IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Pushing the Limits of Reactive Navigation: Learning to Escape Local Minima

Isar Meijer[1], Michael Pantic[1], Helen Oleynikova[1], and Roland Siegwart[1]

***Abstract*—Can a robot navigate a cluttered environment without an explicit map? Reactive methods that use only the robot's current sensor data and local information are fast and flexible, but prone to getting stuck in local minima. Is there a middle-ground between reactive methods and map-based path planners? In this paper, we investigate feed forward and recurrent networks to augment a purely reactive sensor-based navigation algorithm, which should give the robot "geometric intuition" about how to escape local minima. We train on a large number of extremely cluttered simulated worlds, auto-generated from primitive shapes, and show that our system zero-shot transfers to worlds based on real data, 3D man-made environments, and can handle up to 30% sensor noise without degradation of performance. We also offer a discussion of what role network memory plays in our final system, and what insights can be drawn about the nature of reactive vs. map-based navigation. The implementation of the planners and all experiments is made available open-source https://github.com/ethz-asl/rmp_dl.**

***Index Terms*—Reactive and Sensor-Based Planning, Collision Avoidance, Deep Learning Methods**

I. Introduction

IN complex and cluttered environments, collision-free navigation is one of the most fundamental skills a robot must possess. Most collision avoidance methods fall into one of two categories: map-based or reactive. *Map-based* methods rely on a processed, explicit world representation which can be checked for collisions, while *reactive* approaches often only use the robot's local information and current sensor data to decide the robot's next action.

Local reactive methods can be extremely computationally efficient and provide safety without relying on additional processes, such as mapping frameworks or state estimators, or strict assumptions about the environment, such as static world assumptions. However, without memory or a longer-term perspective they are prone to getting stuck in local minima – geometric dead-ends such as large walls, horse-shoe bends, or long corridors. Ideally, even a purely reactive method would have a certain sense of geometric intuition or an implicit understanding of past observations that allows it to make informed decisions even in absence of a consistent map. How can we build such a system? Where are the limits of purely reactive navigation, in terms of geometric and temporal consistency? When is it better to rely on a map?

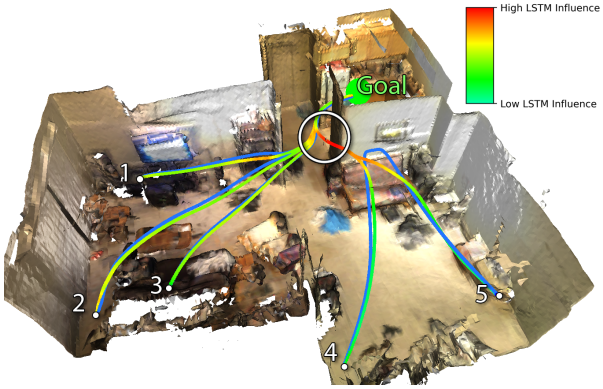


Fig. 1: A comparison of baseline (blue) and learned LSTM (rainbow) trajectories from 5 starting points to the goal location shown by a green sphere. The reactive baseline cannot navigate to the goal when it is directly behind a wall, while our learned method uses its "geometric intuition" (shown by the high influence of the LSTM, highlighted with the white circle) to turn left to escape the local minimum. The environment is a 3D reconstruction of an apartment from the BundleFusion dataset [1], and the learned system is transferred zero-shot to this environment based on real-world data after training on synthetically generated data containing only obstacle primitives.

In this paper, we aim to answer these questions using a succession of novel methods for informed reactive navigation by combining a purely classical, reactive approach with different neural networks. We use simple Feed-Forward Neural Networks (FFNs) and Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) cells trained in a self-supervised environment to provide geometric "intuition" that is then combined with a classical safety layer. These networks essentially bias the classical reactive method to avoid and escape local minima. Our method is made sensor-agnostic by expressing the sensor data as a set of *rays* originating at the robot's current position.

By studying the performance implications of the different variants of our sensor-agnostic method, we provide novel insights into the limits of reactive navigation and to what extent these limitations can be overcome using different network architectures with varying degrees of temporal consistency. Most importantly, we provide insight into the *nature* of the different methods – we investigate how different parts of the network play a larger role on the resulting trajectory and what implications can be drawn from it. Our method is loosely inspired by navigation of humans and animals, which are able to anticipate navigation decisions without a complete or metric map. By training our approaches in auto-generated, unstructured environments with very high obstacle density (Fig. 6), many different shapes of local minima are

Manuscript received: December 20, 2024; Revised March 5, 2025; Accepted April 3, 2025.

This paper was recommended for publication by Editor Markus Vincze upon evaluation of the Associate Editor and Reviewers' comments.

[1]Isar Meijer, Michael Pantic, Helen Oleynikova and Roland Siegwart are with the Autonomous Systems Lab, ETH Zürich, Switzerland. isarmeijer@gmail.com michael.pantic@mavt.ethz.ch helen.oleynikova@mavt.ethz.ch rsiegwart@ethz.ch

Digital Object Identifier (DOI): see top of this page.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

encountered and learned. The resulting approaches are then evaluated on structured, human-made environments (Fig. 1).

The central goal of this work is to explore and understand how far the limits of reactive navigation can be pushed without using an explicit map. To this end, we contribute:

- multiple data-driven methods that provide "geometric intuition" to a classical reactive baseline navigation system,
- a self-supervised training setup using the geodesic distancefield as a supervision signal,
- show how the trained networks generalize to 3D environments based on real data with different levels of sensor noise,
- and a thorough discussion how temporal memory within the network affects the overall navigation system's ability to escape local minima.

The purely reactive baseline method is reviewed in Section III, then combined with an additional feed-forward network in Section IV (FFNav), then extended to a recurrent network in Section V (RecNav). For each method we discuss its performance, short-comings, and implications, based on qualitative examples. We then compare all variants with established methods that rely on a full map (i.e. are non-reactive) in Section VI and discuss the results and fundamental findings in detail in Section VII.

II. Related Work

Many robotic navigation approaches rely on accurate maps of the environment, treating mapping and navigation as separate components with their own respective evaluation metrics. The first works in this direction were *potential field methods* [2], which treated every obstacle as a repulsive force acting on the robot. Such a potential field is computed over the entire map, allowing the addition of other constraints or optimization objectives on the trajectory such explicit kinematic or dynamic constraints [3].

However, even with full map knowledge, methods based on collision potentials get stuck in local minima when the scenes become too cluttered [4], as they often result in non-convex problems. Other map-based methods such as RRT* [5] trade-off computation time for asymptotic optimality and probabilistic completeness. Due to their runtime, they are often used for global planning in combination with a faster local planner, such as any of the potential field methods.

Reactive, map-free navigation often uses the concept of obstacle-induced repulsive forces [6]. Newer approaches such as Riemannian Motion Policies (RMPs) [7] allow the combination of multiple local policies, encoding different robot objectives based on local information. Pantic *et al*. [8] formulate a pure avoidance policy with RMPs which allows fast, safe navigation without a map. However, this approach still occasionally gets trapped in local minima. We use the open source implementation of [8] as a baseline method. Mattamala *et al*. [9] use a geodesic field in a reactive navigation setup to avoid local minima, however, computing the geodesic distance field *a priori* requires a consistent map of the environment.

Other approaches solve reactive navigation by learning an *end-to-end* policy from sensor data to robot action, without the need for a map. Some end-to-end methods end up learning the map *implicitly* as part of the training process, by training in the same environment as the robot navigates in. For example, using only visual input to navigate through 2.5D mazes [10] or flying in 3D through cluttered environments [11].

Other works focus on learning policies that work in a variety of environments. Ross *et al*. [12] learn a mapping from image features to velocity commands from human pilot demonstrations using DAgger [13] for a drone flying rapidly through a forest; however, requiring human pilot demonstrations makes training for new environments difficult. Loquercio *et al*. [14] instead train dynamic drone flight on a variety of simulated environments, and show impressive results of zero-shot transfer to the real world, using stereo disparity images as input and a short-horizon dynamic trajectory as output. They focus on flying *quickly* through semi-cluttered environments, while we specifically want to focus on escaping local minima with a mix of classical and learned approaches.

Reactive navigation methods have also gained traction in the computer vision community [15, 16]. These works generally focus on indoor, 2.5D environments with a strongly reduced action space, where semantic reasoning plays a major role. Impressive results in works such as [15, 17–19], require large vision models and implicitly use a large amount of semantic information to navigate indoor 2.5D scenes. Similar to our work, Partsey *et al*. [17] also investigate the necessity of a map for navigation tasks, however, it is unclear how this extrapolates to more complex 3D environments, especially without the semantic information that an indoor space offers.

Other end-to-end networks have used reinforcement learning with a similar ray-based sensor representation, but also limited to 2/2.5D. Tai *et al*. [20] learn a general navigation policy in 2D from LiDAR scans which map to a velocity output, and generalize across sparse maps. Zhang *et al*. [21] use explicit external memory structures to learn a representation of the visited environment, however, limited to discretized 2D worlds.

By mixing a classical reactive avoidance algorithm with a supervised learning component, we are able to handle full 3D trajectories in vastly more cluttered environments than presented in other works, and without relying on scene semantics. The goal of our work is to see how we can build on the ideas of [8] to overcome the downsides of map-free, purely reactive navigation without requiring an explicit map, while leveraging the best advantages of both classical and learned methods. We rely on the RMP-based repulsive field approach to provide collision-free, safe trajectories, while our new learned component focuses on developing a geometric intuition about the environment. The following sections will describe the classical reactive safety layer, followed by feed-forward and then recurrent network architectures.

III. Classical Reactive Navigation

We use the open-source system presented in [8] as the base method for classical reactive navigation. Obstacle avoidance is formulated as a combination of obstacle repulsive forces, represented as Riemannian Motion Policies (RMPs) [7], with

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

each "obstacle" coming from a sensor ray return. As it serves as our base method, and for the reader's convenience, we reproduce the most important concepts of Riemannian Motion Policies and how they are used to perform obstacle avoidance in this section. For more details we refer the reader to the respective original work [7, 8].

A. Riemannian Motion Policies

An RMP is a policy $\mathcal{P}$ that consists of an acceleration $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \in \mathbb{R}^3$ coupled with a metric $\boldsymbol{A}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \in \mathbb{R}^{3\times3}$, where $\boldsymbol{x}, \dot{\boldsymbol{x}} \in \mathbb{R}^3$ denote the robot's position and velocity. A single policy $\mathcal{P} = (\boldsymbol{f}, \boldsymbol{A})$ describes an acceleration field acting on the system, combined with a metric that captures the directional importance of that acceleration. A set of policies $\{\mathcal{P}_i\}$ can be summed into a single policy by $\sum_i \mathcal{P}_i = \left((\sum_i \boldsymbol{A}_i)^+ \sum_i \boldsymbol{A}_i \boldsymbol{f}_i, \sum_i \boldsymbol{A}_i \right)$ where $^+$ denotes the pseudoinverse. The result is a policy itself, that contains an implicitly metric-optimal joint behavior of all policies. A policy can be a function of the robot's state, the goal location, and any other type of observation that is available. In the next section, we introduce local observations in the form of ray casts.

B. Ray-Casting

The obstacle avoiance policies use ray casts from the robot's current pose as a general abstraction of depth sensor data, as shown in Fig. 3. For simplicity, we treat our robot as a point, but robot shape can be represented by modifying the distance offset of the rays; to model a sphere robot, a constant distance would be subtracted from all ray values for example.

Similar to [8], we sample N equally spaced, quasi-random depth rays by using the Halton sequence [22]. We define the ray-cast function $\boldsymbol{\mathcal{R}}^k(N, L) : \mathbb{N} \times \mathbb{R}_+ \to \mathbb{R}_+^N$ that samples N equally spaced rays according to the Halton sequence from the robot's position $\boldsymbol{x}^k$ at time step k. Due to the Halton sequence' deterministic nature, for function calls with identical N, the direction vector for each element i in the output remains the same across timesteps. By using a GPU-based mapping environment [23], this function can be evaluated in parallel [8] for all rays.

C. Collision Avoidance

The obstacle avoidance policy defined in [7] serves as a building block for collision avoidance. The obstacle policy repels the robot from a point obstacle and consists of a repulsive term $\boldsymbol{f}_{rep}(\boldsymbol{x})$ and a damping term $\boldsymbol{f}_{damp}(\boldsymbol{x}, \dot{\boldsymbol{x}})$:

$$\boldsymbol{f}_{obs}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{f}_{rep}(\boldsymbol{x}) + \boldsymbol{f}_{damp}(\boldsymbol{x}, \dot{\boldsymbol{x}}). \tag{1}$$

The repulsive term applies an acceleration away from obstacles based on distance, while the damping term is based on the velocity towards the obstacle. For more details and the definition of the corresponding metric ($\boldsymbol{A}_{obs}$), the reader is referred to the original work [7]. The combination of the repulsive and damping term using the metric matrix results in a smooth, safe, and velocity-dependent avoidance behavior, where only obstacles that pose a risk of collision in the robot's current direction of motion have an effect. A simple unidirectional repulsor as used in potential field methods is isotropic and takes only the distance of obstacles into account. This method allows us to combine thousands of such policies to perform avoidance in dense and cluttered 3D maps. To do so, we follow the approach from [8], where an obstacle policy is created for every ray that hits an obstacle. A GPU implementation is leveraged such that not only ray-casting, but also policy creation and summation happens in parallel.

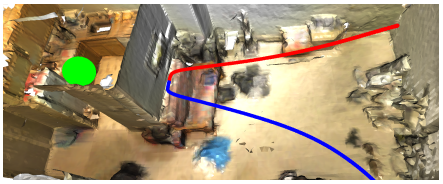


Fig. 2: Example of 2 separate classical reactive planning runs, depicted in red and blue, getting stuck in the same local minimum. The goal location is behind the wall, depicted in green. BundleFusion dataset [1].

D. Goal Seeking

Similar to [8], we use the goal policy as defined in [7] to implement goal seeking behavior. The goal policy pulls the robot towards a goal location $\boldsymbol{x}_g$ and is defined as

$$\begin{aligned} \boldsymbol{f}_g(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \alpha_g \boldsymbol{s}(\boldsymbol{x}_g - \boldsymbol{x}) - \beta_g \dot{\boldsymbol{x}}, \\ \boldsymbol{A}_g(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \mathbb{I}^{3\times3}, \end{aligned} \tag{2}$$

where α_g and β_g are scalar tuning parameters, and $\boldsymbol{s}$ is a soft-normalization function [7]. The robot is then commanded with the sum of all activated policies. Trajectories are generated by numerical integration of the resulting acceleration, starting from an initial position at rest.

E. Limitations

This purely reactive method performs surprisingly well even in very cluttered and complex 3D maps (see also [8]). Its navigation capabilities emerge from the sum of many simple obstacle avoidance policies and the goal seeking behaviour, but it can get trapped in local minima whenever the two parts cancel each other out. Both, natural and man-made environments occasionally contain such local minima. Fig. 2 visualizes a typical example, where the wall obstacle exactly cancels out the goal seeking policy, leading to the approach getting stuck. However, humans are able to navigate these environments without a full map and do not get stuck indefinitely. Clearly, there are heuristics and intuitions about taking decisions that avoid or escape local minima. This naturally raises the question of how such intuition can be provided to a robotic system, preferably through a self-emergent process. In the next section, we introduce such an add-on to the purely reactive method by using a feedforward neural network trained in a self-supervised fashion. In the section after that, we extend this implementation with a recurrent neural network.

IV. Feed Forward Neural Reactive Navigation (FFNav)

Our goal is to train a geometry-aware goal policy, as a function of the goal direction and immediate sensor rays, that

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

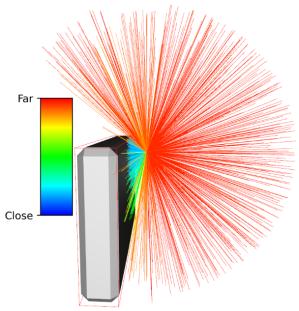


Fig. 3: Example of 1024 ray-casts obtained via Halton sampling. These rays are a map-agnostic method to gain local geometric information.

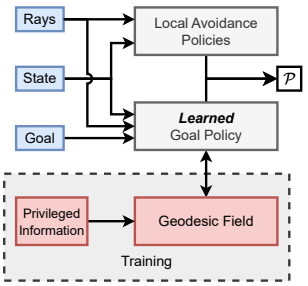


Fig. 4: System overview of the learned reactive planner. We combine the local avoidance policies as a safety layer with a higher level learned system, supervised by the geodesic field during training.

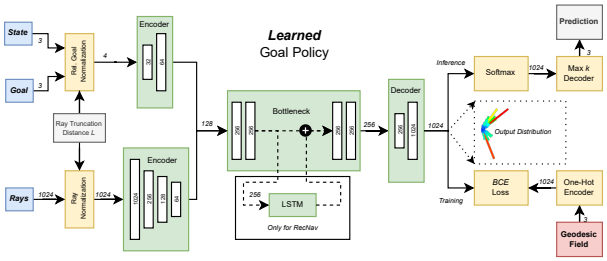
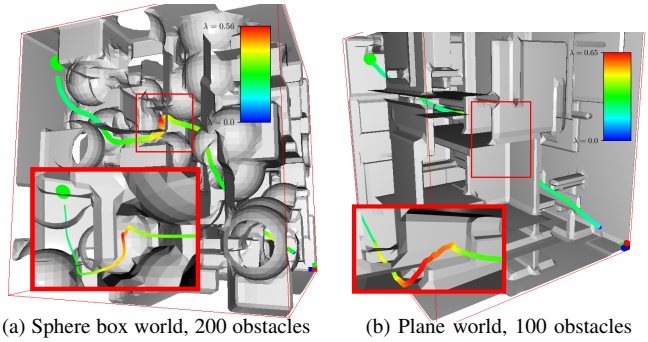


Fig. 5: Network architecture. Learned blocks are highlighted in green, blue blocks are inputs and yellow blocks are non-learned operations. Learned blocks consist of a linear layer followed by layer normalization and a Leaky ReLU, except for the final decoder layer. The LSTM is only for RecNav.



(a) Sphere box world, 200 obstacles (b) Plane world, 100 obstacles

Fig. 6: Examples of different runs. The start and goal are depicted with blue and green spheres respectively. The color gradient encodes the relative influence level of the LSTM on the latent state. Both figures show emerging exploration behavior in RecNav when navigating close to local minima.

is able to bias the system towards a promising intermediate goal direction in order to avoid getting stuck in local minima. To do so, we use the *geodesic distance field* as a training signal. The geodesic distance field is a global function that captures the shortest distance to the goal from anywhere in the world around obstacles. It can be computed using the Fast Marching Method (FMM) [24], a grid-based version of which is implemented in the *scikit-fmm* library. The geodesic distance field requires perfect world information and is expensive to compute, therefore we only use it as privileged information for self-supervision during training. Doing so, the network learns to infer near-optimal decisions that mimic the geodesic distance field directly from raw sensor data, effectively obtaining good heuristics to deal with local geometry. A high-level overview of the proposed system is shown in Fig. 4.

More formally, let the (symmetric) scalar function $\mathcal{G}(\boldsymbol{x}, \boldsymbol{x}_g)$ denote the geodesic distance field between locations $\boldsymbol{x}$ and $\boldsymbol{x}_g$. The shortest direction to the goal $\boldsymbol{x}_g$ from location $\boldsymbol{x}$ can be calculated by taking the (negative) gradient of $\mathcal{G}$ w.r.t. $\boldsymbol{x}$:

$$-\nabla_{\boldsymbol{x}} \mathcal{G}(\boldsymbol{x}, \boldsymbol{x}_g). \tag{3}$$

With this improved goal direction, we can create a goal policy similar to Eq. (2):

$$\boldsymbol{f}_g(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \alpha_g \boldsymbol{s}(-\|\boldsymbol{x}_g - \boldsymbol{x}\| \nabla_{\boldsymbol{x}} \mathcal{G}(\boldsymbol{x}, \boldsymbol{x}_g)) - \beta_g \dot{\boldsymbol{x}}, \tag{4}$$

where we replaced the straight-line direction to the goal with the negative geodesic distance field gradient. The aim is to learn a parameterized function that mimics Eq. (3), using only the relative location of the goal and the sensor rays as input.

A. Architecture and Training

We regress the geodesic distance field gradient using a multi-layer perceptron trained in a self-supervised fashion. Fig. 5 illustrates the network architecture for both FFNav and RecNav (next section). In this section, we discuss architectural details, data generation and training methods for FFNav.

1) Encoders: The input part of the network consists of a ray encoder and a goal direction encoder. All input signals are rescaled to relative quantities with respect to the maximum ray length L. The state and goal information at the current timestep is used to calculate the relative goal direction $\boldsymbol{d}_g^k = \boldsymbol{x}_g - \boldsymbol{x}^k$, which is fed to the state encoder as a unit direction $\hat{\boldsymbol{d}}_g^k = \boldsymbol{d}_g^k / \|\boldsymbol{d}_g^k\|$ vector, concatenated with a scalar distance measure $\mathcal{D}: \mathbb{R} \mapsto [0, 1]$, defined as follows:

$$\mathcal{D}\left(d = \|\boldsymbol{d}_g^k\|\right) = \begin{cases} \frac{d}{2L}, & d \leq L \\ \sigma\left(2\frac{d-L}{L}\right), & d > L \end{cases}, \tag{5}$$

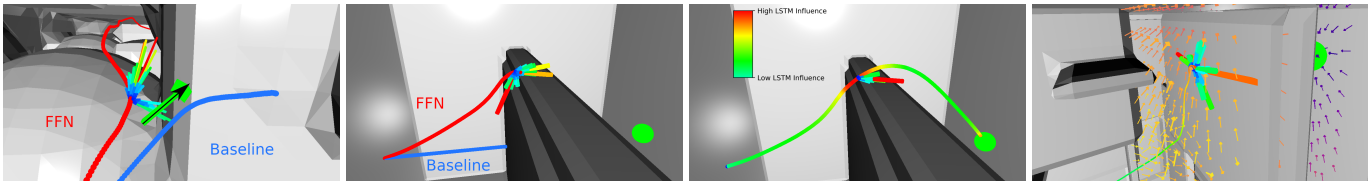
where σ is the sigmoid function. This function maintains linearity below the ray truncation distance and saturates large input values to 1. Doing so, the distance to the goal becomes a relative quantity with respect to the ray lengths, which also implicitly encodes if the goal location is in front or behind a perceived obstacle.

2) Information Bottleneck: The concatenated state and ray latent representations pass through four fully connected layers.

3) Decoder: Instead of directly regressing the geodesic gradient, we decode the latent space into a weighted output distribution of directional rays, similar to the input rays. We use a Binary Cross Entropy Loss (BCE) loss between the decoder output and a one-hot encoded vector that maps the ground-truth geodesic direction vector to the closest (cosine similarity) corresponding Halton unit direction. An additional benefit of using such an output format is that it enables easy introspection by visualizing the output distribution. To obtain an RMP compatible acceleration vector from the output distribution, we pass it through a softmax layer and calculate a weighted sum of the top 50 output direction vectors.

4) Training: We exclusively train on auto-generated worlds obtained through boolean combinations of primitive shapes. Two classes of $10 \times 10 \times 10\text{m}^3$ worlds are created and

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.



(a) Baseline vs. FFNav. (b) Baseline vs. FFNav. (c) RecNav. (d) RecNav.

Fig. 7: Examples of different planning runs, where the goal location is depicted by the green sphere (in (a) the black-with-green-hue arrow is pointing in the direction of the goal). RecNav is depicted with a color gradient depicting the influence of the LSTM. We also show the output probability distribution of the learned systems at certain locations in the trajectory. In (a) FFNav navigates past a local minimum in which the baseline gets stuck. In (b) FFNav gets stuck on the corner of a wall as only having access to the current observation provides insufficient information on where to go, highlighted by the bimodal output distribution. In (c) RecNav is able to navigate around the same wall, and we show that the LSTM has the highest influence exactly where FFNav got stuck, causing the collapse of the second output mode observed in FFNav. In (d) RecNav gets stuck in a large local minimum, and the model has a large number of modes in the output distribution. The arrows depict the gradient of the geodesic field, and are colored by the geodesic distance to the goal.

randomized during training, validation, and testing: *sphere box* worlds (Fig. 6a) and *plane* worlds (Fig. 6b). We use a varying amount of obstacles for each; up to 200 and 100 obstacles for the *sphere box* and *plane* worlds respectively. These classes of worlds are simple to generate in thousands of variations, without the need for human labeling or manual data collection. We train the network with randomized densely sampled data from 6400 random worlds, with 1024 random locations per world.

B. Qualitative Evaluation and Limitations

A qualitative example of FFNav is depicted in Fig. 7a, FFNav is able to navigate past a local minimum, while the classical reactive baseline gets stuck. The network is able to provide geometric intelligence to avoid local minima, with the baseline method still serving as a safety layer. While the system performs much better than the baseline, there are still obvious cases where it gets stuck, as seen for example in Fig. 7b: local observations may provide insufficient information about the free space around the perceived obstacle, confirmed by the bimodal output distribution of the model. These cases can be attributed to the purely reactive nature of even the learned method. In some situations, previously observed data is needed in order to navigate. Those cases give rise to the need of short-term temporal consistency – which we will explore in the next section by using recurrent neural networks.

V. Recurrent Neural Reactive Navigation (RecNav)

We introduce a short-term memory by inserting a single LSTM layer into the bottleneck, as depicted in Fig. 5. The RecNav system thus now requires sequences to train the recurrent elements. Therefore, we train the network by using Data Aggregation (DAgger) [13]; model rollouts from a random start and goal location are aggregated with previously collected runs during training. As DAgger has proven to be less sample efficient in training of this network, we use a 2-stage training setup to speed up the training process. The FFNav architecture is trained as described in Section IV, then all weights are frozen, and the LSTM is inserted to form the RecNav system and trained using DAgger.

A. Qualitative Evaluation and Limitations

Returning to the previouse example in Fig. 7b where FFNav got stuck, Fig. 7c shows that RecNav is able to succesfully navigate over the wall. We color the trajectory based on the relative influence of the LSTM on the latent representation inside the network. We observe that the LSTM has more influence in the exact locations where the FFNav variant got stuck, and that only a single mode remains in the output distribution. Fig. 6 shows similar results on two of the cluttered, autogenerated synthetic worlds. We also evaluate RecNav on real human-made environments scanned with an RGB-D camera, shown in Fig. 1 and Fig. 9. The learned system is able to generalize zero-shot to these structured environments. It only uses the ray-casting as a generalized lidar-like sensor input, and no other privileged information such as the map itself. We see that the LSTM increases its influence at key navigation locations in both the synthetic and real world-based examples.

However, there are limitations. Local minima that are considerably larger or deeper than observed on training data can be a problem, examples are visualized in Fig. 7d and Fig. 9b. We attribute this to *(a)* the training environments and *(b)* the multi-modality of the output. Choosing a more complex strategy to select the most promising "mode" of the multi-modal output instead of the weighted sum could improve the influence of the learned policy. Fig. 7d depicts an example of a case where an improved strategy would be beneficial.

VI. Quantitative Evaluation

In this section, we will quantitatively compare our methods to existing solutions, both on synthetic and real-world scenes.

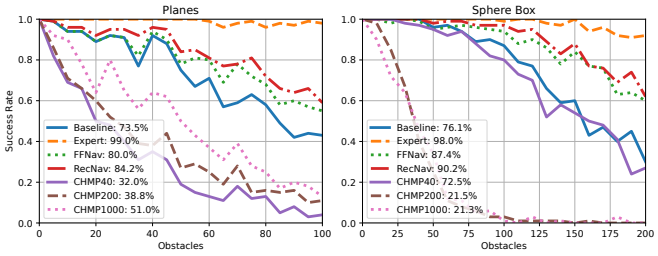


Fig. 8: Success rates as a function of obstacle density. The percentages in the legend denote the overall success rates ($N = 2100$, 100 runs per datapoint). The reactive methods outperform all configurations of the map-based CHOMP, even though the reactive methods do not have access to the map. Both learned systems outperform the baseline, and RecNav also slightly outperforms FFNav.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

A. Synthetic Scenes

To evaluate how the different reactive methods, that use only current sensor data, compare to methods that have access to a complete map, we compare against an "expert", which simply follows the geodesic field gradient, and CHOMP, a well-known local planner [3]. We compare navigation success rates as a function of obstacle density on a separate test set, depicted in Fig. 8. In all following evaluations, "baseline" refers to the system as described in Section III, "FFNav" to the one in Section IV, and "RecNav" to Section V. "CHMP c" corresponds to CHOMP [3] with a collision weight c, and "Expert" is a policy that directly uses the gradient of the geodesic distance field as input, as shown in Eq. (4). The expert does not always achieve a 100% success rate, as it is still part of a larger policy-based system; i.e. in certain cases the local avoidance policies do not let the expert pass through very narrow spaces. It is important to note that CHOMP and the expert both use **privileged** information (the full map), whereas all other systems **only** have access to immediate sensor rays.

The FFNav and RecNav systems outperform all other local methods for almost all scenarios. It seems that despite having access to the full map, CHOMP struggles with the quasi-global planning problems we used here to push local navigation algorithms to their limits. We found that CHOMP requires a large collision weight to navigate around the thin walls present in the *plane* worlds. However, such a high value decimates performance in the *sphere box* worlds, as the more 'bulky' obstacles result in a very strong obstacle gradient, leading to unstable behavior that pushes CHOMP to the edges of, or even outside of, the world. The baseline method and our neural network-based extension methods achieve surprisingly high success rates - especially considering the fact that none of these methods have access to a map.

B. Real-World Scenes

While using very densely occupied simulated worlds gives an idea of the relative performance, the true test is how well it generalizes to real problems outside of the training distribution. We evaluate several variants of the proposed reactive algorithms as shown in the previous sections against each other, as well as CHOMP (CHMP40) and RRT* [5]. We use three datasets: a 120-obstacle variant of the generated *sphere box* world (SB 120), and the 2 worlds based on real environments shown in Fig. 9. While we still evaluate these worlds in simulation, they are built up from raw sensor data, better reflecting real world performance and noise models. For each world we generate 100 random start and goal points at least $3.0m$ apart, without a line-of-sight connection. We additionally ablate network size by introducing sFFNav; a smaller version of FFNav with all learned layers of size ≥ 128 halved. The results are shown in Table I.

We show that our learned methods are able to zero-shot generalize to environments based on real data, despite never having seen anything similar during training time. It is also again important to emphasize that CHOMP and RRT* have access to the complete map, while all the reactive methods only have access to the sensor rays at each timestep. As a result, our reactive methods also have a much smaller memory footprint: requiring only the ray information to be stored, not a complete volumetric reconstruction.

While all learned methods outperform the reactive baseline, RecNav only shows improved performance in simulation, not on the real-world datasets. We hypothesize that RecNav "overfits" more to the distribution of environments used during training, as the effective input space of a recurrent network is much larger. Additionally, the smaller sFFNav performs worse than FFNav on the synthetically generated world, but has roughly similar performance on the datasets based on real world data. Similar to RecNav, we believe that the smaller network size may lead to better generalization to different environments.

As in previous evaluations, all learned methods outperform the CHOMP local planner. RRT*, as a global planning method with a much larger time budget, has the highest success rate, however, requiring access to the full map. We propose that with a limited time budget, it may be advantageous to skip building a map for local planning and instead rely entire on reactive or neural-reactive methods. Another important aspect is that CHOMP and RRT* only provide output once the solver reaches a stopping criterion and a full trajectory is created, whereas the reactive methods simply need to evaluate the next-best acceleration at each time step. The individual query time highlights this contrast, as the reactive methods can be queried orders of magnitude faster than CHOMP and RRT*. There is additional room for optimization: Compiling RecNav using the TensorRT compiler runs inference at 2.7 kHz on an Nvidia Jetson Orin NX, which would lead to a query time of only 0.4 *ms* (note that the times in Table I are using non-compiled, default pytorch implementations).

C. Robustness to Noise

We evaluate the robustness of the systems by adding noise to the rays (given as input to both the network and the avoidance policies). We choose a multiplicative noise model, where a ray distance d_r is converted to a noisy observation using a normal distribution $\mathcal{N}(\cdot)$: $d_r \cdot \left(1 + \mathcal{N}\left(0, \sigma_n^2\right)\right)$, parameterized by the noise standard deviation σ_n. This is done independently for every ray. Table II shows the performance of RecNav on the SUN3D world with different noise levels. The progression of these numbers is similar for different world types; performance is constant up to 30% noise, and quickly drops off afterwards. This indicates that the system is extremely robust to noise levels up to 30%, which is far above the noise levels reached by modern distance sensors.

VII. Discussion

We show that adding a neural network to a purely reactive planner helps navigation in very cluttered environments. By training on a geodesic field on a fully known map, we give our learned planners a form of geometric intuition on how to escape local minima based only on current sensor rays. The addition of an LSTM component further increased the success rate. We hypothesize that some temporal consistency

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

World	Plnr	M	Suc	Len *[m]*	QT *[ms]*	TTA *[ms]*	Mem *[MB]*
SB	CHMP	X	0.78	7.67±2.58	113±69	113±69	3.9
	RRT*	X	0.98	7.54±2.42	100	100	3.9
	Base		0.75	7.22±2.44	169±83	0.4±0.1	0.1
	sFFNav		0.86	7.21±2.45	893±323	2.6±0.2	3.4
	FFNav		0.88	7.13±2.34	939±314	2.8±0.2	7.8
	RecNav		0.91	7.17±2.39	1900±818	5.7±1.7	9.9
S3D	CHMP	X	0.65	6.08±1.76	279±78	279±78	159.0
	RRT*	X	0.95	5.67±1.84	100	100	165.0
	Base		0.56	5.64±1.91	153±66	0.5±0.0	0.1
	sFFNav		0.69	5.56±1.88	718±215	2.6±0.2	3.4
	FFNav		0.68	5.54±1.88	760±239	2.8±0.2	7.8
	RecNav		0.68	5.55±1.88	1540±649	5.6±1.5	9.9
BF	CHMP	X	0.36	4.08±0.88	178±82	178±82	82.0
	RRT*	X	1.00	3.75±0.58	100	100	90.1
	Base		0.60	3.63±0.58	81±16	0.5±0.0	0.1
	sFFNav		0.74	3.64±0.59	445±82	2.6±0.1	3.4
	FFNav		0.74	3.64±0.59	485±99	2.8±0.2	7.8
	RecNav		0.72	3.64±0.59	1011±295	5.9±1.4	9.9

TABLE I: Planning statistics for 100 runs with standard deviation for the RMP planners, CHOMP and RRT* (AMD EPYC 7742, Nvidia GeForce RTX 2080 Ti). We compare the SUN3D (S3D) and BundleFusion (BF) worlds depicted in Fig. 9, and a sphere box world with 120 obstacles (SB). The columns indicate whether the method requires a full map (M), the success rate (Suc), the length of the trajectory (Len), the time to generate a complete trajectory (time to answer, TTA), the query time (QT), and the memory usage (Mem). Length, TTA, and QT are all calculated on the subset of planning runs where *all* methods are succesful per world type, such that the aggregated numbers can be compared directly.

	Noise stddev σ_n	0%	10%	20%	30%	40%
FFNav	Sucess Rate	0.68	0.68	0.68	0.68	0.33
	Trajectory Length *[m]*	4.80	4.81	4.80	4.80	5.12
RecNav	Success Rate	0.68	0.69	0.69	0.69	0.40
	Trajectory Length *[m]*	4.81	4.81	4.81	4.81	5.01

TABLE II: Success rate and trajectory lengths for different levels of noise for FFNav and RecNav on the SUN3D worlds. The length is calculated on the subset of runs where *all* methods in this table are successful, meaning that the aggregated lengths are different than in Table I.

is important to avoid certain types of local minima. Adding an LSTM component also comes with trade-offs - it generally makes training less efficient and generalization outside of the training distribution is slightly worse.

We used the geodesic field as a training supervision signal, as it acts a proxy for a global planner: pointing in the direction of the goal around obstacles throughout the map. However, the geodesic field is constructed such that it always points along the geodesic - the shortest possible path. The drawbacks of this are shown in Fig. 7d, where the geodesic field is pointing in the direction of a very narrow opening above the wall, which might not be the ideal path for a real robot to traverse. While path length is often an important metric, it may not be the ultimate objective. Depending on the environment, robot and scenario, it is sometimes better to take a longer path that fulfills other desired properties, e.g. avoiding narrow geometries or using fewer turns. Especially in reactive navigation, finding the shortest path is not necessarily the ultimate objective; *progress* towards the goal, by going towards large perceived open spaces, is often more useful. Another interesting aspect is the multi-modality of the navigation network output. In many situations there are multiple viable next directions that can be taken, which motivates future research into how to best exploit such output. However, the results of this work provide meaningful and important insight into the difficulty and nature of reactive navigation in potentially cluttered 3D scenes *in general*. Coming back to our introductory question – Can a robot navigate a cluttered environment without a map? – we here present evidence that for many realistic *navigation* use-cases, reactive navigation combined with a higher-level geometric intuition can suffice. However, achieving high success rates in long-horizon planning problems is shown to be difficult to achieve using only the implicit spatial reasoning of a small recurrent element.

Considering a full robotic system, especially floating-base mobile robots where drift-free state estimation can be difficult to guarantee, the use of a reactive method greatly simplifies the operational complexity and state estimation quality requirements. By using a reactive navigation system that only needs the current sensor input to navigate safely, robot designers can use less accurate odometry sources, have fewer requirements on time synchronization between sensors, and reduce overall computational complexity of their systems. Furthermore, the navigation policies themselves are extremely robust to noise on the input sensor data, hopefully bringing safety even with lower-cost sensors.

VIII. Conclusion

In this paper we presented two neural-network-based navigation methods, that when combined with a classical reactive safety layer, enable navigation through very densely cluttered 3D worlds using only local sensor data and without a map. Our system outperforms other local, well-known methods and is trained in a fully self-supervised fashion in auto-generated worlds. Additionally, it is capable of zero-shot transfer to real 3D environments, and has high robustness to noise. The modular architecture facilitates the seamless combination of the navigation stack with any other task formulated as an RMP, and enables introspection to gain intuition about *how* the learned components exert their influence. We exploit the introspectability of the presented system for understanding the challenges and nature of local navigation in 3D spaces. The ability of the system to find its way through extremely cluttered maps with only local data is surprising and highly relevant for practical applications, where robust traversal of our cluttered and semi-structured world with minimal system requirements is highly important.

References

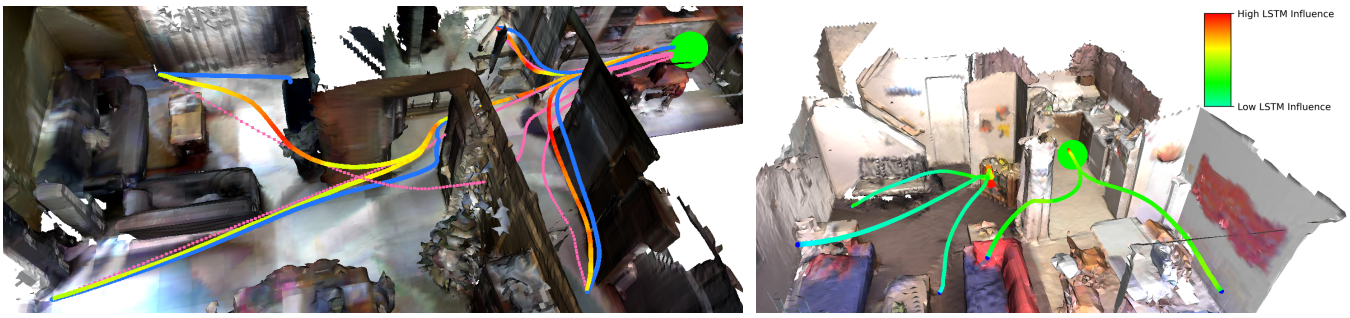
[1] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Trans. Graph.*, 2017.

[2] O. Khatib, "The potential field approach and operational space formulation in robot control," in *Adaptive and Learning Systems: Theory and Applications*. Springer, 1986.

[3] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[4] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe local exploration for replanning in cluttered unknown

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.



(a) *home_at_scan1_2013_jan_1* from the SUN3D dataset [25]. (b) *apt0* from the BundleFusion dataset [1].

Fig. 9: Navigation results on real-world data examples, Baseline (blue), CHOMP (pink), RecNav (gradient showing LSTM influence). The learned system is transferred zero-shot to real world environments, only using the local Lidar-like ray-casting as input and not relaying on a map or privileged information at all. In (a) RecNav is able to navigate past a local minimum in which the baseline gets stuck, and is also more efficient when navigating through dense geometry such as doorways. CHOMP fails on longer trajectories through multiple doorways. In (b) RecNav gets stuck in a larger local minimum.

environments for microaerial vehicles," *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, 2011.

[6] L. Montano and J. R. Asensio, "Real-time robot navigation in unstructured environments using a 3d laser rangefinder," in *IEEE/RSJ International Conference on Intelligent Robot and Systems (IROS)*. IEEE, 1997.

[7] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[8] M. Pantic *et al.*, "Obstacle avoidance using raycasting and riemannian motion policies at khz rates for mavs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[9] M. Mattamala, N. Chebrolu, and M. Fallon, "An efficient locally reactive controller for safe navigation in visual teach and repeat missions," *IEEE Robotics and Automation Letters*, 2022.

[10] P. Mirowski *et al.*, "Learning to navigate in complex environments," in *International Conference on Learning Representations (ICRL)*, 2016.

[11] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza, "Learning perception-aware agile flight in cluttered environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[12] S. Ross *et al.*, "Learning monocular reactive uav control in cluttered natural environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[13] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research. PMLR, 2011.

[14] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, 2021.

[15] D. Batra *et al.*, "Objectnav revisited: On evaluation of embodied agents navigating to objects," *arXiv preprint arXiv:2006.13171*, 2020.

[16] J. Krantz *et al.*, "Navigating to objects specified by images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 10 916–10 925.

[17] R. Partsey, E. Wijmans, N. Yokoyama, O. Dobosevych, D. Batra, and O. Maksymets, "Is mapping necessary for realistic pointgoal navigation?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 232–17 241.

[18] K.-H. Zeng *et al.*, "Poliformer: Scaling on-policy rl with transformers results in masterful navigators," *arXiv preprint arXiv:2406.20083*, 2024.

[19] K. Yadav *et al.*, "Offline visual representation learning for embodied navigation," in *Workshop on Reincarnating Reinforcement Learning at ICLR 2023*, 2023.

[20] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[21] J. Zhang, L. Tai, M. Liu, J. Boedecker, and W. Burgard, "Neural slam: Learning to explore with external memory," *arXiv preprint arXiv:1706.09520*, 2017.

[22] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Communications of the ACM*, 1964.

[23] A. Millane *et al.*, "nvblox: Gpu-accelerated incremental signed distance field mapping," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 2698–2705.

[24] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts." *Proceedings of the National Academy of Sciences*, 1996.

[25] J. Xiao, A. Owens, and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels," in *IEEE International Conference on Computer Vision (ICCV)*, 2013.