Applied Imitation Learning for Autonomous Navigation in Complex Natural Terrain

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

Abstract Rough terrain autonomous navigation continues to pose a challenge to the robotics community. Robust navigation by a mobile robot depends not only on the individual performance of perception and planning systems, but on how well these systems are coupled. When traversing rough terrain, this coupling (in the form of a cost function) has a large impact on robot performance, necessitating a robust design. This paper explores the application of *Imitation Learning* to this task for the Crusher autonomous navigation platform. Using expert examples of proper navigation behavior, mappings from both online and offline perceptual data to planning costs are learned. Challenges in adapting existing techniques to complex online planning systems are addressed, along with additional practical considerations. The benefits to autonomous performance of this approach are examined, as well as the decrease in necessary designer interaction. Experimental results are presented from autonomous traverses through complex natural terrains.

1 Introduction

The capability of autonomous robotic systems to successfully navigate through unstructured environments continues to advance. Ever improving high resolution sensors and perception algorithms allow a mobile robot to build a detailed model of its environment, and advances in planning systems allow for the generation of ever more complex routes and trajectories towards achieving a navigation goal. However, as perception and planning systems become more complex, so does the task of coupling these systems. This coupling often takes the form of a *Cost Function*. Using data from the perception system as input, the cost function maps to a scalar cost value, defined over the state space of the planning system (Figure 1). These costs are then used as the optimization metric by the planning system when determining the next action or sequence of actions.

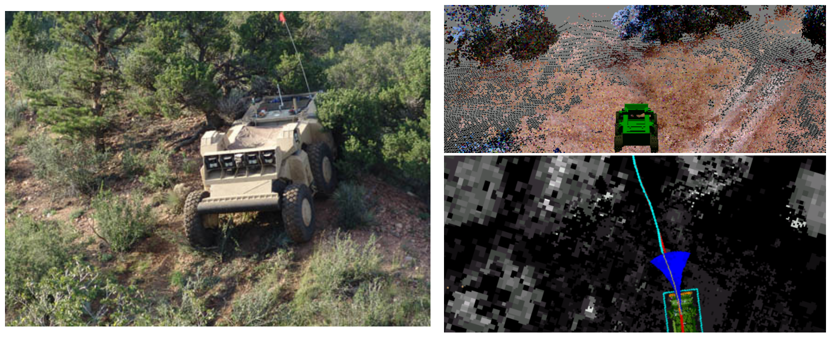


Fig. 1 Crusher (Left) is capable of autonomous navigation through complex outdoor terrain. Perceptual data (Top Right) are converted to costs (Bottom Right) for use by the planning system.

In simple or structured environments, cost functions are often easily defined; for instance, in an indoor environment the cost of traversable freespace should be very low, and the cost of walls or other obstacles should be high. However, in rough or unstructured terrain, it is less intuitive how to define cost. A small obstacle should clearly have larger cost than flat ground, and smaller cost than a large obstacle. Explicitly defining these tradeoffs encodes the desired behavior of the robot and is quite challenging; defining a generalizeable function that maps from perceptual inputs to the proper cost is even more so.

This first step of defining the relative cost of various terrains requires a concrete definition of what metric a robot's performance will be measured against. Common metrics include maximizing safety or probability of success, minimizing distance traveled or time taken, minimizing net energy loss, minimizing observability or maximizing sensor coverage. Often, the actual desired robot behavior optimizes a combination of such metrics; for example, it may be desirable for a robot to approximately maximize safety but take certain risks to minimize distance traveled.

Previous work has focused on several differing approaches. Attempts to explicitly approximate traversability through simulation [3] or proprioception [5] limit the choice of metrics to maximizing safety; they also require a robot model capable of directly computing probabilities of mobility failure. Approaches focused on explicitly combining multiple metrics are limited to optimizing one metric subject to constraints on others [13, 15]. The most common general solution is to manually design and hand tune a cost function until the robot achieves the desired behavior. This can be an incredibly tedious process (as it requires a manual optimization in a potentially high dimensional space) and often results in systems that suffer from poor generalization and a lack of robustness to novel scenarios.

This paper explores the application of imitation learning to this challenge, specifically the LEARCH [10] algorithm described in the next section. The application of this approach to the Crusher autonomous navigation platform [14] (Figure 1) is reviewed, along with a discussion of practical considerations when applying this approach.

```
$C_0 = Prior$;
for $i = 1...T$ do
    foreach $P_e$ do
        foreach $x \in$ getBoundingBox($P_e$) do
            $F[x]$ = getPerceptionFeatures($x$);
            $M[x] = C_{i-1}(F[x]) + L_e(x)$;
        $P_* =$ planPath($s_e, g_e, M$);
        $P_e^* =$ replanExamplePath($P_e, \beta, M$);
        $\{U_+^{P_e}, U_-^{P_e}\}$ = computeVisitationCounts($P_e^*, P_*$);
    $R_i =$ trainBalancedRegressor($F, U_+, U_-$);
    $C_i = C_{i-1} * e^{\eta_i R_i}$;
```

Fig. 2 The LEARCH algorithm

2 Imitation Learning

Although explicitly defining the relative tradeoffs between different actions is a difficult task for a domain expert, indicating or demonstrating examples of correct behavior is often easier (otherwise, the task itself is not well defined). Therefore, the imitation learning framework seeks to learn the correct robot behavior from observation of expert behavior. Many previous applications of this framework to mobile robots [6, 7] sought to learn to predict what action to perform, based on the action performed by an expert at a certain state. In this way, action prediction essentially replaces the lower level motion planning operation on a mobile robot. However, this approach is inherently myopic and does not scale to longer range planning, as it requires all necessary information to be encoded in the current robot state.

Therefore, rather than learn a mapping from features of a state to actions, we seek to learn a mapping from features of a state to costs, such that the planning system will produce the correct behavior when provided with said costs. This approach has its roots in the concept of *Inverse Optimal Control*, and has recently been developed for use in robotic systems [1, 8]. By learning a cost function to reproduce expert behavior, the need for explictly defining a metric or weighting between metrics is eliminated; the new metric is matching human performance, and it is left up to the human expert to define (through behavior) how to balance various options and considerations.

It is important to note that this approach learns the correct cost function for a specific planner or system of planners, and maps to cost from a specific perception system. The purpose is not to try and improve the separate performance of these systems; rather, it is to optimize the coupling of these modules to provide the best overall system performance. As the Crusher system operates with costs defined over a 2D grid, subsequent descriptions will deal specifically with this setting. Without loss of generality, it is easiest for now to consider the planning system as a basic A* planner, and a planned path as a sequence of 2D grid cells. Adaptation to more complex planning systems is covered in the next section.

Our imitation learning approach is based on the LEARCH algorithm, (Figure 2); for a full derivation, see [10]. The input to the algorithm is a set of example paths, each a sequence of 2D locations leading from a start s to a goal g and representing the correct path (according to the expert). LEARCH seeks to find a cost function C such that each example path P_e is the planner output under the cost function. The LEARCH inner loop iterates through each example. For each example path P_e with start and goal s_e and g_e a path P_* is planned under the current candidate cost function C_i (C_0 can be initialized to any prior). Since P_* is the output of an optimal planner, $C_i(P_*) \leq C_i(P_e)$. Since we desire a C such that $C(P_e) = C(P_*)$, we seek to minimize the difference in cost $C(P_e) - C(P_*)$. As the cost of a path is simply the sum of costs at states along it, P_e and P_* provide a list of states where the cost could be changed to lower this cost difference: states in P_e could have their cost lowered and states in P_* could have their cost raised (states in both simply cancel).

This list of candidate states (called the visitation counts) provides a local gradient in the space of cost functions. However, the cost at each state can not simply be raised or lowered, as the goal is not to identify the correct cost for each cell, but rather a function that maps perceptual features to an appropriate cost. If a function ΔC_i could be found that approximated this gradient (the output is positive or negative when provided with the appropriate features), adding it to C_i would lower the cost difference.

Finding a general function to match a list of input/output pairs can be solved through regression analysis. In this case, the inputs are well defined (perceptual features), but the outputs are not; for each input, the required cost delta is not known, just its sign. Therefore, outputs targets are specified as ± 1 depending on whether the costs need to be raised or lowered. In this way, the regressor R generalizes the local cost changes over the entire feature space. Each regression target can also be weighted to indicate that certain cost changes are more important relative to others. Determining these relative weights is discussed in Section 3.

The final learning procedure is summarized as follows: for each example path and the corresponding plan (under C_i), compute the set of visitation counts and the corresponding perceptual features. Next train a regressor R over these input/output pairs, and combine it with C_i, weighted by a learning rate parameter η. The cost of cells along an example path (which an expert specifically chose to encounter) will generally be lowered, while the cost of cells along a temporarily cheaper path (which the expert chose to avoid) will generally be increased. This loop is then iterated until convergence. Figure 3 provides a visual example of this procedure in action.

A few details remain. Rather than summation, we use an update rule of $C_{i+1}(f) = C_i(f)e^{\eta R_i(f)}$, resulting in exponentiated functional gradient descent [10] which makes better use of available dynamic range, as well as naturally enforcing a positivity constraint on costs. The choice of regressor (e.g. linear, neural net, etc.) is also unspecified. This decision helps define the balance between descriptiveness and generalization in the space of possible cost functions, and is discussed further in Section 3. Finally, by augmenting costs with a margin (determined by a loss function $L_e(x)$), trivial cost solutions can be eliminated and generalization improved.

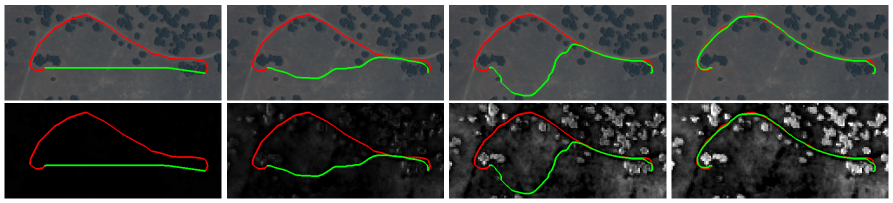


Fig. 3 An example of the LEARCH algorithm learning to interpret satellite imagery (Top) as costs (Bottom). As the cost function evolves (left to right), the current plan (green) recreates more and more of the example plan (red). Quickbird imagery courtesy of Digital Globe, Inc.

In addition to the algorithm as described, there are a few modifications that can increase robustness to noisy or imperfect expert demonstration [11, 12]. Human experts rarely demonstrate exact optimal behaviors, especially in large areas of similar terrain. By performing a balanced regression (normalizing relative weights such that positive and negative targets have equal total weight), the regressor can be forced to more strictly separate between terrains when changing cost. Another addition to the algorithm is to continually replan the example path. Human demonstration often contains a degree of high frequency noise; a smoothing operation is therefore beneficial. This smoothing can be performed by selecting a new example that is entirely contained within a corridor of width β around the original example. The new example is created by planning the optimal path using the current cost function within the corridor (and infinite cost elsewhere). This has the effect of adapting the example to the current cost hypothesis at a small scale (implicitly defined by β), while still adapting the hypothesis to the example at a large scale.

3 Application to Autonomous Navigation

Our imitation learning approach was applied to the task of interpreting perceptual data for the purpose of motion planning on the Crusher system. Crusher is provided with two main forms of perceptual data: static sources of prior data (overhead imagery and LiDAR), and dynamic sources of data collected in real time (onboard cameras and LiDAR). Once these data sources have been converted to 2D cost grids and fused together (at the cost level), Crusher's motion planning system is responsible for choosing vehicle actions. The motion planning system is similar to [4], and combines a global planner based on Field D* [2] and a local planner based on forward simulation of potential vehicle actions (specifically constant curvature commands) for a fixed horizon.

For the Crusher platform, imitation learning was first applied to the task of interpreting overhead data to create prior cost maps [11]. 2D feature maps are extracted from the input data sources, and then converted from maps of features to a map of costs. These maps are then used for global route planning offline, as well as online global planning when fused with current perceptual data. This context provides an ideal setting for the application of the LEARCH algorithm due to the static nature of the perceptual data, and the ease of collecting training examples: each example is simply a path that can be 'drawn' by an expert on top of imagery or other visualization of the underlying data.

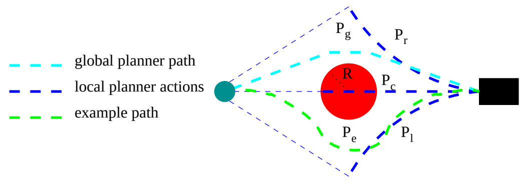
Since overhead costs are only used for planning in regions that have not yet been directly observed by the robot, overhead costs are learned with respect to the Field D* global planner. As Field D* plans an interpolated path over a 2D grid, computing visitation counts is not as straightforward as simply marking which states each path traverses through. Instead, the distance traveled through each grid cell must be recorded. This results in visitation counts that are real valued instead of binary. During regression the output target for a real valued visitation count is still ± 1, but the target is now weighted relative to the visitation count. If a path passes through cell x_1 for twice as long as x_2, then x_1 has twice the impact on the cost of a path; moving the cost in the right direction is therefore twice as important.

Like many motion planning algorithms, Field D* also makes use of a configuration space expansion to account for the dimensions of the vehicle. A configuration space expansion also results in non-binary visitation counts; it is taken into account by incrementing the count of all states x_j relative to their contribution to the cost of state x_i when x_i is on a path. Crusher's planning system performs an expansion by averaging costs over a circular window. Therefore, for a path traveling distance d through cell x_i, all cells x_j within the expansion window have their counts incremented by d. More complex expansions can be accounted for in the same manner.

Imitation learning was next used to learn costs from features generated by Crusher's onboard perception system. Crusher's perception software processes raw sensor data into feature descriptions of voxels in real time; each column of voxels is then converted into a 2D cost. Therefore, unlike learning from overhead data, features are not static. While additional adaption of the LEARCH algorithm is required in order to deal with the dynamic and unknown nature of real time perceptual data, the approach remains conceptually similar and will be treated as such moving forward; for details see [12].

Instead of drawing a path on top of a visualization, collecting expert examples to train the perception system consists of the expert manually driving Crusher through an example behavior. Along with the path traversed, all raw sensor data is logged during this collection. Sensor data is then post-processed via playback through Crusher's perception software to generate the final features that will be converted into costs. By logging the raw sensor data, perception software does not need to remain static after training data collection. Whenever changes or improvements are made to perception software, features can simply be regenerated, and a cost function relearned. Therefore, training data in this form does not need to be recollected every time the system changes; the cost function is simply retrained offline.

Since costs from online perceptual data determine Crusher's actual motion commands, costs must be learned with respect to the local planning system. Figure 4(a) provides a simplified example to demonstrate why this is so. If costs were trained with respect to the global planner, LEARCH would be satisfied with the cost on the obstacle O when it is sufficiently high to make up for the extra distance $|P_g| - |P_c|$. However, since $|P_l|, |P_r| > |P_c| + O > |P_g|$, P_c remains the cheapest local planner op-



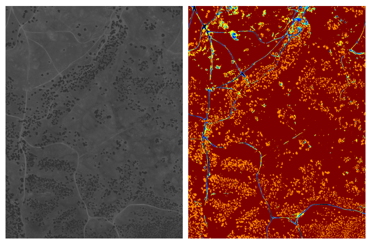


Fig. 4 **(a)** A simplified scenario where the local planner has only 3 possible actions. **(b)** Example of a new feature (right) learned from panchromatic imagery (left).

tion in this case. The result would be that the local planner would still choose to drive over the obstacle. This result is observed empirically in Section 4.

Unfortunately, there are also problems with training directly for the local planner. As the local planner only considers a discrete, kinematically feasible set of actions, it is often the case that no series of actions will sufficiently match the expert example. In this case, the LEARCH termination condition is undefined. Terminating when the example path is lower cost than the planned path will not suffice; in Figure 4(a) this could result in $C(P_e) < C(P_c) < C(P_l), C(P_r)$ (the colliding action would still be preferred). Running until the cost function converges is therefore necessary, but has its own side affects. Once $C(P_c) > C(P_l), C(P_r)$, LEARCH will start to try and raise the cost along P_l or P_r. If the chosen regressor can differentiate between the terrain under P_l or P_r and that under P_e, it will raise those costs without proper cause. The end result is a potential addition of noise to the final costs, and lower generalization. The degree of this noise depends on the resolution of the planner and the regressor.

If there were some way to know the 'right' planner action, then the selection of that action could serve as a termination condition. Collecting this information during demonstration by an expert would be extremely tedious, requiring an expert selection at every planning cycle. Instead, we propose the use of a heuristic approach to approximate this decision. Essentially, we seek to 'project' the expert's example behavior onto the space of possible planner actions. This is performed by first learning a perception cost function for the global planner. As described above, such a cost function will generally underestimate the cost necessary for the local planner. Therefore, we score each local planner action by its *average* cost instead of total cost[1]. An action with low average cost can not be said to be optimal, but it at least traverses desirable(low cost) terrain. An additional distance penalty is added to bias action scores towards those that make progress towards the goal[2]. After scoring each action, that with the lowest score is used as the new example. The result of this initial replanning step is to produce a new example behavior that is feasible to the local planner.

[1] The global planner section of each action is still computed with respect to total cost

[2] the weight of this penalty can be automatically tuned by optimizing performance on a validation set, without any hand tuning

When dealing with static overhead data a cost function, once learned, will generally only be applied once through a data set. In contrast, a perception cost function will be continually applied in real time on a robot when operating autonomously. Therefore, it is important that the cost function be computationally inexpensive. As combining multiple linear functions yields a single linear function, linear regressors have a significant computational advantage over nonlinear regressors (which would require a separate evaluation per regressor). Unfortunately, they also suffer from limited expressiveness. This can be dealt with by adding a feature learning phase, as described in [9, 11]. Such a phase automatically decides to occasionally learn simple non-linear combinations of the original input features that help differentiate terrains that are difficult for a linear cost function to discriminate. This approach is similar to the way in which cost functions are often hand-engineered: simple linear functions handle the general cases, with sets of rules to handle difficult special cases. Additionally, such new features can be used as a guide in the development of further engineered features. Figure 4(b) provides an example in the overhead context, demonstrating a new feature derived from only panchromatic satellite imagery. This new feature strongly disambiguates roads and trails from surrounding terrain, and could be taken as an indication that an explicit road extractor would be useful.

4 Field Results and Conclusions

The described imitation learning approach was implemented to learn mappings from both overhead and onboard perceptual data to cost. As described in [10], this approach is guaranteed to converge when using a properly chosen learning rate [3]. In practice, a decaying learning rate of the form $\eta/\sqrt{n}$ is used at the n^{th} iteration. The parameter η affects only the rate of convergence; a well chosen η usually results in convergence after approximately 50 - 100 iterations. The computation required at each iteration for each example is dominated by the cost of applying the current cost function to local feature maps, and then planning through the resulting cost map. For the training sets used in this work, computation per iteration was approximately 5 minutes on a 2.4 Ghz processor[4].

The Crusher autonomy system originally made use of hand-tuned cost functions for converting overhead and perception features to costs. Comparing autonomous performance when using different cost functions can quantify the differences in performance brought about by these different approaches. Engineered prior cost maps were used for the first 4 Crusher field experiments. This process consisted of first performing a supervised classification of the raw feature maps, and then converting the classifier outputs into costs. This lossy compression of the feature space was performed to make designing a cost function easier and more intuitive. As different test sites provided differing data sources and resolutions, this process was repeated

[3] Use of a smoothing corridor removes the theoretical guarantee; however in practice this has not proven to affect convergence

[4] If faster learning is required, LEARCH can be parallelized by example at each iteration

for each test site. Additionally, the desire to create cost maps from different subsets and resolutions of prior data (in order to perform resolution comparisons), meant multiple cost functions were necessary. For each site, labeling training data and determining parameters for multiple cost functions would involve on average more than a day of a domain expert's time. Learned prior cost maps were then used for the remaining 6 field experiments. For each site, producing a series of example paths would take on average only 1-2 hours of an expert's time. These examples could then be used to train multiple cost maps using different data sources and resolutions.

In a timed experiment on a 2 km^2 test site, producing a supervised classification required 40 minutes of expert involvement, and tuning a cost function required an additional 20 minutes. In contrast, producing example paths required only 12 minutes. As Crusher has been tested on sites ranging up to 200 km^2, this time savings is magnified in importance. The final cost maps were also evaluated by comparing planned routes to an independent validation set of examples. The engineered map produced routes that matched 44% of states along validation paths on average. Using imitation learning to learn just the correct weights for the supervised classification produced a map that scored 48%. Imitation learning from the raw features scored 57%. This result demonstrates that the automated approach performs superior parameter tuning, and makes better use of all the available raw data. It has also been shown that Crusher navigates more efficiently when using learned prior maps online as opposed to engineered maps, driving safer routes at faster speeds [11].

During the more than 3 years of the Crusher program, an engineered perception cost function was continually redesigned and retuned, culminating in a high performance system [14]. However, this performance came at a high cost. Version control logs indicate that 145 changes were made to just the form of the cost function; additionally more than 300 parameter changes were checked in. As each committed change requires significant time to design, implement, and validate, easily hundreds of hours were spent on engineering the cost function. In contrast, the final training set used to learn a cost function consisted of examples collected in only a few hours.

The performance of different perception cost functions was compared through over 150 km of comparison trials. The final results comparing 4 different cost functions are presented in Table 1. In comparison to the engineered system, a cost function learned for the global planner resulted in overly aggressive performance. As discussed in Section 3, learning in this manner does not result in sufficiently high costs; the result is that Crusher drives faster and turns less while appearing to suffer from increased mobility risk. In contrast, the costs learned for the local planner performed very similarly to the high performance of the engineered system. Additionally, adding an initial replanning step further improved performance; by reducing cost noise, average speed increased, with a decrease in turns and direction switches, and no increase in mobility risk.

In conclusion, this work has demonstrated the applicability of imitation learning towards improving the robustness of autonomous navigation systems, while helping to minimize the necessary amount of expert interaction. Specifically, the parameter tuning problem that often results from the coupling of complex perception and planning systems can be automated through expert demonstration instead of expert

Table 1 Averages over 295 different waypoint to waypoint trials per perception system, totaling over 150km of traverse. Statistically significant differences (from Engineered) denoted by *

System	Avg. Distance Made Good (m)	Avg. Cmd. Vel. (m/s)	Avg. Cmd. Ang. Vel.($\circ/s$)	Avg. Lat. Vel. (m/s)	Dir Switch Per m	Avg. Motor Current (A)	Avg. Roll($\circ$)	Avg. Pitch($\circ$)	Avg Vert. Accel (m/s^2)	Avg Lat. Accel (m/s^2)	Susp. MaxΔ (m)	Safety E-stops
Engineered	130.7	3.24	6.56	0.181	0.107	7.53	4.06	2.21	0.696	0.997	0.239	0.027
Global	123.8*	3.34*	4.96*	0.170*	0.081*	7.11*	4.02	2.22	0.710*	0.966*	0.237	0.054*
Local	127.3	3.28	5.93*	0.172*	0.100	7.35	4.06	2.22	0.699	0.969*	0.237	0.034
Local w/replan	124.3*	3.39*	5.08*	0.170*	0.082*	7.02*	3.90*	2.18	0.706*	0.966*	0.234*	0.030

intervention. In the future, we wish to expand this approach to also automate the selection of parameters internal to a planning system, further reducing the need for human tuning.

This work was sponsored by DARPA under contract "Unmanned Ground Combat Vehicle - PerceptOR Integration" (contract MDA972-01-9-0005) and 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 authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

1. Abbeel, P., Ng, A.: Apprenticeship learning via inverse reinforcement learning. In: International Conference on Machine Learning (2004)
2. Ferguson, D., Stentz, A.: Using interpolation to improve path planning: The field d* algorithm. Journal of Field Robotics **23**(2), 79–101 (2006)
3. Green, A., Rye, D.: Sensible planning for vehicles operating over difficult unstructured terrains. IEEE Aerospace Conf. (2007)
4. 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)
5. Kim, D., Sun, J., Oh, S.M., Rehg, J.M., Bobick, A.F.: Traversability classification using unsupervised on-line visual learning. In: IEEE International Conference on Robotics and Automation (2006)
6. LeCun, Y., Muller, U., Ben, J., Cosatto, E., Flepp, B.: Off-road obstacle avoidance through end-to-end learning. In: Advances in Neural Information Processing Systems 18 (2006)
7. Pomerleau, D.: Alvinn: an autonomous land vehicle in a neural network. Advances in neural information processing systems 1 pp. 305 – 313 (1989)
8. Ratliff, N., Bagnell, J., Zinkevich, M.: Maximum margin planning. In: International Conference on Machine Learning (2006)
9. Ratliff, N., Bradley, D., Bagnell, J., Chestnutt, J.: Boosting structured prediction for imitation learning. In: Advances in Neural Information Processing Systems 19. MIT Press (2007)
10. Ratliff, N.D., Bagnell, J.A., Silver, D.: Learning to search: Functional gradient techniques for imitation learning. Autonomous Robots (2009)
11. Silver, D., Bagnell, J.A., Stentz, A.: High performance outdoor navigation from overhead data using imitation learning. In: Proceedings of Robotics Science and Systems (2008)
12. Silver, D., Bagnell, J.A., Stentz, A.: Perceptual interpretation for autonomous navigation through dynamic imitation learning. In: ISRR (2009)
13. Stentz, A.: CD*: a real-time resolution optimal re-planner for globally constrained problems. In: Proceedings of AAAI National Conference on Artificial Intelligence (2002)
14. Stentz, A., Bares, J., Pilarski, T., Stager, D.: The crusher system for autonomous navigation. In: AUVSIs Unmanned Systems (2007)
15. Tompkins, P., Stentz, A., Whittaker, W.: Mission planning for the sun-synchronous navigation field experiment. In: IEEE International Conference on Robotics and Automation (2002)