Apprenticeship Learning for Motion Planning with Application to Parking Lot Navigation

Pieter Abbeel, Dmitri Dolgov, Andrew Y. Ng, Sebastian Thrun

***Abstract*— Motion and path-planning algorithms often use complex cost functions for both global navigation and local smoothing of trajectories. Obtaining good results typically requires carefully hand-engineering the trade-offs between different terms in the cost function. In practice, it is often much easier to demonstrate a few good trajectories. In this paper, we describe an efficient algorithm which—when given access to a few trajectory demonstrations—can automatically infer good trade-offs between the different costs. In our experiments, we apply our algorithm to the problem of navigating a robotic car in a parking lot.**

I. INTRODUCTION

Path-planning algorithms often use complex cost functions (or potentials) for global navigation and local smoothing of trajectories [2], [8], [11], [9], [5], [3]. In practice, when designing a cost function for motion planning, we often have a large number of desiderata that contribute to possibly conflicting terms in the potential. For example, we might care about path smoothness, proximity to obstacles, maximum curvature, lane-keeping, etc. Moreover, we might not know the exact desired functional form for each of the desiderata, and hence include several potential-field terms corresponding to each of the desiderata. To completely specify the potential function for our robot motion-planning problem, we need to quantify exactly how we want to balance all these terms. In practice, this can be highly non-trivial, and it often requires a significant amount of hand-engineering to obtain the desired motion-planning results. At the same time, it is often easy to demonstrate a few good driving trajectories. Such examples inherently contain information about the desired trade-off.

In this paper we describe how the apprenticeship-learning techniques presented in [1] allow us to learn the trade-off between the different potential terms from trajectory demonstrations—thus alleviating the need for extensive hand-engineering. We also describe how prior information about the weighting of the potential-field terms can be incorporated into the learning algorithm.

In our experiments, we consider the problem of navigating a robotic car in a parking lot. Our path-planning algorithm is based on the algorithm used by the Stanford racing team in the DARPA Urban Challenge [7]. For the purposes of this work, we have extended the existing algorithm by introducing several new cost potentials, which allow us to model a wide range of natural driving styles. The planned trajectories greatly depend on the choice of the trade-off between the various potential-function terms. We test our algorithm by observing a human drive the car in a parking lot between various starting points and destinations. We then evaluate how accurately our algorithm plans a previously unseen trajectory relative to the trajectory followed by the human driver for the same task.

The remainder of this paper is organized as follows: Section II covers preliminaries and notation; Section III describes our algorithm; Section IV describes the parking lot navigation problem setup and the path-planning algorithm; Section V describes our experimental results; Section VI concludes with a discussion of our contribution and limitations and an outlook towards future work.

II. PRELIMINARIES AND NOTATION

A. Path-Planning as Optimization

We let S denote the state space of the robot. A trajectory or path $\mathbf{s}$ corresponds to a sequence of states. We formulate the path-planning problem as the problem of minimizing a potential over the path. We denote the potential-field terms by $\{\phi_k(\cdot)\}_{k=1}^p$. We let $w \in \mathbb{R}^p$ denote the vector of weights associated with these potential-field terms. The total potential field $\Phi(\mathbf{s})$ for a path $\mathbf{s}$ is then given by

$$\Phi(s) = \sum_{k=1}^{p} w_k \phi_k(\mathbf{s}).$$

Given a start state s_0 and a goal state s_G, the motion planning problem is given by:

$$\min_{\mathbf{s}\in\mathbf{S}} \Phi(\mathbf{s}). \tag{1}$$

Here $\mathbf{S}$ denotes the set of allowable state sequences. To be an allowable state sequence, the path $\mathbf{s}$ needs to start at the start state s_0 and finish at the goal state s_G. There might also be other requirements such as, e.g., sequential states can be at most a certain distance apart.

In practice, many of the potential-field terms can be decomposed as a sum of potential-field terms, each of which only depends on the state at a single time t. However, such a decomposition is not required for the algorithm we present, and it will not hold for most of the potential terms we use for path planning.

We denote a potential-field motion planning problem by the tuple $M = (\mathbf{S}, s_0, s_G, \{\phi_k(\cdot)\}_{k=1}^p, w)$, and we denote a motion planning problem minus the weight vector w by $\bar{M}$.

Pieter Abbeel, Andrew Y. Ng and Sebastian Thrun are with the Computer Science Department, Stanford University, Stanford CA 94305 {pabbeel,ang,thrun}@cs.stanford.edu

Dmitri Dolgov is with Toyota Research Institute, USA ddolgov@ai.stanford.edu

In general, the terms $\{\phi_k(\cdot)\}_{k=1}^p$ define a complex potential, leading to a non-linear multi-modal optimization landscape in Eqn. (1).

The optimization algorithms therefore tend to be problem specific and vary greatly, depending on the functional form of the various potential-field terms $\{\phi_k(\cdot)\}_{k=1}^p$. We describe the exact potential used in our problem and our solution algorithm in Section IV.

B. Apprenticeship Learning and Potential Fields

In the apprenticeship-learning setting, we are given a set of m motion planning problems without the weight vector $\{\bar{M}^{(i)}\}_{i=1}^m$, and a set of corresponding expert demonstration trajectories, denoted by $\{\mathbf{s_E}^{(i)}\}_{i=1}^m$.

Throughout, we let $\mu_k(\{\mathbf{s}^{(i)}\}) = \sum_{i=1}^m \phi_k(\mathbf{s}^{(i)})$. Thus μ_k is a vector with the cumulative value over all trajectories $\{\mathbf{s}^{(i)}\}$ of each of the potential field terms ϕ_k.

III. ALGORITHM

In this section, we describe the adaptation of the apprenticeship learning algorithm from [1], which was originally formulated in the Markov decision process (MDP) setting, to the potential-field motion-planning setting. In essence the algorithm in [1] solves an inverse optimization problem: given the expert demonstrations, it finds a set of weights for the reward function (in our setting, for the potential function) such that the optimal policy (path) with respect to the resulting reward function (potential function) is close to the expert's policy (path). Here closeness is measured by closeness between the cumulative values of the potential functions for the expert and for the policy (path). For example, if the lengths of the forward driving segments and the backward driving segments were the only two potential field terms, then two paths (going from the same starting point to the same goal point) are considered close when they have a similar amount of forward driving and a similar amount of backward driving.

A straightforward adaptation of the apprenticeship-learning algorithm in [1] gives us the following algorithm to learn the potential-term weights from demonstrations:

Our algorithm takes as input: $\{\bar{M}^{(i)}\}_{i=1}^m$, $\{\mathbf{s}_{\mathbf{E}}^{(\mathbf{i})}\}_{i=1}^m$ and, optionally, a (convex) set $\mathcal{W}$, which describes our prior knowledge on the weight vector w. It then proceeds as follows:[1]

1) Randomly pick a weight vector $w^{(0)}$. Set $j = 0$.
2) Solve the potential-field motion-planning problems for the current weight vector $w^{(j)}$, i.e., find
$$\mathbf{s}^{(i)} = \arg\min_{\mathbf{s}} \Phi_{w^{(j)}}^{(i)}(\mathbf{s}).$$
3) Compute the cumulative values of the potentials:
$$\mu_k^{(j)} = \sum_{i=1}^m \phi_k(\mathbf{s}^{(i)})$$
4) Find the next estimate for the weight vector $w^{(j+1)}$ as the solution to the following convex optimization problem:
$$\begin{aligned} \min_{w,x} \quad & \|w\|_2^2 \\ \text{s.t.} \quad & \mu = \sum_j x_j \mu^{(j)}; \quad x \geq 0; \\ & \sum_j x_j = 1; \quad w \geq 0 \\ & w \geq \mu - \mu_E; \quad w \in \mathcal{W} \end{aligned} \tag{2}$$
If $\|w\| \leq \epsilon$ for some desired accuracy ϵ, then exit, and return $x, \{w^{(0)}, \mu^{(0)}, \ldots, w^{(j)}, \mu^{(j)}\}$. Otherwise, set $j = j + 1$, set $w^{(j)} = \frac{w}{\|w\|}$ and go to Step 2.

Crudely speaking, the algorithm alternates between (smartly) "guessing" a new weight vector, and solving the motion planning problems for this weight vector. The former problem merely requires solving a convex optimization problem, which can be done efficiently (see, e.g., [4] for more details on convex optimization).

Note that our formulation for guessing the new weight vector is a variation on the formulation used in [1]. In particular, in [1], the last three constraints are not used, but replaced with a single constraint $w = \mu - \mu_E$. The constraints $w \geq 0, w \geq \mu - \mu_E$ encode the fact that we know the weights are positive, and the contributions of the various potential terms to the distance are non-zero only when the expert is outperforming the current best path $\mu^{(j)}$.[2] The constraint $w \in \mathcal{W}$ allows us to encode additional prior information. For example, in our experiments we encode the prior information that the weight for the potential term corresponding to backward driving has to be at least as high as the one for forward driving.

When the algorithm exits, we have that $\|\mu - \mu_E\| \leq \|w\| \leq \epsilon$. Hence, when stochastically choosing (according to x) between the paths found throughout the iterations of the algorithm, we can perform as well as the expert up to some accuracy ϵ. To generalize to a new setting, we can correspondingly stochastically choose between the weight vectors $\{w^{(0)}, \ldots, w^{(j)}\}$ according to x, and then solve the resulting path-planning problem. In practice, it is often undesirable to stochastically mix. Instead, we could inspect the paths obtained for the vectors $w^{(j)}$ for which $x(j) > 0$. From convex analysis (see, e.g., [12]) we are guarenteed that the optimization problem in Step 4 has a solution with at most $p + 1$ non-zero entries; we also have that at least one of them performs as well as the expert.[3] See [1] for details.

IV. PATH PLANNING

An algorithm capable of generating human-like trajectories in parking lots must model a cost that takes into account

[1] The algorithm presented corresponds closely to the max-margin version of [1].

[2] Similar to the variation on [1] presented in [13], this allows us to outperform the expert. By contrast in [1] the sign of each potential term is not know, hence the algorithm learns to "match" the expert's behaviour, rather than learning to perform equally well or better.

[3] By contrast, if one were to find a good weight vector by gridding the p dimensional weight vector space, one would end up having a number of trajectories that grows exponentially in the dimensionality p.

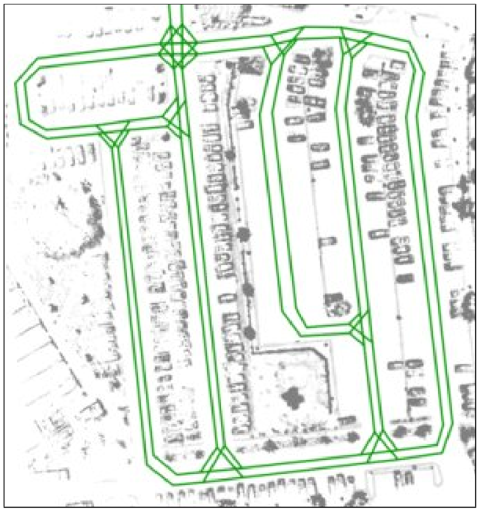

Fig. 1. A graph of driving lanes ($\mathcal{G}$) in a parking lot.

a wide variety of factors such as the following:

- The total length of the trajectory.
- The length of trajectory segments driven in reverse.
- The number of times the direction of motion switches from forward to reverse.
- The proximity of the trajectory to obstacles.
- A measure of smoothness (or aggregate curvature) of the trajectory.
- A measure of distance between the trajectory and the driving lanes in the environment.
- A measure of alignment of the trajectory with the principal driving directions of the parking lot.

The feature corresponding to the distance between the trajectory and the driving lanes is needed to differentiate between drivers that tend to cut across open space in parking lots and those that stay in the appropriate lane until they reach their goal. A network of driving lanes for a typical parking lot is shown in Figure 1. We assume that such a graph is provided as input to the planner.

This feature corresponding to the alignment of the trajectory with the principal driving directions is needed to differentiate between drivers that cut corners to minimize curvature and drivers that take wider turns and prefer to drive along the main driving directions of a parking lot. Such principal driving directions can be automatically computed from sensor data [6]; in this work, we use the road network (as in Figure 1) to define such preferred directions of motion.

Let us define the kinematic state of the vehicle as $\langle \mathbf{x}, \theta, d\rangle$, where $\mathbf{x} = \langle x, y\rangle$ is the position of the vehicle, θ is its orientation, and $\delta = \{0, 1\}$ determines the direction of motion: forward ($\delta = 0$) or backwards ($\delta = 1$). Further, assume that the network of road lanes is given as a directed graph $\mathcal{G} = \langle V, E\rangle$, and let α_E be the angle of edge E. Let us define a distance between a point $\mathbf{x}$ and the graph $\mathcal{G}$:

$$\mathcal{D}(\mathbf{x}, \mathcal{G}) = \min_E \mathcal{D}(E, \mathbf{x}),$$

where $\mathcal{D}(E, \mathbf{x})$ is the 2D Euclidean distance between a point and a line segment. Also, let us define a distance between an *oriented* point $\langle \mathbf{x}, \theta\rangle$ and the graph $\mathcal{G}$:

$$\mathcal{D}(\mathbf{x}, \theta, \mathcal{G}) = \min_{\{E : |\alpha_E - \theta| < \alpha_{min}\}} \mathcal{D}(E, \mathbf{x}),$$

i.e., the distance to the nearest edge whose angle is close—within α_{min}—to the heading of the car θ.

Further, define an indicator function $R(s)$, where $R(s) = 1$ if the car is on the road, i.e., the 2D euclidean distance between $\mathbf{x}_i$ and $\mathcal{G}$ is below a given threshold: $R(s) = 1 \iff \mathcal{D}(\mathbf{x}, \mathcal{G}) < \mathcal{D}_{road}$.

Finally, let $\alpha_i = \alpha\big(\arg\min_E(\mathcal{D}(E, \mathbf{x}_i))\big)$ be the angle of the edge E nearest to the trajectory point $\mathbf{x}_i$.

The objective of the path planner is to minimize the following potential defined over a trajectory $\mathbf{s} = \{\langle \mathbf{x}_i, \theta_i, \delta_i\rangle\}$:[4]

$$\begin{aligned}
&w_{fwd} \sum_{i:i>1,\delta_i=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + \\
&w_{rev} \sum_{i:i>1,\delta_i=1} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + w_{sw} \sum_{i:\delta_i \neq \delta_{i-1}} 1 + \\
&w_{road} \sum_{i:R(s_i)=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + w_{lane} \sum_i \mathcal{D}(\mathbf{x}_i, \theta_i, \mathcal{G}) + \\
&w_{dir} \sum_i \sin^2(2(\theta_i - \alpha_i)) + w_{curv} \sum_{i>1, i<|\mathbf{s}|} (\Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_i)^2,
\end{aligned} \tag{3}$$

where $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$. The terms above respectively correspond to: 1) length of trajectory driven forward, 2) length of trajectory drive in reverse, 3) number of times the direction of motion switches, 4) length of trajectory driven off-road, 5) an aggregate distance of the trajectory to the road-lane graph, 6) a measure of misalignment of trajectory and the principal directions of the parking lot, and 7) a measure of smoothness of the trajectory.

The weights in Eqn. 3 define the weight vector w introduced in Section II and are used in learning.

The path-planning problem defined above is a complex continuous-coordinate optimization program with multiple local minima. For computational reasons, we therefore follow the two-phase approach described in [7]. The first phase performs an approximate discrete global search that finds a solution in a neighborhood of the global optimum; the second phase then fine-tunes the solution in continuous coordinates.

A. Global Search

Our first phase uses a variant of A* search with a discrete set of control actions, applied to the 4-dimensional kinematic state of the vehicle defined above. As this phase uses a highly discretized set of control actions, it cannot cleanly accommodate the potential terms in Eqn. 3 that correspond to local properties of the trajectory (smoothness, alignment). As such, the first phase focuses on a subset of features that

[4] In practice, the potential will also contain terms corresponding to hard constraints such as collision avoidance, minimum turning radius of the vehicle, etc. Since these constraints must be satisfied regardless of driving style, we fix their weights at large values (orders of magnitude higher than other terms) and do not include them in learning. See a more thorough description of the core of our path-planning algorithm [7] for details on modeling and implementing such constraints in optimization.

affect global behavior. The local features are used in the subsequent second phase of the optimization algorithm.

The main components that define the behavior of A* are the cost of a partial solution and the cost-to-go heuristics. The heuristics are described in [7]. The cost function is defined by the terms of the potential in Eqn. 3 that correspond to the global features with the following weights: $\langle w_{fwd}, w_{rev}, w_{sw}, w_{road}, w_{lane} \rangle$.

Due to coarse discretization used in our global search and for computational reasons, we replace the continuous-coordinate version of the lane-attraction potential with a discrete version similar to the on-road potential. Let us define an indicator function $L(s) = 1$ if the car is in the correct lane, i.e., the distance between the (oriented) car and the lane graph $\mathcal{G}$ is below a given threshold: $L(s) = 1 \iff \mathcal{D}(\mathbf{x}, \theta, \mathcal{G}) < \mathcal{D}_{lane}$.

The lane-keeping potential is approximated as follows:

$$w'_{lane} \sum_{i:i>1, L(s_i)=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\|,$$

i.e., this term computes a (weighted) length of the path driven out of lane.

B. Local Smoothing

For computational reasons, the global A* uses a highly discretized set of control actions, leading to paths that are suboptimal. The second phase of our algorithm improves the quality of the solution by using conjugate-gradient, a very efficient numerical continuous-coordinate optimization technique.

The input to the smoother is the trajectory produced by A* as defined in the previous section. Since the global behavior of the trajectory is already determined, the global features of the potential in Eqn. 3 are meaningless in the second phase of our algorithm, since it only performs local adjustment.

The second phase therefore uses the potential terms corresponding to the weights $\langle w_{dir}, w_{curv}, w_{lane} \rangle$, and the optimization is performed using conjugate-gradient descent. The implementation of conjugate-gradient requires a gradient of the objective function, which can be computed analytically for all of these terms.[5].

C. Trajectory Examples

The features described above–in both phase I and phase II—allow our path-planing algorithm to mimic a wide variety of human driving styles, which can be attained by different setting of the weights w. Figure 2 demonstrates several representative examples of trajectories. In all of the shown examples, the start and goal states—defined by their respective locations and orientations $\langle x, y, \theta \rangle$—are the same; the difference is only in the settings of weights $w = \langle w_{fwd}, w_{rev}, w_{sw}, w_{road}, w_{lane}, w'_{lane}, w_{curv} \rangle$. In this figure and others the initial state is shown as a car outline, and the goal state is s shaded rectangle.

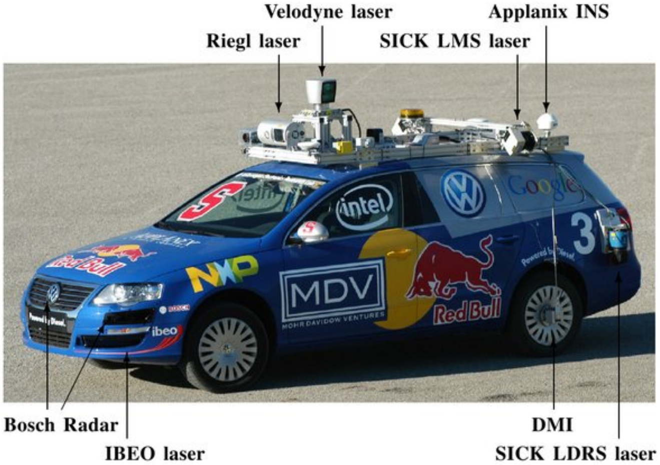

Fig. 3. All data used in this work was gathered using the Stanford Racing Team's robotic vehicle, Junior. Junior is equipped with several LIDAR and RADAR units, and a high-accuracy inertial measurement system. The training data for the algorithm described in this paper was obtained by manually driving Junior and recording it sensor readings. Picture provided at url mentioned below because of paper-size limits.

In this figure and in the rest of the paper, we use the following legend. Gray objects are obstacles; the initial state of the car is denoted by a single rectangle; the goal state is denoted by several concentric rectangles; the path's x-y coordinates are shown by a colored line with dots according to the planner's time granularity. Regularly spaced (in time), we overlay a triangle over the path to show the car's heading. The colored dots are replaced by black dots whenever the car is not in its lane ($L(\mathbf{s}) = 0$). When the car goes "off-road" ($R(\mathbf{s}) = 0$), we use larger-sized black circles. The green lines show the graph of driving lanes $\mathcal{G}$.

Figure 2a corresponds to a setting of weights with a low penalty for switching directions (w_{sw}) and a low cost of driving in reverse w_{rev}). If we increase the penalty for switching directions and for driving in reverse, we obtain the path in Figure 2b, which takes a slightly longer route to the goal, but avoids driving in reverse. Increasing the weight on the alignment with principal directions of the parking (w_{dir}) leads to the path shown in Figure 2c, which chooses the same global path, but is more aligned with the lanes in the parking lot, although it still cuts across the row of parked cars. The latter behavior can be controlled with the penalty for driving off-road: a high setting of the corresponding weight (w_{road}) results in the trajectory shown in Figure 2d, which stays on-road. Notice that this path tends to not stay in the right lane; a high setting to the corresponding weights (w'_{lane}, w_{lane}) pushes the trajectory closer to the right lane, as shown in Figure 2e.

V. EXPERIMENTAL RESULTS

For our experimental evaluation, we used the Stanford Racing Team's robotic vehicle, Junior (Figure 3), which is equipped with several sensors modalities (RADAR, LIDAR, cameras) and a high-precision GPS+IMU system. (See [10] for a detailed description of Junior.) For the purposes of

[5] See [7] for more details about smoothing via conjugate gradient, as well as computing the gradient of such potentials

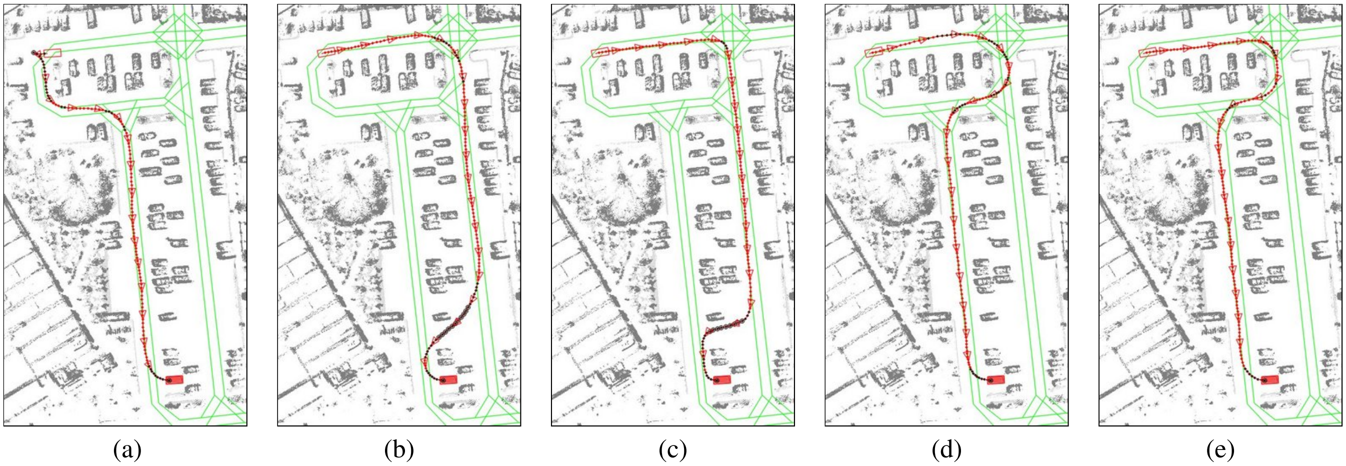

Fig. 2. The trajectories produced by our path-planning algorithm—when initialized with different weights—can mimic a wide range of driving styles with different global and local behavior.

our experiments, the autonomous-driving capabilities of the vehicle were not used, rather the car was driven by a human to collect training data. During such data-collection runs, we logged the messages from the pose-estimation GPS+IMU system as well as the messages from the car's 3D LIDAR, which allowed us to later reproduce the exact obstacle maps of the environment as well as the precise trajectories that were driven.

A. *Experiments*

We asked a human driver to navigate a parking lot using three very distinct driving styles:

- "Nice": we tell the driver to drive in the right lane whenever safely possible.
- "Sloppy": we tell the driver it is okay to deviate from the standard lanes. We also tell the driver to only use forward driving.
- "Backward": we allow the driver to drive backward, but only when it makes for a shorter path to the goal.

For each driving style, we collected five demonstrations and ran our learning algorithm five times: every time we learn from four of the demonstrations, and then evaluate performance on the left-out fifth demonstration. As the planner decomposes into two phases, and the features used in the two phases have no intersection, we run our algorithm in two phases: first run it for the first-phase (global) planner, then run it—using the weights learned in the first phase—to learn the weights for the second phase. Typically our algorithm converges to a good solution in 5 to 10 iterations.[6]

Figures 4, 5, 6, 7, 8, 9 show the nice expert demonstrations, the nice autonomous navigation results, the sloppy expert demonstrations, the sloppy autonomous navigation results, the backward expert demonstrations, and the backward autonomous navigations results respectively.

In these figures, we use the same markers on trajectories, as defined previously in Section IV. Expert demonstrations are shown in blue, while trajectories produced by our path planner are shown in red.

Inspecting the figures, we note that the learned navigation styles are very similar to the expert's styles. For example, whenever learning from a subset of four nice demonstratoins, it learns to keep the right-lane whenever possible. Whenever learning from a subset of four backward demonstrations, it learns that backward driving is allowed to make a shortcut, and successfully executes a shortcut on the left-out fifth navigation task. When learning from the sloppy driver, it successfully learns to make a shortcut through parking space whenever applicable. Interestingly, we learn similarity at the level of the cost terms. E.g., when learning to cut across, it might cut across at a different geographical location than the expert, since the geographical location of the shortcut does not contribute to the cost function.

Table I gives a quantitative evaluation of our experiments. For each of the 15 learning/testing experiments, we report the cumulative values of the cost functions of the expert and the learned planner on the training data, the values of the cumulative values of the cost functions in testing, and the learned weight vector. Inspecting the table, we note that indeed, our algorithm finds a set of weights such that both at training and test time the cumulative values of the cost functions are close to those obtained by the expert. Looking more closely at the weight functions learned, we observe the relative weightings for different driving qualitatively matches our intuition about these styles. For the nice driver the penalty for going backward, off-lane, or off-road is much higher than for the other two styles. The backward driving styles has a cost for going backwards that is as low as the cost for going forward. (Consistent with our constrains on

[6] As the planner is not optimal, it is possible for the planner to not find the optimal path for a certain guessed reward function, then our algorithm can only be shown to converge down to the accuracy of the planning algorithm. In our experiments, our learning algorithm tends to find good optima fairly consistently. In its full generality, the algorithm returns a set of weights, rather than a single weight. As explained in more detail in [1], one of these weights will enable one to perform as well as the expert. In our experiments, it turned out to be typically sufficient to simply pick the set of weights that resulted in the cumulative cost term counts to be closest to the expert amongst all iterations. In fact, out of the 15 experiments, only for one did this heuristic return a bad set of weights, and for this one we followed the strict procedure of inspecting and picking the best.

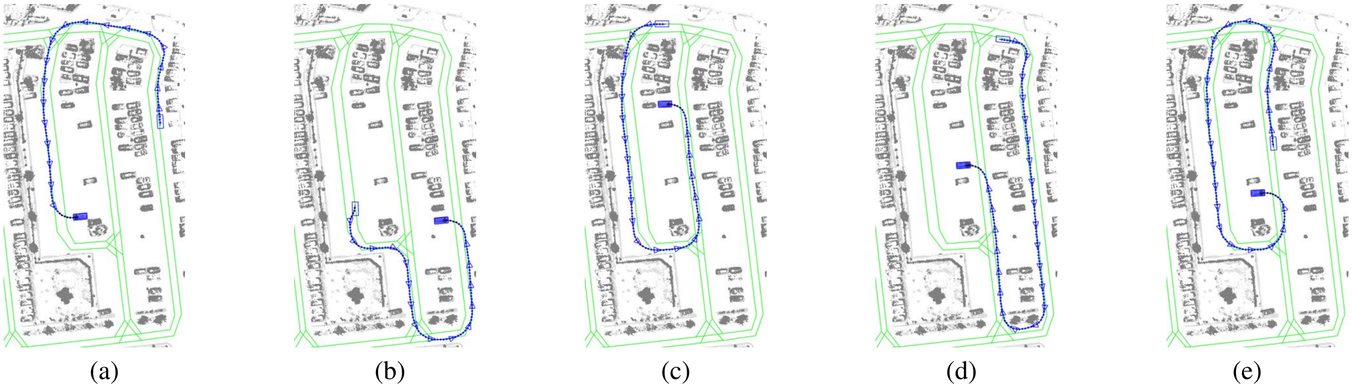
(a) (b) (c) (d) (e)

Fig. 4. “Nice” parking lot navigation driving: expert demonstrations. (See text for details.)

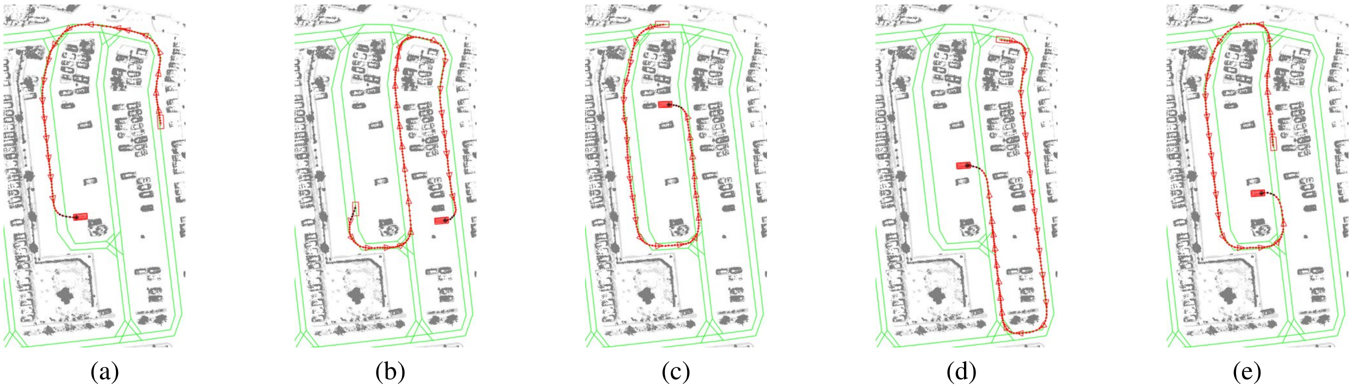
(a) (b) (c) (d) (e)

Fig. 5. “Nice” parking lot navigation driving: trajectories found by learning on four demonstrations, and testing on the fifth. (See text for details.)

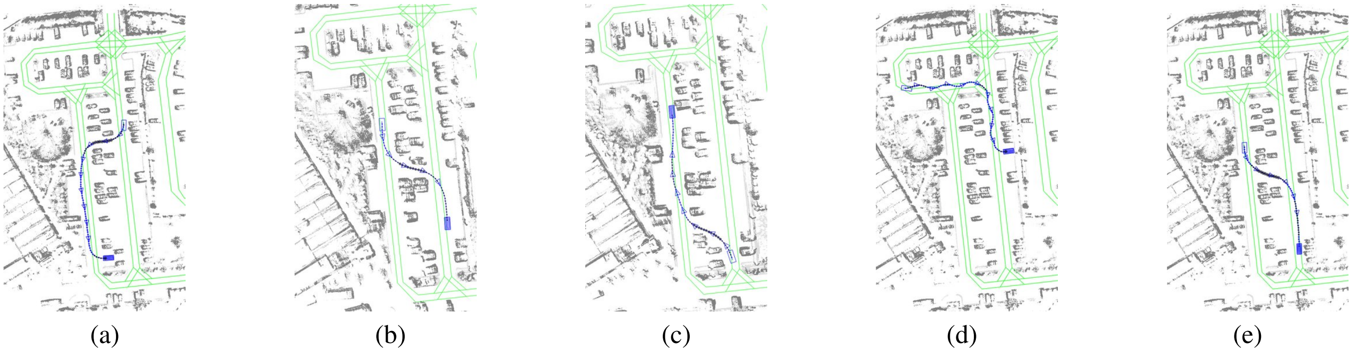
(a) (b) (c) (d) (e)

Fig. 6. “Sloppy” parking lot navigation driving: expert demonstrations. (See text for details.)

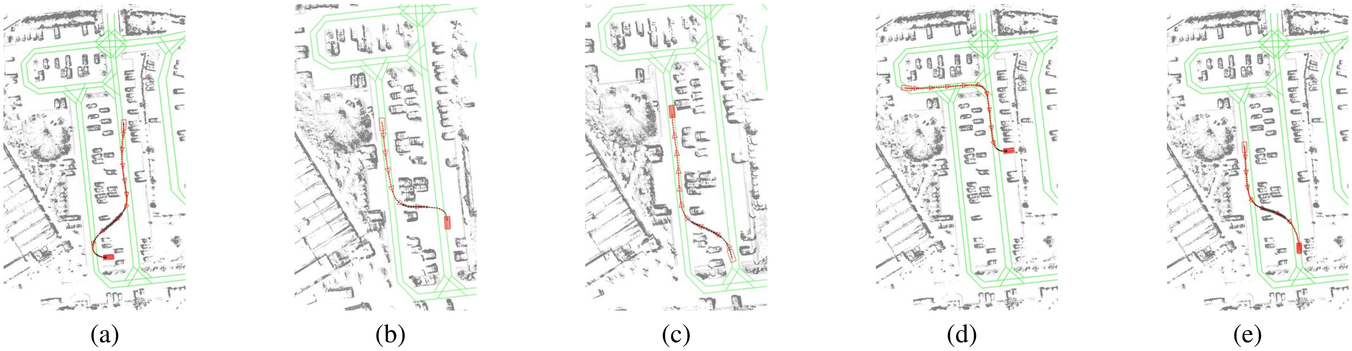
(a) (b) (c) (d) (e)

Fig. 7. “Sloppy” parking lot navigation driving: trajectories found by learning on four demonstrations, and testing on the fifth. (See text for details.)

the weights: $w \in \mathcal{W}$, which enforces that backward driving is at least as expensive as forward driving. This captures the fact we only want to learn about backward driving as a way to make a shortcut, not as a default driving style.)

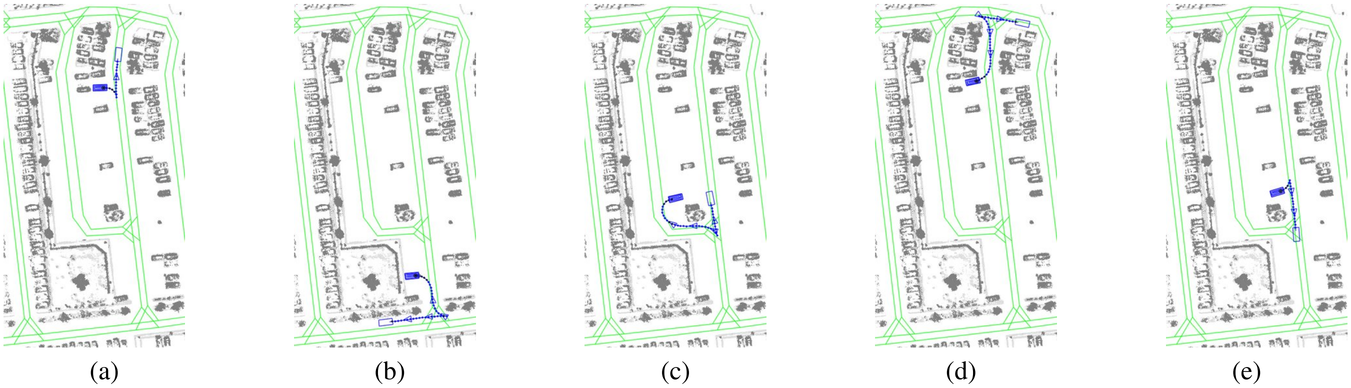
(a) (b) (c) (d) (e)

Fig. 8. "Backward" parking lot navigation driving: expert demonstrations. (See text for details.)

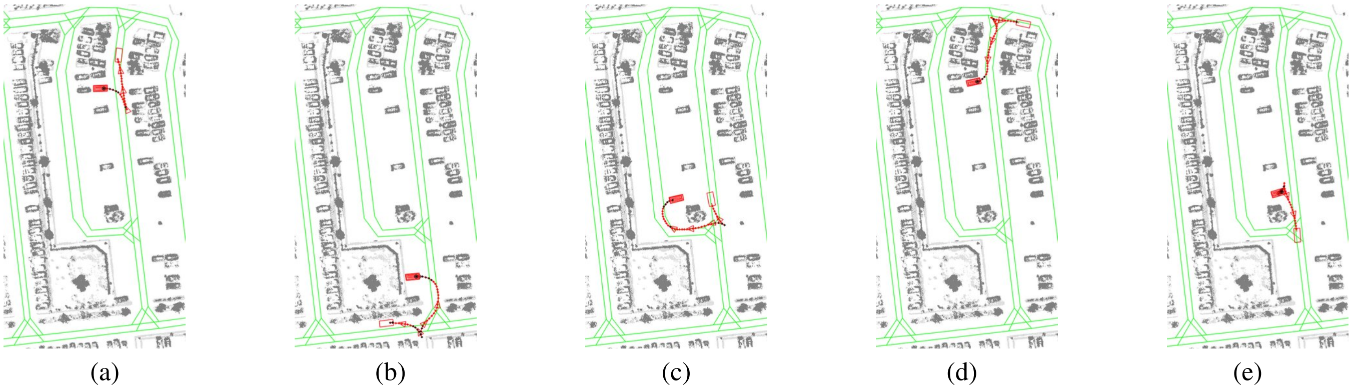
(a) (b) (c) (d) (e)

Fig. 9. "Backward" parking lot navigation driving: trajectories found by learning on four demonstrations, and testing on the fifth. (See text for details.)

Similar observations hold for the weights for other features. We also note the weight vector entries are fairly consistent over different training runs.

VI. DISCUSSION

Motion and path planning algorithms often rely on complex potentials for global navigation and local smoothing of trajectories. The trade-off between the different potential field terms greatly affect the results obtained. In this paper, we showed that we can efficiently learn a trade-off corresponding to expert demonstrations. We applied our algorithm to learn to navigate a parking lot similar to human drivers.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of Mike Montemerlo who helped with data collection and wrote much of Junior's software, Dirk Haehnel who wrote the core of map-generation code, and James Diebel for his conjugate-gradient library.

REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. ICML*, 2004.

[2] J. Andrews and N Hogan. Impedance control as a framework for implementing obstacle avoidance in a manipulator. *Control of Manufacturing Processes and Robotic Systems*, pages 243–251, 1983.

[3] R. C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, pages 92–112, August 1989.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] R. A Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[6] Dmitri Dolgov and Sebastian Thrun. Detection of principal directions in unknown environments for autonomous navigation. In *Proceedings of the Robotics: Science and Systems (RSS-08)*, Zurich, Switzerland, June 2008.

[7] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous driving in unknown environments. In *Proceedings of the Eleventh International Symposium on Experimental Robotics(ISER-08)*, Athens, Greece, July 2008.

[8] O Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IJRR*, 5(1):90–98, 1986.

[9] F. Miyazaki and S. Arimoto. Sensory feedback for robot manipulators. *Journal of Robotic Sys.*, 2(1):53–71, 1985.

[10] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. The stanford entry in the urban challenge. *Journal of Field Robotics*, 2008.

[11] V. Pavlov and A. N. Voronin. The method of potential functions for coding constraints of the external space in an intelligent mobile robot. *Soviet Automatic Control*, 17(6):45–51, 1984.

[12] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[13] U. Syed and R. E. Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS 20*, 2008.

TABLE I
DETAILED EXPERIMENTAL RESULTS. (SEE TEXT FOR DETAILS.)

		Forward	Reverse	Switch Fw/Bw	Off-Road	Lane A*	Curvature	Principal Directions	Lane
		w_{fwd}	w_{rev}	w_{sw}	w_{road}	w'_{lane}	w_{curv}	w_{dir}	w_{lane}
Nice 1	μ_E (train)	160.8239	0	0	1.2500	9.0000	1.3263	51.6937	25.4831
	μ (train)	155.3227	0	0	1.2500	9.2500	1.3549	49.8545	15.3663
	μ_E (test)	140.7808	0	0	1.0000	7.0000	0.8323	40.2750	18.4281
	μ (test)	136.9459	0	0	1.0000	6.0000	1.2000	38.7015	4.1304
	w	1.0000	96.8481	2.4397	4.3915	24.3970	100.0000	1.3551	2.0000
Nice 2	μ_E (train)	165.4818	0	0	1.2500	8.2500	1.1612	47.4633	24.2449
	μ (train)	160.8447	0	0	1.2500	6.7500	1.1530	43.8947	11.5113
	μ_E (test)	122.1495	0	0	1.0000	10.0000	1.4927	57.1965	23.3810
	μ (test)	183.9400	0	0	1.0000	10.0000	1.7211	56.1258	13.7980
	w	1.0000	94.2028	2.3731	10.4415	31.7990	100.0000	2.0000	1.6676
Nice 3	μ_E (train)	155.8637	0	0	1.2500	8.2500	1.2611	50.5259	21.5989
	μ (train)	150.6531	0	0	1.2500	9.5000	0.8679	41.9264	19.7580
	μ_E (test)	160.6220	0	0	1.0000	10.0000	1.0928	44.9460	33.9650
	μ (test)	155.7224	0	0	1.0000	6.0000	0.8218	43.6927	22.1467
	w	1.0000	83.1622	2.3952	17.3654	52.6949	100.0000	2.0000	0.3615
Nice 4	μ_E (train)	150.7121	0	0	1.0000	9.2500	1.1807	52.9129	24.7445
	μ (train)	142.6400	0	0	1.2500	9.2500	1.4102	49.3730	11.9980
	μ_E (test)	181.2283	0	0	2.0000	6.0000	1.4146	35.3982	21.3825
	μ (test)	185.3469	0	0	2.0000	5.0000	1.2997	34.1287	10.4651
	w	1.0000	83.9357	1.9501	19.5009	50.7023	100.0000	2.0000	1.9808
Nice 5	μ_E (train)	151.1952	0	0	1.2500	8.2500	1.2081	44.4539	24.2892
	μ (train)	147.0585	0	0	1.2500	9.7500	1.4697	41.9111	13.0692
	μ_E (test)	179.2960	0	0	1.0000	10.0000	1.3050	69.2340	23.2038
	μ (test)	175.4344	0	0	1.0000	5.0000	1.4308	55.8842	10.8906
	w	1.0000	57.4956	11.5709	1.0000	80.9961	100.0000	2.0000	1.7391
Sloppy 1	μ_E (train)	61.7910	0	0	8.5000	29.5000	0.6136	41.0509	32.0760
	μ (train)	61.8828	0	0	8.5000	19.0000	0.2527	33.8667	15.4792
	μ_E (test)	78.7679	0	0	10.0000	28.0000	0.7656	35.5559	25.1425
	μ (test)	71.9947	0	0	11.0000	27.0000	0.3433	54.2620	15.7552
	w	1.0000	1.0000	0.0309	2.7503	0.0309	100.0000	0.1000	0.1000
Sloppy 2	μ_E (train)	68.8222	0	0	8.5000	28.2500	0.7353	40.1163	31.7714
	μ (train)	66.3593	0	0	9.5000	19.7500	0.3750	22.4864	19.6623
	μ_E (test)	50.6431	0	0	10.0000	33.0000	0.2789	39.2942	26.3608
	μ (test)	53.8260	0	0	10.0000	25.0000	0.3635	22.3366	16.6148
	w	1.0000	2.0000	2.0000	2.0000	0.0010	100.0000	2.0000	0.0200
Sloppy 3	μ_E (train)	65.1835	0	0	8.0000	28.2500	0.7469	39.1279	31.5402
	μ (train)	63.5014	0	0	8.5000	19.5000	0.2923	37.5554	14.5943
	μ_E (test)	65.1980	0	0	12.0000	33.0000	0.2324	43.2479	27.2857
	μ (test)	65.2282	0	0	13.0000	26.0000	0.1277	41.8750	16.1057
	w	1.0000	2.0000	2.0000	2.0000	0.0010	100.0000	0.1000	0.1000
Sloppy 4	μ_E (train)	63.2560	0	0	10.7500	31.7500	0.3912	39.3911	26.7967
	μ (train)	62.2515	0	0	11.5000	24.5000	0.3775	23.2818	16.1718
	μ_E (test)	72.9078	0	0	1.0000	19.0000	1.6552	42.1948	46.2597
	μ (test)	70.4276	0	0	1.0000	7.0000	0.3539	19.1550	30.5769
	w	1.0000	2.0000	2.0000	2.0000	0.0010	100.0000	2.0000	0.0200
Sloppy 5	μ_E (train)	66.8792	0	0	8.2500	28.2500	0.7330	40.0732	31.2622
	μ (train)	65.2396	0	0	8.7500	21.0000	0.2883	36.9720	16.6293
	μ_E (test)	58.4152	0	0	11.0000	33.0000	0.2879	39.4666	28.3976
	μ (test)	58.2752	0	0	12.0000	20.0000	0.1436	44.2084	7.9657
	w	1.0000	2.0000	2.0000	2.0000	0.0010	100.0000	0.1000	0.1000
Backward 1	μ_E (train)	19.9921	157.6950	1.5000	1.0000	5.7500	4.8092	16.2583	17.0732
	μ (train)	26.6345	124.0380	2.7500	1.0000	8.5000	12.3666	49.9002	38.5879
	μ_E (test)	6.1878	134.9460	2.0000	1.0000	5.0000	4.5689	7.6401	12.0153
	μ (test)	12.8795	187.6520	4.0000	1.0000	7.0000	15.4634	63.7094	32.3790
	w	1.0000	1.0000	0.1075	0.6447	2.0416	100.0000	2.0000	1.3070
Backward 2	μ_E (train)	16.5697	141.5435	1.5000	0.7500	5.2500	4.7577	14.4087	13.7456
	μ (train)	23.2810	84.5415	2.2500	0.7500	12.0000	10.9245	36.4021	28.2000
	μ_E (test)	19.8776	199.5520	2.0000	2.0000	7.0000	4.7747	15.0385	25.3258
	μ (test)	28.8845	139.8750	4.0000	2.0000	19.0000	18.3982	100.6706	68.6810
	w	1.0000	1.0000	0.4925	0.3612	1.3517	100.0000	2.0000	1.4534
Backward 3	μ_E (train)	14.4007	162.3215	1.7500	1.2500	5.5000	4.6427	12.2424	16.8130
	μ (train)	26.7334	74.7647	5.0000	1.2500	16.0000	13.2958	89.2681	87.5404
	μ_E (test)	28.5533	116.4400	1.0000	0	6.0000	5.2350	23.7037	13.0560
	μ (test)	31.1568	89.0720	1.0000	0	11.0000	4.6965	28.0617	16.9520
	w	1.0000	1.0000	0.0365	0.1459	0.9481	100.0000	2.0000	1.8103
Backward 4	μ_E (train)	14.9579	157.1987	1.5000	1.0000	5.5000	4.7558	13.1276	14.1942
	μ (train)	23.5130	119.3055	3.7500	1.0000	13.0000	10.9358	64.2291	39.8530
	μ_E (test)	26.3246	136.9310	2.0000	1.0000	6.0000	4.7824	20.1630	23.5315
	μ (test)	24.9511	99.2160	2.0000	1.0000	6.0000	8.8873	35.6120	29.0954
	w	1.0000	1.0000	0.0681	0.6129	1.7026	100.0000	2.0000	1.1220
Backward 5	μ_E (train)	20.2358	146.9673	1.7500	1.0000	6.0000	4.8403	16.6363	18.4822
	μ (train)	21.8514	87.2020	2.0000	3.5000	16.5000	5.4475	28.0484	21.3451
	μ_E (test)	5.2129	177.8570	1.0000	1.0000	4.0000	4.4446	6.1280	6.3796
	μ (test)	4.2384	186.8070	3.0000	3.0000	5.0000	15.7824	16.5083	30.0406
	w	1.0000	1.0000	0.1547	1.5475	6.4993	100.0000	2.0000	0.1217