In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA 2021)* (icra 2021), Xi'an China June 2021

APPLR: Adaptive Planner Parameter Learning from Reinforcement

Zifan Xu[1], Gauraang Dhamankar[2], Anirudh Nair[3], Xuesu Xiao[2], Garrett Warnell[2,4], Bo Liu[2], Zizhao Wang[5], and Peter Stone[2,6]

Abstract— **Classical navigation systems typically operate using a fixed set of hand-picked parameters (e.g. maximum speed, sampling rate, inflation radius, etc.) and require heavy expert re-tuning in order to work in new environments. To mitigate this requirement, it has been proposed to learn parameters for different contexts in a new environment using human demonstrations collected via teleoperation. However, learning from human demonstration limits deployment to the training environment, and limits overall performance to that of a potentially-suboptimal demonstrator. In this paper, we introduce APPLR, *Adaptive Planner Parameter Learning from Reinforcement*, which allows existing navigation systems to adapt to new scenarios by using a parameter selection scheme discovered via reinforcement learning (RL) in a wide variety of simulation environments. We evaluate APPLR on a robot in both simulated and physical experiments, and show that it can outperform both a fixed set of hand-tuned parameters and also a dynamic parameter tuning scheme learned from human demonstration.**

I. INTRODUCTION

Most classical autonomous navigation systems are capable of moving robots from one point to another, often with verifiable collision-free guarantees, under a set of parameters (e.g. maximum speed, sampling rate, inflation radius, etc.) that have been fine-tuned for the deployment environment. However, these parameters need to be re-tuned to adapt to different environments, which requires extra time and energy spent onsite during deployment, and more importantly, expert knowledge of the inner workings of the underlying navigation system [1], [2].

A recent thrust to alleviate the costs associated with expert re-tuning in new environments is to learn an adaptive parameter tuning scheme from demonstration [3]. Although this approach removes the requirement of expert tuning, it still depends on access to a human demonstration, and the learned parameters are typically only applicable to the training environment. Moreover, the performance of the system is limited by the quality of the human demonstration, which may be suboptimal.

In this paper, we seek a new method for adaptive autonomous navigation which does *not* need access to expert tuning or human demonstration, and is generalizable to many deployment environments. We hypothesize that a method based on reinforcement learning in simulation could achieve these goals, and we verify this hypothesis by proposing and studying *Adaptive Planner Parameter Learning from Reinforcement* (APPLR). By using reinforcement learning, APPLR introduces the concept of a *parameter policy* (Fig. 1), which is trained to make planner parameter decisions in such a way that allows the system to take suboptimal actions at one state in order to perhaps perform even better in the future. For example, while it may be suboptimal in the moment to slow down or alter the platform's trajectory before a turn, doing so may allow the system to carefully position itself so that it can go much faster in the future than if it had not. Additionally, as opposed to an end-to-end motion policy (i.e., a mapping from states to low-level motion commands), APPLR's parameter policy interacts with an underlying classical motion planner, and therefore the overall system inherits all the benefits enjoyed by classical approaches (e.g., safety and explainability). We posit that learning policies that act in the parameter space of an existing motion planner instead of in the velocity control space can increase exploration safety, improve learning efficiency, generalize well to unseen environments, and allow effective sim-to-real transfer.

II. RELATED WORK

In this section, we summarize related work on existing parameter tuning approaches for classical navigation and on learning-based navigation systems.

A. Parameter Tuning

Classical navigation systems usually operate under a static set of parameters. Those parameters are manually adjusted to near-optimal values based on the specific deployment environment, such as high sampling rate for cluttered environments or high maximum speed for open space. This process is commonly known as *parameter tuning*, which requires robotics experts' intuition, experience, or trial-and-error [1], [2]. To alleviate the burden of expert tuning, automatic tuning systems have been proposed, such as those using fuzzy logic [4] or gradient descent [5], to find *one* set of parameters tailored to the specific navigation scenario.

Department of [1]Physics zfxu@utexas.edu, [2]Computer Science {dgauraang, xiao, bliu, pstone}@cs.utexas.edu, [3]Mathematics ani.nair@utexas.edu, [5]Electrical and Computer Engineering zizhao.wang@utexas.edu, University of Texas at Austin, Austin, Texas 78712. [4]Computational and Information Sciences Directorate, Army Research Laboratory, Austin, Texas 78712 garrett.a.warnell.civ@mail.mil. [6]Sony AI. 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 (W911NF-19-2-0333), 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.

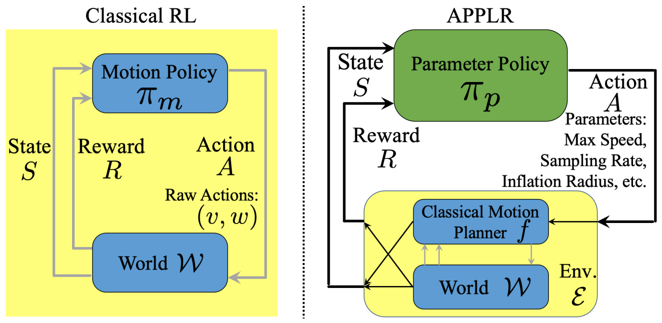

Fig. 1: Instead of learning an end-to-end motion policy π_m which takes state S and reward R from the world $\mathcal{W}$ and produces raw actions A, e.g. linear and angular velocity (v, ω) (left), APPLR treats an underlying classical motion planner f as part of the *meta-environment* $\mathcal{E}$ (along with the world $\mathcal{W}$) and the learned *parameter policy* π_p interacts with it through actions in the *parameter* space (right). In this way, the RL agent selects its action in the form of a set of navigation parameters at *each time step* and reasons about potential *future* consequences of those parameters, rather than *tuning* a single set of parameters for the *entire environment* only considering the *current* situation.

Recently, Xiao *et al.* introduced Adaptive Planner Parameter Learning from Demonstration (APPLD), which learns a *library* of parameter sets for different navigation *contexts* from teleoperated demonstration, and dynamically tunes the underlying navigation system during deployment. APPLD's parameter-tuning "on-the-fly" opens up a new possibility for improving classical navigation systems. However, APPLD makes decisions about which parameters to use based exclusively on the current context in a manner that disregards the potential future consequences of those decisions.

In contrast, APPLR goes beyond this myopic parameter tuning scheme and introduces the concept of a parameter *policy*, where we use the term policy to explicitly denote state-action mappings found through reinforcement learning to solve long-horizon sequential decision making problems. By using such a policy, APPLR is able to make parameter-selection decisions that take into account the possible future consequences of those decisions.

B. Learning-based Navigation

A plethora of recent works have applied machine learning techniques to the classical mobile robot navigation problem [6]–[11]. By directly leveraging experiential data, these learning approaches can not only enable point-to-point collision-free navigation without sophisticated engineering, but also enable capabilities such as terrain-aware navigation [12], [13] and social navigation [14], [15]. However, most end-to-end learning approaches are data-hungry, requiring hours of training time and millions of training data/steps, either from expert demonstration (as in imitation learning) or trial-and-error exploration (RL). Moreover, when an end-to-end policy is trained in simulation using RL, the sim-to-real gap may cause problems in the real-world. Most importantly, learning-based methods typically lack safety and explainability, both of which are important properties for mobile robots interacting with the real-world.

APPLR aims to address the aforementioned shortcomings: as a RL approach, learning in parameter space (instead of in velocity control space) effectively eliminates catastrophic failures (e.g. collisions) and largely reduces costly random exploration, with the help of the underlying navigation system. This change in action space can also help to generalize well to unseen environments and to mitigate the difference between simulation and the real-world (e.g. physics).

III. APPROACH

We now introduce the proposed approach, APPLR, which aims to apply RL to identify an optimal parameter selection policy for a classical motion planner. By doing so, APPLR naturally inherits from the classical planner its safety guarantees and ability to generalize to unseen environments. In addition, through RL, APPLR learns to autonomously and adaptively switch planner parameters "on-the-fly" and in a manner that considers future consequences without any expert tuning or human demonstration. In the rest of this section, we first introduce the background of classical motion planning. Then, we provide the problem definition of APPLR under the standard Markov Decision Process framework. Finally, we discuss the designed reward functions and the chosen reinforcement learning algorithm.

A. Background on Motion Planning

In this work, we assume the robot employs a classical motion planner, f, that can be tuned through a set of planner parameters $\theta \in \Theta$. While navigating in a physical world $\mathcal{W}$,[1] f tries to move the platform to a global navigation goal, e.g., $\beta = (\beta_x, \beta_y) \in \mathbb{R}^2$. At each time step t, f receives sensor observations o_t (e.g. lidar scans), and then computes a local goal $g = (g_x, g_y) \in \mathbb{R}^2$, which the robot attempts to reach quickly. Then, f is responsible for computing the motion commands $u_t = f(o_t, \beta \mid \theta)$ (e.g. u_t can be the angular/linear velocity). Most prior work in the learning community attempts to replace f entirely by a learnable function π_m that directly outputs the motion commands. However, the performance of these end-to-end planners is usually limited due to insufficient training data and poor generalization to unseen environments. In contrast, we focus here on a scheme for adjusting the planner parameters θ "on-the-fly". We expect learning in the planner parameter space to increase the overall system's adaptability while still benefiting from the verifiable safety assurance provided by the classical system.

B. Problem Definition

We formulate the navigation problem as a Markov Decision Process (MDP), i.e., a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, R)$. Assume an agent is located at the state $s_t \in \mathcal{S}$ at time step t. If the agent executes an action $a_t \in \mathcal{A}$, the environment will transition the agent to $s_{t+1} \sim \mathcal{T}(\cdot | s_t, a_t)$ and the agent will receive

[1]In classical RL approaches for navigation, $\mathcal{W}$ is usually defined as an MDP itself with the conventional state and action space (Fig. 1).

a reward $r_t = R(s_t, a_t)$. The overall objective of RL is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that can be used to select actions that maximize the expected cumulative reward over time, i.e. $J = \mathbb{E}_{(s_t,a_t)\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

In APPLR, we seek a policy in the context of an MDP $\mathcal{E}$ that denotes a *meta-environment* composed of both the underlying navigation world $\mathcal{W}$ (the physical, obstacle-occupied world) and a given motion planner f with adjustable parameters $\theta \in \Theta$ and sensory inputs $o \in \mathcal{O}$. Additionally, we assume going forward that a local goal g is always available (as a waypoint along a coarse global path in most classical navigation systems), and we use its angle relative to the orientation of the agent, i.e., $\phi = \arctan 2(g_y, g_x) \in [-\pi, \pi]$, to inject the local goal information to the agent. We assume the agent interacts with the environment at regular time intervals of fixed length. Within $\mathcal{E}$, at each time step t, $s_t = (o_t, \phi_t, \theta_{t-1})$, where $o_t \in \mathcal{O}$ is the current sensory inputs, $\phi_t \in \mathcal{G}$ is the angle towards the local goal, and $\theta_{t-1} \in \Theta$ is the previous planner parameter that f was using. That is, for our MDP $\mathcal{E}$, $\mathcal{S} = \mathcal{O} \times \mathcal{G} \times \Theta$ and $\mathcal{A} = \Theta$. In this context, APPLR aims to learn a policy $\pi_p : \mathcal{O} \times \mathcal{G} \times \Theta \to \Theta$ that selects a planner parameter θ_t that enables f to achieve optimal navigation performance over time. The agent then transitions to the next state $s_{t+1} = (o_{t+1}, \phi_{t+1}, \theta_t)$, where $o_{t+1}, \phi_{t+1} \sim \mathcal{T}(\cdot|s_t, \theta_t)$ are given by $\mathcal{E}$. The overall objective is therefore

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

After training with a reward function (Sec. III-C and III-D), the learned parameter policy π_p is deployed with the underlying navigation system f in the world $\mathcal{W}$, as summarized in Alg. 1.

Algorithm 1 Navigation with APPLR

```
Require: the physical world $\mathcal{W}$, the underlying motion
    planner $f$, the global goal $\beta$, the initial parameter $\theta_0$,
    and the parameter policy $\pi_p$.
1: $t = 1$
2: while $\beta$ is not reached do
3:     receive sensor readings $o_t$ from $\mathcal{W}$ and local goal $\phi_t$
       from a coarse global plan
4:     $\theta_t = \pi_p(o_t, \phi_t, \theta_{t-1})$                {parameter policy}
5:     $u_t = f(o_t, \beta \mid \theta_t)$              {$f$ parameterized by $\theta_t$}
6:     execute $u_t$ in $\mathcal{W}$
7:     $t = t + 1$
8: end while
```

C. Reward Function

We now describe our design of the reward function for APPLR. 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

$$R_t(s_t, a_t, s_{t+1}) = c_f R_f + c_p R_p + c_c R_c. \quad (2)$$

Here, c_f, c_p, c_c are coefficients for the three types of reward functions $R_f, R_p, R_c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. 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 at time t, the absolute coordinates of the robot are $p_t = (p_t^x, p_t^y)$, then we define

$$R_p = \frac{(p_{t+1} - p_t) \cdot (\beta - p_t)}{|\beta - p_t|}. \quad (3)$$

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 $R_c = -1/d(o_{t+1})$, where $d(o_{t+1})$ is a distance function measuring how close the robot is to obstacles based on sensor observations.

D. Reinforcement Learning Algorithm

We consider two major factors for choosing the RL algorithm for APPLR: (1) the algorithm should allow selection of continuous actions since the parameter space of most planners is continuous; (2) the algorithm should be highly sample efficient for physical simulation of navigation. Based on these two criteria, we use the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [16] for APPLR. As one of the state-of-the-art off-policy algorithms, TD3 is very sample efficient and handles continuous actions by design. Specifically, TD3 is an actor-critic algorithm that keeps an estimate for both the policy π_p^{ξ} and the state-action value function Q_p^{ζ}, parameterized by ξ and ζ separately. For the policy, it uses the usual deterministic policy gradient update [17]

$$\nabla_{\xi} J^{\pi_p^{\xi}} = \mathbb{E}_{s\sim\pi_p^{\xi}} \left[\nabla_a Q_p^{\zeta}(s, a)|_{a=\pi_p^{\xi}} \nabla \pi_p^{\xi}(s) \right]. \quad (4)$$

To address the maximization bias on the estimation of Q_p^{ζ}, which can influence the gradient in equation (4), TD3 borrows the idea from double Q learning [18] of keeping two separate Q estimators $Q_p^{\zeta_1}$ and $Q_p^{\zeta_2}$, each updated using the conventional Bellman residual objective $\mathbb{E}_{(s,a,r,s')\sim\mathcal{E}} \left[||Q_p^{\zeta}(s, a) - r - \gamma \max_{a'} Q_p^{\zeta}(s', a')||_2 \right]$. TD3 further stabilizes training by using the clipped value $\min(Q_p^{\zeta_1}(s, a), Q_p^{\zeta_2}(s, a))$ in the place of the target in the Bellman residual objective during the critic update to reduce overestimation of the true value. Then, $Q_p^{\zeta_1}$ is used in equation (4) to update the actor policy. Additionally, a delayed updating strategy that updates the policy less frequently than the value function is employed to further stabilize the training. As physical simulations suffer from high variance and are generally slow, TD3 is a good fit for APPLR.

To further address the sample inefficiency issue, a distributed general reinforcement learning architecture (Gorila) [19] is employed, which enables parallelized acting processes on a computing cluster. Our implementation of Gorila is a simpler version with only one serial learner and a large number of actors running individually in simulation environments to generate large quantities of data for a global replay buffer.

IV. EXPERIMENTS

In this section, we experimentally validate that APPLR can enable adaptive autonomous navigation *without* access to expert tuning or human demonstration and is generalizable to many deployment environments, both in simulation and in the real-world. To perform this validation, APPLR is implemented and applied on a Clearpath Jackal ground robot. 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 human demonstration using APPLD [3].

A. Implementation

We implement APPLR on a ClearPath Jackal differential-drive ground robot. The robot is equipped with a 720-dimensional planar laser scan with a 270° field of view, which is used as our sensory input o_t in Alg. 1. We preprocess the LiDAR scans by capping the maximum range to 2m. The onboard Robot Operating System (ROS) `move_base` navigation stack (our underlying classical motion planner f) uses Dijkstra's algorithm to plan a global path and runs DWA [20] as the local planner. Our u_t is the linear and angular velocity (v, ω). We query a local goal from the global path 1m away from the robot and compute the relative angle ϕ_t. APPLR learns a parameter policy to select DWA parameters θ, including *max_vel_x*, *max_vel_theta*, *vx_samples*, *vtheta_samples*, *occdist_scale*, *pdist_scale*, *gdist_scale*, and *inflation_radius*. We use the ROS `dynamic_reconfigure` client to dynamically change planner parameters. The global goal β is assigned manually, while θ_0 is the default set of parameters provided by the robot manufacturer.

π_p is trained in simulation using the Benchmark for Autonomous Robot Navigation (BARN) dataset [21] with 300 simulated navigation environments generated by cellular automata. Those environments cover different navigation difficulty levels, ranging from relatively open environments to highly-constrained spaces where the robot needs to squeeze through tight obstacles (three example environments with different difficulties are shown in Fig. 2). We randomly pick 250 environments for training and use the remaining 50 as the test set. 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 (Fig. 2). The distance function in $d(o_{t+1})$ in R_c is the minimal value among the 720 laser beams. π_p 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 [16] is implemented to learn the parameter policy π_p in simulation. The policy network and the two

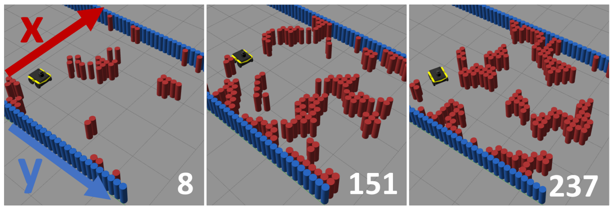

Fig. 2: Indexed Example Navigation Environments in Benchmark for Autonomous Robot Navigation (BARN) [21]

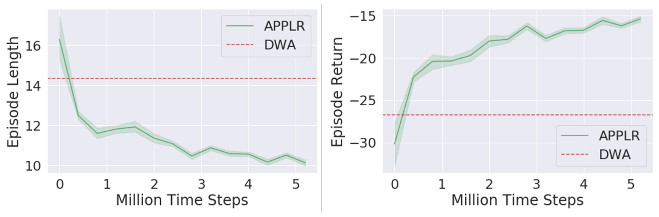

Fig. 3: Learning Curves of Episode Length and Episode Return Averaged over 100 Episodes: The values are moving averaged over 40k steps. The red dashed lines mark the average performance of DWA planner with a static set of default parameters.

target Q-networks are represented as multilayer perceptrons with three 512-unit hidden layers. The policy is learned under the distributed architecture Gorila [19]. 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. Fig. 3 shows episode length and return averaged over 100 episodes and compares with DWA motion planner with default parameters. The episode return continually increases and episode length drops by around 40% by the end of the training. As shown in Fig. 3, APPLR surpasses DWA at an early stage of the training process in terms of both episode length and return. After training, we deploy the learned parameter policy π_p in an example environment and plot four example parameter profiles produced by π_p in Fig. 4. Dashed lines separate different parts of the profile, which correspond to different regions in the example environment.

B. Simulated Experiments

After training, we deploy the learned parameter policy π_p on both the training and test environments. We use the metrics of 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

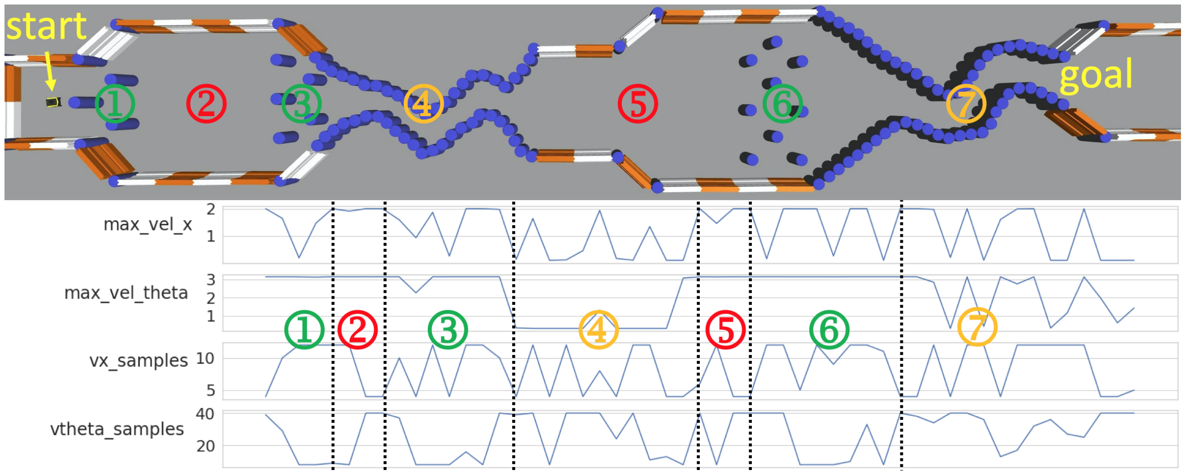

Fig. 4: Example Parameter Profiles Selected by APPLR: Labels of qualitatively similar regions are in the same color.

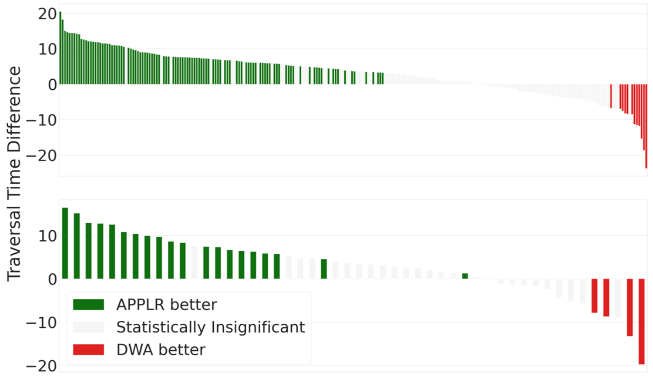

Fig. 5: Traversal time difference between APPLR and DWA for the training environments (top) and the test environments (bottom). The bars represent the environments ordered by traversal time difference. The colored bars indicate the environments that show statistically significant difference.

TABLE I: 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%)

each environment. We also perform the t-test for each pair of APPLR and DWA performance check the statistical significance. The results over the 250 training environments and 50 test environments are shown in Fig. 5. For the majority of the environments (green bars), APPLR shows statistically significantly better navigation performance. Tab. I 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. II compares the number of environments that show significant improvement and deterioration. In both training and test set, the results show that 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.

Furthermore, we analyze the relationship between performance improvement and difficulty level of a particular environment. We classify the first one third of environments (100) where DWA achieves the fastest traversal times as Easy, the one third of environments (100) with slowest traversal times as Difficult, and the remaining one third of environments (100) as Medium (Tab. III). While the advantage of APPLR over DWA is evident for the Easy environments (58% vs. 5%), it diminishes with increased environment difficulty (Medium: 37% vs. 7% and Difficult: 31% vs. 4%). We conjecture that this relationship may be due to a potential performance upper bound of the DWA planner due to its underlying structure. That is, while selecting the right planner parameters at each time step in easy environments can significantly improve its performance, in difficult environments, DWA's performance has saturated such that selecting the right parameters can only lead to marginal improvement. In those environments, it is likely that a completely different planner is required to achieve better performance, e.g. an end-to-end planner [9].

TABLE II: 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%)

TABLE III: Percentage of Environments (in which one method is better compared to the other) under Different Difficulty Levels

	APPLR better	DWA better
Easy Train	59%	5%
Easy Test	53%	5%
Easy All	**58%**	**5%**
Medium Train	36%	6%
Medium Test	43%	14%
Medium All	**37%**	**7%**
Difficult Train	33%	4%
Difficult Test	24%	6%
Difficult All	**31%**	**4%**

C. Physical Experiments

To validate the sim-to-real transfer of APPLR, we also test the learned parameter policy π_p on a physical Jackal robot.

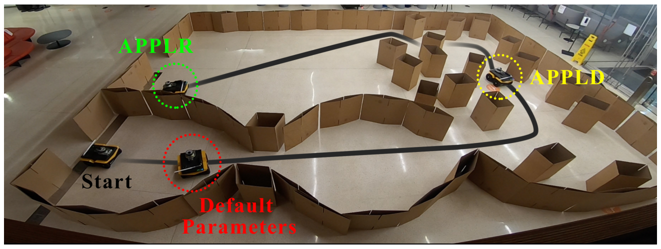

Fig. 6: Physical Experiments: While the DWA planner with a static set of default parameters (red) fails to find feasible motions and executes recovery behaviors in many places, APPLD (yellow) and APPLR (red) can both successfully and smoothly navigate through the entire obstacle course. Using RL, APPLR can achieve faster traversal than APPLD learned from (most likely suboptimal) human demonstration.

TABLE IV: 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**

The physical robot has a Velodyne LiDAR, but we transform the 3D point cloud to the same 2D laser scan as in the simulation (720-dimensional and 270° field of view). The learned policy is deployed in a real-world obstacle course set up by cardboard boxes (Fig. 6). This physical environment is different from any of the navigation environments in the BARN dataset. Therefore, both generalizability and sim-to-real transfer of APPLR can be tested with this unseen real-world environment.

Given this target environment, we further collect a tele-operated demonstration provided by one of the authors and learn a parameter tuning policy based on the notion of navigational context (APPLD [3]). 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 Tab. IV. We observe one failure trial (the robot fails to find feasible motions and keeps rotating in place at the beginning of the narrow part) with DWA. Therefore, the DWA results only contain the four successful trials.

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 [21], where obstacles are generated by cellular automata and therefore very cluttered, the relatively open space in the physical environment (Fig. 6) 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. On the other hand, the RL parameter policy aims at reducing the traversal time and finds better parameter sets to achieve that goal. Another reason is that while APPLD only utilizes three sets of learned parameters for the three navigational contexts, APPLR is given the flexibility to change parameters at each time step, and RL is able to utilize the sequential aspect of the parameter selection problem, e.g. slowing down in order to speed up in the future. However, we observe that in confined spaces APPLR produces less smooth motion compared to APPLD. One possible explanation is that the teleoperated human demonstration in APPLD aims at both fast and smooth navigation, while APPLR only aims at speed. This issue may be addressable in future work through a different reward function.

V. CONCLUSIONS

In this paper, we introduce APPLR, *Adaptive Planner Parameter planning from Reinforcement*, which, in contrast to *parameter tuning*, learns a *parameter policy*. The parameter policy is trained using RL to select planner parameters at each time step to allow the robot to take suboptimal actions in a current state in order to achieve better future performance. Furthermore, instead of learning an end-to-end navigation planner, we treat a classical motion planner as part of the environment and the RL agent only interacts with it through the planner parameters. Learning in this parameter space instead of a velocity control space not only allows APPLR to inherit all the benefits of classical navigation systems, such as safety and explainability, but also eliminates wasteful random exploration with the help of the underlying planner, allows it to generalize well to unseen environments, and reduces the chances of failing to overcome the sim-to-real transfer gap. The unsupervised APPLR paradigm does not require any expert tuning or human demonstration. APPLR is trained on a suite of simulated navigation environments and is then tested in unseen environments. We also conduct physical experiments to test APPLR's sim-to-real transfer. As mentioned in Sec. IV, one interesting direction for future work is to design reward functions that encourage motion smoothness. Currently, APPLR is only useful if there's no switching cost in the planner for changing parameter sets, but, if such a cost exists, future work should take it into account. Another interesting direction is to use curriculum learning to start from easy environments and then transition to difficult ones. Furthermore, the APPLR pipeline has the potential to be applied to other navigation systems, including visual, semantic, or aerial navigation.

REFERENCES

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

[2] 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.

[3] 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.

[4] D. Teso-Fz-Betoño, E. Zulueta, U. Fernandez-Gamiz, A. Saenz-Aguirre, and R. Martinez, "Predictive dynamic window approach development with artificial neural fuzzy inference improvement," *Electronics*, vol. 8, no. 9, p. 935, 2019.

[5] M. Bhardwaj, B. Boots, and M. Mukadam, "Differentiable gaussian process motion planning," *arXiv preprint arXiv:1907.09591*, 2019.

[6] 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 *IEEE International Conference on Robotics and Automation*. IEEE, 2017.

[7] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," *arXiv preprint arXiv:1710.05627*, 2017.

[8] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[9] 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.

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

[11] X. Xiao, B. Liu, and P. Stone, "Agile robot navigation through hallucinated learning and sober deployment," *arXiv preprint arXiv:2010.08098*, 2020.

[12] 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.

[13] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *arXiv preprint arXiv:2002.05700*, 2020.

[14] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[15] A. Pokle, R. Martín-Martín, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang, A. Sadeghian, D. Sadigh, S. Savarese *et al.*, "Deep local trajectory replanning and control for robot navigation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5815–5822.

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

[17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[18] H. V. Hasselt, "Double q-learning," in *Advances in neural information processing systems*, 2010, pp. 2613–2621.

[19] 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.

[20] 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.

[21] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," *arXiv preprint arXiv:2008.13315*, 2020.