# Learning to Map from Raw Images

Devin Schwab
Carnegie Mellon University
dschwab@andrew.cmu.edu

Mohit Sharma
Carnegie Mellon University
mohits1@andrew.cmu.edu

Manasi Muglikar
Carnegie Mellon University
mmuglika@andrew.cmu.edu

## Abstract

*We explore a neural architecture for mapping environments from first person viewpoints. Our architecture uses a convolutional encoder which uses residual connections and produces a representation of the scene in the 2D image space. For free space estimates in a trajectory, we add an additional LSTM layer, allowing the network to make continuous free space estimates over a continuous trajectory.*

## 1. Introduction

One of the most fundamental skills of a mobile robot is the ability to navigate in its environment. In order to successfully navigate, the robot generally requires a map of the environment. Traditionally, robots use Simultaneous Localization and Mapping (SLAM) techniques to produce and use these maps online from live sensor data. Classical SLAM algorithms make use of distance sensors (such as LIDAR), depth cameras (such as kinect or stereo vision), and structure from motion [9, 6]. Between the methods, the sensor models vary, but the maps are all constructed using a geometric interpretation of the sensor data. These methods can provide very accurate maps, but they cannot infer what unobserved parts of the map will look like. Recent work by Gupta et. al. [3], proposes a different approach that learns to map the environment directly from first person images. Using the Stanford large-scale 3D Indoor Spaces (S3DIS) dataset [1, 2], they train a set of neural network functions that can learn to map and plan navigation paths from first-person images. Because the mapping component is learned, patterns in the dataset can be exploited, such as "doorways usually lead to open spaces". This allows the mapper to "guess" what unobserved portions of the world look like.

One potential short-coming of this work, is that they cannot learn to create a map without simultaneously learning to plan. While their planner is impressive from a machine learning perspective, it does not have the guarantees that classical planning can provide [1]. In this project, we plan to examine how the mapping component of the cited architecture can be extracted and learned independently in order to create a traditional navigation map. The advantage being that any planner can then use the resulting map.

## 2. Overview

In this project, we explore how to directly train a network that maps first-person images to a 2D top-down occupancy grid around the robot's current position. We perform our training and testing via simulated environments, which allow us to extract the ground truth occupancy grid. The goal is to learn a network that can not only accurately estimate free-space/occupied space for parts of the map currently visible to the robot, but to also "guess" and what unseen parts are possibly free/occupied.

Our mapping component is based on the mapping portion of the network from [3]. In [3] the authors split the network into a mapping component and a planning component. However, the mapping portion gets its error signal by back-propogating through the planner. Therefore, the learned mapper cannot be used with different planners. Also, there is no guarantee that the mapping portion learns a representation that is compatible with other planning algorithms.

In the next sections we describe the two architectures tested and their training approaches. We also describe how the simulation environment was created, and how the training and testing data was constructed.

## 3. Architecture and Training

Similar to [3] we use an autoencoder architecture to predict the free space estimate from raw images. We use a pretrained ResNet [4] architecture for our encoder. This encoded representation is passed through an upsampling decoder which gives us the final free space estimate for every image. For free space estimates in a trajectory, we add an additional LSTM layer above the upsampling layer. This allows the network to make continuous free space estimates over a continuous trajectory.

[1] Such as guaranteeing the shortest path to the goal.

We use the Adam optimizer for training with a learning rate of 0.001. We experiment with two different loss functions *i.e.* Mean Squared error (MSE) and Binary Cross Entropy (BCE). In our experiments we find the MSE Loss to work better than BCE. We believe this happens because of vanishing gradients while using the BCE loss, which apparently is not manifested while using the MSE loss.

## 4. Data

All of our training data comes from simulated environments. We use the Gazebo simulation package [8]. We have constructed a cylindrical robot model with a simulated camera mounted on the front. Using a custom ROS and Gazebo plugin we can drive the robot around and get the exact coordinates of the robot. This is important, as our approach, like [3] assumes that perfect odometry information is available. The simulated camera outputs a 224x224 color image.

The simulated worlds and ground truth maps are constructed from the Stanford large-scale 3D Indoor Spaces (S3DIS) dataset [1, 2]. This dataset contains meshes of 6 floors of the Computer Science building at Stanford. These meshes were converted into Gazebo models that the simulated robot could drive around in and collect data. Figure 1 shows some examples of what the simulated robot might see when moving around the environment.

Figure 1. Example views of our simulated robot.

To construct the ground truth map the meshes were voxelized via the binvox tool [7]. These voxels are then converted to a ROS compatible Octomap [5] tree. The occupied voxels are then projected down to the 2D plane. Only occupied voxels slightly above the floor plane and up to the height of the robot are projected down. This is to avoid the ceiling and floor voxels from marking the entire map as occupied. The resulting map is 2D occupancy grid with each cell containing a floating point number $[0, 1]$, representing the probability a cell is occupied. Where $0$ is free, $1$ is occupied.

The resulting map has hollow walls and small holes in parts of the wall. The hollow walls is an artifact of the voxelization process. The holes in the wall are due to holes in the original mesh due to inaccuracies in the mesh creation process. The map also includes cells from outside the building. We modify the map manually in a photo editor. Inside parts of the wall are filled in as occupied. Holes are manually filled. Any map cells outside of the building, or in untraversable areas (such as stairwells) are marked with a value representing unknown ($0.5$ using the previously described representation). The resolution of all of the maps is approximately 2.5cm per cell. Figure 2 shows an example of what a ground truth map looks like.

We collect trajectories of data using this ground truth map and the simulated Gazebo environment. We first dilate all obstacles in the map so that any non-occupied cell is safe for the robot to be in without clipping through the environment mesh. Unlike [3] we are not concerned with planning, so to collect data we generate random goal positions. We then run A* to this random goal from the current position with three actions: forward, turn clockwise $90°$, turn counter-clockwise $90°$. Once a path is found, the robot executes the path logging each coordinate, orientation and image to a file.

Figure 2. Example of occupancy grid map used for training. Map shown is for area 4. Left is raw map, right is the manually cleaned up map.

When training, we lookup the coordinate and rotation associated with the saved image. We then extract a 32x32 patch of the occupancy grid map and rotate it so that the patch is in a consistent orientation with respect to the robot's orientation at the time the image was saved. This patch is used as the prediction label when calculated the loss terms.

## 5. Experiments

As discussed above, we experiment with two different settings. Initially, we test free space prediction given single image, as observed by the robot at any point of time. We use the top-down free space estimate as the ground truth. One of the caveats with this setting is that the robot has no way of knowing what is behind it hence the predicted free space estimate can only be correct for the robot's frontal view. To alleviate the error in this setting we combine the free space estimates from all the different points in the area and normalize them to get a final free space estimate for the whole area. We normalize by adding up all of the patches at their respective world frame coordinates and then normalizing the entire image. Figures 3 and 4 shows the results for this experiment setting.

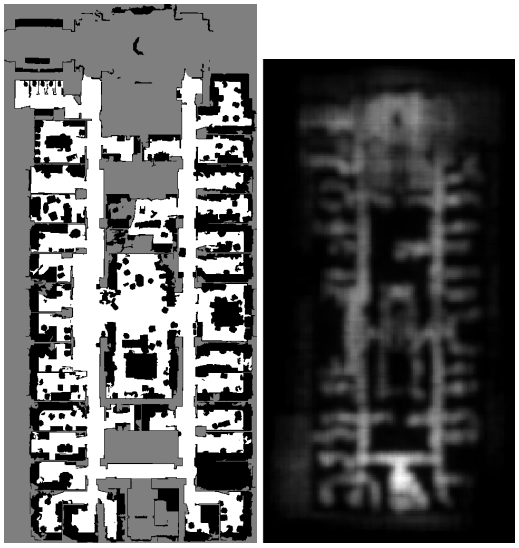

Figure 3. Area 1 predictions (Training dataset) (Left) Ground Truth (Right) Predicted Map

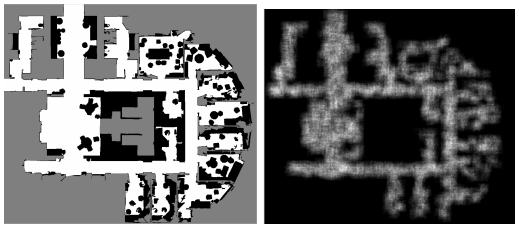

Figure 4. Area 3 predictions (Test dataset) (Left) Ground Truth (Right) Predicted Map

Figure 3 shows the result of overfitting to the training dataset. While this doesn't show that the learned architecture generalizes, it does show how good this architecture can possibly be at predicting free/occupied space. The network is good at capturing coarse features, but is fuzzy around the edges and fails to capture smaller details. The fuzziness is likely because we used MSE, however, we did not get any good results with BCE. The loss of fine-grained features is probably due to the noiseness of the predictions. Each individual output prediction is quite noisy, but when averaged together, the results are reasonable.

Figure 4 shows the result of applying the same network to the test dataset. Even though the network has never seen any images from this environment during training, it is still able to capture most of the coarse features of the map. The network is a little bit noisier though, so there are lots of little blocks marked as occupied, that in reality are completely free. However, looking at the middle left of the image, we can see the network correctly predicts the table and chair area as occupied. Similarly, it captures the large conference table in the upper right corner room of the map.

To alleviate the issues with the above experiment setting, we experiment with free space estimates for fixed length trajectories. We sample these trajectories from random walks that were made in the environment. We use a trajectory length of 10 for our experiment settings. At each point of time we use the previously predicted free space estimate along with the current view to predict the estimate at this point step. We experiment with both MSE and BCE loss functions which are calculated at each time step of the trajectory. Figure 5 shows the result for these experiment settings.

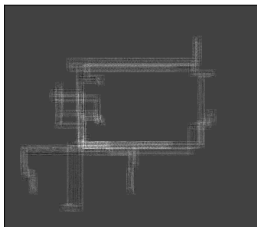

Figure 5. Area 3 predictions from LSTM architecture. Compare with ground truth from figure 4.

Unfortunately, we did not have good results at all with the LSTM network. Looking at individual patches, the results are mostly just noise. So when the patches are pieced together the output doesn't show much. This network basically fails to capture any of the features of the ground truth map. We are not particularly surprised by these results, as looking at the loss curves while training the LSTM showed very small improvements. We examined the gradients while training and found that they were quite small, so it is likely the network did not learn much. Perhaps with signficantly more training or a different loss function, it would be possible to get better results.

## 6. Conclusion

In this project we explored architectures and training methods that can take in a first-person image and output a 2D top-down occupancy grid. We created a custom simulation using an open dataset, collected training data, and trained multiple networks with different loss functions. We found that even with no memory, it is possible for our resnet architecture to capture coarse features of a map when trained with MSE loss. We also attempted to add memory to the network in order to allow the network to "remember" what was behind it, so that its free space estimates would improve. We were unable to find a way to make this work. We believe the network did not learn much due to vanishing gradients. Future work should look into finding a method of giving the network memory, while retaining at least the same performance as the memoryless network. However, we believe that the results from the memoryless network are quite promising and show that this approach has merit.

## References


[1] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017.

[2] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[3] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *CoRR*, 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[6] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. IEEE, 2013.

[7] P. Min. Binvox,. *A 3D Mesh Voxelizer*, 2005.

[8] O. S. R. F. (OSRF). Gazebo, February 2017.

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