Self-Learning Exploration and Mapping for Mobile Robots via Deep Reinforcement Learning

Fanfei Chen, Shi Bai, Tixiao Shan and Brendan Englot [*]
Stevens Institute of Technology, Hoboken, NJ, 07030

Mapping and exploration of *a priori* unknown environments is a crucial capability for mobile robot autonomy. A state-of-the-art approach for mobile robots equipped with range sensors uses mutual information as the basis for a cost metric [5], [14], and reasons about how much information gain is associated with each action a robot can take while constructing an occupancy map from its range measurements. However, the computational cost of such an optimization scales poorly as the number of potential robot actions increases. We propose a novel approach to utilize the local structure of the environment while predicting a robot's optimal sensing action using Deep Reinforcement Learning (DRL) [19]. The learned exploration policy can select an optimal or near-optimal exploratory sensing action with improved computational efficiency. Our computational results demonstrate that the proposed method provides both efficiency and accuracy in choosing informative sensing actions.

I. Introduction

We consider a mobile robot equipped with a range sensor, used to construct an occupancy map [9] as it explores an *a priori* unknown environment in which reliable localization measurements are available. It learns an exploration policy using the information-based reward collected while exploring a variety of different indoor environments with the aid of a Deep Reinforcement Learning (DRL) algorithm. The robot's final policy can predict the optimal sensing action from a local portion of the robot's map, iteratively working toward solution of the autonomous exploration problem [28] in a previously unknown environment. The robot's future reward will also be considered in the action selection process, which means the current action balances short-term maximization of mutual information (MI) with the potential for long-term maximization of future rewards. Our method is motivated by that of [19], which presents a state-of-the-art Deep Q-network (DQN) that learns to play Atari games through raw pixels of the screen. Also, [14] proves that exploring unknown environments by maximizing mutual information will lead a robot to unexplored spaces in the limit. Building on the success of these prior works, we also wish to leverage the fact that there are many similar characteristics among different office buildings, factories, parking garages, and other indoor environments. We wish to examine whether the exploration policy learned from environments with similar characteristics will facilitate faster entropy reduction and the selection of more informative robot sensing actions.

In our proposed approach, a mobile robot does not need to calculate the expected mutual information of each action exhaustively. After a training phase, the policy predicts the optimal sensing action based on the current local map. Specifically, as long as the architecture of a neural network is leveraged, the time complexity in the testing phase is constant. We also employ Recurrent Neural Networks (RNN) [21], which are widely used for sequential inputs, such as speech recognition and handwriting recognition, to train robots through continuous input map information. Finally, we have also modified the traditional DQN algorithm. A dropout layer [23] is implemented to prevent overfitting. Meanwhile, a Bayesian exploration strategy is adopted to accelerate the policy training process without *a priori* knowledge of the environments being explored. Figure 1 summarizes our general approach. In the *training phase*, the agent performs parameter optimization at every step t according to the reward r_t, as it performs reinforcement learning over occupancy maps. In the *testing phase*, at each time-step t', the robot uses its learned policy to predict an optimal sensing action $a_{t'}$ as a function of its current state $s_{t'}$.

Our contributions are as follows. A novel approach is introduced to solve autonomous mobile robot exploration using a robot's local map and DRL. This approach estimates a robot's potential information gain using a policy that is trained in environments with similar characteristics. Our method can achieve nearly equivalent performance to the classic MI approach. Meanwhile, the computational complexity of our approach is O(1), while the computation time of

[*]F. Chen, S. Bai, T. Shan and B. Englot are with the Department of Mechanical Engineering, Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ, 07030. `{fchen7, sbai1, tshan3, benglot}@stevens.edu`

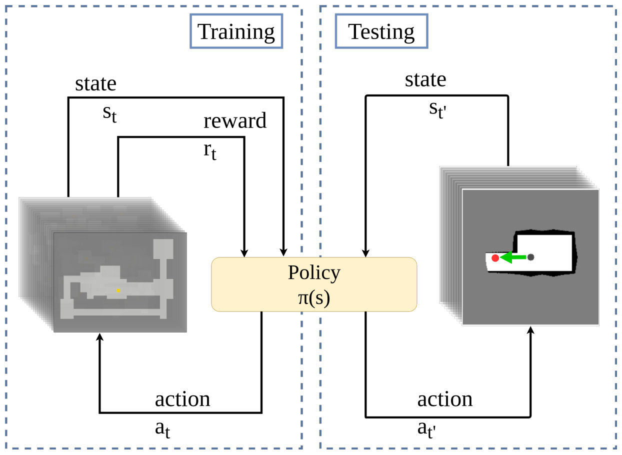

Fig. 1 An illustration of our framework. In the *training phase*, we expose the robot to different environments and the policy optimizes parameters in the neural network by informative reward r_t, where the current state is a local map s_t and the corresponding robot action is a_t. Training concludes after 2 million episodes. Then in the *testing phase*, we provide a different set of diverse environments. The policy is used to estimate the optimal action, which is indicated by the red point a_t' from the current robot's local map s_t'.

the MI approach suffers from the need for repeated ray-casting across the workspace, applied iteratively across all robot actions in a potentially high-dimensional action space.

A. Related Work

Information-theoretic exploration methods were developed to quickly guide robots to the unexplored areas of their environment that will contribute the largest quantities of new information to an occupancy map [5], [6], [14]. Due to the high cost of computing the expected mutual information associated with a series of candidate sensing actions, Bai et al. proposed a Gaussian process based mutual information prediction method [2], and a subsequent Bayesian optimization active sampling approach [3], to speed up the prediction of mutual information throughout a robot's action space. Furthermore, [4] shows that neural networks can estimate the information gain with respect to each action. However, this supervised learning method has a low tolerance for error and will fail quickly for most of the exploration testing.

Model-based methods often accelerate the training process [11] since the agent can obtain training information from a model in addition to the rewards from the environment. A pre-trained policy can also improve the performance of learning and increase a robot's learning efficiency [8]. Jaderberg proposed an auxiliary approach to explore the potentially reward-rich areas of an environment, avoiding low-reward areas [13]. Such methods require *a priori* knowledge of either the environments or the policy model.

Tai introduced a deep learning [25] and a deep reinforcement learning [26] based obstacle avoidance policy that is trained and tested using sensor data from the same environment. The learned obstacle avoidance policy can only be used in the same environment in which it is trained, and is ineffective for use across heterogeneous environments. Deep Recurrent Q-Networks (DRQN) were developed in [17] to play First-Person-Shooting (FPS) games in a 3D environment. By combining an RNN with a DQN, the DRQN can generate appropriate outputs that depend on the temporally consecutive inputs. Meanwhile, a target-driven robot visual navigation policy was also trained using deep reinforcement learning [30]. The robot was trained in a simulated environment and implemented its policy successfully in a physical environment. However, this method requires a sophisticated simulation environment and it is challenging to predict the information gain of many potential future sensing actions using a single camera view. Choudhury et al. proposed the ExpLOre algorithm [7] to gather information using imitation learning. This algorithm provides non-myopic solutions for the autonomous exploration problem, however, the policy is trained and tested over the same maps, with different view nodes. Other works involving reinforcement learning and robot navigation in [12] and [27] have dealt with learning the topology of an environment as well as motion planning, however our work focuses purely on the efficiency of mapping in the absence of a prior map.

B. Paper Organization

We propose and evaluate a methodology to choose sensing actions for mobile robot exploration using policies trained by DRL, with the aim of taking the optimal action to gather the maximal immediate information gain at each step in the process of exploring an *a priori* unknown environment. A formal definition of the problem is given in Section II, including a brief discussion of information theory, which forms the basis for our reward function, and an overview of reinforcement learning and DRL. Experimental results are presented in Section III, with conclusions in Section IV.

II. Problem Formulation and Approach

A. Problem Definition

Instead of choosing the sensing action of maximum expected information gain after a computationally expensive evaluation of all the candidate sensing actions, we employ a neural network to predict the optimal sensing action and train it using reinforcement learning. We use a database of maps with similar characteristics, however with different individual topologies and layouts. A flowchart demonstrating both the training and testing phases is shown in Figure 1. We train and test the neural network on locally visible portions of randomly-generated 2D maps of indoor environments. We consider these locally visible, uniformly-sized regions instead of the entirety of a robot's currently acquired map with the aim of achieving a scalable, computationally efficient approach that may be applied to environments of arbitrary size. However, this is done with the understanding that we are training an information-seeking controller rather than a global optimization method.

B. Entropy and Mutual Information

We use Shannon's entropy [22] as the metric to represent the completeness of our knowledge about the map, which is described in the following equation :

$$H(m) = -\sum_i \sum_j p(m_{i,j}) \log p(m_{i,j}). \tag{1}$$

In Eq. (1), m represents the current occupancy grid map [9], where index i refers to the individual grid cells of the map and index j refers to the possible outcomes of the Bernoulli random variable that represents each grid cell, which is either free or occupied. Cells whose contents have never been observed are characterized as $p(m_{i,j}) = 0.5$, contributing one unit of entropy per cell. Cells whose contents are perfectly known contribute no entropy to the summation.

We define Mutual Information (MI) $I(m, a_i)$ to be the expected information gain with respect to the action a_t:

$$I(m, a_i) = H(m) - H(m|a_t), \tag{2}$$

where $H(m)$ is the entropy of the current map, and $H(m|a_t)$ is the expected entropy after collecting a sensor observation upon executing the action a_t.

C. State Space and Action Space

We initially adopted the robot's entire global map as the state space S_{global}, however it was challenging to achieve efficient convergence for such a large state space, and map size may vary widely from one application to the next. Thus, we instead choose a local region of the map, centered at the robot's current location, as our state space S as shown in Figure 2(a). We define the action space A as a set of quasi-random samples drawn within a constant radius of the robot, shown in Figure 2(b) (here we use the Sobol sequence).

D. Reward Design

Mnih et al. [19] has suggested normalizing the reward so that $r \in [-1, 1]$. We choose 70 percent of the maximum possible mutual information from Eq. 2 as the upper bound for the reward (which is rarely exceeded in practice). We also normalize the MI acquired by an action a_t using 70 percent of the maximum possible MI, if the MI obtained is greater than zero. When an action leads to collision with an obstacle or zero information gain, we assign $r = -1$ and $r = -0.8$ repectively. The reason for assigning negative reward to actions with zero information gain is to discourage driving to previously observed areas of the map.

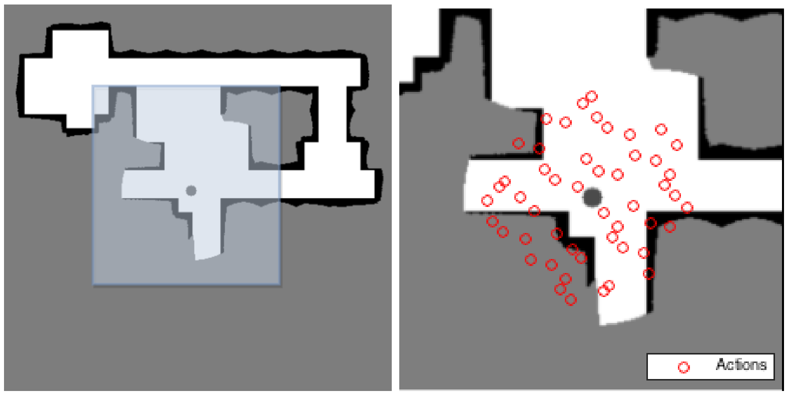

Fig. 2 Left: A local map is extracted from the global map and is used to represent the state space. The gray point in the free space indicates the location of the robot itself. Right: An illustration of the same robot's action space.

E. Value Function

The function V^π is the state-value function for policy π:

$$V^\pi(s) = \mathbb{E}_\pi\{R_t | s_t = s\}. \tag{3}$$

In (3), $\mathbb{E}_\pi$ indicates the expected discounted future reward R provided by the policy π in terms of the current state at time-step t. Furthermore, the function Q^π is the state-action value function for policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi\{R_t | s_t = s, a_t = a\}. \tag{4}$$

In (4), at time-step t, not only the state but also the action are utilized to generate the expected discounted future reward R provided by the policy π. Both value functions V^π and Q^π are used to evaluate the current state and action. Using these value functions, the agent can identify whether the current state and action yield a high reward. Specifically, the larger number shows the better circumstance.

F. Deep Reinforcement Learning

The typical setting for reinforcement learning (RL) [24] is a partially observable Markov decision process (POMDP). The agent chooses an action guided by policy π based on current state S, and expects to maximize its discounted future reward R, which is represented as follows:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{5}$$

where $\gamma \in [0, 1]$ denotes the discount rate. By setting a higher value for γ, it will encourage the model to pay more attention to future rewards. In contrast, the model's training will be more focused on the current action if γ decreases.

We use a Deep Reinforcement Learning (DRL) model [15] and define the transition tuple as $< s, a, r, s' >$, where s denotes a robot's current state, a denotes its action, r denotes the reward and s' denotes the next state, achieved by the transition. The Bellman optimality equation for $Q^\pi(s, a)$ is:

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}\{r_t + \gamma \max_{a' \in A} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \\ &= \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in A} Q^*(s', a')] \\ &= r + \gamma \max_{a' \in A} Q^*(s', a'), \end{aligned} \tag{6}$$

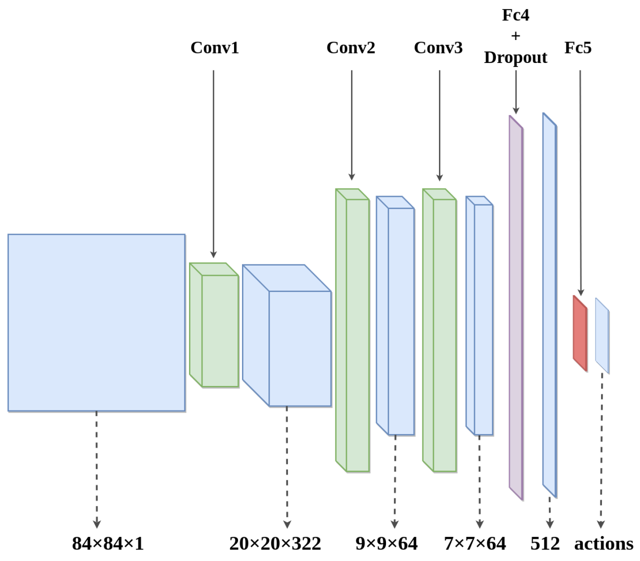

Fig. 3 An illustration of our convolutional neural network, in which the blue cubes represent each data structure. The original input data is a single-frame grayscale image. The output of the network is a one-dimensional vector, where each entry is the reward associated with one specific action.

where $\mathcal{P}_{ss'}$ are the transition probabilities and $\mathcal{R}^a_{ss'}$ is the expected reward for the current transition. In our model, the transition probabiliy from s_t to s_{t+1} is 1, and thus $s' = s_{t+1}$ and $\mathcal{P}^a_{ss'} = 1$. The policy is defined as follows:

$$\pi(s) = \arg\max_{a \in A} Q(s, a), \tag{7}$$

and the loss function is defined as follows:

$$L = \frac{1}{2}[r + \gamma \max_{a' \in A} Q^*(s', a') - Q^*(s, a)]^2. \tag{8}$$

The loss of the action-value function is the squared difference of the policy-provided value and the Bellman optimality equation-provided value. We want to minimize the loss through training, so that the policy π can select a near-optimal action even if it misses the optimal one. We use the Adam first-order gradient-based algorithm [16] when optimizing the parameters of the DRL model.

G. Neural Networks

We integrated two different neural networks in our framework, including the convolutional neural network (CNN) used for Q-learning in [19], as shown in Fig. 3, and a Long Short Term Memory network (LSTM) [21], which is a state-of-the-art structure for RNNs. The output is influenced by the temporal sequential inputs to the RNN model, offering the prospects of decision making that considers the temporal context of an input state. Furthermore, we have added a dropout layer to both neural networks to avoid overfitting.

H. Training Strategies: Bayesian

The work of [10] has proven that the output of the dropout [23] neural network layer can represent the uncertainty of the actions. With probability p, the output of the dropout layer is the input element scaled up by $1/p$. For the networks with a dropout layer, we first define $p = 0.1$ to choose the most uncertain action, and as the training phase runs, p is increased gradually until $p = 1$, which means the robot always take an action using the policy.

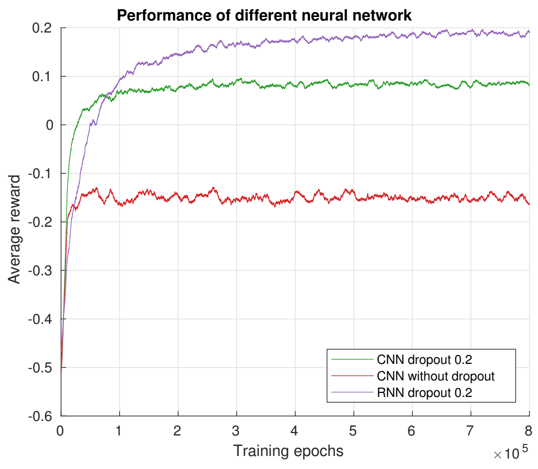

Fig. 4 Convergence rate of neural networks during training. Both the CNN and RNN benefit from having a dropout layer, which improved not only the convergence rate but also the average reward.

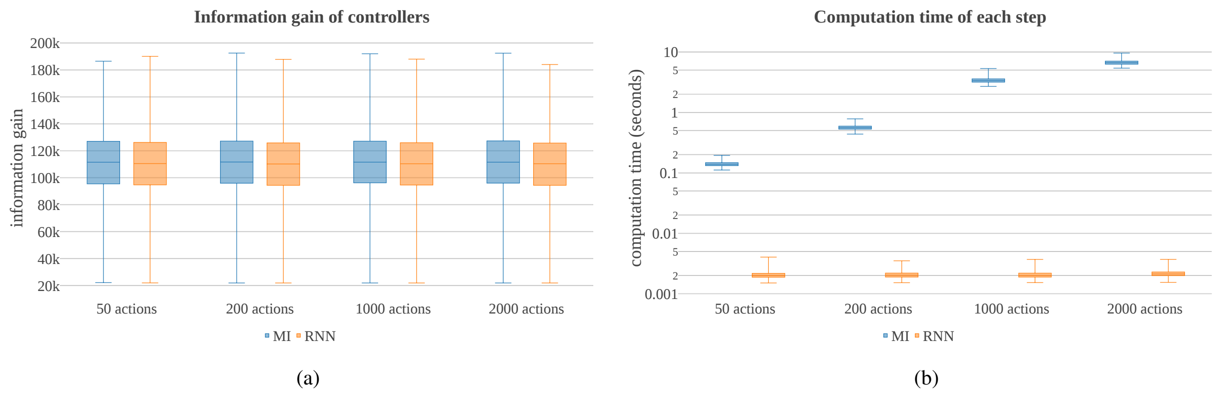

Fig. 5 (a): A comparison of information gain achieved by the MI-maximizing controller and the RNN approaches, over action spaces of different sizes. (b): The average computation time for each decision making step, over action spaces of different sizes. Note that both plots show not only the mean and standard deviation (wide bars), but also the minimum and maximum value of information gain and computation time (narrow bars) among all trials.

III. Experiments and Results

A. Experimental Setup

We first trained a policy over a 2D maze-like environment, using 5663 maps during the training phase. The simulated robot in our experiment is provided a 360-degree field of view noise-free range sensor. Its occupancy grid map is handled deterministically, where $p = 1$ for occupied cells, $p = 0$ for free cells and $p = 0.5$ for unknown cells. Every map is $640m \times 480m$, while the sensor range and each robot step are 80m and 40m, respectively. We use a $240m \times 240m$ local submap captured at the robot's current location as the input state to the neural network. We translate the local map to a grayscale image, where 255 indicates free space, 1 indicates occupied space, and 127 is unexplored space. Further, we denote the robot in the map as a circle with a 6m radius, and it is assigned a grayscale value of 76.

The robot checks for collisions before taking any action, and will include instances of collision in its training. However, the robot will only proceed with exploring after a collision-free action is selected. If the robot navigates into a dead end, this will trigger a "frontier rescue (FR)" strategy, which drives the robot to the nearest frontier [29] in its occupancy map, so that we may continue training with the current map. We explored different neural networks for training, as described in Section II-G.

The simulation environment is written in Python, with a C++ library for the inverse sensor model, and our neural

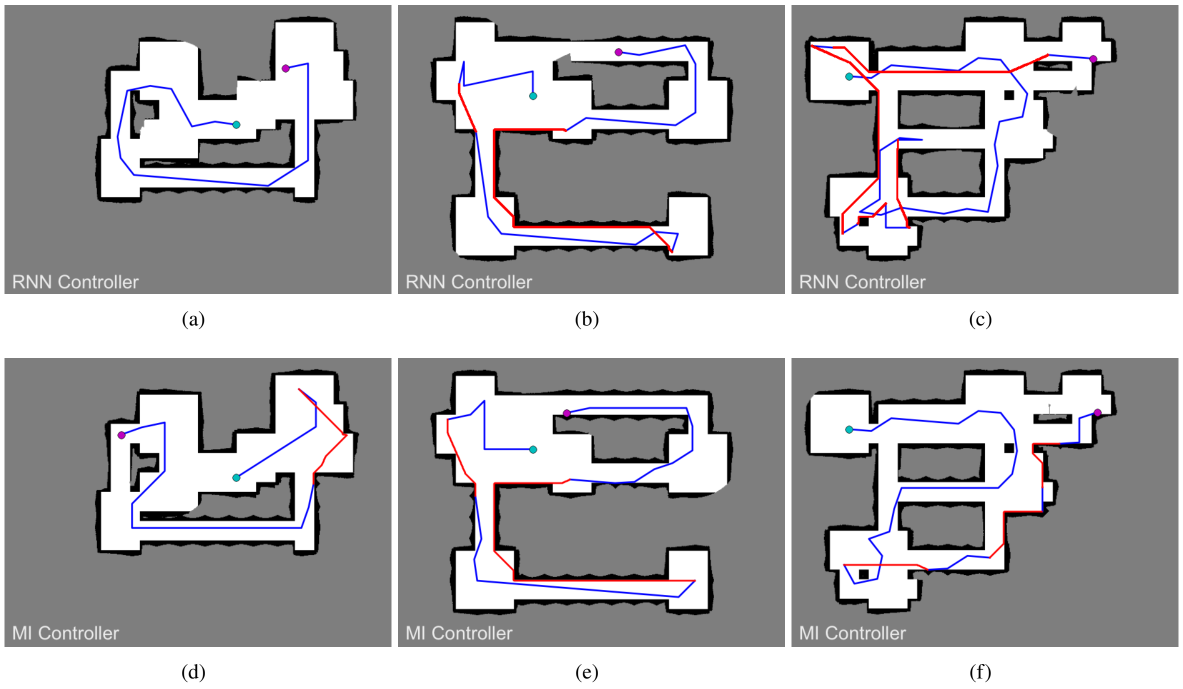

Fig. 6 Representative trajectories produced by the RNN controller (top) and the exhaustive MI controller approach (bottom), over three example maps from our testing phase dataset. The robot starts from the cyan-colored point and ends at the magenta-colored point in each trial (at which point the map has been fully explored). When the robot is stuck in a local mapped region, the frontier rescue method will lead the robot to go to the nearest frontier point by using the A* path planning algorithm, which is indicated by the red lines above. The blue lines are the trajectories provided by the RNN controller and the MI controller. Subfigures (a) and (d) show an RNN controller instance that outperforms the MI controller, (b) and (e) show similar outcomes among the two methods, and (c) and (f) illustrate a lower-performing RNN controller instance relative to the MI controller.

network model is trained using TensorFlow [1]. However, the inverse sensor model cannot be processed in parallel because the workload of each ray casting operation is too low compared with the associated overhead. The training and testing maps are generated randomly by [20].

B. Testing in a 2D environment

We compared the learned RNN controller and an exhaustive MI-optimizing controller on a testing-phase dataset comprised of 5218 maps. The MI controller, along the lines of [14], explicitly evaluates the expected information gain of every possible sensing action, by projecting the sensor's rays throughout the map at each candidate configuration, and choosing the sensing action that offers maximum expected MI. A frontier based rescue mechanism will take place when there is zero information gain resulting from any action. For both methods, the exploration of a map will be terminated only after the entropy of a map has been reduced to zero. The testing of the mutual information controller was performed using an Intel i7 3.5Ghz CPU, while the learned RNN controller was run on an Nvidia GeForce GTX 1080 GPU.

The FR heuristic is introduced to lead the robot to the nearest frontier from obstacle-induced or information-poor dead-ends, where the A* algorithm finds an efficient path to the nearest frontier from the robot's current location. We believe employing an efficient planning algorithm such as A* is a reasonable approach for navigating from *known* areas of the map back to the frontiers, as the focus of this study is learning to select informative sensing actions in partially unknown environments, rather than navigating known maps.

We select two metrics to show the relative performance of both methods. The total information gain from controller-driven sensing actions is a key element to evaluating the performance of the methods, while the information gain acquired by FR heuristic has been excluded. We assume the robot executes each sensing action at the same, constant velocity, as it is straightforward that the computation time of each decision making step is also a critical index

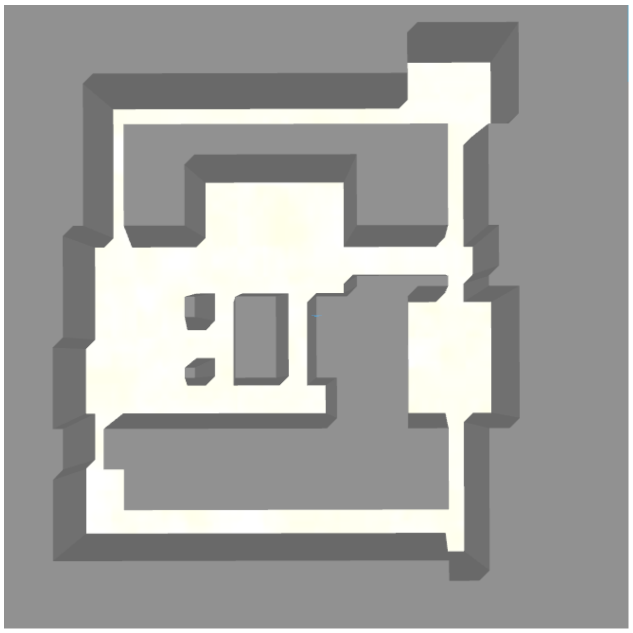
Fig. 7 A 3D synthetic environment (top view) used for both training and testing.

to compare the efficiency of the two controllers.

Fig. 4 demonstrates the outcomes of the training phase for the case of 50 actions, giving a comparison among all of the neural networks examined. In two cases, we set a dropout layer with a rate of 0.2 in the penultimate layer of the neural network. The dropout strategy improves the final converged average reward for both the RNN and CNN. The CNN without a dropout layer offers the worst reward result in the end. Meanwhile, the CNN with a 20 percent dropout layer achieves a higher average reward. Although the RNN has a slower rate of convergence than the other networks examined, because it needs to learn sequential map information, it provides the best average reward in the end.

Fig. 5 gives results showing the performance of the exploration methods over different-sized action spaces in the testing phase. We use box plots to describe the distribution of the testing set of data. Each box plot represent 5218 data which correspond to the number of testing maps. The x-axis of each plot indicates the size of the action space examined. MI indicates a traditional exhaustive MI-maximizing controller. The RNN approach indicates the learned RNN controller from the training phase. In Fig. 5(a), the y-axis indicates the total amount of information gain obtained only by the controllers in each map. The RNN controller offers nearly the same performance as the traditional MI approach across the different action spaces. In Fig. 5(b), the log-scale y-axis indicates the computation time required for decision making at each step. The computation time of the exhaustive MI-maximizing approach grows sharply as the action space grows. Meanwhile, the computation time of the RNN method remains constant as the action space grows.

Fig. 6 shows a comparison of representative trajectories generated by the MI-maximizing controller and our RNN controller over the same three maps. Figs. 6 (a), (b) and (c) are produced by the learned RNN controller, and (d), (e) and (f) are generated by the MI-maximizing controller. Trajectory (a) offers a non-cyclical trajectory which explores more efficiently than the outcome of (d). Trajectories (b) and (e) show a similar result produced by the two respective methods. Trajectory (c) has a worse exploration outcome because FR is implemented several more times than in (f). However, in spite of the occasional need for frontier-guided paths, the trajectory portions produced by the learned RNN controller are straightforward and efficient.

Finally, this paper's video attachment offers three examples of how our RNN controller performs in the testing phase, after different quantities of training epochs[*]. As the learned policies converge over the course of two million epochs, each of the three examples manages to explore its map in entirety with an exclusive reliance on the RNN controller, without the need for the frontier rescue heuristic.

[*]`https://youtu.be/2gNF6efv12s`

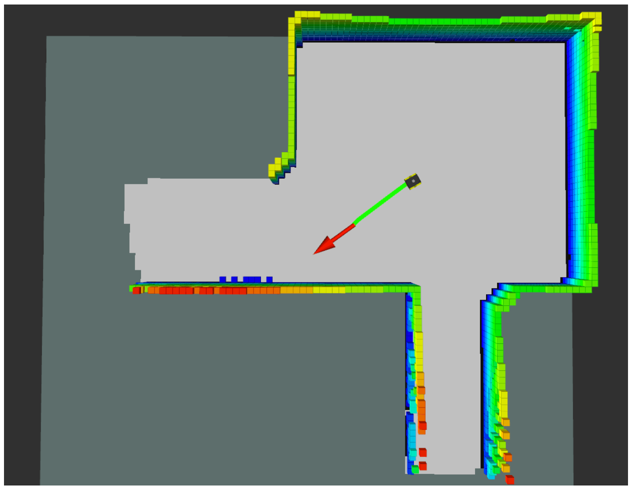

Fig. 8 The RNN controller uses a local 3D occupancy grid submap as the input data and estimates the best sensing action for the robot.

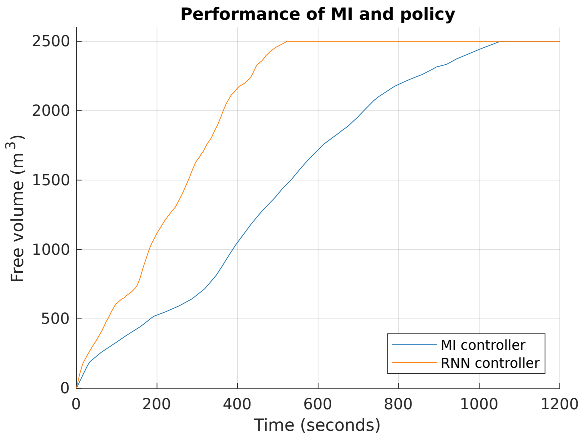

Fig. 9 Comparison of information gain between the RNN and MI controllers in a 3D synthetic environment (average of 6 trials). Note there is no frontier rescue here, and the robot will explore until the mapped free volume reaches the designated threshold, 2500 m^3.

C. 3D exploration

Furthermore, we validate the proposed approach in a 3D synthetic environment. Due to the complexities of training over a much denser input, in this experiment we train and test on the same map, but from different start locations. The testing of the MI controller was performed using an Intel i7 4.2Ghz CPU, while the learned RNN controller was run on an Nvidia Geforce Titan X Pascal GPU. We equipped the robot with a simulated Velodyne VLP-16 3D LiDAR Sensor which provides a 360-degree horizontal field of view and a 30-degree vertical field of view. Fig. 7 shows the environment used for 3D exploration. The robot was initialized from random locations in the map during training, and a selected set of locations when testing (close to the corners of the map). Fig.8 shows an example sensing action selected by the RNN controller. We use a similar RNN framework (which takes a more dense input) and trained it from scratch. We use a $30m \times 30m \times 0.8m$ local occupancy grid map with $0.2m$ resolution as the input state. We assigned a value 10 to indicate occupied cells, 255 for free cells and 127 for unknown cells. The action space is a set of 200 quasi-random points within a 5m radius in the horizontal plane. We apply the same strategy when developing the reward function.

In Fig. 9, the information gain received only by the MI baseline approach and the RNN approach, without the involvement of the FR heuristic, is shown for our 3D scenario. The robot will stop exploration if it maps 2500 m^3 of the free volume of the map (the total volume of the free space is 3200 m^3). The y-axis of the plot shows the average free volume mapped from four different starting points at the corners of the map. The result shows that the RNN policy leads to a 100 percent increase in efficiency in reaching the free-volume goal, compared with the MI method.

In addition to showing three examples of how our RNN controller performs in the testing phase over 2D maps, the video attachment also shows the full 3D exploration process for both the MI and RNN controllers. The robot stops when it has mapped a designated fraction of the available free volume of the map (90 percent of the total free volume). As depicted in Fig. 8, the timing of 3D exploration shown in the video also includes the computation time associated with robot decision making under each of the competing autonomous exploration frameworks.

IV. Conclusions

We have proposed a novel approach to estimate the best mobile robot sensing action in the course of exploring an unknown environment, trained via deep reinforcement learning. In our framework, the DRL policy generates robot sensing actions that are nearly as informative and efficient as those of a standard mutual information maximizing controller, at a substantial reduction in computational effort during the online testing phase. The time complexity in the testing phase is O(1), which means that our proposed method is scalable to higher-dimensional state-action spaces, as long as the policy has been trained in a space of the same dimension. For the mutual information optimizing controller, with increasing dimension of the configuration space and action space, the procedure's computational complexity is no longer be real-time viable. Looking ahead to future work, we also note that it is difficult to identify useful and descriptive frontiers in a 3D map. A key goal for future work is the adoption of a global muti-step planner in conjunction with our current local controller in the training phase. This would enable the robot to use its learned policy to leave dead ends without the frontier rescue strategy.

Acknowledgments

This research has been supported in part by the National Science Foundation, grant number IIS-1723996.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin and S. Ghemawat, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv:1603.04467 [cs.DC]*, 2016.

[2] S. Bai, J. Wang, K. Doherty, and B. Englot, "Inference-Enabled Information-Theoretic Exploration of Continuous Action Spaces," *Proceedings of the International Symposium on Robotics Research*, 16 pp., 2015.

[3] S. Bai, J. Wang, F. Chen, and B. Englot, "Information-theoretic Exploration with Bayesian Optimization," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1816-1822, 2016.

[4] S. Bai, F. Chen, and B. Englot, "Toward autonomous mapping and exploration for mobile robots through deep supervised learning," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2379-2384, 2017.

[5] B. Charrow, G. Kahn, S. Patil, S. Liu, K. Goldberg, P. Abbeel, N. Michael, and V. Kumar, "Information-theoretic Planning with Trajectory Optimization for Dense 3D Mapping," *Proceedings of Robotics: Science and Systems*, 2015.

[6] B. Charrow, S. Liu, V. Kumar, and V. Michael, "Information-theoretic mapping using cauchy-schwarz quadratic mutual information.," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4791-4798, 2015.

[7] S. Choudhury, A. Kapoor, G. Ranade, and D. Dey, "Learning to Gather Information via Imitation.," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017.

[8] Y. Duan, J. Schulman, X. Chen, P.L. Bartlett, I. Sutskever, P. Abbeel, "RL ^{2}: Fast Reinforcement Learning via Slow Reinforcement Learning," *arXiv:1611.02779 [cs.AI]*, 2016.

[9] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22(6), pp. 46-57, 1989.

[10] Y. Gal, Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," *Proceedings of the International Conference on Machine Learning*, pp. 1050-1059, 2016.

[11] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," *Proceedings of the International Conference on Machine Learning*, pp. 2829-2838, 2016.

[12] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," *arXiv:1702.03920 [cs.CV]*, 2017.

[13] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, “Reinforcement learning with unsupervised auxiliary tasks," *arXiv:1611.05397 [cs.LG]*, 2016.

[14] B.J. Julian, S. Karaman, and D. Rus, “On Mutual Information-Based Control of Range Sensing Robots for Mapping Applications," *The International Journal of Robotics Research*, vol. 33(10), pp. 1375-1392, 2014.

[15] A. Juliani. Simple Reinforcement Learning with Tensorflow. `https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df#.snatim6dt`

[16] D. Kingma, J. Ba, “Adam: A method for stochastic optimization," *Proceedings of The International Conference on Learning Representations*, 2014.

[17] G. Lample, D.S. Chaplot, “Playing FPS Games with Deep Reinforcement Learning," *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 2140-2146, 2017.

[18] J. Michels, A. Saxena and A.Y. Ng, “High speed obstacle avoidance using monocular vision and reinforcement learning," *Proceedings of the International Conference on Machine Learning*, pp. 593-600, 2005.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J, Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski and S. Petersen, “Human-level control through deep reinforcement learning," *Nature*, vol. 518(7540), pp. 529-533, 2015.

[20] “Random Dungeon Generator," `http://perplexingtech.weebly.com/random-dungeon-demo.html`

[21] H. Sak, A. Senior, F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling," *Fifteenth Annual Conference of the International Speech Communication Association*, pp. 338-342, 2014.

[22] C.E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949.

[23] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15(1), pp. 1929-1958, 2014.

[24] R.S. Sutton, A.G. Barto, *Reinforcement learning: An Introduction*, MIT Press, 1998.

[25] L. Tai, S. Li and M. Liu, “A deep-network solution towards model-less obstacle avoidance," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2759-2764, 2016.

[26] L. Tai and M. Liu, “Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv:1610.01733 [cs.RO]*, 2016.

[27] A. Tamar, Y. Wu, G. Thomas, S. Levine, P. Abbeel, “Value iteration networks," *Advances in Neural Information Processing Systems*, pp. 2154-2162, 2016.

[28] S. Thrun, W. Burgard, and D. Fox, “Exploration," In *Probabilistic Robotics*, pp. 569-605, MIT Press, 2005.

[29] B. Yamauchi, “A frontier-based approach for autonomous exploration," *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 146-151, 1997.

[30] Y. Zhu, R. Mottaghi, E. Kolve, J.J. Lim, A. Gupta, L. Fei-Fei, A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017.

[31] A. Hornung, K.M. Wurm, M. Bennewitz,C. Stachniss, W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 34(3), pp.189-206, 2013.