Abstract—This paper presents a self-supervised Learning from Learned Hallucination (LfLH) method to learn fast and reactive motion planners for ground and aerial robots to navigate through highly constrained environments. The recent Learning from Hallucination (LfH) paradigm for autonomous navigation executes motion plans by random exploration in completely safe obstacle-free spaces, uses hand-crafted hallucination techniques to add imaginary obstacles onto the robot perception, and then learns motion planners to navigate in realistic highly constrained and dangerous spaces. However, current hand-crafted hallucination techniques need to be tailored for specific robot types (e.g., ground differential drive vehicle), and use approximations heavily dependent on certain assumptions (e.g., short planning horizon). In this work, instead of manually designing hallucination functions, LfLH learns to hallucinate obstacle configurations, where the motion plans from random exploration in open space are optimal, in a self-supervised manner. LfLH is agnostic to robot type and does not make assumptions on planning horizon. Evaluated in both simulated and physical environments with a ground and an aerial robot, LfLH outperforms or performs comparably to previous hallucination approaches, along with sampling- and optimization-based classical methods.

I. INTRODUCTION

Although classical navigation systems can safely and reliably move mobile robots from one point to another within obstacle-occupied environments, recent machine learning techniques have demonstrated improvement over their classical counterparts [1], e.g., by learning local planners [2], [3], learning world representation [4], [5], or learning planner parameters [6]–[8]. However, these learning approaches heavily depend on access to high quality training data.

Learning from Hallucination (LfH) [9], [10] is a recently proposed paradigm to address the difficulty of [1] obtaining high-quality training data for traditional Imitation Learning (IL) from expert demonstrations [11] and 2) Reinforcement Learning (RL) from trial-and-error [12]. During LfH training, the robot executes a variety of random motion plans in a completely safe open space, imagines obstacle configurations for which the motion plans in open space are optimal (called hallucination), and learns an end-to-end local planner as a mapping from the hallucinated obstacle configurations to the optimal motion plans in open space. The inherent safety of navigating in a completely open training environment allows generation of a large amount of training data with no expert supervision or costly failures during trial-and-error learning. Learned local planners react to the deployment environment within constant time (i.e., querying a pre-trained neural network), which is not dependent on how densely packed the surrounding obstacles are.

However, existing LfH methods require manually designed hallucination functions to generate the most constrained [9] or a minimal [10] obstacle set. While the former requires access to a fine-resolution global path and runtime hallucination during deployment, the latter assumes a short planning horizon to assure that one representative minimal unreachable set can approximately represent all possible minimal unreachable sets. Furthermore, these shortcomings prohibit the carefully designed hallucination functions to extend from simple ground differential drive vehicles to different robot types, e.g., aerial robots.

The Learning from Learned Hallucination (LfLH) method proposed in this work removes the necessity of carefully designing hallucination functions specific to certain robot types and assumptions. In a self-supervised manner, LfLH automatically learns distributions of obstacles which make random exploration motion plans in open space optimal, samples obstacle configurations from learned obstacle distributions, and finally learns a local planner that maps hallucinated obstacles to optimal motion plans. LfLH is tested on a ground and an aerial robot, both in simulated benchmark testbeds [13] and physical environments. Superior navigation performance is achieved compared to existing LfH approaches [9], [10] and classical sampling-based [14] and optimization-based [15] planners.

II. RELATED WORK

This section reviews classical motion planning and recent machine learning techniques for mobile robot navigation.

A. Classical Motion Planning

Classical motion planning techniques for mobile robot navigation mostly work in the robot Configuration Space (C-Space) [16] and mainly comprise two categories: sampling-based and optimization-based. Sampling-based motion planners [14] generate sample motion plans and select the
best sample based on a certain metric, such as maximum clearance, shortest path, or a combination thereof [16]. Optimization-based planners [15] start with an initial motion plan, then use optimization techniques to iteratively refine the initial plan to avoid obstacles while observing kinodynamic constraints. One common shortcoming of both categories is when dealing with highly constrained obstacle spaces, classical motion planners require increased computation: sampling-based planners require more samples to find a collision-free motion plan to go through all obstacles, while optimization-based planners require more optimization iterations until a feasible plan can satisfy both collision and kinodynamic constraints.

Compared with classical motion planning algorithms, one advantage of the proposed LfLH approach is that its computation is not dependent on obstacle density during deployment, because LfLH simply queries a pre-trained neural network to produce feasible and fast navigation behaviors.

B. Machine Learning for Navigation

Machine learning approaches have been applied to the classical navigation pipeline in different ways [1], such as constructing world representation [4], [5], fine-tuning planner parameters [6]–[8], improving navigation performance with experience [2], or enabling social [17] and terrain-aware navigation [3]. Most learning methods require either extensive (RL) or high-quality (IL) training data, such as that derived from trial-and-error exploration or from human demonstrations, respectively.

LfH [9], [10] has been recently proposed to alleviate the difficulty of acquiring extensive or high-quality training data: from random exploration in an open space with complete safety, motion planners can be learned by synthetically projecting the most constrained [9] or augmented minimal [10] C-space onto the robot perception. Through carefully designed hallucination functions, these methods have shown fast and agile maneuvers on ground robots compared to classical motion planning and traditional learning approaches. However, the design of specific hallucination functions does not easily extend to other robot types (e.g., aerial robots [9]) and relies on specific assumptions (e.g., a short motion plan/planning horizon to make approximated hallucination valid [10]).

LfLH removes the requirement for a carefully designed hallucination function tailored to a specific robot with strict assumptions, and instead learns hallucinated obstacle distributions which assure the motion plans executed in open space are optimal in a self-supervised manner. Sample obstacle configurations can be drawn from the learned obstacle distributions as training data to learn a motion planner.

III. APPROACH

In this section, we present our Learning from Learned Hallucination (LfLH) approach. We first formulate the problem using the LfH framework, then present the proposed approach to learn (instead of manually design) a hallucination function, from which a motion planner is finally learned, as shown in Fig. 1.

A. Problem Definition

We adopt the same notation used by Xiao et al. to formalize LfH [9] and Hallucinated Learning and Sober Deployment (HLSD) [10]: given a robot’s C-space partitioned by unreachable (obstacle) and reachable (free) configurations, \( C = C_{obst} \cup C_{free} \), the classical motion planning problem is to find a function \( f(\cdot) \) that can be used to produce optimal plans \( p = f(C_{obst} \mid c_c, c_g) \) that result in the robot moving from the robot’s current configuration \( c_c \) to a specified goal configuration \( c_g \) without intersecting (the interior of) \( C_{obst} \).

Here, a plan \( p \in \mathcal{P} \) comprises a sequence of actions \( \{u_i\}_{i=1}^T \), \( u_i \in \mathcal{U} \); \( \mathcal{P} \) and \( \mathcal{U} \) are the robot’s plan and action space, respectively. Considering the inverse problem of finding \( f(\cdot) \), LfH [9] and HLSD [10] have developed hallucination functions, e.g., denoted as \( g(p \mid c_c, c_g) \), to generate the (unique) most constrained and a (not unique) minimal obstacle set, respectively, to make a motion plan \( p \) executed in open space optimal. To instantiate these hallucination functions, hand-crafted rules are designed for specific robot types (e.g. differential drive robots) and do not easily extend to others. LfH [9] further requires a fine-resolution global path and a runtime hallucination function \( h(\cdot) \) to hallucinate the real obstacles during deployment to the most constrained cases during training. HLSD [10] uses one representative minimal unreachable set to approximately represent all of them. This approximation is accurate only for short planning horizons or motion plans, either requiring frequent replanning or limiting navigation speed at runtime.

LfLH aims to learn a parameterized hallucination function \( g_{\psi}(\cdot) \), which outputs probability distributions of obstacles, in a self-supervised manner, without the need to carefully design hallucination functions for certain robot types and thus avoiding the subsequent problems described above. Then we sample many times from this learned distribution, \( C_{obst} \sim g_{\psi}(\cdot) \), to generate many obstacle configurations, in which the free-exploration motion plans in open space are close to optimal.

B. Learning Hallucination

We adopt an encoder-decoder architecture to learn the hallucination function \( g_{\psi}(p \mid c_c, c_g) \) parameterized by \( \psi \). Taking the current configuration \( c_c \), goal configuration \( c_g \), and the corresponding plan \( p \) as input, \( g_{\psi}(\cdot) \) generates probability distributions of obstacles which we assume are ellipsoid, and thus the obstacle distributions are modeled as normal distributions of obstacle locations and sizes in the C-Space. To shape the obstacle distributions such that the given plan \( p \) is optimal. LfLH uses a differentiable decoder \( d(C_{obst} \sim g_{\psi}(p \mid c_c, c_g)) \) to reconstruct the same input motion plans \( p \). The decoder \( d(\cdot) \) samples from the obstacle distributions and optimizes an initial motion plan to reconstruct the output as close to the input motion plan as possible. \( d(\cdot) \) can be an optimization-based classical motion planner, which does not have learnable parameters. To be specific, our encoder-decoder architecture uses gradient descent to find the optimal parameters \( \psi^* \) for \( g_{\psi}(\cdot) \) by minimizing a self-supervised loss using a dataset \( P \) of motion plans collected using random
Fig. 1: The Encoder-Decoder architecture learns hallucination function $g_\psi$ from motion plans in open space (green) to a latent space (red). Sampling from hallucinated obstacle distributions (red), motion planner $f_\theta$ is learned with Behavior Cloning using motion plans collected in open space as ground truth (green).

equation: $\psi^* = \arg\min_{\psi} \mathbb{E}_{p \sim P} \mathbb{E}_{C_{obst} \sim g_\psi(p | c_c, c_g)} [\ell(p, d(C_{obst})) + \ell_r(C_{obst}, p)].$

The choice of the reconstruction loss function $\ell(\cdot, \cdot)$ depends on the representation of a motion plan $p$ (e.g., a sequence of raw motor commands or a bspline trajectory). In addition to the main reconstruction loss $\ell(\cdot, \cdot)$, a regularization loss $\ell_r$ is employed to stabilize training.

$$\ell_r(C_{obst}, p) = \lambda_1 \ell_{prior}(C_{obst}) + \lambda_2 \ell_{coll}(C_{obst}, p),$$  

(2)

where $\ell_{prior}$ prevents $g_\psi$ from overfitting by encouraging a larger probability that $C_{obst}$ is sampled from a prior distribution (i.e., minimizing the discrepancy between the output of $g_\psi$ and the prior), $\ell_{coll}$ is the penalty for collision between obstacles $C_{obst}$ and plan $p$ as well as between each pair of obstacles in $C_{obst}$, and $\lambda_1, \lambda_2$ are corresponding regularization weights. Implementation details of $\ell$, $\ell_{prior}$, and $\ell_{coll}$ can be found in Sec. IV.

After learning $g_\psi(\cdot)$ in a self-supervised manner, we can sample from the learned obstacle distributions $g_\psi(\cdot | c_c, c_g)$ many times to generate many obstacle configurations $C_{obst}$ where the motion plan $p$ is close to optimal. We then form a training set $\mathcal{D}_{train}$ with individual data points $(C_{obst}, p, c_c, c_g)$. The specific instantiation of $C_{obst}$ depends on the perception modality of the mobile robot. For example, for ground robots with 2D LiDAR, we use 2D ray casting from the onboard sensor to the sampled obstacles to determine range readings for each laser beam; for aerial robots with 3D depth cameras, we use 3D rendering to determine each camera pixel’s depth value to the sampled obstacles.

C. Learning from Learned Hallucination

With the training set $\mathcal{D}_{train}$ constructed using the learned hallucination function $g_\psi(\cdot)$, we learn a parameterized motion planner $f_\theta(\cdot)$ from $\mathcal{D}_{train}$ by minimizing a supervised learning loss using gradient descent:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(C_{obst}, p, c_c, c_g) \sim \mathcal{D}_{train}} [\ell(p, f_\theta(C_{obst} | c_c, c_g))].$$

(3)

In general, the loss function $\ell(\cdot, \cdot)$ in Eqsns. 1 and 3 can take the same form. But in practice, the encoder $g_\psi(\cdot)$’s input plan (Eqn. 1) and the motion planner $f_\theta(\cdot)$’s output plan (Eqn. 3) can take different forms to facilitate learning. Details can be found in Sec. IV.

The entire learning pipeline is shown in Alg. 1 including data collection, learning hallucination, learning from learned hallucination, and deployment.

IV. EXPERIMENTS

LfLH is implemented on a ground and an aerial robot to validate our hypothesis that LfLH can automatically learn obstacle configurations where motion plans executed in open space are optimal, and agile motion planners can be learned through the learned hallucination.

A. Ground Robot

We first implement LfLH on a ground robot and compare LfLH’s performance with a classical sampling-based motion planner [14] and state-of-the-art learning approaches from hallucination, including LfH [9] and HLSD [10].

1) Implementation: We use a Clearpath Jackal robot, a four-wheeled, differential-drive, Unmanned Ground Vehicle.

Algorithm 1 Learning from Learned Hallucination

Input: $\pi_{rand}$, $g_\psi(\cdot)$, sampling count, $f_\theta(\cdot)$

1: // Data Collection
2: collect motion plans $(p, c_c, c_g)$ from $\pi_{rand}$ in free space and form motion plan dataset $P$
3: // Learning Hallucination
4: learn $\psi^*$ using Eqn. 1 for $g_\psi(\cdot)$ with $P$
5: $\mathcal{D}_{train} \leftarrow \emptyset$
6: for every $(p, c_c, c_g)$ in $P$
7: for sampling count times do
8: sample $C_{obst} \sim g_\psi(p | c_c, c_g)$
9: $\mathcal{D}_{train} = \mathcal{D}_{train} \cup (C_{obst}, p, c_c, c_g)$
10: end for
11: end for
12: // Learning from Learned Hallucination
13: learn $\theta^*$ using Eqn. 3 for $f_\theta(\cdot)$ with $\mathcal{D}_{train}$
14: // Deployment (each time step)
15: receive $C_{obst}, c_c, c_g$
16: plan $p = \{u_i\}_{i=1}^t = f_\theta(C_{obst} | c_c, c_g)$
17: return $p$
is trained to produce only the first action \((v_{1},\omega_{1})\) in the entire motion plan \(p\) for simplicity (line 13 in Alg. [1]), and it is represented as a fully-connected network with 2 hidden layers of 256 units. During deployment, the UGV reasons in the robot frame, so \(c_{e}\) is the origin and \(c_{g}\) is a point 1.5m away from the robot on the global path (line 15 in Alg. [1]). We use the same Model Predictive Control (MPC) model in HLSD [10] to check for and avoid collisions.

2) Simulated Experiments: We first use the BARN dataset [13] with 300 navigation environments (easy to difficult example environments shown in Fig. 2) randomly generated by Cellular Automata to compare the different motion planners. As a classical sampling-based planner, DWA’s [14] max linear velocity is increased from the default 0.5m/s to 2.0m/s for a fair comparison with other planners. We find that by also quadrupling DWA’s default sampling rate for linear and angular velocity (24 and 80), the UGV’s performance is roughly the same as when using the default parameters (but at quadruple the speed). We train nine different planners with LfH [9], HLSD [10], and LfLH on all three datasets. For all the planners, we run three navigation trials in each of the 300 navigation environments in BARN between a specified start and goal location without a map and record the traversal time (with 50s maximum).

We list the average traversal time with the standard deviation in Tab. [1] The sign indicates the motion planner learned using the corresponding method and dataset fail to navigate to the goal in most of the trials (getting stuck or colliding). For other numbers, a low traversal time means efficient and agile navigation performance. LfLH is the only learning method that works well with all three datasets, and achieves the best navigation performance. For clarity, we only plot the test results in each environment (averaged over three trials) using each variant with its fastest working dataset, i.e., LfH 0.4, HLSD 1.0, and LfLH 2.0, along with DWA 2.0 in Fig. 3 (scattered dots). We also fit a line to show the trend of each method. In Fig. 3 we order the navigation environments from left to right with increasing average traversal time achieved by DWA 2.0 (red). Note that the y axis is set in log scale to better visualize the differences in the small traversal time range. LfLH 2.0 (green) achieves similar results in easy environments (left) and significantly outperforms DWA 2.0 in difficult ones (right). Although LfH 0.4 (orange) and HLSD 1.0 (yellow) have a disadvantage due to slow max speed in easy environments, they exhibit good agility in more constrained difficult ones, especially HLSD 1.0. LfLH 2.0 is the only planner that achieves the best performance in both easy and difficult environments, shown by the flat green line. Note that the physical limit of the UGV maximal velocity is 2.0m/s and the highly constrained BARN environments require slow speeds for agile maneuvering in most places.

3) Physical Experiments: We also deploy the four planners in Fig. 3 in a physical test course, for five trials each (Fig. [4]). The results are shown in Tab. [1] The sampling-based DWA planner [14] fails to sample feasible motions in many constrained spaces, and has to execute many recovery
behavior before re-sampling. Therefore DWA takes a long average time with large variance to finish the traversal. LfH 0.4 requires a fine-resolution global path [9] and drives smoothly but slowly everywhere alone the course. Being too conservative in wide open spaces causes LfH 0.4 to achieve similar results as DWA 2.0, which gets stuck in many places but makes up the time by accelerating in open spaces. HLSD 1.0 also navigates smoothly, but much faster than LfH 0.4, and outperforms DWA 2.0. Our LfLH 2.0 is the only planner that can learn from the fast 2.0m/s dataset, achieving the best performance among all the variants.

B. Aerial Robot

We also evaluate LfLH on an Unmanned Aerial Vehicle (UAV, a quadrotor) where hallucination is more challenging due to higher dimensionality and more agility. We compare LfLH with a state-of-the-art optimization-based UAV trajectory planning algorithm, Ego-Planner [15].

1) Implementation: We apply LfLH to a simulated UAV in Ego-Planner’s simulator and a physical PX4 Vision UAV platform shown in Fig. 5. Both simulated and physical UAVs use depth input, acquired by rendering in simulation and a Structure Core camera, respectively, to instantiate obstacle configuration \( C_{\text{obst}} \).

Due to unavailability of a motion capture system, we collect a dataset of 20-min flight in simulation (line 2 in Alg. 1). We represent the random exploration policy \( \pi_{\text{rand}} \) to collect \( p \) in an open space using Ego-Planner with 2.0m/s max \( v \) in a highly constrained environment. We only record motion plans \( p \) but do not record any perception input (as if the UAV were flying in an open space)\(^2\) which is later synthesized by LfLH. Note that previous learning approaches from hallucination cannot solve this 3D aerial navigation task: LfH [9] is not applicable because no global path is available; The approximated minimal unreachable set in HLSD [10] cannot effectively represent 3D obstacles given the long flight plan.

For our LfLH implementation in 3D, the odometry point contains the same information \((x_i, \phi_i, v_i, \omega_i)\) as the UGV (but in 3D), and each plan \( p \) consists of \( M = 500 \) (2.5s) odometry points. The encoder \( g_{\psi} \), decoder \( d \), reconstruction loss \( \ell \), and regularization \( \ell_{\mathrm{reg}} \) are in the same format as for the UGV, except that \( g_{\psi}(\cdot) \) outputs distributions of 15 obstacles, the obstacle size prior has a different mean of 0.6m and variance of 0.2m\(^2\) in the prior regularization loss \( \ell_{\text{prior}} \), the clearance increases to \( c = 0.6m \) in the collision regularization loss \( \ell_{\text{collision}} \), and \( \lambda_1 = 0.5, \lambda_2 = 5.0 \) for regularization weights.

After training \( g_{\psi}(\cdot) \) (line 4 in Alg. 1), the training set \( \mathcal{D}_{\text{train}} \) is constructed with the collected plans \( p \) and their corresponding observed \( C_{\text{obst}} \), i.e., depth images rendered using ray casting given the 15 sampled hallucinated obstacles and five additional obstacles sampled in the same way as the UGV in Sec. IV-A.1. The motion planer \( f_{\theta} \) is modeled as four convolutional layers and three fully-connected layers used to extract features from the depth image, goal

\(^2\)We leave a true random exploration policy in an open space as future work, such as random teleoperation or a random policy as in the UGV case.
where \( \theta \) is trained to produce positions and linear velocities \( \{ \hat{\xi}_i, \hat{v}_i \}_{i=1}^M \) in the entire motion plan \( p \). During deployment, a low level controller [19] tracks the motion plan \( p \) produced by \( \theta \), and therefore \( p \) is invariant to the UAV dynamics. A similar MPC method in HLD [10] used for ground navigation is also applied to check for collisions by the UAV.

2) Simulated Environments: We first evaluate LfLH and Ego-Planner in simulation with a randomly generated forest shown in Fig. 5 left. Each method is tested for 10 trials, and in each trial the UAV keeps navigating to randomly generated goals until the UAV collides with obstacles. Meanwhile, we record the total traversal time and distance, as well as the individual traversal time \( t_i \) and distance \( p_i \) to each goal \( i \), from which we measure Success weighted by Path Length (SPL) [20]:

\[
SPL = \frac{1}{K} \sum_{i=1}^{K} S_i \frac{t_i}{p_i},
\]

where \( S_i \) is the binary indicator of success for reaching the \( i \)th goal, and \( t_i \) is the Euclidean distance from the \( i \)th start to the \( i \)th goal, which is always smaller than \( p_i \).

We list the average and standard deviation (if applicable) of these metrics in Tab. III. Compared with Ego-Planner, LfLH survives (keeps navigating in a collision-free manner) of these metrics in Tab. III. Compared with Ego-Planner, LfLH also self-supervised learns training in open space. In addition to self-supervised learning in the entire motion plan, a low level controller [19] tracks the motion plan \( p \) produced by \( \theta \), and therefore \( p \) is invariant to the UAV dynamics. A similar MPC method in HLD [10] used for ground navigation is also applied to check for collisions by the UAV.

3) Physical Demonstration: We also deploy LfLH in a physical obstacle course. Note even though real obstacles are completely different from the ellipsoidal hallucinated obstacles, LfLH still avoids obstacles and navigates to the goal in a stable and efficient manner.

V. CONCLUSIONS

LfLH is a self-supervised machine learning technique for mobile robot navigation with safety guaranteed during training in open space. In addition to self-supervised learning of a motion planner, LfLH also self-supervised learns to generate hallucinated obstacle configurations, from which the motion planner is learned, instead of requiring hand-crafted hallucination functions. The hallucinated obstacle configurations are more general and representative than the manually designed ones [9], [10], and can be applied to agile ground and aerial robots navigating at faster speeds. Although in our experiments the ground robot learns from a real random exploration policy, one interesting future research direction is to investigate truly random exploration for aerial vehicles, instead of using trajectories collected from an existing motion planner. In addition to a physical demonstration, quantitative experiments with the physical UAV need to be performed, and thus we plan to build a drone arena to benchmark physical aerial navigation performance.

REFERENCES


