16-811: MATH FUNDAMENTALS FOR ROBOTICS

FINAL PROJECT REPORT

SEMANTIC OBSTACLE AVOIDANCE FOR UAVS BY LEARNING COST FUNCTIONS VIA INVERSE REINFORCEMENT LEARNING

December 15, 2016

Arjun Sharma (arjuns2), Ratnesh Madaan (ratneshm), Rogerio Bonatti (rbonatti)
Carnegie Mellon University
Robotics Institute

Contents

1 Abstract

Explicitly programming a robot to perform a task with a behavior that seems natural to a human being, or even using common-sense assumptions, is extremely difficult. While it is often easy to abstractly describe or even demonstrate a desired behavior [1], designing explicit rules to reproduce the same behavior is difficult, time consuming and often times an expert's behavior cannot be represented by simple functions. Given expert demonstrations, however, it is possible to estimate a cost functions using Inverse Reinforcement Learning (IRL) techniques. In this project we compare two IRL methods, Maximum Margin Planning (MMP) and Maximum Entropy (MaxEnt), for encoding cost functions given 35 synthetic expert demonstrations in 10 different 2D scenarios. After learning cost functions associated with three different classes of obstacles, we estimated cost maps for four new test scenarios. Qualitatively both techniques reproduced the original cost maps with reasonable accuracy, but results using MMP presented a better spatial resolution, given our discrete feature vector.

2 Introduction

Real-life behaviors that oftentimes seem intuitive and effortless for human experts such as avoiding obstacles while controlling a vehicle can be extremely hard to explicitly program in autonomous machines, given the high dimensional space of the actions such vehicle can take. Therefore, somehow learning this intuitive behavior becomes an important task to increase the autonomy of intelligent machines.

Our objective in this project is to learn cost functions from an expert demonstrations to solve a motion planning problem in the context of semantic obstacle avoidance. The idea is that learned cost functions can later be sent as an input to a planner algorithm to generate trajectories for autonomous vehicles in a real-life scenarios.

Semantic obstacle avoidance refers to the act of avoiding different classes of obstacles (*e.g.* trees, wires, buildings) with different behaviors. As an example, flying a drone under a bridge is more costly than flying over it due to potential loss of GPS signal. However, flying either under or over a wire would does not present GPS loss, but drones should maintain a greater distance from wires than from buildings due to difficulty in precise wire localization.

We used Inverse Reinforcement Learning (IRL), a method of imitation learning, to recover obstacle avoidance policies given a dataset of expert-trajectories generated synthetically with a reference A^* planner. We implemented two popular IRL approaches in this project: Maximum Margin Planning (MMP) and Maximum Entropy (MaxEnt). In each of these approaches, the cost (or reward) is expressed as a linear combination of some features over states. In *MMP* [1], imitation learning is framed as a maximum margin structured prediction problem over a space of policies. The *MaxEnt* method [2] resolves the ambiguity of previous approaches where no single reward function is able to make the demonstrated behavior optimal [1], and where a single policy could be optimal for multiple reward functions [3]; by exploiting the principle of maximum entropy, where the probability of preference for any trajectory is proportional to the exponential of reward along that path. This method gives a single stochastic policy.

There are also methods that use deep learning to compute non-linear combinations of features to create the resulting cost functions. [4] outlines a general framework where deep neural networks are used to approximate the reward functions in the IRL problem, thereby modeling them as non-linear combinations of features and improving upon previous approaches, which modeled them as linear [2] and used non-parameteric methods like Gaussian Processes [5]. Non-parametric methods become sub-optimal for large problems. However, these methods were not tested in the current project, but are mentioned here as possible future improvements.

3 PRE-REQUISITES

In this section, we first introduce Markov Decision Processes. Next, we give a short introduction to Primal-dual formulation, which we will use in the derivation for the Max-Entropy Inverse Reinforcement Learning method.

3.1 Markov Decision Process

A finite Markov Decision Process (MDP) is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where

- S is a finite set of N states
- $A = \{a_1, \cdots, a_k\}$ is a set of k actions
- $P_{sa}(.)$ are the state transition probabilities upon taking action a in state s
- $\gamma \in [0, 1]$ is the discount factor
- $R : S \mapsto \mathbb{R}$ is the reward function

A policy is defined as a map $\pi : S \mapsto A$. The (state) value function for a policy π is defined as

$$V^{\pi}(s_1) = E(R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \cdots | \pi)$$

where the expectation is over the distribution of state sequence $(s_1, s_2, s_3, \cdots)$ that we observe when we execute the policy π. The action value function is given by

$$Q^{\pi}(s,a) = R(s) + \gamma E_{s'\ P_{sa}}(V^{\pi}(s'))$$

The state and action value functions satisfy the following equations known as the *Bellman Equations*

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s')V^{\pi}(s')$$
$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s')V^{\pi}(s')$$

3.2 Primal-Dual formulation

Consider the following optimization problem,

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & Ax \le b \\ & Px = q \end{aligned}$$

Let's try and find a lower bound for the minimum value our objective function can take subject to the given constraints. Let $\mathscr{C}$ denote the set of feasible x. Notice that,

$$\min_{x\in\mathscr{C}} f(x) \ge \min_{x\in\mathscr{C}} f(x) + u^T(Ax-b) + v^T(Px-q) \tag{1}$$

for $u \ge 0$. This is because for $x \in \mathscr{C}$, $Ax - b \le 0$ and $Px - q = 0$. Note that there is no constraint on v. Define the Lagrangian function as,

$$\mathscr{L}(x,u,v) = f(x) + u^T(Ax-b) + v^T(Px-q) \tag{2}$$

Then, we can say that,

$$\min_{x\in\mathscr{C}} f(x) \ge \min_{x\in\mathscr{C}} \mathscr{L}(x,u,v) \ge \min_{x} \mathscr{L}(x,u,v) \tag{3}$$

The second inequality stems from the fact that minimizing over an unconstrained set gives you more freedom to chose x which can reduce the function value further. Observing that $\min_x \mathscr{L}(x, u, v)$ is a function of u and v, denote

$$g(u, v) = \min_x \mathscr{L}(x, u, v) \tag{4}$$

Then,

$$\min_{x\in\mathscr{C}} f(x) = f^*(x) \geq g(u, v) \tag{5}$$

A tighter lower bound can be obtained by the solving the following *dual* optimization problem,

$$\begin{aligned} \max_{u,v} \quad & g(u, v) \\ \text{subject to} \quad & u \geq 0 \end{aligned}$$

4 INVERSE REINFORCEMENT LEARNING

Consider the example of creating a model to understand how a bee chooses a flower as it forages for nectar. Its preference for a flower may be a influenced by a variety of factors, such as its belief of flower's nectar content, the distance to the flower and the time it takes to reach it, and the risk of predators. It is difficult to assign weights to each of these factors a priori. Inverse Reinforcement Learning (IRL) deals with the problem of determining the reward function that an agent is trying to optimize given some observations of the agent's behaviour. An IRL algorithm would thus, attempt to learn the correct set of weights for the various factors in the bee foraging problem, given some demonstrations of bees visiting flowers.

In this work we look at finite space and assume that the model is known. Given a finite state S, a set of k actions $A = \{a_1, \cdots, a_k\}$, transition probabilities $\{P_{sa}\}$, discount factor γ and demonstrations from a policy π, we wish to find a reward R such that π is an optimal policy in the MDP $(S, A, \{P_{sa}\}, \gamma, R)$. In both methods discussed below, the reward at a state-action pair is assumed to be linear in its features.

4.1 Maximum Entropy IRL

Maximum Entropy IRL takes a probabilistic approach to determining the unkown reward function. The principle of *maximum entropy* is used to resolve the ambiguity in choosing a distribution over decisions subject to the constraints that the expected feature counts of the learner's behaviour match the empirical feature count of the expert's demonstrations. This ensures that the distribution over decisions is no more committed to any particular path than this constraint requires.

Formally, let τ_i denote trajectory i and let $p_{\tau_i} = p(\tau_i)$ denote its probability. Let $s_{i,t}$ and $a_{i,t}$ be the state and action at time t in trajectory i and let $f(s_{i,t}, a_{i,t})$ denote the feature of this state-action pair. Let $\bar{f}(\tau_i) = \sum_{t=1}^{T} f(s_{i,t}, a_{i,t})$. Then, maximum entropy inverse reinforcement learning maximizes the cross entropy (or minimizes the negative cross entropy) as follows

$$\begin{aligned} \min_{p} \quad & \sum_{\tau} p_\tau \log p_\tau \\ \text{subject to} \quad & Fp = \frac{1}{n}\sum_{i=1}^{n} \bar{f}(\tau_i) \\ & 1^T p = 1 \end{aligned}$$

where $F_{ij} = \bar{f}_i(\tau_j)$, the i^{th} entry in $\bar{f}(\tau_j)$

Then,the Lagrangian is defined as

$$\begin{aligned} \mathscr{L}(p,\theta,v) &= \sum_{\tau} p_\tau \log p_\tau + \theta^T(Fp - b) + v(1^T p - 1) \\ &= -\theta^T b - v + \sum_{\tau} (\theta^T F)_\tau p_\tau + p_\tau \log p_\tau + v p_\tau \\ &= -\theta^T b - v + \sum_{\tau} p_\tau (\log p_\tau + (\theta^T F)_\tau + v) \end{aligned}$$

Minimizing with respect to p_τ by setting

$$\frac{\partial \mathscr{L}(p,\theta,v)}{\partial p_\tau} = 1 + \log p_\tau + (\theta^T F)_\tau + v = 0$$

gives

$$p_\tau^* = \exp(-\theta^T \bar{f}(\tau) - v - 1) = \frac{\exp(-\theta^T \bar{f}(\tau))}{\exp(v+1)}$$

Since p_τ^* should sum to 1 over all possible trajectories,

$$\exp(v+1) = \sum_{\tau} \exp(-\theta^T \bar{f}(\tau)) = Z(-\theta)$$

Thus, the probability of trajectory τ is given by,

$$p(\tau) = \frac{e^{-\theta^T \bar{f}(\tau)}}{Z(-\theta)} = \frac{e^{w^T \bar{f}(\tau)}}{Z(w)} = \frac{e^{\sum_{t=1}^{|\tau|} R(s_t,a_t)}}{Z(w)}$$

for $w = -\theta$ and reward function R. Now,

$$\log p_\tau + (\theta^T F)_\tau + v = -1$$

Therefore,

$$\begin{aligned}
\mathscr{L}(p,\theta,v) &= -\theta^T b - v + \sum_\tau p_\tau(\log p_\tau + (\theta^T F)_\tau + v) \\
&= -\theta^T b - v - \sum_\tau p_\tau \\
&= -\theta^T b - v - 1 \\
&= -\theta^T b - \log Z(-\theta)
\end{aligned}$$

So the dual becomes,

$$\max_\theta \quad -\theta^T b - \log Z(-\theta)$$

Or equivalently,

$$\min_\theta \quad \log Z(-\theta) + \theta^T b$$

Setting $w = -\theta$, we get

$$\min_w \quad \log Z(w) - \left\langle w, \frac{1}{n}\sum_{i=1}^{n} \bar{f}(\tau_i) \right\rangle$$

4.2 Maximum Margin Planning

The goal of Maximum Margin Planning (MMP) is to learn a cost (negative reward) function for which the example policy has lower expected cost than any other alternate policy by a margin that scales with the loss of that policy. If the policy is very similar to the example policy (i.e. low loss), then the margin is small and the example policy needs to to have a cost only slightly less than the policy. On the other hand, if the policy differs from the example policy by a large extent (i.e. high loss), than the margin is large and the cost of the example policy should be significantly lower than the policy.

As before, let S be the set of states and A be the set of actions. Let $\mathscr{M} = S \times A$ be the combined set of state action pairs. Let a policy be represented by its state-action frequency count $\mu \in \mathbb{R}^{|\mathscr{M}|}$ and let $\mathscr{G}$ be the set of all feasible state-action frequency counts. Let the fully observed feature vector for state s and action a be denoted by $f_{sa} \in \mathbb{R}^d$. These features can be written in a combined matrix $F \in \mathbb{R}^{d\times|\mathscr{M}|}$. Finally, each policy μ has an associated loss $\mathscr{L}_i(\mu)$ which quantifies how bad a given policy μ is with respect to the expert policy μ_i for the i^{th} example. Then using this notation, the intuition above is formalized as follows

$$\begin{aligned}
\min_{w\in\mathcal{W},\epsilon\in\mathbb{R}_+} \quad & \frac{1}{N}\sum_{i=1}^{N}\epsilon_i + \frac{\lambda}{2}||w||_2^2 \\
\text{subject to} \quad & \forall i, w^T F_i \mu_i \leq \min_{\mu\in\mathcal{G}_i}\{w^T F_i \mu - \mathcal{L}_i(\mu)\} + \epsilon_i
\end{aligned}$$

where $\{\epsilon_i\}_{i=1}^{N}$ are slack variables which allow constraint violations for a penalty and $w^T F_i \mu$ is the cost for using policy generating μ. Since the slack variables are in the objective function, the minimization drives the slack variables to be as small as possible. In particular, at the minimizer the slack variables will always exactly equal the constraint violation. The following equality condition, therefore, holds at the minimizer

$$\epsilon_i = w^T F_i \mu_i - \min_{\mu\in\mathcal{G}_i}\{w^T F_i \mu - \mathcal{L}_i(\mu)\}$$

This allows moving the constraints directly into the objective function leading to the following Max Margin Planning objective

$$C(w) = \frac{1}{N}\sum_{i=1}^{N}(w^T F_i \mu_i - \min_{\mu\in\mathcal{G}_i}\{w^T F_i \mu - \mathcal{L}_i(\mu)\}) + \frac{\lambda}{2}||w||_2^2$$

5 Generating training data

In order to learn a cost function given expert demonstration, we need a dataset of MDP(s), cost functions corresponding to different classes of obstacles, and the expert demonstrations themselves. The following sub-sections highlight how we accomplish the same.

5.1 Gridworld

We use a simple 2D gridworld of size (100,100) and put point obstacles at random locations. The cost function builds upon the one we had in Assignment 4 in the line optimization problem, as shown below.

To emulate *semantics* in obstacles, for the goal of the robot avoiding different classes of obstacles with different behaviors, we weigh the cost function from the homework assignment by w_k for the k^{th} class, and also change the *zero-out distance* term, ϵ_k. The cost associated with the i^{th} obstacle of semantic class k at state (x, y) is given below. Here $d_{(x,y)}$ is the Euclidean distance between the state and the obstacle, and ϵ_k is a term we call the zero-out distance informally.

Table 1: Training data parameters

	Red (k=1)	Yellow (k=2)	Green (k=3)
No of obstacles	15	15	15
ϵ_k	20	15	25
w_k	0.6	0.4	0.2

Table 2: Testing data parameters

	Red (k=1)	Yellow (k=2)	Green (k=3)
No of obstacles	5	5	5
ϵ_k	20	15	25
w_k	0.6	0.4	0.2

$$cost_i(x,y) = \begin{cases} w_k * \epsilon_k, & \text{if } d_{(x,y)} > \epsilon_k \\ w_k * \frac{1}{2*\epsilon_k}(d_{(x,y)} - \epsilon_k)^2, & d_{(x,y)} < \epsilon_k \end{cases}$$

The total cost for N obstacles is then obtained by adding $cost_i$ as i goes from 1 to N. Our training data set contains of 10 gridworlds and the obstacle parameters just discussed, as shown in Table 1.

5.2 Generating synthetic trajectories

Now that we have gridworlds along with costmaps, we need some expert demonstrations that IRL algorithms will use. The gridworld can be interpreted as a Markov Decision Process, with the cost function being the negative of the reward, the state being the tuple of coordinates (x, y) and the possible action at each state being a tuple of at max 8 numbers, corresponding to left, right, up, down, and the four diagonal directions. To generate expert demonstration, we need to solve this MDP.

We implemented value iteration and the A* algorithm for the same as explained below.

5.2.1 Value Iteration

The goal of the value iteration algorithm is to solve an MDP in a *backwards* fashion, by starting from an arbitrary value function V_0 and updates defined by the following recursively equation (until convergence defined by $V_{i+1} - V_i < \theta$).

$$
\begin{aligned}
V_{i+1}(s) &= \max_a \sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V_i(s')), \text{for stochastic MDP} \\
&= \max_a (R_a(s,s') + \gamma V_i(s')), \text{ for deterministic MDP}
\end{aligned}
$$

By precomputing the values of each state, we could generate synthetic trajectories by sampling a random point in the grid, and move to one of the eight neighboring states which has the maximum value or staying at the current state itself. However, this resulted in states either oscillating or getting stuck towards the end of the trajectories, which were generally of short length (roughly 20-25 points by manual inspection). We tried fixing this by adding a goal point with high reward. However this meant that we needed to run value iteration for *each* synthetic trajectory, which took a lot of time. Hence, we dropped the idea and chose to implement A*. This also lead to an interesting discovery about something we hadn't thought about explicitly before - the conventional planning algorithms, as well the reinforcement learning algorithms based on value or Q function iteration, both solve an MDP, however, in literature, the connection is generally not made.

The resulting value function after our implementation is given in Figure 1

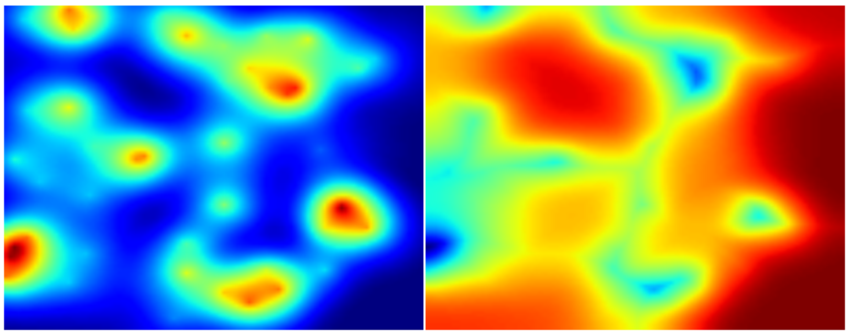

Figure 1: Left: original cost map and right: value function post convergence

5.2.2 A*

A* is a standard path finding algorithm which builds upon the slow but guaranteed to return a shortes path Djikstra, and the greedy heuristic based Best First Search algorithms. Specifically, it finds a path that minimizes the following cost function $f(n)$

$$f(n) = g(n) + h(n)$$

Here, g(n) is the cost function we define in the preceding sections, and $h(n)$ is the heuristic, which we chose to be the Euclidean distance in our case. A*, unlike value iteration, made it easier to specify goal states. For each of the 10 gridworlds from our training set, we chose start and goal states randomly, and generated 35 trajectories, as shown in Figure 2.

5.3 Feature selection

Choosing a good feature representation is essential for learning a reward function via MaxEnt or MMP as the learnt reward is a weighted sum of the features themselves. As we also want to capture the semantics, it's intuitive to look at numbers corresponding to closest obstacles of each class, or better still, a one hot encoding of classes. As our cost function is dependent on $1/x^2$ and has a "cut-off" condition which depends on x and a constant ϵ_k, it makes sense to have numbers proportional to $1/x$, $1/x^2$ and a bias term in the feature set, along with a one hot encoding of the class type of the obstacle. To avoid division by zero, one would want to use terms proportional to $1/(x+1)$ and $1/(x+1)^2$.

Initially, we tried using a feature set of three numbers : $\{1/(x_1+1), 1/(x_2+1), 1/(x_3+1)\}$ where x_i is the distance from the nearest obstacle belonging to the i^{th} class. However, it didn't perform well for obvious reasons : for example, if we're surrounded by multiple obstacles of the same type, we end up picking the feature corresponding to just one of them and ignoring other nearby obstacles which have a lot more effect on the reward rather than closest obstacles of the remaining classes which might be far-off and have a negligible or minor contribution to the cost at the current state.

Therefore, we decided it's better to consider N nearest obstacles(we chose N=5) and encode their semantic class by a one hot feature. We settled on a set of 5 numbers for each obstacles

$[10/(x+1), 10/((x+1)^2), onehot_1, onehot_2, onehot_3]$, and we have this set for the 5 closest obstacles to a state. Finally we add a bias term. Therefore, our feature vector is vector of length 5*5+1 = 26 numbers.

6 Results

Figure 3 shows three examples from a total of ten cost maps that were used for training the algorithm. Each figure contains 4 images (from left to right): the original cost map with expert trajectories, the original cost map with obstacle locations, learned cost map using MMP and learned cost map using MaxEnt.

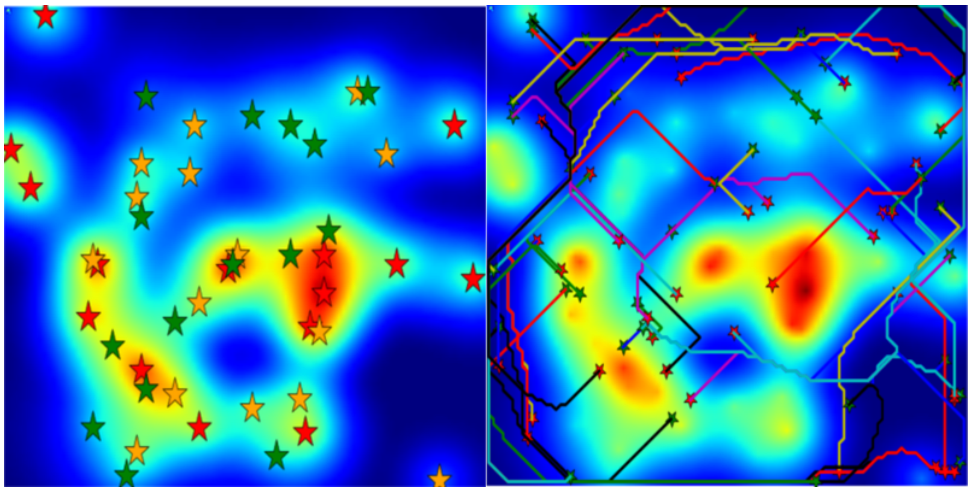

Figure 2: Left: Cost function with random obstacles of different classes, red being the deadliest, followed by yellow and green in that order. Right: Synthetic expert trajectories generated using A* algorithm. Starting at green and ending at red.

Figure 4 shows three examples from a total of five cost maps that were used when testing the IRL algorithms on new scenarios, unseen during the training phase. Each figure contains 3 images (from left to right): the original cost map with obstacle locations, learned cost map using MMP and learned cost map using MaxEnt.

As is evident, Maximum Margin Planning did a better job in approximating the true cost function as compared to MaxEnt. MMP learns cost maps that are better localized and precise than the one learnt from MMP in a sense that they are able to capture the position of the obstacles quite accurately if we compare the local maximas with the true location of obstacles (stars in 2nd column) but the cost function quickly degrades to zero as we get away from them. Whereas in MaxEnt, we are able to capture the diffuseness of the true cost map but the preciseness of the location of the obstacles isn't recovered. It's also worth pointing out that the costmaps learnt via both methods are non differentiable in places due to the one hot features.

6.1 Are we able to capture semantics?

To address this question, we must analyze the learned weights. Before we do that, let's recall some facts:

- Our feature vector is composed of 26 numbers - 5 numbers each for each of the 5 closest obstacles from the state in question.

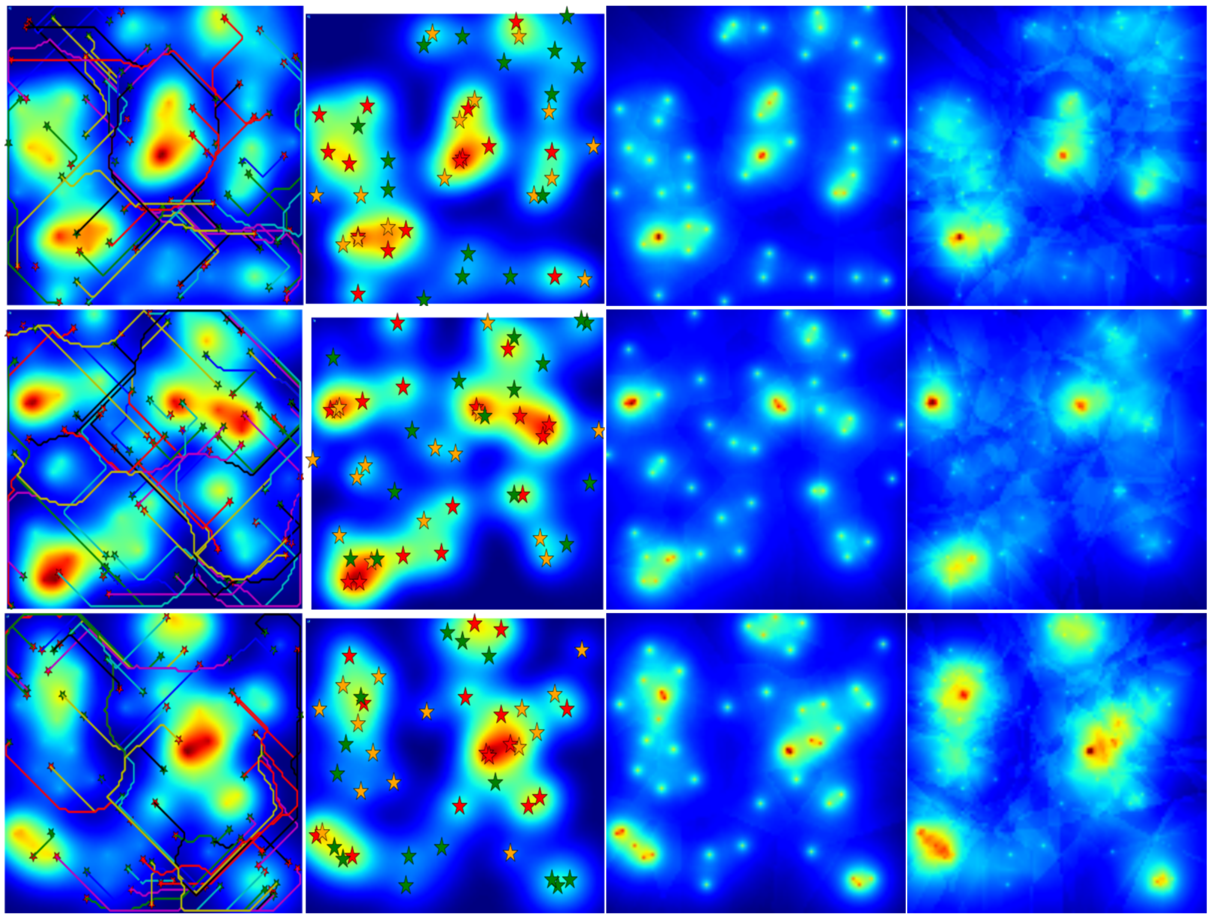

Figure 3: Results on training data. From left to right : Expert demonstrations overlaid on the input costmap; input costmap along with obstacles; learnt cost map via Maximum Margin Planning; and finally learnt cost map via Maximum Entropy IRL

- First three are a one hot encoding of the obstacle class and last two are inversely proportional to distance and squared distance from that obstacle.
- From Table 1, we can see that the zero out distance, ϵ_k for the green class is the highest, but the weight, w_k is the lowest.

Taking into account the above three points, one can say that for the obstacles closest to the state, the one hot feature corresponding to red class should be highest followed by yellow and green. However if we consider the 4th and 5th closest obstacles, the one hot features corresponding to the green and yellow classes should be higher. Figure 5 shows the learned weights.

MMP seems to be able to learn weights corresponding to the one hot features (yellow circles) which match our expectations just discussed. However as we go from the closest to the

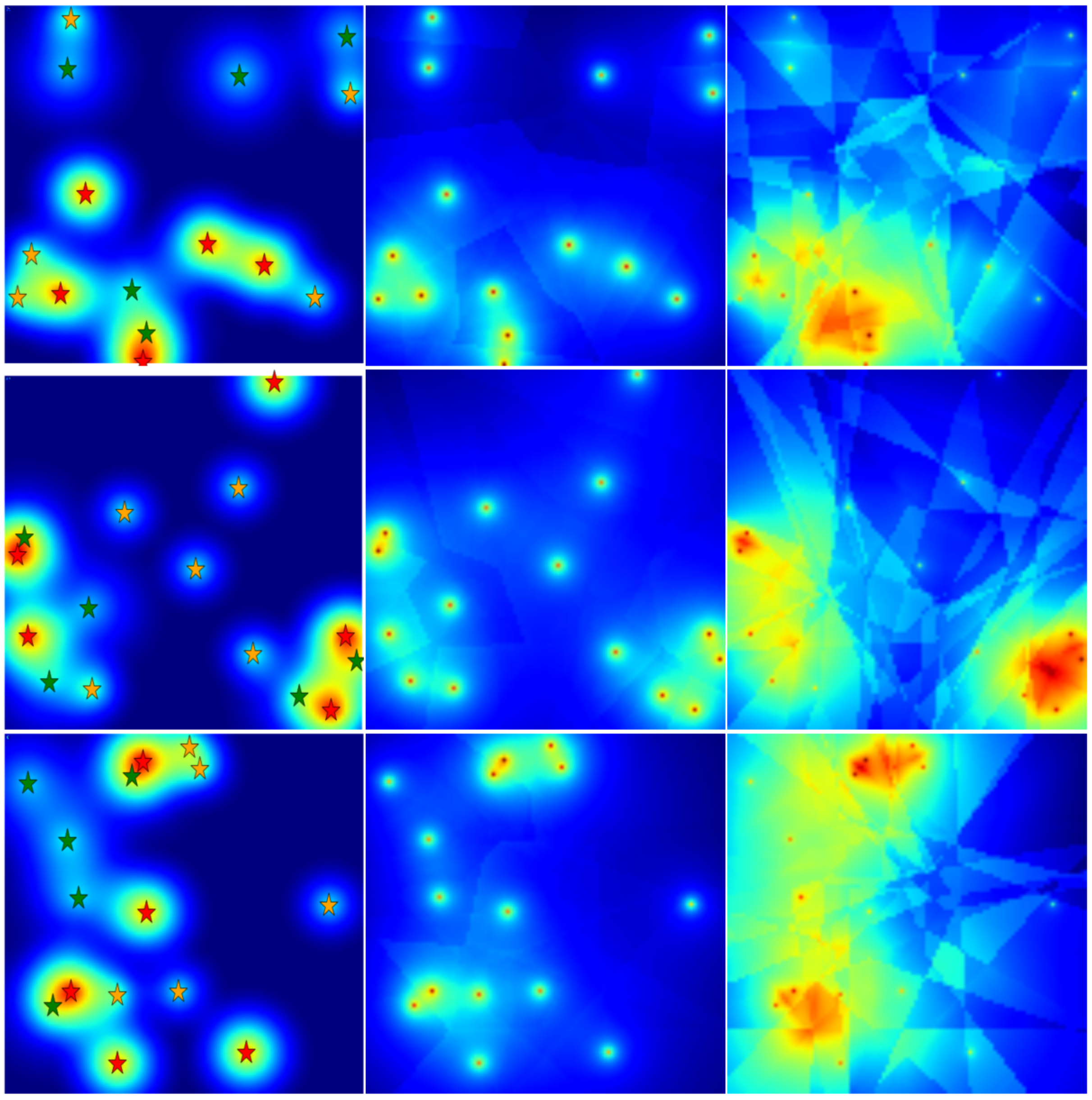

Figure 4: Results on test data. From left to right : Input costmap along with obstacles; learnt cost map via Maximum Margin Planning; learnt cost map via Maximum Entropy IRL

farthest of the five obstacles considered (left to right) in the figure, the weights proportional to the inverse of distance and squared distance features (purple dots) don't always follow the expected pattern.

The Maximum Entropy IRL method, while capturing the obstacle locations at a coarser level, was unable to capture the finer semantics about the different class costs.

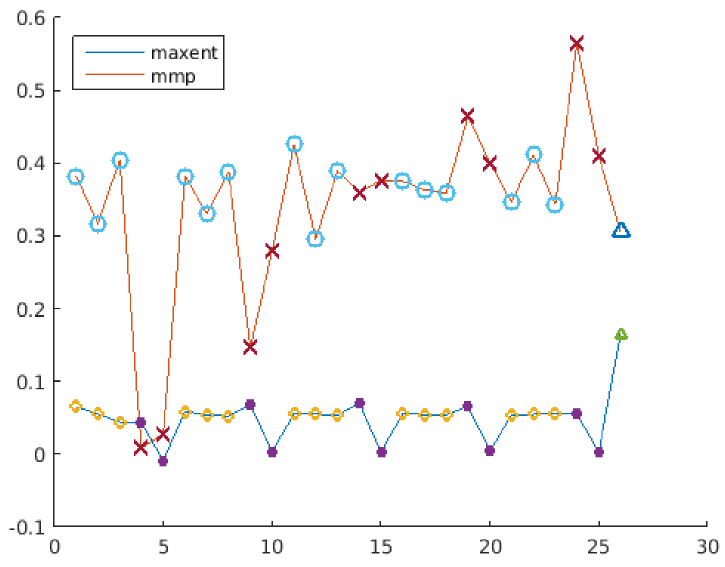

Figure 5: Learnt weights for our feature vector. From left to right, each X coordinate represents the one hot feature encoding (Red, Green, Yellow) of the obstacle class (3 numbers), then 2 numbers correspond to features $10/(x+1)$ and $10/(x+1)^2$, and this repeats for 5 closest obstacles to give a total of 25 features. The last number ($26^t h$) is the feature value corresponding to the bias term.

7 Conclusions

In conclusion, both IRL methods successfully recovered the original location for objects and original cost maps for the training examples, and generated acceptable cost maps for the test scenarios. The MMP method generated smoother cost maps than MaxEnt, where the spacial discontinuities created by the feature vector we used were not so evident.

Future work can be grouped twofold. First, we can use deep methods that create the cost maps based on a non-linear combination of features (unlike the linear combination we obtained with the current methods), and check if that generates a better approximation for the original cost maps. Second, this outputted cost map can be tested as input for the motion planner of a real physical system, and its performance can be compared with that obtained with other techniques used for semantic obstacle avoidance, such as hard-coded rules.

Bibliography

[1] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[2] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438, 2008.

[3] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[4] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[5] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.