Benchmarking Reinforcement Learning Techniques for Autonomous Navigation

Zifan Xu[1], Bo Liu[1], Xuesu Xiao[1], Anirudh Nair[1], and Peter Stone[1,2]

***Abstract*— While many recent successes have been reported from applying deep Reinforcement Learning (RL) for autonomous robot navigation, limitations still exist that prevent real-world use of RL-based navigation systems. For example, most learning approaches lack safety guarantees; learned navigation systems may not generalize well to unseen environments. Despite a variety of recent learning techniques to tackle these challenges in general, a lack of an open-source benchmark and reproducible learning methods specifically for autonomous navigation makes it difficult for roboticists to choose what RL algorithm to use for their mobile robots and for learning researchers to identify current shortcomings of general learning methods for autonomous navigation. In this paper, we identify four major desiderata of applying deep RL approaches for autonomous navigation: (D1) reasoning under uncertainty of partially observed sensory inputs, (D2) safety, (D3) learning from limited trial-and-error data, and (D4) generalization to diverse and novel environments. We explore four major classes of learning techniques with the purpose of achieving one or more of the four desiderata: memory-based neural network architectures (D1), safe RL (D2), model-based RL (D2, D3), and domain randomization (D4). By deploying these learning techniques in a new open-source large-scale navigation benchmark, we perform a comprehensive study of to what extent can these techniques achieve these desiderata for RL-based navigation systems. Our codebase, datasets, and experiment configurations are available at `https://github.com/Daffan/ros_jackal`.**

I. Introduction

Autonomous robot navigation has been studied by the robotics community for decades. Informally, the problem of navigation can be defined as follows (See Sec. IV-A for a formal definition):

Definition 1 (Robot Navigation Problem (Informal)): The problem of autonomous robot navigation is to create a program to move a robot efficiently from a given start location to a goal location without colliding with any obstacle.

Through extensive engineering, various classical navigation systems [1], [2] can successfully solve such navigation problem in many real-world scenarios, e.g., handling noisy, partially observable sensory input but still providing verifiable collision-free safety guarantees.

[1]Computer Science department, University of Texas at Austin, Austin, Texas 78712. [2]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.

Recently, data-driven approaches have also been used to tackle the navigation problem [3] thanks to advances in the machine learning community. In particular, Reinforcement Learning (RL), i.e., learning from self-supervised trial-and-error data, has achieved tremendous progress on multiple fronts, including safety [4]–[6], generalizability [7]–[10], sample efficiency [11], [12], and addressing temporal data [13]–[15]. For the problem of navigation, learned navigation systems from RL [16] have the potential to relieve roboticists from extensive engineering efforts [17]–[21] spent on developing and fine-tuning classical systems.

Despite these progresses in the learning community, learning-based navigation systems are far from finding their way into real-world robotics use cases, which currently still heavily rely on their classical counterparts. Such reluctance in adopting learning-based systems in the real-world stems from a series of fundamental limitations of learning methods, e.g., lack of safety, explainability, and generalizability. It has been reported that most learning-based navigation systems still underperform their classical counterparts [3].

To make things even worse, a lack of well-established comparison metrics and reproducible learning methods further obfuscates a general understanding regarding the effect of different learning approaches on navigation across both the robotics and learning community, making it even more difficult to assess the state-of-the-art and therefore to adopt learned navigation systems in the real-world.

To facilitate research in developing RL-based navigation systems with the goal of deploying them in real-world scenarios, we introduce a new open-source large-scale navigation benchmark with a variety of challenging, highly constrained obstacle courses to evaluate different learning approaches, along with the implementation of several state-of-the-art RL algorithms. We identify four major desiderata that ought to be fulfilled by any learning-based system that is to be deployed: (D1) reasoning under uncertainty of partially observed sensory inputs, (D2) safety, (D3) learning from limited trial-and-error data, and (D4) generalization to diverse and novel environments. By deploying four major classes of learning technique: memory-based neural network architectures, safe RL, model-based RL, and domain randomization, we perform extensive experiments and empirically compare a large range of RL-based methods based on the degree to which they achieve each of these desiderata. Each trial of the method runs for about 8 hours and uses 100 CPU cores for distributed training of the RL agents. The whole experiments include 120 independent trials which count for about 100000 hours of CPU time in total.

In our study, the major findings are: (1) memory-based architectures are only necessary when catastrophic failures may happen by making the wrong long-term decisions; (2) safe RL increases the chances of an agent reaching the goal location without collisions; (3) a model-based model predictive control (MPC) method achieves superior navigation performance, but tends to perform safely and conservatively when deployed in unseen test environments; (4) By increasing the number of training environments, the performance gap between training and test environments gradually shrinks, and finally saturates with about 100 training environments.

II. Desiderata for Learning-based Autonomous Navigation

In this section, we introduce four desiderata for learning-based autonomous navigation systems.

(D1) reasoning under uncertainty of partially observed sensory inputs. Autonomous navigation without explicit mapping and localization is usually formalized as a Partially Observable Markov Decision Process (POMDP), where the agent produces the motion of the robot only based on limited sensory inputs that are usually not sufficient to recover the full state of the navigation environment. Most RL approaches solve POMDPs by maintaining a history of past observations and actions [13], [14]. Then, neural network architectures like Recurrent Neural Networks (RNNs) that process sequential data are employed to encode history and address partial observability. In this study, we investigate various design choices of history-dependent architectures and history lengths.

(D2) safety. Even though in some cases deep RL methods achieve comparable performance to classical navigation, they still suffer from poor explainability and do not guarantee collision-free navigation. The lack of safety guarantee is a major challenge preventing RL-based navigation from being used in the real-world. Prior works have addressed this challenge by formalizing the navigation as a multi-objective problem that treats collision avoidance as a separate objective from reaching the goal and solving it with Lagrangian or Lyapunov-based methods [4]. In this study, we investigate how efficiently these multi-objective methods improve safety in autonomous navigation tasks.

(D3) learning from limited trial-and-error data. Although alleviating roboticists from extensive engineering effort, a large amount of data is still required to train a typical deep RL agent. However, autonomous navigation data is usually expensive to collect in the real-world. Therefore, data collection is usually conducted in simulation, e.g., in the Robot Operating System (ROS) Gazebo simulator, which provides an easy interfaces with real-world robots. However, simulating a full navigation stack from perception to actuation is more computationally expensive compared to other RL domains, e.g., MuJuCo or Atari games [22], [23]. This property presents a high requirement for sample efficiency. Most prior works have used off-policy RL algorithms to improve sample efficiency with experience replay [24], [25]. In addition, model-based RL methods can explicitly improve sample efficiency, and are widely used in robot control problems. In this study, we compare two common classes of model-based RL method [11], [12] combined with an off-policy RL algorithm, and empirically study to what extent model-based approaches improve sample efficiency when provided with different amounts of data.

(D4) generalization to diverse and novel environments. The ultimate goal of deep RL approaches for autonomous navigation is to learn a generalizable policy for all kinds of navigation environments in the real-world. However, this goal is challenging due to the limited number of navigation environments seen during training. A simple but effective strategy is to train the agent in many diverse navigation environments, but it is still unclear what is the necessary number of training environments to achieve good generalization. Utilizing the large-scale navigation benchmark proposed in this paper, we empirically study the dependence of generalization on the number of training environments.

III. Preliminaries

In Sec. III-A, a brief review of Reinforcement Learning (RL) and Markov decision processes (MDPs) is provided. Then, Sec. III-B describes the studied techniques and how they can potentially achieve the desiderata from Sec. II.

A. RL and MDPs

In RL, an agent optimizes its discounted cumulative return thorugh interactions with an environment, which is formulated as an MDP. Specifically, an MDP is a 5-tuple (S, A, T, γ, R), where S, A are the state and action spaces, $T : S \times A \longrightarrow S$ is the transition kernel that maps the agent's current state and its action to the next state, γ is a discount factor and $R : S \times A \longrightarrow \mathbb{R}$ is the reward function. The overall objective is for the agent to find a policy function $\pi : S \longrightarrow A$ such that its discounted cumulative return is maximized: $\pi^* = \arg\max_\pi \mathbb{E}_{s_t, a_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$ [26].

MDPs assume the agent has access to the world state s which encapsulates sufficient information for making optimal decisions. However, in real applications, the agent often only perceives part of the state s at any moment. Such partial observability leads to uncertainty in the world state and the problem becomes a Partially Observable Markov decision process (POMDP). A POMDP is a 7-tuple $(S, A, O, T, \gamma, R, Z)$. In addition to the elements of an MDP, O denotes the observation space and $Z : S \longrightarrow O$ is an observation model that maps the world state to an observation. For instance, at each time step t, the agent receives an observation $o_t \sim Z(\cdot \mid s_t)$. In general, solving a POMDP optimally requires taking the entire history into consideration, which means the objective is then to find a policy that maps its past trajectory $\tau_t = (o_0, a_0, \ldots, o_t)$ to an action a_t such that $\max_\pi \mathbb{E}_{\tau_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$.

B. Studied Techniques

In this section, we introduce the four types of techniques we implement to potentially achieve the desiderata in Sec. II.

Memory-based Neural Network Architectures (D1). Due to the uncertainty from partial observations (e.g. caused by dynamic obstacles or imperfect sensory inputs), a mobile agent often needs to aggregate the information along its trajectory history for successful navigation. When using a parameterized model as the policy, recurrent neural networks (RNNs) such as those incorporating Long-Short Term Memory (LSTM) [27] or Gated Recurrent Units (GRUs) [28] are widely adopted for solving POMDPs [13], [14]. More recently, transformers, a type of deep neural architecture that use multiple layers of attention mechanism to process sequence data, have been proposed. Transformers have demonstrated superior performance over RNN-based models in vision and natural language applications [29]. In this work, we consider both GRU and transformers as the backbone model for the navigation policy. The reason for choosing GRU over LSTM is due to the fact that GRU has a simpler architecture than, but performs comparably with, LSTM in practice [28].

Safe RL (D2). According to Definition 1, successful navigation involves both navigating efficiently towards a user specified goal and avoiding collisions. While most prior RL-based navigation approaches design a *single* reward function that summarizes both objectives, it is not clear whether explicitly treating the two objectives separately will have any benefits. Specifically, assume the reward function R only rewards the agent for making progress to the goal. Additionally, a cost function $C : S \times A \longrightarrow \mathbb{R}^+$ maps a state s and the agent's action a to a penalty c. Then the navigation problem can be transformed into a constrained optimization problem with 2 objectives [30]:

$$\max_{\pi} \mathbb{E}_{s_t,a_t\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\Big] \text{ s.t. } \mathbb{E}_{s_t,a_t\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t C(s_t,a_t)\Big] \leq \epsilon. \tag{1}$$

Here $\epsilon \geq 0$ is a threshold that controls how tolerant we are of the risk of collision. By formulating the navigation problem in this way, existing constrained optimization techniques can be applied for solving (1). One of the most common approaches in constrained optimization is using a Lagrangian multiplier, which transforms the constraint into a penalty term multiplied by a Lagrangian multiplier $\lambda \geq 0$. Specifically, the objective becomes

$$\max_{\pi} \mathbb{E}_{s_t,a_t\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\Big] + \lambda\Big(\mathbb{E}_{s_t,a_t\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t C(s_t,a_t)\Big] - \epsilon\Big). \tag{2}$$

To solve (2), one can either manually specify a λ based on prior knowledge or optimize λ simultaneously [31]. In this work, we perform a grid search over λ but fix it during learning.

Model-based RL (D2, D3). Given an accurate model, model-based RL often benefits from better sample efficiency compared to model-free methods [32]. In addition to improved sample efficiency, prior work on model-based RL also reported that planning with a learned model can improve the agent's safety [5]. Therefore, we investigate whether these claims also hold for autonomous navigation if the agent learns a transition model $\hat{T}$ from its interaction with the world. Although the reward model R can also be learned, as the structure of the reward function is usually designed manually, we let the agent take advantage of the known reward function. In this work, we consider two ways of using a learned model: a Dyna-style method [33] and model-predictive control (MPC) [34]. Specifically, assume the agent's rollout trajectories are saved into a replay buffer B in the form of transition tuples: $B = \{(s_t, a_t, s_{t+1}, r_t)\}_t$.[1] Then model-based methods assume the agent also learns a model $\hat{T} : S \times A \longrightarrow S$ that approximates T. A common learning objective of $\hat{T}$ is to minimize the mean-square-error between $\hat{T}$'s and T's predictions on transitions:

$$\hat{T} = \arg\min_{T'} \quad \mathbb{E}_{(s,a,s',r)\sim B}\big\|T'(s,a) - s'\big\|_2^2. \tag{3}$$

Once $\hat{T}$ is learned, given a transition pair $(s, a) \sim B$, the Dyna-style method samples additional $s' \sim \hat{T}(s, a)$ to enrich the replay buffer that can potentially benefit the learning of the value function. MPC, on the other hand, first uses $\hat{T}$ to form samples of future trajectories, then it outputs the first action corresponding to the trajectory that has the highest return.

Domain randomization (D4). A direct deployment of a policy trained in limited training environments to unseen target environments is usually formalized as a zero-shot transfer problem, where no extra training of the policy is allowed in the target environments. One promising approach for zero-shot transfer has been Domain Randomization (DR) [10]. In DR, the environment parameters (i.e. obstacle configurations) in predefined ranges are randomly selected in each training environment. By randomizing everything that might vary in the target environments, the generalization can be improved by covering target environments as variations of random training environments.

IV. Navigation Benchmark

This section details the proposed navigation benchmark for RL-based navigation systems, which aims to provide a unified and comprehensive testbed for future autonomous navigation research. In Sec. IV-A and IV-B, the navigation task is formally defined and formulated as a POMDP. Then Sec. IV-C describes the procedures to generate three types of navigation environments that benchmark different aspects of navigation performance.

A. Navigation Problem Definition

Building from the informal Definition 1, a navigation problem can be formally defined as follows:

Definition 2 (Robot Navigation Problem (Formal)): Situated within a navigation environment e which includes information of all the obstacle locations at any time t, a start location (x_i, y_i), a start orientation θ_i, and a goal location (x_g, y_g), the navigation problem $\mathcal{T}_e$ is to maximize the

[1] Having a replay buffer is a commonly used technique to learn the value function (critic) in reinforcement learning.

probability p of a mobile robot reaching the goal location from the start location and orientation under a constraint on the number of collisions with any obstacle $C < 1$ and a time limit $t < T_{max}$.

Given the current location (x_t, y_t), the robot is considered to have reached the goal location if and only if its distance to the goal location is smaller than a threshold, $d_t < d_s$, where d_t is the Euclidean distance between (x_t, y_t) and (x_g, y_g), and d_s is a constant threshold.

B. POMDP Formulation

In accordance with the definition of POMDPs from Sec. III-A, a navigation task $\mathcal{T}_e$ can be formulated as a POMDP represented by a 7-tuple $(S_e, A_e, O_e, T_e, \gamma_e, R_e, Z_e)$ conditioned on the navigation environment e. In this POMDP, the state $s_t \in S_e$ is a 5-tuple $(x_t, y_t, \theta_t, c_t, e)$ with x_t, y_t, θ_t the two-dimensional coordinates and the orientation of the robot at time step t, c_t a binary indicator of whether a collision has occurred since the last time step $t-1$, and e the navigation environment. The action $a_t = (v_t, \omega_t) \in A_e$ is a two-dimensional continuous vector that encodes the robot's linear and angular velocity. The observation $o_t = (\chi_t, \bar{x}_t, \bar{y}_t) \in O_e$ is a 3-tuple composed of the sensory input χ_t from LiDAR scans and the relative goal position $(\bar{x}_t, \bar{y}_t)$ in the robot frame. The observation model Z depends on the model of the LiDAR and the physical geometry of the LiDAR and the robot. The reward function for this POMDP is defined as follows:

$$R_e(s_t, a_t) = +b_f \cdot \mathbb{1}(d_t < d_s) + b_p \cdot (d_{t-1} - d_t) - b_c \cdot c_t, \quad (4)$$

where $\mathbb{1}(d_t < d_s)$ is the indicator function of reaching the goal location, d_t is the Euclidean distance to the goal location, and b_f, b_p, b_c are the coefficient constants. In this reward function, the first term is the true reward function that assigns a positive constant b_f for the success of an agent, which matches with the objective of the navigation task in Definition 2. The second and third terms are auxiliary rewards that facilitate the training by encouraging local progress and penalizing collisions.

We perform a grid search over different values of the coefficients in this reward function, and the result shows that the auxiliary reward term $(d_{t-1} - d_t)$ is necessary for successful training, and a much smaller coefficient b_p relative to b_f can achieve a better asymptotic performance. The agent can learn without the penalty reward for collision ($b_c = 0$), but a moderate value of b_c can improve the asymptotic performance and speed up training. For all the experiments in this paper, we fix the coefficients as $b_f = 20$, $b_p = 1$ and $b_c = 4$.

In our experiments, the RL algorithm solves a multi-task RL problem where the tasks are randomly sampled from a task distribution $\mathcal{T}_e \sim p(\mathcal{T}_e)$. Here the task distribution $p(\mathcal{T}_e) := U(\{e_i\}_{i=1}^N)$ is a uniform distribution on a set of N navigation environments $\{e_i\}_{i=1}^N$. To extend the objective defined in Sec. III-A, the overall objective of this multi-task RL problem is to find an optimal policy $\pi^* = \max_\pi \mathbb{E}_{\mathcal{T}_e \sim p(\mathcal{T}_e)}[\mathbb{E}_{\tau_t \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R_e(s_t, a_t)]]$

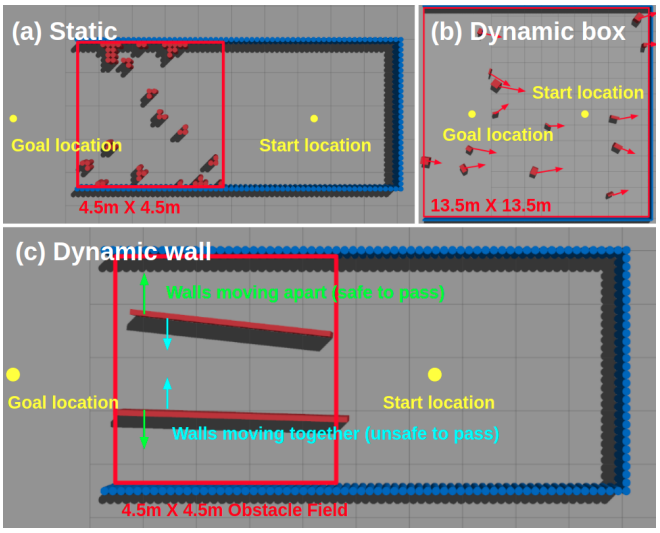

Fig. 1: Three types of navigation environments: (a) static, (b) dynamic box and (c) dynamic wall. The red squares mark the obstacle fields, and the yellow circles mark the start and goal locations. In figure (a), the green (blue) arrows indicate the case when the two walls are moving apart (together). In figure (c), the red arrows indicate the velocities of obstacles.

C. Simulation Specifications and Navigation Environments

The navigation is performed by a ClearPath Jackal differential-drive ground robot in navigation environments simulated by the Gazebo simulator. The robot is equipped with a 720-dimensional planar laser scan with a 270° field of view, which is used as our sensory input χ_t in Sec. IV-B. We preprocess the LiDAR scans by capping the maximum range to 5m which to covers the entire obstacle field. The goal location $(\bar{x}_t, \bar{y}_t)$ is inquired directly from the Gazebo simulator.

The RL agent navigates the robot through a 10m navigation path that passes through a highly constrained obstacle field. Walls are placed in the environment so that passing through the obstacle field is the only path to the goal location (see Fig. 1). Three types of environment are presented in this benchmark: static, dynamic box, and dynamic wall environments. In the remaining part of this section, we detail the procedures of generating these environments.

Static environments. We use the 300 static environments from the BARN dataset [35]. The obstacle fields in these environments are represented by a 30×30 grid, which corresponds to an area of 4.5m×4.5m (see Fig. 1(b)). The grid cells are either kept empty or filled with cylinders using Celluar Automata. Details can be found in the BARN dataset [35]. Among 300 static environments, we randomly select 50 environments as the test set, which we denote as `static-test`. We then randomly select 5, 10, 50, 100, and 250 of the remaining environments as the training sets of different sizes. We denote them as `static-train-5`, `static-train-10`, `static-train-50`, `static-train-100`, and `static-train-250` respectively.

Dynamic box environments. These environments are $13.5m \times 13.5m$ obstacle fields (larger than the static environments) that give the agent more time to respond to the moving obstacles (see Fig. 1(c)). The obstacles are randomly generated without any manually designed challenging scenarios. Each obstacle is a $w \times l \times h$ box with its width w and length l randomly sampled from a range of $[0.1m, 0.5m]$ and a height $h = 1m$. The obstacles start from a random position on the left edge of the obstacle field with a random orientation and a constant linear velocity. The magnitude of the velocity is randomly sampled from a range of $[1m/s, 1.5m/s]$, and the direction of the velocity is randomly sample from all the possible directions pointing into the obstacle field. Each obstacle repeats its motion once it moves out of the obstacle field. A dynamic environment has 10 to 15 such randomly generated obstacles. We randomly generate 100 instances of such dynamic box environments with 50 as the training set and the remaining 50 as the test set, which we denote as `dynamic-box-train` and `dynamic-box-test` respectively.

Dynamic wall environments. These environments are $4.5m \times 4.5m$, with two long parallel walls moving in opposite directions with their velocities perpendicular to the start-goal direction (see Fig. 1(a)). The walls are long enough so that the robot can only pass when the two walls are moving apart. This manually designed navigation scenario requires the agent to maintain a memory of past observations and actions, and estimate the motion of obstacles. To challenge the agent, we add small variances so that each wall's length, tilting angle, and magnitude of the velocity are randomly sampled from the ranges of $[3.5m, 4.5m]$, $[-10°, 10°]$ and $[1m/s, 1.4m/s]$ respectively. 100 instances of such parallel wall environments are generated with 50 as the training set and the remaining 50 as the test set, which we denote as `dynamic-wall-test` and `dynamic-wall-train` respectively.

V. Experiments

In this section, we present experimental results of each studied technique to achieve the proposed desiderata in Sec. II. Unless otherwise specified, all the experiments mentioned in this section use a distributed TD3 RL algorithm (similar to [20]) combined with the corresponding techniques, and all the 40 data points presented are averaged over three independent runs.

A. Memory-based Neural Network Architectures (D1)

To benchmark the performance of different neural network (NN) architectures, deep RL policies represented by architectures of Multilayer Perceptron (MLP), One-dimensional Convolutional Neural Network (CNN), Gated Recurrent Units (GRU), and Transformer with history length of 4 and 8 are trained in `static-train-50`, and the two types of dynamic environments `dynamic-box-train` and `dynamic-wall-train` from Sec. IV-C. After training, the policies are tested in their corresponding test sets. In addition, MLP with history length of one is added as a memory-less baseline. Table I shows the success rates of policies with different architectures and history lengths evaluated in the `static-test` (top) and `dynamic-wall-test` (bottom) respectively. While we do not present test results of `dynamic-box` in Table I, no significant difference is observed between different NN architectures and history lengths in this environment set.

History length	1	4	8
MLP	**65 ± 4%**	57 ± 7%	42 ± 2%
GRU	-	51 ± 2%	43 ± 4%
CNN	-	55 ± 4%	45 ± 5%
Transformer	-	**68 ± 2%**	46 ± 3%

History length	1	4	8
MLP	67 ± 7%	72 ± 1%	69 ± 4%
GRU	-	**82 ± 4%**	**78 ± 5%**
CNN	-	63 ± 3%	43 ± 3%
Transformer	-	33 ± 28%	15 ± 13%

TABLE I: Success rate of policies trained with different neural network architectures and history lengths in `static` (top) and `dynamic-wall` (bottom) environments.

Memory-based NNs do not improve navigation performance in static environments. In Table I, the policy represented by Transformer with a history length of 4 shows the best success rate of 68%, however a similar success rate is achieved by the MLP baseline within the error of standard deviation. Additionally, a monotonic decrease in success rate with increasing history length is observed in each tested NN architecture. For example, a 32% drop in the success rate of Transformer is shown by increasing the history length from 4 to 8. One possible explanation is that, if only the current one single observation is useful to make the decision, including past observations will make it more difficult to extract useful information.

Memory only matters when possible catastrophic failures will happen by making the wrong long-term decisions. Memory usually matters for dynamic environments when a single time frame is not sufficient to estimate the motion of obstacles. Surprisingly, in `dynamic-box` where the dynamic obstacles are completely random, the memory-based NN architectures do not outperform the memory-less baseline. In addition, unlike `static`, the performances does not drop by increasing the history length. On the other hand, in `dynamic-wall` with a manually designed dynamic challenge, the best success rate of 82% is observed in GRU with a history length of 4, which improves about 15% over the non-memory baseline. During our deployment of the policies, we observe that, in `dynamic-box` even though the memory-less agent does not estimate the motion and adjust its plan in advance, it tends to perform safely and avoids the obstacles when they get close enough. This simple strategy works surprisingly well and achieves similar success rate as the memory-based policies. However, this strategy does not work in the manually designed dynamic challenges like `dynamic-wall` where the agent has to estimate the motion

Safe-RL method	MLP	Lagrangian	MPC	DWA
Success rate	$65 \pm 4\%$	$\mathbf{74 \pm 2\%}$	$70 \pm 3\%$	43%
Survival time	$8.0 \pm 1.5s$	$16.2 \pm 2.5s$	$55.7 \pm 4.9s$	$\mathbf{88.6s}$
Traversal time	$\mathbf{7.5 \pm 0.3s}$	$8.6 \pm 0.2s$	$24.7 \pm 2.0s$	$38.5s$

TABLE II: Success rate, survival time and traversal time of policies trained with different safe-RL methods, MPC with probabilistic transition model and DWA.

of the obstacles to pass safely.

B. Safe RL (D2)

To investigate to what extent safe RL methods can help to improve safety, a TD3 agent with the Lagrangian-based safe RL method (Eq. 2) is trained in `static-train-50`, and then tested in `static-test`. The policy is represented by a MLP with its input containing only one history length. Table II shows the success rate, average survival time, and average traversal time of safe RL agents, a vanilla MLP agent. We define survival time as the time cost of an unsuccessful episode either collided or timeout. Traversal time, instead, is the time cost of a successful episode. With the same level of success rate, a longer survival time means that the agent tends to, at least, avoid collisions if it can not succeed. To compare the safe RL methods with classical navigation systems which are believed to have better safety, we also add evaluation metrics from a classical navigation stack with Dynamic Window Approach (DWA) [2] local planner.

Safe RL methods reduce the gap between training and test environments. Our experiments indicate that even though both the vanilla MLP and the safe RL method achieve about 90% success rate in the training environments, the safe RL method has a better success rate in the test environments. We hypothesize that the safety constraint applied by the safe RL methods forms a way of regularization, and therefore, improves the generalization to unseen environments.

Safe RL methods increase the average survival time in failed episodes. As expected, the safe RL method increases the average survival time by $8.2s$ compared to the vanilla MLP at a cost of $1.1s$ longer average traversal time. However, such improved safety are still much worse than the classical navigation systems given the best survival time of $88.6s$ achieved by DWA.

C. Model-based RL (D2 and D3)

We study two major model-based RL methods: Dyna-style and MPC. Dyna-style methods intend to improve sample efficiency by simulate transition samples with learned transition models. Instead, MPC methods utilize the models to plan for a fixed future time horizon and select the best actions to execute. MPC methods usually enable a more efficient exploration by selecting promising actions, and also potentially improve the asymptotic performance with the help of model predictions.

To explore how these model-based approaches help with the autonomous navigation tasks, we implement both Dyna-style and MPC methods, and evaluate the methods in static environments. The transition models are either represented by a deterministic NN or a probabilistic NN that predicts the mean and variance of the next state. During the training in `static-train-50`, the policies are saved when 100k, 500k and 2000k transition samples are collected, then tested in `static-test`. The success rates of these policies are reported in Table III.

Model-based methods do not improve sample efficiency. As shown in the second and third columns in Table III, better success rates of 13% and 58% are achieved by the vanilla MLP method provided by limited 100k and 500k transition samples respectively. In addition, Higher success rates at 500k transition samples are observed in probabilistic models compared to their deterministic counterparts, which indicates a more efficient learning with probabilistic transition models.

Model-based methods with probabilistic dynamic models improve the asymptotic performance. In the last column of Table III, both Dyna-style and MPC with probabilistic dynamic models achieve slightly better success rates of 70% compared to 65% in the vanilla MLP method when sufficient transition samples of 2000k are given to the learning agent.

The MPC policy performs conservatively when deployed in unseen test environments and shows a better safety performance. The safety performances of MPC policies with probabilistic dynamic models are also tested (see Table II). We observe that the agents with MPC policies navigate very conservatively with an average traversal time of $24.7s$, which is about two times more than the MLP baseline. In the meantime, MPC policies achieve improved safety with the best survival time of $55.7s$ among the RL-based methods.

D. Domain Randomization (D4)

To explore how the generalization depends on the degree of randomness in the training environments, baseline MLP policies with one history length are trained in the environment sets with four different sizes: `static-train-5`, `static-train-10`, `static-train-50` and `static-train-100`. The trained policies are tested in the same `static-test` environments. Table IV shows the success rate of policies trained with different number of training environments.

The performance gap between training and test environments gradually shrinks and finally saturates with increasing number of training environments. As shown in Table IV, a large performance gap of around 56% percentage is shown in 5 training environments. Even though this gap gradually decreases to 15% in 100 training environments, further increasing the number of training environments to 250 does not significantly reduce the gap anymore.

VI. Discussion and Conclusion

In this paper, we identify four desiderata for RL-based navigation systems, and evaluate four classes of commonly used learning techniques that are promising for achieving the desiderata with a novel large-scale navigation benchmark.

Transition samples	100k	500k	2000k
MLP	**13 ± 7%**	**58 ± 2%**	65 ± 4%
Dyna-style deterministic	8 ± 2%	30 ± 10%	66 ± 5%
MPC deterministic	0 ± 0%	21 ± 10%	62 ± 3%
Dyna-style probabilistic	0 ± 0%	48 ± 4%	**70 ± 1%**
MPC probabilistic	0 ± 0%	45 ± 4%	**70 ± 3%**

TABLE III: Success rate of policies trained with different model-based methods and different number of transition samples.

Environments	5	10	50	100	250
Success rate	43 ± 3%	54 ± 8%	65 ± 4%	72 ± 6%	74 ± 2%

TABLE IV: Success rate of policies trained with different different number of training environments.

In this section, we point out potential future directions to fully achieve all desiderata and to build a robust RL-based navigation system that is deployable in the real-world.

(D1) reasoning under uncertainty of partially observed sensory inputs might not be so important in static environments, and even in very random dynamic environments. Our results indicate that roboticists should consider the characteristics of the deployment environment to determine if history is really necessary. Even though the benefit of memory is demonstrated in one specially designed dynamic challenge where correct long-term decision making is essential, this one single challenge does not cover all the possible challenging dynamic navigation scenarios in the real-world. To further demonstrate and evaluate the ability of memory-based systems to reason under uncertainty, more diverse but also challenging dynamic environments will be needed in the future.

(D2) safety is improved by both safe RL and model-based MPC methods. However, classical navigation systems still achieve the best safety performance at a cost of very long traversal time and relatively low success rate. Whether RL-based navigation systems can achieve similar level of safety guarantee as classical navigation systems and whether safety can be improved without significantly sacrificing the traversal time and success rate are still open questions.

(D3) the ability to learn from limited trial-and-error data is not improved by the proposed model-based methods. However, our implementation is based on an off-policy TD3 algorithm which already ensures a relatively good sample efficiency. Since we did not include on-policy RL algorithms in this study, it is still unclear whether model-based methods combined with on-policy RL algorithms can achieve better sample efficiency or not.

(D4) the generalization to diverse and novel environments is improved by increasing the randomness of training environments. However, a noticeable gap of about 10% between training and test environments can not be eliminated by further increasing the randomness. Some recent advances in open-ended curriculum learning [36] are reported to more actively improve the generalization by using teachers to control the environment generation process and propose novel environments to train RL agents, which forms promising future directions to further improve the generalization performance reported in this paper.

To conclude, we introduce a new open-source large-scale navigation benchmark for RL-based navigation systems that contains a variety of challenging navigation environments, and report on a detailed comparison between different navigation systems. Additionally, we identify four major desiderata for RL-based navigation systems and compare the effectiveness of four major classes of learning techniques at achieving the desiderata using the proposed navigation benchmark.

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] 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.

[4] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. A. Duéñez-Guzmán, "Lyapunov-based safe policy optimization for continuous control," *CoRR*, vol. abs/1901.10031, 2019. [Online]. Available: http://arxiv.org/abs/1901.10031

[5] G. Thomas, Y. Luo, and T. Ma, "Safe reinforcement learning by imagining the near future," 2022.

[6] E. Rodríguez-Seda, D. Stipanovic, and M. Spong, "Lyapunov-based cooperative avoidance control for multiple lagrangian systems with bounded sensing uncertainties," in *2011 50th IEEE Conference on Decision and Control and European Control Conference, CDC-ECC 2011*, ser. Proceedings of the IEEE Conference on Decision and Control, Dec. 2011, pp. 4207–4213, 2011 50th IEEE Conference on Decision and Control and European Control Conference, CDC-ECC 2011 ; Conference date: 12-12-2011 Through 15-12-2011.

[7] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *ICML*, 2019.

[8] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," *arXiv preprint arXiv:1912.01588*, 2019.

[9] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating generalization in deep reinforcement learning through procedural level generation," *arXiv: Learning*, 2018.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[11] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bull.*, vol. 2, no. 4, p. 160–163, jul 1991. [Online]. Available: https://doi.org/10.1145/122344.122377

[12] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[13] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Fall Symposia*, 2015.

[14] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Solving deep memory pomdps with recurrent policy gradients," in *ICANN*, 2007.

[15] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[16] 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.

[17] 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.

[18] 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.

[19] Z. Wang, X. Xiao, G. Warnell, and P. Stone, "Apple: Adaptive planner parameter learning from evaluative feedback," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7744–7749, 2021.

[20] 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.

[21] X. Xiao, Z. Wang, Z. Xu, B. Liu, G. Warnell, G. Dhamankar, A. Nair, and P. Stone, "Appl: Adaptive planner parameter learning," *arXiv preprint arXiv:2105.07620*, 2021.

[22] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[24] 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, pp. 2007–2014, 2019.

[25] A. Wahid, A. Toshev, M. Fiser, and T.-W. E. Lee, "Long range neural navigation policies for the real world," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 82–89, 2019.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] E. Altman, *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[31] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[32] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of international conference on robotics and automation*, vol. 4. IEEE, 1997, pp. 3557–3564.

[33] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.

[34] J. A. Rossiter, *Model-based predictive control: a practical approach*. CRC press, 2017.

[35] 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.

[36] M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, and S. Levine, "Emergent complexity and zero-shot transfer via unsupervised environment design," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 049–13 061, 2020.