A Lifelong Learning Approach to Mobile Robot Navigation

Bo Liu, Xuesu Xiao, and Peter Stone

***Abstract*—This letter presents a self-improving lifelong learning framework for a mobile robot navigating in different environments. Classical static navigation methods require environment-specific in-situ system adjustment, e.g., from human experts, or may repeat their mistakes regardless of how many times they have navigated in the same environment. Having the potential to improve with experience, learning-based navigation is highly dependent on access to training resources, e.g., sufficient memory and fast computation, and is prone to forgetting previously learned capability, especially when facing different environments. In this work, we propose Lifelong Learning for Navigation (LLfN) which (1) improves a mobile robot's navigation behavior purely based on its own experience, and (2) retains the robot's capability to navigate in previous environments after learning in new ones. LLfN is implemented and tested entirely onboard a physical robot with a limited memory and computation budget.**

***Index Terms*—Motion and path planning, autonomous vehicle navigation, sensorimotor learning, machine learning for robot control, imitation learning.**

I. Introduction

CLASSICAL mobile robots are designed to be adaptive to different navigation environments by in-situ adjustment of the underlying navigation system, such as by sensor calibration [1] or by parameter tuning [2]. However, without adjustment from expert knowledge, the untuned system may repeat the same mistakes (e.g. stuck in the same bottleneck) even though it has navigated in the same environment multiple times.

Recent success in using machine learning for mobile robot navigation indicates the potential of improving navigation performance from a robot's past experience in the same environment [3]. When facing different navigation environments, however, learning methods cannot generalize well to unseen scenarios: They must re-learn to navigate in the new environments. More importantly, the learned system is prone to *catastrophic forgetting*, which causes the robot to forget what was learned in previous environments [4].

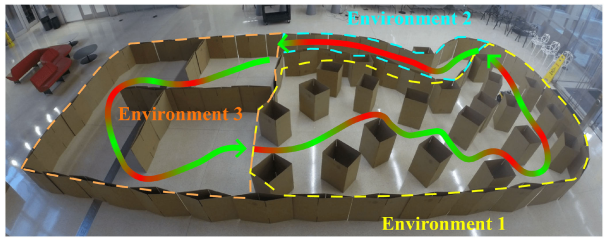


Fig. 1. Three navigation environments: An initial navigation policy navigates well most of the time (green), but occasionally behaves suboptimally (red, e.g. moving extremely slowly or getting stuck). Lifelong Learning for Navigation learns a complementary policy deployed in conjunction with the initial policy, which gradually eliminates the suboptimal behaviors in the current environment while not diminishing performance in previous environments. During deployment, the learned policy is mostly used in the red segments.

This letter introduces a Lifelong Learning for Navigation (LLfN) framework that addresses the aforementioned challenges: Instead of learning from scratch, the navigation policy is initialized through a classical navigation algorithm, whose navigation performance does not improve with increasing experience. The robot is able to identify its suboptimal actions and learn from them. The navigation performance then improves in a self-supervised manner. When facing different navigation environments, the navigation policy is able to learn to adapt to new environments, while not forgetting how to navigate in previous ones. LLfN is implemented entirely onboard a physical robot with limited memory and computation, and demonstrated to allow the robot to navigate in three different environments (Fig. 1). The main contributions of the letter are:

- A self-improvement strategy that complements an initial static planner to dynamically increase navigation performance with more experience, deployed in conjunction with the initial planner to minimize learning overhead;
- A lifelong learning scheme that allows a robot to navigate in new environments while not forgetting previous ones; and
- An implementation of the Lifelong Learning for Navigation framework entirely onboard a physical robot platform with limited memory and computation.

II. Related Work

This section first reviews how classical and learning-based methods improve navigation performance and adapt to

Manuscript received October 15, 2020; accepted January 24, 2021. Date of publication February 2, 2021; date of current version February 16, 2021. This letter was recommended for publication by Associate Editor P. Tokekar and Editor D. Popa upon evaluation of the reviewers' comments. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1 739 964, IIS-1 724 157, NRI-1 925 082), 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. *(Bo Liu and Xuesu Xiao contributed equally to this work.) (Corresponding author: Bo Liu.)*

Bo Liu and Xuesu Xiao are with the Department of Computer Science, The University of Texas at Austin, Austin 78712, TX, United States of America (e-mail: xiao@cs.utexas.edu; bliu@cs.utexas.edu).

Peter Stone is with the Department of Computer Science, The University of Texas at Austin, Austin 78712, TX, United States of America, and also with the Sony AI (e-mail: pstone@cs.utexas.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3056373, provided by the authors.

Digital Object Identifier 10.1109/LRA.2021.3056373

different navigation environments, and then briefly discusses recent successes in the continual learning community.

a) Classical navigation: Classical navigation systems [5]–[8] are designed to be applicable to a wide variety of environments, but they are usually static, operating under a fixed set of pre-specified hyper-parameters and therefore lacking the ability to improve with experience and adapt to a specific environment. Parameter Tuning is the current practice to address the aforementioned problems [9], which requires human experts' intuition, experience, and trial-and-error. To reduce the reliance on expert knowledge, Xiao *et al.* [2] proposed to improve navigation for a given environment (defined as "context") by tuning parameters through Behavior Cloning (BC) from teleoperated demonstration. In most cases, tuning requires human knowledge, either in the form of direct tuning or of navigation demonstrations. Furthermore, once tuned, the static navigation system lacks the ability to further improve with more experience or adapt to new environments. In contrast, the proposed Lifelong Learning for Navigation does not require human knowledge and can dynamically improve with more navigation experience when facing new environments.

b) Learning-based navigation: Data-driven machine learning techniques have also been widely applied to navigation problems [10]–[18]. As for physical robot navigation, learning approaches typically either imitate an expert [19], [20] or learn from trial-and-error using reinforcement learning [3], [21]. While these learning methods enable improvement in a specific environment with increased navigation experience, if the agent were to be placed in multiple environments in a sequential fashion, which is common in real-world navigation, learning methods may not generalize well and can easily forget the past knowledge. By contrast, LLfN explicitly considers how to prevent forgetting and increase generalization. From a lifelong learning perspective, Wyeth and Milford [22] studied continual mapping for navigation and Wang *et al.* [23] improved generalization in vision-language navigation. Both methods focus on the problem of *where* to navigate instead of *how* to navigate. To the best of the authors' knowledge, no existing work has tackled lifelong/continual learning of navigation behaviors across different navigation environments. We conjecture that unconstrained offboard computation resources, e.g., memory, power, and time, allow learning from an extensive body of training data pre-collected in different environments. However, for onboard resource-constrained robot platforms without pre-collected supervised data, learning how to navigate in new environments while not forgetting previous ones requires further research, and is the focus of this work.

c) Lifelong/continual supervised learning: Lifelong or continual learning studies the problem of learning in an ongoing fashion. One of the earliest attempts at lifelong learning originates from the robotics community [24]. Ring [25] provides the earliest introduction to continual learning in reinforcement learning problems. Recently, much progress has been made for continual learning with neural networks. There are mainly three categories of approaches: 1) use regularization to prevent the learned weights from deviating too much from the old weights [26], [27]; 2) train a generative model to recover old data for joint optimization [28]; and 3) adopt a dynamic network architecture for learning more tasks [29], [30]. Among the above approaches, the first approach applies to a fixed capacity network, which is often much more computationally efficient than training a generative model or adopting a dynamic network architecture. This computational efficiency is essential for learning onboard resource-constrained mobile robot platforms. Specifically, when a few past data points can be saved, Gradient Episodic Memory (GEM) [31] can be very efficient and powerful. All these methods demonstrate success on continual image classification problems, but there remains much room for studying continual learning in other applications like in robotics, especially when supervised labels from human experts are not available a priori. It is worth to note lifelong learning's resemblance to transfer learning [32]. However, in addition to transfer learning's focus on forward transfer (how previous knowledge can help learning the current task), LLfN also considers backward transfer (how learning the current task can maintain or improve performance of old ones).

III. Background

Lifelong Learning for Navigation (LLfN) aims to address a novel variant of the standard navigation problem in which the agent learns to improve navigation performance online with increasing experience, or across environments, under a limited memory budget. We think of this problem variant as the "lifelong navigation" problem.[1] In this section, we first present the problem setup of lifelong navigation and the notation we use in this work. We then leverage a lifelong learning algorithm previously used in continual image classification for the LLfN framework.

A. Problem Setup and Notation

The high-level objective of lifelong navigation can be summarized as learning to navigate in a sequence of m environments $\{\mathcal{E}_i\}_{i=1}^{m}$.[2] In each of those environments, the robot aims at navigating from one fixed start point to another fixed goal point. We assume a fixed global planner (e.g. Dijkstra's algorithm [33], A* [34] or D* [35]) generates a path connecting start and goal, and while navigating, the robot needs to produce motion commands which follow this global path, observe its kinodynamic constraints, and avoid obstacles. Whenever the agent advances to $\mathcal{E}_k$, it no longer has access to $\{\mathcal{E}\}_{i=1}^{k-1}$. Within the environment $\mathcal{E}_k$, the agent, at each time step t, computes a motion command $a_t \in \mathcal{A} \sim \pi_\theta(s_t)$, where $s_t \in \mathcal{S}$ is the agent's state, π_θ is a policy parameterized by θ. After executing a_t, the agent advances to s_{t+1} and the process continues. During the learning phase, the agent will have a limited onboard memory size n, i.e. maximally n pairs of (s_t, a_t) can be stored at any moment. Once the agent has seen all m environments, its navigation performance will be evaluated on the same m environments.

B. Gradient Episodic Memory

The key challenge of continual learning is catastrophic forgetting, i.e. an agent forgets what it learned previously when adapting to a new environment. The phenomenon is especially prominent when feature-rich parametric models, i.e. neural networks, are used as the underlying learning module. To

[1] We define "lifelong navigation" in the sense of learning navigation with increasing experience, across environments, rather than in the literal sense, i.e. navigation over extended periods of time.

[2] Informally, an environment is a contiguous space consists of similar distribution of obstacles where the optimal navigation behavior does not vary much. We intentionally design the simulated and physical environments to be very different to magnify the effect with and without LLfN.

address catastrophic forgetting, we use Gradient Episodic Memory (GEM) [31] within our Lifelong Learning for Navigation paradigm described in Section III-A. From a high-level perspective, GEM prevents forgetting by ensuring each update will not increase the loss on previous tasks. Note that GEM allows new experience to improve performance on old tasks. Specifically, assume the agent has already seen environments up to $\mathcal{E}_{k-1}$ and the learned policy is $\pi_{\theta_{k-1}}$. GEM assumes the agent keeps a small memory buffer $\mathcal{B} = \{\mathcal{M}_i\}_{i<k}$ that, for each previous environment $\mathcal{E}_i$, stores a few exemplary data points $\mathcal{M}_i$. GEM then optimizes the following objective:

$$\min_{\theta} \ell(\pi_\theta, \mathcal{E}_k), \text{ s.t. } \ell(\pi_\theta, \mathcal{M}) \leq \ell(\pi_{\theta_{k-1}}, \mathcal{M}), \ \forall \mathcal{M} \in \mathcal{B}, \quad (1)$$

where $\ell(\pi, X)$ is the loss function that evaluates performance of π on data X.[3] For instance, in a regression task where we aim to predict the regression label a from a given state s, then $\ell(\pi_\theta, X) = \mathbb{E}_{(s,a)\sim X}||\pi_\theta(s) - a||_2$. To efficiently solve the above optimization, GEM observes that the constraints are satisfied as long as 1) the new θ is initialized from θ_{k-1}, and 2) at each optimization step, the loss on previous tasks does not increase. Assume the optimization steps are small, we can determine whether a new update increases the loss on a previous task by computing the inner product between the gradients on the current and previous tasks. For example, the loss on a previous task will only increase if the inner product is negative. The optimization problem then becomes

$$\min_{\theta} \ell(\pi_\theta, \mathcal{E}_k), \text{ s.t. } \left\langle \frac{\ell(\pi_\theta, \mathcal{E}_k)}{\partial \theta}, \frac{\ell(\pi_\theta, \mathcal{M})}{\partial \theta} \right\rangle \geq 0, \ \forall \mathcal{M} \in \mathcal{B}. \quad (2)$$

In practice, to solve Equation (2), GEM uses stochastic gradient descent with a modified gradient. In particular, denote $g = \frac{\ell(\pi_\theta, \mathcal{E}_k)}{\partial \theta}$ and $g_i = \frac{\ell(\pi_\theta, \mathcal{M}_i)}{\partial \theta}$, $\forall i < k$. Then, GEM finds the update direction $\tilde{g}$ by solving:

$$\tilde{g} = \underset{z}{\operatorname{argmin}} ||g - z||_2, \text{ s.t. } \langle z, g_i \rangle \geq 0, \quad \forall i < k. \quad (3)$$

The above optimization is already in a nice quadratic form, but the decision variable $\tilde{g}$ has the same dimension as θ, which can be millions for deep architectures. Fortunately, its dual problem is only associated with $k-1$ variables and can be efficiently solved by standard quadratic programming solvers. Formally, the dual problem of (3) is

$$v^* = \arg\max_{v} \ v^T G^T G v + g^T G v \ \text{ s.t. } \ v \geq 0 \in \mathbb{R}^{k-1},$$

where $G = -[g_1, g_2, \ldots, g_{k-1}]$ is a matrix having $-g_i$ as its columns. As a result, the final $\tilde{g} = G^T v^* + g$ and the update rule is $\theta \leftarrow \theta - \alpha \tilde{g}$, where α is the learning rate. GEM has no requirement for dynamically expanding the parameter size and often requires very few exemplar data points from past experience to maintain the learned behavior. Therefore, GEM is particularly suitable for robot navigation tasks since mobile robots often have very limited onboard memory resources.

IV. Lifelong Learning for Navigation

Intuitively, catastrophic forgetting is caused by the dilemma of overwriting old knowledge when learning new things. In real-world navigation, it is unlikely that the agent needs to keep learning new things every second. The agent can easily navigate in many places with classical approaches and only have trouble navigating in particular scenarios (e.g. the green and red path segments in Fig. 1). As a result, we regard Lifelong Learning for Navigation (LLfN) as a framework that learns an auxiliary planner π_θ to assist a classical planner π_0 only for navigating those difficult instances. Moreover, updating π_θ should minimally influence the past learned behaviors. To achieve this goal, the agent should be able to first identify those "difficult" scenarios, i.e. identify in what particular states s does the agent generate suboptimal motion commands.

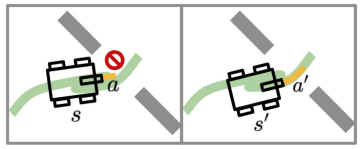


Fig. 2. Learning from $(s,' a')$ helps improve the navigation performance around s.

In addition, since we do not assume access to expert demonstrations, the agent must keep sampling motion commands until it overcomes the difficulty around s. Then, by looking at the trajectories around s, the agent should be able to identify a good behavior, i.e. a state-action pair $(s,' a')$, such that learning from $(s,' a')$ will increase the agent's probability of overcoming the difficulty around s (see Fig. 2). Next, we list the key components of the LLfN framework and then explain how we use them.

- An initial sampling-based navigation planner π_0 and a learnable policy π_θ, parameterized by θ.
- A scoring function $D : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluates how good an action a is at the state s, i.e. larger $D(s, a)$ indicates a is a better action at s.
- A streaming memory buffer $\mathcal{B}_{\text{stream}}$ that stores the past T-step trajectory, i.e. $\mathcal{B}_{\text{stream}} = \{s_j, a_j\}_{j=t-T+1}^{t}$.
- A per-environment memory $\mathcal{M}_k : |\mathcal{M}_k| = n/k$ (n is the memory budget) that stores the exemplar training data (self-generated data that are worth learning from) from environment $\mathcal{E}_k$. The entire memory before entering $\mathcal{E}_k$ is therefore a set of sets: $\mathcal{B} = \{M_i\}_{i<k}$.
- An algorithm A_{correct} that given a recent suboptimal behavior $(s, a) \in \mathcal{B}_{\text{stream}}$, finds exemplar training data $(s,' a') \in \mathcal{B}_{\text{stream}}$ such that learning from $(s,' a')$ improves the navigation performance at s.
- A continual learning algorithm A_{cl} that updates π_θ given $\mathcal{M}_k$ and $\mathcal{B}$. A_{cl} should retain performance on previous environments.

With the above components, the agent will be manually placed at fixed start locations of a sequence of m different environments, in each of which it navigates to a fixed goal, keeps identifying suboptimal behaviors, and improves upon them while preserving its past knowledge.[4] The pipeline of Lifelong Learning for Navigation is summarized in Algorithm 1.

In Algorithm 1, the history memory buffer $\mathcal{B}$ and the streaming memory buffer $\mathcal{B}_{\text{stream}}$ are initialized to empty, and π_θ to a random policy in line 2. Upon entering $\mathcal{E}_k$ (line 4), the per-environment memory $\mathcal{M}_{\text{k}}$ is initialized to empty (line 5). While

[3]Here we abuse the notation so that $\ell(\pi, \mathcal{E})$ refers to the loss evaluated on data generated from environment $\mathcal{E}$.

[4]It may be possible for the agent to automatically detect environment shift, but we leave that for future work.

Algorithm 1: Lifelong Learning for Navigation (LLfN).

1: **Inputs**: π_0, π_θ, D, A_{correct}, A_{cl}, $\mathcal{E}_{k=1}^{m}$, and a threshold η.
2: $\mathcal{B} \leftarrow \emptyset$, $\mathcal{B}_{\text{stream}} \leftarrow \emptyset$, and initialize θ_0 randomly
3: // Training
4: **for** environment $k = 1 : m$ **do**
5: $\mathcal{M}_{\text{k}} \leftarrow \emptyset$
6: **while** navigating in $\mathcal{E}_k$ **do**
7: progress to state s_t and generate $a_t \sim \pi_0(s_t)$
8: execute a_t and update $\mathcal{B}_{\text{stream}}$ with (s_t, a_t)
9: let $p = \lfloor t - T/2 \rfloor$ and select $(s_p, a_p) \in \mathcal{B}_{\text{stream}}$
10: **if** $D(s_p, a_p) < \eta$ **then**
11: $(s,' a') = A_{\text{correct}}(s_p, \mathcal{B}_{\text{stream}})$
12: update $\mathcal{M}_{\text{k}}$ with $(s,' a')$
13: **end if**
14: **end while**
15: $\theta_k \leftarrow A_{\text{cl}}(\pi_{\theta_{k-1}}, \mathcal{M}_{\text{k}}, \mathcal{B})$ ▷ Lifelong learning
16: Shrink $\mathcal{B}$ to size $(n - |\mathcal{M}_{\text{k}}|)$ and $\mathcal{B} = \mathcal{B} \cup \{\mathcal{M}_k\}$
17: **end for**
18: // Execution
19: **while** navigating in $\mathcal{E}$ **do**
20: progress to state s_t
21: generate $a_0 \sim \pi_0(s_t)$, $\hat{a} \sim \pi_{\theta_k}(s_t)$
22: execute $a_t = \text{argmax}_{a \in \{a_0, \hat{a}\}} D(s_t, a)$
23: **end while**

learning to navigate in $\mathcal{E}_k$ (line 6), when the agent progresses to a state s_t (line 7), it executes the motion command $a_t \sim \pi_0(s_t)$ and saves the state-action pair (s_t, a_t) to the streaming buffer $\mathcal{B}_{\text{stream}}$ (lines 7-8). Then, the agent looks at the mid-point[5] of the recent trajectory $(s_p, a_p) \in \mathcal{B}_{\text{stream}}$ (line 9) and evaluates its score $D(s_p, a_p)$ (line 10). If the score is below a pre-defined threshold η, then we regard (s_p, a_p) as suboptimal and use A_{correct} to identify a nearby state-action pair $(s,' a') \in \mathcal{B}_{\text{stream}}$, learning from which could potentially help improve the navigation performance around s (line 11). In practice, we implement D with the extracted heuristics from DWA which discriminates recovery behaviors from regular ones (see Section V-A for details). In other words, suboptimal actions are the ones that trigger DWA's recovery behavior.

Intuitively, if we search state-action pairs around (s_p, a_p) that have scores above the threshold η, and find the state-action pair $(s,' a')$ that has the most similar state to s_p, then learning from $(s,' a')$ could potentially help the agent navigate from s_p. Therefore, we propose that A_{correct} selects $(s,' a')$ according to

$$(s,' a') = \underset{(s,a) \sim \mathcal{B}_{\text{stream}}}{\text{argmax}} \ \text{sim}(s, s_p) \ \text{ s.t. } \ D(s, a) \geq \eta, \qquad (4)$$

where $\text{sim}(s, s_p)$ measures how similar s is to s_p. What we really want is to learn from (s, a^*), where a^* is the optimal motion command at state s and is unknown by the policy. Here, $(s,' a')$ serves as the nearest neighbor to (s, a^*). The underlying intuition is that taking the same action in similar states should result in similar scores. Inversely, the similarity score indicates how confident we are that learning from the nearest neighbor $(s,' a')$ can actually help the agent successfully navigate s_p, provided that $D(s,' a') \geq \eta$. In our implementation, the state consists of raw sensor readings, such as LiDAR, so the negative Euclidean distance $-||s' - s_p||_2$ is a reasonable measure of similarity. Importantly, although it is possible that within $\mathcal{B}_{\text{stream}}$, no state-action pair has sufficiently similar state to s_p, learning from the nearest neighbor $(s,' a')$ is not detrimental to π_θ since $D(s,' a') \geq \eta$. In line 12, $(s,' a')$ is stored to $\mathcal{M}_k$. Note that $\mathcal{M}_k$ is also size-constrained, i.e. $|\mathcal{M}_k| \leq n/k$. So if adding $(s,' a')$ exceeds the memory budget, we remove the $(s,' a') \in \mathcal{M}_k$ with the *lowest similarity score* to their own corresponding s. By doing so, $\mathcal{M}_k$ eventually consists of state-action pairs that most likely contribute to improvement of navigation performance. In line 15, the agent updates its knowledge with the current $\mathcal{M}_{\text{k}}$, using the continual learning algorithm A_{cl}, while preserving what it has learned before by considering $\mathcal{B}$. In particular, to continually update π_θ, we adopt the Gradient Episodic Memory (GEM) as A_{cl}. Specifically, we would like to solve

$$\theta_k = \text{argmin}_\theta \ell(\pi_\theta, \mathcal{M}_{\text{k}}) \ \text{ subject to}$$
$$\left\langle \frac{\ell(\pi_\theta, \mathcal{M}_{\text{k}})}{\partial \theta}, \frac{\ell(\pi_\theta, \mathcal{M})}{\partial \theta} \right\rangle > 0, \ \ \forall \mathcal{M} \in \mathcal{B}. \qquad (5)$$

Here $\ell(\pi_\theta, \mathcal{M}) = \mathbb{E}_{(s,a) \in \mathcal{M}} ||a - \pi_\theta(s)||_2$, is the standard behavior cloning objective. The optimization in Equation (5) will then be solved with stochastic gradient descent using the same method shown in Equation (3). Finally, to accommodate the online memory budget, we shrink $\mathcal{B}$ by removing entries with the lowest similarity score to their own corresponding s and append data from $\mathcal{M}_k$ to $\mathcal{B}$ (line 16).

For executing the learned policy from LLfN, the only change is that given the state s_t, we compute both $a_0 \sim \pi_0(s_t)$ and $\hat{a} \sim \pi_\theta(s_t)$ and execute the action with the highest score (lines 19–23). For description simplicity, we learn once per environment in Algorithm 1. However, in practice, the method can be easily extended to learn multiple times while navigating in the same environment.

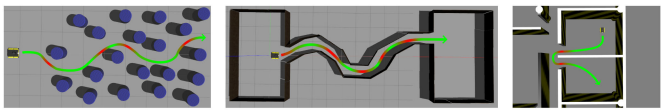
V. Experiments

LLfN is tested in simulated and physical experiments. We hypothesize that through LLfN (1) navigation performance can improve as the robot gathers more experience within a single environment, and (2) navigation in new environments can be learned while not forgetting how to navigate well in previous ones.

A. Robot Platform and Implementation

Clearpath Jackal, a four-wheeled differential-drive unmanned ground vehicle, is used for both simulated and physical experiments. The robot is equipped with a laser scanner to perceive surrounding obstacles and runs the basic Robot Operating System (ROS) `move_base` navigation stack. While the global planner is based on Dijkstra's algorithm [33], the local planner uses DWA [6], a sampling-based planner which may fail to sample feasible actions if not being properly tuned for the deployment environment. We fix the global planner and improve the local DWA planner using LLfN. We include the local goal provided by the global planner as part of the state, along with LiDAR input. Based on the local DWA planner as an initial policy (π_0), LLfN learns to complement this π_0 and improve navigation performance in multiple environments. When the sampling-based DWA cannot find feasible motion, it

[5]By looking at the midpoint (s_p, a_p), we search the neighboring behaviors (within $T/2$ steps) for $(s,' a')$.



(a) Env. 1 (b) Env. 2 (c) Env. 3

Fig. 3. Simulated Navigation Environments: Green segments are primarily traversed using the initial policy π_0, while red segments are mostly traversed using the learned planner π_θ.

starts recovery behavior, including rotation in place or backing up [9]. LLfN first improves navigation by eliminating those sub-optimal behaviors and producing alternative motions for a given environment. Second, it allows the robot to adapt to new environments while still remembering previous ones.

The learning problem is formulated as finding a policy π_θ that maps from the current state s, which includes LiDAR input (720- and 2095-dimensional for simulated and physical experiments, respectively) and local goal ((x, y), 1 m away on the global path), to the action a, i.e. the linear velocity v and angular velocity ω. In our implementation, the scoring function D prioritizes actions proposed by DWA, if they result in a steady forward motion ($v \geq 0.15$ m/s, which is slightly larger than the minimal $v = 0.1$ m/s to overcome friction). Importantly, all recovery behaviors from the DWA planner, including slowing down, turning in place, and moving backward, result in $v < 0.15$ m/s. So with the above definition, D will classify all recovery behaviors as suboptimal. Future works can investigate other implementations of D, or even learning it on the fly. If DWA fails to find feasible actions and would have started to execute recovery behaviors, the learned policy π_θ takes over. We acknowledge this implementation makes the scoring function D not always accurate, e.g. it is possible that D may prefer a bad action by π_θ when π_0 produces recovery behaviors. But in practice we observe LLfN remains effective even with such a simple D.

In consideration of limited onboard resources, we implement a streaming buffer $\mathcal{B}_{\text{stream}}$ as a regular queue to store 300 online streaming data points sequentially, and a separate memory $\mathcal{M}_{\text{k}}$ as a priority queue to save exemplar training data in the current environment. The number of data points within $\mathcal{B} \cup \{\mathcal{M}_k\}$ is also constrained to 300. The buffer sizes are empirically determined as the minimal value to assure efficient learning. For every sub-optimal action generated by DWA at a certain state, we compute a similarity score between this state and all states in the streaming buffer with a successful action (L2-norm of the LiDAR reading difference). We replace the data point of lowest similarity score in the training buffer with that from the streaming buffer. The onboard data memory overhead to implement LLfN is *at most* a total of 600 data points.

B. Simulated Experiments

The effect of LLfN is first studied with extensive simulated trials. The simulated Jackal has a SICK LMS111 laser scanner onboard providing 270° 720-dimensional laser scan. The three simulated navigation environments are shown in Fig. 3, where the robot navigates from a fixed start to a fixed goal in the environment. Env. 1's dense obstacles require fast response for obstacle avoidance; In Env. 2, the robot needs to keep a slow and consistent pace to drive through the narrow passage; Env. 3 requires the robot to slow down to make a sharp turn to enter the other room smoothly.

The robot is placed at its start location in each environment, and the learned policy π_θ to be tested is selected manually.

We use the initial policy π_0 to execute three trials in each of the three navigation environments to collect the training data. π_0 (DWA) can eventually navigate through, given sufficient time to recover, re-sample, and re-plan. To test in-environment learning improvement, training data of each given environment is *divided into* 5 segments and incrementally presented to the learner. For example, the first training buffer for learning is constructed from the first one-fifth of the robot's experience, the second from the first two-fifths, etc. The last training buffer is from the entire experience. Note all these training buffers contain only 300 data points, who have the highest similarities to those states where the initial policy performs sub-optimally, given the presented navigation experience. Therefore, 6 incrementally learned navigation policies are produced, including the initial policy and the five policies learned from seeing 1/5, 2/5, . . . , 5/5 of the training data from three training trials. Then the robot starts to learn the next navigation environment. For cross-environment learning, we use Sequential Training and the proposed LLfN. Sequential Training first uses the training buffer of environment 1. Starting with the final policy trained from environment 1, it sequentially trains on the training buffer of environment 2, before moving on to the training buffer of environment 3. In LLfN, while training environment 2, it only uses 150 data points from training buffer 2 with the highest similarity score, and still keeps a memory of 150 data points with the highest similarity score from training buffer 1 to assure new gradient updates won't increase the loss of environment 1. While training on environment 3100 data points with the highest similarity score for each environment are used, to avoid forgetting environment 1 and 2.

We implement a small neural network of three hidden layers with 64 hidden neurons each, to compute linear and angular velocity based on the LiDAR input and local goal. After training, we evaluate the navigation performance in terms of traversal time in the current and previous environments. Each policy is executed three times, resulting in a total of 198 evaluation trials. We report the mean and standard deviation of the performance in Fig. 4. If the robot is not able to reach the goal, e.g. gets stuck, a penalty time of 100 s is given. Within each environment, LLfH is able to decrease traversal time with increasing learning experience. Across environments, LLfH learns new environments with increasing data while avoids catastrophic forgetting of previous ones. Sequential Training can improve navigation performance of a given environment when being presented the data from that particular environment, faster than LLfN. However, navigation performance in previous environments deteriorates with increasing experience in the current environment. The catastrophic forgetting is apparent in the diverging red line (Sequential Training) from the green line (LLfH). Note that the learned policies are used in conjunction with the initial policy in both cases.

C. Physical Experiments

LLfN is also implemented on a physical Jackal robot. In the physical experiments (Fig. 1), we use the same setup, while all computation is done onboard the robot using an Intel Core i5-4570TE CPU. The physical Jackal has a Velodyne LiDAR, whose 3D point cloud data is converted to 360° 2095-dimensional laser scan. We design the physical test environments

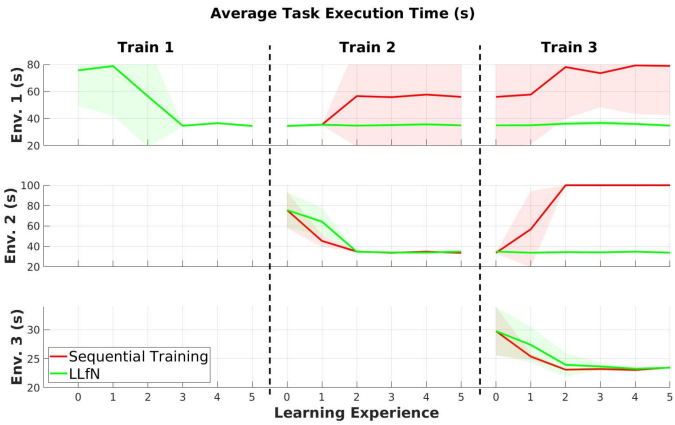


Fig. 4. Simulation Results: Each subplot (row i, column j) shows navigation performance with respect to increasing experience (x-axis), being trained using the jth and deployed in the ith environment. (Since there is no previous environment for Env. 1 and Train 1, both curves are the same).

TABLE I
PHYSICAL RESULTS AFTER TRAINING ON 3 ENVIRONMENTS: TIME (IN S), NUMBER OF RECOVERY BEHAVIORS (REC.) AND COLLISIONS (COL.)

		DWA	Sequential Training	LLfN	Individual Models
Env. 1	Time	38.36±1.69	39.11±6.45	**28.32**±1.85	30.17±**0.97**
	Rec./Col.	1.8/0	1.4/0.6	0/0	0/0
Env. 2	Time	28.05±3.48	49.41± 13.94	**19.98**±1.07	23.41±**0.66**
	Rec./Col.	1.4/0.2	1.25/0.8	0/0	0/0
Env. 3	Time	39.80±6.39	21.12±1.17	21.80±1.51	**21.12**±**1.17**
	Rec./Col.	2.2/0.2	0/0	0/0	0/0

such that the goal of one environment smoothly transitions to the start of the next. Therefore the robot can traverse through all three environments in one shot. For training, we manually label the current environment for the learner. During deployment, the robot uses only one π_θ for each traversal.

In the physical experiments, we compare LLfN with three baselines: DWA, Sequential Training, and Individual Models. Sequential Training and LLfN are conducted in the same way as in the simulated experiments. Individual Models are checkpoints of Sequentially Trained models up to the corresponding environment. Since these Individual Models do not suffer from forgetting, they serve as the *best* models in Sequential Training. In contrast, the Sequentially Trained model is the final model after training on all 3 environments (thus ends up being much more effective on Environment 3 than Environment 1 or 2). The downside of Individual Models is that the number of models increases with the number of environments. Thanks to the constraint on data size, training each model takes less than two minutes on the robot's onboard CPU. We deploy the trained models to navigate in the three environments. For each method, the robot navigates each environment five times, resulting in a total of 60 physical trials. Table I reports the mean execution time for each environment with standard deviation and average number of recovery behaviors (Rec.) and collisions (Col.). DWA exhibits the most recovery behaviors, because whenever the robot fails to sample a feasible motion, it starts recovery behavior. One collision happens in one of the environment 2 and 3 trials. Applying the model sequentially learned on environment 1, 2, and 3 causes catastrophic forgetting of the first two environments. It leads to longer execution time and higher standard deviation, with frequent recovery behaviors and collisions. LLfN can successfully avoid catastrophic forgetting: It achieves a similar time to the Individual Models approach in environment 3, while, *surprisingly*, it outperforms Individual Models in environment 1 and 2 in terms of average time. One possible explanation is that the LLfN model has good backward transfer ability after gathering more diverse experience. Utilizing extra training data and models specifically trained for each environment, the Individual Models approach is more stable (lowest standard deviation). (Video of representative trials of the four methods: www.youtube.com/watch?v=ja_Rjc63xiY&t=68s.)

VI. CONCLUSION

In this letter, we propose and implement the first self-supervised Lifelong Learning for Navigation framework (LLfN). Building upon an initial static sampling-based model predictive control policy, which does not improve with increasing navigation experience, the robot is able to self-identify sub-optimal actions, search for similar scenarios where good actions are performed, learn from those data, and improve navigation in a continual manner. Furthermore, in a multi-environment setting, LLfN is able to adapt to new environments, while not forgetting previous ones. Extensive simulated trials are performed to test LLfN's in-environment and cross-environment learning capability. The entire LLfN is also implemented and tested on limited computational resources onboard a physical robot and operates in real time without requiring any off board computation. One interesting future direction is to extend the current simple heuristic-based scoring function to a more general formulation, e.g. in terms of a learnable value function. Then the scoring function will not depend on heuristics specific to the initial policy (DWA in our case). One shortcoming of the current framework is that when the base policy π_0 is incompetent everywhere, LLfN will fail to learn. Therefore future research can investigate adding active exploration when π_0 is always incompetent. Other interesting directions include investigating better methods to prioritize experiences for updating the online buffers with a limited budget and extending LLfN to address dynamic obstacles.

REFERENCES

[1] X. Xiao, J. Dufek, T. Woodbury, and R. Murphy, "Uav assisted usv visual navigation for marine mass casualty incident response," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 6105–6110.
[2] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone, "APPLD: Adaptive planner parameter learning from demonstration," *IEEE Robot. Automat. Lett.*, vol. 5, no. 3, pp. 4541–4547, Jul. 2020.
[3] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1–8.
[4] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends Cogn. Sci.*, vol. 3, no. 4, pp. 128–135, 1999.
[5] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1993, pp. 802–807.
[6] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Automat. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.
[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[8] J. J. Kuffner and S. M. LaValle, "Rrt-Connect: An efficient approach to single-query path planning," in *Proc. IEEE ICRA Millennium Conf. Int. Conf. Robot. Automat. Symp.*, vol. 2, 2000, pp. 995–1001.
[9] K. Zheng, "Ros navigation tuning guide," 2017, *arXiv:1706.09068*.
[10] M. Pfeiffer *et al.*, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robot. Automat. Lett.*, vol. 3, no. 4, pp. 4423–4430, Oct. 2018.
[11] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 31–36.
[12] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in unknown dynamic environments based on deep reinforcement learning," *Sensors*, vol. 19, no. 18, 2019, Art. no. 3837.
[13] Y. Wang, H. He, and C. Sun, "Learning to navigate through complex dynamic environment with modular deep reinforcement learning," *IEEE Trans. Games*, vol. 10, no. 4, pp. 400–412, Dec. 2018.
[14] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 6276–6283.
[15] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.
[16] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," 2017, *arXiv:1710.05627*.
[17] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 3357–3364.
[18] Y. Wang *et al.*, "Dual sequential monte carlo: Tunneling filtering and planning in continuous pomdps," 2019, *arXiv:1909.13003*.
[19] 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 *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1527–1533.
[20] D. Silver, J. A. Bagnell, and A. Stentz, "Applied imitation learning for autonomous navigation in complex natural terrain," in *Proc. Field Serv. Robot.*, 2010, pp. 249–259.
[21] S.-H. Han, H.-J. Choi, P. Benz, and J. Loaiciga, "Sensor-based mobile robot navigation via deep reinforcement learning," in *Proc. IEEE Int. Conf. Big Data Smart Comput.*, 2018, pp. 147–154.
[22] G. Wyeth and M. Milford, "Towards lifelong navigation and mapping in an office environment," in *Proc. Robot. Res.*, 2011, pp. 589–603.
[23] X. Wang *et al.*, "Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 6629–6638.
[24] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robot. Auton. Syst.*, vol. 15, no. 1–2, pp. 25–46, 1995.
[25] M. B. Ring, "Continual learning in reinforcement environments," Ph.D. dissertation, University of Texas at Austin Austin, Texas 78712, 1994.
[26] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," in *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.
[27] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. Mach. Learn. Res.*, vol. 70, 2017, Art. no. 3987.
[28] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2990–2999.
[29] A. A. Rusu *et al.*, "Progressive neural networks," 2016, *arXiv:1606.04671*.
[30] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," 2017, *arXiv:1708.01547*.
[31] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6467–6476.
[32] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 56, pp. 1633–1685, 2009.
[33] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[34] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[35] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d* algorithm," *J. Field Robot.*, vol. 23, no. 2, pp. 79–101, 2006.