APPL: Adaptive Planner Parameter Learning

Xuesu Xiao[1], Zizhao Wang[1], Zifan Xu[1], Bo Liu[1], Garrett Warnell[1,2], Gauraang Dhamankar[1], Anirudh Nair[1], and Peter Stone[1,3]

***Abstract*—While current autonomous navigation systems allow robots to successfully drive themselves from one point to another in specific environments, they typically require extensive manual parameter re-tuning by human robotics experts in order to function in new environments. Furthermore, even for just one complex environment, a single set of fine-tuned parameters may not work well in different regions of that environment. These problems prohibit reliable mobile robot deployment by non-expert users. As a remedy, we propose *Adaptive Planner Parameter Learning* (APPL), a machine learning framework that can leverage non-expert human interaction via several modalities—including teleoperated demonstrations, corrective interventions, and evaluative feedback—and also unsupervised reinforcement learning to learn a *parameter policy* that can dynamically adjust the parameters of classical navigation systems in response to changes in the environment. APPL inherits safety and explainability from classical navigation systems while also enjoying the benefits of machine learning, i.e., the ability to adapt and improve from experience. We present a suite of individual APPL methods and also a unifying cycle-of-learning scheme that combines all the proposed methods in a framework that can improve navigation performance through continual, iterative human interaction and simulation training.**

***Index Terms*—Mobile Robot Navigation, Machine Learning, Motion Planning**

I. INTRODUCTION

WHILE engineered systems developed over the past several decades have enabled autonomous mobile robot navigation in certain scenarios, it is typically the case that the parameters of these systems need to be adjusted for each new deployment environment. This process is referred to as "parameter-tuning," i.e., the process through which a human with expert knowledge of the underlying system and the role of each parameter—along with trial-and-error, experience, and intuition—manually finds the best set of system parameters for a new environment. This traditional setup (1) precludes non-expert users from optimizing the default robot system; (2) assumes a *single* set of parameters will work well for all regions of a specific environment; and (3) completely abandons previously fine-tuned parameters when using a new set of parameters.

In contrast, many humans—even those with little to no robotics experience—can easily teleoperate mobile robots in new environments, and even more humans are able to provide simple evaluative feedback (e.g., a numerical assessment of performance) on the robot's behavior. These *in situ* end-user interactions can provide a rich source of knowledge regarding how the system should behave in deployment environments. We hypothesize that systems that can capture and leverage these interactions will be able to quickly adapt to their deployment environments, i.e., exhibit robust and reliable navigation system performance.

In this paper, we investigate this hypothesis by developing and studying a novel family of algorithms called Adaptive Planner Parameter Learning (APPL). In particular, we present an algorithm that can learn from human demonstrations (APPLD), an algorithm that can learn from human interventions (APPLI), and an algorithm that can learn from human-generated evaluative feedback (APPLE). Additionally, we present an algorithm in this family that can, if available, leverage simulation experience in an unsupervised fashion using reinforcement learning (APPLR). Importantly, the framework under which we develop APPL does not replace existing systems for autonomous navigation, but rather *augments* them by providing a new learned module that adjusts only certain hyperparameters of these systems (e.g., obstacle inflation radius, sampling rate, cost function coefficients, etc.). By adopting this approach, APPL systems inherit the advantages of existing navigation systems (e.g., safety and explainability), while also enjoying the benefits of machine learning methods (e.g., adaptivity and improvement from experience). Moreover, the APPL framework explicitly allows for the possibility of adjusting these hyperparameters "on-the-fly", i.e., dynamically changing parameters at every time step to achieve robust navigation. Finally, we also present a unifying vision for how our APPL algorithms can be used in concert to enable a cycle-of-learning framework for continual system improvement through iterative interaction with simulation and end users.

We demonstrate the efficacy of the proposed algorithms in a series of experiments in which a small ground robot, accompanied by a human, must autonomously navigate in a number of complex, constrained environments. Our results show that leveraging human interaction does indeed allow the robot to quickly adapt to each new environment, learning to overcome failures and other suboptimal behavior to the point where the human is no longer necessary. Moreover, we show that combining the proposed approaches in the unified framework that can also leverage unsupervised simulated training in a series of system iterations exhibits increasing performance with each new deployment. All experiment videos can be found at https://www.cs.utexas.edu/~xiao/Research/APPL/APPL.html

II. RELATED WORK

This section reviews state-of-the-art approaches that have applied machine learning to the problem of mobile robot navigation and related work in terms of using different human interaction modalities to assist machine learning.

[1]The University of Texas at Austin, Austin, Texas 78712. [2]Computational and Information Sciences Directorate, Army Research Laboratory. [3]Sony AI.

A. Learning for Navigation

Autonomous mobile robot navigation has been a topic of interest to the robotics community for decades [1], [2]. For example, the DWA planner [2] samples feasible motion commands within a dynamic window, evaluates each sample using a forward prediction model, and selects the best sample based on a cost function that considers distance to obstacles, deviation from the global path, and progress toward the goal. While providing verifiable guarantees, these classical approaches still require extensive knowledge from robotics expert onsite during deployment, e.g., through in-situ parameter tuning, to adapt to different navigation scenarios [3], [4]. For example, max linear and angular velocities and their sampling rates determine if the DWA planner can find good motion commands with limited onboard computation, while its different optimization weights affect the final navigation behavior. All experiments reported in this paper use DWA as the underlying motion planning algorithm. Furthermore, those classical systems generally do not learn with increasing navigation experience [5].

With the recent advances in machine learning research, data-driven techniques have started being applied to the mobile robot navigation problem. Xiao, et al. [6] presented a survey on using machine learning for motion control in mobile robot navigation: in addition to some works in the emerging social [7] and terrain-based [8] navigation, most existing learning approaches for navigation adopt an end-to-end learning paradigm to address the classical collision-free navigation problem and are capable of generating navigational behaviors [9], [10]. However, those end-to-end approaches still cannot outperform classical navigation methods or are not even compared to them. More importantly, end-to-end learning is extremely data-hungry, usually requiring millions of training data or training steps, and forgoes verifiable guarantees such as safety and explainability. On the other hand, other work which targeted a specific navigational component [11], [12], [13] has achieved superior performance when being compared to their classical counterparts. Therefore, it is promising to use machine learning at the subsystem or component level and to combine it with the structure of classical approaches [6].

APPL uses machine learning at the parameter level and devises extra learning components that interact with classical navigation systems. Therefore, APPL inherits all the benefits of classical approaches, while enjoying the adaptivity and flexibility of learning methods.

B. Learning from Humans

Using interactions with humans is an effective approach to facilitate learning in general, e.g., Learning from Demonstration or Imitation Learning [14]. Of particular interest to the robotics community is learning through interactions with non-expert users, which relaxes the requirement for expert roboticists. For mobile robot navigation, it is relatively easy for most non-expert users to provide a teleoperated demonstration [15]. If a robot can successfully navigate in most scenarios, it is only necessary to teach the robot where it makes mistakes. Learning from Intervention can therefore be applied only at those trouble-some situations [16]. For non-expert users who are not able to take control of the robot, learning from evaluative feedback [17] provides another interaction modality that teaches the robot with a simple scalar feedback value. Most aforementioned approaches of learning from humans have been proposed in the learning community and applied to test domains such as Atari games or simulations, and have yet been used to tackle physical mobile robot navigation problems. We posit that this gap can be caused by the requirement for extensive training data and training time, which is prohibitive for physical mobile robots navigating in the real world.

APPL utilizes all these human interaction modalities (demonstration, interventions, and evaluative feedback), and combines them with Reinforcement Learning (RL) in a Cycle-of-Learning scheme [18] for autonomous mobile robot navigation. To alleviate the heavy dependency on huge amounts of high quality training data from the real world, APPL learns a *parameter policy* that interacts with an underlying navigation system, in contrast to replacing it by learning an end-to-end motion policy.

III. ADAPTIVE PLANNER PARAMETER LEARNING

In this section, we formalize the *Adaptive Planner Parameter Learning* (APPL) problem, which serves as the foundation for all the following APPL methods.

1) Underlying Navigation Planner: APPL assumes that a mobile robot has an underlying navigation planner $G : \mathcal{X} \times \Theta \rightarrow \mathcal{A}$, where $\mathcal{X}$ is the planner's state space (e.g., robot odometry, sensory inputs, navigation goal), Θ is the space of free parameters for G (e.g., inflation radius, sampling rate, planner optimization coefficients), and $\mathcal{A}$ is the planner's action space (e.g., linear and angular velocity v ω). Using G and a particular set of parameters θ, the robot performs navigation by repeatedly estimating its state x and applying action $a = G(x;\theta) = G_\theta(x)$. Traditional learning methods for navigation replace classical navigation planner G with a differentiable neural network, and use gradient descent to find thousands (or millions) of neural network weights. In contrast, APPL works within the framework of classical planner G, but treats it as a black box, e.g., it does not need to be differentiable, and APPL does not need to understand what each component of θ does.

2) Meta-Environment: APPL works in the context of a *meta-environment* $\mathcal{E}$ composed of both the underlying navigation world $\mathcal{W}$ (the physical, obstacle-occupied world) and the given classical planner G with adjustable parameters $\theta \in \Theta$ and state input $x \in \mathcal{X}$. An APPL agent interacts with this meta-environment $\mathcal{E}$ through a meta-state $s_t \in \mathcal{S}$, which includes both the planner's current state $x_t \in \mathcal{X}$ and previous parameters $\theta_{t-1} \in \Theta$, i.e., $\mathcal{S} = \mathcal{X} \times \Theta$. Instead of the raw action $a \in \mathcal{A}$, APPL's meta-action takes the form of $\theta \in \Theta$, which is the current parameters to be used by G.

3) Parameter Policy: We formulate the APPL problem as a meta Markov Decision Process (MDP) in this meta-environment, i.e., a tuple $(\mathcal{S}, \Theta, \mathcal{T}, \gamma, R)$. Note this meta-MDP differs from the conventional MDP defined by traditional learning-based motion planners, whose states are x and actions are a (Fig. 1 left). Based on a meta-state s_t, an APPL agent

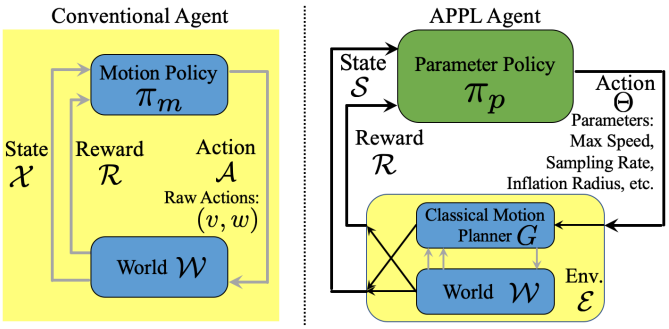

Fig. 1: Contrast between Conventional Agent vs. APPL Agent

takes action θ_t so that the state advances to s_{t+1} based on the transition function $s_{t+1} \sim \mathcal{T}(\cdot|s_t, \theta_t)$, and then receives a reward $r_t = R(s_t, \theta_t)$ (Fig. 1 right). In general, the objective of an APPL agent is to learn a *parameter policy* $\pi : \mathcal{S} \to \Theta$ that can be used to select actions (as parameters θ) that maximize the expected cumulative reward over time,

$$\max_{\pi} J^{\pi} = \mathbb{E}_{s_0, \theta_t \sim \pi(s_t), s_{t+1} \sim \mathcal{T}(s_t, \theta_t)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \quad (1)$$

Within the meta-environment $\mathcal{E}$, with the selected θ, $G_\theta(x)$ produces navigation action a to interact with the physical world $\mathcal{W}$. On top of this general form of parameter policy based on a general reward function, we impose additional structures on the policy and on the reward for each individual APPL method. For example, APPLR adopts this general notion of parameter policy and optimizes a reward function; an APPLE policy maximizes the expected evaluative feedback from human users as an approximation of the underlying reward; APPLI and APPLD simplify the general parameter policy in terms of a pre-trained context predictor to select appropriate parameters from a parameter library.

4) Parameter Library: A specific type of APPL parameter policy $\pi : \mathcal{S} \to \Theta$ can be instantiated by imposing two intermediate mappings, i.e. a parameterized context predictor $B_\phi : \mathcal{S} \to \mathcal{C}$, and a one-to-one mapping $M : \mathcal{C} \to \Theta$, where $\mathcal{C}$ is the space of (finite) *contexts* c (a relatively cohesive region in the physical world $\mathcal{W}$). Therefore, $\theta = \pi(s) = M(B_\phi(s))$. Each predefined context is associated with a set of static parameters. These parameter sets comprise a *parameter library* $\mathcal{L}$. APPLD and APPLI impose such structure on the general parameter policy, and selecting an appropriate parameter set which minimizes a Behavior Cloning loss approximates maximizing the underlying reward in Eqn. 1.

The high-level algorithm of APPL is shown in Alg. 1. The subroutine $\pi = LearnParameterPolicy(\mathcal{I}, \Theta, G)$ is instantiated differently in each APPL method, as specified in Secs. IV-VII.

IV. APPL FROM DEMONSTRATION (APPLD)

Based on the formulation in Sec. III, *Adaptive Planner Parameter Learning from Demonstration* (APPLD) [19] utilizes a context predictor and a *parameter library* learned from human demonstration. A teleoperated demonstration with planner state x and demonstrated action a is collected, which is first

Algorithm 1 APPL

1: // Training
2: **Input:** human interaction $\mathcal{I}$, space of possible parameters Θ, and navigation stack G.
3: $\pi = LearnParameterPolicy(\mathcal{I}, \Theta, G)$.
4: // Deployment
5: **Input:** navigation stack G, parameter policy π.
6: **for** $t = 1 : T$ **do**
7: construct meta-state s_t from x_t and θ_{t-1}
8: $\theta_t = \pi(s_t)$.
9: Navigate with $G_{\theta_t}(x_t)$.
10: **end for**

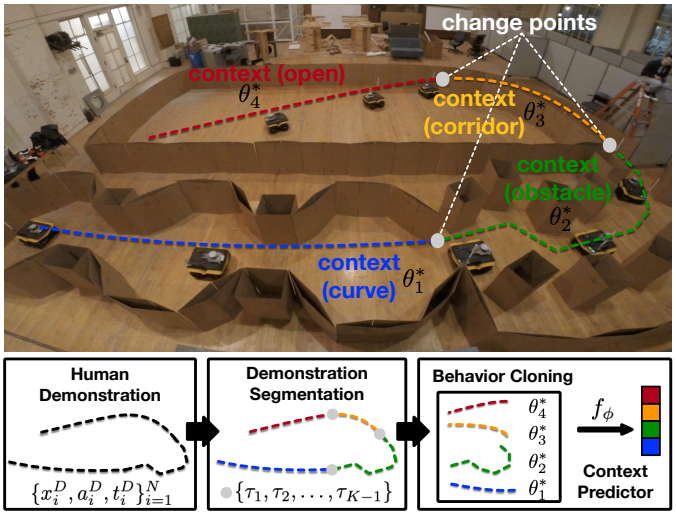

Fig. 2: APPLD: a human demonstration is segmented into different contexts, for each of which, a set of parameters θ_k^* is learned via Behavior Cloning. During deployment, proper parameters are selected by an online context predictor.

segmented into different *contexts* using Bayesian change point detection. Within each context, we use behavior cloning to find a set of planner parameters θ to match the planner's output with the human demonstration (Fig. 2).

A. Learning APPLD Policy

A human demonstration of successful navigation is recorded as time series data $\mathcal{I} = \mathcal{D} = \{x_i^D, a_i^D, t_i^D\}_{i=1}^N$, where N is the length of the series, and x_i^D and a_i^D represent the planner state and demonstrated action at time t_i^D (Line 2 Alg. 1). As mentioned in Sec. III, we impose additional structures on the APPLD policy, i.e. $\pi_D : \mathcal{S} \to \Theta$ is instantiated by a parameterized context predictor $B_\phi : \mathcal{S} \to \mathcal{C}$, and an one-to-one mapping $M : \mathcal{C} \to \Theta$ (Line 3 Alg. 1). Given M and B_ϕ, our system then performs navigation by selecting actions according to $G_\theta(x) = G_{M(B_\phi(x))}(x)$ (Lines 8-9 Alg. 1).

1) Demonstration Segmentation: The key of learning π_D is to construct the space of *contexts* $\mathcal{C}$, each of which corresponds to a cohesive navigation region where a single set of parameters suffices. Further learning is then applied for each specific context. This general segmentation problem can be, in principle, solved by any changepoint detection

method. Given $\mathcal{D}$, a changepoint detection algorithm A_{segment} is applied to automatically detect how many changepoints exist and where those changepoints are within the demonstration. Denote the number of changepoints found by A_{segment} as $K-1$ and the changepoints as $\tau_1, \tau_2, \ldots, \tau_{K-1}$ with $\tau_0 = 0$ and $\tau_K = N+1$, the demonstration $\mathcal{D}$ is then segmented into K pieces $\{\mathcal{D}_k = \{x_i^D, a_i^D, t_i^D \,|\, \tau_{k-1} \leq i < \tau_k\}\}_{k=1}^K$.

2) Parameter Learning: Following demonstration segmentation, we then seek to learn a suitable set of parameters θ_k^* for each segment $\mathcal{D}_k = \{x_i^D, a_i^D, t_i^D \,|\, \tau_{k-1} \leq i < \tau_k\}$. To find this θ_k^*, we employ *behavioral cloning* (BC) [20], i.e., we seek to minimize the difference between the demonstrated actions and the actions that G_{θ_k} would produce on $\{x_i^D\}$. More specifically,

$$\theta_k^* = \underset{\theta}{\operatorname{argmin}} \sum_{(x,a)\in\mathcal{D}_k} ||a - G_\theta(x))||_H, \tag{2}$$

where $||v||_H = v^T H v$ is the induced norm by a diagonal matrix H with positive real entries, which is used for weighting each dimension of the action. A black-box optimization method $A_{\text{black-box}}$ is then applied to solve Eqn. 2. Having found each θ_k^*, the *parameter library* $\mathcal{L}$ is formed and the mapping M is simply $M(k) = \theta_k^*$.

3) Online Context Prediction: The context predictor B_ϕ is learned with a supervised dataset $\{x_i^D, c_i\}_{i=1}^N$, where $c_i = k$ if $x_i^D \in \mathcal{D}_k$. A parameterized function $f_\phi(x)$ is learned via supervised learning to classify which segment x_i^D comes from:

$$\phi^* = \underset{\phi}{\operatorname{argmax}} \sum_{i=1}^{N} \log \frac{\exp\left(f_\phi(x_i^D)[c_i]\right)}{\sum_{c=1}^K \exp\left(f_\phi(x_i^D)[c]\right)}. \tag{3}$$

Given f_ϕ, we define our context predictor B_ϕ according to

$$B_\phi(x_t) = \text{mode}\left\{ \underset{c}{\operatorname{argmax}}\, f_\phi(x_i)[c],\ t-p < i \leq t \right\}. \tag{4}$$

In other words, B_ϕ acts as a mode filter on the context predicted by f_ϕ over a sliding window of length p.

The $LearningParameterPolicy$ subroutine in Alg. 1 for APPLD is shown in Alg. 2, where the above three stages are applied sequentially to learn a *parameter library* $\mathcal{L} = \{\theta_k^*\}_{k=1}^K$ (hence the mapping M) and a *context predictor* B_ϕ, both of which constitute the APPLD policy π_D. During deployment, Eqn. 4 is applied online to pick the right set of parameters for G. π_D, as a special case of π, does not consider θ_{t-1} as part of s_t (only x_t, line 7 Alg. 1), but consults a history of p steps of x_t for smoothness.

Algorithm 2 $LearnParameterPolicy$ (APPLD)

1: **Input:** $\mathcal{I} = \mathcal{D} = \{x_i^D, a_i^D, t_i^D\}_{i=1}^N$, Θ, G.
2: Call A_{segment} on $\mathcal{D}$ to detect changepoints $\tau_1, \ldots, \tau_{K-1}$ with $\tau_0 = 0$ and $\tau_K = N+1$.
3: Segment D into $\{\mathcal{D}_k = \{x_i^D, a_i^D, t_i^D \,|\, \tau_{k-1} \leq i < \tau_k\}\}_{k=1}^K$.
4: Train a classifier f_ϕ on $\{x_i^D, c_i\}_{i=1}^N$, where $c_i = k$ if $x_i^D \in \mathcal{D}_k$.
5: **for** $k = 1 : K$ **do**
6: Call $A_{\text{black-box}}$ with objective defined in Eqn. 2 on $\mathcal{D}_k$ to find parameters θ_k^* for context k.
7: **end for**
8: Form the map $M(k) = \theta_k^*$, $\forall 1 \leq k \leq K$ and context predictor $B_\phi(x)$.

B. Experiments

We implement APPLD on a physical robot[1] to experimentally validate that using the learned parameter library and context predictor from a teleoperated demonstration can achieve better navigation performance in complex environments compared to that obtained by the underlying navigation system using *(a)* its default parameters (DEFAULT) from the robot platform manufacturer, and *(b)* parameters we found using behavior cloning but without context (APP. (NO CTX.)).

A four-wheeled, differential-drive, unmanned ground vehicle, a ClearPath Jackal, is tasked to move through a custom-built maze (Fig. 2). The Jackal is a small and agile platform with a top speed of 2.0m/s. To leverage this agility, the complex maze consists of four qualitatively different areas: *(i)* a pathway with curvy walls (curve), *(ii)* an obstacle field, *(iii)* a narrow corridor, and *(iv)* an open space (Fig. 2). A Velodyne LiDAR provides 3D point cloud data, which is transformed into 2D laser scan for 2D navigation. The Jackal runs Robot Operating System (ROS) onboard, and APPLD is applied to the local planner, DWA [2], in the commonly-used `move_base` navigation stack. Other parts of the navigation stack, e.g. global planning with Dijkstra's algorithm, remain intact.

Teleoperation commands are captured via an Xbox controller from one of the authors with previous experience with video games, who is unfamiliar with the inner workings of the DWA planner and attempts to operate the platform to quickly traverse the maze in a safe manner. The teleoperator follows the robot and controls the robot from a third person view. The 52s demonstration is recorded using `rosbag` configured to record all joystick commands and all inputs to the `move_base` node.

We use CHAMP as A_{segment} (Line 2 Alg. 2), a state-of-the-art Bayesian segmentation algorithm [21]. The recorded LiDAR range data statistics (mean and standard deviation) from x_i^D and the recorded demonstrated actions $a_i^D = (v_i^D, w_i^D)$ are provided as input to CHAMP. As expected, CHAMP determines $K = 4$ segments in the demonstration, each corresponding to a different context (Line 3). f_ϕ trained for online context prediction is modeled as a two-layer neural network with ReLU activation functions (Line 4).

For the purpose of finding θ_k^* for each context, the recorded input is played to a ROS `move_base` node using DWA as the local planner with query parameters θ and the resulting output navigation commands are compared to the demonstrator's actions. For the metric in Eqn. 2, we use mean-squared error, i.e. H is the identity matrix. We find each θ_k^* using CMA-ES [22] as our black-box optimizer (Line 6 Alg. 2). θ

[1]Extra experimental results on a different robot with a different navigation system in a different environment can be found in the conference version of the APPLD paper [19].

TABLE I: Loss and Time Comparison of Jackal Experiments

Context	BC Loss	Real-world Time (s)
(a) Curve		
Demonstration	N/A	12.10
DEFAULT	0.1755±0.0212	30.20±3.87
APP. (NO CTX.)	0.1856±0.0030	*55.14±13.84
APPLD	**0.0780±0.0002**	**9.39±0.73**
(b) Obstacle Field		
Demonstration	N/A	9.00
DEFAULT	0.2061±0.0540	12.32±0.59
APP. (NO CTX.)	0.2537±0.0083	*60.00±0.00
APPLD	**0.1586±0.0216**	**7.69±0.35**
(c) Narrow Corridor		
Demonstration	N/A	24.06
DEFAULT	0.0953±0.0916	*49.52±19.88
APP. (NO CTX.)	0.0566±0.0419	*60.00±0.00
APPLD	**0.0198±0.0010**	**19.11±0.82**
(d) Open Space		
Demonstration	N/A	7.28
DEFAULT	0.8597±0.0013	15.07±0.61
APP. (NO CTX.)	0.2094±0.0013	15.08±7.42
APPLD	**0.2071±0.0021**	**7.19±0.42**

includes DWA's *max_vel_x* (v), *max_vel_theta* (w), *vx_samples* (s), *vtheta_samples* (t), *occdist_scale* (o), *pdist_scale* (p), *gdist_scale* (g) and costmap's *inflation_radius* (i).

Table I shows the results of evaluating the overall navigation system in the training environment using the different parameter-selection schemes (with ROS `dynamic_reconfigure` client) along with the demonstrator's performance as a point of reference. We report both the time it takes for each system to navigate a pre-specified route and also the BC loss (Eqn. 2) compared to the demonstrator. We choose to study traversal time since most suboptimal navigation behavior will cause stop-and-go motions, induce recovery behaviors, cause the robot to get stuck, or collide with obstacles (termination) – each of which will result in a higher traversal time.

For each metric, lower is better, and we compute mean and standard deviation over 10 independent trials. For trials that end in failure (e.g., the robot gets stuck), we add an asterisk (*) to the reported results and use penalty time value of 60s. The results show that, for every context, APPLD achieves the lowest BC loss and fastest real-world traversal time, compared to DEFAULT and APPLD (NO CONTEXT). In fact, while APPLD is able to successfully navigate in every trial, DEFAULT fails in 8/10 trials in the narrow corridor due to collisions in `recovery_behaviors` after getting stuck, and APPLD (NO CONTEXT) fails in 9/10, 10/10, and 10/10 trials in curve, obstacle field, and narrow corridor, respectively. In open space, APPLD (NO CONTEXT) is able to navigate quickly at first, but is not able to precisely and quickly reach the goal due to low angular sample density (*vtheta_samples*). Surprisingly, in all contexts, the navigation stack with APPLD parameters even outperforms the human demonstration in terms of time, and leads to qualitatively smoother motion than the demonstration. Average overall traversal time from start to goal, 43s, is also faster than the demonstrated 52s.

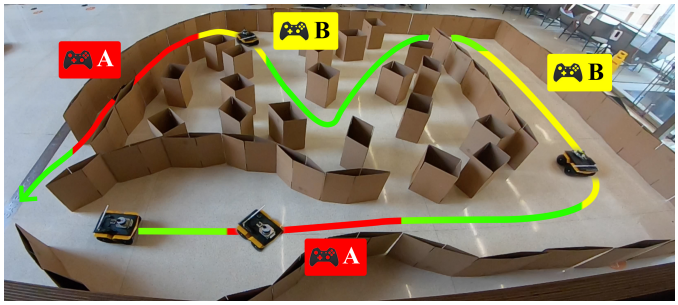

Fig. 3: While classical navigation systems perform well in most places (green), they may fail (red) or suffer from suboptimal behavior (yellow) in others. APPLI learns from human interventions in these two scenarios (Type A and Type B).

The superior performance achieved by APPLD compared to DEFAULT and even the demonstrator validates that with the learned parameter library and context predictor from a teleoperated demonstration, tuning DWA navigation parameters is possible without a roboticist. The fact that APPLD outperforms APPLD (NO CONTEXT) indicates the necessity of the high-level context switch strategy.

V. APPL FROM INTERVENTIONS (APPLI)

APPLD assumes human demonstration is a good approximation for optimal navigation behavior and default parameters do not work well in most places, which are not always the case. In *Adaptive Planner Parameter Learning from Interventions* (APPLI) [23], instead of providing a full demonstration of the entire navigation task (such as APPLD), non-expert users can easily identify failure or suboptimal cases by watching and then provide a few teleoperated *interventions* to correct the failure or suboptimal behaviors (Fig. 3). APPLI utilizes this human interaction modality and learns sets of planner parameters specifically for the scenarios where failure and suboptimal behaviors take place, to create a *parameter library* including the default parameters. During deployment, APPLI applies those learned parameters, only when it is confident that they will benefit the current navigation based on a confidence measure of the context predictor, to the underlying navigation system in those troublesome places, while maintaining good performance in others by switching back to the default parameters. This confidence measure also enables APPLI to generalize well to unseen environments.

A. Learning APPLI Policy

APPLI assumes a default parameter set $\bar{\theta}$ is tuned by a human designer trying to achieve good performance in most environments. However, being good at everything often means being great at nothing: $\bar{\theta}$ usually exhibits suboptimal performance in some situations and may even fail (is unable to find feasible motions, or crashes into obstacles) in particularly challenging ones.

1) Interventions as Contexts: To mitigate this problem, a human can supervise the navigation system's performance at state x by observing its action a and judging whether she

should intervene. Here, we consider two types of interventions. A Type A intervention is one in which the system performs so poorly that the human *must* intervene (e.g. imminent collision or a signal for help). A Type B intervention is one in which a human *might* intervene in order to improve otherwise suboptimal performance (e.g., driving too slowly in an open space). For the i^{th} intervention, we assume that the human resets the robot to the position where the failure or suboptimal behavior first occurred and then gives a short teleoperated intervention $I_i = \{x_t, a_t\}_{t=1}^{T_i}$ of length T_i, where $x_{1:T_i}$ is the trajectory starting from the reset state induced by intervention actions $a_{1:T_i}$. As this short demonstration naturally shows a cohesive navigation behavior in a specific segment of the environment (open space, narrow corridor, etc), it automatically forms a *context* $c_i \in \mathcal{C}$. Using human interaction composed of N interventions $\mathcal{I} = I_{1:N}$ instead of the full demonstration sequence $\mathcal{D} = \{x_i^D, a_i^D, t_i^D\}_{i=1}^N$ (Sec. IV), APPLI finds the mapping $M : \mathcal{C} \to \Theta$ that determines the parameter set θ_i for each intervention context c_i, and the parameterized context predictor $B_\phi : \mathcal{X} \to \mathcal{C}$ that determines to which context (if any) the current state x belongs. After collecting a set of N interventions $I_{1:N}$, for each I_i, we learn a set of navigation parameters θ_i that can best imitate the demonstrated correction with the same Behavior Cloning loss (Eqn. 2) with a black-box optimizer, such as CMA-ES [22]. After identifying parameters in each context, the mapping M is simply $M(i) = \theta_i$.

2) Confidence-Based Context Prediction: In contrast to APPLD, APPLI does not learn parameters for places where the original navigation performance is good. Therefore, it determines if the current state x_t falls into any one of the collected intervention contexts c_i. If so, APPLI directs the robot to use the parameter set θ_i to avoid making the same mistake as before. If not, APPLI directs the robot to use the default parameters $\bar{\theta}$, as they are optimized for most cases and are expected to generalize better than any parameter set learned for a specific scenario. In our system, the determination above is made using a new context predictor, B_ϕ, with confidence measure. To train this predictor, we build a similar dataset as in APPLD, $\{\{x_t, c_i\}_{t=1}^{T_i}\}_{i=1}^N$, and train an intermediate classifier $f_\phi(x)$ with parameter set ϕ using the Evidential Deep Learning method (EDL) [24]. A feature of EDL is that it supplies both a predicted label and a confidence measure, $u_i \in (0, 1]$, i.e.,

$$f_\phi(x_i) = (c_i, u_i). \tag{5}$$

After training f_ϕ and during deployment, we build a confidence-based classifier g_ϕ as

$$g_\phi(x_i) = c_i \mathbb{1}(u_i \geq \epsilon_u), \tag{6}$$

where ϵ_u is a threshold on confidence and $\mathbb{1}$ is the indicator function. For state x_i, g_ϕ determines its context from $N + 1$ contexts (N intervention contexts and one default context). If $u_i \geq \epsilon_u$, it suggests the classifier f_ϕ is confident and g_ϕ predicts c_i. Otherwise, when f_ϕ is unsure about its prediction, $c_i \mathbb{1}\{u_i \geq \epsilon_u\} = 0$. In this case, g_ϕ believes the current state x_i is not similar to any intervention context and instead classifies x_i as the default context. For this default context labeled as $c_i = 0$, navigation utilizes the default navigation parameters $\bar{\theta}$ (i.e., we set $M(0) = \bar{\theta}$). The new confidence-based context predictor B_ϕ is then defined as:

$$B_\phi(x_t) = \text{mode}\Big\{g_\phi(x_i),\ t - p < i \leq t\Big\}. \tag{7}$$

Again, B_ϕ acts as a mode filter and chooses the context c_t that the majority agrees with over the past p time steps.

The *LearningParameterPolicy* subroutine in Alg. 1 for APPLI is shown in Alg. 3. APPLI learns a confidence-based classifier (Line 2) and navigation parameters $\theta_{1:N}$ for each context (Lines 3-5). During deployment, we use $\theta_t = \pi_I(x_t) = M(B_\phi(x_t))$ to select the parameters for the navigation system at time t. Similar to π_D, π_I does not consider θ_{t-1} as part of s_t (only x_t, line 7 Alg. 1), but consults a history of p steps of x_t for smoothness.

Algorithm 3 *LearningParameterPolicy* (APPLI)

1: **Input:** $\mathcal{I} = I_{1:N} = \{\{x_t, a_t\}_{t=1}^{T_i}\}_{i=1}^N$, Θ, G.
2: Train a confidence-based classifier f_ϕ on $\{\{x_t, c_i\}_{t=1}^{T_i}\}_{i=1}^N$.
3: **for** $i = 1, \ldots, N$ **do**
4: Find parameter θ_i for context i using Eqn. 2 on I_i.
5: **end for**
6: Form the map $M(i) = \theta_i$, $\forall 1 \leq i \leq N$, and confidence-based context predictor $B_\phi(x)$.

B. Experiments

In our experiments, we aim to show that APPLI can improve navigation performance by learning from only a few interventions and, with the confidence measurement, that the overall system can generalize well to unseen environments. We apply APPLI on the same ClearPath Jackal ground robot with the same setup as in APPLD (Section IV-B) in a physical obstacle course. Navigation performance learned through APPLI is then tested both in the same training environment, and also in another unseen physical test course, which is qualitatively similar to the training environment (i.e., similar contexts were created in a different ordering). Furthermore, to investigate generalizability, we test the learned systems on a benchmark suite of 300 unseen simulated navigation environments [25].

During data collection, one of the authors (the *intervener*) follows the robot through the test course and intervenes when necessary, reporting if the intervention is to drive the robot out of a failure case (Type A) or to correct a suboptimal behavior (Type B). The four interventions are shown in Fig. 3: before the two Type A interventions (shown in red), the default system (DWA with $\bar{\theta}$) fails to plan feasible motions and starts recovery behaviors (rotates in place and moves backward); before the two Type B interventions (shown in yellow), the robot drives unnecessarily slowly in a relatively open space and enters the narrow corridor with unsmooth motions. For every intervention, the intervener stops the robot, drives it back to where they deem the failure or suboptimal behavior to have begun, and then provides recorded teleoperation I that avoids the problematic behavior. To compare the performance learned from interventions and learned from a full demonstration, we

TABLE II: APPLI Traversal Time

	Default	Type A	Type A+B	Full Demo
Training	134.0±60.6s	77.4±2.8s	70.6±3.2s	78.0±2.7s
Unseen	109.2±50.8s	71±0.7s	59±0.7s	62.0±2.0s

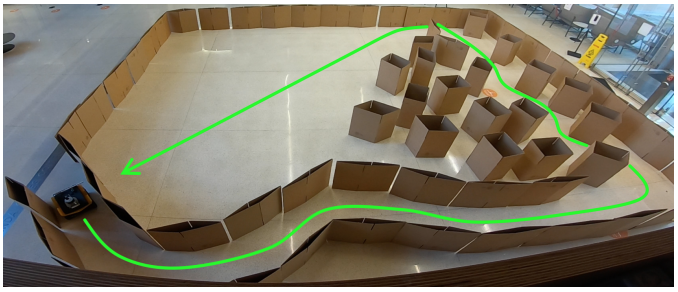

Fig. 4: APPLI Running in an Unseen Physical Environment

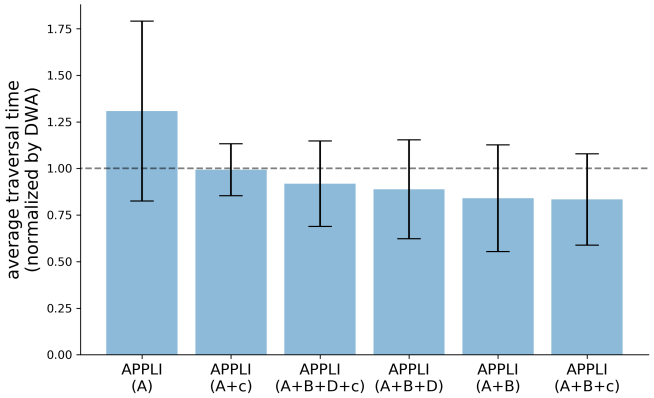

Fig. 5: Normalized Performance in 300 Simulation Environments from 12 Runs Each. Error bars: standard deviation.

also collect extra demonstrations for those places where the default planner already works well (shown in green in Fig. 3).

1) Physical Experiments: After training, we deploy π_I with learned mapping M and context predictor B_ϕ on the `move_base` navigation stack G with threshold $\epsilon_u = 0.8$.

We first deploy APPLI in the same training physical environment (Fig. 3). We compare the performance of the DWA planner with default parameters, APPLI learned only with Type A interventions, APPLI learned with Type A *and* Type B interventions, and APPLI learned with a *full demonstration* (which is basically APPLD enhanced by manual context segmentation and the confidence measure). The motivation for the variation of APPLI learned only with Type A interventions is to study the effect of an unfocused or inexperienced human intervener. In this case, the human would still conduct all Type A interventions, as those mistakes are severe and easy to identify—some robots may even actively ask for help (e.g. by starting recovery behaviors). However, the human may fail to conduct Type B interventions as she is not paying attention, or isn't equipped with the knowledge to identify suboptimal behaviors. For each method, we run five trials and report the mean and standard deviation of the traversal time in Tab. II 1st row. If the robot gets stuck, we introduce a penalty value of 200 seconds. We also deploy the same sets of variants in an unseen physical environments (Fig. 4 and Tab. II 2nd row).

For both the training and unseen environments, Type A interventions alone significantly improve upon the default parameters, by correcting all recovery behaviors such as rotating in place or driving backwards, and eliminating all failure cases. Adding Type B interventions further reduces traversal time, since the robot learns to speed up in relatively open spaces and to execute smooth motion when the tightness of the surrounding obstacles changes. All the interventions are able to improve navigation in both training and unseen environments, suggesting APPLI's generalizability. Surprisingly, in both environments, APPLI learned from only Type A and Type B interventions can even outperform APPLI learned from an entire demonstration. One possible reason for this better performance from fewer human interactions is the additional human demonstrations may be suboptimal, especially since they are collected in places where the default navigation system was already deemed to have performed well. For example, in the full demonstration, we find the human intervener is more conservative than the default navigation system and drives slowly in some places. Hence, learning from these suboptimal behaviors introduces suboptimal parameters and consequently worse performance in contexts similar to that intervention.

2) Simulated Experiments: To further test APPLI's generalizability to unseen environments, we test our method and compare it with two baselines on the Benchmark for Autonomous Robot Navigation (BARN) dataset [25]. The benchmark dataset consists of 300 simulated navigation environments randomly generated using Cellular Automata, ranging from easy ones with a lot of open spaces to challenging ones where the robot needs to get through dense obstacles. Using the same training data collected from the physical environment shown in Fig. 3, we test the following seven variants: (1) APPLI (A+B+c): APPLI learned from *Type A and B* inventions *with* confidence measure, (2) APPLI (A+B): APPLI learned from *Type A and B* inventions *without* confidence measure, (3) APPLI (A+c): APPLI learned from only *Type A* interventions *with* confidence measure, (4) APPLI (A): APPLI learned from only *Type A* interventions *without* confidence measure, (5) APPLI (A+B+D+c): APPLI learned from *full demonstration with* confidence measure, (6) APPLI (A+B+D): APPLI learned from *full demonstration without* confidence measure, (7) the DWA planner with default parameters.

Testing these variations aims at studying the effect of learning from different modes of interventions caused by different degrees of human attention and experience levels, i.e., imperative interventions (A), optional interventions (A + B), and a full demonstration (A + B + D). They also provide an ablation study for the confidence measure in the EDL context classifier f_ϕ: when deployed without the confidence measure, the robot has to choose among the parameters learned from interventions and never uses the default parameters.

For each method in each simulation environment, we measure the traversal time for 12 different runs, resulting in 25200 total navigation trials. The average traversal time for each

method in all simulation environments is normalized by DWA and is shown in order of increasing performance in Fig. 5.

APPLI (A+B+c) and APPLI (A+B+D+c) outperform DWA: their average traversal time is shorter than that of DWA by 8% and 17% respectively. However, for APPLI (A+c), its traversal time is only 1% better than DWA, which suggests that even though type B inventions only correct suboptimal performances, they are crucial for performance improvement.

In terms of the effect of confidence, APPLI (A) only selects parameters learned from 2 Type A inventions and never uses the default parameters even when they are more appropriate. Removing confidence greatly harms its performance, making its traversal time even longer than DWA by 31%. However, APPLI (A+B+c) and APPLI (A+B+D+c), which use more interventions or even the full demonstration to train the parameter mapping M and context predictor B_ϕ, are more confident about their predictions most of the time. As a result, removing confidence in the context predictor doesn't result in a significant difference.

Lastly, a counterintuitive, but similar result as in the physical experiments is that APPLI (A+B+c) learned from only Type A and B interventions achieves shorter traversal time than APPLI (A+B+D+c) and APPLI (A+B+D) by 9.2% and 6.1%, respectively. Similar to the discussions about physical experiments, unnecessary human demonstrations are most likely suboptimal. In this sense, APPLI not only reduces the required human interactions from a full demonstration to only a few interventions, but also reduces the chance of performance degradation caused by suboptimal demonstrations.

VI. APPL FROM EVALUATIVE FEEDBACK (APPLE)

APPLD and APPLI require non-expert users to take full control of the moving robot with a joystick, which some non-expert users may not feel comfortable with, e.g., due to perceived risk of human error and causing collisions. For these users, we introduce *Adaptive Planner Parameter Learning from Evaluative Feedback* (APPLE) [26], where they only need to observe the robot navigating and provide real-time positive or negative assessments of the observed navigation behavior through *evaluative feedback*. This more-accessible modality provides a new interaction channel for a larger community of non-expert users with mobile robots (Fig. 6).

Fig. 6: For non-expert users who are unable or unwilling to take control of the robot, evaluative feedback, e.g. *good job* (green thumbs up) or *bad job* (red thumbs down), is a more accessible human interaction modality, but still valuable for improving navigation systems during deployment.

A. Learning APPLE Policy

APPLE learns a parameter policy from human evaluative feedback with the goal of selecting the appropriate parameter set θ (from either a discrete parameter set library or from a continuous full parameter space) for the current deployment environment. In detail, a human can supervise the navigation system's performance at state x by observing its action a and giving corresponding evaluative feedback e. Here, the evaluative feedback can be either discrete (e.g., "good job"/"bad job") or continuous (e.g., a score ranging in $[0, 1]$). During feedback collection, APPLE finds (1) a parameterized predictor $F_\phi : \mathcal{X} \times \Theta \to \mathcal{E}$ that predicts human evaluative feedback for each state-parameter pair (x, θ), and (2) a parameterized parameter policy $\pi_\psi : \mathcal{X} \to \Theta$ that determines the appropriate planner parameter set for the current state.

1) Discrete Parameter Policy: In some situations, the user may already have K candidate parameter sets (e.g., the default set or sets tuned for specific environments like narrow corridors, open spaces, etc.) which together make up a parameter library $\mathcal{L} = \{\theta^i\}_{i=1}^K$ (superscript i denotes the index in the library). In this case, APPLE uses the provided evaluative feedback e in order to learn a policy that selects the most appropriate of these parameters given the state observation x.

To do so, we parameterize the feedback predictor F_ϕ in a way similar to the value network in DQN [27], where the input is the observation x and the output is K predicted feedback values $\{\hat{e}^i\}_{i=1}^K$, one for each parameter set θ^i in the library $\mathcal{L}$, as a prediction of the evaluative feedback a human user would give if the planner were using the respective parameter set at state x. We form a feedback dataset for supervised learning, $\mathcal{F} = \{x_j, \theta_j, e_j\}_{j=1}^N$ ($\theta_j \in \mathcal{L}$, subscript j denotes the time step) using the evaluative feedback collected so far, and F_ϕ is learned via supervised learning to minimize the difference between predicted feedback and the label,

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \underset{(x_j,\theta_j,e_j)\sim\mathcal{F}}{\mathbb{E}} \ell(F_\phi(x_j, \theta_j), e_j) \tag{8}$$

where $\ell(\cdot, \cdot)$ is the binary cross entropy loss if the feedback e_j is discrete (e.g., $e_j = 1$ for "good job", and $e_j = 0$ for "bad job"), or mean squared error given continuous feedback.

To achieve the best possible performance, the parameter policy $\pi(\cdot|x)$ chooses the parameter set that maximizes the expected human feedback (the discrete parameter policy doesn't require any additional parameters beyond ϕ for F, so the ψ is omitted here for simplicity). More specifically,

$$\pi(\cdot|x) = \underset{\theta\in\mathcal{L}}{\operatorname{argmax}} F_{\phi^*}(x, \theta). \tag{9}$$

2) Continuous Parameter Policy: If a discrete parameter library is not available or desired, APPLE can also be used over continuous parameter spaces (e.g., deciding the max speed from $[0.1, 2]$ m/s). In this scenario, we assume the human will provide continuous evaluative feedback, as we find that discrete feedback is not informative enough to learn a continuous policy efficiently.

In this setting, we parameterize the parameter policy π_ψ and the feedback predictor F_ϕ in the actor-critic style. With the collected evaluative feedback $\mathcal{F} = \{x_j, \theta_j, e_j\}_{j=1}^N$, the

training objective of F_ϕ is still to minimize the difference between predicted and collected feedback, as specified by Eqn. 8. For the parameter policy π_ψ, beyond choosing the action that maximizes expected feedback, its training objective is augmented by maximizing the entropy of policy $\mathcal{H}(\pi_\psi(\cdot|x))$ at state x. Using the same entropy regularization as Soft Actor Critic (SAC) [28], π_ψ favors more stochastic policies, leading to better exploration during training:

$$\psi^* = \underset{\psi}{\operatorname{argmin}} \underset{\substack{x_j \in \mathcal{F} \\ \tilde{\theta}_j \sim \pi_\psi(\cdot|x_j)}}{\mathbb{E}} \left[-F_\phi(x_j, \tilde{\theta}_j) + \alpha \log \pi_\psi(\tilde{\theta}_j|x_j) \right], \tag{10}$$

where α is the temperature controlling the importance of the entropy bonus and is automatically tuned as in SAC [28].

The $LearningParameterPolicy$ subroutine in Alg. 1 for APPLE to learn π_E is simply learning F_{ϕ^*} with Eqn. 8 and learning π_{ψ^*} with SAC [28] for the discrete and continuous parameter policy, respectively.

B. EXPERIMENTS

In our experiments, we find that APPLE can improve navigation performance by learning from evaluative feedback, in contrast to a teleoperated demonstration or a few corrective interventions, both of which require the non-expert user to take control of the moving robot. We also show APPLE's generalizability to unseen environments. We implement APPLE on the same ClearPath Jackal ground robot in BARN [25], and in two physical obstacle courses. For discrete APPLE, we construct the parameter library with the default DWA parameter set θ_1 and parameter sets $\theta_{2\sim7}$ learned in APPLI [23]. $F_\phi(x, \theta)$ is a fully-connected neural network with 2 hidden layers of 128 neurons, taking the 720 dimensional laser scan as input x_t and outputting the 7 predicted feedback signals $\hat{e}_{t,1\sim7}$ for $\theta_{1\sim7}$ respectively. Parameter policy π_ψ uses ϵ-greedy exploration with ϵ decreasing from 0.3 to 0.02 during the first half of the training. For continuous APPLE, we manually define the parameter ranges for the parameter policy π_ψ to select from [26]. $F_\phi(x, \theta)$ shares the same architecture as discrete APPLE, except for different input (concatenation of x_t and θ_t) and output (scalar $\hat{e}_t$). The parameter policy π_ψ also uses the same architecture, mapping x_t to θ_t.

To evaluate the performance of APPLE, we use APPLI with the same parameter library and the Default DWA planner as two baselines. Despite using the same library, APPLI chooses the parameter set based on the similarity between the current observation and the demonstrated environments, while discrete APPLE uses the expected feedback.

1) Simulated Experiments: We begin by testing APPLE on the BARN dataset, with simulated evaluative feedback generated by an oracle (proxy human). We randomly select 250 environments for training APPLE and hold the remained 50 environments as the test set. For the simulated feedback, we use the projection of the robot's linear velocity along the local goal direction, i.e., $e_t = v_t \cdot \cos(g_t)$, and it greedily (thus suboptimally) encourages the robot to move along the global path as fast as possible. The oracle provides its evaluative feedback at 4Hz, while APPLE and APPLI dynamically adjust the parameter set for the DWA planner at the same frequency.

TABLE III: Percentage of Test Environments where Method 1 is Significantly Worse than Method 2 by Traversal Time
(Methods are listed in order of increasing performances. Best method when comparing with Method 1 is bold)

Method 1 \ Method 2	Default	APPLI	APPLE (disc.)	APPLE (cont.)
Default	0	23	**33**	**33**
APPLI	6	0	**29**	**29**
APPLE (disc.)	2	4	0	**27**
APPLE (cont.)	4	4	**10**	0

After training in 250 environments with a total of 12.5M feedback signals collected, we evaluate APPLE with discrete and continuous parameter policies (denoted as APPLE (disc.) and APPLE (cont.)), as well as two baselines, on the 50 test environments by measuring the traversal time for 20 runs per environment. We then conduct a pair-wise t-test for all methods in order to compute the percentage of environments in which one method (denoted as Method 1) is significantly worse ($p < 0.05$) than another (denoted as Method 2). For better illustration, we order the method by their performance and show the pairwise comparison in Tab. III.

Despite learning from suboptimal evaluative feedback, APPLE (disc.) and APPLE (cont.) still outperform the Default DWA and APPLI in 33% and 29% of test environments respectively, and they only perform significantly worse in a few cases. These results demonstrate the advantage of APPLE over APPLI, which selects the parameter set with a performance-based predictor (Eqns. 9 and 10) rather than a similarity-based predictor. APPLE (cont.) is much better than APPLE (disc.), because of its larger model capacity in the parameter space.

2) Physical Experiments: We also apply APPLE on a physical Jackal robot. In a highly-constrained obstacle course (Fig. 6), we first apply APPLI which provides 5 sets of parameters (1 default and 4 learned). Then to train APPLE which uses the same parameter library, one of the authors follows the robot autonomously navigating and uses an Xbox joystick to give evaluative feedback at 2Hz. To reduce the burden of giving a large amount of feedback, the user is only requested to give negative feedback by pressing a button on the joystick when he thinks the robot's navigation performance is bad, while for other instances, positive feedback is automatically given. In other words, we interpret the absence of human feedback to be the same as if the human had provided positive feedback. The entire APPLE training session lasts roughly 30 minutes, in which the robot navigates 10 trials in the environment shown in Fig. 6. APPLE learns in an online fashion with 30% probability of random exploration.

After the training, the learned APPLE model is deployed in the same training environment. We compare APPLE to APPLI with the same sets of parameters and the confidence measure of context prediction [23], and the DWA planner with static default parameters. Each experiment is repeated five times. The results are shown in Tab IV. APPLE achieves the fastest average traversal time with the smallest variance.

To test APPLE's generalizability, we also test APPLE in an unseen environment (Fig. 7) and show the results in Tab. IV. In the unseen environment, APPLE has slightly increased vari-

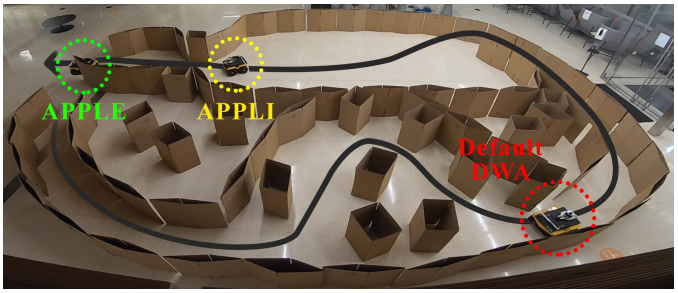

Fig. 7: APPLE Running in an unseen physical environment, which is qualitatively similar to the training environment.

TABLE IV: Traversal Time (Training and Unseen)

	Default	APPLI	APPLE (disc.)
Training	143.1±20.0s	79.8±8.1s	75.2±4.1s
Unseen	150.5±24.0s	86.4±1.1s	83.9±4.6s

ance, but still has the fastest average traversal time compared to the other two baselines.

VII. APPL FROM REINFORCEMENT (APPLR)

Adaptive Planner Parameter Learning from Reinforcement (APPLR) [29] adopts the general notion of *parameter policy* (Sec. III Fig. 1). One disadvantage of learning planner parameters from different human interaction modalities is that the learner's performance is limited by the human's (most likely suboptimal) teleoperated demonstration, corrective interventions, and evaluative feedback. With reinforcement learning in a wide variety of simulation environments, APPLR does *not* need access to any human interaction, and learns to make planner parameter decisions in such a way that allows the system to take suboptimal actions at one state in order to perform even better in the future.

A. Learning APPLR Policy

APPLR uses an existing RL algorithm and a reward function to learn a parameter policy π_R.

1) Reward Function: In general, we encourage three types of behaviors: (1) behaviors that lead to the global goal faster; (2) behaviors that make more local progress; and (3) behaviors that avoid collisions and danger. Correspondingly, the designed reward function can be summarized as

$$R_t(s_t, a_t, s_{t+1}) = c_f R_f + c_p R_p + c_c R_c. \tag{11}$$

Here, c_f, c_p, c_c are coefficients for the three types of reward functions R_f, R_p, R_c. Specifically, $R_f(s_t, a_t) = \mathbb{1}(s_t \text{ is terminal}) - 1$ applies a -1 penalty to every step before reaching the global goal. To encourage the local progress of the robot, we add a dense shaping reward R_p. Assume $\beta = (\beta_x, \beta_y) \in \mathbb{R}^2$ is the global goal and at time t, the absolute coordinates of the robot are $p_t = (p_t^x, p_t^y)$, we define

$$R_p = \frac{(p_{t+1} - p_t) \cdot (\beta - p_t)}{|\beta - p_t|}, \tag{12}$$

In other words, R_p denotes the robot's local progress $(p_{t+1} - p_t)$ projected on the direction toward the global goal $(\beta - p_t)$. Finally, a penalty for the robot colliding with or coming too close to obstacles is defined as $R_c = -1/d(p_{t+1})$, where $d(p_{t+1})$ is a distance function measuring how close the robot is to obstacles based on sensor observations (e.g., using LiDAR).

2) Reinforcement Learning Algorithm: For continuous actions and high sample efficiency, we use the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [30], an actor-critic algorithm that keeps an estimate for both the policy and two state-action value functions.

B. Experiments

Again, we use the same ClearPath Jackal and the same setup to validate that APPLR can enable adaptive autonomous navigation *without* access to expert tuning or human demonstrations and is generalizable to many deployment environments, both in simulation and in the real-world. The results of APPLR are compared with those obtained by the underlying navigation system using its default parameters from the robot platform manufacturer. For the physical experiments, we also compare to parameters learned from APPLD.

1) Training: x_t is composed of 720-dimensional laser scan (capped to 2m) and a relative goal angle, computed from a local goal taken from the global path 1m away from the robot. The meta-state s_t is composed of x_t and θ_{t-1}. APPLR learns a parameter policy π_R to select the same DWA parameters θ_t as other APPL methods. π_R is trained in simulation using 250 randomly selected training environments from the BARN dataset [25]. In each of the environments, the robot aims to navigate from a fixed start to a fixed goal location in a safe and fast manner. For the reward function, while R_f penalizes each time step before reaching the global goal, we simplified R_p by replacing it with its projection along the y-axis (longitudinal direction) of all BARN environments. The distance function in $d(p_{t+1})$ in R_c is the minimal value among the 720 laser beams. π_R produces a new set of planner parameters every two seconds.

This simulated navigation task is implemented in a Singularity container, which enables easy parallelization on a computer cluster. TD3 [30] is implemented to learn the parameter policy π_R in simulation. The policy network and the two target Q-networks are represented as multilayer perceptrons with three 512-unit hidden layers. The policy is learned under the distributed architecture Gorila [31]. The acting processes are distributed over 500 CPUs with each CPU running one individual navigation task. On average, two actors work on a given navigation task. A single central learner periodically collects samples from the actors' local buffers and supplies the updated policy to each actor. Gaussian exploration noise with 0.5 standard deviation is applied to the actions at the beginning of the training. Afterward the standard deviation linearly decays at a rate of 0.125 per million steps and stays at 0.02 after 4 million steps. The entire training process takes about 6 hours and requires 5 million transition samples in total.

2) Simulated Experiments: After training, we deploy the learned parameter policy π_R on both the 250 training and

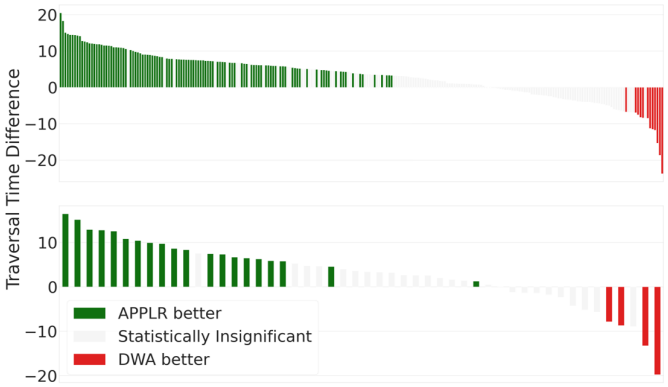

Fig. 8: Traversal time difference between APPLR and DWA for the training environments (top) and the test environments (bottom). The bars represent the environments ordered by traversal time difference. The colored bars indicate the environments that show statistically significant difference.

TABLE V: Average Traversal Time of APPLR and DWA

	APPLR	DWA	Improvement
Training	23.36	26.49	3.13 (11.8%)
Test	23.69	26.70	3.01 (11.2%)

50 test environments. We also use traversal time to evaluate the performance of the policy. A maximum traversal time of 50s is used, and the failing trials are set to be 50s plus a 20s penalty. We average over the traversal time of 40 trials for DWA and APPLR in each environment. We also perform t-tests for each pair of APPLR and DWA performance to check statistical significance. The results over the 250 training environments and 50 test environments are shown in Fig. 8. For the majority of the environments (green bars), APPLR shows statistically significantly better navigation performance. Tab. V shows the average traversal time of APPLR and DWA, and relative improvement. APPLR yields an improvement of 11.8% in the training environments, and 11.2% in the test environments. In addition, Tab. VI compares the number of environments that show significant improvement and deterioration. In both training and test set, APPLR achieves statistically significantly better navigation performance in over 40% of environments than DWA does, while DWA is only better in 4.8% and 8% of environments in the training and test set, respectively.

3) Physical Experiments: To validate the sim-to-real transfer of APPLR, we also test the learned parameter policy π_R on a physical Jackal robot. The learned policy is deployed in a real-world obstacle course (Fig. 9). This physical environment is different from any of the navigation environments in BARN. Therefore, both generalizability and sim-to-real transfer of

TABLE VI: Number and Percentage of All Environments in which One Method is Better Compared to the Other

	APPLR better	DWA better
Training	106 (42.4%)	12 (4.8%)
Test	20 (40%)	4 (8%)

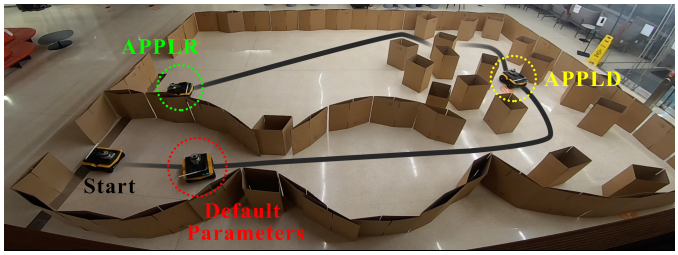

Fig. 9: APPLR Physical Experiments

TABLE VII: Traversal Time in Physical Experiments
(* denotes one additional failure trial)

DWA	APPLD	APPLR
72.8±10.1s*	43.2±4.1s	**34.4±4.8s**

APPLR can be tested with this unseen real-world environment. Note that the use of LiDAR input also reduces the difference between simulation and the real-world.

Given this target environment, we further collect a teleoperated demonstration provided by one of the authors and learn a parameter tuning policy based on the notion of navigational context (APPLD). The author aims at driving the robot to traverse the entire obstacle course in a safe and fast manner. APPLD identifies three contexts using the human demonstration and learns three sets of navigation parameters.

We compare the performance of APPLR with that of APPLD and the DWA planner using a set of hand-tuned default parameters. For each trial, the robot navigates from the fixed start point to a fixed goal point. Each trial is repeated five times and we report the mean and standard deviation in Table VII. In all DWA trials, the robot gets stuck in many places, especially where the surrounding obstacles are very tight. It has to engage in many recovery behaviors, i.e. rotating in place or driving backwards, to "get unstuck". Furthermore, in relatively open space, the robot drives unnecessarily slowly. All these behaviors contribute to the large traversal time and high variance (plus an additional failure trial). Unlike many simulation environments in BARN [25], where obstacles are generated by cellular automata and therefore very cluttered, the relatively open space in the physical environment (Fig. 9) allows faster speed and gives APPLR a greater advantage. Surprisingly, APPLR even achieves better navigation performance than APPLD, which has access to a human demonstration in the same environment. One of the reasons we observe this result in the physical experiments is that the human demonstrator is relatively conservative in some places; the parameters learned by APPLD are upper-bounded by this suboptimal human performance. Another reason is that APPLR is given the flexibility to continually change parameters, and RL is able to utilize the sequential aspect of the parameter selection problem, in contrast to APPLD's three sets of static parameters.

VIII. CYCLE-OF-LEARNING

The four individual APPL methods are combined to form a cycle-of-learning scheme [18], in which a mobile robot

TABLE VIII: Cycle-of-Learning Performance in Env. 1-4

Env.	DWA	APPLR1	APPLD	APPLI	APPLE	APPLR2
1	9.8 ±0.5s	**4.4 ±0.3s**				
2	30.2 ±1.4s	18.6 ±3.6s	**12.5 ±0.4s**			
3	56.9 ±2.5s	57.5 ±5.8s	38.1 ±2.3s	**37.0 ±2.0s**		
4	109.2 ±6.5s	103.4 ±12.8s	90.6 ±6.0s	88.8 ±2.9s	**76.4 ±3.5s**	81.6 ±5.6s

can interact with different users in different deployment environments and continually improve its navigation in a cyclic fashion. In this section, we present one full cycle (Fig. 10), which starts from APPLR, goes through APPLD, APPLI, and APPLE, and comes back to APPLR. To distinguish the start and end of the cycle, we name them APPLR1 and APPLR2.

Before any real-world deployment, we first train the APPLR1 policy on the BARN dataset. The first deployment environment is a relatively open space, where DWA slowly drives at the default 0.5m/s max velocity and APPLR1 is able to accelerate to 2.0m/s. Moving on to the second environment, APPLR1 produces unsmooth motion at the narrow gap. User 1 provides a full demonstration for environment 2, from which APPLD learns two contexts and corresponding parameters. When deployed in the third enviroment, APPLD gets stuck multiple times in the narrow corridor, where User 2 intervenes and uses APPLI to learn an extra context. In environment 4, APPLI suffers from suboptimal behaviors in the obstacle field. User 3 provides evaluative feedback and uses APPLE to improve the context predictor of APPLI. The performance of different APPL variants in the four environments is shown in Tab. VIII (averaged over 5 trials). The method which directly utilizes human interaction in the particular environment achieves the best results in the corresponding environment (bold).

After deploying in Environment 4 with APPLE, we close the cycle by training APPLR2 in the BARN dataset. We initialize the policy with APPLR1, identify the BARN environments where the APPL variants perform better than APPLR1, and use the learned APPLD, APPLI, and APPLE policies for exploration [2] with a probability which decreases from 1 to 0 in those environments. APPLR2 achieves significantly better performance in these environments due to the better exploration policy learned within the cycle from different human interaction modalities. We further test APPLR2's performance against APPLR1's in all BARN environments and do not observe significant deterioration in other environments. In fact, APPLR2's average traversal time in BARN improves from APPLR1's 24.7s to 24.0s. We also deploy APPLR2 in Environment 4 in Fig. 10. Although the training experience in BARN slightly increases traversal time compared to APPLE, APPLR2 significantly outperforms APPLR1 by incorporating all human interactions, including the learned context predictor and parameters, into a stand-alone parameter policy.

[2]Other techniques for combining policies may be possible, e.g., [32].

IX. CONCLUSIONS

We present APPL, a learning paradigm that leverages different human interaction modalities, including demonstration, interventions, evaluative feedback, and unsupervised reinforcement learning, to dynamically fine-tune classical navigation planner parameters to achieve better navigation performance. In addition to the individual APPL methods and experiments, we also introduce a cycle-of-learning scheme, in which different human interaction modalities can be utilized to continually improve future navigation performance in a cyclic fashion. One interesting direction for future work is to investigate other methods to incorporate human interaction to RL training in simulation, e.g., using inverse reinforcement learning to infer an underlying reward function, or creating high fidelity simulations similar to real-world deployment environments.

ACKNOWLEDGMENTS

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARO (W911NF19-2-0333), DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 802–807.

[2] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[3] K. Zheng, "Ros navigation tuning guide," *arXiv preprint arXiv:1706.09068*, 2017.

[4] X. Xiao, J. Dufek, T. Woodbury, and R. Murphy, "Uav assisted usv visual navigation for marine mass casualty incident response," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6105–6110.

[5] B. Liu, X. Xiao, and P. Stone, "Lifelong navigation," *arXiv preprint arXiv:2007.14486*, 2020.

[6] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion control for mobile robot navigation using machine learning: a survey," *arXiv preprint arXiv:2011.13112*, 2020.

[7] J. Liang, U. Patel, A. J. Sathyamoorthy, and D. Manocha, "Crowd-steer: Realtime smooth and collision-free robot navigation in dense crowd scenarios trained using high-fidelity simulation," *arXiv preprint arXiv:2004.03089*, 2020.

[8] X. Xiao, J. Biswas, and P. Stone, "Learning inverse kinodynamics for accurate high-speed off-road navigation on unstructured terrain," *arXiv preprint arXiv:2102.12667*, 2021.

[9] S. Thrun, "An approach to learning mobile robot navigation," *Robotics and Autonomous systems*, vol. 15, no. 4, pp. 301–319, 1995.

[10] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 ieee international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.

[11] X. Xiao, B. Liu, and P. Stone, "Agile robot navigation through hallucinated learning and sober deployment," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[12] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Toward agile maneuvers in highly constrained spaces: Learning from hallucination," *arXiv preprint arXiv:2007.14479*, 2020.

Fig. 10: One Learning Cycle. White Arrows: Learning in the Real-World. Black Arrow: Learning in Simulation.

[13] Z. Wang, X. Xiao, A. J. Nettekoven, K. Umasankar, A. Singh, S. Bommakanti, U. Topcu, and P. Stone, "From agile ground to aerial navigation: Learning from learned hallucination."

[14] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[15] S. Siva, M. Wigness, J. Rogers, and H. Zhang, "Robot adaptation to unstructured terrains by joint representation and apprenticeship learning," in *Robotics: Science and Systems (RSS)*, 2019.

[16] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2462–2470.

[17] W. B. Knox, P. Stone, and C. Breazeal, "Training a robot via human feedback: A case study," in *International Conference on Social Robotics*. Springer, 2013, pp. 460–470.

[18] N. R. Waytowich, V. G. Goecks, and V. J. Lawhern, "Cycle-of-learning for autonomous systems from human interaction," in *AI-HRI Symposium, AAAI Fall Symposium Series*, 2018.

[19] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "Appld: Adaptive planner parameter learning from demonstration," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4541–4547, 2020.

[20] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[21] S. Niekum, S. Osentoski, C. G. Atkeson, and A. G. Barto, "Online bayesian changepoint detection for articulated motion models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1468–1475.

[22] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[23] Z. Wang, X. Xiao, B. Liu, G. Warnell, and P. Stone, "Appli: Adaptive planner parameter learning from interventions," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[24] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," in *Advances in Neural Information Processing Systems*, 2018, pp. 3179–3189.

[25] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2020, pp. 116–121.

[26] Z. Wang, X. Xiao, G. Warnell, and P. Stone, "Apple: Adaptive planner parameter learning from evaluative feedback."

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[29] Z. Xu, G. Dhamankar, A. Nair, X. Xiao, G. Warnell, B. Liu, Z. Wang, and P. Stone, "Applr: Adaptive planner parameter learning from reinforcement," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[30] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[31] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," 2015.

[32] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments," in *International Conference on Autonomous Agents and MultiAgent Systems*, 2020.