Feasible Path Planning for Cars and Pedestrians using Imitation Learning

Nitin J. Sanket
School of Engineering and
Applied Science
University of Pennsylvania
Email: nitinsan@seas.upenn.edu

***Abstract*—This project presents an approach for learning a cost map as a linear combination of myriad of features constituting color, texture and shape of an aerial image of UPenn's campus. The alogrithm tires to mimze the cost such that the paths generated are similar to the ones drawn by an expert in both cases. The undeylying idea is to learn the valid and invalid regions for driving/walking.**

I. Problem Statement

Learn a cost map to give reasonably valid paths for driving and walking, i.e., a car cannot drive on buildings and so on. Assume a holonomic car and person and no one-ways. Also neglect the flyovers.

II. Assumptions Made

A. Car Path Planner

Some assumptions made about the car path planner were:

1) The car can drive on roads
2) The car cannot distinguish between roads and pavements and hence can drive on pavements (though I have tried to avoid pavements, the car sometimes goes over it)
3) The car cannot drive through buildings or trees or internal roads like locust walk

B. Pedestrian Path Planner

Some assumptions made about the pedestrian path planner were:

1) The pedestrian can walk on roads (though unsafe he can, not that he should prefer to)
2) The pedestrian can walk on pavements
3) The pedestrian can walk near trees or internal roads like locust walk
4) The pedestrian cannot walk through buildings

III. Feature Maps

A multi-modal gaussian distribution was used to segment out the YCbCr color space to extract different parts of the map. The whole idea was to extract different colored roofs, buildings, trees and roofs. A simple approach of manually combining these features one should be able to extract the roads. The colorThresholder app on Matlab on YCbCr color space was used. The app provides a GUI for thresholding colors of the image.

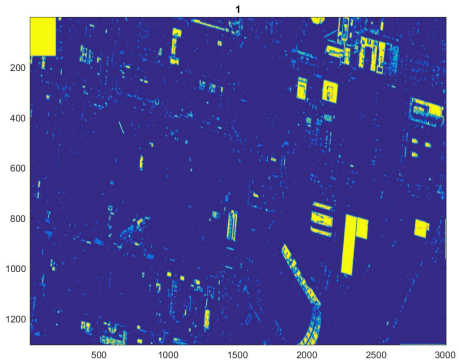

Fig. 1: Feature Layer 1.

The feature maps for car path planner are shown in Figs. 10 to 14. One can clearly observe the presence of Bright Roofs in Fig. 10. Fig. 2 shows the extraction of shadows, this layer serves as a positive sample for the valid paths because due to the angle of the photo the shadows always fall on the roads and pavements. Figs. 3 and 4 depict the green regions like trees, grass with some more roofs. Red colored/ brown colored regions like pathways are seen on Fig. 5. Fig. 6 shows a combination of roads and roofs, this feature-set acts as the roads when the roofs are subtracted from it. Shape statistics like eccentricity, solidity and area of the blobs are used to remove the road features. Figs. 7 to Fig. 14 show more building features. All the extracted building features are combined into a feature layer with logical OR operation. Fig. 15 shows the combined buildings feature by using convex hull and regionprops. Finally, to make the potential functions sort-of continunous/smooth, a gaussian blur with $\sigma = 2$ is used.

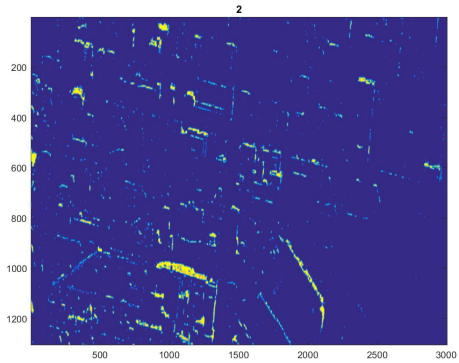

Fig. 2: Feature Layer 2.

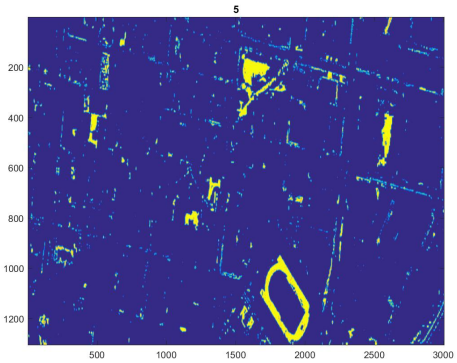

Fig. 5: Feature Layer 5.

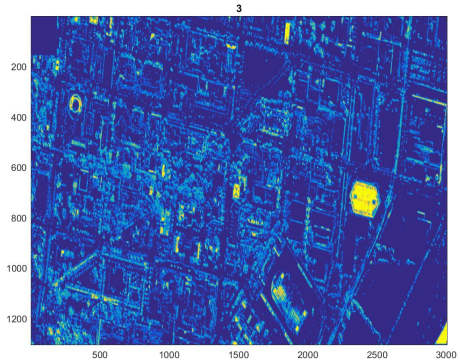

Fig. 3: Feature Layer 3.

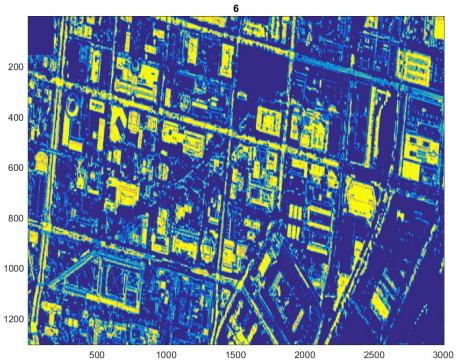

Fig. 6: Feature Layer 6.

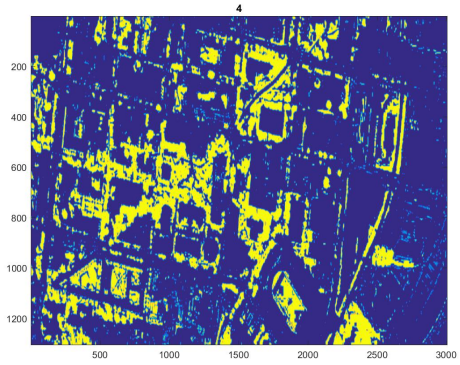

Fig. 4: Feature Layer 4.

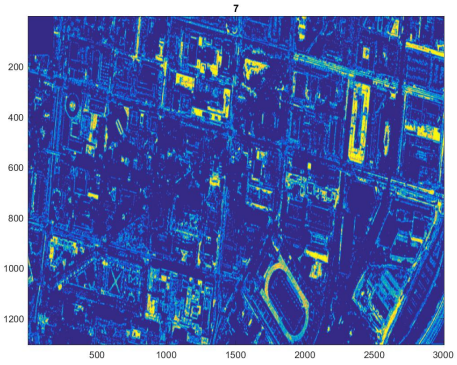

Fig. 7: Feature Layer 7.

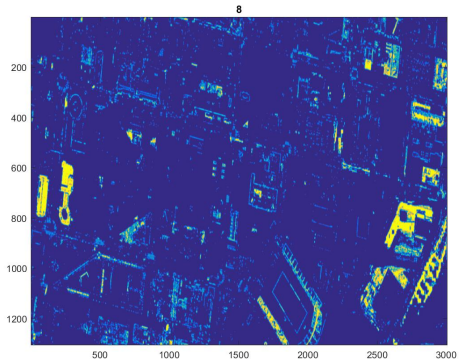

Fig. 8: Feature Layer 8.

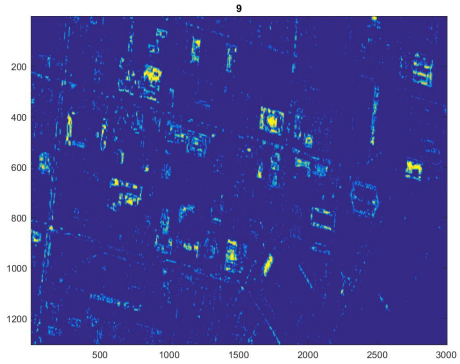

Fig. 9: Feature Layer 9.

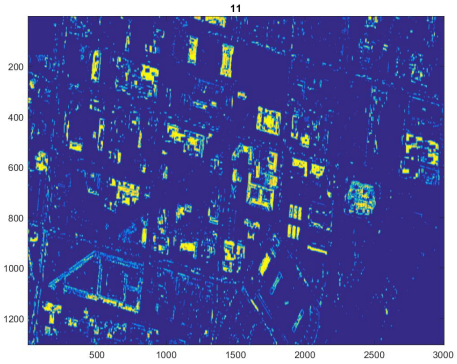

Fig. 11: Feature Layer 11.

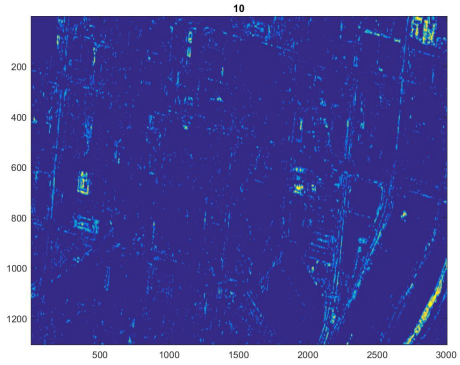

Fig. 10: Feature Layer 10.

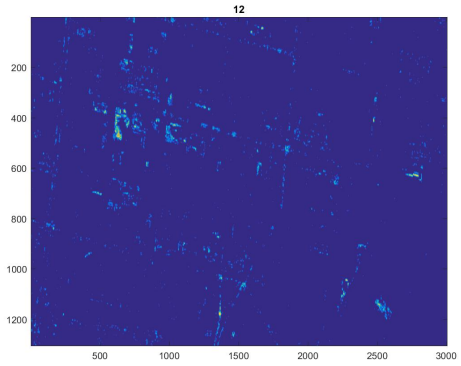

Fig. 12: Feature Layer 12.

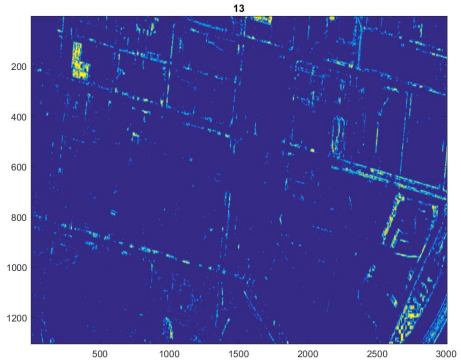

Fig. 13: Feature Layer 13.

The feature maps for car path planner are shown in Figs. 25 to 30.

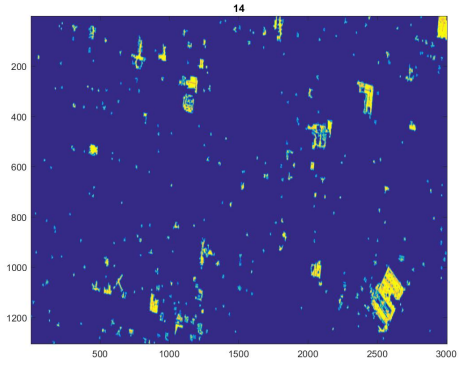

Fig. 14: Feature Layer 14.

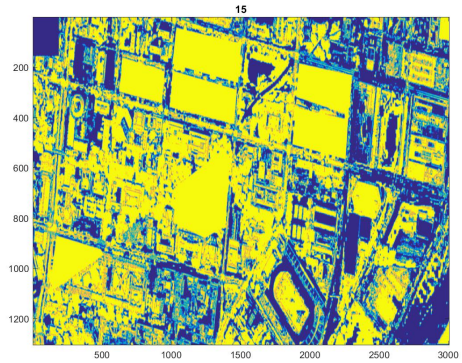

Fig. 15: Feature Layer 15.

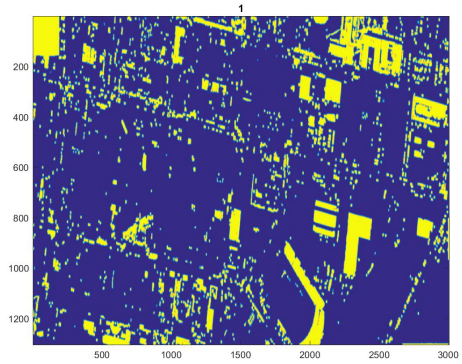

Fig. 16: Feature Layer 1.

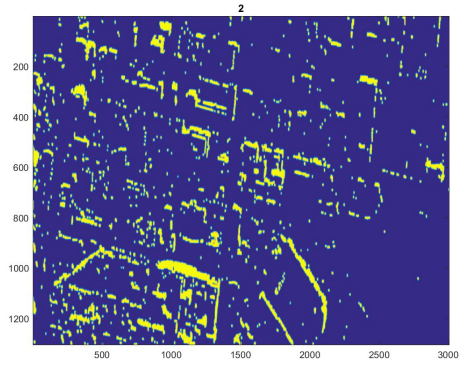

Fig. 17: Feature Layer 2.

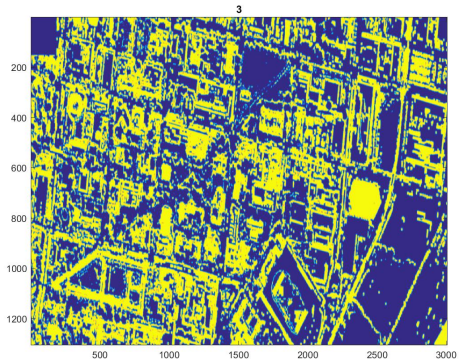

Fig. 18: Feature Layer 3.

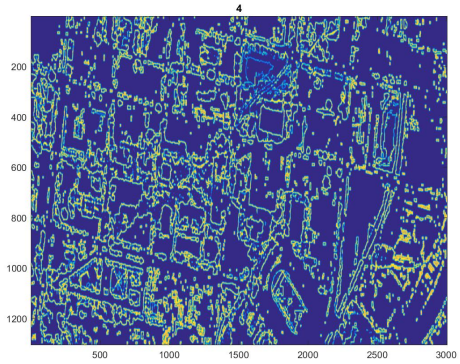

Fig. 19: Feature Layer 4.

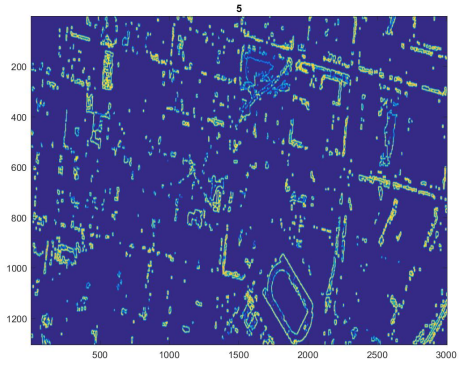

Fig. 20: Feature Layer 5.

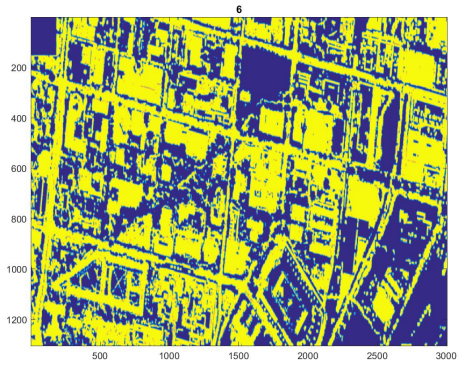

Fig. 21: Feature Layer 6.

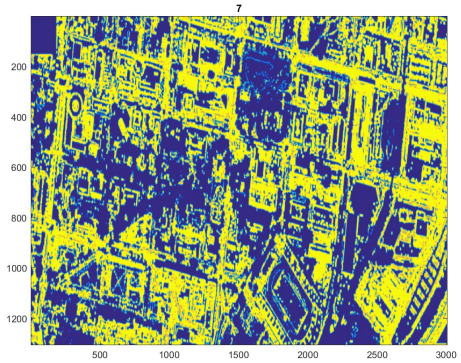

Fig. 22: Feature Layer 7.

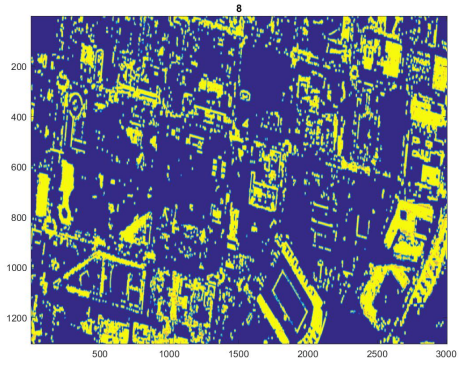

Fig. 23: Feature Layer 8.

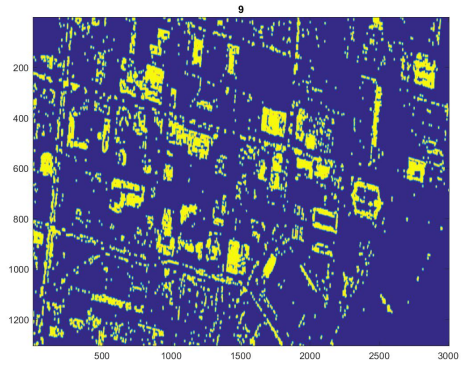

Fig. 24: Feature Layer 9.

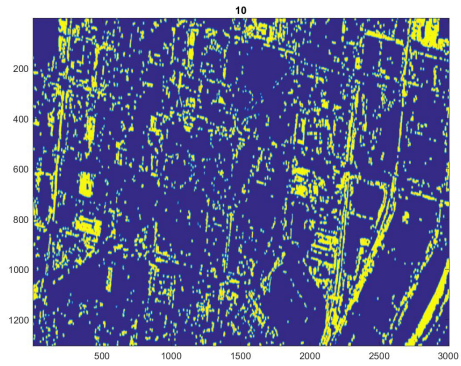

Fig. 25: Feature Layer 10.

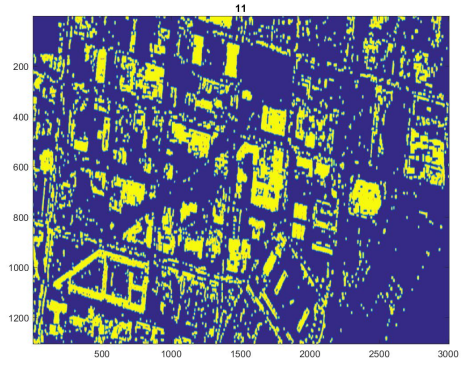

Fig. 26: Feature Layer 11.

IV. Other Features which I tried but did not end up using

A. K-Means Clustering

I tried K-means clustering on a*b* color space from La*b* color space and the results looked promising, however because the colorThresholder was more robust, K-means features were not used. A sample image of the clustering is shown in Fig. 31. Clearly the clusters are not orthogonal and I was aiming to sort of get orthogonal features.

B. Gaussian Mixture Model for clustering

To obtain a continuous/ soft clustering GMM was used on different color-spaces. The clustering was not as robust as expected and I did not end up using these features. A sample feature map with posterior probabilities for cluster 1 of 10 clusters is shown in Fig. 32.

C. Euclidian Distance on color segmentation for a continuous feature map

The idea here is while threhsolding the colormap, we have a minimum and a maximum value of threshold for each channel.

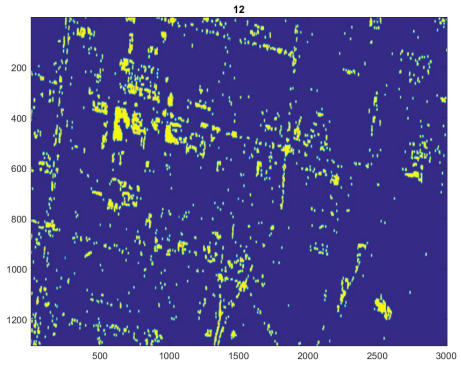

Fig. 27: Feature Layer 12.

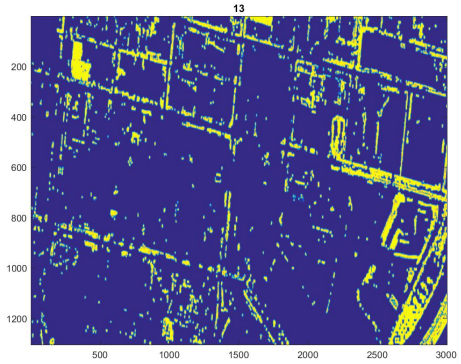

Fig. 28: Feature Layer 13.

The value for each pixel is computed as the average eucledian distance to the mean threshold value for each channel. This is summarized below:

$$EDMap = \sqrt{\frac{\left(\sum_{i=1}^{N} C_i - \frac{Max_i + Min_i}{2}\right)^2}{N}}$$

Here, N is the number of channels. Because the small values got added up for where the cost should've been low this resulted in sub-optimal paths. Refer to Fig. 33.

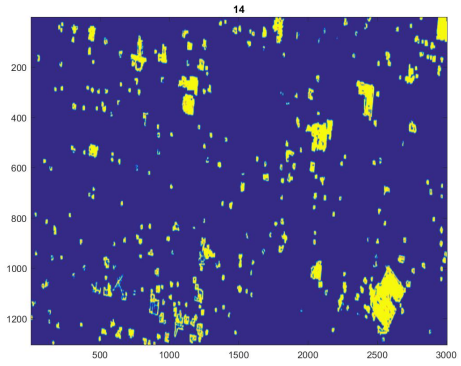

Fig. 29: Feature Layer 14.

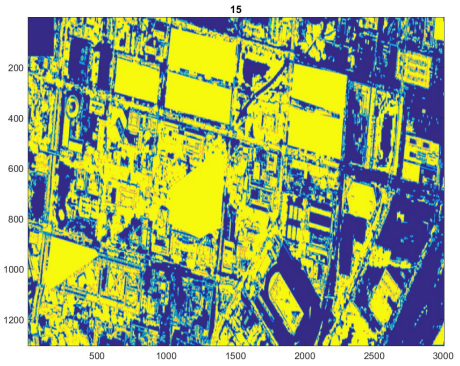

Fig. 30: Feature Layer 15.

D. RGB Color Diagonal Image to extract gray color

Because, the color gray is doesn't have chroma component it is more a function of intensity. So it can be modelled using a single degree of freedom. So all the pixels lying on the body diagonal of the RGB color cube are gray. Because the real-world data is never perfect, i.e., it is noisy, a cylindrical potion around the body diagonal was treated as gray bands. This was extracted using the same formula used in the previous subsection on the RGB color-space and thresholding on a value. This feature was quite useless as almost the whole image fell into this band. The roads and roofs were almost the same shade of gray. Refer to Fig. 34.

E. Standard Deviation Filter for extracting texture

Here, the local standard deviation in a window was computed and used as the feature map, the hope was to capture texture information and that roads generally are smooth and have low texture, hence low standard deviation. This feature was expected to work well, however, due to the presence of zebra crossing and cars on the road this feature was found to be sort of useless as the weight associated with it was very small. The output of standard deviation filter is shown in Fig. 35.

F. Entropy Filter for extracting texture

Here, the local entropy in a window was computed and used as the feature map, the hope was to capture texture information and that roads generally are smooth and have low texture, hence low entropy as it resembles a peak distribution along one color and texture. This feature was expected to work well, however, due to the presence of zebra crossing and cars on the road this feature was found to be sort of useless as the weight associated with it was very small. The output of standard deviation filter is shown in Fig. 36.

G. Range Filter for extracting texture

Here, the local range in a window was computed and used as the feature map (here, range is defined as the absolute value of difference between maximum and minimum value in the window), the hope was to capture texture information and that

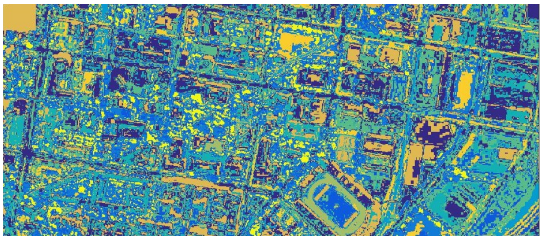

Fig. 31: K-Means on a*b* color space.

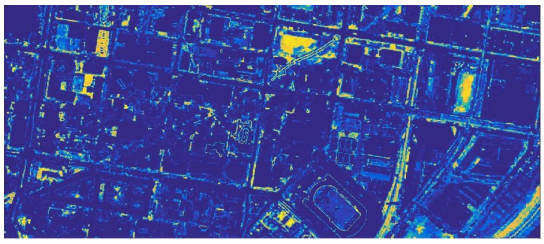

Fig. 32: GMM on a*b* color space.

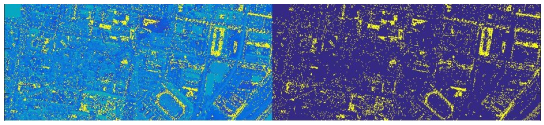

Fig. 33: Left: Eucledian Distance on YCbCr color space, Right: Binary Map obtained from band-pass filtering.

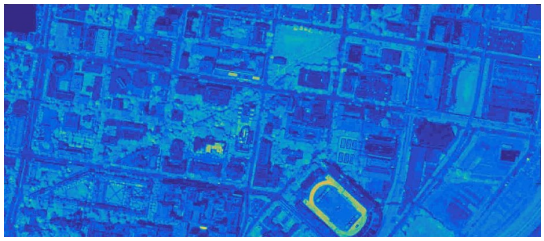

Fig. 34: Extracting gray region based on euclidian distance in RGB color space.

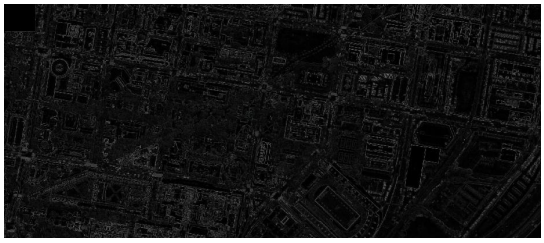

Fig. 35: Standard Deviation Filter output.

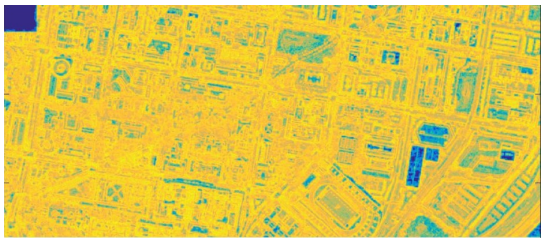

Fig. 36: Entropy Filter output.

Fig. 37: Range Filter output.

roads generally are smooth and have low texture, hence low range. This feature was expected to work well, however, due to the presence of zebra crossing and cars on the road this feature was found to be sort of useless as the weight associated with it was very small. The output of standard deviation filter is shown in Fig. 37.

H. Image Gradient

Image gradients along x and y direction were calculated and the magnitude was computed as the square root of sum of squares of gradients along x and y direction. The idea was to have low gradient along the roads, but because of cars the roads had almost the same gradient value as roof tops. Hence this feature was not used. The image gradient magnitude is shown in Fig. 38.

I. Stationary Wavelet Transform (SWT)

SWT can split the image into 4 sub-bands using the haar wavelet, namely, cA, cH, cV and cD. cA has the intensity value, cH has the horizontal edges, cV has the vertical edges and cD has the diagonal edges. Because the image is tilted at an angle, SWT features were coupled and were not used. This could've been fixed by rotating the image which I did not think of when writing the code. All the SWT components are shown in Fig. 39.

J. Discrete Cosine Transform (DCT)

DCT can be used to split the image into various frequency bands. Lower frequencies are placed on the left top corner of the DCT matrix. So we can make the right bottom corner of the DCT matrix zero remove high frequency components. Sadly, due to JPEG compression there were a lot of artifacts as JPEG also uses DCT for compression. Hence, this feature was not used. DCT reconstruction of the image using only the top 50×50 pixel grid is shown in Fig. 40.

V. Gabor Filters

Gabor filter is a linear multi-scale and oriented filter bank. Scale 4 and 8 and angles 0 and 90 degrees were used to obtain the Feature shown in Fig. 41. Though, the orinented edges were helpful, these were not used because of speed and memory constraints when working with large number of filters. Phase data was also very noisy.

VI. Imitation Learning

The overall algorithm for Imitation Learning [1] is described in Algorithm 1.

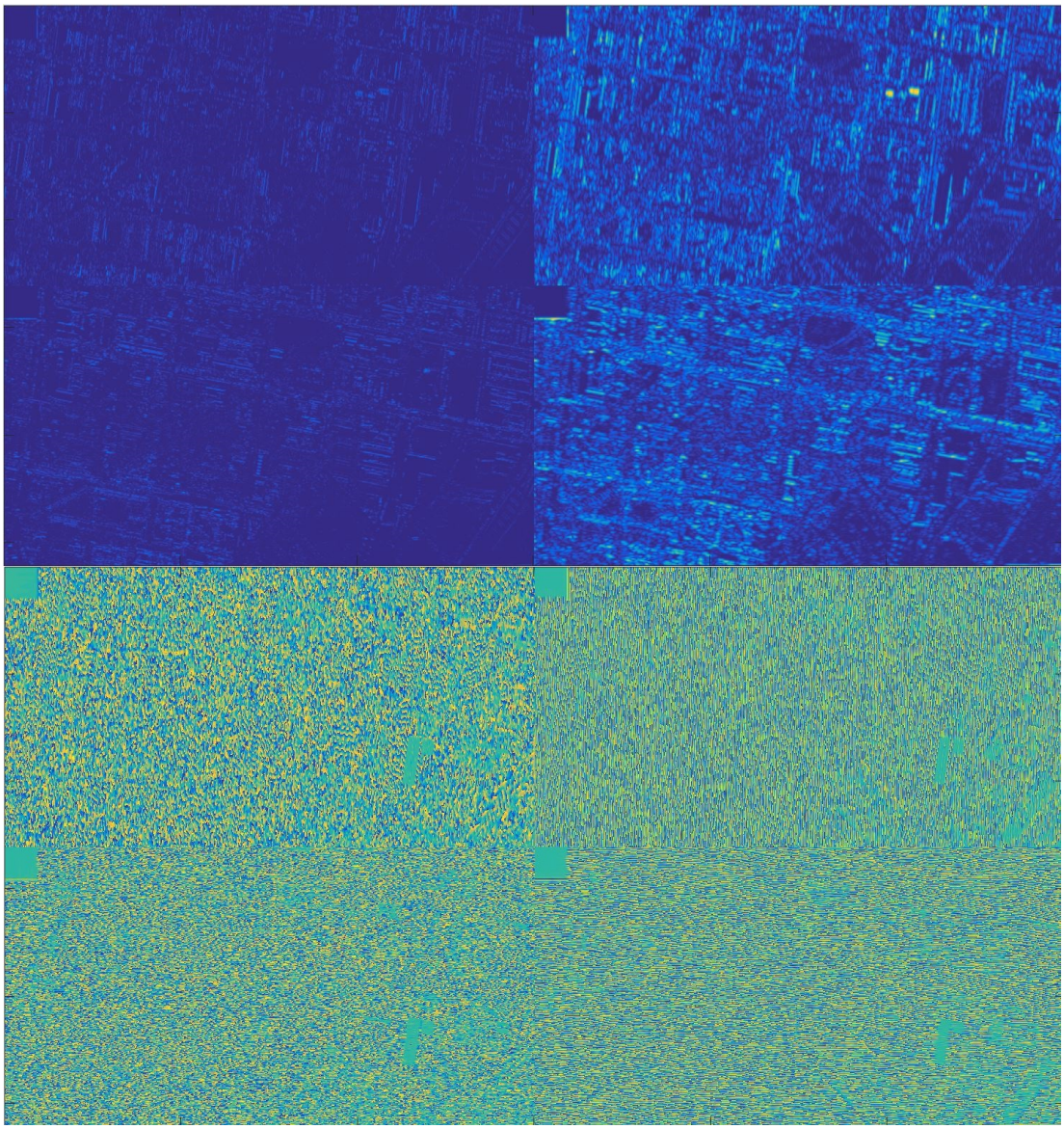

Fig. 41: Gabor filter-bank output for scales 4,8 and angles 0 and 90 degrees. Top 2 rows show magnitudes and bottom 2 rows show phase.

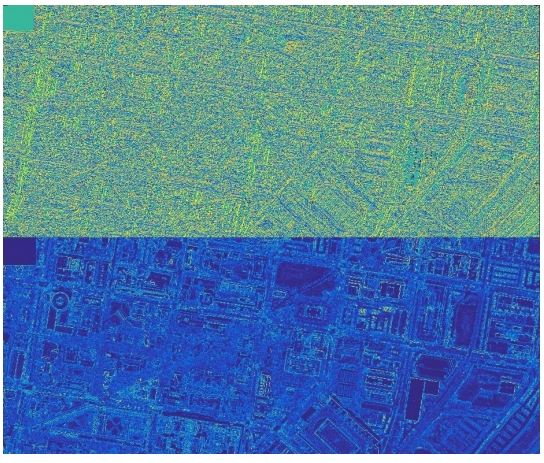

Fig. 38: Image output, top: phase, bottom: magnitude.

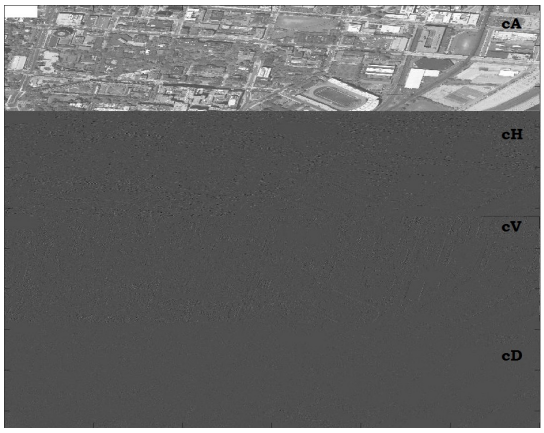

Fig. 39: Stationary Wavelet Transform Output.

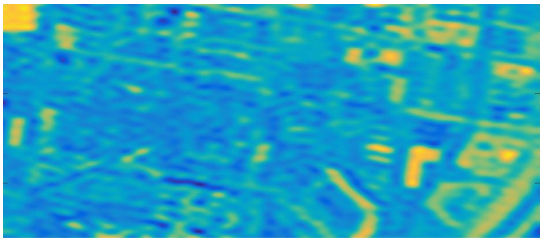

Fig. 40: DCT Output just using the top left 50×50 block from DCT matrix.

Data: P_{ex}, w, F, MaxIter, Thld, JPrev
Result: Optimized w
Compute Initial cost map,
$C(x,y) = e^{\sum_i w_i F_i(x,y)}$
while *t¡MaxIter — ΔJ¿Thld* **do**
  for $i \in [1, NumFeat]$
  do
    % Compute Dijkstra Path P_{ex}=Dijkstra(start, goal) % Compute gradients for each feature map

$$\frac{\partial J}{\partial w_i} = \sum_{P_{ex}(x,y)} F_i(x,y) e^{\sum_i w_i F_i(x,y)} - \sum_{P_{dijk}(x,y)} F_i(x,y) e^{\sum_i w_i F_i(x,y)}$$

    % Update weights

$$w_i = w_i - \alpha \frac{\partial J}{\partial w_i}$$

$$\Delta J = J - JPrev$$

$$JPrev = J$$

  end
end

Algorithm 1: Imitation Learning Procedure

A. *Convergence of gradient descent*

To make the code converge fast, a manual-adaptive approach for changing learning rate is performed. Gradient descent is run for 10 iterations with a learning rate of 10^{-3} (all features are scaled between 0 and 1). The next few iterations are run with a learning rate of 10^{-4} where 50 is chosen as the maximum number of iterations and the algorithm terminates if the change in J values between 2 iterations is less than 0.1. A graph of variation of weights for car path training is shown in Fig. 42. We can clearly see that the weights sort of try to converge to a uniform value due to the absence of regularization. Also, adding boundary costs along the training paths would've avoided this effect. Nevertheless, the J value keeps decaying and becomes almost constant which shows that the algorithm is converging to a local minima. This is shown in Fig. 43. Similar plots of weights and J value for the pedestrian paths are shown in Figs. 44 and 45 respectively. They follow the similar trend observed in the car case.

The outputs of the paths for car and pedestrian for the first run with learning rate 10^{-3} are shown in Figs. 46 and 47 respectively. Also, the outputs of the paths for car and

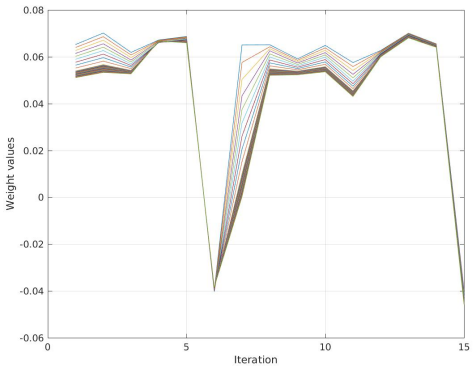

Fig. 42: Variation of weights over iterations for car path training.

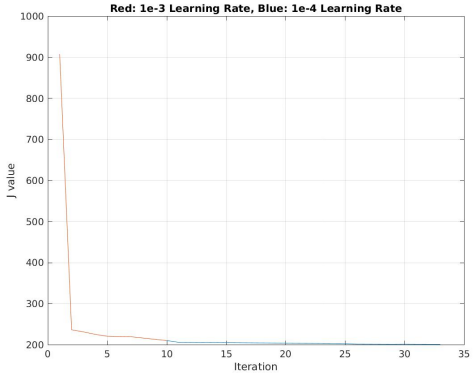

Fig. 43: Variation of J value over iterations for car path training.

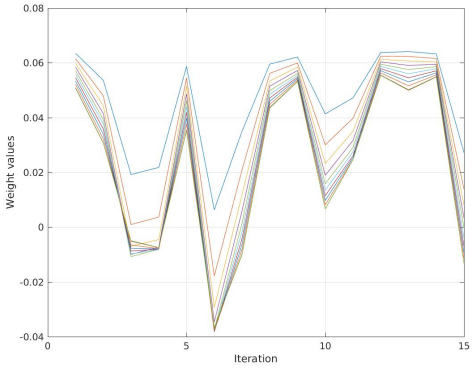

Fig. 44: Variation of weights over iterations for pedestrian path training.

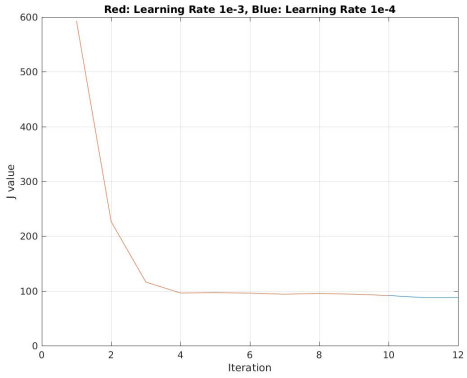

Fig. 45: Variation of J value over iterations for pedestrian path training.

pedestrian for the fine tuning with learning rate 10^{-4} are shown in Figs. 48 and 49 respectively. One can observe that the paths are much closer to the expert paths and are more smoother.

B. Final Cost Maps

The final generated cost maps for car and pedestrian path planner are shown in Figs. 50 and 51 respectively. In the car cost map we can clearly observe that roads have lower cost than the buildings and trees. However, due to extreme usage of color, parking lots also have very low cost. Also, in the pedestrian cost map we can clearly see that roads, pavements and trees have lower cost compared to buildings. To visualize the surface on which dijkstra is running, a surface plot is used. This plot shows us how non-linear the objective function is. Surface plots of the cost maps for car and pedestrian planner are shown in Figs. 52 and 53 respectively.

VII. Some Mathematical hacks for stability

I used the exponential cost map to avoid negative values in the cost map. However, due to badly scaled values the exponential value was sometimes too big or too small. To avoid this, the values obtained after the exponential map are subtracted by the minimum value and a small offset (lowest value of a node like 0.02) is added to it. This provided numerical stability and weirdly enough made Dijksta run faster.

VIII. Results

A. Train Set

The outputs for training set for car and pedestrian planner are shown in Figs. 48 and 49 respectively. One can observe almost perfect following albeit some minor variations which can be associated with batch gradient descent, because all the paths are trying to be optimized at once, it's very hard to come out of a local minima of a single path. The training set outputs are overall satisfactory.

B. Test Set

The outputs for testing set for car and pedestrian planner are shown in Figs. 54 and 55 respectively. Except the blue path in pedestrian all the paths look good. Due to drastic change in road color, the pedestrian planner decides to cross a building due to its lower cross section instead of going around it. This happens partially because I trained my code to go inside the stadium and hence it thinks bright things are walkable regions. Apart from the superman case the code works very well (better than expected).

IX. Important Lessons Learnt

Never hand-tune too many parts of the code.
Batch gradient descent's convergence is highly dependent on the learning rate.

X. Conclusions

In general, the imitation learning code could generate feasible paths most of the time for both cars and pedestrians. This is a very hard challenge because we only had a satellite image to look at, it was hard to tell what are flyovers. The algorithm performed brilliantly well on both training and testing set. Although, the performance severely depends on good orthogonal feature-set.

Acknowledgment

The author would like to thank Dr. Daniel D. Lee and all the Teaching Assistants of ESE 650 course for all the help in accomplishing this project.

References

[1] Nathan D. Ratliff, David Silver, and J. Andrew Bagnell, *Learning to search: Functional gradient techniques for imitation learning*, Autonomous Robots, Vol. 27, No. 1, pp.25–53, 2009.

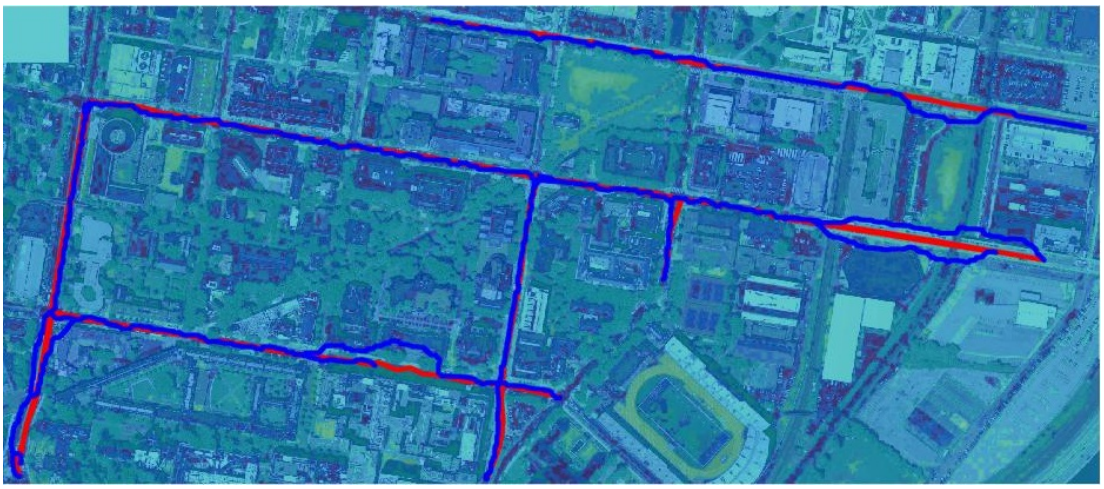

Fig. 46: Car paths after training with learning rate 10^{-3}, red shows expert path and blue shows learnt path.

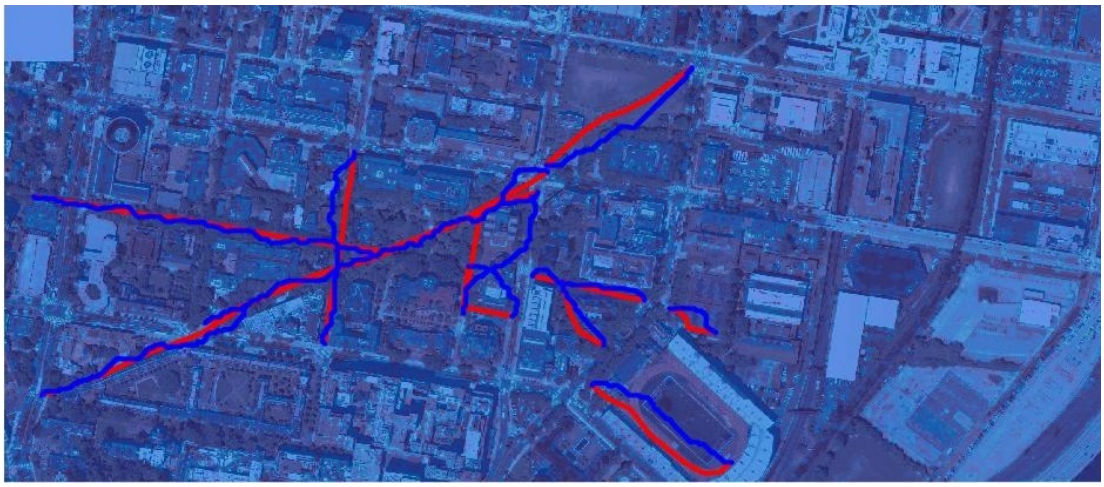

Fig. 47: Pedestrian paths after training with learning rate 10^{-3}, red shows expert path and blue shows learnt path.

Fig. 48: Car paths after refining with learning rate 10^{-4}, blue shows expert path and red shows learnt path, NOTE: colors are inverted as against previous set of figures.

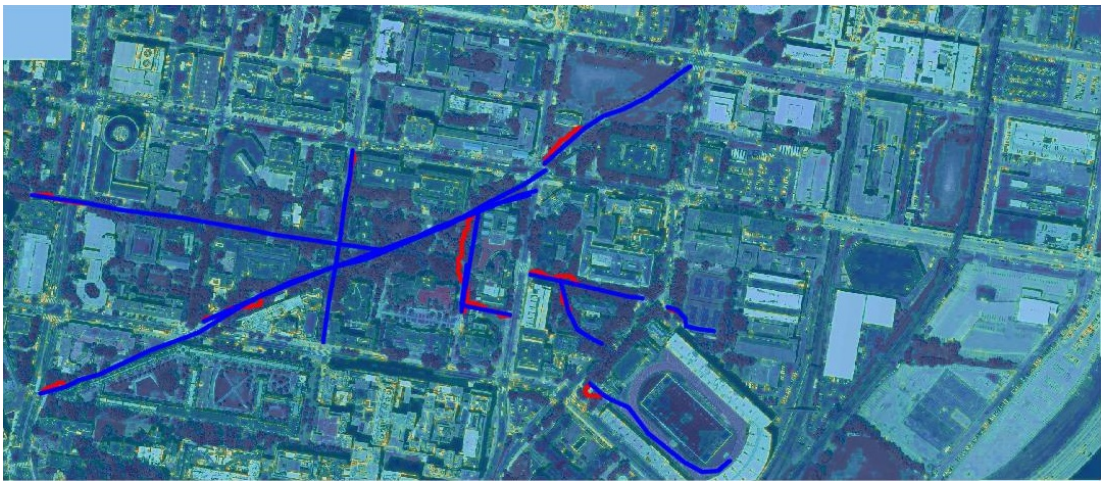

Fig. 49: Pedestrian paths after refining with learning rate 10^{-4}, blue shows expert path and red shows learnt path, NOTE: colors are inverted as against previous set of figures.

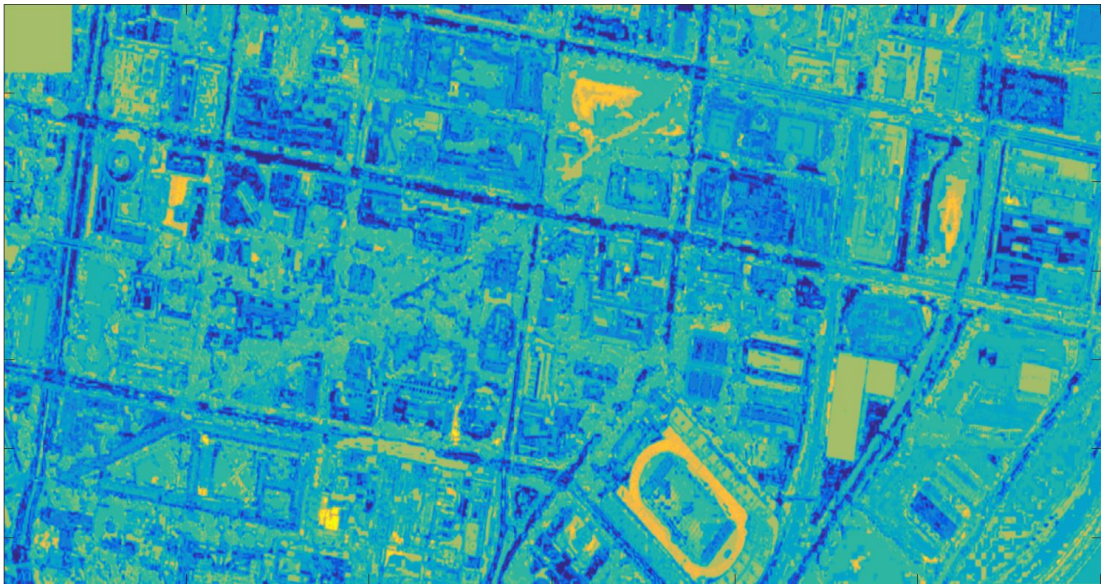

Fig. 50: Final learnt cost map for car path planning.

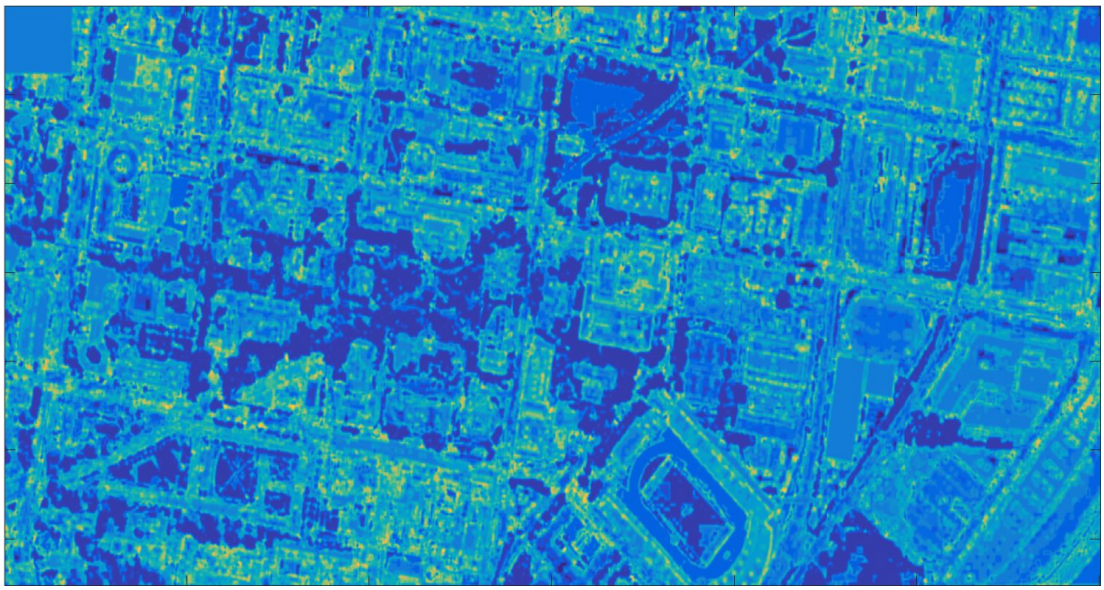

Fig. 51: Final learnt cost map for pedestrian path planning.

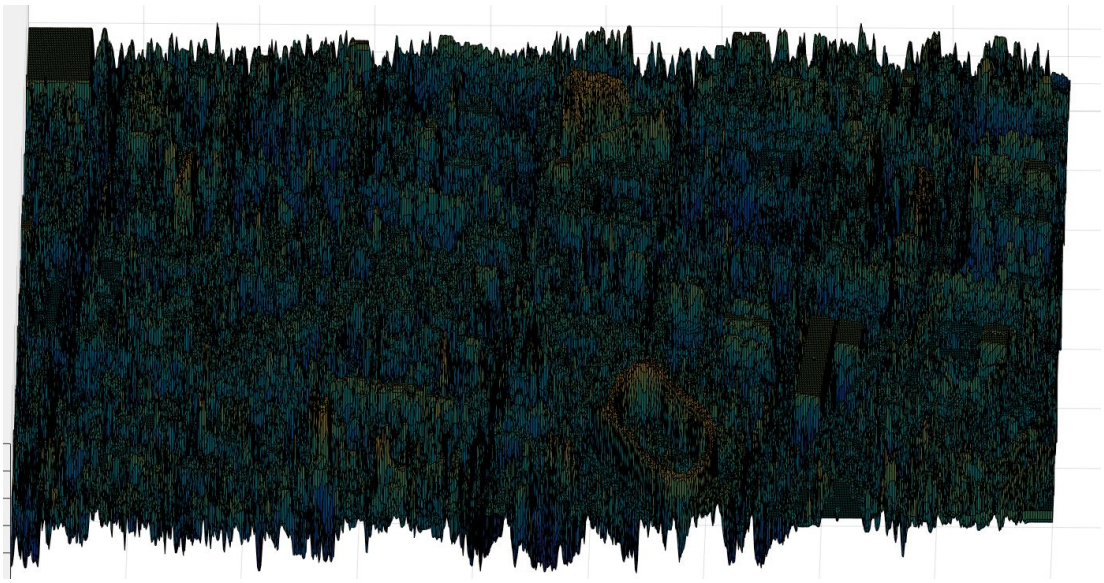

Fig. 52: Surface plot of final learnt cost map for car path planning.

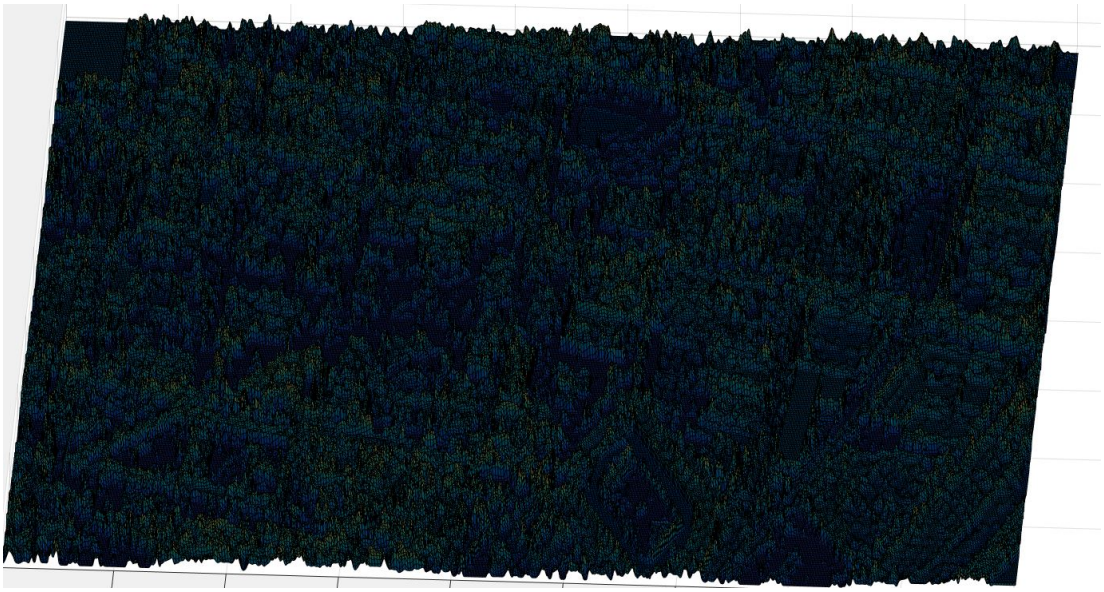

Fig. 53: Surface plot of final learnt cost map for pedestrian path planning (smaller values are artificially boosted for better visualization).

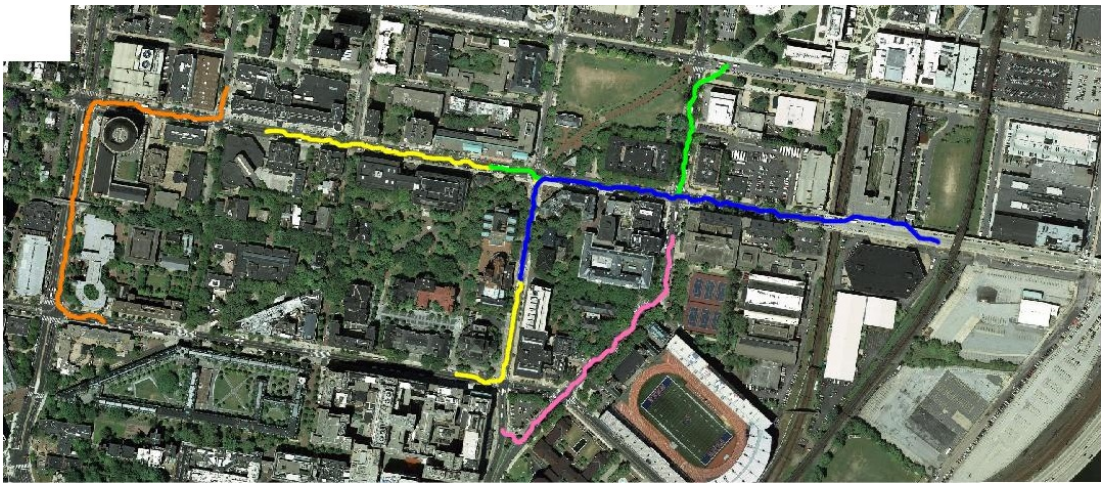

Fig. 54: Car path planner outputs for testing paths, different color indicate different paths.

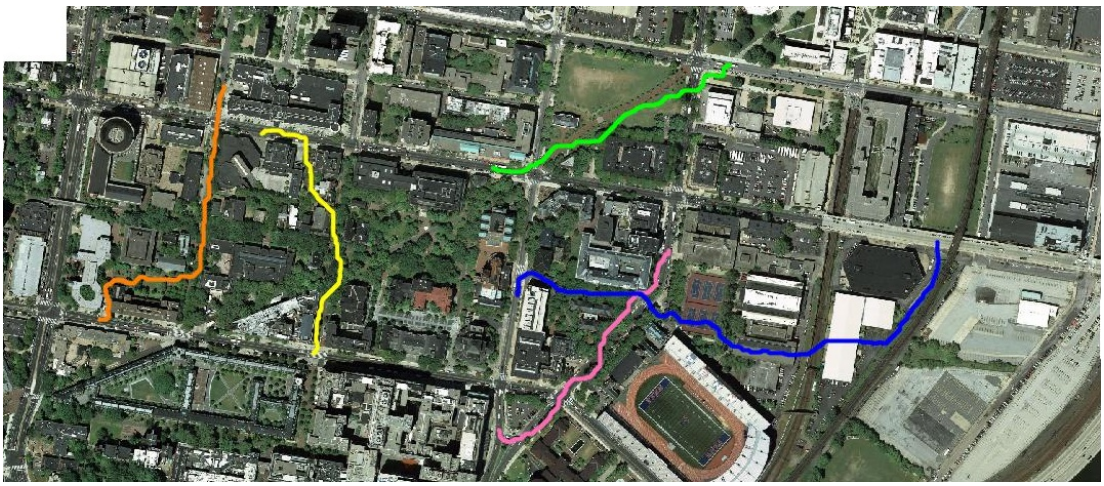

Fig. 55: Pedestrian path planner outputs for testing paths, different color indicate different paths.