# Structured Action Prediction for Teleoperation in Open Worlds

Patrick Naughton[1] and Kris Hauser[1], IEEE Senior Member

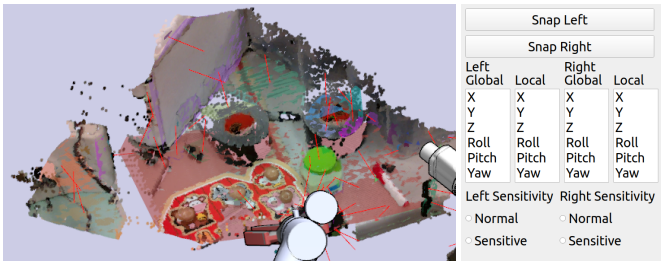
***Abstract*—Shared control can assist a human tele-operator in performing tasks on a remote robot, but also adds complexity in the user interface to allow the user to select the mode of assistance. This paper presents an expert action recommender framework that learns what actions are helpful to accomplish a task, and generates a minimal set of recommendations for display in the user interface. We address the learning problem in an open world context where the action choice depends on an unknown number of objects, i.e., the output domain of the prediction problem changes dynamically. Using structured prediction, we can simultaneously learn what actions to suggest and what objects those actions should act on. In experiments on three tasks in cluttered table-top environments, this method achieves over 90% accuracy in producing the correct suggestion in the top 5 predictions, and also generalizes well to novel tasks with limited training data.**

***Index Terms*—Telerobotics and Teleoperation, Learning from Demonstration, Intention Recognition**

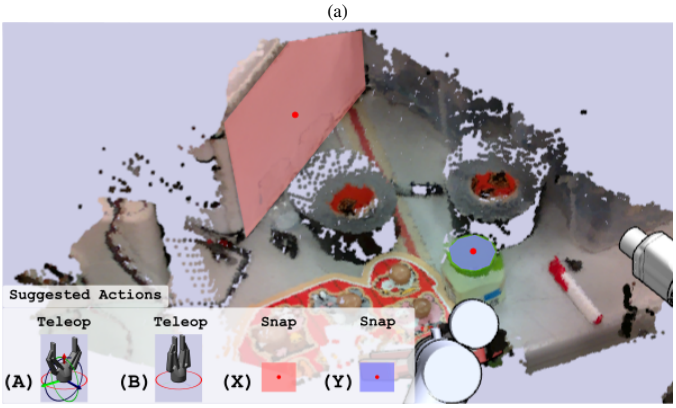
## I. Introduction

TELEOPERATION of robotic manipulators has potential to aid a wide variety of applications including search-and-rescue operations, assistive robotics, and remote medical care. However, multiple factors such as kinematic differences between the robot and the operator, limited perceptual feedback, and network latency prevent the operator from completing tasks as capably as they could in person. Much research has been devoted to developing different kinds of assistive actions to close this performance gap. For example, several methods have been proposed to assist operator grasping [1, 2, 3]. Many previous works have also used machine learning to infer the operator's intent and autocomplete tasks [4, 5, 6, 7]. Each of these methods performs well in some context, but the variety of contexts in which a robot operates means teleoperators need access to many kinds of actions at different points during operation. For complex systems with many actions, presenting them in a standard menu interface can quickly become overwhelming.

This highlights a fundamental trade-off: we wish to maximize the capabilities of a teleoperated robot by supplying many actions, while minimizing the time to access these actions through the user interface. A standard menu interface in Fig. 1a offers extensive functionality, but can be confusing and time consuming to navigate, even if the user has significant experience with the system [8]. We address this problem using an expert action recommender (EAR) approach, illustrated in Fig. 1b. The EAR infers the most likely actions an expert user would perform at a given time, taking into account the desired task, the robot's history of states, and perception information from the environment. It prompts the operator with the $k$ most probable actions, much like a predictive text system on smartphones, so that their decision reduces to only considering which suggestion is best or falling back to the original menu. Higher $k$ values increase the likelihood of including the best suggestion at the expense of increasing the user's cognitive load. A well-designed EAR aids in scaling shared control systems to accept more modes of assistance while enhancing fluency for both novice and expert users.




(a)




(b)

Fig. 1: A standard menu interface for selecting actions and their parameters (a) compared to a mockup of a menu incorporating an expert action recommender (EAR) that suggests the most likely next expert actions automatically (b). (A), (B), (X), and (Y) refer to buttons on the operator's controllers which can be clicked to accept the corresponding suggested action. `Teleop` actions allow the user to move the gripper in the indicated DoFs while `Snap` actions align the gripper with the selected plane. Best viewed in color.

The main contribution of this work is an EAR for action prediction in open worlds, where the set of feasible actions

Manuscript received: September, 9, 2021; Revised December, 10, 2021; Accepted January, 10, 2022.

This paper was recommended for publication by Editor Jee-Hwan Ryu upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by NSF Grant #2025782.

[1]P. Naughton and K. Hauser are with the Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA. {pn10, kkhauser}@illinois.edu

Digital Object Identifier (DOI): see top of this page.

changes over time. Because the set of perception objects — and hence the feasible set of actions — is dynamic, standard classification techniques cannot be applied. We evaluate structured action prediction (SAP) and a combination of feature-space regression with a nearest neighbors search (R+NN) for finding these recommendations. Structured prediction learns a scoring function that rates all candidate actions on a shared scale and suggests the action with highest score [9], whereas regression with nearest neighbors predicts the feature vector of an optimal action (which may or may not be feasible) and suggests the feasible action nearest to this prediction in feature space.

Based on data gathered on a physical robot from cluttered table-top scenes, SAP outperforms R+NN in terms of action selection accuracy across multiple different tasks, achieving top 5 accuracy rates above 90%. We also find that SAP generalizes well in the few-shot setting on novel tasks, requiring fewer demonstrations to achieve high accuracy once examples from other tasks have been collected.

## II. Related Work

Assistive teleoperation has received much attention in the literature, especially after multiple teams identified significant human-robot interaction challenges in the 2013 DARPA robotics challenge [8, 10, 11]. One of the primary obstacles to completing tasks was the large array of interface elements the operators had to use: in one system, the resulting cognitive overload caused the robot to remain still 65% of the operation time, waiting for instructions [8]. Current methods for aiding teleoperation offload some control to the robot itself, and may predict the user's intent to inform how it can help achieve their desired goal. We categorize these works by how heavily their predictions use environment information, since the objects present and task context provided can significantly affect the type and amount of aid the system can offer.

Without any environment information, current methods can provide useful but limited assistance. For example, Zein et al. showed that using previous input commands to predict a motion primitive for a drone can improve the pilot's speed and cognitive load when driving around a track [5]. This method however is difficult to extend to manipulation tasks, where useful geometric motion primitives are more difficult to define. The lack of environmental dependence also prevents learned behaviors from generalizing to new environments.

Another line of work focuses on inferring the operator's intent to guide the robot's end-effector to a desired target automatically [4, 7, 12, 13, 14]. When the operator's intent is known, Leeper et. al introduced a technique to assist pose tracking in the presence of obstacles [15]. These works tend to either focus on lower level functionality, such as reaching for target points or tracking a trajectory [4, 15], or they require hand-identified frames of interest for objects and restrict experiments to well-known scenes [7, 12, 14], or both [13]. Our contribution seeks to provide assistance for higher level tasks without relying on hand-specified frames of interest and instead learns these from data.

Kent et al. [1, 3] and Wang et al. [6] use environment information most heavily out of the work reviewed here. Kent et al. examined the effect of increasingly autonomous operation for grasping objects in clutter. They showed that more assistance allowed users to complete tasks more quickly and with higher success rates. However, their interface for post-grasp manipulation was restricted to a few motion primitives. This paper addresses a higher level problem of creating a framework that accommodates many different types of actions. Wang et al. is the most similar to this work: they used 6 DoF input to control a simulated robot and deep learning to predict the user's desired action and the target of that action based on the operator's motions and high-level environment information (such as the states of entire objects). Such predictions could identify, for example, that the user wanted to grasp a ball and help correct the arm's trajectory to achieve that goal. However, Wang et al. restricted the robot's environments to a maximum of 8 objects and used training and testing environments with the same number and type of objects. In contrast, this paper introduces a new framework to handle open-world scenarios where the types and quantities of objects are not known a-priori, and our work is applied to a physical robot in cluttered scenarios.

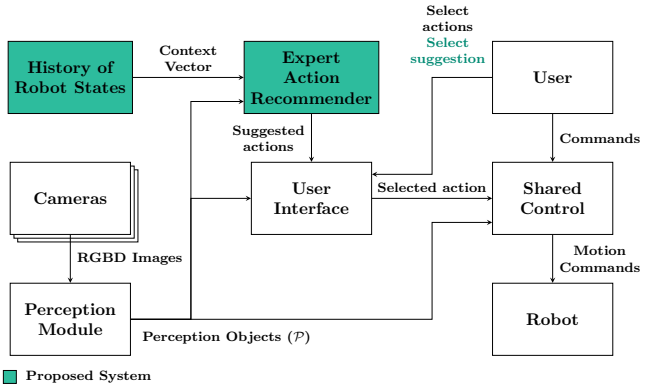



Fig. 2: Block diagram of traditional shared control architecture and proposed modifications (green) to produce expert action recommendations.

## III. Approach

Our EAR considers a generic robot with sensors and a perception pipeline that can identify the robot's state (joint positions, transforms of key points on the robot) as well as relevant *perception objects* (denoted $\mathcal{P}$) such as objects to grasp, planes fitted to the environment, or point clouds of the robot's surroundings. A block diagram of a typical teleoperation system, along with the proposed EAR pipeline, is shown in Fig. 2. The EAR considers the history of robot states and available perception objects to produce contextually relevant action suggestions. Each action configures a shared control module, that allows the operator to command different behaviors to the robot.

### A. Learning Problem

The robot's shared control system is configured according to an action $(a, \psi^{(a)})$, in which $a \in \mathcal{A}$ is an *action type* and $\psi^{(a)} = (\phi_1, \ldots, \phi_{q^{(a)}})$ is a list of *action parameters*. Actions

are policies that map the joint state of the robot, user, and environment to robot motions and a termination flag. These policies are described by arbitrary code blocks. Once an action completes, it sets its termination flag, and the robot reverts to an idle state. The user can also preemptively set this flag to end an action when they choose. For example, a `pick` action on a bimanual robot could accept a list of parameters $\psi^{(\texttt{pick})} = (\phi^{(\texttt{item})}, \phi^{(\texttt{arm})})$ where the two parameters denote the item in the environment to grasp and which arm to use. This action would terminate after grasping the desired object. To assist a user during teleoperation, the EAR decides when to suggest an action(s) to the user, and which action(s) to suggest. For simplicity, we consider the case in which the EAR produces suggestions whenever the robot is not performing an action. This suggests actions only when the user is considering starting a new one to avoid distracting them from the current task.

The goal of prediction is to find $k$ distinct actions $(a, \psi^{(a)})$ that achieve the highest values of $p(a, \psi^{(a)}|x)$ where $p$ encodes the probability distribution over action types and parameter lists that an expert teleoperator would select given the context $x$. We represent context as a vector encoding a history of robot states and previous actions. We predict the actions an expert, rather than the current user, would take because these suggestions can guide novices to more efficiently accomplish their task. We model $p$ by learning an expert's likely action choices from data. To collect such a dataset, we record several sequences of actions $\mathcal{S}_i$ an expert uses to complete a given task and denote the full dataset $\mathcal{D} = \cup_{i=1}^{N} \mathcal{S}_i$. We then split demonstrations for each task into training, validation, and testing sets.

Although the set of action types $\mathcal{A}$ is known a-priori, for actions that use perception objects, the domain of $\psi$ will depend on $\mathcal{P}$, so the learned model of $p$ and the top $k$ predictions must be adaptable to dynamic domains. Specifically, we denote the domain of $\psi^{(a)}$ as $\Psi^{(a)}(\mathcal{P})$, and throughout this paper our notation will drop the domain's dependence on $\mathcal{P}$ if it is clear from context.

### B. Implemented Actions in Our System

Our implementation uses a single robot arm with a parallel-jaw gripper and a wrist-mounted camera. We implement two action types: `teleop(s, x, y, z, roll, pitch, yaw)` that applies different filters to the user's input when sending commands to the robot arm, and `snap(pl)` that aligns the robot's gripper with the selected plane. Thus, $\mathcal{A} = \{\texttt{teleop}, \texttt{snap}\}$.

The `teleop` action accepts a discrete sensitivity setting ("Normal" or "Sensitive"), and 6 flags denoting constraints on the translation and rotation of the end effector ("On" or "Off"). We can then write the domain of $\psi^{(\texttt{teleop})}$ as $\Psi^{(\texttt{teleop})} = \{\text{Normal}, \text{Sensitive}\} \times \{\text{On}, \text{Off}\}^6$. While the user is using a `teleop` action, the pose of their controller relative to its pose when the action is started is tracked and may be modified depending on the action's parameters. At "Normal" sensitivity, the target pose of the hand is the result of applying the same relative transformation of the controller to the starting pose of the end-effector. Switching the sensitivity mode scales down this relative transform by $\frac{1}{4}$ to enable more precise manipulation. For each constraint marked as active in the action's parameters, the target pose is constrained to have the same value in that task space DoF as the initial pose of the end-effector. For example, `teleop(Normal, Off, Off, Off, On, On, On)` will only allow the gripper to translate, its orientation will remain constant regardless of how the user moves the controller. These flags activate different virtual fixtures constraining the end-effector to a lower-dimensional space to simplify operation [16, 17]. This final target pose is then tracked by compliantly moving the robot's arm to the inverse kinematic solution nearest its current configuration. Compliance is achieved with Cartesian impedance control [18]. To track the user's hand motion we use the controllers of an Oculus Quest 2. The user can initiate a `teleop` action by holding down one of the controller buttons and ends the action by releasing it.

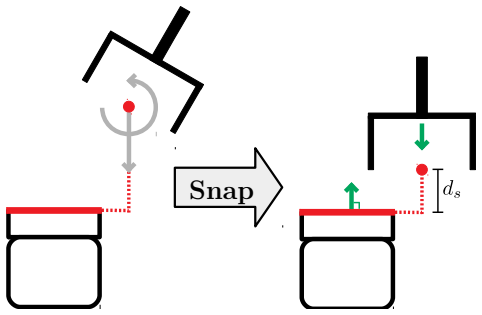



Fig. 3: Effect of the `snap` action in 2D applied to a plane detected from the lid of a jar (red). After the snap action, the vector pointing out of the gripper is parallel to the normal of the plane. Best viewed in color.

The `snap(pl)` action accepts argument `pl`, a plane detected from the environment (i.e., in $\mathcal{P}$), and prepares the user to manipulate objects on or near it. This action aligns the outward normal of the arm's tool-flange with the normal (or negative normal, whichever is closer) of `pl` and moves the robot's tool-tip $d_s$ cm away from `pl`. We set $d_s = 3\,\text{cm}$. Fig. 3 illustrates this process in 2D. This motion keeps the point projected from the robot's tool-tip to the plane stationary, assuming that the user has previously used a `teleop` action to move to a suitable position for their task. This is similar to the third strategy for grasping presented in Ref. [2], but the robot will immediately execute the plan to its target pose after the user selects the desired plane. To find candidate planes in the scene, we modify the agglomerative hierarchical clustering algorithm in Ref. [19] to work with unstructured point clouds by gridding the input point cloud volumetrically instead of in image space. $\mathcal{P}^{(\texttt{plane})}$ denotes the set of planes extracted from the environment using this algorithm, and hence the action parameter domain is $\Psi^{(\texttt{snap})} = \mathcal{P}^{(\texttt{plane})}$. Each $\phi^{(\texttt{plane})}$ consists of the normal vector $\hat{n}$ and a list of the supporting points $\rho$. The user initiates `snap` actions by clicking a button in the standard menu interface and then clicking on one of the planes that appear. If the resulting desired pose is reachable, the `snap` action moves the end-effector to that pose and terminates automatically once it has been reached.

These two actions can help a user complete certain tasks more effectively. For example, if the user wants to press buttons on an instrumentation panel, they can align their end effector to the plane of the panel (`snap(panel)`) and restrict their motion to be normal to this plane and reduce their sensitivity (`teleop(Sensitive, Off, On, On, On, On, On)`). This prevents them from raking the end-effector across multiple buttons or pushing the buttons too hard and damaging them.

## IV. Learning Methods

Although the domain of each parameter list $\psi^{(a)}$ is potentially dynamic, for learning we assume an embedding $\overline{\psi}^{(a)}$ of fixed dimension $\overline{q}^{(a)}$ (Sec. IV-D). This lets us build approximations of $p$ and score top $k$ actions across a dynamic domain that contains any number of perception objects in $\mathcal{P}$.

### A. Structured Action Prediction

Structured prediction is well suited for this problem because it rates each candidate action on a shared scale, rather than trying to directly predict the best one [9]. This aids in directly comparing different action types in top $k$ prediction.

Let $x \in \mathbb{R}^d$ be a context vector, including a short history of the robot's state and previous actions performed by the operator. Let $E(x, a, \psi) : \mathbb{R}^d \times \mathcal{A} \times \sum_{a\in\mathcal{A}} \Psi^{(a)}(\mathcal{P}) \to \mathbb{R}$ be a scoring function. We want to train $E$ so that the most likely next action $(a^*, \psi^*)$ satisfies $(a^*, \psi^*) = \arg\max_{(a,\psi)} E(x, a, \psi) \ \forall x \in \mathcal{D}$. We define $E$ in terms of neural networks $G$ and $A$. The parameter scorer $G^{(a)}\left(x, \overline{\psi}^{(a)}; \theta_G^{(a)}\right) : \mathbb{R}^{\overline{q}^{(a)}+d} \to \mathbb{R}$ for each $a \in \mathcal{A}$ assigns a scalar score to a candidate parameter list for action type $a$ given the context vector $x$ and is parameterized by weights $\theta_G^{(a)}$. The action scorer $A(x; \theta_A) : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{A}|}$ is parameterized by $\theta_A$ and maps each action type to a scalar. In the rest of this paper, we may omit functional dependencies on $\theta_G^{(a)}$ and $\theta_A$. Using $G^{(a)}$ and $A$, we express $E$ as

$$E(x, a, \psi) = e_a^\intercal A(x) + G^{(a)}(x, \overline{\psi}) \tag{1}$$

where $e_a$ denotes the $a$th standard basis vector. $E$ is undefined if the type of $\psi$ is not $a$. To perform inference, we must solve the program

$$\arg\max_{a\in\mathcal{A}, \psi\in\Psi^{(a)}} E(x, a, \psi) \tag{2}$$

taking the solution as the most likely next action and parameter list. We perform this optimization by exhaustively enumerating each action and parameter. This is feasible because $|\Psi^{(\texttt{teleop})}| = 128$, and for typical scenes $|\Psi^{(\texttt{snap})}(\mathcal{P})| \leq 50$. To make this approach more scalable for larger parameter domains, we could implement a branch-and-bound search [20] or use approximate inference [21].

For training, we minimize the structured SVM loss function [21, 22]:

$$\sum_{\{x_i,a_i,\psi_i\}\in\mathcal{D}} \lambda_i \max_{(\hat{a},\hat{\psi})} [\Delta((\hat{a}, \hat{\psi}), (a_i, \psi_i)) + E(x_i, \hat{a}, \hat{\psi}) - E(x_i, a_i, \psi_i)]_+ \tag{3}$$

where $[\cdot]_+$ denotes $\max(\cdot, 0)$ and $\Delta(\cdot, \cdot)$ is a distance between candidate [action type, parameter list] pairs. Let $c_{a_i}$ denote the number of times $a_i$ appears in $\mathcal{D}$. $\lambda_i = |\mathcal{D}|/c_{a_i}$. To evaluate (3), we need to find the candidate output with maximum loss augmented margin violation for each training point, solving

$$\max_{\hat{a},\hat{\psi}} \Delta((\hat{a}, \hat{\psi}), (a_i, \psi_i)) + E(x_i, \hat{a}, \hat{\psi}). \tag{4}$$

We use the same optimization procedure to solve (4) as in (2). We define $\Delta(\cdot, \cdot)$ as a variant of the Hamming loss:

$$\Delta((a, \psi), (\hat{a}, \hat{\psi})) = \begin{cases} \delta_a(\psi, \hat{\psi}) & a = \hat{a} \\ \Delta_{\text{max}} & \text{else} \end{cases} \tag{5}$$

where $\Delta_{\text{max}}$ is a hyperparameter specifying the distance between distinct action types (simply set to $\max_a q^{(a)}$ in our implementation), and $\delta_a$ is the Hamming distance

$$\delta_a(\psi, \psi') = \sum_{k=1}^{q^{(a)}} \mathbb{I}[\phi_k = \phi'_k] \tag{6}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function.

### B. Regression + Nearest Neighbors

For comparison, we also implement a simpler approach, R+NN, which predicts action type probabilities and regresses the parameter feature vector for the chosen action. First, we train a multi-layer perceptron (MLP) with $|\mathcal{A}|$ output units with a standard cross entropy loss, taking the $x$ context vector as input. This network outputs $p(a|x)$, a probability distribution over action types given the input $x$ and we select the action type $\arg\max_{a\in\mathcal{A}} p(a|x)$ with maximum likelihood to suggest. To predict the parameters of actions, we train a separate MLP $B^{(a)}(x) : \mathbb{R}^d \to \mathbb{R}^{\overline{q}^{(a)}}$ for each action, each outputting the predicted concatenation of all parameter vectorizations for context vector $x$. This can be thought of as the features of the "optimal" parameter list, but it may not be feasible (for example, the predictor may regress a feature vector for a plane that does not exist in the environment). To select the top $k$ parameters, we find the $k$ feasible parameters whose vectorizations are closest to $B(x)$, that is, $\psi^* = \arg\min_{\psi\in\Psi^{(a)}} \left\|B^{(a)}(x) - \overline{\psi}\right\|$. During training, each parameter predictor optimizes the squared loss:

$$\sum_{\{x_i,a_i,\psi_i\}\in\mathcal{D}} \left\|B^{(a_i)}(x_i) - \overline{\psi}_i\right\|^2. \tag{7}$$

### C. Learner Architectures

Fig. 4 shows a block diagram of the architectures used for R+NN and SAP. To make comparisons between techniques fair, the internal structure of the networks were kept as consistent as possible. The action predictor and action scorer have the same architecture: we found that a linear predictor sufficed for this submodule. Each *parameter scorer* in SAP begins with a feature MLP $F(x; \theta_F^{(a)}) : \mathbb{R}^d \to \mathbb{R}^{d_f}$ that produces a feature embedding $f$ of the input. This is then concatenated with the embedding of a candidate parameter list $o = [f, \overline{\psi}]$. We concatenate the vector $o$ with the unique

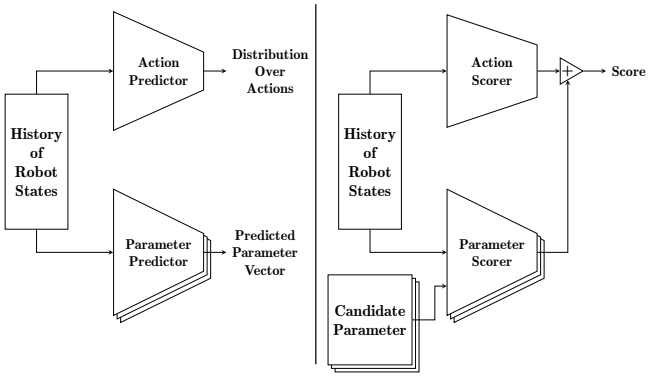



Fig. 4: R+NN (left) and SAP (right) architectures for learning actions.

TABLE I: Context features.

| | |
|---|---|
| Gripper | $[q_g, q_g > 0.4, q_g > 0.5, q_g > 0.6]$ |
| Vision | Distance to closest point to the wrist camera<br>$3 \times 3 \times 1$ (5 cm) occupancy map around $T_{bt}$ |
| History ($\eta = 4$) | Action type ($\lvert\mathcal{A}\rvert$ element one-hot vector)<br>$[\Delta q_g, \Delta T_{bt}.z, \sqrt{(\Delta T_{bt}.x)^2 + (\Delta T_{bt}.y)^2}]$ |

terms of the outer product $o \cdot o^\intercal$ and pass the result into another MLP $H$ that produces the final score. Experiments find that the outer product operation provides a small accuracy boost. Each *parameter predictor* in R+NN has an MLP with the same architecture as $F$ but whose output $f$ has size $d_f + \overline{q}^{(a)}$. We then perform the same flattening and deduplication procedure on $f \cdot f^\intercal$ before concatenating it with $f$ and passing the resulting vector through another MLP whose architecture is the same as $H$ but with $\overline{q}^{(a)}$ output units. We lightly tuned the hyper parameters of training based on validation set performance, but kept them constant across task type, using a learning rate of 0.001 for both the structured and regressive models, while training the structured model for 200 epochs and the regressive one for 500. We used the Adam optimizer [23] to train our networks with a batch size of 10, updating the entire weight set $\theta = \left[\theta^{(A)}, \theta_G^{(\texttt{teleop})}, \theta_G^{(\texttt{snap})}\right]$.

### D. Feature Selection

To learn to predict we must represent parameter lists and the context of the robot as vectors. The context vector $x$ is a concatenation of *gripper*, *vision*, and *history* features, which are described in more detail in Table I. In this listing, $q_g$ denotes the joint angle of the robot's gripper and $T_{bt}$ the transform of the robot's tool tip in the robot's base frame. The current state is measured just before the start of the next action. History is recorded for the previous $\eta = 4$ actions and $\Delta$ indicates the difference in a value between the start and end of the corresponding action.

For parameter $\phi$, $\overline{\phi}$ denotes its representation as a feature vector in $\mathbb{R}^n$ ($n$ different for each type) and $\overline{\psi}^{(a)}$ denotes the concatenation of a list of parameter vectorizations, $[\overline{\phi}_1, \ldots, \overline{\phi}_q]$. Sensitivity $\phi^{(\texttt{s})}$ is encoded as a two-element one-hot vector, and Booleans are either 0 or 1. Plane encodings are much more complex. Denote the transforms $T = (R, p)$ of the robot's base, end effector, and tool-tip in the world frame respectively as $T_{wb}$, $T_{we}$, $T_{wt}$. We also compute the centroid of the supporting points ($c$), the convex hull of those points projected to the plane ($h$), the RMSE fitting error ($e$), and the projections of the points onto the plane ($\rho^P$). Let $r_m = \max_{s \in \rho^P} \|s - c\|$.

TABLE II: Parameter features.

| | | |
|---|---|---|
| $\overline{\phi}^{(\texttt{plane})}$ (SAP) | Point count | $\lvert\rho\rvert/10^6$ |
| | RMSE | $e$ |
| | "Circleness" | $\texttt{Area}(h)/(\pi r_m^2)$ |
| | Plane distance | $\lvert\hat{n} \cdot (p_{wt} - c)\rvert$ |
| | Alignment | $R_{wt}.x \cdot \hat{n}$ |
| | Centroid distance | $\lVert c - p_{wt}\rVert$ |
| | $\hat{n}$ in end-effector frame | $R_{we}^\intercal \hat{n}$ |
| | $c$ in end-effector frame | $T_{we}^{-1} c$ |
| | $c$ in base frame | $T_{wb}^{-1} c$ |
| $\overline{\phi}^{(\texttt{plane})}$ (R+NN) | $c$ in end-effector frame | $T_{we}^{-1} c$ |
| | $\hat{n}$ in base frame | $R_{wb}^\intercal \hat{n}$ |
| | $c$ in base frame | $T_{wb}^{-1} c$ |

TABLE III: Number of demonstrations collected for each task type.

| Task | Train | Val | Test | Avg. Actions per Demo |
|---|---|---|---|---|
| Jar | 25 | 5 | 9 | 9.4 |
| Animal | 4 | 1 | 2 | 52.0 |
| Erase | 15 | 4 | 6 | 8.7 |

We picked $\overline{\phi}^{(\texttt{plane})}$ for each of SAP and R+NN by picking a set of features we thought would indicate how likely a plane is to be snapped to. We down-selected from this set by iteratively ablating each one and measuring the resulting model's validation accuracy on all tasks. This was done individually for SAP and R+NN. Table II shows the resulting features for each method.

## V. Experiments

We demonstrate our system on three tasks: opening a jar, solving an animal puzzle, and erasing a white board. These are illustrated in Fig. 5. Each task can benefit from autonomous alignment with planes: aligning with the lid of the jar makes it easier to isolate a twisting motion along the correct axis to unscrew it; aligning with the plane of the puzzle makes it easier to drop pieces in their slots; and aligning with the white board makes it easier to erase with large motions while maintaining contact. Demonstrations are collected using a single UR5e robot arm with a Robotiq 2F-140 parallel-jaw gripper mounted to its tool-flange. The UR5e arm has a force-torque sensor at its wrist whose feedback is used to achieve impedance control. Perception data comes from a wrist-mounted Intel L515 LiDAR camera. When teleoperating, the expert demonstrator (one of the authors) uses an Oculus Quest 2 headset and controllers, as well as an on-screen interface to send commands to the robot, and can view both the robot itself and the colored point cloud from the wrist camera.

Table III shows how many demonstrations were collected for each type of task as well as the train/validation/test split of the data and the average number of actions in each demonstration. Typically, a `teleop` action lasts between 10 and 15s while a `snap` action takes 3 to 5s. Demonstrations for the jar and erase tasks take about 200s on average while animal tasks take about 15 min.

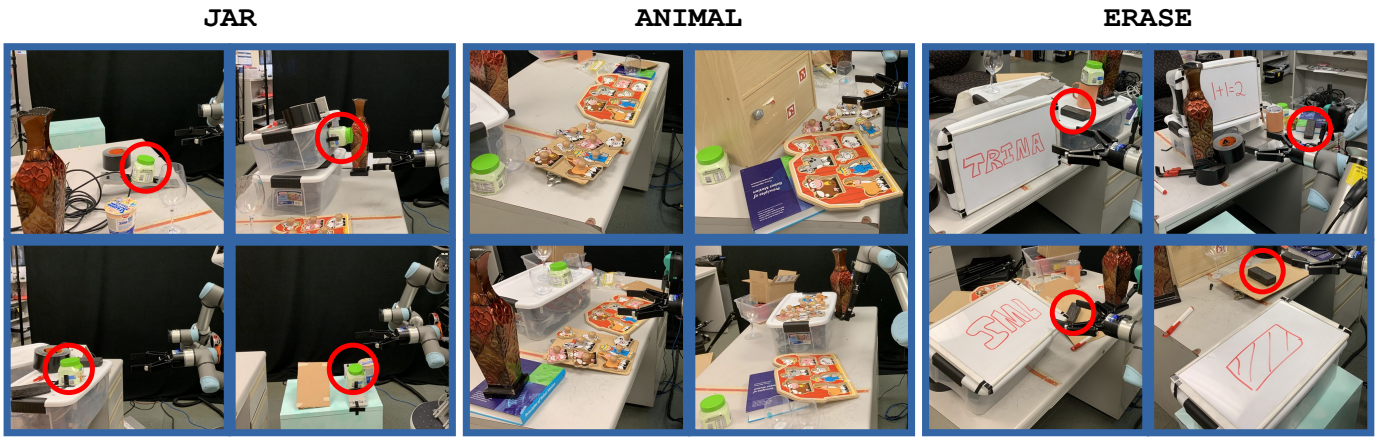



Fig. 5: Examples of the initial setups for the jar, animal puzzle, and whiteboard tasks. Red circles in the jar and erase tasks highlight the jar to unscrew and eraser to use respectively. Best viewed in color.

TABLE IV: Top 1, 3, and 5 accuracies on the single task test.

| Task | Model | Top 1 | Top 3 | Top 5 |
|---|---|---|---|---|
| Jar | R+NN | 65.4% | 67.9% | 69.1% |
| | SAP | **82.7%** | **85.2%** | **87.7%** |
| Animal | R+NN | 73.2% | 82.1% | 84.8% |
| | SAP | **79.5%** | **89.3%** | **91.1%** |
| Erase | R+NN | 72.0% | 76.0% | 82.0% |
| | SAP | **84.0%** | **90.0%** | **90.0%** |

TABLE V: Top 5 accuracy on the multi task test. Rows compare settings when the task type is known and unknown to the model. Columns compare test accuracy for each action type and overall.

| Task Type Known? | Model | `teleop` | `snap` | **Total** |
|---|---|---|---|---|
| Known | R+NN | 84.7% | 56.6% | 78.6% |
| | SAP | 90.5% | 90.6% | **90.5%** |
| Unknown | R+NN | 86.3% | 58.5% | 80.2% |
| | SAP | 88.9% | 94.3% | **90.1%** |

### A. *Single Task*

In this test, we train each model only on demonstrations of a specific task and test on different demonstrations of that same task. Table IV shows that SAP outperforms R+NN by a 6.3-18.6% margin on each task. The Jar task is most difficult, likely because there are often multiple nearby planes that are nearly aligned (e.g., detected from the sloped sides of the jar). In cases like these, SAP has an advantage because it explicitly reasons over each plane in the environment, rather than relying on nearest neighbors. Both methods have $\sim$10% jumps in performance when going from top 1 to top 3 accuracy on the animal task. This performance boost highlights how a modest increase in $k$ can dramatically increase the chances of providing a useful recommendation.

### B. *Multi-Task*

Next, we combine all of the training demonstrations and train a single Unknown-Task model that must predict the next action without knowledge of the task. We also trained Known-Task models where the current task is encoded as a four element one-hot vector (jar, animal, erase, unknown which is unused in this case) and appended to the $x$ vector. Table V compares the performance of these models. As expected, SAP benefits slightly from knowing the overall task. R+NN actually performs slightly worse when aware of the task, which we hypothesize is due to overfitting to the specific task type. The gap between SAP and R+NN is much larger for `snap` than for `teleop` actions. This is expected because the domain of $\Psi^{(\texttt{teleop})}$ is not dynamic, making it easier for traditional regression methods to learn likely `teleop` than `snap` parameters. Perhaps with additional tuning of the nearest neighbors search, this gap could be reduced. Structured prediction on the other hand naturally handles any number of candidate parameters, allowing it to predict variables with dynamic domains much better.

Interestingly, the unknown multi task SAP model outperforms single task SAP on the jar and erase tasks, and only performs slightly worse on the animal task. This indicates that our features capture the utility of various constrained teleoperation settings, and that the notion of "useful" planes to snap to generalizes well across tasks.

Next, to study the importance of various context features in SAP we perform an ablation study. *Full SAP* denotes the use of the whole system for prediction as previously described. *No History* sets $\eta = 0$ so that the model can only observe the robot's current state, *No Gripper* eliminates the gripper context features from the input, and *No Vision* removes vision features (distance measurement and occupancy map). Finally, *No Outer Product* replaces the outer product operation on the concatenated input and parameter list feature embeddings with the identity operation. Table VI shows the top 5 accuracy achieved under each of these treatments, trained and tested in the Known Task setting.

History features provide the largest improvement. Most of this gain comes from improved `teleop` prediction accuracy. This is expected because `teleop` parameters are difficult to score in isolation, but are highly correlated with previous actions. For example, it is much easier to determine that the user would want to constrain the gripper's pitch and yaw if it is known that a `snap` action just occurred, than to infer this from the other context features. Gripper and vision features, as well as the outer product operation, provide modest accuracy boosts as well.

TABLE VI: Top 5 accuracy with different parts of the model or input removed, Known-Task test.

| Treatment | Test Accuracy |
|---|---|
| Full SAP | 90.5% |
| No History | 65.0% |
| No Gripper | 86.8% |
| No Vision | 88.1% |
| No Outer Product | 89.7% |

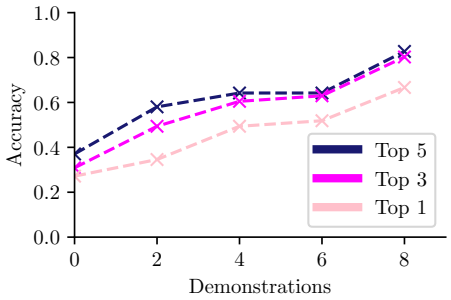



Fig. 6: Few-shot generalization. Top 1, 3, and 5 accuracy of SAP on the jar task as more examples of the jar task are included in the training dataset of animal and erase task demonstrations.

### C. Few-Shot Generalization

Finally, we explore whether SAP generalizes to novel tasks. We train each model on the full training sets of the animal and erase tasks, and include between 0 and 8 jar demonstrations. We then examine their performance on the entire test set of jar demonstrations. Task type is not included in the model input. Fig. 6 shows the top 1, 3, and 5 accuracy on the jar task as more jar demonstrations are added to the training set. Even with relatively few demonstrations, SAP can generalize from the animal and erase tasks, achieving over 80% top 5 accuracy with only 8 demonstrations. Practically, this means that once the model has been trained on a few task types, it may achieve acceptable prediction accuracy for novel tasks without requiring extensive teleoperation data.

## VI. Conclusion

We demonstrated an application of structured prediction to generate accurate suggestions for a teleoperator in open-world scenarios based on expert demonstrations. Even without knowledge of the task to perform, the predictor can produce accurate suggestions and can generalize to new tasks with only a few demonstrations. In future work, we would like to apply this framework to more types of actions such as `pick` that make use of higher level perception objects like segmented items. Additionally, we would like to perform a user study on novice operators with and without the EAR to determine how this affects task completion time, cognitive load, and required menu interactions.

## References

[1] D. Kent, C. Saldanha, and S. Chernova, "A comparison of remote robot teleoperation interfaces for general object manipulation," in *ACM/IEEE Int. Conf. Human-Robot Interaction*, 2017, pp. 371–379.

[2] A. E. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *J. Human-Robot Interaction*, ACM Press, 2012, pp. 1–8.

[3] D. Kent, C. Saldanha, and S. Chernova, "Leveraging depth data in remote robot teleoperation interfaces for general object manipulation," *Int. J. Robotics Research*, vol. 39, no. 1, pp. 39–53, 2020.

[4] K. Hauser, "Recognition, prediction, and planning for assisted teleoperation of freeform tasks," *Autonomous Robots*, vol. 35, no. 4, pp. 241–254, 2013.

[5] M. K. Zein, M. A. Aawar, D. Asmar, and I. H. Elhajj, "Deep Learning and Mixed Reality to Autocomplete Teleoperation," in *IEEE Int. Conf. Robotics and Automation*, May 2021.

[6] C. Wang, S. Huber, S. Coros, and R. Poranne, "Task autocorrection for immersive teleoperation," in *IEEE Int. Conf. Robotics and Automation*, May 2021.

[7] A. K. Tanwani and S. Calinon, "A generative model for intention recognition and manipulation assistance in teleoperation," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2017, pp. 43–50.

[8] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, *et al.*, "An architecture for online affordance-based perception and whole-body planning," *J. Field Robotics*, vol. 32, no. 2, pp. 229–254, 2015.

[9] A. Deshwal, J. R. Doppa, and D. Roth, "Learning and Inference for Structured Prediction: A Unifying Perspective," in *Int. Joint Conf. Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 6291–6299.

[10] E. Krotkov, D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski, "The darpa robotics challenge finals: Results and perspectives," *Journal of Field Robotics*, vol. 34, no. 2, pp. 229–240, 2017.

[11] H. A. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice, "Analysis of human-robot interaction at the darpa robotics challenge trials," *J. Field Robotics*, vol. 32, no. 3, pp. 420–444, 2015.

[12] C. Z. Qiao, M. Sakr, K. Muelling, and H. Admoni, "Learning from Demonstration for Real-Time User Goal Prediction and Shared Assistive Control," in *IEEE Int. Conf. Robotics and Automation*, May 2021.

[13] A. D. Dragan, S. Siddhartha Srinivasa, and K. Kenton Lee, "Teleoperation with Intelligent and Customizable Interfaces," *J. Human-Robot Interaction*, vol. 2, no. 2, pp. 33–79, Jun. 2013.

[14] G. Quere, A. Hagengruber, M. Iskandar, S. Bustamante, D. Leidner, F. Stulp, and J. Vogel, "Shared control templates for assistive robotics," in *IEEE Int. Conf. Robotics and Automation*, 2020, pp. 1956–1962.

[15] A. Leeper, K. Hsiao, M. Ciocarlie, I. Sucan, and K. Salisbury, "Methods for collision-free arm teleoperation in clutter using constraints from 3D sensor data," in *IEEE-RAS Int. Conf. Humanoid Robots*, Oct. 2013, pp. 520–527.

[16] S. A. Bowyer, B. L. Davies, and F. Rodriguez y Baena, "Active Constraints/Virtual Fixtures: A Survey," *IEEE Trans. Robotics*, vol. 30, no. 1, pp. 138–157, Feb. 2014.

[17] D. Aarno, S. Ekvall, and D. Kragic, "Adaptive virtual fixtures for machine-assisted teleoperation tasks," in *IEEE Int. Conf. Robotics and Automation*, 2005, pp. 1139–1144.

[18] A. Albu-Schaffer, C. Ott, U. Frese, and G. Hirzinger, "Cartesian impedance control of redundant robots: Recent results with the DLR-light-weight-arms," in *IEEE Int. Conf. Robotics and Automation*, vol. 3, 2003, pp. 3704–3709.

[19] C. Feng, Y. Taguchi, and V. R. Kamat, "Fast plane extraction in organized point clouds using agglomerative hierarchical clustering," in *IEEE Int. Conf. Robotics and Automation*, May 2014, pp. 6218–6225.

[20] M. Sun, M. Telaprolu, Honglak Lee, and S. Savarese, "An efficient branch-and-bound algorithm for optimal human pose estimation," in *IEEE Conf. Computer Vision and Pattern Recognition*, Jun. 2012, pp. 1616–1623.

[21] D. Belanger and A. McCallum, "Structured prediction energy networks," in *Int. Conf. Machine Learning*, PMLR, 2016, pp. 983–992.

[22] B. Taskar, C. N. Guestrin, and D. Koller, "Max-Margin Markov Networks," *Neural Inf. Processing Systems*, p. 8, Dec. 2003.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.