Active Mapping using Deep RL

Shane Barratt
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
sbarratt@stanford.edu

Abstract

We propose a learning algorithm for active robotic mapping, where the robot's goal is to map the environment as quickly as possible. The robot learns to map efficiently by moving around in simulated environments and receiving rewards for mapping the environment quickly. In contrast to previous work, our approach learns an exploration policy based on a probabilistic prior over environments and an inverse sensor model, allowing it to specialize. We evaluate our approach through a simulated Disaster Mapping scenario and find that it achieves performance on-par with a near-optimal exploration scheme.

1 Introduction

For most of the paper, we will use the following terminology borrowed from [1].

- robot: an autonomous agent in an environment
- pose (x_t): position and orientation of the robot in that environment
- observations (z_t): readings from on-board sensors
- controls (u_t): control inputs to actuators

Two fundamental problems that autonomous robots face are that of *mapping* and *localization*. The goal of mapping is to obtain a spatial model of the robot's environment using poses and observations. Simultaneously, the goal of localization is to estimate the robot's pose using observations and a map. These problems are inherently linked, as mapping requires poses and localization requires maps. Addressing both is commonly referred to simultaneous localization and mapping (SLAM) problem [2]. In this work, we focus on the problem of efficient mapping with known poses, assuming that the localization problem is solved. We view the integration of localization into our approach as future work.

There are three main hurdles we must overcome when designing robotic mapping algorithms. First, we need a representation of the map and a representation of our belief over possible maps. Second, we need a way to update our belief over maps given observations. Third, we need to choose controls to reduce our uncertainty in our belief as quickly as possible.

One of the most widely used (and elegant) ways to represent the map and incorporate observations sequentially is using occupancy grids [3]. Occupancy grids exhibit several desirable properties: they are adaptable for many applications, they compute an exact posterior distribution over maps (under some independence assumptions), they are strongly convergent, they can deal with any type of sensor noise, and they can handle raw sensor data. However, occupancy grid maps lack a principled method for our third hurdle: how to choose control inputs. This work proposes a solution.

We pose the problem of designing agents which can actively map the environment as a Markov Decision Process (MDP), where the state includes the robot's pose and its belief of the environment. From this information, the robot should be able to identify areas in the map which require exploration

and to move to them and map them. However, the robot's belief is a high-dimensional matrix of probability values. Accordingly, we turn to deep reinforcement learning techniques to find policies which act directly based on the belief. To train the policy, we use a prior over maps the robot might encounter and repeatedly train the robot to explore these maps in simulation. We evaluate our algorithm on a simulated Disaster Mapping environment, and find that it is able to achieve performance on-par with a near-optimal greedy strategy. This leads us to believe that the algorithm will be able to scale to more complex environments and sensors.

2 Related Work

The two most popular methods for addressing the third hurdle are frontier-based exploration [4], where the robot actively seeks to visit new poses, and information-based exploration [5], where the robot myopically chooses control inputs to maximize information gain over one step.

2.1 Frontier-Based Exploration

In frontier-based exploration, the robot keeps a set of *frontiers*, which are defined as regions on the boundary between open and unexplored space. The robot then navigates to the closest frontier by performing a depth-first-search (DFS) on the maximum likelihood map. If, after a predetermined number of steps, the robot fails to reach the frontier region it repeats the process over again. There is also a natural multi-robot extension proposed by the same author [6].

2.2 Information-Based Adaptive Robotic Exploration

In information-based exploration, the robot moves in the direction which maximizes the expected information gain

$$\arg\max_{u_t} H(b_t) - \mathbb{E}_{x_{t+1}, z_{t+1}}[H(b_{t+1})|u_t, x_t],$$

where

$$\mathbb{E}_{x_{t+1}, z_{t+1}}[H(b_{t+1})|u_t, x_t] = \sum_{x_{t+1}} \sum_{z_{t+1}} P(x_{t+1}|x_t, u_t) P(z_{t+1}|x_{t+1}) H(b_{t+1}|x_{t+1}, z_{t+1}).$$

Note that this summation can become extremely hard to compute, as the summation becomes an integral when the sensor outputs continuous values and the summation can be exponential with the number of possible sensor outputs. Also, this is a myopic exploration strategy; it only looks one step into the future to decide its next action. However, in simple environments it can be a near-optimal exploration scheme. This is the baseline which we compare our approach against.

3 Preliminaries

In this section, we introduce Occupancy Grids, Markov Decision Processes and the Advantage Actor-Critic Algorithm. The section also serves as an introduction of the mathematical notation that will be used throughout the paper.

3.1 Occupancy Grids

The basic idea behind occupancy grid maps is to represent the map as a two-dimensional grid of binary random variables which represent whether or not the location is *occupied* or *not occupied*.

More concretely, we posit that there is an underlying grid map $m \in M = \{0, 1\}^{N \times N}$ that is a-priori unknown to the robot. We wish to calculate our belief over maps M at time t given all previous poses and observations leading up to that time step $b_t(m) = p(m|x_{1:t}, z_{1:t})$. Reasoning about all possible maps quickly becomes intractable, as there are 2^{N^2} possible maps. To simplify the problem, we assume the individual map random variables, indexed as m_i, are independent given the poses and measurements, or

$$b_t(m) = \prod_i p(m_i | x_{1:t}, z_{1:t}) = \prod_i b_t(m_i).$$

We work with the log-odds posterior $l_{t,i} = \log \frac{b_t(m_i=1)}{1-b_t(m_i=1)}$ for simplicity and can recover our posterior probabilities using $b_t(m_i = 1) = 1 - \frac{1}{1+\exp\{l_{t,i}\}}$. We can update our posterior given a sensor reading via an inverse sensor model $p(m_i|x_t, z_t)$ using a recursive Bayes filter [1]:

$$l_{t,i} = l_{t-1,i} + \log \frac{p(m_i = 1|x_t, z_t)}{1 - p(m_i = 1|x_t, z_t)} - \log \frac{p(m_i = 1)}{p(m_i = 0)}. \tag{1}$$

Later, we will use the information-theoretic entropy of the belief state to quantify our uncertainty, which factorizes over the individual map random variables because they are assumed to be independent,

$$H(b_t(m)) = \sum_i H(b_t(m_i)) = \sum_i b_t(m_i = 1) \log b_t(m_i = 1) + b_t(m_i = 0) \log b_t(m_i = 0).$$

3.2 Markov Decision Processes

Markov decision processes (MDP) are a mathematical framework for decision making where the outcomes are random and potentially affected by the decision maker [7]. An MDP is formally defined as a tuple $< S, A, T, R, \gamma >$. S is a set of states the decision maker might be in. A is the set of actions the decision maker can take. $T(s'|s, a)$ is the distribution over next states given the decision maker took action a in state s. $R(s, a)$ is the reward the agent receives for taking action a in state s. γ is the discount factor, weighing short-term rewards versus long-term rewards.

3.3 Advantage Actor-Critic

The Advantage Actor-Critic (A2C) algorithm is the single-threaded version of [8], and has enjoyed a lot of success in playing ATARI from raw pixel observations. It prescribes a method for learning a differentiable stochastic policy $\pi(a|s; \theta')$ and value function $V(s; \theta'_v)$ through interactions with an environment. More concretely, after n interactions with the environment, it performs a gradient step on the following loss functions for each interaction:

$$L(\theta) = -\log \pi(a_t|s_t; \theta')(R - V(s_t; \theta'_v)) - \lambda H(\pi(a_t|s_t; \theta')),$$

$$L(\theta'_v) = (R - V(s_t; \theta'_v))^2,$$

where R is the bootstrapped $n-$step reward.

4 Approach

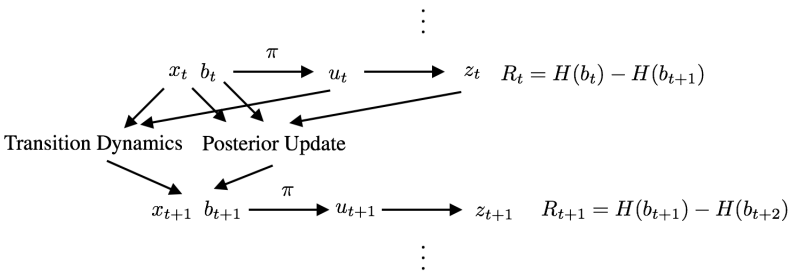
We separate the problem of updating the posterior from selecting actions. At each time step t, the robot receives an observation z_t and its pose x_t. It updates its posterior b_t using Equation 1. At this point, the robot is faced with the decision of what control input u_t to select given b_t and x_t. Instead of choosing the action myopically or planning, we resort to reinforcement learning (RL) to train our mapping agent to act directly based on its belief state.

We convert our notation to the MDP notation in Section 3.2. We let the state be defined as $s_t = [b_t, x_t]$ and the action as $a_t = u_t$. The state evolves according to Equation 1 and the underlying map. In our RL formulation, the robot seeks a policy $\pi(a|s)$ which reduces uncertainty as quickly as possible, or equivalently one which maximizes the following discounted expected cumulative reward

$$\mathbb{E}_{\pi,m}\big[\sum_{t=0}^{T-1} \gamma^t R_t\big],$$

where $R_t = H(b_t) - H(b_{t+1})$ is the reduction in the entropy of the robot's belief state at time step t and the expectation is taken over maps, the policy and the imposed MDP dynamics. The dynamics of the environment are presented in Figure 4.

Figure 1: A full transition of our mapping environment.



We can train a policy to maximize the expected reward in simulation using a prior over initial maps and the A2C algorithm from Section 3.3. Our prior over initial maps can be thought of as a representative distribution over the environments our robot might encounter. We could learn this distribution by training a generative model over a database of previously acquired maps, or hand-design it. The full learning procedure is summarized in Algorithm 1.

This learning procedure can accommodate a variety of posterior updates, priors over maps and pose/control spaces, as long as the architecture of the actor and critic is chosen correctly.

4.1 Why is the posterior update part of the environment?

This formulation treats the posterior update as part of the environment, which seems counter-intuitive as the robot normally updates its posterior itself. However, we assume we already have a good posterior update rule and we seek a complementary policy which can reduce uncertainty as quickly as possible.

5 Application: Disaster Mapping

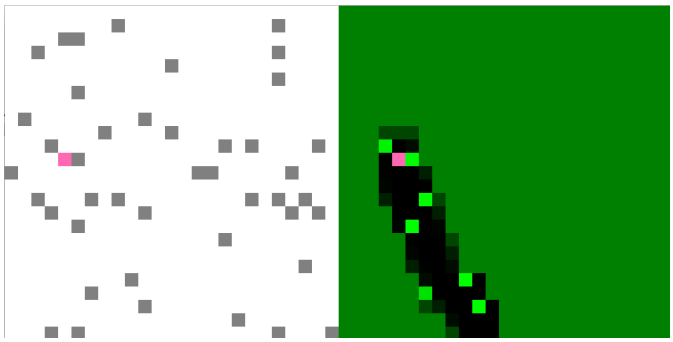
We will use the following motivating scenario to set up an environment to test our approach. A large earthquake has occurred on the San Andreas Fault, decimating many of the buildings in San Francisco. We would like to find out which buildings are still standing, but the entire city is covered in smoke so we cannot use imaging from planes or satellites. Luckily, we have a quad-copter which is able to fly through the smoke and get close to the buildings. This is a robotic mapping problem. We represent the map of buildings as a 25×25 occupancy grid. We discretize the location of the drone so that it is in a single position in the grid and assume it is flying at a fixed altitude. The drone can sense whether or not there is a building in the adjacent positions, but has a noisy sensor which is wrong 20% of the time. The drone cannot can move to adjacent positions which do not contain buildings. We have an independent Bernoulli(.1) prior over maps. Figure 5 is a visual representation of the environment.

Algorithm 1 Learning to Map

Require: $p(m)$, prior distribution over maps. $p(x)$, prior distribution over initial poses. $p(m|x, z)$, inverse sensor model. π and V, differentiable actor and critic.

1: **for** N episodes **do**
2: Sample $m \sim p(m)$
3: Simulate episode updating π and V using A2C algorithm
4: **end for**

Figure 2: The Disaster Mapping Environment. Left, the fully observable environment with the quad-copter (pink) and buildings (gray). Right, the belief state of the robot with the quad-copter (pink) where occupancy probabilities are represented as a gradient from black (0) to green (1).



5.1 Architecture Design

To use A2C, we need a differentiable policy $\pi(a_t|b_t, x_t; \theta)$ and value function $V(b_t, x_t; \theta')$. For the case of disaster mapping, our belief state is a two-dimensional grid of occupancy probability values $b_t(m_i)$ and our pose x_t is a position in that grid. In order to take advantage of the spatial representation of the belief and pose, we form a $N \times N$ matrix B_t where each entry has its corresponding belief value $b_t(m_i)$. If our robot is at a pose $x_t = (i, j)$, we form a *centered* version of the belief: a $(2N-1) \times (2N-1)$ matrix $C_t = B_t[i-(N-1) : i+(N-1), j-(N-1) : j+(N-1)]$. We pad B with 1's before forming C_t, because we are sure that positions outside the map contains obstacles. We note that the index (N, N) in C_t always represents the robot's current pose and that all of B is always present in C_t, thus making it a sufficient statistic of $[b_t, x_t]$.

This whole process can be made more concrete with an example. Suppose we have a 3×3 belief state B_0 which is .5 everywhere and 0 at the current pose $x_0 = (0, 0)$. Then C_0 is 5×5 and defined as

$$C_t = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & .5 & .5 \\ 1 & 1 & .5 & .5 & .5 \\ 1 & 1 & .5 & .5 & .5 \end{bmatrix} \quad H(C_t) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .69 & .69 \\ 0 & 0 & .69 & .69 & .69 \\ 0 & 0 & .69 & .69 & .69 \end{bmatrix}.$$

We propose to also add the point-wise entropy map of C_t as a feature channel in our state, to allow our robot to guide itself to high entropy areas, resulting in the state $s_t = [C_t, H(C_t)]$.

We experiment with three different network architectures, of increasing complexity. They all result in a tensor $\in \mathbb{R}^{256}$ which is processed in parallel by a fully-connected linear (FCL) layer followed by a soft-max for π and a FCL layer for V. All of the architectures are implemented using PyTorch [9] and the code is available upon request. A video of our model in action is available online[1].

Multi-Layer Perceptron (MLP) The input features s_t are flattened and passed through a single FCL layer to a hidden layer of size 256 followed by a rectified linear unit (ReLU) nonlinearity.

Convolutional Neural Network (CNN-MLP) The input features s_t are processed by the following convolutional layers with ReLU nonlinearities:

(1) 32 8×8 filters with stride 2
(2) 64 4×4 filters with stride 2
(3) 32 3×3 filters with stride 1
(4) A FCL layer to a hidden layer of size 256 followed by ReLU

[1] https://www.youtube.com/watch?v=6m4U7mWNOzs

Residual Network (ResNet) The input features s_t are processed by a single convolutional layer then 8 residual blocks [10] with ReLU nonlinearities and Batch Normalization [11]. Each convolutional layer in the network has 64 3×3 filters with stride 1. Finally, the flattened features are passed through a FCL layer to a hidden layer of size 256 followed by a ReLU.

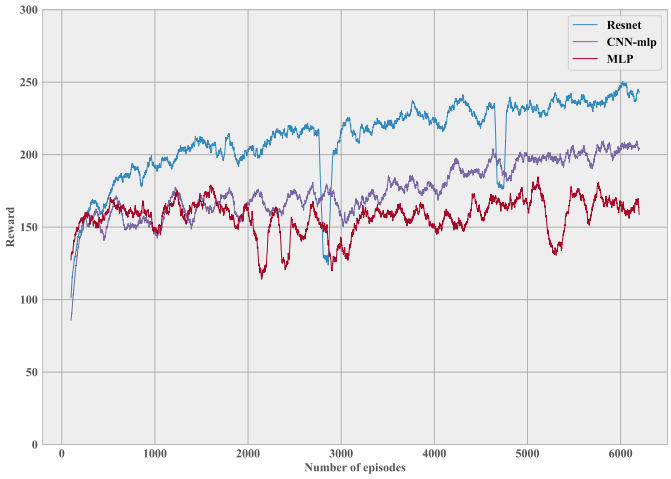
5.2 Training Details

We use a discount rate $\gamma = .99$ and episode length of 200. We use 20 steps until an A2C update. We run the learning algorithm for 5k episodes, resulting in 20k gradient updates. We use the Adam [12] optimization algorithm with a learning rate of 10^{-4} annealed every 1k episodes by a multiplicative factor of .5. We set a maximum gradient norm of an update to be 50. We use an entropy regularization constant of .010 for the A2C updates. Even though we have a .1 prior over maps, we initialize the belief map with all .5 to guarantee that entropy decreases throughout each episode.

5.3 Results

We first compare the training characteristics of the three network architectures. The training curves for the three architectures are presented in Figure 5.3, where we plot the smoothed reward over the number of training episodes. We also plot the average performance of the myopic information-based exploration strategy (as detailed in Section 2.2) as a horizontal line in the plot. The ResNet architecture performed the best, which is what we expected given its expressiveness and success in computer vision applications. Surprisingly, however, the MLP performed better than the CNN-MLP. We believe that this is because the first few layers of the CNN-MLP have filters with large stride and lose fine-grained information about the environment.

Figure 3: Training curves for different network architectures.



After the training procedure, we evaluate all of the networks, the myopic exploration strategy and a random exploration strategy on 1000 random, unseen maps. We present the average full episode rewards in Table 1. Ultimately, all three of the architectures reach the performance of the myopic exploration strategy. This leads us to believe that this training methodology could scale to more complex inverse sensor models where calculating myopic exploration strategies becomes infeasible. A video of the approach in action is available online.

Table 1: Episode Rewards averaged over 1000 out-of-sample episodes.

Approach	Performance
ResNet	240.74 ± 46.77
CNN-MLP	204.76 ± 61.15
MLP	173.01 ± 63.06
Myopic	251.07 ± 29.2
Random	92.19 ± 23.84

6 Conclusion

In this work, we introduce an architecture for efficient robotic mapping which is trained in simulation using reinforcement learning. This work is preliminary in that it only evaluated the algorithm on a scenario with a simple inverse sensor model. One could imagine switching the sensor model to be, say, a sonar sensor which depends on the robot's orientation. Then rotating the robot could become part of the action space. Since A2C can be used with continuous action spaces as long as there is an analytical log-probability of actions, we can generalize this method to robotic exploration problems which have continuous action spaces. However, it is not clear how to generalize information gain-based methods to continuous action spaces.

In future work, we would like to apply this to problem scenarios where there are multiple agents that have limited communication. This is a challenging, unsolved problem, for which we believe our approach could lead to some benefits. We would also like to integrate it with recent work on Active Neural Localization [13], leading to a fully trainable SLAM system, the holy grail of autonomous navigation robots.

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[2] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382, 1991.

[3] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[4] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.

[5] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 540–545. IEEE, 2002.

[6] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, pages 47–53. ACM, 1998.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[8] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[9] Pytorch. `http://www.pytorch.org`.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Neural active localization, 2017. `https://openreview.net/forum?id=ry6-G_66b`.