ESE650 - Project 4
Learning Planning Costs

Arunkumar Byravan
Department of Mechanical Engineering
University of Pennsylvania

I. Introduction

The concept of Motion Planning is a key step towards making robots more autonomous. Encoding a behaviour into a robot, such as for a car to drive on roads, is a difficult task. Learning by imitation is the method of programming behaviour by demonstration. The aim of the project was to use an overhead map of the Penn campus, define features on it, learn to assign weights to each of these features and convert the map into a costmap for planning. Given an expert path on the map, learn the combination of weights for the existing features that would make the expert path the most optimal from a planning point of view. An example of such an expert path for an automobile is shown in figure 1. The sections below explain the workings of the system in more detail. Section 2 talks about the general concept and implementation, Section 3 about the training while section 4 discusses the results of testing using the weights that were trained from section 3.

II. Concept & Implementation

Imitation learning studies the algorithmic formalization for programming behavior by demonstration. Since many robot control systems are defined in terms of optimization (such as those designed around optimal planners), imitation learning can be modeled as finding optimization criteria that make the expert look optimal. This intuition is formalized by the maximum margin planning (MMP) and the LEARCH (Learning to Search) framework that we are going to make use of. A simple algorithmic implementation of LEARCH, incorporating the concept of MMP would have the following steps:

- Given a part of a map with an expert path, extract a bunch of features from the map
- Define a ”Loss map” either in euclidean or in feature space
- Combine the features to produce a Costmap. Subtract the Loss map from it.
- Plan on this loss-augmented costmap. Identify the optimal path
- Update the weights based on the differences between the optimal and expert path. Go back to the third step. Iterate till convergence

Each of the steps will be discussed in greater detail below.

A. *Feature Extraction*

Defining a good discriminative set of features is the most important step in this approach. If the features that are defined

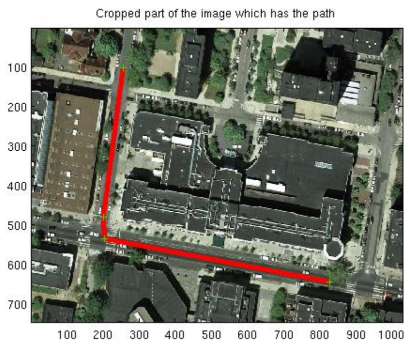

Fig. 1. An example ”Expert Path”

are not discriminative enough, the optimal path will never converge to the expert path or be along features similar to those in the expert path. In our implementation, we use 8 different features, namely six different colors, a binary building detector and a binary mask for intensity. Each step in feature detection is explained in more detail below.

1) Color Features: Color is one of the simplest features in a map. To detect different colors, color classifiers for six different colors were trained, namely for Black, White, Green, Gray, Red and Brown. Green was good for vegetation, Black & Gray for buildings and roads, Brown for pedestrian paths etc. Each color classifier had a single gaussian to represent the points belonging to that color and would return a mask containing the probability that a point belonged to that particular color. The sum of probabilities over all the six colors was set to 1. A sample result for the region of the map in figure 1 is shown in figure 2. This was generated by finding the max of the probabilities for each point and setting it to that color.

2) Building Detector: Buildings are a good source of features in urban environments. Also, if buildings are not detected, the paths may not converge properly to the expert path. The detection can be divided into a number of steps:

- Detecting Shadows in the image
  - This is done by detecting regions in the image which have constant intensity and whose borders show sharp changes with respect to its surroundings
  - The R,G & B channels are smoothed and subtracted from the original values
  - If abs(diff) ¡ 10 for all 3 channels, it is considered a

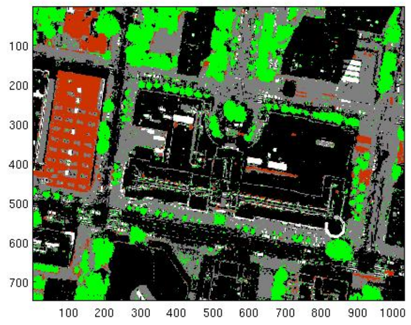

Fig. 2. Color feature

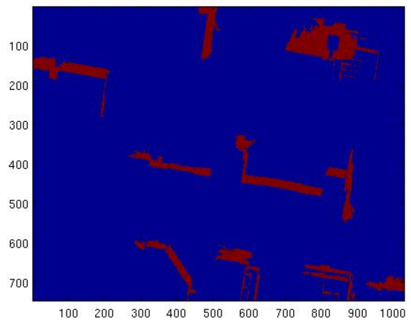

Fig. 3. Detected Shadows

shadow

- Compute the convex hulls of the shadows. Remove the original shadows from them
- Find blobs in the gray and black color channels that overlap with the convex hull. Assign them as buildings
- Also detect the black regions with the best solidity in the image. Assign them as buildings as well
- Finally include the shadows themselves as part of buildings

At the end of this process, we have a binary mask with ones at points that are buildings and zero otherwise. The various steps in this process are illustrated in figures 3-8, taking the region in figure 1 as an example. As you can see if figure 6, part of the road gets detected as a building, which shows that the system is not perfect.

3) Intensity Mask: Intensity of the image is also a godd discriminating feature. In our implementation, we use a binary mask where bright pixels get a value of one while darker ones get a value of 0. Initially, the image is converted to grayscale. Values greater than 128 are set to 1 and vice versa. This feature serves to discriminate between roads & sidewalks and acts as a ”pseudo” sidewalk detector. A result from the sidewalk detector is shown in figure 9.

B. Loss Map & Maximum Margin Planning

The concept of Maximum Margin Planning is simple. Given a set of features ”F” along the expert path, increase the costs

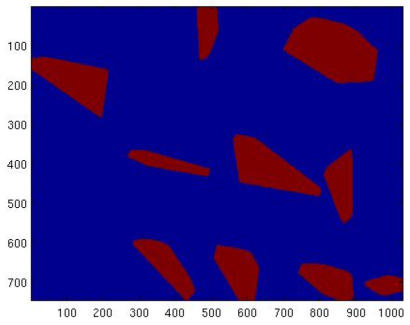

Fig. 4. Convex Hull of the shadows

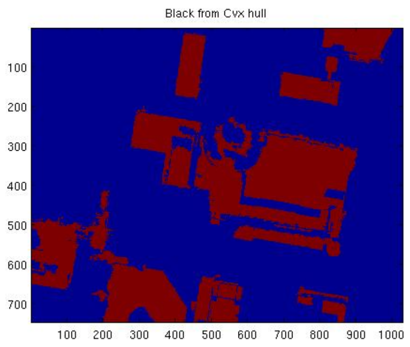

Fig. 5. Buildings identified from Black channel

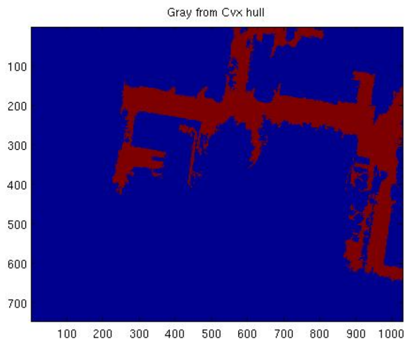

Fig. 6. Buildings identified from Gray channel

of points in the map which have features that are similar to F and decrease the costs for those whose features differ from F. This would ideally make the regions with ”favorable” features more costly to cross to while making other regions cheaper. If the system is still able to find a set of weights that make the optimal path converge to the desired path, it will be more robust. To facilitate this, we can compute the ”Loss map”, which will be subtracted from the costmap to give the loss-augmented costmap that we use for planning. In our implementation, we compute the Loss Map in feature space rather than in euclidean space. The algorithmic implementation is as follows:

- Given a set of feature maps and the expert path, find the

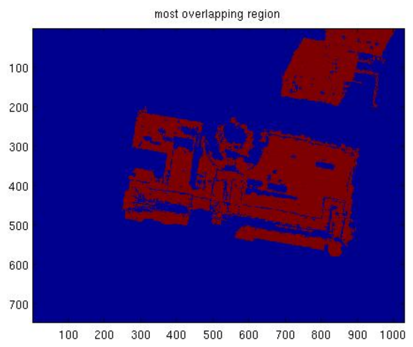

Fig. 7. Most solid Black Blobs

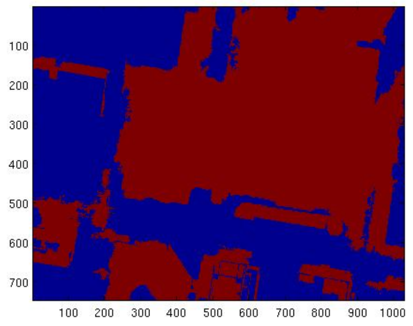

Fig. 8. Final mask

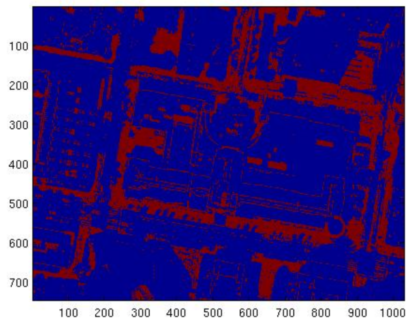

Fig. 9. Intensity Mask

features along the expert path

- Compute the Mean and SD of each of these features
- For each feature map, all points within 1.5*SD from the mean are set to zero and vice versa
- Now take the mean across all these "Feature-Loss Maps" to get the "Loss Map"

A sample loss map for the region shown in fig 1 is shown in figure 10. The Loss map is used only for training and not for testing.

C. Costmap generation

Once the features and the Loss map have been generated, we can combine them to generate the "Loss-Augmented

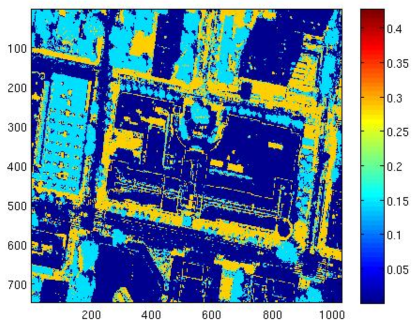

Fig. 10. An example "Loss Map"

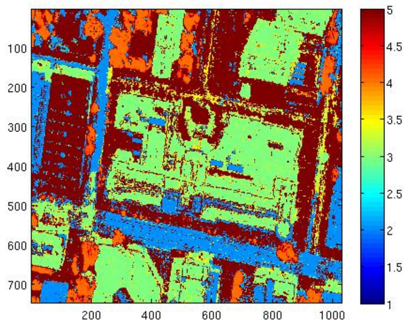

Fig. 11. An example Costmap

Costmap". Each feature has a weight assigned to it. The concept of imitation learning and maximum margin is to find an optimal set of weights that make the desired path to be the optimal one. In our implementation, we take the exponent of a linear combination of the weights and features as the initial costmap. Now, the Loss Map is subtracted from this to get the "Loss-Augmented Costmap". A value of 1 is added to this to ensure that none of the costs become negative as this would create problems for the planner. An example costmap is shown in figure 11.

D. Planning

Once the costmap has been generated, we can plan on this costmap. In this particular implementation, the Dijkstra's algorithm is used to generate the most optimal path.

E. Updating the weights

After planning, we have the most optimal path. Now, we have to update the weights such that the desired path becomes the most optimal. This is done in the following way:

- Identify all the features along the optimal path. Take them as positive examples
- Identify all features along the desired path. Set them as negative examples
- Run a regressor to find the hyperplane that best classifies the features

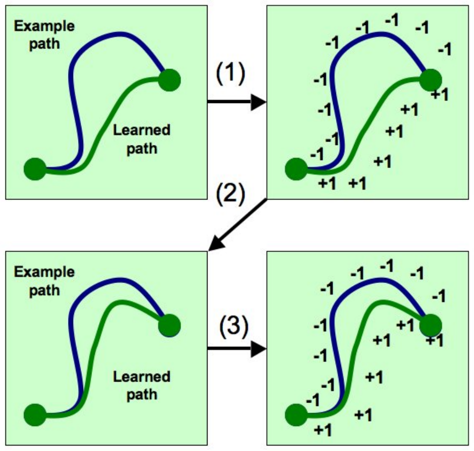

Fig. 12. Iterative setup - taken from [1]

- Move a small distance along the normal of this hyperplane. Add the value to the old weights to get the new ones
- The distance is defined by the stepping parameter
  - step size = learn rate/iteration num
- Here, the learning rate is set to 1.

The system is conceptually pictured in figure 12. Once the weights have been updated, we can compute the costmap again and plan on the costmap to identfy the new optimal path. This is done until the change in weights is very small (until convergence). If the features are discriminative enough, this should result in the desired path being set as the optimal path (or a path along the same features).

III. Training

The system was trained for two different modes, a pedestrian mode where the paths would be along sidewalks and a vehicle mode where the paths would be along the roads. Training followed the above procedure until the path converged to the desired path or another path along similar features that was more optimal. THe reults of training for both the vehicle mode and pedestrian mode are shown below. Figures 13 & 14 show the paths at the start and after convergence. The setting is the same one as in all the above figures. FOr pedestrian mode, the setting is the same except that the desired path is along the sidewalks of the building in the center. Figures 15 & 16 show the paths for pedestrian mode. In the figures, the path in red shows the optimal path while the one in blue shows the desired.

IV. Results & Discussion

Once training was done and the weights were identified, they were used to test the system on a number of different cases. All the figures shown below have 2 paths - one in blue which is what a human would want to do in that case and one in red, which the optimal path from the planner. The blue paths in this case are just there for a reference. They are not actually used in the testing.

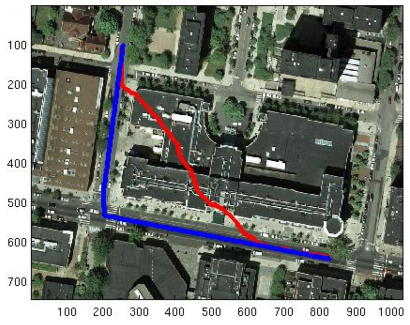

Fig. 13. Training - Iteration 1 - Vehicle Mode

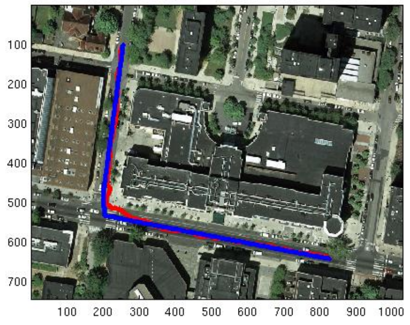

Fig. 14. Training - After Convergence - Vehicle Mode

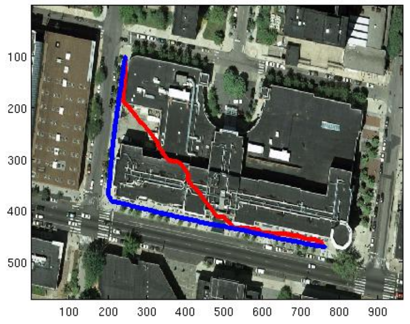

Fig. 15. Training - Iteration 1 - Pedestrian Mode

A. Vehicle Mode tests

For vehicle mode, the tests were done on regions with different roads and lighting conditions. For testing, the loss map was not subtracted from the costmap. Also, just one iteration is performed as the weights have already been identified. Figures 17 & 18 show the costmap and the result for one of the tests. As the shadow was included as a building, it can be seen that it ends up having a high cost. The path that is generated (in red) tries to avoid the shadow as it goes towards the goal. Figures 19 & 20 show two other tests using the trained weights for vehicle mode. The results show that the system works pretty well under the current set of features.

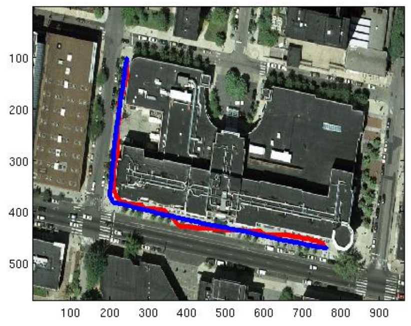
Fig. 16. Training - After Convergence - Pedestrian Mode

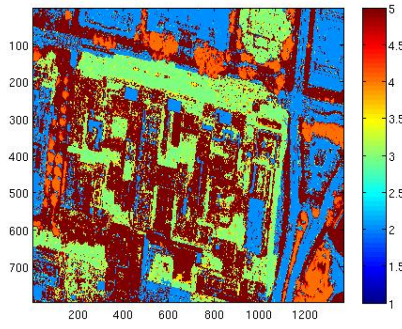
Fig. 17. Test 1 - Vehicle Mode - costmap

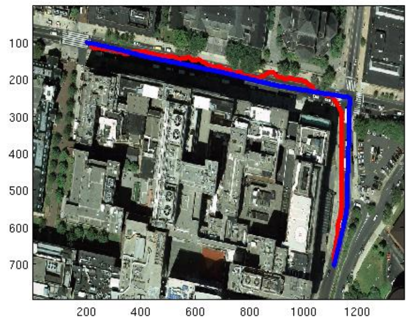
Fig. 18. Test 1 - Vehicle Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

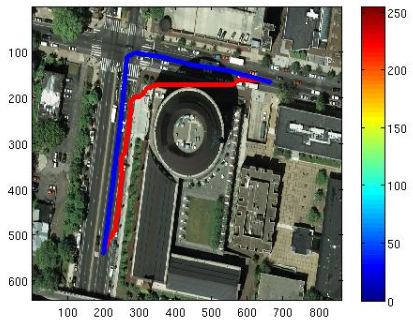
Fig. 19. Test 2 - Vehicle Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

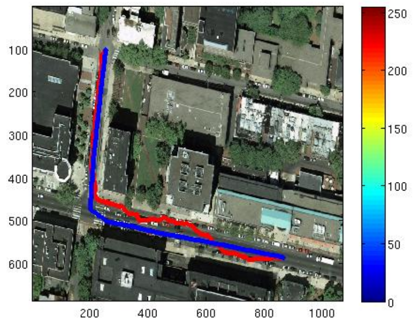
Fig. 20. Test 3 - Vehicle Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

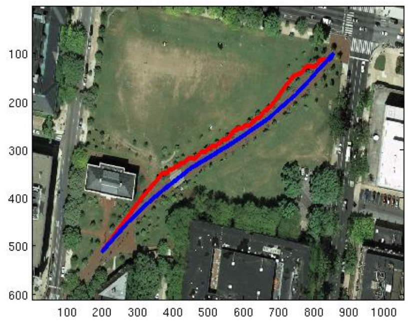
Fig. 21. Test 1 - Pedestrian Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

B. Pedestrian Mode Tests

Similar tests were carried out in pedestrian mode where the desired path was along sidewalks. The training setup was shown in figures 15 & 16 above. From that one can see that the desired path was along the sidewalks and vegetation (gray, green & some white color) and devoid of any brown,black or red. The weights generated reflect this, which can be seen from the tests below.

- Fig 21 shows an ”ideal” path between two points on the brown region. But the optimal path avoids any brown and travels only along the green
- Fig 22 shows a similar setup where the optimal path is only along the vegetation
- Fig 23 shows a setup where the robot is able to find a simple path between the start and goal through sidewalks and crossings. The ideal path is shown for a reference
- Fig 24 shows a result similar to that of fig 23. There is no building detection in the white channel which leads the path to touch the building.

C. Discussion

The results above indicate that the current setup works well under varied start and goal positions. The combination of color

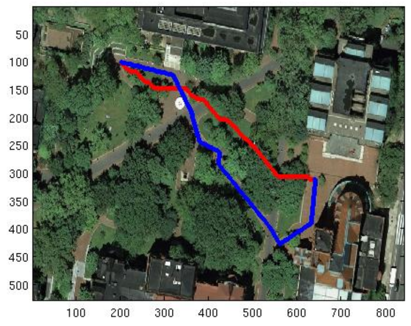

Fig. 22. Test 2 - Pedestrian Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

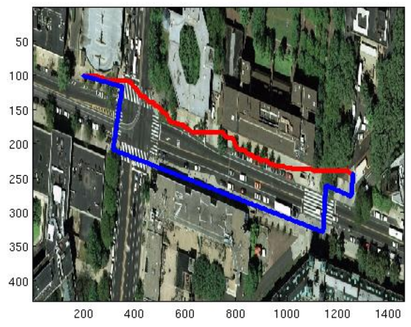

Fig. 23. Test 3 - Pedestrian Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

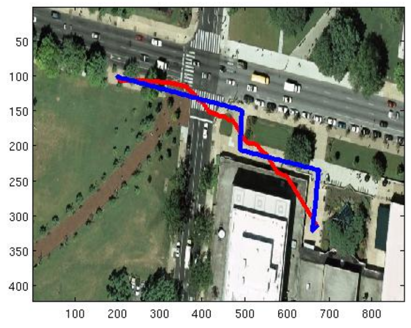

Fig. 24. Test 4 - Pedestrian Mode - Optimal path(Red) vs ”ideal” path (Ideal path just for reference)

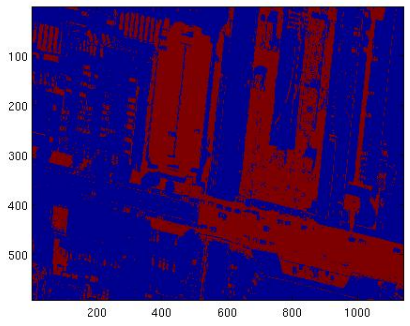

Fig. 25. Vehicle Mode bad test - Sidewalk detector

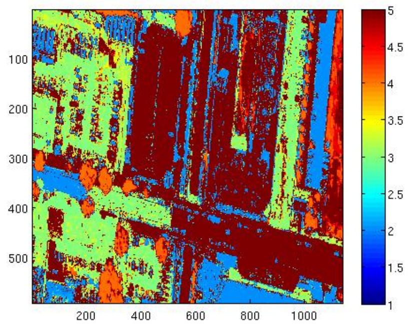

Fig. 26. Vehicle Mode bad test - Costmap

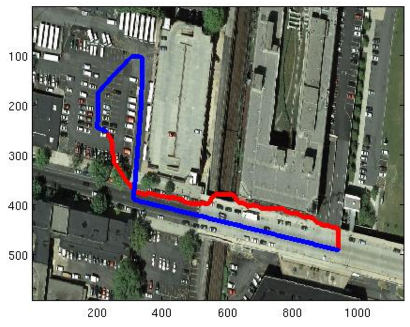

Fig. 27. Vehicle Mode bad test - Optimal path in red

features and intensity along with the building detector is able to produce sufficiently discriminative features. The building detector though, can produce a lot of fasle positives as can be seen in fig 6 where a part of the road gets detected as a building. There are a few cases where the system fails to work due to bad features. Also, the system will not work if it is tested on something different from its training data. Figures 25 - 27 shows a case where the system does not perform well. Part of the road, which is gray gets detected in the sidewalk detector (fig 25) which inturn increases its cost (fig 26). So, the path tries to avoid the gray part of the road. Based on the training data, this is the right thing to do as in the training data, the path avoids sidewalks.

In terms of computation, the most expensive step is the computation of the features. The computation of the convex hull is very costly. The setup as a whole takes about 15 seconds to run, which is pretty slow. This can be speeded up. In terms of things to improve performance, the training can be done over more than a single dataset as this would give us better positive and negative examples enabling us to get a better classifier and thereby updating the weights in a much better fashion.

V. Conclusion

Using the concepts of Maximum Margin Planning and LEARCH, a simple imitation learning setup was created. Expert paths were chosen. Different features, namely color,intensity and buildings were computed. A Loss-Augmented costmap was generated from the various features and weights. Dijkstra's algorithm was used to plan on the costmap. The weights were updated using a classifier. Iterating through this process until convergence, a set of weights were recovered. The weigts were used to test on a variety of different scenarios and were found to work well. Performance was analysed and improvements were suggested. Together with a simple controller and a map building scheme, different behaviours can be "learned" and executed in a satisfactory manner.

References

[1] N. D. Ratliff, D. Silver and J. A. Bagnell, *Learning to Search: Functional Gradient Techniques for Imitation Learning*, Autonomous Robots Vol.27, No.1, 2009.

[2] N. D. Ratliff, J. A. Bagnell and M. Zinkevich, *Maximum Margin Planning*, ICML, Pittsburgh, PA, 2006.