an estimate of $P(s_{t+1}|s_t, a_t)$ and is not expected to exactly follow the real world distribution. Using $\widehat{\mathcal{T}}$ and s_0, the planner conducts a search to a set of goal states, $\mathcal{S}_g^*$. $\mathcal{S}_g^*$ is an intermediate goal. If the planner is being used to solve the planning problem without the model-free policy, then $\mathcal{S}_g^* = \mathcal{S}_g$.

The planner can generate a suitable plan if the transition model is sufficiently accurate. If the model fails during the execution then the robot should stop following the plan and use a different strategy. Therefore, our approach determines for which $\widetilde{\mathcal{S}} \subseteq \mathcal{S}$ the agent should stop following π_{PLAN} and instead learn (during training) or execute (during testing) the model-free policy, π_{SKILL}. π_{SKILL} is learned from a series of expert actions, $[\hat{a}_t, \hat{a}_{t+1}, \ldots, \hat{a}_m], [\hat{s}_t, \hat{s}_{t+1}, \ldots, \hat{s}_{m+1}]$. In

Algorithm 1 Training procedure

$\mathcal{D}_s \leftarrow \{\}$
$\widetilde{\mathcal{D}}_s \leftarrow \{\}$
$\mathcal{S}_g^* \leftarrow \mathcal{S}_g$
precompute $\pi_{\text{PLAN}}(\mathcal{S}_g^*)$
for $t = 0$ to N **do**
  $s_{t+1} \leftarrow \text{EXECUTE}(a_t)$
  if $P(s_{t+1} \mid s_t, a_t) > p$ **then**
    $\mathcal{D}_s \leftarrow \mathcal{D}_s \cup \{s_t\}$
  else
    $\widetilde{\mathcal{D}_s} \leftarrow \widetilde{\mathcal{D}_s} \cup \{s_t\}$
    $\mathcal{D}_\mathcal{I} \leftarrow \mathcal{D}_\mathcal{I} \cup (\hat{s}_t, \hat{a}_t)$ from expert demonstration
    break
  end if
end for
$\pi_{\text{SKILL}} \leftarrow \text{TRAIN}(\mathcal{D}_\mathcal{I})$

Algorithm 2 Testing procedure

$\mathcal{S}_g^* \leftarrow \mathcal{S}_g$
precompute $\pi_{\text{PLAN}}(\mathcal{S}_g^*)$
for $t = 0$ to N **do**
  if $s_t \in \widetilde{\mathcal{S}}$ **then**
    $\mathcal{S}_g^* \leftarrow \mathcal{I}_{\text{SKILL}}$
    break
  end if
end for
$t \leftarrow 0$
while $s_t \notin \mathcal{S}_g^*$ and $s_t \in \widetilde{\mathcal{S}}$ or $s_t \in \mathcal{I}_{\text{SKILL}}$ **do**
  $s_{t+1} \leftarrow \text{EXECUTE}(a_t)$
  $t \leftarrow t + 1$
end while
while $s_t \notin \mathcal{S}_g^*$ **do**
  $a'_t \sim \pi_{\text{SKILL}}(s_t)$
  $s_{t+1} \leftarrow \text{EXECUTE}(a'_t)$
  $t \leftarrow t + 1$
end while

this work, we sometimes refer to these policies as a *skill*, but symbolically represent them as π_{SKILL} to indicate that the skill is a learned policy.

IV. Technical Approach

A. Overall Approach

Our method is designed to reliably complete a robotic manipulation task using sub-optimal models and limited real-world robot data. A model-based planner is used until model failure is detected and then a policy is learned from demonstrations to reach the goal. At test time, the robot executes the learned skill in the regions where the model has failed. Thus, much of the task is initially completed using the model-based planner to increase generalization and decrease data needs. If no model failure occurs, then no skill is learned. However, if the robot encounters or predicts model failure, a skill is learned to patch the plan and complete the task.

At training time, state regions are gathered where the transition model $\widehat{\mathcal{T}}$ holds, $\mathcal{D}_s$, and does not hold, $\widetilde{\mathcal{D}_s}$. While training, we use π_{PLAN} to obtain samples of (s_{t+1}, s_t, a_t). These samples are labeled according to a confidence interval, $P(s_{t+1} \mid s_t, a_t) > p$, where p is set to 0.98 for our experiments. Our transition model for action a_t from s_t over s_{t+1} is modeled as $\mathcal{N}(s_{t+1}, k_0|s_{t+1} - s_t|)$. This allows the acceptable error to grow proportionally with the distance of the movement, where k_0 is a proportionality constant that can be determined experimentally by fitting expected deviation from the target s_t for different state transitions. We show pseudocode for training our method in Algorithm 1.

The action and state spaces for anomaly detection are high-level; anomaly detection is executed only after the joint trajectory controller has terminated. We use a joint impedance controller so if the controller deviates from the expected path between t and $t+1$, but corrects itself before the next anomaly detection step, then s_{t+1} is not in $\widetilde{\mathcal{S}}$. If a sample is within the confidence interval, then the corresponding state is added to $\mathcal{D}_s$. Otherwise, the sample is added to $\widetilde{\mathcal{D}_s}$. $\widetilde{\mathcal{D}_s}$ and $\mathcal{D}_s$ are then used to estimate $\widetilde{\mathcal{S}}$ using a Gaussian Process [29].

When a state is added to $\widetilde{\mathcal{D}_s}$, the model has failed, and the human operator is asked to provide a training demonstration for completing the task from this state. The skill demonstration is added to the imitation learning dataset, $\mathcal{D}_\mathcal{I}$. This dataset is used to estimate the skill π_{SKILL} and its corresponding initiation set, $\mathcal{I}_{\text{SKILL}}$, a distribution over starting states which the learned skill is likely to achieve the goal from. We further elaborate on the skill learning in Section IV-D.

At test time, π_{PLAN} outputs a series of states. If any states are estimated to be in $\widetilde{\mathcal{S}}$ then π_{PLAN} instead computes a path to S_g^*, which is now $\mathcal{I}_{\text{SKILL}}$. In our experiments, the expert demonstrations for shape insertion produced better results when the shape was reset to a position outside of the hole. The shape was then inserted into the hole using a sliding motion, as shown in Fig. 6. This insight motivated our decision to have the agent change $\widetilde{\mathcal{S}}_g^*$ to the initiation set. Our test algorithm is shown in Algorithm 2. In the following sections, we describe each step in more technical detail.

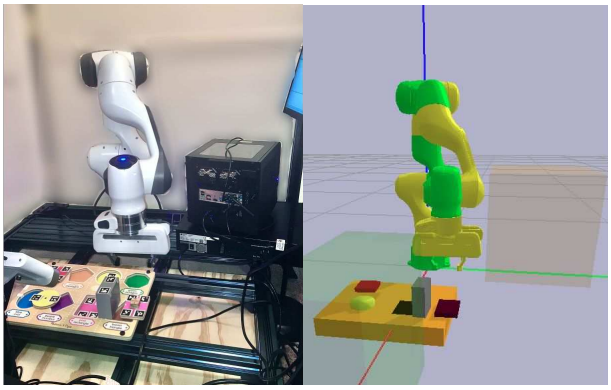

Fig. 3. The real world environment (left) and the corresponding Bullet3D environment for planning and collision detection (right).

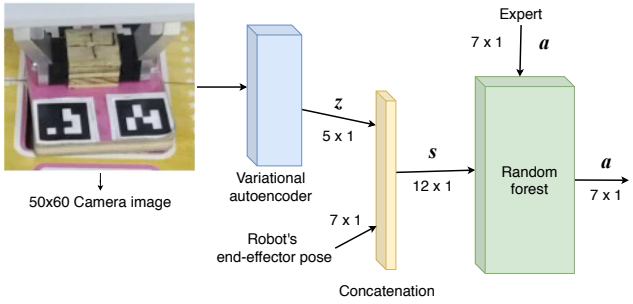

Fig. 4. Dataflow and architecture for model free policy.

B. Model-based Planner

The purpose of the model-based planner is to quickly find a sequence of waypoints, $s_0, s_1, \ldots, s_{n+1}$ and corresponding n actions to get from each s_t to s_{t+1} such that $s_{n+1} \in \mathcal{S}_g^*$ according to $\widehat{\mathcal{T}}$.

Planning is done using a PyBullet simulation [30] to detect collisions and predict transitions, as shown in Fig. 3. The planner has access to approximate CAD models of the objects and their poses. We also assume that objects do not deform during planning.

The high level planner samples grasps and inverse kinematics solutions, and then chooses the grasp that minimizes $|q_0 - q_{n+1}|$, the distance between start and end joint configurations, conditioned on a collision-free plan existing for the task. This high level sampling is important during highly constrained placement scenarios, such as the one shown in Fig. 1, as not all grasps will have a final feasible plan. This long-horizon reasoning facilitates generalization across different situations even in constrained domains.

The motion planner used to achieve goal states is a bi-directional RRT (Rapidly Exploring Random Trees) [31] in order to return a collision-free solution quickly and with high probability. We apply smoothing and restarts to improve the quality of the trajectories: 5 restarts, 100 smoothing iterations, and 200 search iterations for motion planning. We built our planning stack using library tools provided by Garrett [32].

For the door handle turning tasks, we specify that the trajectory is an interpolation in $SE(3)$ between the start handle pose and the end handle pose once grasped. If the initial configuration or end configuration is in collision, our planner returns $\{\}$. Additionally, if s_{t+1} is expected to penetrate an object in simulation, then contact is also predicted. End configurations can be sampled if there is at least one goal state, i.e. $|\mathcal{S}_g| > 1$.

C. Model failure detection during execution

At the beginning of the plan execution, we initialize two datasets for classification: one dataset with expected s_{t+1} given $(s_t, a_t) \sim \pi_{\text{PLAN}}$ $(\mathcal{D}_{\mathcal{S}})$ and one dataset with unexpected s_{t+1} based on the transition model used by the planner, $(\mathcal{D}_{\mathcal{S}})$.

We measure state using our multimodal perception system, which uses vision, joint state estimation, and contact forces. Coordinating across multiple modes addresses partial observability within individual sensor modalities. For instance, precisely measuring the 6 DOF pose of the manipulator is difficult when the robot is grasping it, but binary contact detection and estimation of the robot's end-effector pose is trivial. Thus, we use both deviation of the end-effector in Cartesian space and binary contact sensing for anomaly detection. We describe our perception system in detail in Section V-A.

At test time, if the agent predicts that $\exists s \in [s_0, s_1, \ldots, s_n]$ such that $s \in \widetilde{\mathcal{S}}$, then the planner replans to the skill's initiation set $\mathcal{I}_{\text{SKILL}}$ and then samples actions from the skill policy π_{SKILL} until the task is complete.

We use Gaussian Process (GP) regression on a decision rule, $g(s_t)$, to predict whether $s_t \in \widetilde{\mathcal{S}}$ using a GP, f. The inputs $\mathbf{x}$ are rows of $[s_t]$. The labels, denoted as $\mathbf{y}$, are our decision rule, $g(s_t)$ where $g(s_t) = 1$ if $s_t \in \widetilde{\mathcal{D}_{\mathcal{S}}}$ and 0 otherwise. The kernel is a 5/2 Matérn kernel, a special case for which computation is very efficient. The kernel function is shown in Eq. 1 where d is the distance between x and x', such as $|x - x'|$ [29]. We optimize θ, the hyperparameters of the kernel, which include σ and ρ for each dimension, using Large-scale Bound-constrained Optimization (L-BFGS-B) [33]. Equation 2 describes the likelihood to be maximized.

$$K(d) = \sigma^2 \left(1 + \frac{\sqrt{5}d}{\rho} + \frac{5d^2}{3p^2}\right) \exp\left(-\frac{\sqrt{5}d}{\rho}\right) \quad (1)$$

$$\log p(\mathbf{y}|\theta, \mathbf{x}) = -\frac{N \log 2\pi}{2} - \frac{\log\det(K + \sigma_n^2 I)}{2} - \frac{\mathbf{y}^T K^{-1} \mathbf{y}}{2} \quad (2)$$

The threshold for $g(x)$ to indicate model failure is an application-specific hyperparameter, $0 \leq \tau \leq 1$. We chose $\tau = 0.75$ to be conservative, because model failure in the shape insertion domain can lead to movement of the shape in the robot's gripper, making it less likely that π_{SKILL} will succeed.

D. Model-free Skill Learning

Once the robot encounters a model failure during training, it notifies a human operator to help finish the task and collects samples of $(\hat{s}_t, \hat{a}_t)$, which are added to $\mathcal{D}_{\mathcal{I}}$. The human gives the robot keyboard teleoperated actions to go to a skill starting location and then complete the task. Uniform random noise between $[-\beta, +\beta]$ was added to each action executed. β needs to be high enough to make the skill reliable, but not so high that the human cannot complete the task. We found that the policy produced from this demonstration data alone, without the noise, had a limited distribution of visited states, leading to poor generalization in unfamiliar states. This noise injected demonstration approach was inspired from DART (Disturbances for Augmenting Robot Trajectories) [34], which showed that noise injection lead to a more robust policy.

For state representation, we use a variational autoencoder, which learns an embedding that can be used to reconstruct an

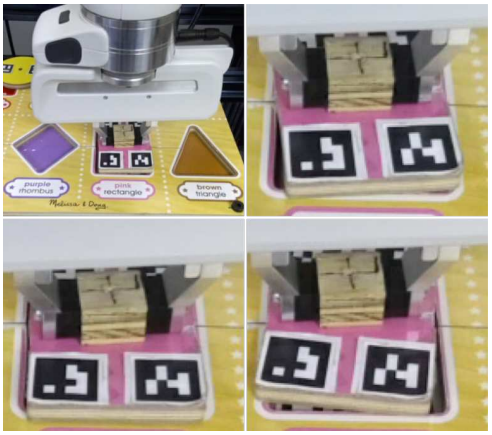

Fig. 5. Camera images corresponding to states where the transition model failed. These are used as inputs to the autoencoder mentioned in Section IV-D. Not all variations of model failure are depicted.

image [35]. The inputs to the autoencoder are camera images of the scene taken as the demonstrations are being performed. Examples of these images are shown in Fig. 5. The embeddings produced, along with the end-effector positions, are then used as inputs to π_{SKILL}. We show the data flow in Fig. 4. The autoencoder loss is the mean-squared error between the original image and the reconstructed image. The architecture is a convolutional layer with 32 5×5 filters, 16 3×3 filters, and a Gaussian noise layer with $\sigma = 0.001$. The output is flattened, and then passed through two more fully connected layers, μ_z and σ_z. We use the re-parameterization trick from [35] to sample from a unit Gaussian, $\epsilon \sim \mathcal{N}(0, I)$ and then sample a latent vector z as $\mu_z + \epsilon\sqrt{\sigma_z}$. We chose z to be 5-dimensional. The z was reconstructed to the original image size using fully connected layers. We based our architecture on [36]. Because our dataset is small, we also perform data augmentation from [37] to reduce overfitting, including rotation, shear transforms, color, and random noise to increase our dataset size by a factor of 50.

After the data is collected, we use $\mathcal{D}_{\mathcal{I}}$ to fit a function $\pi_{\text{SKILL}} := \mathcal{S} \rightarrow A$ that maps each state to an action. π_{SKILL} can also be represented using other policy forms, such as neural networks or Dynamic Movement Primitives [38]. We chose random forest regression (RFR) for our model-free policy due to its interpretability and data efficiency [39]. The representational ability of RFR was sufficient for our tasks.

RFR outputs numerical values instead of class labels, using the mean-squared generalization error. The random forest takes in training data, $\mathbf{X}$, and outputs a tree predictor, $h(x)$ that minimizes the mean-squared generalization error, $\mathbb{E}_{X,Y}\left[(Y - h(X))^2\right]$ over all available demonstrations. Each decision tree in the random forest uses a randomly selected subset of the features. $\mathbf{X}$ is our $\dim(z) \times N$ latent visual features concatenated with the $\dim(a) \times N$ end-effector pose, to predict a $\dim(a) \times N$ action, which is our $\mathbf{Y}$. We perform grid-search cross-validation for hyper-parameter optimization, which include the number of trees, maximum number of features for splitting, minimum number of samples to be at an internal node, minimum number of samples to be at a leaf node, and maximum tree depth. We use the RFR implementation in scikit-learn [37].

Fig. 6. Demonstration where the human operator moves the robot arm, via teleoperation, to complete the shape insertion task after model failure. We found that demonstrations worked best when the operator reset the shape to a position outside of the hole and performed a sliding motion into the hole.

We fit the initiation set to be a Gaussian distribution: $\mathcal{N}(\overline{s_0}, \Sigma_{s_0})$. Initial starting states are sampled from $\mathcal{I}_{\text{SKILL}}$ when setting $\mathcal{S}_g^*$ after model failure or expected model failure.

V. Results

In this section, we describe the experiments used to evaluate our proposed method of combining model-based planning and model-free skill learning. For both tasks, the action is a 3D translation Δx of the end-effector in the world frame and a rotation represented using a quaternion.

A. Experimental Setup

Shape Insertion Setup: The experimental setup for the shape insertion experiment is shown in Figure 1. We use a children's 8-piece knob puzzle (or *board*) mounted at a fixed location. The puzzle pieces are manipulated by a 7 DoF Franka Panda arm towards their respective goal locations. A 7cm x 8cm x 4cm PLA obstacle is placed in a location that obstructs any straight line paths to the goal for the indicated trials. The obstacle is never placed directly over the goal position. For 3D object pose estimation, we attached AprilTags [40] to the top of the puzzle pieces, obstacles, and at the goal locations. An overhead Microsoft Azure Kinect sensor is used to retrieve images of the scene. A second Azure Kinect sensor records downward angled images of the scene as input to the autoencoder described in Section IV-A. The Franka arm indirectly estimates end-effector forces and torques using the joint torques. Force detected relative to the end-effector is converted into a binary signal by setting a force threshold that detects contact with objects, but does not trigger during acceleration for movement in free space.

Baselines: We compare our method to two different baselines. The first is the planner described in Section IV-B. For the second baseline, we learned a policy using Imitation Learning (IL) in conjunction with Dynamic Movement Primitives (DMPs) [38], [41]. For the DMP baseline, initial grasping is performed using the planner. The transport and

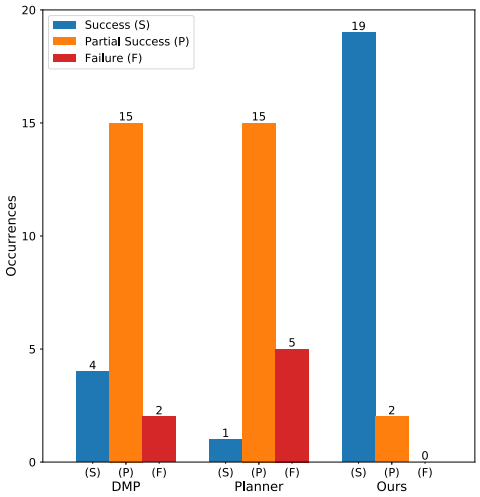

Fig. 7. Real world shape insertion results of all trials. The figure shows the number of times an outcome occurred for each implementation when there was no obstacle obstructing the path. Section V-B defines success.

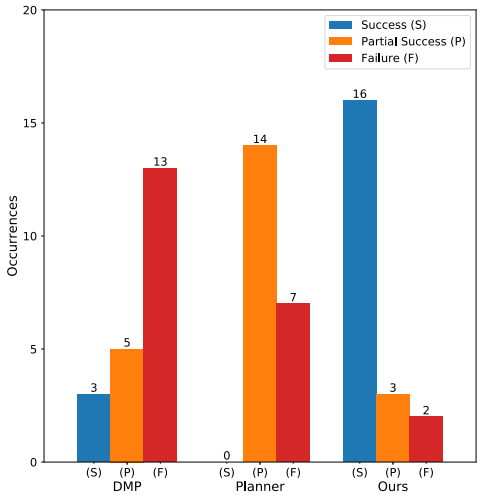

Fig. 8. Real world shape insertion results of all trials with an obstacle obstructing the path. The figure shows the number of times an outcome occurred for each implementation. Success is defined in Section V-B.

# Expert Demos:	1	5	10	20
Success Rate:	$7/21$	$15/21$	$16/21$	$19/21$

Fig. 9. Overall success of our method on the shape insertion task depending on the number of training samples. The first row is the number of training samples used and the second row is the rate of success for the 21 trials. Success and the experimental trials performed are explained in V-B

insertion are done with DMPs. The DMP for this task was trained by collecting a trajectory using a kinesthetic human demonstration, while the obstacle was obstructing the path to the goal. More specifically, the DMP was trained from the oval starting position using the rectangle puzzle piece as the manipuland. Each translational Cartesian dimension was modeled as a separate DMP component. We used the formulation shown in Eq. 3 and 4. We use the modification mentioned in Section IV-A of [38] to include a goal state parameter that scales the DMP trajectory towards the goal state. This formulation also allows for the use of object features that can be used to scale the amplitude of a trajectory. This is useful if the start and end positions are close to one another, since the trajectory's amplitude can become sensitive to the goal state parameter's scaling. We scaled the z dimension trajectory by 0.5 to quell this issue. The DMP parameters for execution are learned through linear ridge regression.

$$\vec{y} = \alpha_z \left(\beta_z \tau^{-2} \left(y_0 - y\right) - \tau^{-1} \dot{y}\right) + \tau^{-2} \sum_{j=1}^{M} \phi_j f\left(x; \boldsymbol{w}_j\right) \tag{3}$$

$$f\left(x; \boldsymbol{w}_j\right) = \alpha_z \beta_z \left(\frac{\sum_{k=1}^{K} \psi_k(x) w_{jk} x}{\sum_{k=1}^{K} \psi_k(x)} + w_{i0} \psi_0(x)\right) \tag{4}$$

B. Experimental Results

In this section, we show how our method compares to the baseline methods described in Section V-A on two real-world tasks. The first task is to insert a puzzle piece into its corresponding hole. The experiments are performed with three shapes: a rectangle, a circle, and a square. Each shape goes into one of 8 corresponding slots in a 4×2 grid, shown in the bottom left of Figure 3. We perform 7 trials for each shape where the starting position is one of the 8 regions, excluding the goal region. These experiments are then repeated with an obstacle obstructing the straight line path from start to goal. Success occurs when the shape is completely in the hole, see the bottom right of Figure 6. A partial success is when only part of the shape is in the hole, see Figure 5. Failure indicates that either the robot hit an obstacle during execution or the shape was not in the hole at all. We show the performance of our method after training, as well as the baseline performances, in Figure 7. The results for the trials with an obstacle are shown in Figure 8.

For the 21 shape insertion trials without the obstacle, we found that the method using the planner was only able to completely insert the puzzle pieces once, which occurred with the circle shape. The failure rate was $5/21$ and the partial success rate was $15/21$. The DMP baseline had a success rate of $4/21$, partial success rate of $15/21$, and failure rate of $5/21$. Lastly, our method had a success rate of $19/21$, partial success rate of $2/21$, and failure rate of $0/21$ when trained on 20 samples.

The next set of experiments show how well each method generalizes when an obstacle is introduced. The DMP method generates trajectories that collided with the obstacle $13/21$ times. Failures occur because the DMP parameters learned do not generalize over all possible obstacle configurations and starting locations. It should also be noted that the robot's joint configuration before the DMP was executed, i.e. the joint positions after the planner finished executing, seemed to affect the DMP's performance. We believe this was due to the goal configuration approaching the robot's joint limits from certain starting configurations.

The method utilizing the planner hit the obstacle two times. This occurred when the task was constrained to a small area, causing the trajectory to go close to the obstacle and making the 1 cm perception error become more significant. An example of this is shown in Figure 3. Our method, which relies on the planner for obstacle avoidance for most of the trajectory, also hits the obstacle $2/21$ trials. The remaining

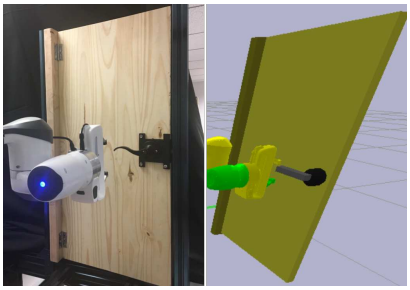

Fig. 10. Task setup for door opening. The real world environment is shown to the left and the simulation environment is shown to the right.

trials show performance similar to the trials without an obstacle, with a $16/21$ success and $3/21$ partial success rate.

We also tested what effect using more trajectory demonstrations to train our method had on success rate. We observed an increase in success rate of insertion when more demonstrations were used. This is expected since the smaller $|\widetilde{\mathcal{D}}|$ is, the smaller the predicted region of $\widetilde{S}$ becomes. Furthermore, more demonstrations lead to a better quality π_{SKILL}. These results are shown in Figure 9 and were performed without an obstacle.

The second task is a door opening task where the robot does not have access to an accurate model of the door handle. The environment we use for planning in simulation is shown in Fig 10. Note that the handle used in the planner is a simple rectangle, while the handle in our setup (shown in the left of Fig 10) is more decorative and harder to grasp with parallel jaw grippers due to the curvature.

We performed 10 trials of the task by executing actions from π_{PLAN}. All 10 trials led to successful opening of the door. There were no unexpected states observed during these trials, so $|\widetilde{\mathcal{D}_{\mathcal{S}}}| = 0$. Since the model-free method was not necessary, our framework only used the model-based planner and displayed the same results.

VI. Discussion

Model failure for the shape insertion task typically occurs shortly after non-sliding contact, e.g. contact with the target hole's border due to misalignment. Misalignment is mainly due to inaccurate localization of the hole from perception, which does not provide a pose of the hole that is accurate enough for this task. Another factor that affected alignment was the rotation that often occurred while puzzle pieces were being handled by the robot. Any deviation from the expected orientation can prevent the puzzle pieces from being inserted into the hole, since it can cause unexpected contact with the surrounding surface. We show examples of failures in Fig. 5.

Once the model fails at time t, s_t is then by definition in $\widetilde{S}$. This implies that the states around s_t are also likely to be in $\widetilde{S}$, so we add s_t to $\widetilde{S}$. When we started collecting data at s_t, we found that in order for the expert to insert the peg, the shape needed to be moved out of the hole so it could be slid back in, as shown in Fig. 6, which motivated our decision to have the agent change the goal state of the planner S_g^* to be the initiation set of the skill $\mathcal{I}_{\text{SKILL}}$.

We observed that nearly all states in $\widetilde{\mathcal{S}}$ have low z-axis values, which corresponds to board contact or sliding. We also found that most of these states are clustered around the target location, which corresponds to the switch from π_{PLAN} to π_{SKILL} being close to the target location. A lower proportionality constant k_0 did lead to some spurious model failure detection. Measuring accumulated error rather than error between s_t and s_{t+1} could solve this.

Combining the planner with the learned skill allows for more of the task to be completed using the planner, which means a better overall system with a lower number of samples needed for training the local skill. For shape insertion, a reasonably reliable policy can be trained using only 5 data samples, although more samples were helpful. The trade-off for using only one data point with no domain-specific knowledge about symmetries is that the precondition space is smaller. For example, if the planner cannot find a plan to the precondition, such as if the states in $\mathcal{I}_{\text{SKILL}}$ were obstructed by obstacles, then π_{PLAN} cannot be used to complete the task.

The door opening experiments demonstrated how our method can be used to identify which tasks need data to compensate for modelling deficiencies. In some tasks, the model is sufficient and there are no unexpected observations, meaning we can rely on the planner to generate trajectories and do not need expensive human demonstrations.

This work is an example of a system that combines learning and planning algorithms to collect imitation trajectories only in the region of state space where it is necessary. We can leverage the ability of planners to generalize in regions where the model works and is fast enough to compute with, while also compensating for issues in perception error, modelling error, and search complexity.

Improving the performance of sub-optimal planners using local learned skills has many potential directions for algorithmic development using local learned skills. The primary limitation to our approach is that there is only one local π_{SKILL}. A more effective use of our method would be to learn multiple local policies or use more expressive policies (e.g. neural networks) with larger initiation sets. Additionally, integrating more sensory modalities, such as [7] did, would help mitigate issues caused by inaccuracies in perception.

VII. Conclusion

We proposed an approach that uses a planner, with a coarse probabilistic transition model, to find a trajectory and then switch to a model-free policy when the robot expects or observes model failure. During training, the robot learns a model of states from which we should learn a skill policy and collects expert demonstrations for the learned skill. We show results quantifying our method's improvement over pure planning or imitation learning for shape insertion. We also applied our method to a door opening task as another example of a contact-rich manipulation task. However, it did not need to learn a skill to complete the task. For future work, our method can be applied to dynamic tasks where approximate models are especially useful. Furthermore, we can train multiple local policies for different behaviours, select preconditions, and then choose between them through high-level planning.

ACKNOWLEDGEMENTS

We thank Kevin Zhang, Jacky Liang, Mohit Sharma, and many others for providing the infrastructure needed for the robot experiments.

This work was in part supported by the Office of Naval Research under Grant No. N00014-18-1-2775, and the Army Research Laboratory under grant W911NF-18-2-0218 as part of the A2I2 program. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR or ARL.

REFERENCES

[1] L. E. Kavraki and S. M. LaValle, "Motion planning," in *Springer Handbook of Robotics*. Springer, 2016, pp. 139–162.

[2] T. A. Meriçli, M. Veloso, and L. Akin, "Experience guided mobile manipulation planning," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[3] X. Ji and J. Xiao, "Planning motions compliant to complex contact states," *The International Journal of Robotics Research*, vol. 20, no. 6, pp. 446–465, 2001.

[4] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear (Amsterdam, Netherlands)*, vol. 143, no. 2, pp. 307–330, 1991.

[5] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "Planar in-hand manipulation via motion cones," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 163–182, 2020.

[6] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," in *Conference on Robot Learning*, 2017, pp. 357–368.

[7] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8943–8950.

[8] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," 2017.

[9] M. E. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, R. Ramos, and D. Fox, "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning," in *Advances in Neural Information Processing Systems*, 2009, pp. 1015–1023.

[10] P. Shyam, W. Jaśkowski, and F. Gomez, "Model-based active exploration," 2018.

[11] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1071–1079.

[12] K. Lu, I. Mordatch, and P. Abbeel, "Adaptive online planning for continual lifelong learning," 2019.

[13] S. Hoppe, Z. Lou, D. Hennes, and M. Toussaint, "Planning approximate exploration trajectories for model-free reinforcement learning in contact-rich manipulation," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4042–4047, 2019.

[14] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[15] G. Konidaris and A. G. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *Advances in neural information processing systems*, 2009, pp. 1015–1023.

[16] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[17] H. Wu, Y. Guan, and J. Rojas, "Analysis of multimodal bayesian nonparametric autoregressive hidden markov models for process monitoring in robotic contact tasks," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419834840, 2019.

[18] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[19] A. Vemula, Y. Oza, J. A. Bagnell, and M. Likhachev, "Planning and execution using inaccurate models with provable guarantees," in *Robotics: Science and Systems*, 2020.

[20] C. Guan, W. Vega-Brown, and N. Roy, "Efficient planning for near-optimal compliant manipulation leveraging environmental contact," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 215–222.

[21] M. Toussaint, K. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning." in *Robotics: Science and Systems*, 2018.

[22] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[23] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[24] E. Páll, A. Sieverling, and O. Brock, "Contingent contact-based motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6615–6621.

[25] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[26] P. Englert and M. Toussaint, "Combined optimization and reinforcement learning for manipulation skills." in *Robotics: Science and systems*, vol. 2016, 2016.

[27] ——, "Learning manipulation skills from a single demonstration," in *The International Journal of Robotics Research*, vol. 37, no. 1. SAGE Publications Sage UK: London, England, 2018, pp. 137–154.

[28] B. D. Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," 2012.

[29] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[30] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[31] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[32] C. R. Garrett, "Pybullet planning," 2020. [Online]. Available: https://pypi.org/project/pybullet-planning

[33] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[34] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "DART: Noise injection for robust imitation learning," 2017.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[36] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 512–519.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] O. Kroemer and G. Sukhatme, "Meta-level priors for learning manipulation skills with sparse features," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 211–222.

[39] B. Jia, Z. Pan, Z. Hu, J. Pan, and D. Manocha, "Cloth manipulation using random-forest-based imitation learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2086–2093, 2019.

[40] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4193–4198.

[41] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *Robotics research. the eleventh international symposium*. Springer, 2005, pp. 561–572.