Apprenticeship Learning for Motion Planning with Application to Parking Lot Navigation — A Summary

Twan van Laarhoven

5th April 2009

1 Introduction

In this essay I will give a summary of the article *Apprenticeship Learning for Motion Planning with Application to Parking Lot Navigation* by Pieter Abbeel, Dmitri Dolgov, Andrew Ng and Sebastian Thrun [1].

There are two sides to this article. On the one hand it deals with planning paths for a car driving around in a parking lot. That problem is solved by searching for a path that minimizes some cost function.

On the other hand, a good cost function needs to be designed. In the paper the cost function is learned from examples made by a human driver.

2 Path planning

The goal of path planning is to find a way to get from a given start point to given end point, while minimizing the cost $\Phi(\mathbf{s})$ of the path.

Many different cost functions can be used. For instance we want short paths, but perhaps also paths without many corners (low curvature), etc. Table 1 lists all the cost functions used in the paper.

Some of these functions are based on a graph $\mathcal{G}$ of the roads in the parking lot, which has to be specified by hand. There is a cost to being far away from a road in the graph, and for the angle between the road and the car.

forward length	$\sum_{i:\delta_i=0} \|x_i - x_{i-1}\|$
backward length	$\sum_{i:\delta_i=1} \|x_i - x_{i-1}\|$
switching direction	$\sum_{i:\delta_i\neq\delta_{i-1}} 1$
off-road driving	$\sum_{i:\neg R(x_i)} \|x_i - x_{i-1}\|$
distance to lane	$\sum_i \mathcal{D}(x_i, \theta_i, \mathcal{G})$
lane alignment	$\sum_i \sin^2(2(\theta_i - \alpha_i))$
curvature	$\sum_i (x_{i+1} - 2x_i + x_{i-1})^2$

Table 1: Different cost functions used in the paper. x_i are the points in the path, δ_i is the driving direction (forward or backward), $R(x_i)$ indicates whether a point is on the road, $\mathcal{D}(x_i, \theta_i, \mathcal{G})$ is the distance to a predefined graph of roads, θ_i is the angle of the car and α_i is the angle of the nearest road.

The overall cost function will be some linear combination of the cost functions from table 1,

$$\begin{aligned}\Phi(\mathbf{s}) &= \sum_{k=1}^{p} w_k \phi_k(\mathbf{s}) \\ &= w_{\text{fwd}}\phi_{\text{fwd}}(\mathbf{s}) + w_{\text{bwd}}\phi_{\text{bwd}}(\mathbf{s}) + \cdots\end{aligned}$$

With different weights w many different styles of paths are possible.

For now, assume that the weight vector is known, so that $\Phi(\mathbf{s})$ can be calculated. The paper then uses a two step approach to find a path that is close to optimal:

1. Global A* search in a discretized space.
2. Local refinement with conjugate gradient method using the actual cost function.

The A* algorithm is a good algorithm for finding optimal paths, but it requires a discrete search space. The state of the car can be specified by a 4-dimensional vector: two dimensional location, angle and driving direction. To be able to use A* this 4-dimensional state space is discretized.

Not all cost functions are suitable for such a discretization, however. In particular the last three functions (distance to lane, lane alignment and curvature) can not be discretized, since they are sensitive to local properties of the path. Therefore, after finding a rough global path that ignores these aspects of the cost, they are improved by using a conjugate gradient method.

3 Learning cost functions

The algorithm described in the previous paragraph depends on a set of weights w for combining the cost functions. Manually picking these weights is difficult, so the authors chose to train them based on examples from a human expert. The path planning should produce paths that are 'close' to these examples.

Directly comparing paths from the experts to paths found by the algorithm is not possible. There may, for instance, be two equally good ways around an obstacle. A better option of comparing the paths is by looking at the cost functions.

Define $\mu_k = \sum_i \phi_k(\mathbf{s}^{(i)})$ where $\mathbf{s}^{(i)}$ ranges over the paths found for different training examples (all with the same driving style). Let the vector μ contain the costs of paths found by the algorithm and let μ_E be the corresponding vector for the expert's paths. The goal is then to find a set of weights that makes μ match μ_E as close as possible, i.e. that minimizes $\|\mu - \mu_E\|$.

Surprisingly, the apprenticeship learning algorithm [2] can *always* find such weights. Informally the algorithm works as follows:

1. Pick an initial weight vector $w^{(0)}$.
2. Find the optimal paths for the current weight vector $w^{(j)}$.
3. Calculate the corresponding $\mu^{(j)}$.
4. Calculate μ as a linear combination of all previous $\mu^{(j)}$.
5. Pick a new weight vector that is in some sense orthogonal to the previous weight vectors.
   (steps 4 and 5 involve solving a single quadratic optimization problem).
6. Goto 2 unless $\|\mu - \mu_E\| \leq \epsilon$

In the end at least one of the weight vectors allows the path planner to behave as good as the expert. The algorithm is explained in more detail in [2] and [1].

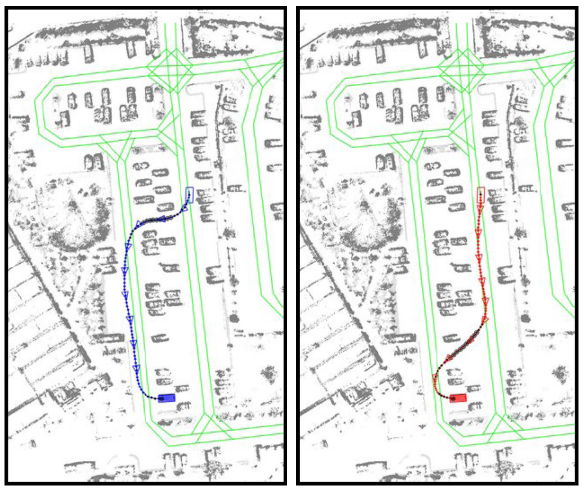
4 Experiments

The authors have recorded paths using a car equipped with sensors. The human drivers were instructed to use several different driving styles:

- 'nice': stick to the roads as much as possible.
- 'sloppy': the driver is allowed to deviate from the roads.
- 'backward': the driver may drive backward.

In all these cases the learning algorithm has found a set of weights that produce paths that are very similar to those used by the expert.

Figure 1 compares one of the expert's paths with a path found by the algorithm.



(a) the expert's path (b) the algorithm's path

Figure 1: Paths with the 'sloppy' driving style. Note that while the algorithm finds a slightly different path, the costs will be roughly equal. The paper contains many more examples.

5 Discussion

Defining a good cost function is often a difficult thing to do. While it is not hard to come up with different costs, figuring out which ones to use can be very hard. I therefore think that the approach of learning a policy tho match an 'expert' is widely applicable to reinforcement learning tasks.

A difficulty of the parking lot navigation problem is that there is no good way to quantify the results. While the planned paths look similar to the expert's paths, it is hard judge "how good" these paths actually are.

The parking lot navigation problem is also very small: there are just 7 cost functions, and only a 5 training examples were used for each driving style. It is not clear how well apprenticeship learning scales to larger problems. One can for example imagine introducing many variations of the costs, such as squared distance, log of distance, etc. I would expect this to lead to overfitting, but when enough data is available it might help to find a closer match to the expert.

References

[1] Pieter Abbeel, Dmitri Dolgov, Andrew Ng, and Sebastian Thrun. Apprenticeship learning for motion planning, with application to parking lot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-08)*, Nice, France, September 2008. IEEE.

[2] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning.* ACM Press, 2004.