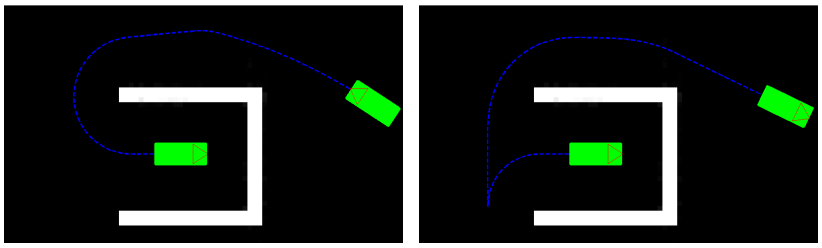
Learning Autonomous Driving Styles and Maneuvers from Expert Demonstration

David Silver, J. Andrew Bagnell and Anthony Stentz
Carnegie Mellon University

Abstract One of the many challenges in building robust and reliable autonomous systems is the large number of parameters and settings such systems often entail. The traditional approach to this task is simply to have system experts hand tune various parameter settings, and then validate them through simulation, offline playback, and field testing. However, this approach is tedious and time consuming for the expert, and typically produces subpar performance that does not generalize. Machine learning offers a solution to this problem in the form of learning from demonstration. Rather than ask an expert to explicitly encode his own preferences, he must simply demonstrate them, allowing the system to autonomously configure itself accordingly. This work extends this approach to the task of learning driving styles and maneuver preferences for an autonomous vehicle. Head to head experiments in simulation and with a live autonomous system demonstrate that this approach produces better autonomous performance, and with less expert interaction, than traditional hand tuning.

1 Introduction

Building truly robust and reliable autonomous navigation systems remains a challenge to the robotics community. One of the many barriers to successful deployment of such systems is the large number of parameters and settings they often entail, with robust behavior dependent on correct determination of these values. While it is difficult enough to properly build and parameterize the systems themselves, it is even harder to determine the correct settings that will properly encode desired behavior in known scenarios, while also generalizing to the unknown. The traditional approach to this task involves system experts hand tuning various parameters, which then must be validated through actual system performance. However, this approach is tedious and time consuming, while typically resulting in poor system performance that does not generalize. Machine learning, specifically learning from demonstration, offers



(a) No Maneuver Preferences (shortest path) (b) Desired Behavior

Fig. 1 A simulated example in a binary environment, demonstrating the necessity of preferences over maneuvers as well as chosen paths. In this example, simply following the shortest path (a) leads to a long drive in reverse. Especially for vehicles with sensors only in front, a longer turning maneuver (b) that drives in the forward direction is often preferable.

a potential solution to this problem. Rather than ask an expert to explicitly encode his own preferences, he must simply demonstrate them, allowing the system to autonomously configure itself accordingly. Recent work has demonstrated that this approach can both reduce the amount of expert interaction required, while improving the resulting system performance.

In the domain of autonomous navigation, this learning approach has mostly focused on the task of analyzing perceptual information and determining the preferability of traversing various sections of terrain. However, a robot's planning system must determine the best plan that not only considers these preferences, but also more dynamic considerations such as velocities or accelerations on a vehicle, stability, etc. This latter problem involves its own set of parameters and tuning, and the coupled problem requires proper balancing of the tradeoffs of the component problems. Failure to properly tune the overall system can result in poor performance. For example, a vehicle that is too averse to swerving may be too willing to traverse certain obstacles it should avoid, while one that is too willing to swerve will drive in a jittery fashion. Given that both of these considerations can be scalars derived from high dimensional vectors, it can be very difficult to properly tune all the necessary parameters by hand. Since a real system requires tuning for an enormous set of such problems, producing a truly robust system by hand approaches infeasibility.

This work extends the approach of [14] to the task of learning driving styles and maneuver preferences for an autonomous vehicle. That is, it not only learns *where* a robot should drive, it also learns *how* to drive. Solving these coupled problems from the same training inputs ensures the the resulting parameterization produces good system performance when applied in an online setting. The next section discusses related work in both autonomous navigation and learning from demonstration. Sections 3 and 4 discuss learning cost functions from demonstration, and extend known approaches to the coupled problem of learning maneuvers and driving styles. This approach is then validated through experimental results presented in Section 5.

2 Related Work

Autonomous navigation is generally framed as finding the lowest cost (i.e. optimal) feasible sequence from a start to a goal through some state space. These states could represent locations in the world, configurations of the robot, actions the robot could perform, or some combination thereof. The cost of a plan is usually defined as the sum of costs of individual states. To allow for generalization, costs are usually not explicitly assigned to specific states, but are rather produced as a function of features describing individual states. The function mapping features of states to costs essentially encodes the preferences that the robot will exhibit; therefore its configuration will have a dramatic impact on the robot's performance. Historically, the mapping of features to costs has been performed by simple manual construction of a parameterized function, and then hand tuning various parameters to create a cost function that produces desired behavior. This manual approach has been frequently used whether the cost functions in question describe locations in the world [8, 16, 17] or actions to be performed [4, 11, 19]. Unfortunately, this tedious approach typically produces subpar results, potentially leading to subpar autonomous performance.

Supervised learning is a popular solution to such tuning tasks, by automatically adjusting parameters to meet desired criteria. As opposed to just learning parameters within a specific component of an autonomous system (e.g. Terrain Classification), learning from demonstration seeks to learn parameters to directly modify end system behavior. Traditional learning from demonstration [3] seeks to learn a mapping directly from states, or features of states, to actions. The advantage of this model free formulation is that it creates a straightforward learning task. However, this comes at the cost of difficulty with both generalization to new problems, and sequentially combining decisions to achieve longer horizon planning.

Recently, model based approaches to learning from demonstration have become more popular. In contrast to model free, model based learning continues to use the optimal planning algorithms popular in navigation, and seeks to learn a feature to cost mapping such that the optimal plan in a given scenario produces desired behavior; in this way these approaches are essentially applications of inverse optimal control. Numerous applications of this approach [7, 9, 14, 20] have demonstrated its effectiveness at learning navigation cost functions over patches of terrain, improving autonomous performance and reducing the need for manual tuning.

Although learning from demonstration has been previously applied to the task of learning preferences over actions and trajectories [1, 6, 18], this work has generally been focused on more short range vehicle control and path tracking. A notable exception is [2], which investigated learning driving preferences in a parking lot from demonstration. However, the coupled problem of learning both terrain and driving preferences has not been previously addressed.

3 Learning Navigation Cost Functions from Demonstration

In previous work, the Maximum Margin Planning (MMP) framework [12, 14] was developed to allow learning cost functions from expert demonstration. MMP seeks to find the simplest cost function such that an example plan P_e is lower cost than all other plans, and by a margin. The margin is based on a loss function L_e that encodes the similarity of a plan to the example. Rather than enforce this constraint over all possible plans, it is only necessary to enforce it against the current minimum cost plan P_* under the current cost function C. Thus, MMP can be represented as a constrained optimization problem

$$\begin{aligned} \text{minimize } O[C] = |C| \quad \text{subject to} \\ \sum_{x \in P_*} (C(F_x) - L_e(x)) \geq \sum_{x \in P_e} (C(F_x)) \\ P_* = \arg\min_P \sum_{x \in P} (C(F_x) - L_e(x)) \end{aligned} \tag{1}$$

F_x represents the feature vector describing state x in some planning state space. Since it is generally not possible to meet this constraint, a slack term ζ is added, and the constraint is rewritten as

$$\begin{aligned} \text{minimize } O[C] = \lambda |C| + \zeta \quad \text{subject to} \\ \sum_{x \in P_*} (C(F_x) - L_e(x)) - \sum_{x \in P_e} (C(F_x)) + \zeta \geq 0 \end{aligned} \tag{2}$$

Since ζ is in the minimizer, it will always be tight against the constraint and equal to the difference in costs. Therefore, ζ can be replaced by this difference to produce an unconstrained optimization

$$\text{minimize } O[C] = \lambda |C| + \sum_{x \in P_e} (C(F_x)) - \sum_{x \in P_*} (C(F_x) - L_e(x)) \tag{3}$$

This objective can be optimized by (sub)gradient descent, using the subgradient

$$\nabla O[C] = \lambda \nabla |C| + \sum_{x \in P_e} \delta_F(F_x) - \sum_{x \in P_*} \delta_F(F_x) \tag{4}$$

Simply speaking, the subgradient is positive at values of F corresponding to states in the example plan, and negative at values of F corresponding to states in the current plan. Applying gradient descent directly would involve raising or lowering the cost at specific values of F, according to the (negative) gradient. To encourage generalization and limit $|C|$, this gradient can instead be projected onto a limited set of directions. As in [10, 12] this projection takes the form of learning a classifier or regressor to differentiate between states whose cost needs to be raised or lowered, and then adding this learner to the current cost function.

The final algorithm, known as LEArning to seaRCH (LEARCH) [12] iteratively computes the set of states (under the current cost function) whose cost should be raised or lowered, computes a learner to reproduce and generalize this distinction,

and adds this learner to the cost function. The result is a cost function that is the weighted sum of set of learners (of whatever form and complexity is desired).

Real world motion planning systems, when fed by a stream of onboard perceptual data, generally recompute their plan several times a second to account for the dynamics of and errors in sensing, positioning, and control. As a result, applying LEARCH to an actual vehicle requires an understanding of these dynamics with respect to expert demonstration. As a consequence, rather than treat a single demonstration as a single example, it must be considered as a large set of examples, which are chained together by actual robot motion and the passage of time. Each single example at an instant in time forms its own constraints as in (3), with the full problem the sum of these individual optimizations. Solving this problem therefore involves summing the individual gradients steps from (4) into a single learner update. For more details on the application of this approach to real world navigation problems with multiple and noisy example demonstrations, see [14].

4 Learning Driving Styles and Maneuver Preferences

In addition to the practical considerations mentioned above, modern planning systems are often composed of a hierarchy of individual planners, with each planner refining the results of previous plans [8, 16]. In such a scenario, LEARCH must be applied to the lowest level planner, that actually makes the final decision about the next course of action. Application to higher level planners may be necessary as well, if they use different cost functions than the lowest level.

The formulation in the previous section referred to generic states in a planner's state space. Depending on the planner, these could be locations in the world, actions to be performed, or combined state-action pairs. A straightforward application of LEARCH to state action pairs would be possible, directly learning the coupled problem of balancing terrain and action preferences. However, such a formulation would be overly complex, in that it would learn a solution in the space of terrains and actions, rather than each one individually. Depending on the specific learner used, this could result in poor generalization; for example, preferring soft turns on flat terrain would not necessarily teach the system to prefer soft turns on hilly terrain.

Instead, this work proposes to decouple the problems by defining the cost of a state action pair as the decoupled cost of the state plus the cost of the action.

$$C(P) = \sum_{x \in P} C(x) = \sum_{x \in P} C_s(F_x^s) + C_a(F_x^a) \quad x \in S \times A \tag{5}$$

That is, two separate cost functions are proposed, one over locations the vehicle will traverse, and one over actions the vehicle will perform. The overall cost of a plan is the sum of the costs over individual locations and actions. Specific couplings between locations and actions (e.g. driving through a ditch or over a bump slowly may be fine, but is very costly at high speeds) are still possible, by adding features to express such couplings specifically in F^s or F^a (the state or action feature vectors).

If this cost formulation is plugged directly into (3), taking the partial derivatives with respect to C_s and C_a yields

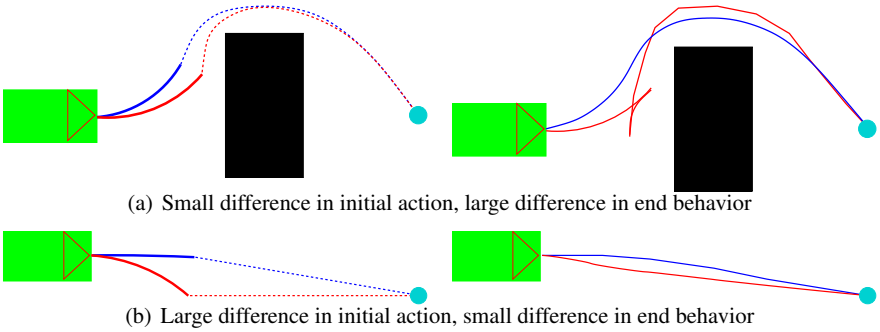
$$\begin{aligned}
\frac{\partial O}{\partial C_s} O[C_s, C_a] &= \lambda \nabla |C_s| + \sum_{x \in P_e} \delta_F^s(F_x^s) \; - \; \sum_{x \in P_*} \delta_F^s(F_x^s) \\
\frac{\partial O}{\partial C_a} O[C_s, C_a] &= \lambda \nabla |C_a| + \sum_{x \in P_e} \delta_F^a(F_x^a) \; - \; \sum_{x \in P_*} \delta_F^a(F_x^a) \\
P_* &= \arg\min_P \sum_{x \in P} (C_s(F_x^s) + C_a(F_x^a) - L_e(x))
\end{aligned} \tag{6}$$

These partials describe an interleaving optimization, where C_s and C_a are each updated one at a time. However, there is still a dependence between the two, as P_* is defined with respect to both. That is, for a current pair of cost functions, the functional gradient describes a set of actions to be desired or avoided (given the current terrain preferences) and a set of terrains to be desired or avoided (given the current action preferences). Applying LEARCH in this manner leads to the construction of two separate cost functions, both of which attempt to reproduce expert behavior in concert with the other.

The LEARCH formulation up to now has assumed that the example plan P_e is something the planning system could exactly achieve and match. However, in practice this is rarely the case. Due to finite resolution and sampling of state, and small differences in perceived and actual vehicle models (and implied kinematic or dynamic motion constraints), it is rarely the case that the finite set of plans the planner could produce will exactly contain P_e. One way to avoid this dilemma is to simply consider the set of all paths P in (1) to be all possible plans the planner could produce given a specific planning problem, as opposed to all possible plans that could exist. In this way, the basic MMP constraint ensures not that the example plan is the lowest cost plan, but simply that it is preferable to anything else the planner might produce. Unfortunately, in practice this provides no actual guarantee that the planner will produce the right behavior online. That is, depending on various resolutions and discretization, it is possible that during online application, plans will become available to the planner that are undesirable and lower cost. The result is that learning in this manner may require a much larger training set than would otherwise be needed.

An alternate solution to this problem is to in some way project the example plan onto the possible planner options. Depending on the structure of the problem, the loss function may provide a natural way to perform this projection. For instance, if actions are defined simply by a vehicle curvature and velocity, than choosing the planner action that is closest in these measures to the example action would seek to ensure similar behavior to the example, while allowing the MMP constraint to be achievable. This approach has been shown to provide beneficial results to generalization, as it allows learning to terminate when it is as close to achieving the example as resolution will allow [14].

A final challenge in learning cost functions over actions is the issue of receding horizon control. Rather than compute a dynamically feasible plan all the way from a



(a) Small difference in initial action, large difference in end behavior

(b) Large difference in initial action, small difference in end behavior

Fig. 2 Simple examples of planning problems where, due to receding horizon planning, error between immediate actions (left) and final behavior (right) do not correspond. Desired action/behavior is in blue, actual action/behavior is in red.

vehicle's current state to its goal, it is common for modern motion planners to only compute such a plan to a set horizon, using a heuristic or lower dimensional planner as a cost-to-go from the horizon to the goal. The result is that the planner will make a plan at a certain instant, with no expectation of running this plan to completion; rather a small section is executed before replanning. This creates a challenge for learning as it is now possible for very small errors in an immediate action to produce large errors in final behavior; this is especially problematic when the cost-to-go does not capture dynamic or kinematic constraints. It is also possible for large errors in an immediate action to end up being inconsequential, with the final behavior still quite similar. Figure 2 demonstrates both these cases for a simple planner that uses constant curvature arcs (see Section 5).

One way to identify when a small error in an immediate action choice will have a large final effect, or vice versa, is to forward simulate the repeated decision making of the planner to produce a simulated final behavior. This raises the question of whether this should actually be the current plan that is used for learning (e.g. P_*). Unfortunately, using this plan does not produce the proper derivative as in (6) as it potentially ignores states that, while not encountered in repeated simulation, would have been encountered in a single choice at a finite point in time. With the effect of such states on end behavior hidden from the learner, the cost function can not be properly modified. Therefore, the best way to ensure the correct end behavior is to ensure the correct choice is made at each instant in time ([13]).

However, while this simulation may not be useful for providing a partial derivative, it is useful for determining how important an error in an immediate action selection is. That is, if we define a penalty function $\mathscr{P}_e$ that defines error over end behavior (similar to how L_e defines error over an immediate action) then we can weight individual examples by how consequential they will be. The result is an optimization that, rather than bounding L_e, bounds $\mathscr{P}_e L_e$. In practice, this effect is achieved by weighting individual $\frac{\partial O}{\partial C_a}$ in (6) by $\mathscr{P}_e$

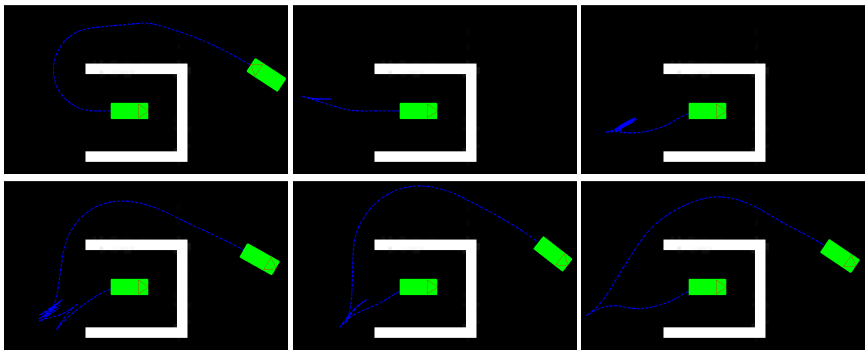


Fig. 3 Progression of planner behavior in simulation(from top left to bottom right) as a cost function is learned. Through successive iterations, the planner learns to exit the cul-de-sac to align itself so as to drive forward to the goal, instead of in reverse

$$\frac{\partial O}{\partial C_a} O[C_s, C_a] = \lambda \nabla |C_a| + \mathscr{P}_e[\sum_{x \in P_e} \delta_F^a(F_x^a) \; - \; \sum_{x \in P_*} \delta_F^a(F_x^a)] \tag{7}$$

This approach is formally known as slack re-scaling; for more information (including proof of bounds and correctness) see [13]. A nice side affect of this approach is that it becomes another source of robustness to noisy demonstration (since small differences in end behavior are lessened in importance).

5 Experimental Results

To test the application of LEARCH and its modifications to learning driving styles and maneuvers, a local-global planning system was created as in [8, 16], with a local planner choosing between constant curvature arcs, and a global planner operating without kinematic constraints. This planner architecture was chosen because it is simple, well understood, and in practice very effective. However, its effectiveness is dependent on proper tuning. Without proper tuning it can exhibit jittery or other undesirable behavior. In addition, since it only plans a single (kinematic) action at a time, it is incapable of performing more complex maneuvers (e.g. 3 point turns) without additional tuning. This tuning often takes the form of a state machine based on the relationship between the current vehicle pose and the current global plan, as well as a state history.

In addition to any cost accrued due to the locations a plan would traverse, the planner was implemented with costs based on specific actions (e.g. the final chosen motion arc). Costs were computed as a function of the features of an action, such as direction (forward or reverse), curvature, alignment with the heading of the global plan, and changes in direction or curvature from previous actions. These features provide enough information such that a state machine is not necessary to perform

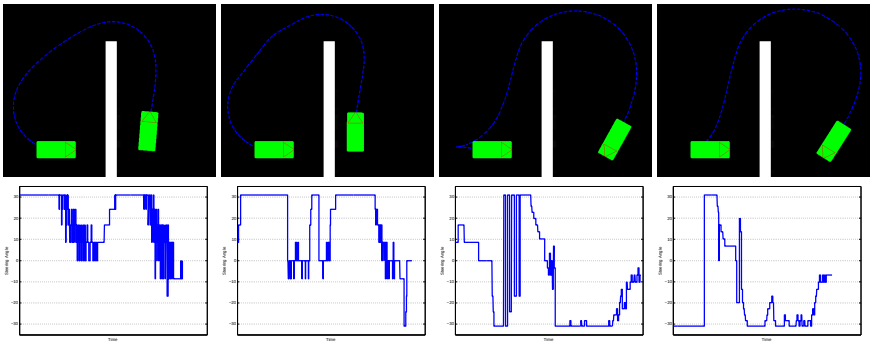


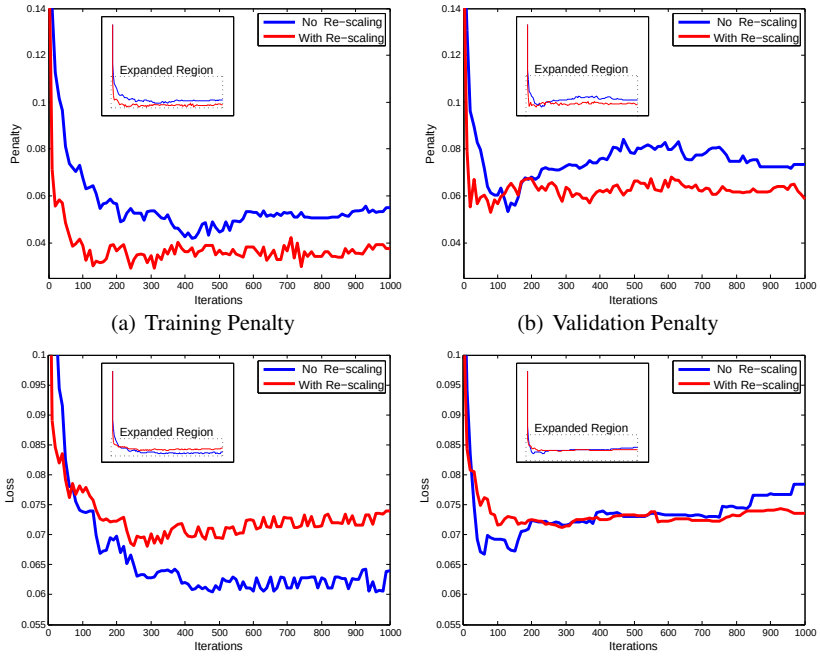
Fig. 4 The progression of learned preference models (from left to right) in a validation scenario. The commanded curvature at each point in time is also shown for each example. The planner first learns to avoid unnecessary turning, and then to favor driving forward and being aligned with the global path.

more complex maneuvers; however, achieving this capability requires careful tuning.

5.1 Simulated Experiments

Experiments were first performed in simulation in binary environments, to decouple any effects of terrain costing. A training set was created by manually driving the simulated robot through a simple set of obstacle configurations. A cost function was learned over constant curvature actions that resulted in the planner reproducing the expert's driving style (e.g. soft turns were preferred to hard turns when appropriate, changes in curvature were limited when possible, but obstacles were still avoided). In addition, the cost function resulted in the planner generating complex maneuvers by properly chaining together simple actions, despite never planning more than one action ahead at a time.

Figure 3 provides an example of learning a simple maneuver. Initially, the planner simply prefers the action that minimizes its cost-to-go (the global planner cost) and so tracks the global plan in reverse. Over successive iterations, a preference is developed for aligning the vehicle's heading with the global path, allowing the robot to perform a multi-point turn maneuver to turn in reverse and then drive forward. Eventually, a preference is also learned to avoid changing directions, allowing a single reverse turn and then a forward motion. Figure 4 provides an example of how the learned cost function affects driving style and control. In early iterations, the system is heavily underdamped and very jittery. Over time, it first learns to smooth out these oscillations, but then re-allows them in order to achieve other preferences (namely turning while in reverse). Finally, the system learns to balance both desires, performing a maneuver by chaining together multiple actions, while at the same time limiting unnecessary sudden turning. The single large oscillation seen in the



(a) Training Penalty (b) Validation Penalty

(c) Training Loss (d) Validation Loss

Fig. 5 Learning planner preference models with and without slack re-scaling (weighting examples by penalty)

final iteration was in response to almost hitting the wall, demonstrating that these preferences were properly balanced with those necessary to avoid obstacles.

Experiments were also performed to demonstrate the effect of slack re-scaling (penalty weighting) as described in Section 4. The training and validation performance during these experiments is shown in Figure 5. As expected, penalty weighting results in an optimization that lowers the training error between example and simulated end behavior, while increasing the error in immediate actions. The interesting result is in the validation performance, where the optimization without penalty weighting suffers from increased error and overfitting. This demonstrates the negative effects of trying too forcefully to correct small examples in actions (when they don't significantly effect end performance) and the advantage of penalty weighting.

A final simulated experiment compared the performance of learning to hand tuning. A human expert (different than the one who drove the vehicle for the training set) was tasked with manually constructing and tuning a cost function to achieve the same basic goals (a clean and reasonable driving style, while still producing complex maneuvers when necessary). During hand tuning, the expert could see the current performance of the planner in simulation. Every parameter configuration encountered during hand tuning was recorded for evaluation. The performance of both the learned and hand tuned cost function was evaluated over a large set of validation behaviors. The metrics for comparison are the average loss and average penalty; that

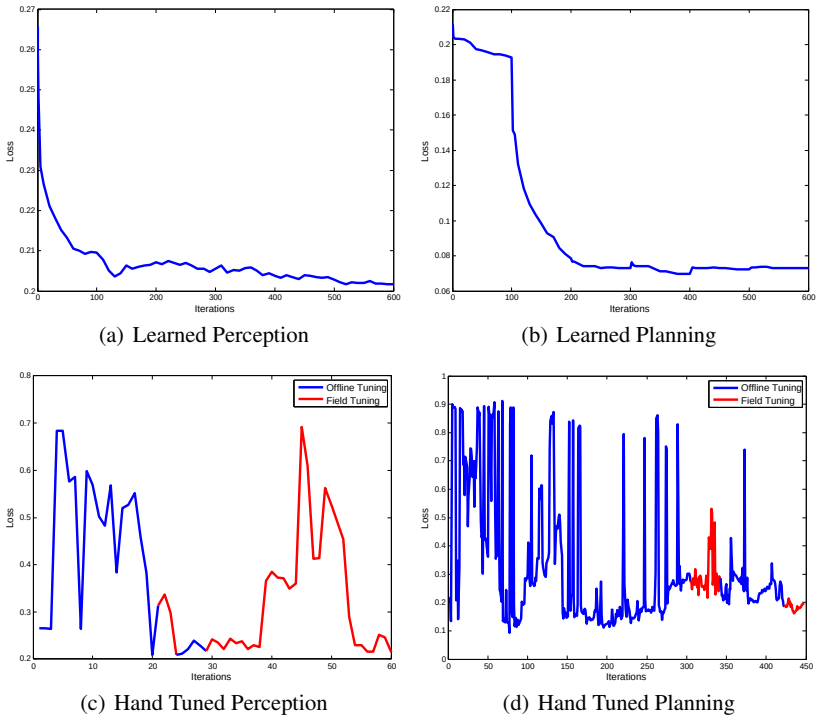
Fig. 6 The E-Gator Robotic Platform used in these experiments

is the average error between the example and planner immediate actions and end behavior. The hand tuned system had a final validation loss that was more than 25% higher than the learned system, and a final validation penalty that was 20% higher. Of note is that that the final configuration of the hand tuned system was not its best; the best configuration's performance was essentially equivalent to the learned system. This exemplifies one of the major issues with hand tuning: in general a human can only evaluate a small set of examples (relative to an automated system), and so may not fully understand the quality of a specific configuration during tuning. Also of note is the amount of expert interaction that was required in each case: approximately 12 hours to hand tune the system, versus less than 2 hours to demonstrate examples in simulation.

5.2 E-Gator Experiments

Next, the planning system was applied to an autonomous E-Gator vehicle (Figure 6) for operation in outdoor, unstructured terrain. The E-Gator was equipped with a single actuated LiDAR scanner for producing 3D point clouds. Perception software described in [5] was ported to this system, allowing the computation of a supporting ground surface through vegetation, as well as geometric features relating to size and shape of potential positive obstacles. These features were used as input to a terrain cost function. The vehicle's preferred behavior was therefore determined by two cost functions: one mapping sensor data to costs over terrain (perception), and another mapping actions to costs (planning). Live expert demonstration (i.e. driving the vehicle) was then used to learn these two cost functions.

For comparison, yet another robotics expert was tasked with manually creating and tuning perception and planning cost functions to achieve the same navigation tasks for the E-Gator. The expert could test parameter configurations in offline simulation/playback, or on the live vehicle. Every tested parameter configuration was recorded for comparison. The performance of both sets of cost functions was then evaluated over a large set of validation behaviors. These results are shown in Figure 7. The metric for comparison across all experiments was validation loss.



(a) Learned Perception (b) Learned Planning

(c) Hand Tuned Perception (d) Hand Tuned Planning

Fig. 7 Validation Loss for the Learned and Hand Tuned autonomy systems.

These experiments demonstrate how the system performance changes for both the learned and hand tuned systems during tuning/learning. The learned system undergoes a relatively stable optimization procedure, quickly converging to a good solution and then iteratively refining it. In contrast, during hand tuning the expert continually changed parameters with large (and often negative) effects on system performance. As seen in [14], the expert was able to tune a perception system that was close to learned quality (within 6%). However, in contrast to simulation (with a binary world), the expert performed significantly poorer when tuning the planning system. The expert's final configuration resulted in more than 2.5 times the error in action selection, and even the expert's best configuration (unknown during tuning) had 25% more error than the learned system. This demonstrates the inherent challenge not only in tuning perception and planning cost functions to solve independent problems, but to properly couple them to produce good autonomous performance.

The difference in performance was further explored through a serious of live experiments on the actual autonomous platform. The two final system configurations (learned or hand tuned perception/planning cost functions) were tested head to head over a large set of test courses, totaling more than 4 km of autonomous driving. Several statistics were measured and compared over these runs. Table 1 shows these statistics, and the statistical significance of any differences. Of note is that the hand tuned system spent much more time in reverse (dangerous for a vehicle with sen-

	Avg Dist (m)	Extra Dist (m)	Avg Roll (○)	Avg Pitch(○)	% Time in Reverse
Learned (μ/σ^2)	69.27/785.5	12.02/148.5	3.71/2.61	4.45/2.29	0.075/0.012
Hand-Tuned (μ/σ^2)	73.09/1791.7	17.15/1293.24	4.24/3.90	4.39/2.15	0.19/0.079
P-value	0.34/**0.014**	0.23/**0.0**	0.13/0.14	0.56/0.57	**0.016/0.0**

	Dir Switch Per m	Avg Steer Angle (○)	Avg Δ Angle (○)	% Time Angle $\neq 0$	Avg Δ Angle $\neq 0$ (○)	Safety E-Stops
Learned (μ/σ^2)	0.029/0.0014	9.43/15.46	1.69/0.20	0.33/0.0064	5.19/0.49	0.13/0.12
Hand-Tuned (μ/σ^2)	0.044/0.0079	11.74/14.43	2.08/0.76	0.19/0.0033	10.76/3.60	0.90/1.49
P-value	0.18/**0.0**	**0.011**/0.57	**0.016/0.0**	**0.0/0.038**	**0.0/0.0**	**0.0/0.0**

Table 1 Results of 4 km of autonomous E-Gator experiments comparing learned to hand-tuned performance.All statistics are on a per test average.

sors in front) than the learned system. It would also occasionally oscillate between forward and reverse driving very rapidly. The hand tuned system exhibited a higher preference for driving perfectly straight; when it did turn, it generally turned much faster (and changed curvature faster and more often) than the learned system.

These differences manifest themselves in the safety of each configuration, as the hand tuned system required nearly 7 times as many operator interventions (e.g. to prevent hitting a dangerous obstacle). One of the most common causes of these interventions was the hand tuned system trying to turn too hard around obstacles. The obstacles were clearly seen by the perception system; however the coupled tuning of perception and planning cost functions was such that the vehicle tended to clip them. It is interesting to note that this behavior was clearly observed during hand tuning, and much of the final tuning was an unsuccessful attempt to combat this problem. This demonstrates the difficulty in manually mapping desired behavior changes into appropriate parameter changes.

A final point of note is the time required to produce each system. The hand tuned system required 38 hours of expert tuning time, and an additional 18 hours from a safety operator while tuning on the live vehicle. In contrast, the learned system required less than 4 hours of combined expert and operator time to collect the necessary training data. So in addition to achieving safer and more stable autonomous performance, the learning approach required only 1/14 as much human interaction. For additional data and details of these experiments, see [13].

6 Conclusion

This work addressed the issue of producing properly coupled cost functions for an autonomous navigation system, to balance preferences describing both where and how a vehicle should drive. The LEARCH algorithm of the MMP framework [12]

was extended to handle this coupled problem by learning cost functions separately while still considering their affect on eachother. Issues with finite planner resolution and receding horizon control were addressed through modifications to the LEARCH algorithm, with experimental results validating these new approaches.

Previous work [14] has shown that human experts can hand tune a perception cost function that is almost equivalent in performance to one that is learned, with the primary advantage of learning being a major reduction in the amount of expert interaction required. This same result was duplicated when tuning just a planner cost function: the human expert was able to do almost as well as the learned system, but in a far less efficient manner. A closer analysis of the hand tuning process shows that human experts do not necessarily know when they have achieved their best performance, whereas a learned system has the advantage of automatic validation. In addition, it was shown that by learning the proper parameters from demonstration, a very simple planning system could be configured to result in more complex driving maneuvers.

When a human expert was tasked with tuning both a perception and planning system at the same time, this additional coupling proved a major detriment to tuning performance. As opposed to simply taking longer, the hand tuned approach also produced a system that was significantly lower quality. This decrease in quality was demonstrated in both offline (simulated) validation, and in online autonomous validation. In the latter case, the poorer quality of the cost functions manifested itself as an increase in the number of safety interventions required during autonomy, as well as a more jittery driving style. These results further demonstrate the effectiveness of the learning from demonstration approach, both for reducing the amount of time required to tune an autonomous system, and for improving the quality and robustness of the result.

Future work is currently focused on application to a variety of autonomous vehicles with a range of planning systems. In addition, the challenge of collecting and curating large amounts of training data must be addressed. While these approaches have drastically reduced the amount of time necessary to properly tune an autonomous system, further improvement is possible. This includes approaches for identifying the most useful training data [15], as well as alternate approaches to soliciting expert input and feedback [13].

Acknowledgements This work was sponsored by the U.S. Army Research Laboratory under contract "Robotics Collaborative Technology Alliance" (contract DAAD19-01-2-0012) The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

1. Abbeel, P., Coates, A., Ng, A.Y.: Autonomous helicopter aerobatics through apprenticeship learning. International Journal of Robotics Research **29**(13), 1608–1639 (2010)

2. Abbeel, P., Dolgov, D., Ng, A.Y., Thrun, S.: Apprenticeship learning for motion planning with application to parking lot navigation. In: IROS (2008)
3. Argall, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems (2008)
4. Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Reinholtz, C., Hong, D., Wicks, A., Alberi, T., Anderson, D., Cacciola, S., Currier, P., Dalton, A., Farmer, J., Hurdus, J., Kimmel, S., King, P., Taylor, A., Covern, D.V., Webster, M.: Odin: Team victortango's entry in the darpa urban challenge. Journal of Field Robotics **25**(8), 467–492 (2008)
5. Bagnell, J.A., Bradley, D., Silver, D., Sofman, B., Stentz, A.: Learning for autonomous navigation: Advances in machine learning for rough terrain mobility. IEEE Robotics & Automation Magazine **17**(2), 74–84 (2010)
6. Hamner, B., Singh, S., Scherer, S.: Learning obstacle avoidance parameters from operator behavior. Journal of Field Robotics **23**(1), 1037–1058 (2006)
7. Kalakrishnan, M., Buchli, J., Pastor, P., Schaal, S.: Learning locomotion over rough terrain using terrain templates. In: IROS, pp. 167–172 (2009)
8. Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., Happold, M., Herman, H., Mandelbaum, R., Pilarski, T., Rander, P., Thayer, S., Vallidis, N., Warner, R.: Toward reliable off road autonomous vehicles operating in challenging environments. International Journal of Robotics Research **25**(5-6), 449–483 (2006)
9. Kolter, J.Z., Abbeel, P., Ng, A.Y.: Hierarchical apprenticeship learning with application to quadruped locomotion. In: Neural Information Processing Systems (2008)
10. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent. In: Advances in Neural Information Processing Systems 12. MIT Press, Cambridge, MA (2000)
11. Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., Thrun, S.: Junior: The stanford entry in the urban challenge. Journal of Field Robotics **25**(9), 569–597 (2208)
12. Ratliff, N., Silver, D., Bagnell, J.A.: Learning to search: Functional gradient techniques for imitation learning. Autonomous Robots **27**, 25–53 (2009)
13. Silver, D.: Learning preference models for autonomous mobile robots in complex domains. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA (2010)
14. Silver, D., Bagnell, J.A., Stentz, A.: Learning from demonstration for autonomous navigation in complex unstructured terrain. International Journal of Robotics Research **29**(12), 1565–1592 (2010)
15. Silver, D., Bagnell, J.A., Stentz, A.: Active learning from demonstration for robust autonomous navigation. In: International Conference on Robotics and Automation (2012)
16. Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., Schwehr, K.: Recent progress in local and global traversability for planetary rovers. ICRA (2000)
17. Stelzer, A., Hirschmuller, H., Gorner, M.: Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain. International Journal of Robotics Research **31**(4), 381–402 (2012)
18. Sun, J., Mehta, T., Wooden, D., Powers, M., Rehg, J., Balch, T., Egerstedt, M.: Learning from examples in unstructured, outdoor environments. Journal of Field Robotics **23** (2007)
19. Urmson, C., Anhalt, J., Bagnell, J.A., Baker, C., Bittner, R., Clark, M.N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T.M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.W., Singh, S., Snider, J., Stentz, A., Whittaker, W.L., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., Ferguson, D.: Autonomous driving in urban environments: Boss and the urban challenge. Journal of Field Robotics **25**(8), 425–466 (2008)
20. Zucker, M., Ratliff, N., Stolle, M., Chestnutt, J., Bagnell, J.A., Atkeson, C.G., Kuffner, J.: Optimization and learning for rough terrain legged locomotion. International Journal of Robotics Research **30**, 175–191 (2011)