# Benchmarking Reinforcement Learning Techniques for Autonomous Navigation

**Anonymous Author(s)**
Affiliation
Address
email

**Abstract:** Deep reinforcement learning (RL) has brought many successes for autonomous robot navigation. However, there still exists important limitations that prevent *real-world* use of RL-based navigation systems. For example, most learning approaches lack safety guarantees; and 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 learning methods 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*, (D2) *safety*, (D3) *learning from limited trial-and-error data*, and (D4) *generalization to diverse and novel environments*. Then, 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 and real-world environments, we perform a comprehensive study aimed at establishing to what extent can these techniques achieve these desiderata for RL-based navigation systems.

**Keywords:** RL, Autonomous navigation, Benchmark

## 1 Introduction

Autonomous robot navigation, i.e., moving a robot from one point to another without colliding with any obstacle, has been studied by the robotics community for decades. 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. However, these systems require extensive engineering effort, and can still be brittle in challenging scenarios, e.g., in highly constrained environments [3]. Recently, data-driven approaches have also been used to tackle the navigation problem [4] 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 [5, 6, 7], generalizability [8, 9, 10, 11], sample efficiency [12, 13], and addressing temporal data [14, 15, 16]. For the problem of navigation, learned navigation systems from RL [17] have the potential to relieve roboticists from extensive engineering efforts [18, 19, 20, 21, 22] spent on developing and fine-tuning classical systems. Moreover, a simple case study conducted in five randomly generated obstacle courses where classical navigation systems often fail shows that a RL-based navigation has the potential to achieve superior behaviors in terms of both navigation efficiency and collision avoidance (Fig. 1 left).

Despite such promising advantages, learning-based navigation systems are far from finding their way into real-world robotics use cases, which currently still rely heavily on their classical counter-

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

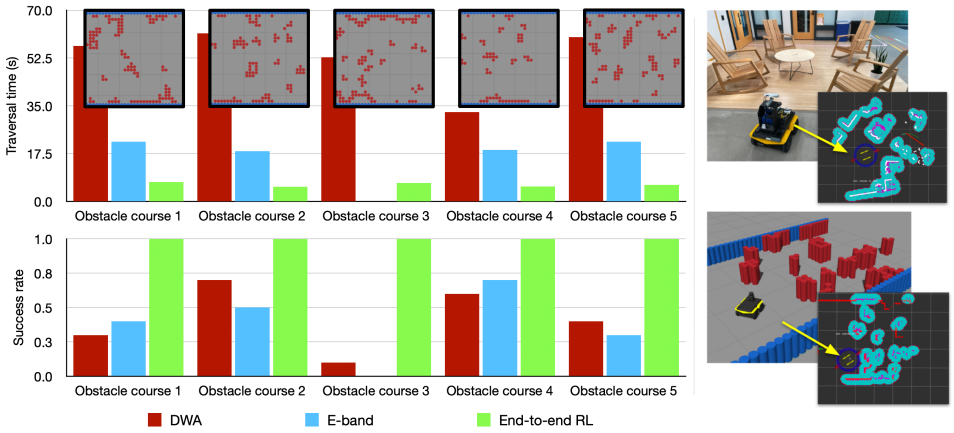


Figure 1: **Left:** Traversal time (top) and success rate (bottom) of two classical navigation systems, DWA [2] (red) and E-band [1] (blue), and vanilla end-to-end RL-based navigation systems (individually trained) in five randomly generated difficult obstacle courses (green). The insets at the top show top-down views of the five obstacle courses. **Right:** Navigation environments in the real world (top) and the proposed benchmark (bottom) are similar to the robot perception system (e.g., white/red laser scans and cyan/purple costmaps).

parts. 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. To make things even worse, a lack of well-established comparison metrics and reproducible learning methods further obfuscates the effects of different learning approaches on navigation across both the robotics and learning communities, making it 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. The obstacle courses resemble highly-constraint real-world navigation environments (right of Fig. 1), and present major challenges to existing classical navigation systems, while RL-based navigation systems have the potential to perform well in them (left of Fig. 1).

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 techniques: *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 set of experiments includes 120 independent trials which account for about 100000 hours of CPU time in total. Moreover, by deploying 6 selected navigation systems in three qualitatively different real-world navigation environments, we investigate to what degree the conclusions drawn from the benchmark can be applied to the real world.

## 2 Desiderata for Learning-based Autonomous Navigation

In this section, we introduce four desiderata for learning-based autonomous navigation systems and briefly discuss the learning techniques as their corresponding solutions. More detailed descriptions of these techniques can be found in Appendix A.

**(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 [14, 15]. 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 [5]. For simplicity, we only explore Lagrangian method and investigate whether explicitly treat safety as a separate objective leads to safer and smoother learned navigation behavior.

**(D3) learning from limited trial-and-error data.** Although deep RL approaches can alleviate 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 interface 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 [23, 24], which presents a high requirement for sample efficiency. Most prior works have used off-policy RL algorithms to improve sample efficiency with experience replay [25, 26]. 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 [12, 13] 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. A simple strategy is to train the agent in as many diverse navigation environments as possible or domain randomization, but it is unclear what is the necessary amount of training environments to efficiently 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.

# 3 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. 3.1 and 3.2, the navigation task is formally defined and formulated as a POMDP. More detailed background of MDP and POMDP can be found in Appendix A.1. Finally, Sec. 3.3 introduces simulated and real-world environments that benchmark different aspects of navigation performance.

## 3.1 Navigation Problem Definition

**Definition 1 (Robot Navigation Problem)** *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 $\theta_i$, and a goal location $(x_g, y_g)$, the navigation problem $\mathcal{T}_e$ is to maximize the 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 $t < T_{max}$.*

A navigation problem can be formally defined as above. 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.

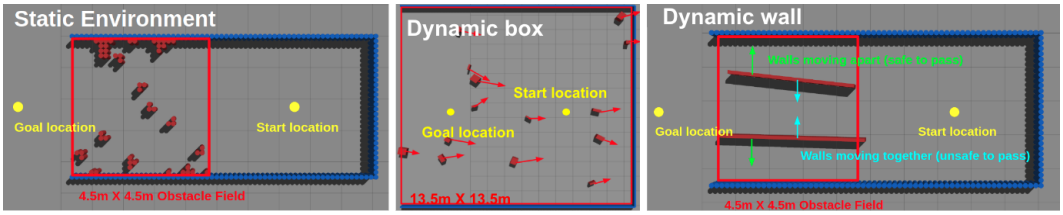


Figure 2: Three types of navigation environments: `static`, `dynamic box`, and `dynamic-wall`. The red squares mark the obstacle fields, and the yellow circles mark the start and goal locations. In `dynamic-wall`, the green (blue) arrows indicate the case when the two walls are moving apart (together). In `dynamic box`, the red arrows indicate the velocities of obstacles.

### 3.2 POMDP Formulation

A navigation task $\mathcal{T}_e$ can be formulated as a POMDP conditioned on a navigation environment $e$, which can be represented by a 7-tuple $(S_e, A_e, O_e, T_e, \gamma_e, R_e, Z_e)$ . In this POMDP, the state $s_t \in S_e$ is a 5-tuple $(x_t, y_t, \theta_t, c_t, e)$ with $x_t, y_t, \theta_t$ 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 $\chi_t$ from LiDAR scans and the relative goal position $(\bar{x}_t, \bar{y}_t)$ in the robot frame. The observation model $Z : S \to O$ maps the state to the observation. The reward function for this POMDP is defined as follows:

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

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 1. 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 lead to 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 $\{e_i\}_{i=1}^N$. The overall objective of this multi-task RL problem is to find an optimal policy $\pi^* = \max_\pi \mathbb{E}_{\mathcal{T}_e \sim p(\mathcal{T}_e), \tau_t \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R_e(s_t, a_t)\right]$.

### 3.3 Navigation Environments

The navigation is performed by a ClearPath Jackal differential-drive ground robot in simulated by the Gazebo simulator. More details of the robot and simulation can be found in Appendix C. Each environment in this benchmark will have a navigation system navigating the robot through a 10m navigation path that passes through a highly constrained obstacle course. Walls are placed at three edges of a square so that passing through the obstacle field is the only path to the goal location (see Fig. 2). The benchmark includes 300 `static` environments, 100 `dynamic-box` environments, and 100 `dynamic-wall` environments. The `static` environments contains a diverse set of obstacle course covering a large range of difficulty levels from easy to hard. A `dynamic-box` environment has small boxes with random shapes and velocities to test the system's immediate reactions to small moving obstacles. A `dynamic-wall` has two walls moving oppositely that requires the system to make a longer-term decision of whether to pass or wait. The detailed procedures of generating these environments can be found in the Appendix

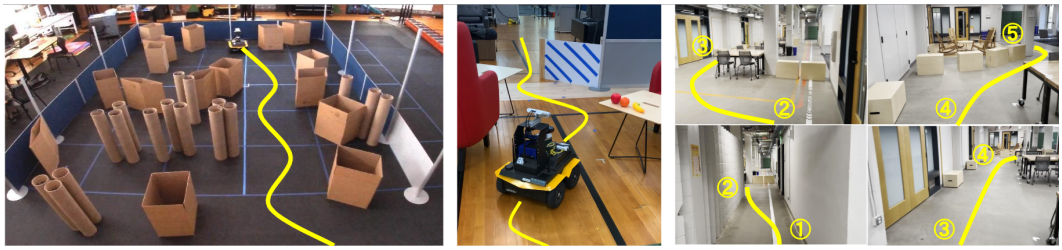

Figure 3: Real-world benchmark-like (left), in-door highly-constrained (middle), and large-scale (right) environments. The yellow curves mark roughly the paths of navigation.

B. We randomly select 50 environments from each type as the test sets, which are denoted as `static-test`, `dynamic-box-test`, and `dynamic-wall-test`. The remaining environments are denoted as `static-train`, `dynamic-box-train`, and `dynamic-wall-train` respectively. To study the effect of randomization, `static-train` is further separated as `static-train-5`, `static-train-10`, `static-train-50`, `static-train-100`, and `static-train-250` by randomly sampling 5, 10, 50, 100, and all 250 environments from `static-train`.

To test the sim-to-real transferability of the policies learning with different techniques, the navigation systems are deployed in three qualitatively different static navigation environments including a benchmark-like environment (Fig. 3 left), an indoor highly-constrained environment (Fig. 3 right), and a large-scale environment of 30 meters in length. We denote them as `real-world-1`, `real-world-2`, and `real-world-3` respectively.

# 4 Experiments

In this section, we present experimental results of each studied technique to achieve the proposed desiderata in Sec. 2. Unless otherwise specified, all the experiments mentioned in this section use a distributed TD3 RL algorithm (similar to [21]) combined with the corresponding techniques, and all the data points presented are averaged over three independent runs.

| | `static env.` | | | `dynamic-box env.` | | | `dynamic-wall env.` | | |
|---|---|---|---|---|---|---|---|---|---|
| | $H=1$ | $H=4$ | $H=8$ | $H=1$ | $H=4$ | $H=8$ | $H=1$ | $H=4$ | $H=8$ |
| MLP | $65 \pm 4$ | $57 \pm 7$ | $42 \pm 2$ | $50 \pm 5$ | $35 \pm 2$ | $46 \pm 3$ | $67 \pm 7$ | $72 \pm 1$ | $69 \pm 4$ |
| GRU | - | $51 \pm 2$ | $43 \pm 4$ | - | $48 \pm 4$ | $45 \pm 1$ | - | $\mathbf{82 \pm 4}$ | $78 \pm 5$ |
| CNN | - | $55 \pm 4$ | $45 \pm 5$ | - | $42 \pm 5$ | $40 \pm 1$ | - | $63 \pm 3$ | $43 \pm 3$ |
| Transformer | - | $\mathbf{68 \pm 2}$ | $46 \pm 3$ | - | $\mathbf{52 \pm 1}$ | $44 \pm 4$ | - | $33 \pm 28$ | $15 \pm 13$ |

Table 1: (D1) Success rate (%) of policies trained with different neural network architectures and history lengths in `static`, `dynamic-box`, and `dynamic-wall` environments.

| Methods | Baseline (model-free) | Lagrangian method | MPC (model-based) | DWA |
|---|---|---|---|---|
| Success rate (%) | $65 \pm 4$ | $74 \pm 2$ | $70 \pm 3$ | **82** |
| Survival time (s) | $8.0 \pm 1.5$ | $16.2 \pm 2.5$ | $55.7 \pm 4.9$ | **62.7** |
| Traversal time (s) | $\mathbf{7.5 \pm 0.3}$ | $8.6 \pm 0.2$ | $24.7 \pm 2.0$ | 35.6 |

Table 2: (D2) Success rate, survival time, and traversal time of policies trained with Lagrangian method, MPC with probabilistic transition model, and DWA.

## 4.1 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. 3.3. 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

1 shows the success rates of policies with different architectures and history lengths evaluated in `static-test` ( left), `dynamic-wall-test` (middle) and `dynamic-box-test` respectively.

**Memory-based NNs only marginally improve navigation performance in static environments.** In Table 1, the policy represented by Transformer with a history length of 4 shows the best success rate of $68\%$, with a slightly worse success rate of $65\%$ achieved by the baseline MLP. 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 few past observations are useful to make the decision, including more history will make it more difficult to learn a generalized policy in this very diverse training set.

**Memory is essential 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. 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 of the obstacles to pass safely.

### 4.2 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 (Appendix A.3) 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 2 shows the success rate, average survival time, and average traversal time of the safe RL agent trained with Lagrangian method and a baseline MLP agent tested in `static-test`. We define survival time as the time cost of an unsuccessful episode (collision or exceeding a time limit of 80s). 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 cannot succeed. To compare the safe RL method with classical navigation systems which are believed to have better safety, we also add evaluation metrics from a classical navigation stack with the Dynamic Window Approach (DWA) [2] local planner.

**Lagrangian method reduces the gap between training and test environments.** When deployed in the training environments, both the baseline MLP and the safe RL method achieves about $80\%$ success rate. However, in the test environments, the Lagrangian method has a better success rate of $74\%$ compare to $65\%$ by the baseline MLP. 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.

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

### 4.3 Model-based RL (D2 and D3)

To explore how the model-based approaches help with the autonomous navigation tasks, we implement both Dyna-style and MPC methods (Appendix A.4), 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 3.

**Model-based methods do not improve sample efficiency.** As shown in the second and third columns in Table 3, better success rates of $13\%$ and $58\%$ are achieved by the baseline 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 3, both Dyna-style and MPC with probabilistic dynamic models achieve slightly better success rates of $70\%$ compared to $65\%$ in the baseline 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 2). 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.

### 4.4 Domain Randomization (D4)

To explore how model 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 five different sizes: `static-train-5`, `static-train-10`, `static-train-50`, `static-train-100`, and `static-train-250`. The trained policies are tested in the same `static-test`. To investigate the performance gap between training and test, the policies trained with 50, 100, and 250 environments are also tested on `static-train-50`, which is part of their training sets. Fig 4 shows the success rate of policies trained with different number of training environments.

**The generalization to unseen environments monotonously improves with increasing number of training environments.** As shown in Fig. 4, the performances on the unseen test environments monotonously increase from 43% to 74% with the number of training environments increasing from 5 to 250. Moreover, the gaps between training and test environments gradually shrink by adding more training environments provided by that the polices are robust enough to maintain similar performances of about 80% on the training environments.

| Transition samples | 100k | 500k | 2000k |
|---|---|---|---|
| MLP | $\mathbf{13 \pm 7}$ | $\mathbf{58 \pm 2}$ | $65 \pm 4$ |
| Dyna-style deterministic | $8 \pm 2$ | $30 \pm 10$ | $66 \pm 5$ |
| MPC deterministic | $0 \pm 0$ | $21 \pm 10$ | $62 \pm 3$ |
| Dyna-style probabilistic | $0 \pm 0$ | $48 \pm 4$ | $\mathbf{70 \pm 1}$ |
| MPC probabilistic | $0 \pm 0$ | $45 \pm 4$ | $\mathbf{70 \pm 3}$ |

Table 3: (D3) Success rate (%) of policies trained with different model-based methods and different number of transition samples.

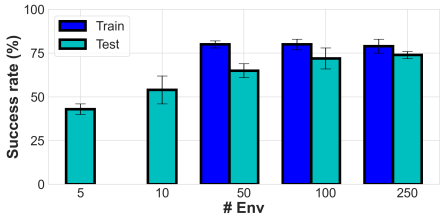


Figure 4: (D4) Success rate (%) of policies trained with different number of training environments.

### 4.5 Physical experiments

To study the consistency of the above observations in simulation and the real world, we deploy one baseline MLP policy, one best policy for each studied desideratum, and one classical navigation system (DWA [2]) in the three real-world environments introduced in Sec. 3.3. Each deployment is repeated three times, and the average traversal time and the number of successful trials are reported in Table. 4. Even though the best memory-based policy, transformer architecture with 4 history length, was only marginally better than the baseline MLP in simulation, in the real world it can navigate very smoothly and fails only once in `real-world-2` and `real-world-3`, while baseline MLP fails most of the trials in all the environments including the benchmark-like environment. Similarly,

| | history | # envs | real-world-1 | real-world-2 | real-world-3 |
|---|---|---|---|---|---|
| MLP | 1 | 50 | 6.9 (1/3) | 10.6 (1/3) | N (0/3) |
| MLP | 1 | 250 | $4.6 \pm 0.8$ (3/3) | $\mathbf{6.6 \pm 0.6}$ **(3/3)** | $\mathbf{22.6 \pm 0.5}$ **(3/3)** |
| Transformer | 4 | 50 | $6.1 \pm 0.4$ (3/3) | $6.1 \pm 0.1$ (2/3) | $20.5 \pm 2$ (2/3) |
| Lagrangian | 1 | 50 | $\mathbf{4.4 \pm 0.6}$ **(3/3)** | $7.1 \pm 0.1$ (2/3) | 26.2 (1/3) |
| MPC | 1 | 50 | $13.2 \pm 0.7$ (3/3) | $24.8 \pm 3.7$ (3/3) | N (0/3) |
| DWA | - | - | $16.2 \pm 0.7$ (3/3) | $35.2 \pm 8.2$ (2/3) | $66.9 \pm 0.6$ (3/3) |

Table 4: Physical experiments. The table shows the traversal time (s) and the number of successful trials of 5 RL-based navigation systems and a classical navigation system (DWA) evaluated in `real-world-1`, `real-world-2`, and `real-world-3`.

MLP policy trained with 250 environments can successfully navigate in all the environments without any failures, while baseline MLP trained with 50 environments fails most of the trials. Safe RL improves the chances of success in all the environments and can navigate more safely by performing backups and small adjustments of robots' poses. Similar to the simulation, MPC navigates very conservatively and succeeds in all the trials in `real-world-1` and `real-world-2`, but has much more difficulty generalizing to large-scale `real-world-3`.

# 5 Limitations

In this section, we discuss the limitations of this benchmark and the current stages of the studied learning techniques in solving the four desiderata. We also point out potential future directions to improve the benchmark and to better solve the desiderata.

**(D1) Reasoning under uncertainty of partially observed sensory inputs.** While our results do not show an obvious benefit by adding memory in static and very random dynamic (`dynamic-box`) environments, much more significant improvements have been shown in the real world and in more challenging dynamic environments (`dynamic-wall`). However, `dynamic-wall` is still very toy-like. Due to the lack of real-world dynamic environments, we cannot justify whether the proposed dynamic environments in the benchmark is instructive to the real-world deployment or not. To further evaluate the ability of RL-based systems with memory to reason under uncertainty, more realistic 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. 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 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 $5\%$ between training and test environments can not be eliminated by further increasing the randomness. While domain randomization is a simple but efficient method, open-ended curriculum learning [27] that leverages procedural content generation is reported to further improve the zero-shot generalization.

**(D5) Navigation is not solved yet!** Although the proposed benchmark cannot represent every real-world navigation scenario, we believe it serves as a simple yet comprehensive testbed for RL-based navigation methods. Because we observed that for every desideratum, no methods can achieve $100\%$ success rate on all *training* environments. Moreover, we have made sure that every environment is indeed individually solvable even using the naive TD3 method with a simple MLP architecture. This alone indicates that there exists an optimization and generalization challenge when we have a large number of training environments as in our proposed 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*, pages 802–807. IEEE, 1993.

[2] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[3] Icra 2022 benchmark autonomous robot navigation (barn) challenge. https://www.cs.utexas.edu/~xiao/BARN_Challenge/BARN_Challenge.html, 2022. Accessed: 2022-06-09.

[4] X. Xiao, B. Liu, G. Warnell, and P. Stone. Motion planning and control for mobile robot navigation using machine learning: a survey. *Autonomous Robots*, pages 1–29, 2022.

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

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

[7] 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*, Proceedings of the IEEE Conference on Decision and Control, pages 4207–4213, Dec. 2011. ISBN 9781612848006. doi:10.1109/CDC.2011.6160783. 2011 50th IEEE Conference on Decision and Control and European Control Conference, CDC-ECC 2011 ; Conference date: 12-12-2011 Through 15-12-2011.

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

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

[10] 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.

[11] 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)*, pages 23–30. IEEE, 2017.

[12] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991. ISSN 0163-5719. doi:10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

[13] 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)*, pages 7559–7566. IEEE, 2018.

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

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

[16] 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*, 31, 2018.

[17] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[18] X. Xiao, B. Liu, G. Warnell, J. Fink, and P. Stone. Appld: Adaptive planner parameter learning from demonstration. *IEEE Robotics and Automation Letters*, 5(3):4541–4547, 2020.

[19] 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.

[20] Z. Wang, X. Xiao, G. Warnell, and P. Stone. Apple: Adaptive planner parameter learning from evaluative feedback. *IEEE Robotics and Automation Letters*, 6(4):7744–7749, 2021.

[21] 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.

[22] X. Xiao, Z. Wang, Z. Xu, B. Liu, G. Warnell, G. Dhamankar, A. Nair, and P. Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022.

[23] 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*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109.

[24] 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*, 47: 253–279, jun 2013.

[25] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4:2007–2014, 2019.

[26] 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)*, pages 82–89, 2019.

[27] 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*, 33:13049–13061, 2020.

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

[29] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] 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.

[31] 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*, 30, 2017.

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

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

[34] 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*, volume 4, pages 3557–3564. IEEE, 1997.

[35] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

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

[37] F. Sadeghi and S. Levine. CAD$^2$RL: Real single-image flight without a single real image. *ArXiv*, abs/1611.04201, 2017.

[38] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[39] 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)*, pages 116–121. IEEE, 2020.

[40] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601, 1983.

[41] OSRF. Ros wiki move_base. `http://wiki.ros.org/move_base`, 2018.

# A Studied Techniques

In Sec. A.1, a brief review of Reinforcement Learning (RL) and Markov decision processes (MDPs) is provided. Then, Sec. A.2 to A.5 describes in detail the studied techniques and how they can potentially achieve the desiderata.

## A.1 RL and MDPs

In RL, an agent optimizes its discounted cumulative return through interactions with an environment, which is formulated as an MDP. Specifically, an MDP is a 5-tuple $(S, A, T, \gamma, R)$, where $S, A$ are the state and action spaces, $T : S \times A \to S$ is the transition kernel that maps the agent's current state and its action to the next state, $\gamma$ is a discount factor and $R : S \times A \to \mathbb{R}$ is the reward function. The overall objective is for the agent to find a policy function $\pi : S \to A$ such that its discounted cumulative return is maximized: $\pi^* = \arg\max_\pi \mathbb{E}_{s_t, a_t \sim \pi}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big]$ [28].

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 $Z : S \to O$ 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 $\tau_t = (o_0, a_0, \ldots, o_t, a_t)$ to an action $a_t$ such that $\max_\pi \mathbb{E}_{\tau_t \sim \pi}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big]$.

## A.2 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 (as in many classical systems, a costmap is being continuously built based on previous perceptions). When using a parameterized model as the policy, recurrent neural networks (RNNs) such as those incorporating Long-Short Term Memory (LSTM) [29] or Gated Recurrent Units (GRUs) [30] are widely adopted for solving POMDPs [14, 15]. More recently, transformers [31], 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 [31]. 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 [30].

## A.3 Safe RL (D2)

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 $C : S \times A \to \mathbb{R}^+$ 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 [32]:

$$\max_\pi \mathbb{E}_{s_t, a_t \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \quad \text{s.t.} \quad \mathbb{E}_{s_t, a_t \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)\right] \leq \epsilon. \tag{2}$$

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 (2). 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

$$\max_{\pi} \mathbb{E}_{s_t,a_t\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\right] + \lambda\left(\mathbb{E}_{s_t,a_t\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t C(s_t,a_t)\right] - \epsilon\right). \tag{3}$$

To solve (3), one can either manually specify a $\lambda$ based on prior knowledge or optimize $\lambda$ simultaneously [33]. In this work, we perform a grid search over $\lambda$ but fix it during learning.

### A.4 Model-based RL (D2, D3)

Given an accurate model, model-based RL often benefits from better sample efficiency compared to model-free methods [34]. 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 [6]. 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 [35] and model-predictive control (MPC) [36]. 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 $\hat{T} : S \times A \to S$ 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:

$$\hat{T} = \arg\min_{T'} \ \mathbb{E}_{(s,a,s',r)\sim B}\left|\left|T'(s,a) - s'\right|\right|_2^2. \tag{4}$$

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. MPC usually enables a more efficient exploration by selecting promising actions, and also potentially improves the asymptotic performance with the help of model predictions.

### A.5 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) [11]. 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. This simple yet strategy has been reported to be efficient in practice when solving many sim-to-real transfer problems [11, 37, 38].

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

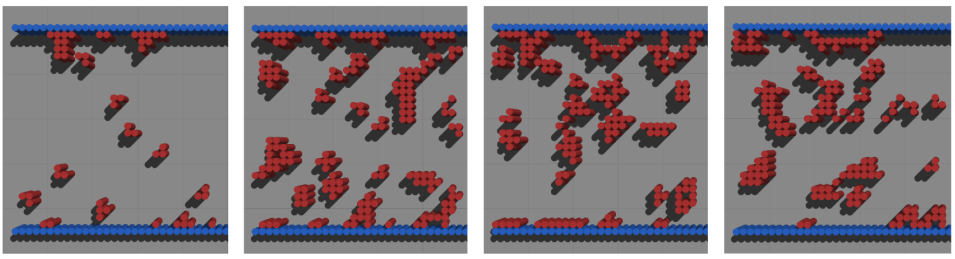

Figure 5: Examples of `static` environments.

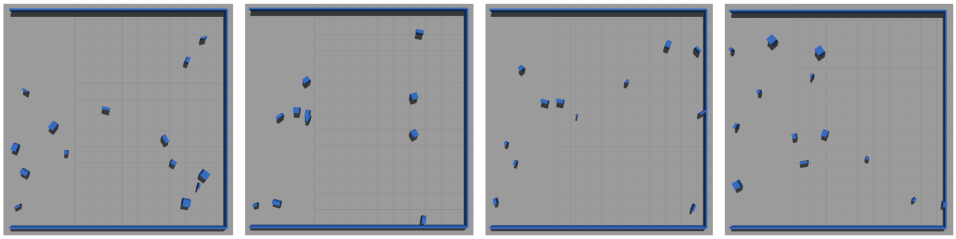

Figure 6: Examples of `dynamic-box` environments.

## B Navigation environments

**Static environments.** We use the 300 static environments from the BARN dataset [39]. The obstacle fields in these environments are represented by a 30×30 black-white grid, which corresponds to an area of 4.5m×4.5m with black and white cells representing obstacle-occupied and free space respectively (see Fig. 5). The grid cells are generated by a method of cellular automation [40], which is originally designed to generate a collection of black cells on a white grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells. In BARN, such generation process begins with randomly filling the grid cells by a ratio of *initial fill percentage*, then performs *smoothing iterations* that either fill an empty cell if the number of its filled neighbors is larger than *fill threshold* or empty a filled cell if its filled neighbors is smaller than *clear threshold*. The grids that are not navigable will be discarded. Due to the cellular automation, the resulting grid resembles real-world obstacles more than the initial randomly filled grid does. To generate BARN dataset, 12 different sets of hyper-parameters are used with *initial fill percentage* chosen from $\{0.15, 0.2, 0.25, 0.3\}$ and *smoothing iterations* ranges from 2 to 4. The *fill threshold* and *clear threshold* are kept at 5 and 1 respectively. Each set of hyper-parameters generates 25 environments which constitute 300 environments in total.

**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. 6). 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 sampled from all the possible directions pointing into the obstacle field. Each obstacle repeats its motion once it moves out of the obstacle field. Each dynamic box environment has 10 to 15 such randomly generated obstacles.

**Dynamic wall environments.** These environments are $4.5m \times 4.5m$, which have two long parallel walls moving in opposite directions with their velocities perpendicular to the start-goal direction (see Fig. 7). 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 to 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.

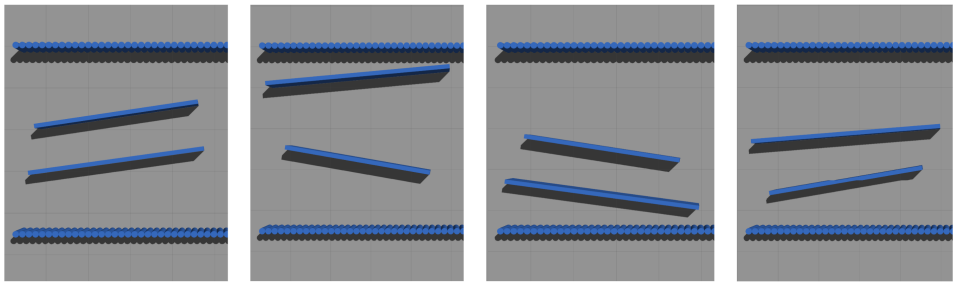

Figure 7: Examples of `dynamic-wall` environments.

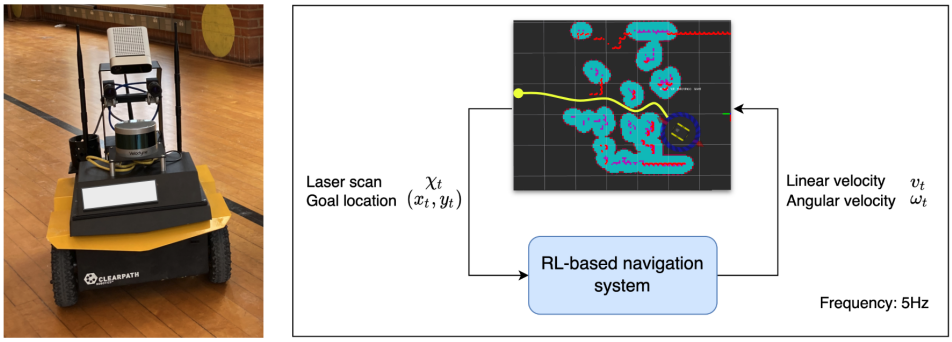


Figure 8: The picture of a Jackal Robot (left) and the diagram of the RL-based navigation system.

## C Jackal Robot and Navigation system

As shown in Fig. 8 on the left, 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 $\chi_t$. We preprocess the LiDAR scans by capping the maximum range to 5m which covers the entire obstacle field. The goal location $(\bar{x}_t, \bar{y}_t)$ is queried directly from the Gazebo simulator. The robot receives velocity commands at a frequency of 5 Hz. A time limit of $80s$ is used which corresponds to a maximum of 400 time steps in an episode. In the real world, the robot is deployed in the same way as simulation except for that the goal location is queried from the on-board ROS `move_base` stack [41] that localizes the robot simultaneously. Fig. 8 on the right shows a diagram of the RL-based navigation system.