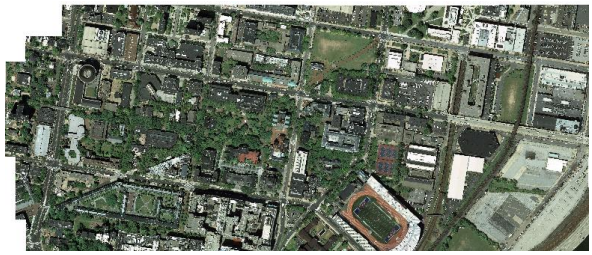
Path Planning with Imitation Learning

Yiren Lu
GRASP Lab, University of Pennsylvania
April 2016

1 Introduction

In ESE 650 5th project, we did path planning on University of Pennsylvania's high resolution (12480×5344) aerial photo with image processing, Dijstra's algorithm and imitation learning [1]. First, I extracted image feature maps using image processing techniques. And then, I used Dijstra's algorithm and imitation learning to iteratively learn the cost map from feature maps.



2 Dataset and Image Processing

Since the aerial image is so big, I downsample the image by 8 and perform training and testing on that. Initially for this project, I used less-detailed feature set including RGB/HSV values, edges, and some GMM probability maps. However, the results are not as good as expected. Clearly, since the learning part largely depends on the features, they are critical to this project. Later I did more feature engineering. I thresheld with pixels of different color spaces, including HSV, L*A*B, YCBCR to separate green areas, roads, shadows and rooftops. More detailed feature maps were included in the *slides* attached. Edges are good features for path planning, I used Canny, Prewitt and Sobel features. I also use kmeans algorithms to segment the map, and extraced nice side walk mask. On the other hand, I also extracted green areas, roads and buildings probability maps using Gaussian Mixture Model.

3 Feature Maps

By combining different feature maps together, I could basically reconstruct the area mask of driving and walking by hand designed features as shown in the images.

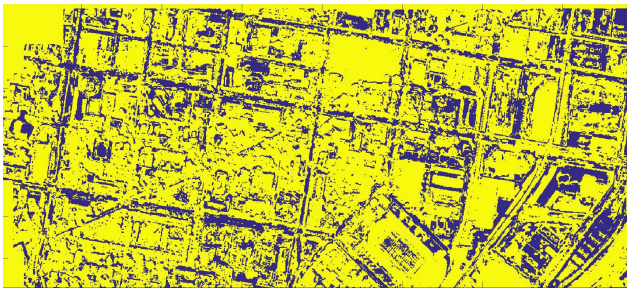


Figure 1: Drive area map

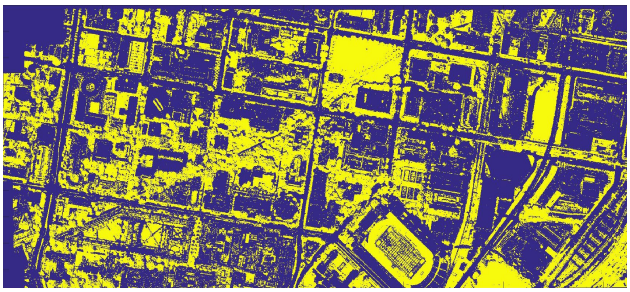


Figure 2: Walk area map

4 Imitation Learning

After extracted multiple feature maps, we then combine them by exponential of weighted sum. The basic path planning procedure is to compute the cost to go function

using cost map by a dynamic programming approach, Dijstra's algorithm. And then use cost to go function to greedy plan the path. The cost function is defined by

$$C(x,y) = e^{\sum_{k=1}^{K} a_k F_k(x,y)}$$

Where, a_k is the weight for the kth feature map F_k. Since in the practical learning process, the weights a are not restricted to positive, but the Dijstra's algorithm can only take positive cost values, we exponentiated the cost map to make all the cost positive.
After that, I did gradient descent based imitation learning iteratively compute the optimal cost map.

$$\text{Minimize } J = \sum_{(x,y)^*} C(x,y)^* - \sum_{(x,y)} C(x,y)$$

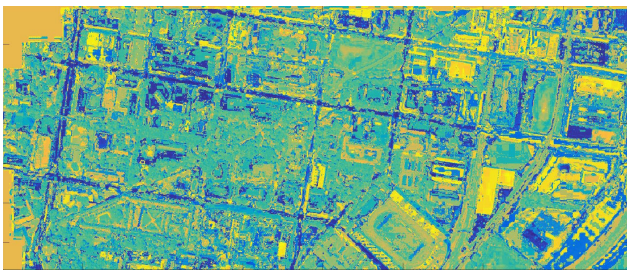
Where, $\sum_{(x,y)^*} C(x,y)^*$ is the total cost of the training path, $\sum_{(x,y)} C(x,y)$ is the tatal cost of the predicted path. In order to learn the weights a, we used gradient descent method.

$$\frac{\partial J}{\partial a_k} = \sum_{(x,y)^*} e^{\sum_{k=1}^{K} a_k F_k(x,y)^*} - \sum_{(x,y)} F_k(x,y) e^{\sum_{k=1}^{K} a_k F_k(x,y)}$$

$$\frac{\partial J}{\partial a_k} = \sum_{(x,y)^*} C(x,y)^* - \sum_{(x,y)} F_k(x,y) C(x,y)$$

$$a_k = a_k - \mu \frac{\partial J}{\partial a_k}$$

For the training set, I labeled 30 samples for driving and 30 for walking. When I used less-detailed feature maps, the cost maps learning was as follows



It basically worked, however, without detailed features about buildings, shadows, etc, the planner gives paths went across buildings from time to time. Further, I found that without stop criterion or early stop, the gradient descent tends to over fit the training data by zering out all the features with high cost under the predicted paths as well as lowering all the weights in order to obtain a lower cost.
Since my hand-designed feature maps are similiar to the expected cost maps, the predicted paths quickly converged to the training paths.
For training, here are some tips

- I normalized the feature maps into between 0 and 1, since the initialization of features could largely influence the results of non-convex optimization.
- The learning rate could vary for different setting of weights and cost maps.
- Normalizing the weights in each iteration to some extent prevent overfitting.

5 Results and Discussion

- There is a bug in the given code getMapCellsFromRay.cpp, the output paths are always ordered by x from lower to higher, which is bad for applications requiring sequential order. I reimplemented this function and included it in the code submission *mex* folder.

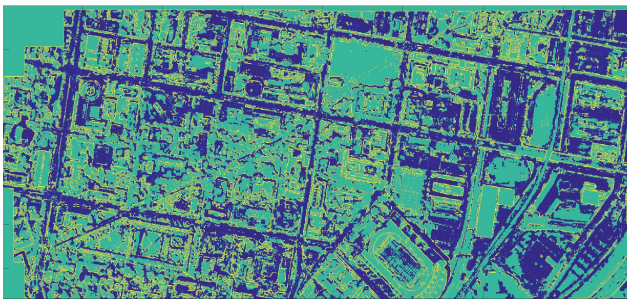


Figure 3: Final cost map for driving

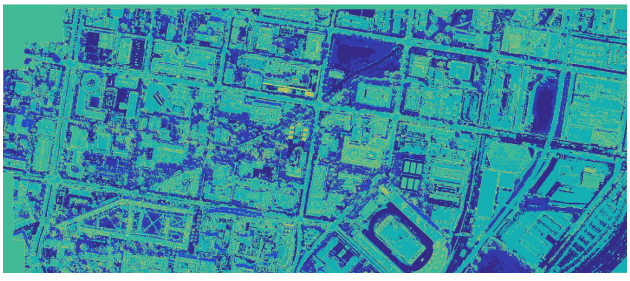


Figure 4: Final cost map for walking

Acknowledgement

Thanks Dr. Lee for designing and preparing this project and thanks TAs for timely responses to our questions.

References

[1] RATLIFF, N. D., SILVER, D., AND BAGNELL, J. A. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots 27*, 1 (2009), 25–53.

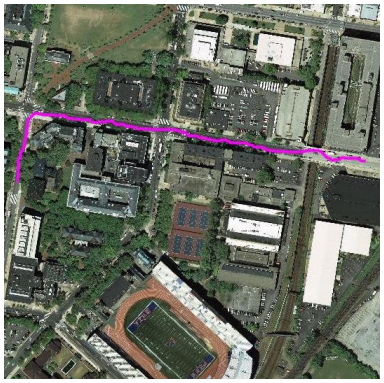


Figure 5: Drive testcase 1

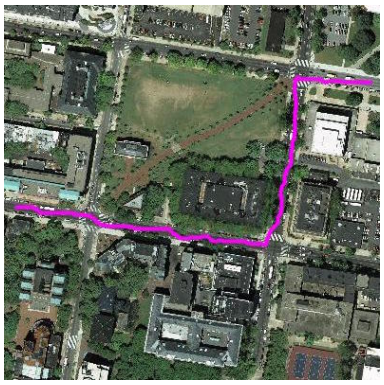


Figure 6: Drive testcase 2

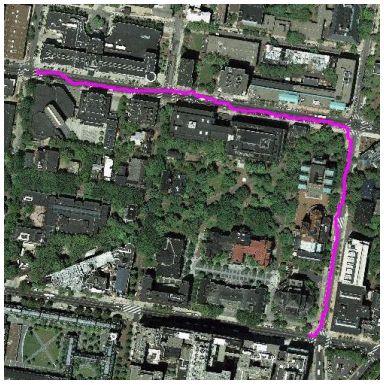


Figure 7: Drive testcase 3

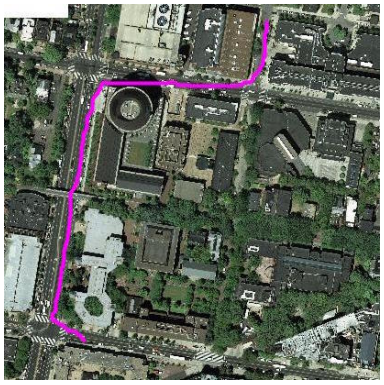


Figure 8: Drive testcase 4

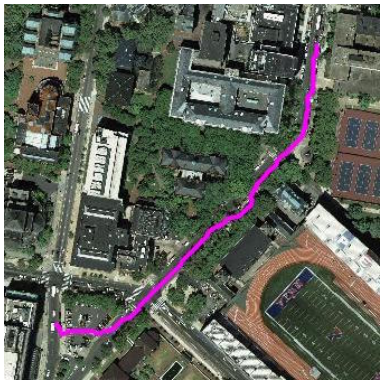


Figure 9: Drive testcase 5

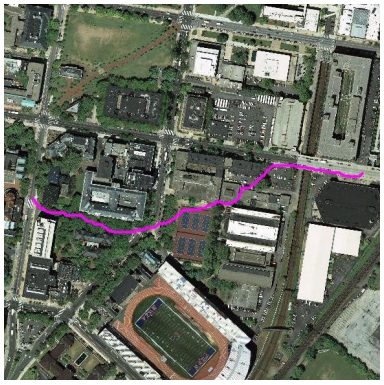


Figure 10: Walk testcase 1

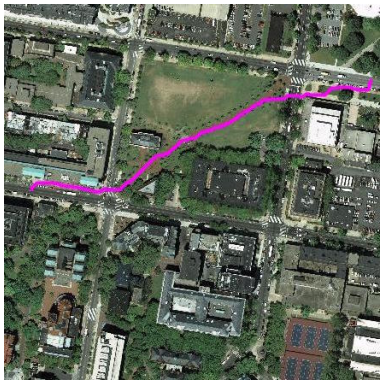


Figure 11: Walk testcase 2

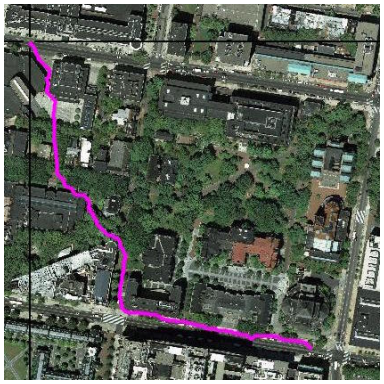


Figure 12: Walk testcase 3

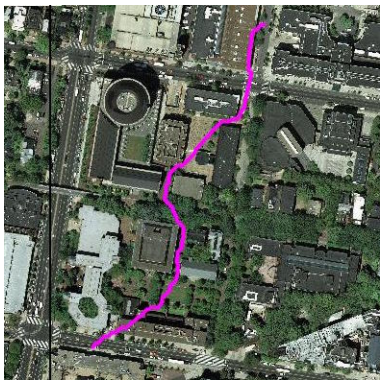


Figure 13: Walk testcase 4

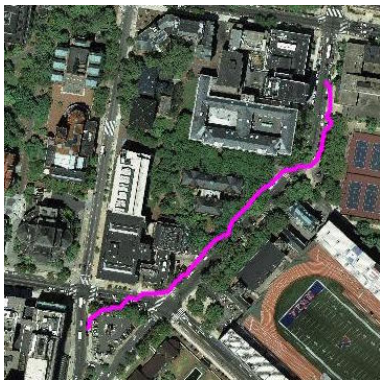


Figure 14: Walk testcase 5