2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Learning Navigation Costs from Demonstration in Partially Observable Environments

Tianyu Wang Vikas Dhiman Nikolay Atanasov

Abstract— **This paper focuses on inverse reinforcement learning (IRL) to enable safe and efficient autonomous navigation in unknown partially observable environments. The objective is to infer a cost function that explains expert-demonstrated navigation behavior while relying only on the observations and state-control trajectory used by the expert. We develop a cost function representation composed of two parts: a probabilistic occupancy encoder, with recurrent dependence on the observation sequence, and a cost encoder, defined over the occupancy features. The representation parameters are optimized by differentiating the error between demonstrated controls and a control policy computed from the cost encoder. Such differentiation is typically computed by dynamic programming through the value function over the whole state space. We observe that this is inefficient in large partially observable environments because most states are unexplored. Instead, we rely on a closed-form subgradient of the cost-to-go obtained only over a subset of promising states via an efficient motion-planning algorithm such as A* or RRT. Our experiments show that our model exceeds the accuracy of baseline IRL algorithms in robot navigation tasks, while substantially improving the efficiency of training and test-time inference.**

I. Introduction

Practical applications of autonomous robot systems increasingly require operation in unstructured, partially observed, unknown, and changing environments. Achieving safe and robust navigation in such conditions is directly coupled with the quality of the environment representation and the cost function specifying desirable robot behavior. Designing a cost function that accurately encodes safety, liveness, and efficiency requirements of navigation tasks is a major challenge. In contrast, it is significantly easier to obtain demonstrations of desirable behavior. The field of inverse reinforcement learning [1]–[3] (IRL) provides numerous tools for learning cost functions from expert demonstration.

We consider the problem of safe navigation from partial observations in unknown environments. We assume that demonstrations containing expert poses, controls, and sensory observations over time are available. The objective is to infer the cost function, which depends on the observation sequence, and explains the demonstrated behavior. This inference can only be done indirectly by comparing the control inputs, that a robot may take based on its current cost representation, to the expert's actions in that situation.

Learning a cost function requires a differentiable control policy with respect to the stage cost parameters. Computing such derivatives has been addressed by several successful approaches [4]–[7]. Ratliff et al. [4] developed algorithms with regret bounds for computing subgradients of planning algorithms (e.g., A* [8], RRT [9], [10], etc.) with respect to the cost features. Ziebart et al. [5] developed a dynamic programming algorithm for computing the expected state visitation frequency of a policy extracted from expert demonstrations to enforce that it earns the same reward as the demonstrated policy. Tamar et al. [6] showed that the value iteration algorithm could be approximated using a series of convolution (computing value function expectation over stochastic transitions) and maxpooling (choosing the best action) allowing automatic differentiation. Okada et al. [7] proposed path integral networks in which the control sequence resulting from path integral control may be differentiated with respect to the controller parameters. All of these works, however, assume a known environment and only optimize the parameters of a cost function defined over it.

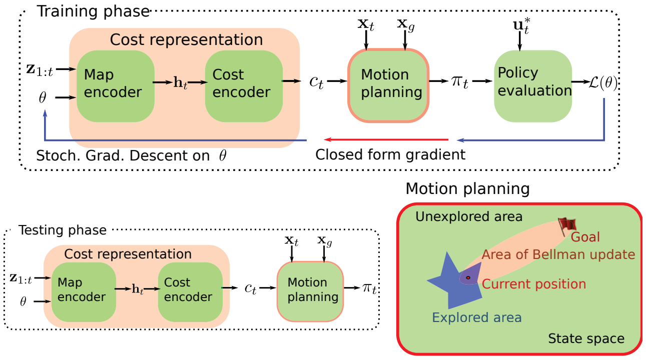


Fig. 1: Architecture for learning cost function representations from demonstrations. Our main contribution lies in a non-stationary cost representation, combining a probabilistic occupancy *map encoder*, with recurrent dependence on observations $\mathbf{z}_{1:t}$, and a *cost encoder*, defined over the occupancy features. We also perform efficient (in the size of the state-space) forward non-stationary policy computation and efficient (closed-form subgradient) backpropagation. The bottom-right plot illustrates that motion planning may be used to update and differentiate cost-to-go estimates only in promising areas of the state space rather than using full Bellman backups, which is particularly redundant in partially observable environments.

A series of recent works [11]–[16] address IRL under partial observability. Wulfmeier et al. [13] train deep neural network representations for Ziebart et al.'s Maximum Entropy IRL [5] method and consider streaming lidar scan observations of the environment. Karkus et al. [16] formulate the IRL problem as a POMDP [17], including the robot pose and environment map in the state. Since a naïve representation of the occupancy distribution may require exponential memory in the map size, the authors assume partial knowledge of the environment (the structure of a building is known but the furniture placement is not). A control policy is obtained via the SARSOP [18] algorithm,

We gratefully acknowledge support from NSF CRII IIS-1755568, ARL DCIST CRA W911NF-17-2-0181, and ONR SAI N00014-18-1-2828.

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093, USA {tiw161,vdhiman,natanasov}@eng.ucsd.edu

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

↓Reference \Feature →	POE	PBB	DNN	CfG
VIN [6]	✗	✗	✓	✗
CMP [14]	✓	✗	✓	✗
MaxEntIRL [5]	✗	✗	✗	✓
MaxMarginIRL [4]	✗	✓	✗	✓
DAN [16]	✓†	✗	✓	✗
Ours	✓	✓	✓	✓

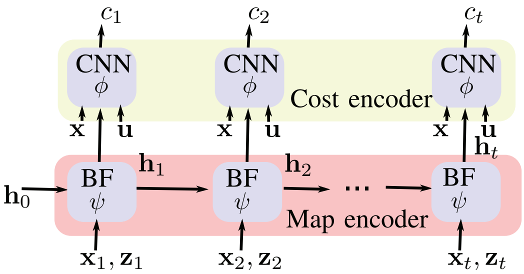
TABLE I: Comparison with closely related work based on the use of Partially Observable Environments (POE), Partial Bellman Backups (PBB), Deep Neural Network (DNN) representation, and Closed-form Gradients (CfG). PBB refers to computing and differentiating values over a subset of promising states as opposed to the whole state space. † DAN [16] works with uncertainty only on the robot or furniture location, while the main environment structure is known.

which approximates the cost-to-go function only over an optimally reachable space. Gupta et al. [14] address visual navigation in partially observed environments while using hierarchical VIN as the planner. Khan et al. [19] introduce a memory module to VIN to address partial observability.

Many IRL algorithms rely on dynamic programming, including VIN and derivatives [14], [19], which requires updating cost-to-go estimates over *all* possible states. Our insight is that in partially observable environments, the cost-to-go estimates need to be updated and differentiated only over a *subset* of states. Inspired by [4], we obtain cost-to-go estimates only over promising states using a motion planning algorithm. This helps to obtain a closed-form subgradient of the cost-to-go with respect to the learned cost function from the optimal trajectory. While Ratiff et al. [4] exploit this observation in fully observable environments, none of the works focusing on partial observability take advantage of this. Our work differs from closely related works in Table I. In summary, we offer two **contributions** illustrated in Fig. 1: Firstly, we develop a non-stationary cost function representation composed of a probabilistic occupancy *map encoder*, with recurrent dependence on the observation sequence, and a *cost encoder*, defined over the occupancy features (Sec. III). Secondly, we optimize the cost parameters using a closed-form subgradient of the cost-to-go obtained only over a subset of promising states (Sec. IV).

II. Problem Formulation

Consider a robot navigating in an unknown environment with the task of reaching a goal state $\mathbf{x}_g \in \mathcal{X}$. Let $\mathbf{x}_t \in \mathcal{X}$ be the robot state, capturing its pose, twist, etc., at discrete time t. For a given control input $\mathbf{u}_t \in \mathcal{U}$ where $\mathcal{U}$ is assumed finite, the robot state evolves according to known deterministic dynamics: $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$. Let $m^* : \mathcal{X} \to \{-1, 1\}$ be a function specifying the *true* occupancy of the environment by labeling states as either feasible (-1) or infeasible (1) and let $\mathcal{M}$ be the space of possible environment realizations m^*. Let $c^* : \mathcal{X} \times \mathcal{U} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$ be a cost function specifying desirable robot behavior in a given environment, e.g., according to an expert user or an optimal design. We assume that the robot does not have access to either the true occupancy map m^* or the true cost function c^*. However, the robot is able to make observations $\mathbf{z}_t \in \mathcal{Z}$ (e.g., using a lidar scanner or a depth camera) of the environment in its vicinity, whose distribution depends on the robot state $\mathbf{x}_t$ and the environment m^*. Given a training set $\mathcal{D} := \left\{(\mathbf{x}_{t,n}, \mathbf{u}^*_{t,n}, \mathbf{z}_{t,n}, \mathbf{x}_{g,n})\right\}_{t=1,n=1}^{T_n,N}$ of N expert trajectories with length T_n to demonstrate desirable behavior, our goal is to



c_1
c_2
c_t
CNN ϕ
CNN ϕ
CNN ϕ
Cost encoder
$\mathbf{x}$ $\mathbf{u}$
$\mathbf{x}$ $\mathbf{u}$
$\mathbf{x}$ $\mathbf{u}$
$\mathbf{h}_t$
$\mathbf{h}_0$
BF ψ
$\mathbf{h}_1$
BF ψ
$\mathbf{h}_2$
...
BF ψ
Map encoder
$\mathbf{x}_1, \mathbf{z}_1$
$\mathbf{x}_2, \mathbf{z}_2$
$\mathbf{x}_t, \mathbf{z}_t$

Fig. 2: Neural network model of a cost function representation. A Bayes filter with likelihood function parameterized by $\boldsymbol{\psi}$, takes in sequential observations $\mathbf{z}_{1:t}$ and outputs a latent map representation $\mathbf{h}_t$. A convolutional neural network, parameterized by $\boldsymbol{\phi}$, extracts features from the map state to specify the cost c_t at a given robot state-control pair $(\mathbf{x}, \mathbf{u})$. The learnable parameters are $\boldsymbol{\theta} = \{\boldsymbol{\psi}, \boldsymbol{\phi}\}$.

- learn a cost function estimate $c_t : \mathcal{X} \times \mathcal{U} \times \mathcal{Z}^t \times \Theta \to \mathbb{R}_{\geq 0}$ that depends on an observation sequence $\mathbf{z}_{1:t}$ from the true latent environment and is parameterized by $\boldsymbol{\theta}$,
- derive a stochastic policy π_t from c_t such that the robot behavior under π_t matches the prior experience $\mathcal{D}$.

To balance exploration in partially observable environments with exploitation of promising controls, we specify π_t as a Boltzmann policy [3], [20] associated with the cost c_t:

$$\pi_t(\mathbf{u}_t|\mathbf{x}_t; \mathbf{z}_{1:t}, \boldsymbol{\theta}) = \frac{\exp(-Q^*_t(\mathbf{x}_t, \mathbf{u}_t; \mathbf{z}_{1:t}, \boldsymbol{\theta}))}{\sum_{\mathbf{u}\in\mathcal{U}} \exp(-Q^*_t(\mathbf{x}_t, \mathbf{u}; \mathbf{z}_{1:t}, \boldsymbol{\theta}))}, \quad (1)$$

where the optimal cost-to-go function Q^*_t is:

$$Q^*_t(\mathbf{x}_t, \mathbf{u}_t; \mathbf{z}_{1:t}, \boldsymbol{\theta}) := \min_{\mathbf{u}_{t+1:T-1}} \sum_{k=t}^{T-1} c_t(\mathbf{x}_k, \mathbf{u}_k; \mathbf{z}_{1:t}, \boldsymbol{\theta}) \quad (2)$$
$$\text{s.t.} \quad \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k),\ \mathbf{x}_T = \mathbf{x}_g.$$

Problem. Given demonstrations $\mathcal{D}$, optimize the cost function parameters $\boldsymbol{\theta}$ so that log-likelihood of the demonstrated controls $\mathbf{u}^*_{t,n}$ is maximized under the robot policies $\pi_{t,n}$:

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) := -\sum_{n=1}^{N} \sum_{t=1}^{T_n} \log \pi_{t,n}(\mathbf{u}^*_{t,n}|\mathbf{x}_{t,n}; \mathbf{z}_{1:t}, \boldsymbol{\theta}). \quad (3)$$

The problem setup is illustrated in Fig. 1. While Eqn. (2) is a standard deterministic shortest path (DSP) problem, the challenge is to make it differentiable with respect to $\boldsymbol{\theta}$. This is needed to propagate the loss in (3) back through the DSP problem to update the cost parameters $\boldsymbol{\theta}$. Once the parameters are optimized, the robot can generalize to navigation tasks in new partially observable environments by evaluating the cost c_t based on the observations $\mathbf{z}_{1:t}$ iteratively and (re)computing the associated policy π_t.

III. Cost Function Representation

We propose a cost function representation comprised of two components: a *map encoder* and a *cost encoder*.

The map encoder incrementally updates a hidden state $\mathbf{h}_t$ using the most recent observation $\mathbf{z}_t$ obtained from robot state $\mathbf{x}_t$. For example, a Bayes filter with likelihood function parameterized by $\boldsymbol{\psi}$ can convert the sequential input $(\mathbf{x}_{1:t}, \mathbf{z}_{1:t})$ into a fixed-sized hidden state $\mathbf{h}_{t+1}$:

$$\mathbf{h}_{t+1} = \mathbf{BF}(\mathbf{h}_t, \mathbf{x}_t, \mathbf{z}_t; \boldsymbol{\psi}). \tag{4}$$

The cost encoder uses the latent environment map $\mathbf{h}_t$ to define the cost function estimate $c_t(\mathbf{x}, \mathbf{u})$ at a given state-control pair $(\mathbf{x}, \mathbf{u})$. A convolutional neural network (CNN) [21] with parameters $\boldsymbol{\phi}$ can extract cost features from the environment map:

$$c_t(\mathbf{x}, \mathbf{u}) = \mathbf{CNN}(\mathbf{h}_t, \mathbf{x}, \mathbf{u}; \boldsymbol{\phi}). \tag{5}$$

This conceptual model, combining recurrent estimation of a hidden environment state, followed by feature extraction to define the cost at $(\mathbf{x}, \mathbf{u})$ is illustrated in Fig. 2. The model is differentiable by design, allowing its parameters $\boldsymbol{\theta} = \{\boldsymbol{\psi}, \boldsymbol{\phi}\}$ to be optimized. We propose an instantiation of this general model, specific to modeling occupancy costs from depth measurements in robot navigation tasks.

A. Map Encoder

We encode the occupancy probability of different environment areas into a hidden state $\mathbf{h}_t$. In detail, we discretize $\mathcal{X}$ into N cells and let $\mathbf{m}^* \in \{-1, 1\}^N$ be the vector of true occupancy values over the cells. Since $\mathbf{m}^*$ is unknown to the robot, we maintain the occupancy likelihood $\mathbb{P}(\mathbf{m}^* = \mathbf{1} | \mathbf{x}_{1:t}, \mathbf{z}_{1:t})$ given the history of states $\mathbf{x}_{1:t}$ and observations $\mathbf{z}_{1:t}$. See Fig. 3 for an example of a depth measurement $\mathbf{z}_t$ and associated occupancy likelihood over the map $\mathbf{m}^*$. The representation complexity may be simplified significantly if one assumes independence among the map cells $\mathbf{m}^*_j$:

$$\mathbb{P}(\mathbf{m}^* = \mathbf{1} | \mathbf{x}_{1:t}, \mathbf{z}_{1:t}) = \prod_{j=1}^{N} \mathbb{P}(\mathbf{m}^*_j = 1 | \mathbf{x}_{1:t}, \mathbf{z}_{1:t}) . \tag{6}$$

We use inspiration from occupancy grid mapping [22], [23] to design recurrent updates for the occupancy probability of each cell $\mathbf{m}^*_j$. Since $\mathbf{m}^*_j$ is binary, its likelihood update can be simplified by defining the log-odds ratio of occupancy:

$$\mathbf{h}_{t,j} := \log \frac{\mathbb{P}(\mathbf{m}^*_j = 1 | \mathbf{x}_{1:t}, \mathbf{z}_{1:t})}{\mathbb{P}(\mathbf{m}^*_j = -1 | \mathbf{x}_{1:t}, \mathbf{z}_{1:t})} . \tag{7}$$

The recurrent Bayesian update of $\mathbf{h}_{t,j}$ is:

$$\mathbf{h}_{t+1,j} = \mathbf{h}_{t,j} + \log \frac{p(\mathbf{z}_{t+1} | \mathbf{m}^*_j = 1, \mathbf{x}_{t+1})}{p(\mathbf{z}_{t+1} | \mathbf{m}^*_j = -1, \mathbf{x}_{t+1})} , \tag{8}$$

where the increment is a log-odds observation model. The occupancy posterior can be recovered from the occupancy log-odds ratio $\mathbf{h}_{t,j}$ via a sigmoid function:

$$\mathbb{P}(\mathbf{m}^*_j = 1 | \mathbf{x}_{1:t}, \mathbf{z}_{1:t}) = \sigma\left(\mathbf{h}_{t,j}\right). \tag{9}$$

The sigmoid function satisfies the following properties:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \;\; 1 - \sigma(x) = \sigma(-x), \;\; \log \frac{\sigma(x)}{\sigma(-x)} = x \tag{10}$$

To complete the recurrent occupancy model design, we parameterize the log-odds observation model:

$$\log \frac{p(\mathbf{z} | \mathbf{m}^*_j = 1, \mathbf{x})}{p(\mathbf{z} | \mathbf{m}^*_j = -1, \mathbf{x})} = \underbrace{\log \frac{\mathbb{P}(\mathbf{m}^*_j = 1 | \mathbf{z}, \mathbf{x})}{\mathbb{P}(\mathbf{m}^*_j = -1 | \mathbf{z}, \mathbf{x})}}_{\mathbf{g}_j(\mathbf{x}, \mathbf{z})} - \mathbf{h}_{0,j} \tag{11}$$

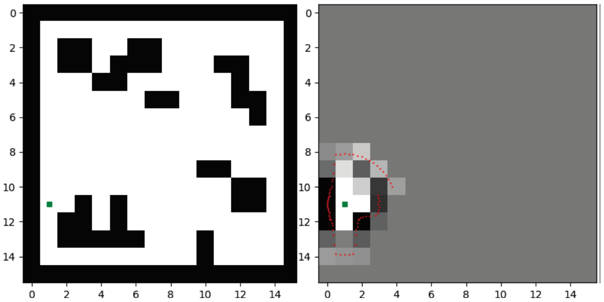


Fig. 3: A robot (green) navigating in a 2-D grid environment. The true environment (left) is a 16×16 grid where black regions are obstacles and white regions are free. On the right, the noisy lidar beams with endpoints in red have maximum range 2.5. The robot estimates the occupancy probability through an inverse observation model in Eqn (14) (darker means higher probability of occupancy).

where Bayes rule was used to represent it in terms of an inverse observation model $\mathbf{g}_j(\mathbf{x}, \mathbf{z})$ and a prior occupancy log-odds ratio $\mathbf{h}_{0,j}$, whose value may depend on the environment (e.g., $\mathbf{h}_{0,j} = 0$ specifies a uniform prior over occupied and free cells). Note that map cells $\mathbf{m}^*_j$ outside of the sensor field of view at time t are not affected by $\mathbf{z}_t$, in which case $\mathbf{g}_j(\mathbf{x}_t, \mathbf{z}_t) = \mathbf{h}_{0,j}$. Now, consider a cell $\mathbf{m}^*_j$ along the direction of the k-th sensor ray, whose depth measurement is $\mathbf{z}_{t,k}$. Let $d(\mathbf{x}_t, \mathbf{m}^*_j)$ be the distance between the robot position and the center of mass of cell $\mathbf{m}^*_j$. We model the occupancy likelihood of $\mathbf{m}^*_j$ as a truncated sigmoid around the true distance $d(\mathbf{x}_t, \mathbf{m}^*_j)$:

$$\mathbb{P}(\mathbf{m}^*_j = 1 | \mathbf{x}_t, \mathbf{z}_{t,k}) = \begin{cases} \sigma(\boldsymbol{\psi}_k \delta \mathbf{z}_{t,k}) & \text{if } \delta \mathbf{z}_{t,k} \leq \epsilon \\ \sigma(\mathbf{h}_{0,j}) & \text{if } \delta \mathbf{z}_{t,k} > \epsilon \end{cases} , \tag{12}$$

where $\boldsymbol{\psi}_k$ is a learnable parameter, $\delta \mathbf{z}_{t,k} := d(\mathbf{x}_t, \mathbf{m}^*_j) - \mathbf{z}_{t,k}$, and ϵ is a distance threshold on the influence of the k-th sensor ray on the occupancy of cells around the point of reflection. Using (10), this implies the following log-odds inverse sensor model for cells $\mathbf{m}^*_j$ along the k-th sensor ray:

$$\mathbf{g}_j(\mathbf{x}_t, \mathbf{z}_t) = \begin{cases} \boldsymbol{\psi}_k \delta \mathbf{z}_{t,k} & \text{if } \delta \mathbf{z}_{t,k} \leq \epsilon \\ \mathbf{h}_{0,j} & \text{otherwise} \end{cases} , \tag{13}$$

This model suggests that one may also use a more expressive multi-layer neural network in place of the linear transformation $\boldsymbol{\psi}_k \delta \mathbf{z}_{t,k}$ of the distance differential along the k-th ray:

$$\mathbf{g}_j(\mathbf{x}_t, \mathbf{z}_t) = \begin{cases} \mathbf{NN}(\mathbf{z}_{t,k}, d(\mathbf{x}_t, \mathbf{m}^*_j); \boldsymbol{\psi}_k) & \text{if } \delta \mathbf{z}_{t,k} \leq \epsilon \\ \mathbf{h}_{0,j} & \text{otherwise} \end{cases} \tag{14}$$

In summary, the map encoder starts with prior occupancy log-odds $\mathbf{h}_0$, updates them recurrently via:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{g}(\mathbf{x}_t, \mathbf{z}_t; \boldsymbol{\psi}) - \mathbf{h}_0, \tag{15}$$

where the log-odds inverse sensor model $\mathbf{g}_j(\mathbf{x}_t, \mathbf{z}_t; \boldsymbol{\psi}_k)$ is specified for the j-th cell along the k-th ray in (14), and provides the cell occupancy likelihood in (9) as output.

B. Cost Encoder

Given a state-control pair $(\mathbf{x}, \mathbf{u})$, the cost encoder uses the output $\sigma\left(\mathbf{h}_t\right)$ of the map encoder to obtain a cost function estimate $c_t(\mathbf{x}, \mathbf{u})$. A complex interacting structure over the feature representation $\sigma(\mathbf{h}_t)$ can be obtained via a deep neural network with parameters $\boldsymbol{\phi}$. Wulfmeier et al. [13] proposed several CNN architectures, including pooling and residual connections of fully convolutional networks, to model $c_t(\mathbf{x},\mathbf{u})$ from $\sigma(\mathbf{h}_t)$. While complex architecture may provide better performance in practice, we also develop a simpler model for comparison using the inductive bias of obstacle avoidance in robot navigation.

Let s and l be a small and large positive parameter, respectively. The cost of applying control $\mathbf{u}$ in robot state $\mathbf{x}$ can be modeled as large when the transition $f(\mathbf{x},\mathbf{u})$ encounters an obstacle and as small otherwise:

$$c(\mathbf{x},\mathbf{u}) := \begin{cases} s & \text{if } m^*(\mathbf{x}) = -1 \text{ and } m^*(f(\mathbf{x},\mathbf{u})) = -1 \\ l & \text{if } m^*(\mathbf{x}) = 1 \text{ or } m^*(f(\mathbf{x},\mathbf{u})) = 1 \end{cases}$$

Since m^* is unknown, we use the estimated occupancy probabilities $\sigma(\mathbf{h}_t)$ to compute the expectation of $c(\mathbf{x},\mathbf{u})$ over m^*:

$$\begin{aligned} c_t(\mathbf{x},\mathbf{u}) := \mathbb{E}[c(\mathbf{x},\mathbf{u})] = s\,\sigma(\mathbf{h}_t[\mathbf{x}])\,\sigma(\mathbf{h}_t[f(\mathbf{x},\mathbf{u})]) \\ + l\left(1 - \sigma(\mathbf{h}_t[\mathbf{x}])\,\sigma(\mathbf{h}_t[f(\mathbf{x},\mathbf{u})])\right), \end{aligned} \tag{16}$$

where $\mathbf{h}_t[\mathbf{x}]$ is the entry of the map encoder state that corresponds to the environment cell containing $\mathbf{x}$. This simple cost encoder has parameters $\boldsymbol{\phi} := [s,\ l]^T$ and its output $c_t(\mathbf{x},\mathbf{u})$ is differentiable with respect to $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ through $\sigma(\mathbf{h}_t)$.

IV. Cost Learning via Differentiable Planning

We focus on optimizing the parameters $\boldsymbol{\theta}$ of the cost representation $c_t(\mathbf{x},\mathbf{u};\mathbf{z}_{1:t},\boldsymbol{\theta})$ developed in Sec. III. Since the true cost c^* is not directly observable, we need to differentiate the loss function $\mathcal{L}(\boldsymbol{\theta})$ in (3), which, in turn, requires differentiating through the DSP problem in (2) with respect to the cost function estimate c_t.

Value Iteration Networks (VIN) [6] shows that T iterations of the value iteration algorithm can be approximated by a neural network with T convolutional and minpooling layers. This allows VIN to be differentiable with respect to the stage cost. While VIN can be modified to operate with a finite horizon and produce a non-stationary policy, it would still be based on full Bellman backups (convolutions and minpooling) over the entire state space. As a result, VIN scales poorly with the state-space size, while it might not even be necessary to determine the optimal cost-to-go $Q_t^*(\mathbf{x},\mathbf{u})$ at every state $\mathbf{x} \in \mathcal{X}$ and control $\mathbf{u} \in \mathcal{U}$ in the case of partially observable environments.

Instead of using dynamic programming to solve the DSP (2), any motion planning algorithm (e.g., A* [8], RRT [9], [10], etc.) that returns the optimal cost-to-go $Q_t^*(\mathbf{x},\mathbf{u})$ over a subset of the state-control space provides an accurate enough solution. For example, a backwards A* search applied to problem (2) with start state $\mathbf{x}_g$, goal state $\mathbf{x} \in \mathcal{X}$, and predecessors expansions according to the motion model f provides an upper bound to the optimal cost-to-go:

$$\begin{aligned} Q_t^*(\mathbf{x},\mathbf{u}) &= c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u})) \quad && \forall f(\mathbf{x},\mathbf{u}) \in \text{CLOSED}, \\ Q_t^*(\mathbf{x},\mathbf{u}) &\leq c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u})) \quad && \forall f(\mathbf{x},\mathbf{u}) \notin \text{CLOSED}, \end{aligned}$$

where g are the values computed by A* for expanded nodes in the CLOSED list and visited nodes in the OPEN list. Thus, a Boltzmann policy $\pi_t(\mathbf{u} \mid \mathbf{x})$ can be defined using the g-values returned by A* for all $\mathbf{x} \in \text{CLOSED} \cup \text{OPEN} \subseteq \mathcal{X}$ and a uniform distribution over $\mathcal{U}$ for all other states $\mathbf{x}$. A* would significantly improves the efficiency of VIN [6] or other full backup Dynamic Programming variants by performing local Bellman backups on promising states (the CLOSED list).

In addition to improving the efficiency of the forward computation of $Q_t^*(\mathbf{x},\mathbf{u})$, using a planning algorithm to solve (2) is also more efficient in back-propagating errors with respect to $\boldsymbol{\theta}$. In detail, using the subgradient method [4], [24] to optimize $\mathcal{L}(\boldsymbol{\theta})$ leads to a closed-form (sub)gradient of $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$ with respect to $c_t(\mathbf{x},\mathbf{u})$, removing the need for back-propagation through multiple convolutional or minpooling layers. We proceed by rewriting $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$ in a form that makes its subgradient with respect to $c_t(\mathbf{x},\mathbf{u})$ obvious. Let $\mathcal{T}(\mathbf{x}_t,\mathbf{u}_t)$ be the set of feasible state-control trajectories $\boldsymbol{\tau} := \mathbf{x}_t,\mathbf{u}_t,\mathbf{x}_{t+1},\mathbf{u}_{t+1},\ldots,\mathbf{x}_{T-1},\mathbf{u}_{T-1}$ starting at $\mathbf{x}_t$, $\mathbf{u}_t$ and satisfying $\mathbf{x}_{k+1} = f(\mathbf{x}_k,\mathbf{u}_k)$ for $k = t,\ldots,T-1$ with $\mathbf{x}_g = \mathbf{x}_T$. Let $\boldsymbol{\tau}^* \in \mathcal{T}(\mathbf{x}_t,\mathbf{u}_t)$ be an optimal trajectory corresponding to the optimal cost-to-go function $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$ of a deterministic shortest path problem, i.e., the controls in $\boldsymbol{\tau}^*$ satisfy the additional constraint $\mathbf{u}_k = \arg\min_{\mathbf{u}\in\mathcal{U}} Q_t^*(\mathbf{x}_k,\mathbf{u})$ for $k = t+1,\ldots,T-1$. Let $\mu_{\boldsymbol{\tau}}(\mathbf{x},\mathbf{u}) := \sum_{k=t}^{T-1} \mathbb{1}_{(\mathbf{x}_k,\mathbf{u}_k)=(\mathbf{x},\mathbf{u})}$ be a state-control visitation function indicating if $(\mathbf{x},\mathbf{u})$ is visited by $\boldsymbol{\tau}$. With these definitions, we can view the optimal cost-to-go function $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$ as minimum over $\mathcal{T}(\mathbf{x}_t,\mathbf{u}_t)$ of the inner product of the cost function c_t and the visitation function $\mu_{\boldsymbol{\tau}}$:

$$Q_t^*(\mathbf{x}_t,\mathbf{u}_t) = \min_{\boldsymbol{\tau}\in\mathcal{T}(\mathbf{x}_t,\mathbf{u}_t)} \sum_{\mathbf{x}\in\mathcal{X},\mathbf{u}\in\mathcal{U}} c_t(\mathbf{x},\mathbf{u})\mu_{\boldsymbol{\tau}}(\mathbf{x},\mathbf{u}) \tag{17}$$

where $\mathcal{X}$ can be assumed finite because both T and $\mathcal{U}$ are finite. This form allows us to (sub)differentiate $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$ with respect to $c_t(\mathbf{x},\mathbf{u})$ for any $\mathbf{x} \in \mathcal{X}$, $\mathbf{u} \in \mathcal{U}$.

Lemma 1. *Let $f(\mathbf{x},\mathbf{y})$ be differentiable and convex in $\mathbf{x}$. Then, $\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{y}^*)$, where $\mathbf{y}^* := \arg\min_{\mathbf{y}} f(\mathbf{x},\mathbf{y})$, is a subgradient of the piecewise-differentiable convex function $g(\mathbf{x}) := \min_{\mathbf{y}} f(\mathbf{x},\mathbf{y})$.*

Applying Lemma 1 to (17) leads to the following subgradient of the optimal cost-to-go function:

$$\frac{\partial Q_t^*(\mathbf{x}_t,\mathbf{u}_t)}{\partial c_t(\mathbf{x},\mathbf{u})} = \mu_{\boldsymbol{\tau}^*}(\mathbf{x},\mathbf{u}), \tag{18}$$

which can be obtained from the optimal trajectory $\boldsymbol{\tau}^*$ corresponding to $Q_t^*(\mathbf{x}_t,\mathbf{u}_t)$. This result and the chain rule allow us to obtain the complete (sub)gradient of $\mathcal{L}(\boldsymbol{\theta})$.

Proposition 1. *A subgradient of the loss function $\mathcal{L}(\boldsymbol{\theta})$ in* (3) *with respect to the cost function parameters $\boldsymbol{\theta}$ can be obtained as follows:*

$$\begin{aligned} \frac{d\mathcal{L}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} &= -\sum_{n=1}^{N}\sum_{t=1}^{T_n} \frac{d\log\pi_{t,n}(\mathbf{u}_{t,n}^* \mid \mathbf{x}_{t,n})}{d\boldsymbol{\theta}} \\ &= -\sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{\mathbf{u}_{t,n}\in\mathcal{U}} \frac{\partial\log\pi_{t,n}(\mathbf{u}_{t,n}^* \mid \mathbf{x}_{t,n})}{\partial Q_{t,n}^*(\mathbf{x}_{t,n},\mathbf{u}_{t,n})}\frac{dQ_{t,n}^*(\mathbf{x}_{t,n},\mathbf{u}_{t,n})}{d\boldsymbol{\theta}} \end{aligned} \tag{19}$$

Algorithm 1 Training: learn cost function parameters $\boldsymbol{\theta}$

```
**Input:** Demonstrations $\mathcal{D} = \left\{(\mathbf{x}_{t,n}, \mathbf{u}^*_{t,n}, \mathbf{z}_{t,n}, \mathbf{x}_{g,n})\right\}_{t=1,n=1}^{T_n,N}$
**while** $\boldsymbol{\theta}$ not converged **do**
    $\mathcal{L}(\boldsymbol{\theta}) \leftarrow 0$
    **for** $n = 1, \ldots, N$ **and** $t = 1, \ldots, T_n$ **do**
        Update $c_{t,n}$ based on $\mathbf{x}_{t,n}$ and $\mathbf{z}_{t,n}$ as in Sec. III
        Obtain $Q^*_{t,n}(\mathbf{x}, \mathbf{u})$ from DSP (2) with stage cost $c_{t,n}$
        Obtain $\pi_{t,n}(\mathbf{u}|\mathbf{x}_{t,n})$ from $Q^*_{t,n}(\mathbf{x}_{t,n}, \mathbf{u})$ via Eq. (1)
        $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) - \log \pi_{t,n}(\mathbf{u}^*_{t,n}|\mathbf{x}_{t,n})$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta})$ via Prop. 1
**Output:** $\boldsymbol{\theta}$
```

Algorithm 2 Testing: compute control policy for learned $\boldsymbol{\theta}$

```
**Input:** Start state $\mathbf{x}_s$, goal state $\mathbf{x}_g$, optimized $\boldsymbol{\theta}$
Current state $\mathbf{x}_t \leftarrow \mathbf{x}_s$
**while** $\mathbf{x}_t \neq \mathbf{x}_g$ **do**
    Make an observation $\mathbf{z}_t$
    Update $c_t$ based on $\mathbf{x}_t$ and $\mathbf{z}_t$ as in Sec. III
    Obtain $Q^*_t(\mathbf{x}_t, \mathbf{u})$, $\mathbf{u} \in \mathcal{U}$ from DSP (2) with stage cost $c_t$
    Obtain $\pi_t(\mathbf{u}|\mathbf{x}_t)$ from $Q^*_t(\mathbf{x}_t, \mathbf{u})$ via Eq. (1)
    Update $\mathbf{x}_t \leftarrow f(\mathbf{x}_t, \mathbf{u}_t)$ via $\mathbf{u}_t := \arg\max_{\mathbf{u}} \pi_t(\mathbf{u}|\mathbf{x}_t)$
**Output:** Navigation succeeds or fails.
```

where the first term has a closed-form, while the second term is available from (18) *and the cost representation in Sec. III:*

$$\frac{\partial \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n})}{\partial Q^*_{t,n}(\mathbf{x}_{t,n}, \mathbf{u}_{t,n})} = \left(\mathbb{1}_{\{\mathbf{u}_{t,n} = \mathbf{u}^*_{t,n}\}} - \pi_{t,n}(\mathbf{u}_{t,n}|\mathbf{x}_{t,n})\right)$$

$$\frac{dQ^*_{t,n}(\mathbf{x}_{t,n}, \mathbf{u}_{t,n})}{d\boldsymbol{\theta}} = \sum_{(\mathbf{x},\mathbf{u}) \in \boldsymbol{\tau}^*} \frac{\partial Q^*_{t,n}(\mathbf{x}_{t,n}, \mathbf{u}_{t,n})}{\partial c_t(\mathbf{x}, \mathbf{u})} \frac{\partial c_t(\mathbf{x}, \mathbf{u})}{\partial \boldsymbol{\theta}} \quad (20)$$

The computation graph structure implied by Prop. 1 is illustrated in Fig. 1. The graph consists of a cost representation layer and a differentiable planning layer, allowing end-to-end minimization of $\mathcal{L}(\boldsymbol{\theta})$ via stochastic (sub)gradient descent. Full algorithms for the training and testing phases (Fig. 1) are shown in Alg. 1 and Alg. 2.

Although the form of the gradient in Prop. 1 is similar to that in Ziebart et al. [5], our contributions are orthogonal. Our contribution is to obtain a *non-stationary* cost-to-go function and its (sub)gradient for a finite-horizon problem using forward and backward computations that scale efficiently with the size of the state space. On the other hand, the results of Ziebart et al. [5] provide a *stationary* cost-to-go function and its gradient for an infinite horizon problem. The maximum entropy formulation of the stationary policy is a well grounded property of using a soft version of the Bellman update, which can explicitly model the suboptimality of expert trajectories. However, we can show the benefits of our approach without including the maximum entropy formulation and will leave it as future work.

V. Experiments

We evaluate our approach in 2D grid world navigation tasks at two scales. Obstacle configurations are generated randomly in maps $\mathbf{m}^*_n$ of sizes 16×16 or 100×100. We use an 8-connected grid so that a control $\mathbf{u}_t$ causes a transition $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ from $\mathbf{x}_t$ to one of the eight neighbor cells $\mathbf{x}_{t+1}$. At each step, the robot receives a $360°$ lidar scan $\mathbf{z}_t$ at $5°$ resolution, resulting in 72 beams $\mathbf{z}_{t,k}$ in each scan (see Fig. 3). The lidar range readings are corrupted by an additive zero mean Gaussian noise. The standard deviation of the noise is 0.05 and 0.2 (grid cell = 1) and the lidar maximum range is 2.5 and 10 in 16×16 and 100×100 domains, respectively. Note that the lidar range is smaller than the map size to demonstrate environment partial observability. During test time, the domain size is the maximum size allowed for the observed area along a trajectory. Demonstrations are obtained by running an A* planning algorithm to solve the deterministic shortest path problem on the true map $\mathbf{m}^*_n$. The number of maps and training samples generated are shown in Table II.

	16×16		100×100	
Dataset	#maps	#samples	#maps	#samples
Train	7638	514k	970	460k
Validation	966	66k	122	58k
Test	952	-	122	-

TABLE II: Dataset size. We sample 10 trajectories in each map in training and validation, and each sample takes the form $(\mathbf{x}_{1:t,n}, \mathbf{u}^*_{t,n}, \mathbf{z}_{1:t,n}, \mathbf{x}_{g,n})$. In testing, the robot's task is to navigate from one randomly sampled start to goal location on each map.

A. Baseline and model variations

We use DeepMaxEnt [13] as a baseline and compare it to three variants of our model. In all variants, we parameterize an inverse observation model and use the log-odds update rule in Eqn. (13) as the map encoder. This map encoder is sufficiently expressive to model occupancy probability from lidar observations in a 2D enviroment. The A* algorithm is used to solve the DSP (2), providing the optimal cost-to-go $Q^*_t(\mathbf{x}, \mathbf{u})$ for $\mathbf{x} \in$ CLOSED $\cup$ OPEN and a subgradient of $Q^*_t(\mathbf{x}_t, \mathbf{u}_t)$ according to (18). All the neural networks are implemented in the PyTorch library [25] and trained with the Adam optimizer [26] until convergence.

DeepMaxEnt uses a neural network to learn a cost function directly from lidar observations without explicitly having a map representation. The neural network in our experiments is the "Standard FCN" in [13] in the 16×16 domain, and the encoder-decoder architecture in [27] in the 100×100 domain. Value iteration is approximated by a finite number of Bellman backup iterations, equal to the map size. The experiments in the original DeepMaxEnt paper [13] use the mean and variance of the height of the 3D lidar points in each cell, as well as a binary indicator of cell visibility, as input features to the neural network. Since our synthetic experiments are set up in 2D, the count of lidar beams in each cell is used as replacement of the height mean and variance. This is a fair adaptation because Wulfmeier et. al. [13] argued that obstacles generally represent areas of larger height variance which means more beam counts in our observations.

Ours-HCE stands for hard-coded cost encoder. This simple variant of our model uses Eqn. (16) with $s = 1$ and $l = 100$ set explicitly as constants.

Ours-SCE stands for soft-coded cost encoder and has s and l in Eqn (16) as learnable parameters.

Ours-CNN is our most generic variant using a convolutional neural network as cost encoder. The network architecture is the same as in DeepMaxEnt for fair comparison.

	16 × 16				100 × 100			
Model	Val. loss	Val. acc. (%)	Test traj. succ. rate (%)	Test traj. diff.	Val. loss	Val. acc. (%)	Test traj. succ. rate (%)	Test traj. diff.
DeepMaxEnt [13]	0.18	93.6	90.9	0.145	0.23	93.7	31.1	6.528
Ours-HCE	1.10	59.5	99.7	0.378	1.32	42.2	100.0	2.876
Ours-SCE	0.66	62.1	97.2	0.174	0.66	62.1	84.4	1.569
Ours-CNN	0.27	90.5	96.7	0.144	0.14	95.1	90.1	1.196

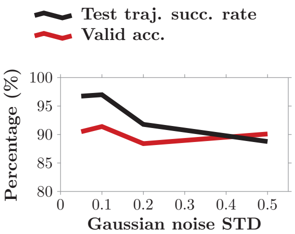


Fig. 4: Validation and test results for the 16 × 16 and 100 × 100 grid world domains. Cross entropy loss (3) and prediction accuracy for the validation set are reported. Test trajectories are iteratively rolled out from the non-stationary policy π_t. A trial is classified as successful if the goal is reached without collisions within twice the number of steps of a shortest path in the groundtruth enviroment. Ours-CNN is capable of matching the expert demonstrations while generalizing to new robot navigation tasks in test time. Right: Plot showing the effect of noise on the accuracy of Ours-CNN model.

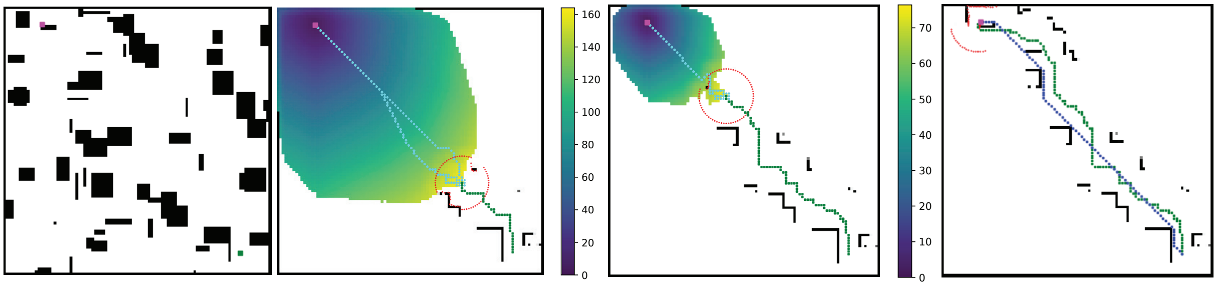


Fig. 5: Examples of occupancy estimation, A* motion planning and subgradient computation during a successful test trajectory. The first figure shows the *true* occupancy map $\mathbf{m}^*$ with the robot start and goal locations in green and magenta, respectively. The second and third figures show the current lidar observation $\mathbf{z}_t$ in red and robot trajectory thus far $\mathbf{x}_{1:t}$ in green. The map occupancy estimate $\sigma(\mathbf{h}_t)$ in grayscale is in the background. The optimal cost-to-go estimate g from A* is shown in a blue-yellow colormap in the foreground (brighter means higher cost-to-go). The optimal trajectories τ^* in cyan corresponding to $Q_t^*(x_t, u_t)$ are obtained during A* planning for subgradient computation in Eqn. (18). The last figure shows the final successful trajectory in green and an optimal shortest path in the fully observable enviroment in blue.

B. Experiments and Results

a) Model generalization: Fig 4 shows the comparison of multiple measures of accuracy for different algorithms. Both Ours-HCE and Ours-SCE explicitly incorporate cell traversability through the cost design in (16). Test results show that this explicit cost encoder is successful at obstacle avoidance, regardless of whether the parameters s, l are constants in Ours-HCE or learnable in Ours-SCE. The performance of Ours-HCE also shows that the map encoder is learning a correct map representation from the noisy lidar observations since the only trainable parameters are ψ in the inverse observation model (13). However, both models fail at matching demonstrations in validation because the cost encoder (16) emphasizes obstacle avoidance explicitly, leaving little capacity to learn from demonstrations. Ours-CNN combines the strength of learning from demonstrations and generalization to new navigation tasks while avoiding obstacles. Its validation results are on par with DeepMaxEnt, showing the validity of the closed-form subgradient in (18). Ours-CNN significantly outperforms DeepMaxEnt in new tasks at test time. The performance gap of DeepMaxEnt in the two domains shows that a general CNN architecture applied directly to the lidar scan measurements is not as effective as the map encoder in Ours-CNN at modeling occupancy probability. Fig 5 shows the map occupancy estimation, as well as the optimal trajectories necessary for subgradient computation in Sec IV.

b) Robustness to noise: The robustness of Ours-CNN to the observation noise is evaluated in the 16 × 16 domain. Fig 4 shows that the performance degrades as the noise increases but our inverse observation model (13) still generalizes well considering that noise with standard deviation of 0.5 is significant when the lidar range is only 2.5.

c) Computational efficiency: Finally, we compare the efficiency of a forward pass through our A* planner and the value iteration algorithm in DeepMaxEnt. The A* algorithm in our models is implemented in C++ and evaluated on a CPU. The VI algorithm is implemented using convolutional and minpooling layers in Pytorch as described in [6] and is evaluated on a GPU. We record the time that each models takes to return a policy π_t given a cost function c_t. Our A* algorithm takes only 0.02 ms as compared to VI's 0.2 ms on the 16 × 16 map. In the 100 × 100 domain, our A* algorithm takes 0.6 ms, compared to VI's 14 ms, illustrating the scalability of our model in the size of the state space.

VI. Conclusion

We proposed an inverse reinforcement learning approach for infering navigation costs from demonstration in partially observable enviroments. Our model introduces a new cost representation composed of a probabilistic occupancy encoder and a cost encoder defined over the occupancy features. We showed that a motion planning algorithm can compute optimal cost-to-go values over the cost representation, while the cost-to-go (sub)gradient may be obtained in closed-form. Our work offers a promising model for encoding occupancy features in navigation tasks and may enable efficient online learning in challenging operational conditions.

References

[1] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 663–670.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469 – 483, 2009.

[3] G. Neu and C. Szepesvári, "Apprenticeship learning using inverse reinforcement learning and gradient methods," in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2007, pp. 295–302.

[4] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 729–736.

[5] B. D. Ziebart, A. Maas, J. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008, pp. 1433–1438.

[6] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2154–2162. [Online]. Available: http://papers.nips.cc/paper/6046-value-iteration-networks.pdf

[7] M. Okada, L. Rigazio, and T. Aoshima, "Path integral networks: End-to-end differentiable optimal control," *arXiv preprint arXiv:1706.09597*, 2017.

[8] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," in *in Advances in Neural Information Processing Systems*, 2004, p. 767774.

[9] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] J. Choi and K.-E. Kim, "Inverse reinforcement learning in partially observable environments," *Journal of Machine Learning Research*, vol. 12, no. Mar, pp. 691–730, 2011.

[12] T. Shankar, S. K. Dwivedy, and P. Guha, "Reinforcement learning via recurrent convolutional neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2592–2597.

[13] M. Wulfmeier, D. Z. Wang, and I. Posner, "Watch this: Scalable cost-function learning for path planning in urban environments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2089–2095.

[14] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[15] P. Karkus, D. Hsu, and W. S. Lee, "QMDP-Net: Deep learning for planning under partial observability," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4694–4704. [Online]. Available: http://papers.nips.cc/paper/7055-qmdp-net-deep-learning-for-planning-under-partial-observability.pdf

[16] P. Karkus, X. Ma, D. Hsu, L. Kaelbling, W. S. Lee, and T. Lozano-Perez, "Differentiable algorithm networks for composable robot learning," in *Proceedings of Robotics: Science and Systems*, Freiburgim-Breisgau, Germany, June 2019.

[17] K. strm, "Optimal control of markov processes with incomplete state information," *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174 – 205, 1965. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022247X6590154X

[18] W. S. L. Hanna Kurniawati, David Hsu, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

[19] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee, "Memory augmented control networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[20] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[23] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[24] N. Z. Shor, *Minimization methods for non-differentiable functions*. Springer Science & Business Media, 2012, vol. 3.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[26] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," in *ICLR*, 2014.

[27] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *PAMI*, 2017.