Visual Memory for Robust Path Following

Ashish Kumar* **Saurabh Gupta*** **David Fouhey** **Sergey Levine** **Jitendra Malik**
University of California, Berkeley
{ashish_kumar, sgupta, dfouhey, svlevine, malik}@eecs.berkeley.edu

Abstract

Humans routinely retrace a path in a novel environment both forwards and backwards despite uncertainty in their motion. In this paper, we present an approach for doing so. Given a demonstration of a path, a first network generates an abstraction of the path. Equipped with this abstraction, a second network then observes the world and decides how to act in order to retrace the path under noisy actuation and a changing environment. The two networks are optimized end-to-end at training time. We evaluate the method in two realistic simulators, performing path following both forwards and backwards. Our experiments show that our approach outperforms both a classical approach to solving this task as well as a number of other baselines.

1 Introduction

Consider the first morning of a conference in a city you have never been to. Rushing to the first talk, you might follow your phone's directions through a series of twists and turns to reach the venue. When you return later in the day, you can retrace your steps to your hotel relatively robustly, remembering to take a left turn at the bistro and keep straight past the coffee shop. The next day, you may probably only look at your phone to check your email. At first glance, this seems like a trivial ability. Humans routinely do this, for instance when a friend shows you the bathroom in their apartment or when you go to a room in a new building. On second glance though, it is an amazing ability since one never retraces one's steps exactly and the visual experience is constantly changing in fairly dramatic ways: people move their cars, shops open and close, and seasons change. This paper aims to replicate this ability to retrace paths (including reversals) in new environments with imperfect ability to replicate one's actions (actuation) as well as a changing world.

How might we solve this problem? One classical approach, common in robotics, would be to build a full 3D model of the world via SLAM, from building facades to the side-mirrors of cars, during the first pass; after this is done, path following reduces to localizing in the model and selecting the best action. For the task of navigation, this is simultaneously too much work – precisely reconstructing the facade is less important than recognizing it as "the bistro at which I turn left" – as well as too little work – the parts of the reconstruction that provide stable localization and the parts that do not are mixed together with no way to disentangle them.

Given the difficulties of the classical approaches, a large number of learning-based approaches have sprung up aiming to solve this problem and related navigation tasks (*e.g.* [27, 26, 17, 34, 42, 29]). In these works, an agent learns in an end-to-end fashion from a set of images to perform a navigation task. However, unlike our setup of a single demonstration in a new environment, many of these setups require the agent to have a great deal of experience with the *test* environment that is either supervised (through rewards) [27, 26] or self-supervised [34, 30], akin to a tour guide or a day of wandering. Moreover, unlike most work in navigating in new environments [17], our setup poses the additional

*Equal contribution. Project website with videos: https://ashishkumar1993.github.io/rpf/.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

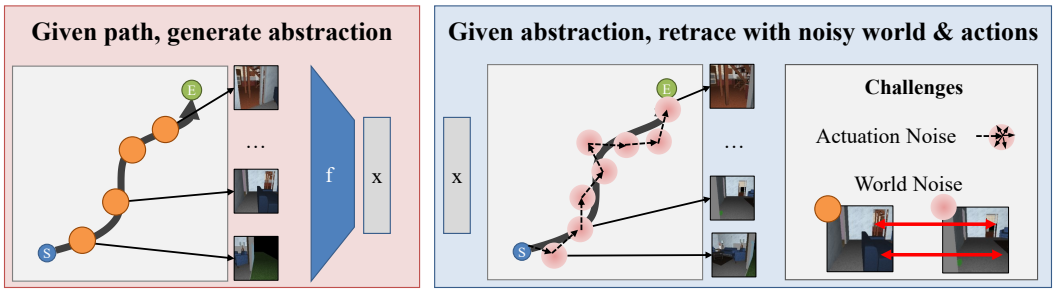

Figure 1: Problem setup: Given a path, we generate an abstraction. Equipped with this abstraction, an agent can then retrace the path both forwards and backwards both under actuation noise (i.e., uncertainty in movement) as well as in a changing world (e.g., the chairs have been moved between the demonstration and repetition). This paper presents an end-to-end way of learning how to both generate and follow a path abstraction.

challenge of noisy actuation as well as a world that can change between the initial demonstration and path execution.

Our approach, which we describe in Section 3, consists of a module that learns to convert a series of observations of a path to an abstract representation, and a learned controller that *implicitly* localizes the agent along this abstracted path using the current observation and outputs actions that bring the agent to the desired goal location. We see a number of advantages to training the whole approach end-to-end on data in comparison to classical approaches. First, the learned model can use statistical regularities to make its performance more robust: for example, it can learn to count doors in a textureless hallway rather than localize at each point and when it returns along a path, it can learn to look on the right for a table that was previously on the left. Second, by virtue of being learned entirely end-to-end, rather than being learned and designed piecemeal, the approach can automatically learn the features that are necessary for the task at hand. As a concrete demonstration, we evaluate a homing task in which the agent retraces a path in reverse; the network learns to produce features necessary for solving this task without explicitly designing any wide-baseline features or proxy tasks.

We evaluate our approach on multiple datasets in Section 4 in a series of experiments that aim to probe to what extent we can learn to retrace a path under noisy actuation and in a changing world. We compare to a variety of classical and learned alternate approaches, and outperform them. In particular, our experimental results show the value of end-to-end learning the entire path-following process.

2 Related Work

In this work, we study the problem of retracing a path under noisy actuation and a changing world. This touches on both classical work, primarily in robotics, as well as recent works that are more learning-based.

One classic approach to solving the path retracing problem (also referred to as Visual Teach and Repeat [14]) is to chain together core robotics primitives of mapping, localizing, and planning: on the initial demonstration, the agent builds a 3D map; during subsequent navigation attempts, the agent localizes itself in the map and generates actions accordingly. Each problem has been studied extensively in robotics, typically focusing on geometric solutions, *i.e.* using metric maps and locations. For example, mapping has been often studied as the classic simultaneous localization and mapping (SLAM) problem [11–13, 18] in which one builds a 3D map of the world. Localization is often framed as relocalizing a camera with respect to a global or local 3D map [9, 8] or by performing visual odometry [28, 41] to obtain a metric displacement from a start position. Finally, planning is often done assuming direct access to a noiseless map and often a noiseless agent location [5, 21, 24].

The distinction from this line of work is that our approach is learned end-to-end. This means that intermediate representations are automatically learned, rich non-geometric information can be incorporated, and modules are optimized jointly for the end-goal. The communities that developed purely geometric and pipelined approaches are, of course, aware of their limitations and have developed extensions and improvements that for instance explicitly incorporate semantics into SLAM [3] or add margins in planning to account for uncertainty [2], or use learning for sub-sets of the

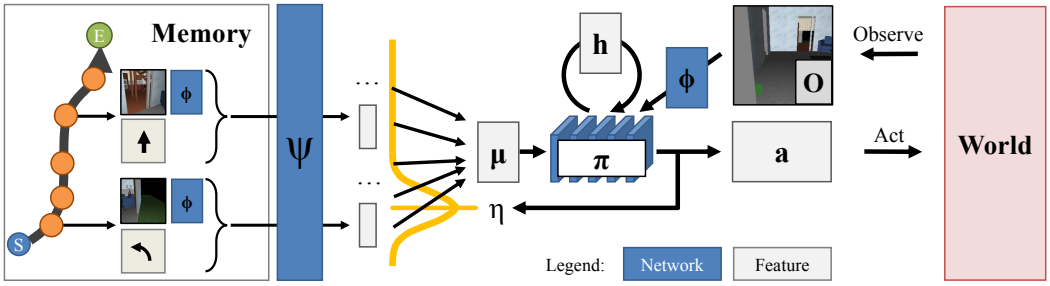

Figure 2: Proposed Approach: As input, our model takes a sequence of images (processed by ϕ) and actions at these images $\mathbf{a}(\mathbf{p})_j$. It abstracts this sequence into a sequence of memories. A second, recurrent, network $\boldsymbol{\pi}$ uses this sequence of memories to emit actions that retrace the path. At each timestep, $\boldsymbol{\pi}$ reads in: (a) the sequence, softly attending to the relevant part at past timestep η; (b) the current observation from the world (also processed by ϕ); as well as its recurrent hidden state $\mathbf{h}$. As output, $\boldsymbol{\pi}$ updates the attention location η and its hidden state and emits an action a.

SLAM problem [4, 7, 20, 22, 31, 35, 39]. The strength of our proposed method is that these strategies are learned automatically and are specified by what is needed empirically to navigate an environment as opposed to human intuition.

We are not the first to recognize the potential for end-to-end task-driven navigation. In recent years there has been a flurry of work in this area. A lot of this work focuses on the shorter time-scale task of collision avoidance, *e.g.* how do I move through the door without bumping into it [10, 15, 16, 19, 32, 33]. In contrast, our work focuses on the longer time-scale problem of path following *e.g.* how do I get back to my office from the coffee machine. At this time-scale, some works have framed the problem of navigation as learning to reach different goals in a fixed training environment [26, 27], designing policies that can directly act in new test environments [17, 23, 29, 40], self-supervised learning for reaching goals in a well-traversed environment [34] or by following a demonstration [30]. Our work is more similar to this last line of work [30, 34]. It is distinguished, however, by its lack of explicit localization as well as its navigation in an entirely *new* environment under *actuation noise* and *changing environments*. This noise is a crucial distinction because perfect replay of actions and an unchanged environment means that memorization is sufficient. We also study the task of homing, where no direct demonstration images are available.

3 Approach

3.1 Problem Setup

Consider an agent that we are trying to train to operate in a new environment E that it has never been in before. Let us assume that the agent's state at time t is represented by s_t, and that the agent has some primitive set of *stochastic* actions $\mathcal{A}$ that it can execute. Executing an action $a \in \mathcal{A}$, takes the agent to state s_{t+1} via a stochastic transition function f *i.e.* s_{t+1} is sampled randomly from the distribution $f(s_t, a, E)$. We assume that the agent is equipped with a first person RGB camera to obtain visual observations of the environment $I = \rho(E, s)$, where the function ρ renders an environment E from agent's current location specified by s. We do not tackle the problem of higher-level planning and abstract away low-level motor control to focus on the problem of robust path following. Our approach can be used with classical (or even learned) approaches for path planning and low-level motor control.

Suppose we move this agent around in this new environment along a path $\mathbf{p}$. Given such a traversal, we want the agent to be able to reliably re-trace the path $\mathbf{p}$ or to follow a related path such as the reversed path (denoted $\tilde{\mathbf{p}}$). We want to be able to do this in situations where the agent has noisy actuation and sensing, and the environment changes (to E') between the demonstration of $\mathbf{p}$ and when the agent is tested to autonomously traverse $\mathbf{p}$ (or $\tilde{\mathbf{p}}$). Note that, the only sensory information that is available to the agent as it is trying to traverse the path $\mathbf{p}$ (or $\tilde{\mathbf{p}}$) in the environment E (or E') is via ρ. It does not have access to the ground truth state of the system s_t, but only RGB image observations of the environment from that state.

Our goal is to learn a policy Π that will achieve this. As input, Π assumes access to the: path $\mathbf{p}$; actions $\mathbf{a}(\mathbf{p})$ taken while executing the path; visual observations along the path $\mathbf{I} = \{I_1 \dots I_J : I_j = \rho(E, \mathbf{p}_j)\}$; as well as the current visual observation O. As output, $\mathbf{\Pi}$ predicts actions from $\mathcal{A}$ that successfully and efficiently convey the agent to the destination $\mathbf{p}_J$, *i.e.* $\hat{a} = \Pi(\mathbf{p}, \mathbf{I}, O)$. This action $\hat{a}$ is executed in the stochastic environment; the agent obtains the new visual observation at the next state; and this process is repeated for a fixed number of time steps.

We now describe the policy function Π. Intuitively, our policy function Π uses the observation O to implicitly localize the agent with respect to its memory of the path. Given this localization, the policy reads out relevant information (such as relative pose and actions) and uses this in context of O to take an action that conveys the agent to the desired target location. The entire policy is implemented using neural network modules that are differentiable and learned end-to-end using training data for the task of efficiently going to the desired goal location.

We first describe the basic architecture for Π where we want to retrace the same path $\mathbf{p}$ that was demonstrated and then describe the extension to retrace a related path $\tilde{\mathbf{p}}$. We call our proposed policy Π as Robust Path Following policy and denote it by RPF.

3.2 Learned Controller for Robust Path Following

The policy $\Pi(\mathbf{p}, \mathbf{I}, O)$ is realized as follows. We first use $\mathbf{p}$ and $\mathbf{I}$ to compute a path description $M(\mathbf{p})$ that captures the local information needed to follow the path. $M(\mathbf{p})$ is a sequence of tuples consisting of features of the reference image, associated reference position and the associated reference action for each step j in the trajectory, or

$$M(\mathbf{p}) \quad = \quad \{(\mathbf{a}(\mathbf{p})_j, \phi(I_j)) : j \in [1 \dots J]\}. \tag{1}$$

We use this path description with a learned controller that takes as input the current image observation O to output actions to follow the path under noisy actuation. We represent the policy π with a recurrent neural network that iterates over the path description $M(\mathbf{p})$ as the agent moves through the environment. This iteration is implemented using attention that traverses over the path description. At each step, the path signature is read into μ_t with differentiable soft attention centered at η_t:

$$\mu_t \quad = \quad \sum_j \psi\left(M(\mathbf{p})_j\right) e^{-|\eta_t - j|}. \tag{2}$$

The recurrent function π with state h_t is implemented as:

$$h_{t+1}, \eta, \hat{a} = \pi(h_t, \mu_t, \phi(O)) \text{ and } \eta_{t+1} = \eta_t + \sigma(\eta). \tag{3}$$

As input, it takes the internal state, h_t, attended path signature μ_t and featurized image observation $\phi(O)$. In return, it gives a new state h_{t+1}, pointer increment η, and action $\hat{a}$ that the agent should execute. This pointer increment is added, after a sigmoid, to yield the new pointer η_{t+1}. We set $\eta_1 = 1$ and $h_1 = \mathbf{0}$.

Note that we factor out the controller from the path description. This factorization of the environment and goal specific information into a path description that is separate from the policy lets us learn a *single* policy π that can do different things in different environments with different path descriptions without requiring any re-training or adaptation. The policy can then also be thought of as a robust parameterized goal-oriented closed-loop controller.

3.3 Feature Synthesis for Following Related Paths

So far our approach can only repeat the paths that we have already taken, but our approach can be extended to follow paths $\tilde{\mathbf{p}}$ that are related to but not the same as the path $\mathbf{p}$ that was demonstrated. We do this by *synthesizing* features for the path $\tilde{\mathbf{p}}$ using whatever information is available for the demonstrated path $\mathbf{p}$. We synthesize features $\hat{\phi}(\tilde{\mathbf{p}}_j)$ for location $\tilde{\mathbf{p}}_j$ using observed features $\phi(I_i)$ from location $\mathbf{p}_i$ as follows:

$$\omega_{i,j} = \Omega\left((\phi(I_i), \delta(\mathbf{p}_i, \tilde{\mathbf{p}}_j))\right) \text{ and } \hat{\phi}(\tilde{\mathbf{p}}_j) = \Sigma_g\left(\omega_{1,j}, \omega_{2,j}, \dots, \omega_{N,j}\right). \tag{4}$$

Here, the function δ computes the relative pose of image I_i with respect to the desired synthesis location $\tilde{\mathbf{p}}_j$. ϕ computes the representation for image I_i through a CNN followed by two fully

connected layers. Ω fuses the relative pose with the representation for the image to obtain the contribution $\omega_{i,j}$ of image I_i towards representation at location $\tilde{\mathbf{p}}_j$. These contributions $\omega_{i,j}$ from different images are accumulated through a weighted addition by function Σ_g to obtain the synthesized feature $\hat{\phi}(\tilde{\mathbf{p}}_j)$ at location $\tilde{\mathbf{p}}_j$. The path description $M(\tilde{\mathbf{p}})$ can then be obtained as a collection of tuples $(\mathbf{a}(\tilde{\mathbf{p}})_j, \hat{\phi}(\tilde{\mathbf{p}}_j))$.

Implementation Details. We now describe the particular architecture that we use throughout the paper. ϕ is a 5 layer Convolutional Network with [32, 64, 128, 256, 512] filters respectively. Each Conv layer is followed by a maxpooling. ψ and Ω are fully connected networks consisting of two layers; π is implemented using GRUs. We train the entire network from scratch in an end-to-end manner using Adam optimizer for 120000 iterations, where each episode is 40 steps long.

4 Experiments

This paper studies the task of retracing a route in a *new* environment (either forwards or backwards) under noisy actuation and a changing world given a demonstration. Our experiments are designed to evaluate a) to what extent can we solve this task, b) what is the role of visual memories in doing so, and c) how our proposed solution compares to classical geometry based-solutions. Crucially, we also study how well our policies perform on settings outside of what they were trained on.

4.1 Experimental setup

Simulators. We use two simulators that permit rendering from arbitrary viewpoints and allow separation of held-out environments for testing. The first simulator is based on real world scans from *Stanford Building Parser Dataset* [1] (SBPD) and the Matterport 3D Dataset [6](MP3D). These scans have been used to study navigation tasks in [17], and we adapt their simulation code. We use splits that ensure that the testing environment comes from an entirely different building than the training environment. The second simulation environment is based on *SUNCG* [38]. SUNCG consists of synthetic indoor environments with manually created room and furniture layouts that have corresponding textured meshes [38]. Because these environments are graphics codes, SUNCG permits evaluation of the effect of environmental changes. In particular, objects can be removed without inducing artifacts that a network will pick up on.

Agent Actions. We assume that the agent has 4 macro-actions, stay in place, rotate left or right by θ ($= 30^\circ$), and move forward x units ($= 40cm$).

Noise Model. Our work studies path retracing both with actuation noise (*i.e.* the outcome of actions is stochastic) and a changing world (the world changes between the demonstration and autonomous operation of the agent). *Actuation Noise.* In both environments, when the agent outputs the rotation actions it actually rotates by $\sim N_{trunc}(\theta, 0.2|\theta|)$. When the agent executes a move forward action it rotates by $\sim N_{trunc}(0, 0.2|\theta|)$ and then translates by $\sim N_{trunc}(x, 0.2x)$. We vary the 0.2 factor in our experiments. *World Changes.* World changes are studied in the SUNCG environment. Demonstrations are provided in an environment with objects removed uniformly with a probability of r ($= .5$). The task is to get to the desired target location in presence of even fewer objects (r is .1, or .3) or additional objects (r is .7, or .9).

Tasks. Given a path $\mathbf{p}$ from $\mathbf{p}_0$ to $\mathbf{p}_T$, we consider the two tasks of *trajectory following i.e.* going from $\mathbf{p}_0$ to $\mathbf{p}_T$ as well as *homing i.e.* going from $\mathbf{p}_T$ to $\mathbf{p}_0$. We evaluate these tasks under a variety of noise conditions and environments.

Evaluation Criteria. We characterize the success of the agent by measuring how close it gets to the goal location. We analyze each approach over 500 trials in a novel environment not seen during training and compute the normalized distance-to-the-goal (the distance-to-the-goal at end of episode divided by the distance-to-the-goal at start of episode). We report both the *Median Normalized Distance* as well as the *Success Rate*, which we define as being within 10% of the initial distance to goal or within 2 steps, whichever is larger.

Model Training. We use imitation learning to train our policies. Although the agent never has access to its true location as it traverses the environment, the true location is available in the simulator. This is used to compute a set of 'good actions' that will convey the agent to the desired target location. This set of good actions are actions that lead to a larger reduction in the distance to goal when compared

Table 1: Performance over 500 trials on the Stanford Building Parser Dataset (area4) in base settings for the Following and Homing tasks. We report Success Rate and Median Normalized Distance. We also indicate a bootstrapped 95% confidence interval. See text for details.

	Open Loop	Visual Servoing	3D Recons. + Localize	RPF (no visual memory)	RPF
Following					
Success Rate	0.22 (0.19, 0.26)	0.32 (0.28, 0.36)	0.83 (0.79, 0.86)	0.77 (0.73, 0.80)	0.84 (0.80, 0.87)
Median Norm. Dist.	0.26 (0.23, 0.29)	0.20 (0.16, 0.26)	0.09 (0.08, 0.09)	0.07 (0.07, 0.08)	0.06 (0.06, 0.06)
Homing					
Success Rate	0.22 (0.19, 0.26)	-	0.00 (0.00, 0.00)	0.77 (0.73, 0.80)	0.81 (0.78, 0.85)
Median Norm. Dist.	0.26 (0.23, 0.29)	-	0.85 (0.83, 0.87)	0.07 (0.07, 0.08)	0.06 (0.06, 0.07)

to forward action. We optimize the policy to minimize the negative log probability of the sum of probabilities of all good actions at each time step.

4.2 Baselines

We compare with a number of baselines that represent either classical approaches or test the importance of various components of our system.

Open Loop. We repeat the reference actions (or their reverse for homing). Under perfect actuation, this would achieve perfect performance. With actuation noise, this baseline is a measure of the hardness of the task and tests to what extent learning to act under actuation noise is necessary.

Visual Servoing. For each action, we compute the $L2$ distance between SIFT feature matches of target reference image (initially set to first reference image) and image expected after executing that action (we obtain this image by virtualy executing this step in the simulator). The policy actually executes the action that has the lowest distance. To decide when to increment the target image, we check the ground truth proximity to the next reference image. We stop when the agent reaches the last target image.

3D Reconstruction and Localization. We use the publicly available COLMAP package [36, 37] that implements a variety of geometric mapping and localization algorithms. Note that these geometry-based methods require high-resolution images at high frame rates. Thus, we sample high-resolution images (1024×1024 *vs.* 224×224 for our policies) at $5\times$ the frame rate (145 *vs.* 30 images for our policy for a trajectory of length 30) along the reference trajectory along with ground truth poses. This ensures that the reconstruction via SIFT key-point matching [25] and bundle adjustment always succeeds. Given this reconstruction, the agent localizes itself by registering the SIFT key-points on the current image with the 3D reconstruction obtained from the reference images. It then estimates free space by marking a small region around each point on the reference trajectory as free. Given this inferred free space and localization, it executes the action that can most efficiently convey it to the goal location. If the localization module fails, we rotate left at each step until localization succeeds (or the episode ends).

RPF with No Visual Memory. We also compare to a policy without any visual memory. We do this by only using the action and pose as part of the memory $M(\mathbf{p})$ in Eq. 1 *i.e.* removing $\phi(I_j)$. We retain the rest of the architecture of RPF. Note that this is a competitive comparison as the policy can learn to replay actions meaningfully in context of current images from the environment.

All learned policies are trained in *base settings*: the noise level is at 20%, reference trajectories are of length 30, the policy is executed for 40 time steps, and these reference trajectories are sampled to be far from obstacles. Figure 3 and Table 1 present our experimental results. We report the success rate and median normalized distance and compare against the baselines described above. In addition, we also report a bootstrapped 95% confidence interval.

4.3 Results

Experiments on Matterport Data. We first study the following and homing tasks in static environments. As the environment does not need to change, we do these experiments on realistic Matterport

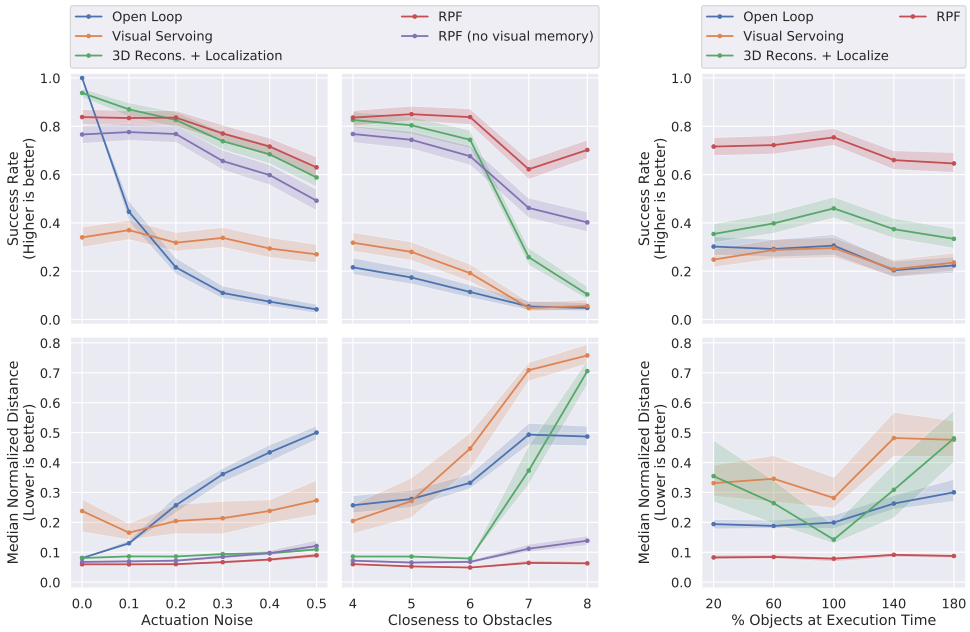

Figure 3: Generalization Performance: In addition to generalizing to new environments, our approach is able to generalize to (left) actuation noise levels, (center) obstacle distance, and (right) changing environments.

data. In particular, we train policies on 4 floors from SBPD and 6 buildings from MP3D. All policies are tested on `area4` from SBPD which is from an altogether different building than the 4 floors used for training. We found adding visually diverse data from the MP3D dataset was crucial for good performance as our models are trained entirely from scratch.

Following Task. Table 1 (top) presents results for the trajectory following task. Open loop replay only succeeds 22% of the time. This is because as actuation noise compounds over time causing both drift and collisions. Visual feedback through servoing improves success rate to 32%, though it is limited as it only localizes against a single image at a time. This is addressed by '3D Reconstruction + Localization' which uses all reference images for localization (by reconstructing the environment), and achieves a success rate of 83%. RPF achieves a similar success rate, though achieves a better median normalized distance. RPF without visual memories does worse at 77%. This points at the utility of visual memory for the task of retracing trajectories.

Homing Task. We set up the homing task by sampling the same trajectories but simply providing images from a 180° rotated camera at each reference location. Thus open loop replay and RPF without visual memories perform the same as before. Note that this scenario is impossible for visual servoing as there are no direct images to servo to. While, in principle 3D Reconstruction and Localization can tackle this scenario, it performs poorly (0% success, 0.85 median normalized distance) as SIFT features don't match well across large baselines. In comparison, RPF that learns to speculate features still performs well at 81%. Note that this is better than RPF without visual memories, demonstrating that our feature prediction technique is able to extract meaningful signal from related images.

Testing on Out-of Train Settings. We have shown so far that our trained policies outperform appropriate baselines for the task when tested on novel environments. However, we are still training and testing on the same settings, such as noise level and distribution of trajectories. We next test how well our learned policies work when we test them on settings they have not been trained on.

We do this by testing the policies trained in the 'Base Setting', in different settings in novel environments. We explore two novel settings: a) *Actuation Noise*: we vary the noise level of the environment and b) *Harder Trajectories*: we pick trajectories that are closer to obstacles and require careful maneuverability. We do not retrain our policies for these settings, and simply execute the policy learned in the 'Base Setting' on these additional settings.

Figure 3 (left) presents the results. The top row plots the success rate and the bottom row plots the median normalized distance. All plots also include a bootstrapped 95% confidence interval. Plots

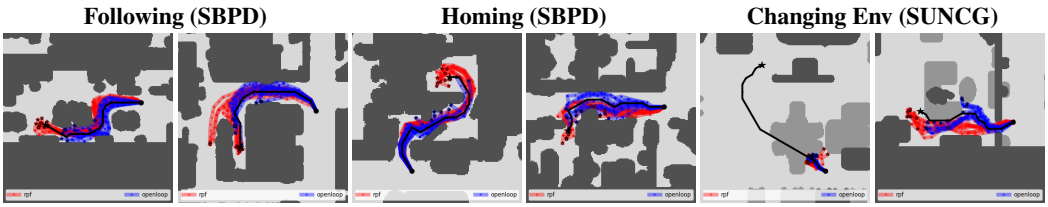

Figure 4: Trajectory Visualizations: We visualize sample success and failure trajectory in top view for the three different scenarios that we study. We show multiple roll-outs by resampling multiple times from the noise model. Note that these top views are shown here only for visualization, the policy does not receive them and operates purely based on the first person views. RPF trajectories are shown in red, and open loop trajectories are showin in blue. **Following and Homing**: RPF trajectories more tightly follow the reference trajectory, while open loop roll-outs spread out and collide with nearby obstacles. Overall, RPF get to the goal more reliably. RPF fails when it drifts too far from the reference trajectory. **Execution in Changing Envirnonment in SUNCG:** Our approach is able to go around obstacles, but fails if a very large deviation is required.

in the left column show performance as a function of actuation noise. Policies trained at 20% noise are tested under noise varying between 0% and 50%. Open loop perform perfectly with no noise, but its performance rapidly degrades as noise increases. In comparison, RPF performance degrades gracefully. Furthermore, the gap between RPF with and without visual memories widens further, emphasizing that visual fixes are useful in settings with actuation noise. Plots in the center column shows performance as we move to harder trajectories that are sampled to be closer to obstacles (as we move from the left to the right on the plot). Once again, RPF degrades much more gracefully than 3D reconstruction based method.

Finally, Figure 4 shows multiple rollouts from our RPF policy and contrasts them with purely open loop rollouts.

Experiments on SUNCG. We next report experiments on SUNCG for the base setting. We trained on 48 houses from the House3D training set and report performance on 12 entirely disjoint houses from the test set. We plot the metrics in Figure 3 (right). Base setting here corresponds to when there are 100% objects at execution time *i.e.* the environment is the same between when the reference trajectory was provided, and when the agent has to execute the trajectory. As RPF learns features, it can better adapt to change in visual imagery in the synthetic dataset as compared to feature-based geometric methods. Once again, RPF was trained in base settings.

Robustness to Environmental Changes. Figure 3 (right) also plots performance as the environment changes. Points to the left of the base setting (objects-at-execution-time less than 100%) correspond to the setting where objects are removed from the environment, while points to the right correspond to when objects are added into the environment. In both these regimes, RPF continues to perform well. In contrast, performance for 3D reconstruction and localization method degrades sharply (see median normalized distance plot) as the environment at execution time deviates from one at demonstration time. This is known and expected of geometry based methods that do not cope well with changes in the environment.

Not only does the agent need to be robust to visual changes between reference images and the current observations, but it must also exhibit local going-around-behaviour as the reference trajectory may go through the newly added obstacles. While it is able to do so when there is a minor detour, it fails when a much larger detour is required as shown in Figure 4(right).

References

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 5

[2] Brian Axelrod, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Provably safe robot navigation with obstacle uncertainty. In *RSS*, 2017. 2

[3] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *ICRA*, 2017. 2

[4] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. MapNet: Geometry-aware learning of maps for camera localization. In *CVPR*, 2018. 3

[5] John Canny. *The complexity of robot motion planning*. MIT press, 1988. 2

[6] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *3DV*, 2017. 5

[7] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In *ICLR*, 2018. 3

[8] Lee Clement, Jonathan Kelly, and Timothy D Barfoot. Robust monocular visual teach and repeat aided by local ground planarity and color-constant imagery. *JFR*, 2017. 2

[9] Mark Cummins and Paul Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *IJRR*, 2008. 2

[10] Shreyansh Daftry, J Andrew Bagnell, and Martial Hebert. Learning transferable policies for monocular reactive mav control. In *ISER*, 2016. 3

[11] Andrew J Davison and David W Murray. Mobile robot localisation using active vision. In *ECCV*, 1998. 2

[12] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014.

[13] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 2015. 2

[14] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *JFR*, 2010. 2

[15] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017. 3

[16] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *RAL*, 2016. 3

[17] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 1, 3, 5

[18] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. *UIST*, 2011. 2

[19] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. *arXiv preprint arXiv:1709.10489*, 2017. 3

[20] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *NIPS*, 2017. 3

[21] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *RA*, 1996. 2

[22] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, 2015. 3

[23] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. In *ICLR*, 2018. 3

[24] Steven M Lavalle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 2000. 2

[25] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004. 6

[26] Piotr Mirowski, Matthew Koichi Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, and Raia Hadsell. Learning to navigate in cities without a map. *arXiv preprint arXiv:1804.00168*, 2018. 1, 3

[27] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. In *ICLR*, 2017. 1, 3

[28] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *CVPR*, 2004. 2

[29] Emilio Parisotto and Ruslan Salakhutdinov. Neural Map: Structured memory for deep reinforcement learning. In *ICLR*, 2018. 1, 3

[30] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 1, 3

[31] Sudeep Pillai and John J Leonard. Towards visual ego-motion learning in robots. *arXiv preprint arXiv:1705.10279*, 2017. 3

[32] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1989. 3

[33] Fereshteh Sadeghi and Sergey Levine. (CAD)^{2}RL: Real singel-image flight without a singel real image. In *RSS*, 2017. 3

[34] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018. 1, 3

[35] Johannes L Schönberger, Marc Pollefeys, Andreas Geiger, and Torsten Sattler. Semantic visual localization. In *CVPR*, 2018. 3

[36] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 6

[37] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, 2016. 6

[38] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2018. 5

[39] Tristan Swedish and Ramesh Raskar. Deep visual teach and repeat on path networks. In *CVPR*, 2018. 3

[40] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam. *arXiv preprint arXiv:1706.09520*, 2017. 3

[41] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 2

[42] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 1