Cognitive Mapping and Planning for Visual Navigation

Saurabh Gupta[1,2] James Davidson[2] Sergey Levine[1,2] Rahul Sukthankar[2] Jitendra Malik[1,2]
[1]UC Berkeley [2]Google
[1]{sgupta, svlevine, malik}@eecs.berkeley.edu, [2]{jcdavidson, sukthankar}@google.com

Abstract

We introduce a neural architecture for navigation in novel environments. Our proposed architecture learns to map from first-person views and plans a sequence of actions towards goals in the environment. The Cognitive Mapper and Planner (CMP) is based on two key ideas: a) a unified joint architecture for mapping and planning, such that the mapping is driven by the needs of the planner, and b) a spatial memory with the ability to plan given an incomplete set of observations about the world. CMP constructs a top-down belief map of the world and applies a differentiable neural net planner to produce the next action at each time step. The accumulated belief of the world enables the agent to track visited regions of the environment. Our experiments demonstrate that CMP outperforms both reactive strategies and standard memory-based architectures and performs well in novel environments. Furthermore, we show that CMP can also achieve semantically specified goals, such as "go to a chair".

1. Introduction

As humans, when we navigate through novel environments, we draw on our previous experience in similar conditions. We reason about free-space, obstacles and the topology of the environment, guided by common sense rules and heuristics for navigation. For example, to go from one room to another, I must first exit the initial room; to go to a room at the other end of the building, getting into a hallway is more likely to succeed than entering a conference room; a kitchen is more likely to be situated in open areas of the building than in the middle of cubicles. The goal of this paper is to design a learning framework for acquiring such expertise, and demonstrate this for the problem of robot navigation in novel environments.

However, classic approaches to navigation rarely make use of such common sense patterns. Classical SLAM based approaches [14, 57] first build a 3D map using LIDAR, depth, or structure from motion, and then plan paths in this map. These maps are built purely geometrically, and nothing is known until it has been explicitly observed, even when there are obvious patterns. This becomes a problem for goal directed navigation. Humans can often guess, for example, where they will find a chair or that a hallway will probably lead to another hallway but a classical robot agent can at best only do uninformed exploration. The separation between mapping and planning also makes the overall system unnecessarily fragile. For example, the mapper might fail on texture-less regions in a corridor, leading to failure of the whole system, but precise geometry may not even be necessary if the robot just has to keep traveling straight.

Inspired by this reasoning, recently there has been an increasing interest in more end-to-end learning-based approaches that go directly from pixels to actions [45, 48, 64] without going through explicit model or state estimation steps. These methods thus enjoy the power of being able to learn behaviors from experience. However, it is necessary to carefully design architectures that can capture the structure of the task at hand. For instance Zhu *et al*. [64] use reactive memory-less vanilla feed forward architectures

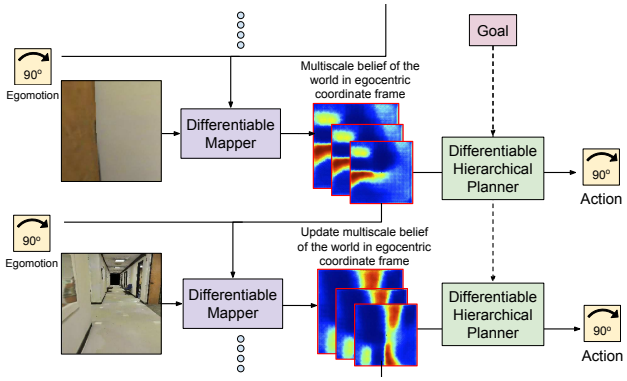

Figure 1: Overall network architecture: Our learned navigation network consists of mapping and planning modules. The mapper writes into a latent spatial memory that corresponds to an egocentric map of the environment, while the planner uses this memory alongside the goal to output navigational actions. The map is not supervised explicitly, but rather emerges naturally from the learning process.

Work done when S. Gupta was an intern at Google.

Project website with videos, appendix: https://sites.google.com/view/cognitive-mapping-and-planning/.

for solving visual navigation problems, In contrast, experiments by Tolman [58] have shown that even rats build sophisticated representations for space in the form of 'cognitive maps' as they navigate, giving them the ability to reason about shortcuts, something that a reactive agent is unable to.

This motivates our Cognitive Mapping and Planning (CMP) approach for visual navigation (Figure 1). CMP consists of a) a spatial memory to capture the layout of the world, and b) a planner that can plan paths given partial information. The mapper and the planner are put together into a unified architecture that can be trained to leverage regularities of the world. The mapper fuses information from input views as observed by the agent over time to produce a metric egocentric multi-scale belief about the world in a top-down view. The planner uses this multi-scale egocentric belief of the world to plan paths to the specified goal and outputs the optimal action to take. This process is repeated at each time step to convey the agent to the goal.

At each time step, the agent updates the belief of the world from the previous time step by a) using the egomotion to transform the belief from the previous time step into the current coordinate frame and b) incorporating information from the current view of the world to update the belief. This allows the agent to progressively improve its model of the world as it moves around. The most significant contrast with prior work is that our approach is trained end-to-end to take good actions in the world. To that end, instead of analytically computing the update to the belief (via classical structure from motion) we frame this as a learning problem and train a convolutional neural network to predict the update based on the observed first person view. We make the belief transformation and update operations differentiable thereby allowing for end-to-end training. This allows our method to adapt to the statistical patterns in real indoor scenes without the need for any explicit supervision of the mapping stage.

Our planner uses the metric belief of the world obtained through the mapping operation described above to plan paths to the goal. We use value iteration as our planning algorithm but crucially use a trainable, differentiable and hierarchical version of value iteration. This has three advantages, a) being trainable it naturally deals with partially observed environments by explicitly learning when and where to explore, b) being differentiable it enables us to train the mapper for navigation, and c) being hierarchical it allows us to plan paths to distant goal locations in time complexity that is logarithmic in the number of steps to the goal.

Our approach is a reminiscent of classical work in navigation that also involves building maps and then planning paths in these maps to reach desired target locations. However, our approach differs from classical work in the following significant way: except for the architectural choice of maintaining a metric belief, everything else is learned from data. This leads to some very desirable properties: a) our model can learn statistical regularities of indoor environments in a task-driven manner, b) jointly training the mapper and the planner makes our planner more robust to errors of the mapper, and c) our model can be used in an online manner in novel environments without requiring a pre-constructed map.

2. Related Work

Navigation is one of the most fundamental problems in mobile robotics. The standard approach is to decompose the problem into two separate stages: (1) mapping the environment, and (2) planning a path through the constructed map [17, 37]. Decomposing navigation in this manner allows each stage to be developed independently, but prevents each from exploiting the specific needs of the other. A comprehensive survey of classical approaches for mapping and planning can be found in [57].

Mapping has been well studied in computer vision and robotics in the form of structure from motion and simultaneous localization and mapping [20, 30, 33, 55] with a variety of sensing modalities such as range sensors, RGB cameras and RGB-D cameras. These approaches take a purely geometric approach. Learning based approaches [26, 61] study the problem in isolation thus only learning generic task-independent maps. Path planning in these inferred maps has also been well studied, with pioneering works from Canny [11], Kavraki *et al*. [36] and LaValle and Kuffner [44]. Works such as [18, 19] have studied the joint problem of mapping and planning. While this relaxes the need for pre-mapping by incrementally updating the map while navigating, but still treat navigation as a purely geometric problem, Konolige *et al*. [41] and Aydemir *et al*. [6] proposed approaches which leveraged semantics for more informed navigation. Kuipers *et al*. [43] introduce a cognitive mapping model using hierarchical abstractions of maps. Semantics have also been associated with 3D environments more generally [23, 42].

As an alternative to separating out discrete mapping and planning phases, reinforcement learning (RL) methods directly learn policies for robotic tasks [38, 40, 51]. A major challenge with using RL for this task is the need to process complex sensory input, such as camera images. Recent works in deep reinforcement learning (DRL) learn policies in an end-to-end manner [48] going from pixels to actions. Follow-up works [22,47,54] propose improvements to DRL algorithms, [29, 47, 50, 60, 63] study how to incorporate memory into such neural network based models. We build on the work from Tamar *et al*. [56] who study how explicit planning can be incorporated in such agents, but do not consider the case of first-person visual navigation, nor provide a framework for memory or mapping. [50] study the generalization behavior of these algorithms to novel environments

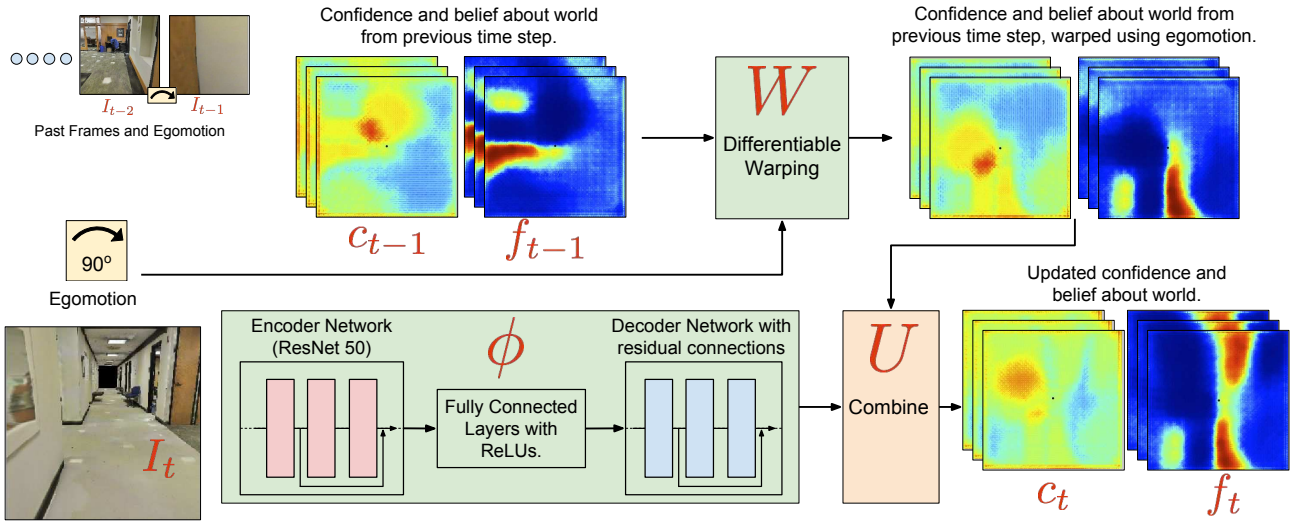

Figure 2: Architecture of the mapper: The mapper module processes first person images from the robot and integrates the observations into a latent memory, which corresponds to an egocentric map of the top-view of the environment. The mapping operation is not supervised explicitly – the mapper is free to write into memory whatever information is most useful for the planner. In addition to filling in obstacles, the mapper also stores confidence values in the map, which allows it to make probabilistic predictions about unobserved parts of the map by exploiting learned patterns.

they have not been trained on.

In context of navigation, learning and DRL has been used to obtain policies [4, 12, 13, 21, 35, 50, 56, 59, 64]. Some of these works [21, 35], focus on the problem of learning controllers for effectively maneuvering around obstacles directly from raw sensor data. Others, such as [9, 50, 56], focus on the planning problem associated with navigation under full state information [56], designing strategies for faster learning via episodic control [9], or incorporate memory into DRL algorithms to ease generalization to new environments. Most of this research (except [64]) focuses on navigation in synthetic mazes which have little structure to them. Given these environments are randomly generated, the policy learns a random exploration strategy, but has no statistical regularities in the layout that it can exploit. We instead test on layouts obtained from real buildings, and show that our architecture consistently outperforms feed forward and LSTM models used in prior work.

The research most directly relevant to our work is the contemporary work of Zhu *et al*. [64]. Similar to us, Zhu *et al*. also study first-person view navigation using macro-actions in more realistic environments instead of synthetic mazes. Zhu *et al*. propose a feed forward model which when trained in one environment can be finetuned in another environment. Such a memory-less agent cannot map, plan or explore the environment, which our expressive model naturally does. Zhu *et al*. also don't consider zero-shot generalization to previously unseen environments, and focus on smaller worlds where memorization of landmarks is feasible. In contrast, we explicitly handle generalization to new, never before seen interiors, and show that our model generalizes successfully to floor plans not seen during training.

Relationship to Contemporary Work. Since conducting this research, numerous other works that study visual navigation have come out. Most notable among these is the work from Sadeghi and Levine [53] that shows that simulated mobility policies can transfer to the real world. Mirowski *et al*. [46] study sources of auxiliary supervision for faster training with RL. Bhatti *et al*. [8] incorporate SLAM based maps for improving the performance at playing Doom. Brahmbhatt and Hays [10] study navigation in cities using feed-forward models. Zhang *et al*. [62] and Duan *et al*. [16] show how to speed up learning for related tasks. [8, 16, 46] show results in synthetic mazes, and only [10, 53, 62] show results with real images. While all these works study visual navigation, none of them leverage mapping and planning modules or propose end-to-end architectures for jointly mapping and planning which is the focus of our work.

3. Problem Setup

To be able to focus on the high-level mapping and planning problem we remove confounding factors arising from low-level control by conducting our experiments in simulated real world indoor environments. Studying the problem in simulation makes it easier to run exhaustive evaluation experiments, while the use of scanned real world environments allows us to retains the richness and complexity of real scenes. We also only study the static version of the problem, though extensions to dynamic environments would be interesting to explore in future work.

We model the robot as a cylinder of a fixed radius and height, equipped with vision sensors (RGB cameras or depth cameras) mounted at a fixed height and oriented at

a fixed pitch. The robot is equipped with low-level controllers which provide relatively high-level macro-actions $\mathcal{A}_{x,\theta}$. These macro-actions are a) stay in place, b) rotate left by θ, c) rotate right by θ, and d) move forward x cm, denoted by a_0, a_1, a_2 and a_3, respectively. We further assume that the environment is a grid world and the robot uses its macro-actions to move between nodes on this graph. The robot also has access to its precise egomotion. This amounts to assuming perfect visual odometry [49], which can itself be learned [25], but we defer the joint learning problem to future work.

We want to learn policies for this robot for navigating in *novel* environments that it has not previously encountered. We study two navigation tasks, a *geometric* task where the robot is required to go to a target location specified in robot's coordinate frame (*e.g.* $250cm$ forward, $300cm$ left) and a *semantic* task where the robot is required to go to an object of interest (*e.g.* a chair). These tasks are to be performed in novel environments, neither the exact environment map nor its topology is available to the robot.

Our navigation problem is defined as follows. At a given time step t, let us assume the robot is at a global position (position in the world coordinate frame) P_t. At each time step the robot receives as input the image of the environment $\mathcal{E}$, $I_t = I(\mathcal{E}, P_t)$ and a target location $(x_t^g, y_t^g, \theta_t^g)$ (or a semantic goal) specified in the coordinate frame of the robot. The navigation problem is to learn a policy that at every time steps uses these inputs (current image, egomotion and target specification) to output the action that will convey the robot to the target as quickly as possible.

Experimental Testbed. We conduct our experiments on the Stanford large-scale 3D Indoor Spaces (S3DIS) dataset introduced by Armeni *et al*. [5]. The dataset consists of 3D scans (in the form of textured meshes) collected in 6 large-scale indoor areas that originate from 3 different buildings of educational and office use. The dataset was collected using the Matterport scanner [2]. Scans from 2 buildings were used for training and the agents were tested on scans from the 3rd building. We pre-processed the meshes to compute space traversable by the robot. We also precompute a directed graph $\mathcal{G}_{x,\theta}$ consisting of the set of locations the robot can visit as nodes and a connectivity structure based on the set of actions $\mathcal{A}_{x,\theta}$ available to the robot to efficiently generate training problems. More details in Appendix [1].

4. Mapping

We describe how the mapping portion of our learned network can integrate first-person camera images into a top-down 2D representation of the environment, while learning to leverage statistical structure in the world. Note that, unlike analytic mapping systems, the map in our model amounts to a latent representation. Since it is fed directly into a learned planning module, it need not encode purely free space representations, but can instead function as a general spatial memory. The model learns to store inside the map whatever information is most useful for generating successful plans. However to make description in this section concrete, we assume that the mapper predicts free space.

The mapper architecture is illustrated in Figure 2. At every time step t we maintain a cumulative estimate of the free space f_t in the coordinate frame of the robot. f_t is represented as a multi-channel 2D feature map that metrically represents space in the top-down view of the world. f_t is estimated from the current image I_t, cumulative estimate from the previous time step f_{t-1} and egomotion between the last and this step e_t using the following update rule:

$$f_t = U\left(W\left(f_{t-1}, e_t\right), f_t'\right) \quad \text{where, } f_t' = \phi\left(I_t\right). \tag{1}$$

here, W is a function that transforms the free space prediction from the previous time step f_{t-1} according to the egomotion in the last step e_t, ϕ is a function that takes as input the current image I_t and outputs an estimate of the free space based on the view of the environment from the current location (denoted by f_t'). U is a function which accumulates the free space prediction from the current view with the accumulated prediction from previous time steps. Next, we describe how each of the functions W, ϕ and U are realized.

The function W is realized using bi-linear sampling. Given the ego-motion, we compute a backward flow field $\rho(e_t)$. This backward flow maps each pixel in the current free space image f_t to the location in the previous free space image f_{t-1} where it should come from. This backward flow ρ can be analytically computed from the ego-motion Appendix [1]. The function W uses bi-linear sampling to apply this flow field to the free space estimate from the previous frame. Bi-linear sampling allows us to back-propagate gradients from f_t to f_{t-1} [34], which will make it possible to train this model end to end.

The function ϕ is realized by a convolutional neural network. Because of our choice to represent free space always in the coordinate frame of the robot, this becomes a relatively easy function to learn, given the network only has to output free space in the current coordinate, rather than in an arbitrary world coordinate frame determined by the cumulative egomotion of the robot so far.

Intuitively, the network can use semantic cues (such as presence of scene surfaces like floor and walls, common furniture objects like chairs and tables) alongside other learned priors about size and shapes of common objects to generate free space estimates, even for object that may only be partiality visible. Qualitative results in Appendix [1] show an example for this where our proposed mapper is able to make predictions for spaces that haven't been observed.

The architecture of the neural network that realizes function ϕ is shown in Figure 2. It is composed of a convo-

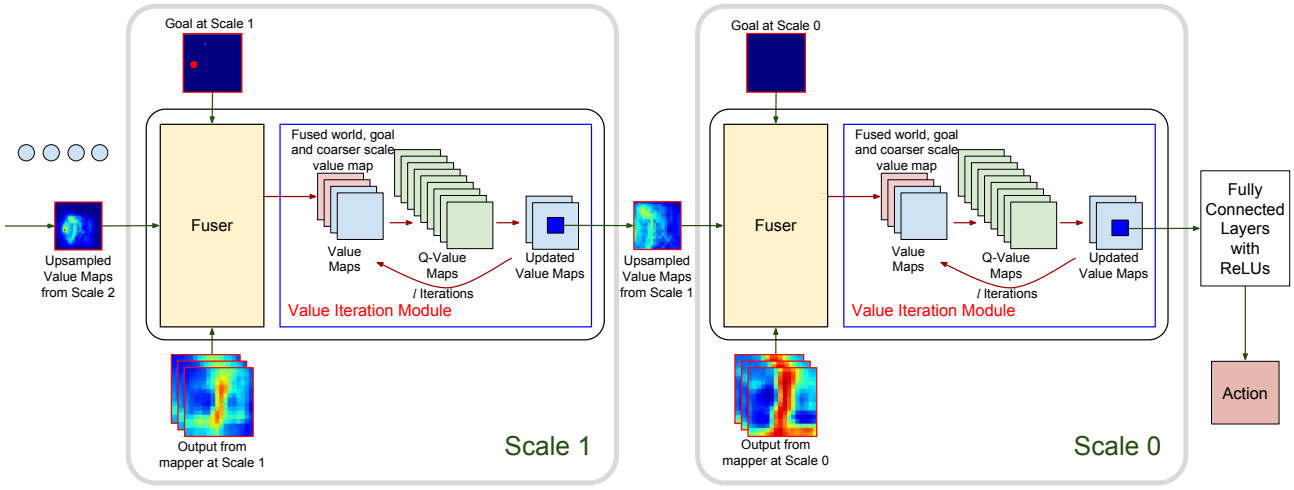

Figure 3: Architecture of the hierarchical planner: The hierarchical planner takes the egocentric multi-scale belief of the world output by the mapper and uses value iteration expressed as convolutions and channel-wise max-pooling to output a policy. The planner is trainable and differentiable and back-propagates gradients to the mapper. The planner operates at multiple scales (scale 0 is the finest scale) of the problem which leads to efficiency in planning.

lutional encoder which uses residual connections [27] and produces a representation of the scene in the 2D image space. This representation is transformed into one that is in the egocentric 2D top-down view via fully connected layers. This representation is up-sampled using up-convolutional layers (also with residual connections) to obtain the update to the belief about the world from the current frame.

In addition to producing an estimate of the free space from the current view f'_t the model also produces a confidence c'_t. This estimate is also warped by the warping function W and accumulated over time into c_t. This estimate allows us to simplify the update function, and can be thought of as playing the role of the update gate in a gated recurrent unit. The update function U takes in the tuples (f_{t-1}, c_{t-1}), and (f'_t, c'_t) and produces (f_t, c_t) as follows:

$$f_t = \frac{f_{t-1}c_{t-1} + f'_t c'_t}{c_{t-1} + c'_t} \quad \text{and} \quad c_t = c_{t-1} + c'_t \tag{2}$$

We chose an analytic update function to keep the overall architecture simple. This can be replaced with more expressive functions like those realized by LSTMs [31].

Mapper performance in isolation. To demonstrate that our proposed mapper architecture works we test it in isolation on the task of free space prediction. Appendix [1] shows qualitative and quantitative results.

5. Planning

Our planner is based on value iteration networks proposed by Tamar *et al*. [56], who observed that a particular type of planning algorithm called value iteration [7] can be implemented as a neural network with alternating convolutions and channel-wise max pooling operations, allowing the planner to be differentiated with respect to its inputs. Value iteration can be thought of as a generalization of Dijkstra's algorithm, where the value of each state is iteratively recalculated at each iteration by taking a max over the values of its neighbors plus the reward of the transition to those neighboring states. This plays nicely with 2D grid world navigation problems, where these operations can be implemented with small 3×3 kernels followed by max-pooling over channels. Tamar *et al*. [56] also showed that this reformulation of value iteration can also be used to learn the planner (the parameters in the convolutional layer of the planner) by providing supervision for the optimal action for each state. Thus planning can be done in a trainable and differentiable manner by very deep convolutional network (with channel wise max-pooling). For our problem, the mapper produces the 2D top-view of the world which shares the same 2D grid world structure as described above, and we use value iteration networks as a trainable and differentiable planner.

Hierarchical Planning. Value iteration networks as presented in [56](v2) are impractical to use for any long-horizon planning problem. This is because the planning step size is coupled with the action step size thus leading to a) high computational complexity at run time, and b) a hard learning problem as gradients have to flow back for as many steps. To alleviate this problem, we extend the hierarchical version presented in [56](v1).

Our hierarchical planner plans at multiple spatial scales. We start with a k times spatially downsampled environment and conduct l value iterations in this downsampled environment. The output of this value iteration process is center cropped, upsampled, and used for doing value iterations at a finer scale. This process is repeated to finally reach the resolution of the original problem. This procedure allows

us to plan for goals which are as far as $l2^k$ steps away while performing (and backpropagating through) only lk planning iterations. This efficiency increase comes at the cost of approximate planning.

Planning in Partially Observed Environments. Value iteration networks have only been evaluated when the environment is fully observed, *i.e.* the entire map is known while planning. However, for our navigation problem, the map is only partially observed. Because the planner is not hand specified but learned from data, it can learn policies which naturally take partially observed maps into account. Note that the mapper produces not just a belief about the world but also an uncertainty c_t, the planner knows which parts of the map have and haven't been observed.

6. Joint Architecture

Our final architecture, Cognitive Mapping and Planning (CMP) puts together the mapper and planner described above. At each time step, the mapper updates its multi-scale belief about the world based on the current observation. This updated belief is input to the planner which outputs the action to take. As described previously, all parts of the network are differentiable and allow for end-to-end training, and no additional direct supervision is used to train the mapping module – rather than producing maps that match some ground truth free space, the mapper produces maps that allow the planner to choose effective actions.

Training Procedure. We optimize the CMP network with fully supervised training using DAGGER [52]. We generate training trajectories by sampling arbitrary start and goal locations on the graph $\mathcal{G}_{x,\theta}$. We generate supervision for training by computing shortest paths on the graph. We use an online version of DAGGER, where during each episode we sample the next state based on the action from the agent's current policy, or from the expert policy. We use scheduled sampling and anneal the probability of sampling from the expert policy using inverse sigmoid decay.

Note that the focus of this work is on studying different architectures for navigation. Our proposed architecture can also be trained with alternate paradigms for learning such policies, such as reinforcement learning. We chose DAGGER for training our models because we found it to be significantly more sample efficient and stable in our domain, allowing us to focus on the architecture design.

7. Experiments

All our models are trained asynchronously with 16 parallel GPU workers and 16 parameter servers using TensorFlow [3]. We used ADAM [39] to optimize our loss function and trained for 60K iterations with a learning rate of 0.001 which was dropped by a factor of 10 every 20K iterations (we found this necessary for consistent training across different runs). We use weight decay of 0.0001 to regularize the network and use batch-norm [32].

Method	Mean		75^{th} %ile		Success %age	
	RGB	Depth	RGB	Depth	RGB	Depth
Geometric Task						
Initial	25.3	25.3	30	30	0.7	0.7
No Image LSTM	20.8	20.8	28	28	6.2	6.2
Reactive (1 frame)	20.9	17.0	28	26	8.2	21.9
Reactive (4 frames)	14.4	8.8	25	18	31.4	56.9
LSTM	10.3	5.9	21	5	53.0	71.8
Our (CMP)	**7.7**	**4.8**	**14**	**1**	**62.5**	**78.3**
Semantic Task (Aggregate)						
Initial	16.2	16.2	25	25	11.3	11.3
Reactive	14.2	14.2	22	23	23.4	22.3
LSTM	13.5	13.4	20	23	23.5	27.2
Our (CMP)	**11.3**	**11.0**	**18**	**19**	**34.2**	**40.0**

Table 1: Navigation Results: We report the mean distance to goal location, 75^{th} percentile distance to goal and success rate after executing the policy for 39 time steps. The top part presents results for the case where the goal is specified geometrically in terms of position of the goal in the coordinate frame of the robot. The bottom part presents aggregate results for the case where the goal is specified semantically in the form of 'go to a chair' (or door or table).

We use ResNet-50 [28] pre-trained on ImageNet [15] to represent RGB images. We transfer supervision from RGB images to depth images using cross modal distillation [24] between RGB-D image pairs rendered from meshes in the training set to obtain a pre-trained ResNet-50 model to represent depth images.

We compare our proposed CMP architecture to other alternate architectures such as a reactive agent and a LSTM based agent. Since the goal of this paper is to study various architectures for navigation we train all these architectures the same way using DAGGER [52] as described earlier.

Geometric Task. We first present results for the task where the goal is specified geometrically in terms of position of the goal in robot's coordinate frame in Table 1 (top part) and Figure 4. Problems for this task are generated by first sampling a start node on the graph and then sampling an end node which is within 32 steps from the starting node and preferably in another room or in the hallway (we use room and hallway annotations from the dataset [5]). The same sampling process is used during training and testing. We sample 4000 problems for testing and these remain fixed across different algorithms that we compare. We measure performance using the distance to goal after running the learned policy for the episode length (39 time steps). We report multiple error metrics, the mean distance to goal, the 75^{th} percentile distance to goal and the success rate (the agent succeeds if it is within a distance of three steps to the goal location at the end of the episode). Table 1 reports these metrics at end of the episode, while Figure 4 plots them across time steps. We report all numbers on the test set. The test set consists of a floor from an altogether different building not contained in the training set. (See dataset website and Appendix [1] for environment visualizations.)

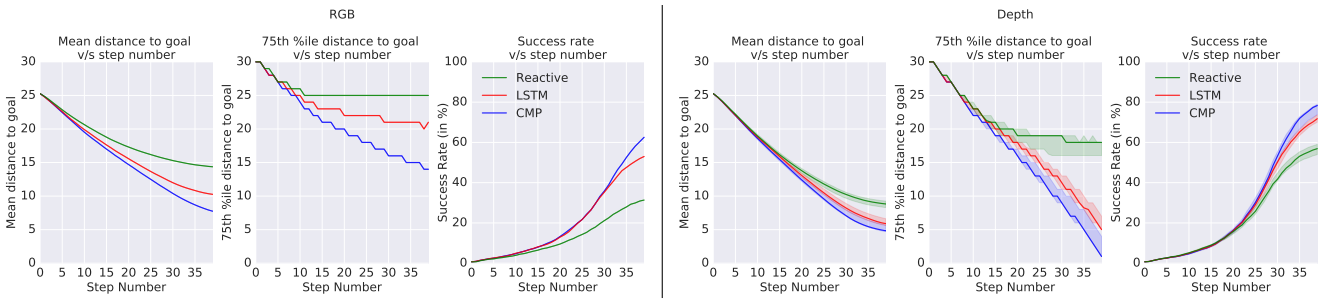

Figure 4: We report the mean distance to goal, 75$^{\text{th}}$ percentile distance to goal (lower is better) and success rate (higher is better) as a function of the number of steps for the 4 frame reactive agent, LSTM based agent and our proposed CMP based agent when using RGB images as input (left three plots) and when using depth images as input (right three plots). We note that CMP outperforms the two baselines in both cases, and generally using depth images as input leads to better performance than using RGB images as input. We also show the variance in performance over five re-trainings from different random initializations of the agents when using depth images as input (the solid line plots the median performance and the surrounding shaded region represents the minimum and maximum value across five different runs). We note that the variation in performance is reasonably small for all models and CMP consistently outperforms the two baseline.

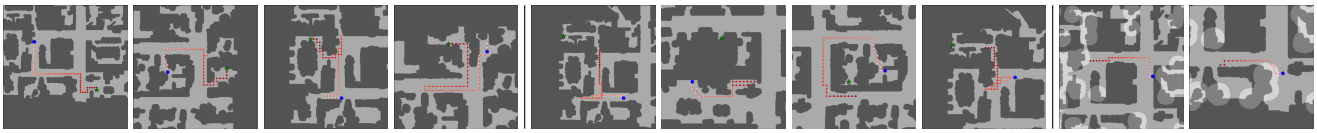

Figure 5: Representative Success and Failure Cases for CMP: We visualize trajectories for some typical success and failure cases for CMP. Dark gray regions show occupied space, light gray regions show free space. The agent starts from the blue dot and is required to reach the green star (or semantic regions shown in light gray). The agent's trajectory is shown by the dotted red line. While we visualize the trajectories in the top view, note that the agent only receives the first person view as input. **Left plots** show success cases for geometric task. We see that the agent is able to traverse large distances across multiple rooms to get to the target location, go around obstacles and quickly resolve that it needs to head to the next room and not the current room. The last two plots show cases where the agent successfully backtracks. **Center plots** show failure cases for geometric task: problems with navigating around tight spaces (entering through a partially opened door, and getting stuck in the corner (the gap is not big enough to pass through)), missing openings which would have lead to shorter paths, thrashing around in space without making progress. **Right plots** visualize trajectories for 'go to the chair' semantic task. The first figure shows a success case, while the right figure shows a typical failure case where the agent walks right through a chair region.

Nearest Neighbor Trajectory Transfer: To quantify similarity between training and testing environments, we transfer optimal trajectories from the train set to the test set using visual nearest neighbors (in RGB ResNet-50 feature space). This transfer is done as follows. At each time step we pick the location in the training set which results in the most similar view to that seen by the agent at the current time step. We then compute the optimal action that conveys the robot to the same relative offset in the training environment from this location and execute this action at the current time step. This procedure is repeated at each time step. Such a transfer leads to very poor results. The mean and median distance to goal is 22 and 25 steps respectively, highlighting the differences between the train and test environments.

No image, goal location only with LSTM: This refers to the experimental setting where we ignore the image and simply use the relative goal location (in robot's current coordinate frame) as input to a LSTM, and predict the action that the agent should take. The relative goal location is embedded into a K dimensional space via fully connected layers with ReLU non-linearities before being input to the LSTM. As expected, this does rather poorly.

Reactive Policy, Single Frame: We next compare to a reactive agent which uses the first-person view of the world. As described above we use ResNet-50 to extract features. These features are passed through a few fully connected layers, and combined with the representation for the relative goal location which is used to predict the final action. We experimented with additive and multiplicative combination strategies and both performed similarly. Note that this reactive baseline is able to perform well on the training environments obtaining a mean distance to goal of about 9 steps, but perform poorly on the test set only being able to get to within 17 steps of the goal on average. This suggests that a reactive agent is able to effectively memorize the environments it was trained on, but fails to generalize to novel environments, this is not surprising given it does not have any form of memory to allow it to map or plan. We also experimented with using Drop Out in the fully connected layers for this model but found that to hurt performance on both the train and the test sets.

Reactive Policy, Multiple Frames: We also consider the case where the reactive policy receives 3 previous frames in addition to the current view. Given the robot's step-size is fairly large we consider a late fusion architecture and fuse the information extracted from ResNet-50. Note that this architecture is similar to the one used in [64]. The primary differences are: goal is specified in terms of relative off-

set (instead of an image), training uses DAGGER (which utilizes denser supervision) instead of A3C, and testing is done in novel environments. These adaptations are necessary to make an interpretable comparison on our task. Using additional frames as input leads to a large improvement in performance, specially when using depth images.

LSTM Based Agent: Finally, we also compare to an agent which uses an LSTM based memory. We introduce LSTM units on the multiplicatively combined image and relative goal location representation. Such an architecture also gives the LSTM access to the egomotion of the agent (via how the relative goal location changes between consecutive steps). Thus this model has access to all the information that our method uses. We also experimented with other LSTM based models (ones without egomotion, inputting the egomotion more explicitly, *etc*.), but weren't able to reliably train them in early experiments and did not pursue them further. This LSTM based model is able to consistently outperform the reactive baseline.

We compare these baselines with of our proposed method. CMP is able to outperform all these baselines across all metrics for both RGB and depth image case. CMP achieves a lower 75^{th} %ile distance to goal (14 and 1 as compared to 21 and 5 for the LSTM) and improves the success rate to 62.5% and 78.3% from 53.0% and 71.8%.

We also report variance in performance over five re-trainings from different random initializations of the network for the 3 most competitive methods (Reactive with 4 frames, LSTM & CMP) for the depth image case. Figure 4 (right) shows the performance, the solid line shows the median metric value and the surrounding shaded region represents the minimum and maximum metric value over the five re-trainings. Variation in performance is reasonably small for all models and CMP leads to significant improvements.

Ablations. We also studied ablated versions of our proposed method. We summarize the key takeaways, a learned mapper leads to better navigation performance than an analytic mapper, planning is crucial (specially for when using RGB images as input) and single-scale planning works slightly better than the multi-scale planning at the cost of increased planning cost. More details in Appendix [1].

Additional comparisons between LSTM and CMP. We also conducted additional experiments to further compare the performance of the LSTM baseline with our model in the most competitive scenario where both methods use depth images. We summarize the key conclusions here and provide more details in Appendix [1]. We studied how performance changes when the target is much further away (64 time steps away). We see a larger gap in performance between LSTM and CMP for this test scenarios. We also compared performance of CMP and LSTM over problems of different difficulty and observed that CMP is generally better across all values of hardness, but for RGB images it is particularly better for cases with high hardness. We also evaluate how well these models generalize when trained on a single scene, and when transferring across datasets. We find that there is a smaller drop in performance for CMP as compared to LSTM. More details in Appendix [1]. Figure 5 visualizes and discusses some representative success and failure cases for CMP, video examples are available on the project website.

Semantic Task. Here we present experiments where the target is specified semantically. We consider three tasks: 'go to a chair', 'go to a door' and 'go to a table'. The agent receives a one-hot vector indicating the object category it must go to and is considered successful if it can reach any instance of the indicated object category. We use object annotations from the S3DIS dataset [5] to label nodes in the graph $\mathcal{G}_{x,\theta}$ with object categories. Note that this is only done to generate supervision for optimal actions during training and to evaluate the performance of the agents at test time. This supervision is not used by the agent in any way, it must learn appearance models for chairs jointly with the policy to reach them. We initialize the agent such that it is within 32 time steps of at least one instance of the indicated category, and train it to go towards the nearest instance. Table 1 (bottom) reports average and 75^{th} %ile distance to nearest category instance and the success rate after executing a fixed number of steps (39 steps) across tasks from all three categories.

We compare our method to the best performing reactive and LSTM based baseline models from the geometric navigation task[1]. This is a challenging task specially because the agent may start in a location from which the desired object is not visible, and it must learn to explore the environment to find the desired object. CMP is able to achieve a higher success rate than the baselines. Figure 5 shows some sample trajectories for this task for CMP. Per category performance and more analysis is presented in Appendix [1]. Performance is currently hindered by the limited ability of the network at recognizing objects and incorporating stronger appearance models may boost performance.

Visualizations. We also analyzed the mapper representation and the value maps learned by our networks. We found the mapper representation to capture free space, and the value maps being indicative of the agent's behavior. More details in Appendix [1].

Acknowledgments: We thank Shubham Tulsiani, David Fouhey and Christian Häne for useful discussion and feedback on the manuscript, as well as Marek Fiser and Sergio Guadarrama for help with the Google infrastructure and Tensorflow.

[1]This LSTM is impoverished because it no longer receives the egomotion of the agent as input (because the goal can not be specified as an offset relative to the robot). We did experiment with a LSTM model which received egomotion as input but weren't able to train it in initial experiments.

References

[1] Appendix. Available on Project Website. 4, 5, 6, 8

[2] Matterport. https://matterport.com/. 4

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 6

[4] D. Abel, A. Agarwal, F. Diaz, A. Krishnamurthy, and R. E. Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint arXiv:1603.04119*, 2016. 3

[5] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 4, 6, 8

[6] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 2013. 2

[7] R. Bellman. A markovian decision process. Technical report, DTIC Document, 1957. 5

[8] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016. 3

[9] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016. 3

[10] S. Brahmbhatt and J. Hays. Deepnav: Learning to navigate large cities. *arXiv preprint arXiv:1701.09135*, 2017. 3

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

[12] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, 2015. 3

[13] S. Daftry, J. A. Bagnell, and M. Hebert. Learning transferable policies for monocular reactive mav control. In *ISER*, 2016. 3

[14] A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. In *ECCV*, 1998. 1

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[16] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016. 3

[17] A. Elfes. Sonar-based real-world mapping and navigation. *RA*, 1987. 2

[18] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 1989. 2

[19] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *IROS*, 2012. 2

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

[21] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *RAL*, 2016. 3

[22] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016. 2

[23] S. Gupta, P. Arbeláez, and J. Malik. Perceptual organization and recognition of indoor scenes from RGB-D images. In *CVPR*, 2013. 2

[24] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer. In *CVPR*, 2016. 6

[25] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *NIPS*, 2016. 4

[26] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 2009. 2

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[28] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 6

[29] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015. 2

[30] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *ISER*, 2010. 2

[31] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997. 5

[32] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 6

[33] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. *UIST*, 2011. 2

[34] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 4

[35] G. Kahn, T. Zhang, S. Levine, and P. Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, 2017. 3

[36] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *RA*, 1996. 2

[37] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IJRR*, 1986. 2

[38] H. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via reinforcement learning. In *NIPS*, 2003. 2

[39] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[40] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *ICRA*, 2004. 2

[41] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. *IJRR*, 2010. 2

[42] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *NIPS*, 2011. 2

[43] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems*, 1991. 2

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

[45] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016. 1

[46] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *ICLR*, 2017. 3

[47] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 2

[48] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 1, 2

[49] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *CVPR*, 2004. 4

[50] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. In *ICML*, 2016. 2, 3

[51] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 2008. 2

[52] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011. 6

[53] F. Sadeghi and S. Levine. $(CAD)^2RL$: Real singel-image flight without a singel real image. *arXiv preprint arXiv:1611.04201*, 2016. 3

[54] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. In *ICML*, 2015. 2

[55] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *IJCV*, 2008. 2

[56] A. Tamar, S. Levine, and P. Abbeel. Value iteration networks. In *NIPS*, 2016. 2, 3, 5

[57] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005. 1, 2

[58] E. C. Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948. 2

[59] M. Toussaint. Learning a world model and planning with a self-organizing, dynamic neural system. In *NIPS*, 2003. 3

[60] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 2010. 2

[61] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese. Generic 3d representation via pose estimation and matching. In *ECCV*, 2016. 2

[62] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*, 2016. 3

[63] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel. Learning deep neural network policies with continuous memory states. In *ICRA*, 2016. 2

[64] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 1, 3, 7