Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain

David Silver, J. Andrew Bagnell, Anthony Stentz
Carnegie Mellon University
Pittsburgh, PA, USA

June 24, 2010

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 complex unstructured terrain, this coupling (in the form of a cost function) has a large impact on robot behavior and performance, necessitating a robust design. This paper explores the application of *Learning from Demonstration* to this task for the Crusher autonomous navigation platform. Using expert examples of desired 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 and imperfect demonstration 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 effort. Experimental results are presented from autonomous traverses through complex natural environments.

1 Introduction

The capability of autonomous robotic systems to successfully navigate through difficult environments continues to advance [Kelly et al., 2006, Buehler, 2006]. 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.

A common division of responsibility in mobile robotics tasks a perception system with constructing a discrete model of the environment. This model is then interpreted into an appropriate form for use by a planning system, which determines the robot's actions. In the simplest case, this interpretation determines which locations in the environment model the robot can and cannot traverse through (i.e. an obstacle versus freespace or traversable versus non-traversable distinction). A planning system then computes a path through traversable regions of the environment. In this way, perception and planning are coupled through a binary classification of the environment.

While such a binary approach was utilized in the early days of outdoor autonomous navigation [Olin and Tseng, 1991], more recent work of the past decade has shown it to be insufficient for navigating complex unstructured environments. Rough natural terrain is not easily partitioned into clear traversable and non-traversable classes; while certain objects may be obvious non-traversable obstacles, unstructured environments generally present a continuum of terrain that could fall into the traversable category such as steep slopes, ditches, smaller (surmountable) objects, and widely varying vegetation (Figure 1). A binary interpretation results in no distinction between these different terrain features, resulting in behavior that is either overly aggressive or conservative (Figure 2).

As these issues have become better understood, the result has been a move to systems that use a continuous coupling between perception and planning [Kelly et al., 2006, Lacaze et al., 2002, Stentz et al., 2007, Singh et al., 2000, Urmson et al., 2006, Biesiadecki and Maimone, 2006]. Such a coupling is commonly called a *Cost Function*. A cost function maps terrain features produced by perception to a scalar cost value, with lower cost terrain being preferable to higher cost terrain. A planning

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
24 JUN 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. 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 complex unstructured terrain this coupling (in the form of a cost function) has a large impact on robot behavior and performance, necessitating a robust design. This paper explores the application of Learning from Demonstration to this task for the Crusher autonomous navigation platform. Using expert examples of desired 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 and imperfect demonstration 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 effort. Experimental results are presented from autonomous traverses through complex natural environments.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

system then computes a trajectory that minimizes the accrued cost of traversed terrain.

While using a continuous definition of cost allows for more intelligent behavior, it also increases the complexity of deploying a properly functioning robot. Systems that used a traversable or non-traversable distinction only had to solve a binary classification problem. This mapping completely determined the behavior of the robot (with respect to where it preferred to drive); essentially, the desired behavior of the robot was encoded in the classification function mapping perceptual data to traversable or non-traversable. However, in a system with continuous costs, in essence a full regression problem must be solved. That is, the desired behavior of a robot is encoded not in a mapping from terrain properties to a binary class, but in a mapping from terrain properties to a scalar value that encodes preferences (see Figure 2).

This mapping from terrain properties to cost is far more complex than a binary mapping, as it encodes far more complex behavior (through continuous output). Worse, the preferences amongst terrain types are not always well understood, because the metric for performance is rarely concretely defined. Common metrics for autonomous behavior include maximizing safety or probability of success, minimizing distance traveled or time taken, minimizing net energy loss, minimizing observability or maximizing sensor coverage. However, 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. Encoding a single metric into a cost function is sufficiently difficult, but properly inferring the correct weighting to construct a multi-criterion optimization problem is even more daunting. Fundamentally, while humans are good at driving in complex or off-road environments, they are not good at articulating their preferences in doing so.

Therefore, as preferences that mobile robotic systems are expected to exhibit become more complex, the task of encoding these preferences in a cost function will become both more difficult and time consuming, while at the same time becoming more central to improving performance. Unfortunately, this challenge has not received a great deal of focus; it is often only briefly mentioned in the literature. With respect to costs defined over patches of terrain, the mapping to cost from terrain parameters or features is rarely described in detail, usually with the statement that it was simply constructed and hand-tuned to provide good empirical performance.

The issue of cost function design and tuning was central during the development of Crusher [Stentz et al., 2007, Bagnell et al., 2010](Figure 1), a vehicle designed from scratch for off-road autonomous mobility. Crusher is capable of traversing steep slopes, deep washes and ditches, large boulders, and dense vegetation. Robust navigation through such terrain requires a proper description of preferences over such terrain (e.g. how far should Crusher travel through dense vegetation to avoid a small ditch). Complicating matters further is the complexity of the perceptual inputs that navigating such environments necessitates; complex terrain requires a high dimensional description in order to sufficiently encode all the relevant features (see Figure 3). Finally, Crusher receives multiple sources of perceptual input: along with the ever changing stream of perceptual data produced onboard the vehicle, static prior data was available at varying resolution over large areas of operation (potentially hundreds of square kilometers) [Silver et al., 2006, Silver et al., 2008]. This necessitated multiple cost functions for interpreting these disjoint data sources that encoded robust behavior both individually and when fused together.

This paper explores the application of a learning from demonstration approach to this challenge. Specifically, algorithms are presented for learning generalizable terrain cost functions from expert demonstration of desired behavior. This approach can both reduce development effort and improve performance when applied to mobile robotic systems. The next section provides a brief overview of previous and related work in coupling perception and planning systems through cost functions. Section 3 presents the basic theory of our approach, and Section 4 describes its practical adaptation to mobile robotics. Experimental results from the Crusher system are provided in Section 5, with discussion and conclusion in Section 6.

2 Related Work

2.1 Hand Tuned Cost Functions

Manual hand tuning and engineering has by far been the most common approach to constructing cost functions for mobile robotic systems. Often, this is done with little or no formalism; cost function design by hand remains one of the 'black arts' of mobile robotics, and has been applied to untold numbers of robotic systems. However, there has previously been work that created reusable frameworks for manual design and tuning of cost functions to map perceptual features into scalar costs [Stentz and Hebert, 1995, Singh et al., 2000, Balakirsky and Lacaze, 2000,

Figure 1: The Crusher autonomous mobility platform is capable of cross-country traverse through rough, complex, and unstructured terrain

Seraji and Howard, 2002, Huertas et al., 2005, Biesiadecki and Maimone, 2006] The common thread amongst these approaches is that they construct a parameterized mapping from a perceptual input space to a scalar cost, and then adjust the mapping to produce costs that result in good robot performance. In this way, the behavior of an entire robotic system is simply tuned to get a desired result.

This approach has several disadvantages. It is potentially quite time consuming; as complex environments necessitate full featured and high dimensional descriptions, often on the order of dozens of features per discrete location. Worse still, there is often not a clear relationship between these features and cost. Therefore, engineering a cost function by hand is akin to manually solving a high dimensional optimization problem using local gradient methods. Evaluating each candidate function requires validation through either actual or simulated robot performance. Such a manual process requires a very detailed knowledge of both a robot's perception and planning systems; therefore the necessary effort must come from a potential small pool of full system experts.

Additionally, this tedious process is never truly completed, but rather remains ongoing. Whenever incorrect behavior is observed, the cost function may need to be revisited. This is especially problematic when operating in a novel environment or scenario, as there is no guarantee that a manually tuned cost function will generalize well. It is also possible that multiple cost functions may be necessary to support different subsets of available perceptual data. Finally, if the perception system itself is ever modified to add, remove, or modify existing features (a very common occurrence during development of a fielded system), the cost function must also be redesigned or retuned.

However, perhaps the biggest issue with manually tuning a cost function is that there is no formalism behind it. That is, there is no theory to explain why a certain cost function produces good behavior, or how well it will generalize to novel environments. Further, even if sufficient validation is performed on a candidate cost function, there is nothing to indicate its absolute performance; that is, it may be sufficient, but could it still be better? As with many optimization procedures, manual parameter tuning is likely to suffer diminishing returns, and will quickly reach a point where human effort and patience is unlikely to improve a parameterization further. Such a forced early stopping will always leave lingering questions, and can make blame assignment difficult. That is, if the robot experiences a navigation failure (e.g. drives over something it should not have, or avoids something it did not need to) its unclear whether the blame lies with perception, planning, or their coupling (the cost function).

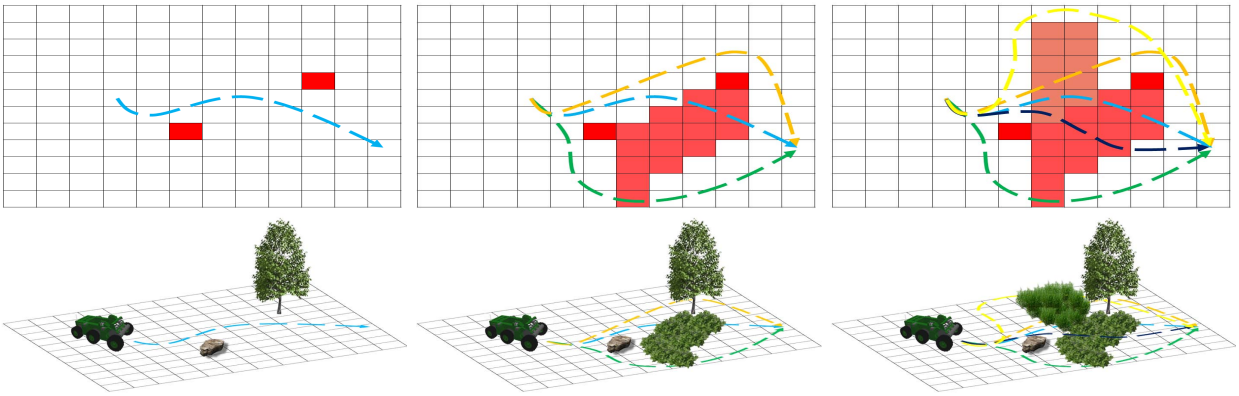


Figure 2: Examples illustrating the effect of different cost functions on a robot's behavior. **Left:** In simple environments with only well defined obstacles and freespace, a wide range of cost functions will produce nearly equivalent behavior. **Center:** With a single class of intermediate terrain (bushes), several different paths are optimal depending on the relative cost of bushes and freespace. **Right:** With multiple classes of intermediate terrain, the variety of paths that could be optimal (depending on the relative cost of bushes, grass, and freespace) is further increased. In such scenarios, the tuning of a cost function will dramatically affect a robot's behavior. If only binary cost functions were utilized, the variety of reproducible behavior would be diminished.

2.2 Physical Simulation

Another common approach to engineering this problem away is the use of accurate physical simulation [Olin and Tseng, 1991, Kelly, 1995, Iagnemma et al., 1999, Helmick et al., 2009] to attempt to predict the consequences of a robot traversing a patch of terrain. Instead of requiring a mapping from perceptual features to cost, this can transform the problem to one of mapping from predicted vehicle state to cost. At times, this can result in a simpler or more intuitive domain for constructing a cost function; however, it could also make the problem more difficult (if the vehicle state is higher dimensional than the perceptual feature space). Regardless, there is still a requirement to construct a mapping from a description of a behavior to a cost function that implies preferences.

Another possible use of physical simulation is to model the probability of terrain being traversable. That is, a simulation could compute the probability that interaction with a specified terrain patch would result in a vehicle failure (e.g. exceed a tip-over angle or known force limits). The concept of explicitly using analog (as opposed to binary) traversability as a cost function (and therefore defining terrain preferences with respect to traversability) is yet another way of engineering around a manually tuned cost function. However, it is not without serious drawbacks. Fundamental amongst them is that it limits the metric that the autonomous system can optimize to pure maximization of vehicle safety. While in certain contexts this may be desirable, the inability to ever indicate that slight risk should be taken to balance a different tradeoff (distance traveled, time taken, energy consumption, etc.) can seriously limit a system. For example, a mobile system that navigated purely based on traversability would likely have difficulty properly differentiating between perfectly flat and slightly angled terrain, or between small obstacles of different size, or sparse vegetation of varying height. The end result is that a traversability score rearely directly maps to a cost that produces desired behavior; a cost function (that may heavily depend on traversability estimates) is still required. Finally, performing such a full, accurate simulation is quite difficult, especially using noisy perceptual data as input.

2.3 Supervised Classification

A different approach that can also simplify the parameter tuning problem is the use of supervised classification. The general technique is to reduce a high dimensional perceptual feature space into a lower dimensional space with more semantic meaning. Supervised classification is an obvious choice for such a transformation: while an engineer may have difficulty in designing rules to classify different patches of terrain, he can much more easily define the class that each patch should belong to. Rather than manually constructing rules to map from perceptual features to

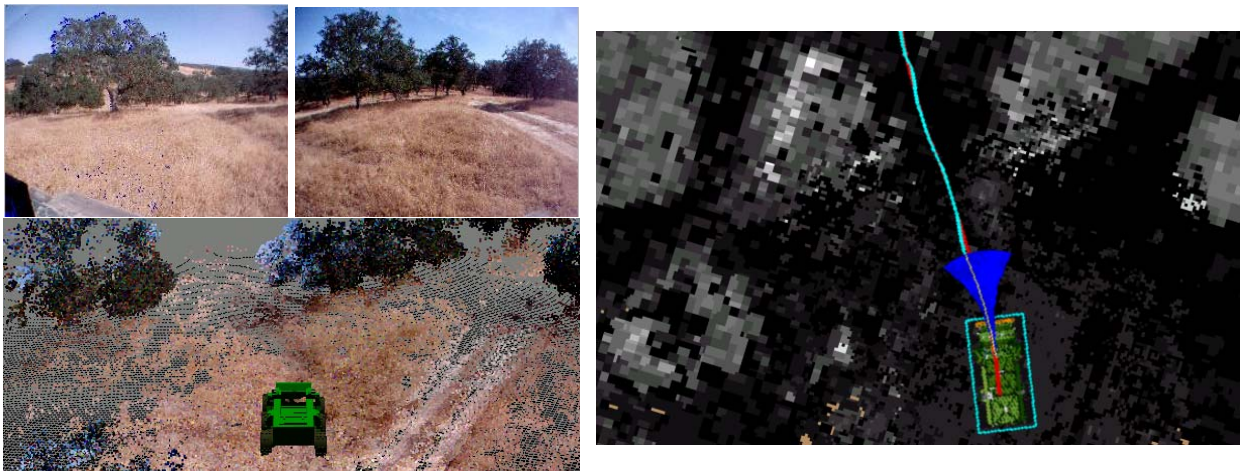


Figure 3: **Left:** Raw perception data from Crusher, in the form of camera images and colorized LiDAR. **Right:** 2D costs derived from perception data, and a resulting planned path. Brighter pixels indicate higher cost.

classifications, a learning system can automatically generate the necessary rules, given examples of sets of features and their desired classification.

The primary advantage (with respect to autonomous behavior) of performing a supervised classification is a remapping of a perceptual feature space into one that is lower dimensional and potentially more intuitive. Some perceptual features have an intuitive, monotonic relationship to concepts such as safety and speed. However, many features do not provide this intuition; for example, if a terrain patch has a high blue tint, does that make it more or less dangerous? If a supervised classification stage is inserted after perceptual feature generation but before costing, it may simplify the parameter tuning problem; not only may there be fewer parameters, but they may be more intuitive, especially if classes are defined as semantic or material distinctions (bush, rock, tree grass, dirt, etc.). For this reason, this general approach is popular for the purpose of perceptual interpretation in unstructured environments, and has been widely applied [Talukder et al., 2002, Manduchi et al., 2005, Lalonde et al., 2006, Bradley et al., 2007, Dima et al., 2004, Rasmussen, 2002, Angelova et al., 2007, Halatci et al., 2007, Karlsen and Witus, 2007].

However, while supervised multi-class classification may make costing more tractable, it does not actually solve the core problem; the parameter tuning task has only been simplified. While classifier outputs may have a more intuitive relationship to the correct behavior, it can still be difficult and time consuming to determine the proper relative weightings of various classes. As there is no guarantee that the selected taxonomy is relevant to mobility behavior, it may take a lot of effort for a classification system to distinguish varying classes of vegetation, but that effort is wasted if those vegetation classes are indistinguishable with regards to mobility. In such a case, a large error in classification would produce only a small error in vehicle behavior. The converse can also occur; tiny errors in classification between certain classes may produce large errors in behavior (if for example traversable vegetation is confused with a rigid obstacle). This makes it difficult to relate classification performance to mobility performance. Defining classes specifically with respect to relative mobility [Happold and Ollis, 2007] is a partial solution, but makes data labeling far less intuitive.

Additionally, this approach can potentially hurt overall performance. Classification into a lower dimensional space can be viewed as a form of dimensionality reduction. However, as classes may not be directly related to robot mobility or behavior, this reduction does not take into account the final use of the data, and can potentially obscure or eliminate useful information. Finally, while supervised classification can reduce the time and effort required in one tuning problem, the total effort throughout the system is not necessarily reduced; the classification system now must be tuned. Labeling a large representative data set for training and validation also entails a significant manual effort. Whenever the classifier changes, due to perception system change or additions to the training set, the costing of the classifications must be re-examined and possibly re-tuned.

An alternate use of supervised classification has

been to treat the task as a specific two class problem: classifying terrain as either traversable or non-traversable. Labels can be gathered either offline [Seraji and Howard, 2002, Howard et al., 2007] or online [Thrun et al., 2006, Sun et al., 2007] by observing where an expert drives a robot; nearby terrain that is not traversed is treated as having been labeled non-traversable in a noisy manner[1]. [Ollis et al., 2007] involves similar data collection, but only for labeling of traversable terrain; explicit examples of non-traversable regions are not required. The issues with this approach are twofold. First, the labeling is likely to be very noisy, especially if examples of non-traversable terrain are obtained by using terrain located near traversable examples. More importantly, just as with physical simulation a probability of traversability, even if accurate, rarely directly maps to the correct behavior.

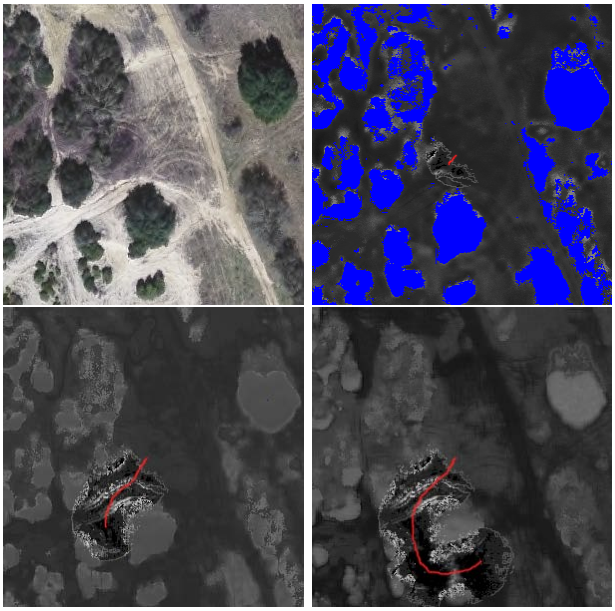


Figure 4: An example of near-to-far learning from [Sofman et al., 2006]. In this example, as a robot drives through an environment (left to right) it uses its short range sensors to train the interpretation of a far range sensor (in this case the satellite image at left)

2.4 Self-Supervised Learning from Experience

In contrast with learning approaches that require explicit supervision from an (usually human) expert, self-supervised approaches require no expert interaction. Instead, a robot uses its own interactions with the world to slowly learn how to interpret what it perceives; the robot learns and adapts from experience. As opposed to requiring outside supervision and instruction, the robot can learn from its own mistakes. This allows robots equipped with self-supervision to adapt to novel environments or scenarios with little to no human intervention, and is a powerful tool for achieving both robustness and reusability. Unsurprisingly, online self-supervised approaches to learning have gained increasing popularity in recent years.

Approaches for self-supervised online learning can be divided into two distinct classes. The first is near-to-far learning that learns how to interpret a far-range, lower resolution sensor using a near range, higher-resolution sensor. Near-to-far learning is achieved by recalling how specific patches of terrain were perceived by a far range sensor, and then later observing them with a near-range sensor. This provides a correspondence between the output of the two sensing modalities, and provides the necessary data to learn a mapping. Examples of such approaches include learning to interpret monocular vision systems from shorter range LiDAR [Dahlkamp et al., 2006] or stereo [Hadsell et al., 2009] range data, learning to map near range traversability estimates to far range estimates [Bajracharya et al., 2008, Procopio et al., 2007], and learning to map full cost functions from near to far [Sofman et al., 2006] (Figure 4). However, near-to-far learning still requires a (preexisting) correct interpretation of near-range sensors. In fact, this near-range interpretation is increased in importance, since it is being used as a ground truth signal for learning.

The other distinct class of self-supervised learning is learning from proprioception, also called learning from underfoot. As opposed to near-to-far learning, the ground truth signal comes not from a higher resolution exteroceptive sensor, but from proprioceptive sensors. Aside from this distinction, the methodology is quite similar: as the robot drives over a patch of terrain, it recalls how that terrain appeared in its near-range sensors just moments ago. This sets up a correspondence between how terrain appears, and how the robot interacts with it. These correspondences can be used to learn to model the robot's interactions with terrain by predicting various terramechanical properties, such as roughness [Stavens and Thrun, 2006], vehicle slip [Angelova et al., 2007], soil cohesion [Iagnemma et al., 2004], or vegetation height

[1]This approach is fundamentally different from standard imitation learning in that demonstration is used purely as a data labeling technique; offline hand labeling could be used with similar results

[Wellington et al., 2006]. Predictions can then be either be used directly to determine robot behavior (i.e. as input into a cost function), or used to improve the accuracy of a physical simulation [Helmick et al., 2009]. While these approaches certainly provide useful information that can drastically improve a robot's robustness and adaptation, they do not directly address the general problem of mapping from features of terrain to terrain preferences.

Another approach to learning from proprioception is to attempt to directly learn traversability by observing what terrain a robot can and can not successfully traverse [Shneier et al., 2008, Kim et al., 2006]. However, this approach has the same drawbacks as previously described techniques based on pure traversability. That is, it ignores the possibility of relative preferences amongst equally traversable patches of terrain. This issue also applies to other single metric learning from proprioception; for example, a robot that learns to estimate its own speed over different terrains would never be able to differentiate between terrains on which maximum speed is possible but different mobility risk is encountered. More importantly, this online approach to labeling terrain as non-traversable requires explicit interaction with non-traversable terrain (e.g. the robot must drive into an obstacle to learn that it is an obstacle). Not only is this dangerous, it also results in a potentially difficult blame assignment problem (determining which patch or patches of terrain were responsible for a failure).

2.5 Learning From Expert Demonstration

Given the difficulty in manually engineering a coupling between perception and planning systems, an alternative solution is to avoid this problem altogether by learning to directly map perception to actions. This can be accomplished through learning from demonstration, also known as imitation learning. A key principle of imitation learning is that while it may be very difficult to quantify why a certain behavior is desirable, the actual correct behavior is usually known by a human expert. Therefore, rather than having a human expert tune a system to achieve desired behavior, the expert can demonstrate desired behavior and the robot can tune itself to match the demonstration. Learning from demonstration has a long history with mobile robotics, starting with the ALVINN system [Pomerleau, 1989] and continuing through more recent applications [LeCun et al., 2006, Howard et al., 2005, Hamner et al., 2006].

These approaches to learning from demonstration all fall into the category of action prediction: given the state of the world, or features of the state of the world, they attempt to predict the action that the human demonstrator would have selected. The fundamental problem with pure action prediction is that it is a purely reactive approach to planning. There is no attempt to model or reason about the consequences of a chosen action. Therefore, all relevant information must be encoded in the current state or features of the state. For certain scenarios this is feasible; path trackers are a classic example. However, in general the use of purely reactive systems is incredibly difficult for planning long range, goal directed behavior through complex unstructured environments.

An alternative to action prediction finds its roots in the idea of *Inverse Optimal Control* [Kalman, 1964]. While optimal control seeks a trajectory through a state space that optimizes some known metric, inverse optimal control seeks a metric such that a known trajectory through a state space is optimal under said metric. Within mobile robotics, the parallel would be to learn a cost function such that a robot's planning system will reproduce expert[2] demonstrated behavior. Such an approach automates the hand construction and tuning of cost functions that is prevalent in actual deployments. Not only does automating this procedure result in large reduction in design effort, it can potentially produce a better coupling; automatic optimization need not be nearly as concerned with diminishing returns, and can continue until convergence. Further, an automated approach can take advantage of standard cross validation techniques to ensure that a learned coupling is robust and will generalize well. Finally, if collection of expert demonstration includes raw sensor data, changes to sensor processing within a perception system do not result in additional human effort with respect to a cost function; existing demonstrations can simply be reprocessed and coupling relearned.

Inverse Reinforcement Learning [Ng and Russell, 2000] was the first application of this idea to the framework of Markov Decision Processes commonly used for motion planning in mobile robotics. This framework was later modified into a new approach known as *Apprenticeship Learning* [Abbeel and Ng, 2004]. Apprenticeship learning resulted in an algorithm that produced linear cost functions such that any planned behavior would have the same cost as demonstrated behavior (between equivalent start and end conditions); however there

[2]This expert need no longer be a full system expert, but rather only needs to have the necessary intuition to understand how the robot should drive

was no mechanism for explicitly matching expert behavior. Additionally, the final solution was a stochastic mixture of multiple policies. The *Maximum Margin Planning* (MMP) [Ratliff et al., 2006] framework addressed these problems by producing a single deterministic solution while also ensuring an upper bound on the mismatch between demonstrated and planned behavior. More recent work has extended the MMP framework to non-linear cost functions [Ratliff et al., 2009, Silver et al., 2008], and was applied to Crusher for the task of learning a cost function to interpret both static and dynamic perceptual data [Silver et al., 2009a, Silver et al., 2009b].

3 Learning to Interpret Perceptual Data from Expert Demonstration

The MMP framework treats learning from demonstration as a constrained optimization problem. In decision theory, a utility function is defined as a relative ordering over all possible options. Since most mobile robotic systems encounter an infinite number of possible plans, such an explicit ordering is not possible. Instead, plans are scored by a cost function, and cost defines the ordering. However, this core idea of an ordering of preferences remains useful. If an expert can concretely define a relative ordering over even a small subset of possible plans, this ordering becomes a constraint on the cost function: the costs assigned to each possible plan must match the relative ordering. The more information on relative preferences an expert provides, the more the cost function is constrained.

This section first derives the basic MMP algorithm for learning a linear cost function to reproduce expert demonstration. Next the LEARCH algorithm (LEArning to seaRCH) [Ratliff et al., 2009, Silver et al., 2008] is presented for extending this approach to non-linear cost functions. The MMP framework and associated algorithms are defined over general Markov Decision Processes. However, without loss of generality, this derivation is restricted to deterministic MDPs with a set of absorbing states; that is, goal directed path or motion planners. This change is purely for notational simplification, as most mobile robotic systems use planners of this form. Additionally, only cost functions defined over states are considered, as opposed to full state-action pairs (see Section 6).

3.1 Maximum Margin Planning with Linear Cost Functions

Consider a state space $\mathcal{S}$ through which a planner operates (e.g. $\mathcal{S} = \mathbb{R}^2$). A feature space $\mathcal{F}$ is defined over $\mathcal{S}$. That is, for every $x \in \mathcal{S}$, there exists a corresponding feature vector $F_x \in \mathcal{F}$. F_x can be considered as the raw output of a perception system at state x[3]. For the output of a perception system to be used by a planner, it must be mapped to a scalar cost value; Therefore, C is defined as a cost function, $C : \mathcal{F} \rightarrow \mathbb{R}^+$. The cost of a state x is $C(F_x)$. For the moment, only linear cost functions of the form $C(F) = w^T F$ are considered; the weight vector w completely defines the cost function. Finally, a path P is defined as a sequence of states in $\mathcal{S}$ that lead from a start s to a goal g. The cost of an entire path is simply defined as the sum of the costs of all states along the path, or alternatively the cost of the cumulative feature counts

$$C(P) = \sum_{x \in P} C(F_x) = \sum_{x \in P} w^T F_x = w^T \sum_{x \in P} F_x \quad (1)$$

Now consider an example path P_e from a start state s_e to a goal state g_e. If this example path is provided via expert demonstration, then its is reasonable to consider applying inverse optimal control; that is, seeking to find a cost function C such that P_e is the optimal path from s_e to g_e. While a single example demonstration does not imply a single cost function, it does constrain the space of cost functions $\mathcal{C}$: only cost functions that consider P_e the optimal path are acceptable. For now, it is assumed that all P_e are at least near optimal under at least one $C \in \mathcal{C}$; Section 4.2 will relax this assumption.

If a regularization term is also added to encourage simple solutions, then the task of finding an acceptable cost function from an example can be phrased as the following constrained optimization problem:

$$\begin{aligned} &\text{minimize } O(w) = ||w||^2 \quad (2) \\ &\text{subject to the constraints} \\ &\sum_{x \in \hat{P}} (w^T F_x) \geq \sum_{x \in P_e} (w^T F_x) \\ &\forall \hat{P} \; s.t. \; \hat{P} \neq P_e, \;\; \hat{s} = s_e, \;\; \hat{g} = g_e \end{aligned}$$

Unfortunately, this optimization has a trivial solution: $w = \vec{0}$. This issue can be addressed by including a margin in each constraint. The size of the

[3]For now it is assumed that the assignment of a feature vector to a state is static; the extension to dynamic assignment is addressed in Section 4.1

margin is dependent on the similarity between paths; the example only needs to be slightly lower cost than a very similar path, but should be much lower cost than a very different path. Similarity between P_e and an arbitrary path P is encoded by a loss function $L(P_e, P)$, or alternatively $L_e(P)$[4]. The definition of the loss function is somewhat application dependant; the simplest form would be to simply consider how many states the two paths share (a Hamming loss). The effect of the scale of the margin is removed by the regularization term. The constrained optimization can now be rewritten as

$$\begin{aligned} &\text{minimize } O(w) = ||w||^2 \qquad (3)\\ &\text{subject to the constraints}\\ &\sum_{x\in\hat{P}}(w^T F_x - L_e(x)) \geq \sum_{x\in P_e}(w^T F_x)\\ &\forall \hat{P}\ s.t.\ \hat{P} \neq P_e,\ \ \hat{s} = s_e,\ \ \hat{g} = g_e \end{aligned}$$

$$L_e(x) = \begin{cases} 1 & \text{if } x \in P_e \\ 0 & \text{otherwise} \end{cases}$$

Depending on the state space, and the distance from s_e to g_e, there are likely to be an infeasible (and possibly infinite) number of constraints; one for each alternate path to the demonstrated example. However, it is not necessary to enforce every constraint. For any candidate cost function, there is a minimum cost path between any two waypoints, P_*. It is only necessary to enforce the constraint for P_*, as once it is satisfied by definition all other constraints will be satisfied. With this single constraint, (3) becomes

$$\begin{aligned} &\text{minimize } O(w) = ||w||^2 \qquad (4)\\ &\text{subject to the constraint}\\ &\sum_{x\in P_*}(w^T F_x - L_e(x)) \geq \sum_{x\in P_e}(w^T F_x)\\ &P_* = \arg\min_{\hat{P}} \sum_{x\in\hat{P}}(w^T F_x - L_e(x)) \end{aligned}$$

It may not always be possible to exactly meet this constraint (the margin may make it impossible). Therefore, a slack term ζ is added to allow for this possibility.

$$\begin{aligned} &\text{minimize } O(w) = \lambda||w||^2 + \zeta \qquad (5)\\ &\text{subject to the constraint}\\ &\sum_{x\in P_*}(w^T F_x - L_e(x)) \ - \sum_{x\in P_e}(w^T F_x) + \zeta \geq 0 \end{aligned}$$

The slack term ζ accounts for the error in meeting the constraint, while λ balances the tradeoff in the objective between regularization and meeting the constraint. However, the slack variable will always be tight; that is ζ will always be exactly equal to the difference in path costs. Therefore, ζ can be replaced in the objective by the constraint, resulting in the following (unconstrained) optimization problem

$$\begin{aligned} &\text{minimize } O(w) = \lambda||w||^2 + \qquad (6)\\ &\sum_{x\in P_e}(w^T F_x) \ - \sum_{x\in P_*}(w^T F_x - L_e(x)) \end{aligned}$$

or alternatively

$$\begin{aligned} &\text{minimize } O(w) = \lambda||w||^2 + \qquad (7)\\ &\sum_{x\in P_e}(w^T F_x) \ - \min_{\hat{P}}\left[\sum_{x\in\hat{P}}(w^T F_x - L_e(x))\right] \end{aligned}$$

The final optimization seeks to minimize the difference in cost between the example path P_e and the (loss augmented) optimal path P_*, subject to regularization. $O(w)$ is convex, but non-differentiable; therefore, instead of gradient descent, it can be minimized using the sub-gradient method with learning rate η. The sub-gradient of O with respect to w is

$$\nabla O = 2\lambda w + \sum_{x\in P_e} F_x \ - \sum_{x\in P_*} F_x \qquad (8)$$

Intuitively, (8) says that the direction that will most minimize the objective function is found by comparing feature counts. If more of a certain feature is encountered on the example path than the current minimum cost path P_*, the weight on that feature (and therefore the cost) should be decreased. Likewise, if less of a feature is encountered on the example path than on P_*, the weight should be increased. Although the margin does not appear in the final sub-gradient, it does affect the computation of P_*. The final linear MMP algorithm consists of iteratively computing feature counts and then updating the cost function until convergence. However, one final caveat is to ensure that only cost functions that map to $\mathbb{R}^+$ are considered (a requirement of most

[4]As a path is considered just a sequence of states, the loss function can be defined either over a full path or over a single state.

Algorithm 1: The linear MMP algorithm

Inputs : Example Paths $P_e^1, P_e^2, ..., P_e^n$, Feature Map $\mathcal{F}$

$w_0 = \vec{0}$;
for $j = 1...K$ **do**
  $\mathcal{M} = \texttt{buildCostmap}(w_{j-1}, \mathcal{F})$;
  $F_e = F_* = \vec{0}$;
  foreach P_e^i **do**
    $P_*^i = \texttt{planLossAugPath}(\texttt{start}(P_e^i), \texttt{goal}(P_e^i), \mathcal{M})$;
    foreach $x \in P_e^i$ **do**
      $F_e += F_e + F_x$;
    foreach $x \in P_*^i$ **do**
      $F_* = F_* + F_x$;
  $w_j = w_{n-1} + \eta_i[F_* - F_e - \lambda w_{j-1}]$;
  $\texttt{enforcePositivityConstraint}\ (w_j, \mathcal{F})$;
return w_K

motion and path planners). This is achieved by identifying F such that $w^T F \leq 0$, and projecting w back into the space of allowable cost functions.

The MMP framework easily supports the use of multiple example paths. Each example implies its own constraints as in (4), its own objective as in (7), and its own sub-gradient as in (8). Updating the cost weights can take place either on a per example basis, or the feature counts can be computed in a batch with a single update. The latter is computationally preferable, as it may result in fewer cost function evaluations, and projections back into the space of allowable cost functions. The final linear MMP algorithm is presented in Algorithm 1.

3.2 MMP with Non-Linear Cost Functions

The derivation to this point has assumed that the space of possible cost functions $\mathcal{C}$ consists of all functions of the form $C(F) = w^T F$. Extension to other, more descriptive spaces of cost functions is possible by considering (7) for any cost function C

$$\text{minimize } \mathcal{O}[C] = \lambda \text{REG}(C) + \sum_{x \in P_e} C(F_x) - \min_{\hat{P}} \left[\sum_{x \in \hat{P}} (C(F_x) - L_e(x)) \right] \quad (9)$$

$\mathcal{O}[C]$ is now an objective functional over a cost function, and REG represents a regularization functional. We can now consider the sub-gradient in the space of cost functions

$$\nabla \mathcal{O}_F[C] = \lambda \nabla \text{REG}_F[C] + \sum_{x \in P_e} \delta_F(F_x) - \sum_{x \in P_*} \delta_F(F_x) \quad (10)$$
$$P_* = \arg\min_{\hat{P}} \sum_{x \in \hat{P}} (C(F_x) - L_e(x))$$

where δ is the Dirac delta at the point of evaluation. Simply speaking, the functional gradient is positive at values of F corresponding to states in the example path, and negative at values of F corresponding to states in the current planned path. If the paths both contain a state corresponding to F, their contributions cancel.

Applying gradient descent directly in this space would result in an extreme form of overfitting; essentially, it would involve raising or lowering the cost associated with specific values of F encountered on either path, and would therefore produce no generalization whatsoever. Instead, a different space of cost functions is considered

$$\mathcal{C} = \{C \mid C = \sum_i \eta_i R_i(F), \ R_i \in \mathcal{R}, \ \eta_i \in \mathbb{R}\} \quad (11)$$
$$\mathcal{R} = \{R \mid R : \mathcal{F} \to \mathbb{R} \ \wedge \ \text{REG}(R) < \nu\}$$

$\mathcal{C}$ is now defined as the space of weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of functions of limited complexity that map from the feature space to a scalar. Choices of $\mathcal{R}$ include linear functions, parametric functions, neural networks, decision trees, etc. As in gradient boosting [Mason et al., 2000], this space represents a limited set of 'directions' for which a small step can be taken; the choice of the direction set in turn controls the complexity of $\mathcal{C}$.

With this new definition, a gradient descent update takes the form of projecting the functional gradient[5] onto the direction set by finding the element $R_* \in \mathcal{R}$ that maximizes the inner product $\langle -\nabla \mathcal{O}_F[C], R_* \rangle$. The maximization of the inner product between the functional gradient and the hypothesis space can be understood as a learning problem:

$$\begin{aligned} R_* &= \arg\max_R \langle -\nabla \mathcal{O}_F[C], R \rangle \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} -\nabla \mathcal{O}_F[C] R(F_x) \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} \alpha_x y_x R(F_x) \\ \alpha_x &= |\bigtriangledown \mathcal{O}_{F_x}[C]| \quad y_x = -\text{sgn}(\bigtriangledown \mathcal{O}_{F_x}[C]) \end{aligned} \quad (12)$$

In this form, it can be seen that finding the projection of the functional gradient involves solving a weighted

[5] For the moment, the regularization term is ignored.

classification problem; the element of $\mathcal{R}$ that best discriminates between features vectors for which the cost should be raised or lowered maximizes the inner product. Alternatively, defining $\mathcal{R}$ as a class of regressors adds an additional regularization to each individual R_* [Ratliff et al., 2009]. Intuitively, the regression targets y_x are positive in regions of the feature space that the planned path visits more than the example path (indicating a desire to raise the cost), and negative in regions that the example path visits more than the planned path. Each regression target is weighted by a factor α_x based on the magnitude of the functional gradient.

In comparison to the linear MMP formulation, this approach can be understood as trying to minimize the error in *visitation counts* instead of feature counts. For a given feature vector F and path P, the visitation count U is the cumulative count of the number of states $x \in P$ such that $F_x = F$. The visitation counts can be split into positive and negative components, corresponding to the current planned and example paths. Formally

$$\begin{aligned} U_+(F) &= \sum_{x \in P_*} \delta_F(F_x) \\ U_-(F) &= \sum_{x \in P_e} \delta_F(F_x) \\ U(F) = U_+ - U_- &= \sum_{x \in P_*} \delta_F(F_x) \quad - \quad \sum_{x \in P_e} \delta_F(F_x) \end{aligned} \tag{13}$$

Comparing this formulation to (10) demonstrates that the planned visitation counts minus the example visitation counts equals the negative functional gradient (ignoring regularization). This allows for the computation of regression targets and weights purely as a function of the visitation counts, providing a straight forward implementation (Algorithm 2) making use of off the shelf regression approaches (represented by R_j).

A final addition to this algorithm involves a slightly different approach to optimization. Gradient descent can be understood as encouraging functions that are 'small' in the l_2 norm; by controlling the learning rate η and the number of epochs, it is possible to constrain the complexity of the learned cost function. However, instead we can consider *exponentiated functional gradient descent*, which is a generalization of exponentiated gradient to functional gradient descent [Ratliff et al., 2009]. Exponentiated functional gradient descent encourages functions that are 'sparse' in the sense of having many small values and a few potentially large values. This change results in $\mathcal{C}$ being

Algorithm 2: The LEARCH algorithm

Inputs : Example Paths $P_e^1, P_e^2, ..., P_e^n$, Feature Map $\mathcal{F}$

$C_0 = 1$;
for $j = 1...K$ **do**
  $\mathcal{M} =$ `buildCostmap`$(C_{j-1}, \mathcal{F})$;
  $U_+ = U_- = \vec{0}$;
  foreach P_e^i **do**
    $P_*^i =$ `planLossAugPath(start(`P_e^i`), goal(`P_e^i`),` $\mathcal{M}$`)`;
    foreach $x \in P_e^i$ **do**
      $U_-(\mathcal{F}_x) = U_-(\mathcal{F}_x) + 1$;
    foreach $x \in P_*^i$ **do**
      $U_+(\mathcal{F}_x) = U_+(\mathcal{F}_x) + 1$;
  $T_f = T_o = T_w = \emptyset$;
  $U = U_+ - U_-$;
  foreach $\mathcal{F}_x$ *such that* $U(\mathcal{F}_x) \neq 0$ **do**
    $T_f = T_f \bigcup \mathcal{F}_x$;
    $T_o = T_o \bigcup$ `sgn(`$U(\mathcal{F}_x)$`)`;
    $T_w = T_w \bigcup |U(\mathcal{F}_x)|$;
  $R_j =$ `trainWeightedRegressor(`T_f, T_o, T_w`)`;
  $C_j = C_{j-1} * e^{\eta_j R_j}$;
return C_K

redefined as

$$\mathcal{C} = \{C \mid C = e^{\sum_i \eta_i R_i(F)}, \;\; R_i \in \mathcal{R}, \;\; \eta_i \in \mathbb{R}\} \tag{14}$$

Another beneficial effect of this redefined space is that $\mathcal{C}$ naturally maps to $\mathbb{R}^+$ without any need for projecting the result of each gradient descent update into the space of valid cost functions. This final algorithm for non-linear inverse optimal control is called LEARCH [Ratliff et al., 2009, Silver et al., 2008] and is presented in Algorithm 2. An example of the algorithm in action is presented in Figure 5.

It should be noted that while seemingly similar, LEARCH is fundamentally different from supervised classification approaches presented in [Thrun et al., 2006, Sun et al., 2007]. While examples of 'good' terrain are collected in a similar manner, these approaches simply assume that terrain near where the vehicle was demonstrated driving should be labeled as 'bad'; the assumption is that a classifier will be able to deal with the noisy labeling. In contrast, LEARCH determines a set of states for which the total cost *must* be increased; otherwise the demonstration would have traversed through those states. Essentially, negative examples of where to drive are implied by where the demonstrator explicitly chose not to drive, rather than simply nearby regions that the demonstrator could have driven. Addi-

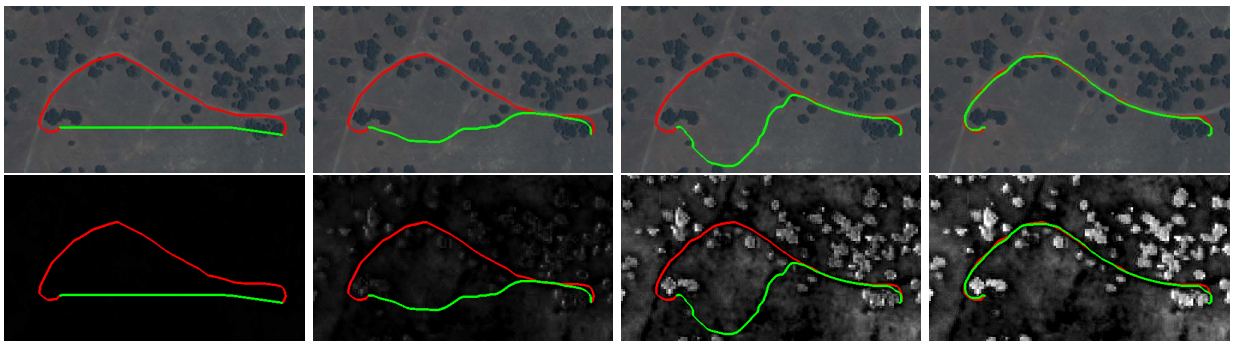


Figure 5: An example of the LEARCH algorithm learning to interpret satellite imagery (Top) as costs (Bottom). Brighter pixels indicate higher cost. As the cost function evolves (left to right), the current plan (green) recreates more and more of the example plan (red). Quickbird imagery courtesy of Digital Globe, Inc. Images cover approximately 300 m X 250 m.

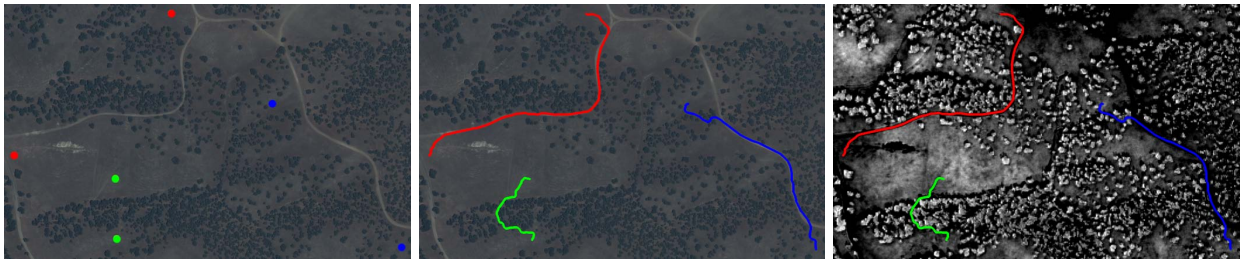


Figure 6: Generalization of the LEARCH algorithm. The cost function learned from the single example in Figure 5 generalizes over terrain never seen during training (shown at approximately 1/2 scale) resulting in similar planner behavior. 3 sets of waypoints (Left) are shown along with the corresponding paths (Center) planned under the learned cost function (Right).

tionally, the terrain that the demonstration traversed is not explicitly considered as 'good' terrain; rather its costs are only lowered until the path is preferred (for specified waypoints); there could still be high cost regions along it. This distinction allows LEARCH to generalize well over areas for which is was not explicitly trained (Figure 6).

4 Adaptation and Application to Mobile Robotic Systems

4.1 Extension to Dynamic and Unknown Environments

The previous derivation of MMP and LEARCH only considered the scenario where the mapping from states to features is static and fully known a priori. Recent work [Silver et al., 2009b] has extended the LEARCH algorithm to the scenario where neither of these assumptions holds, such as when features are generated from a mobile robot's perception system. The limited range inherent in onboard sensing implies a great deal of the environment may be unknown; for truly complex navigation tasks, the distance between waypoints is generally at least one or two orders of magnitude larger than the sensor range. Further, changing range and point of view from environmental structures means that even once an object is within range, its perceptual features are continually changing. Finally, there are the actual dynamics of the environment: objects may move, lighting and weather conditions can change, etc.

Since onboard perceptual inputs are not static, a robot's current plan must be continually recomputed. The original MMP constraint must be altered in the same way: rather than enforcing the optimality of the entire example behavior once, the optimality of all example behavior must be continually enforced as the current plan is recomputed. Formally, we add a time index t to account for dynamics. F_x^t represents the perceptual features of state x at time t. P_e^t represents

the example behavior starting from the current state at time t to the goal, with associated loss function L_e^t. The objective becomes

$$\text{minimize } \mathcal{O}[C] = \lambda \text{REG}(C) + \sum_t \left(\sum_{x \in P_e^t} C(F_x^t) - \min_{\hat{P}^t} \left[\sum_{x \in \hat{P}^t} (C(F_x^t) - L_e^t(x)) \right] \right) \tag{15}$$

the new functional gradient is

$$\bigtriangledown \mathcal{O}_F[C] = \sum_t \left(\sum_{x \in P_e^t} \delta_F(F_x^t) \;\; - \sum_{x \in P_*^t} \delta_F(F_x^t) \right) \tag{16}$$

$$P_*^t = \arg\min_{P^t} \left[\sum_{x \in P^t} (C(F_x^t) - L_e^t(x)) \right]$$

The cost function C does not have a time index: the optimization is searching for the single cost function that best reproduces example behavior over an entire time sequence.

It is important to clarify exactly what P_e^t represents. Until now, the terms *plan* and *behavior* have been interchangeable. This is true in the static case since the environment never evolves; as long as a plan is sufficiently followed, it does not need to be recomputed. However, in the dynamic case, an expert's plan and behavior are different notions: the plan is the currently intended future behavior, and the behavior is the result of previous plans. Therefore, P_e^t would ideally be the expert's *plan* at time t, not example behavior from time t onwards.

However, this information is generally not available: it would require the recording of an expert's instantaneous plan at each point in time. Even if a framework for such a data collection were to be implemented, it would turn the collection of training data into an extremely tedious and expensive process. Therefore, in practice we approximate the current plan of an expert P_e^t with the expert's behavior from t onwards. Unfortunately, this approximation can potentially create situations where the example at certain timesteps is suboptimal or inconsistent. The consequences of this inconsistency and possible solutions are discussed in Section 4.2 (see Figure 9).

Once dynamics have been accounted for, the limited range of onboard sensing can be addressed. At time t, there may be no perceptual features available corresponding to the (potentially large) section of the example path that is outside of current perceptual range. In order to perform long range navigation, a mobile robotic system must already have some approach to planning through terrain it has not directly sensed. Solutions include the use of prior knowledge [Silver et al., 2008], extrapolation from recent experience [Urmson et al., 2006], or simply to assume uniform properties of unknown terrain.

Therefore, we define the set of visible states at time t as $\mathcal{V}^t$. The exact definition of visible depends on the specifics of the underlying robotic system's data fusion: $\mathcal{V}^t$ should include all states for which the cost of state x at time t is computed with the cost function currently being learned, C [6]. For all other states $\bar{\mathcal{V}}^t$, we can assume the existence of some alternate function for computing cost, $C_{\bar{\mathcal{V}}}(x)$; again this could be as simple as a constant.

Since we have explicitly defined $\mathcal{V}^t$ as the set of states at time t for which C is the correct mechanism for computing cost, the cost of a general path P^t is now computed as

$$\sum_{x \in P^t \cap \mathcal{V}^t} C(F_x^t) \;\; + \sum_{x \in P^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x)$$

It is important to note that $C_{\bar{\mathcal{V}}}(x)$ is not dependent on F_x^t. With this new formulation for the cost of a path, the objective functional becomes

$$\begin{aligned} \text{minimize } \mathcal{O}[C] = \; & \lambda \text{REG}(C) \\ + \; & \sum_t \left(\sum_{x \in P_e^t \cap \mathcal{V}^t} C(F_x^t) + \sum_{x \in P_e^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right) \\ - \; & \sum_t \min_{\hat{P}^t} \left(\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) \right. \\ & \left. + \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right) \end{aligned} \tag{17}$$

with functional gradient

$$\bigtriangledown \mathcal{O}_F[C] = \sum_t \left(\sum_{x \in P_e^t \cap \mathcal{V}^t} \delta_F(F_x^t) \;\; - \sum_{x \in P_*^t \cap \mathcal{V}^t} \delta_F(F_x^t) \right) \tag{18}$$

$$P_*^t = \arg\min_{\hat{P}^t} \left[\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) + \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right]$$

[6] In the case of Crusher, $\mathcal{V}^t$ includes all locations that are within current sensor range, and have been observed by said sensors.

since the gradient is computed with respect to C, it is only nonzero for visible states; the $C_{\bar{\mathcal{V}}}(x)$ terms disappear. However, $C_{\bar{\mathcal{V}}}$ still factors into the planned behavior, and therefore does affect the learned component of the cost function. Just as LEARCH learns C to recreate desired behavior when using a specific planner, it learns C to recreate behavior when using a specific $C_{\bar{\mathcal{V}}}$. However, if the example behavior is inconsistent with $C_{\bar{\mathcal{V}}}$, it will be more difficult for the planned behavior to match the example. Such an inconsistency could occur if the expert has different prior knowledge than the robot, or interprets the same knowledge differently. Inconsistency can also occur due to the previously discussed mismatch between expert plans and expert behavior. A solution to this problem is discussed in Section 4.2.

The projection of the functional gradient onto the hypothesis space becomes

$$R_* = \arg\max_R \sum_t \left(\sum_{x \in (P_e \cup P_*) \cap \mathcal{V}^t} \alpha_x^t y_x^t R(F_x^t) \right) \quad (19)$$

Contrasting the final form for R_* with that of (12) helps to summarize the changes that result in the LEARCH algorithm for dynamic environments. Specifically, a single expert demonstration from start to goal is discretized by time, with each timestep serving as an example of what behavior to plan given all data to that point in time. For each of these discretized examples, only visitation counts in visible states are used. The resulting algorithm is presented as Algorithm 3.

A final detail for a LEARCH implementation is the source of the input perceptual features. Rather than computing and logging these features online, it is useful to record all raw sensor data, and then to compute the features by simulating perception offline. This allows existing expert demonstration to be reused if new feature extractors are added, or existing ones modified; perception is simply re-simulated to produce the new inputs. In this way, learning a cost function when the perception system is modified requires no additional expert interaction.

Algorithm 3: The dynamic LEARCH algorithm

Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Cost Map $C_{\bar{\mathcal{V}}}$

$C_0 = 1$;
foreach P_e^i **do**
  for τ =`firstTime`(P_e^i): $\Delta\tau$:`lastTime`(P_e^i) **do**
    $P_e^{\tau,i}$ = `extractPathSeg`(P_e^i, τ,`lastTime`(P_e^i));
    $[\mathcal{F}^{\tau,i}, \mathcal{V}^{\tau,i}]$ = `simPerception`($\mathcal{H}^i$,`firstTime`(P_e^i),τ);
for $j = 1...K$ **do**
  $U +_= U_- = \vec{0}$;
  foreach $P_e^{t,i}$ **do**
    $\mathcal{M}^{t,i}$ = `buildCostmap`($C_{i-1}, \mathcal{F}^{t,i}, \mathcal{V}^{t,i}, C_{\bar{\mathcal{V}}}$);
    $P_*^{t,i}$ = `planLossAugPath`(`start`($P_e^{t,i}$), `goal`($P_e^{t,i}$), $\mathcal{M}^{t,i}$);
    foreach $x \in P_e^{t,i} \bigcap \mathcal{V}^{t,i}$ **do**
      $U_-(\mathcal{F}_x^{t,i}) = U_-(\mathcal{F}_x^{t,i}) + 1$;
    foreach $x \in P_*^{t,i} \bigcap \mathcal{V}^{t,i}$ **do**
      $U_+(\mathcal{F}_x^{t,i}) = U_+(\mathcal{F}_x^{t,i}) + 1$;
  $T_f = T_o = T_w = \emptyset$;
  $U = U_+ - U_-$;
  foreach $\mathcal{F}_x^{t,i}$ *such that* $U(\mathcal{F}_x^{t,i}) \neq 0$ **do**
    $T_f = T_f \bigcup \mathcal{F}_x^{t,i}$;
    $T_o = T_o \bigcup$ `sgn`$(U(\mathcal{F}_x^{t,i}))$;
    $T_w = T_w \bigcup |U(\mathcal{F}_x^{t,i})|$;
  R_j = `trainWeightedRegressor`(T_f, T_o, T_w);
  $C_j = C_{j-1} * e^{\eta_j R_j}$;
return C_K

4.2 Imperfect and Inconsistent Demonstration

The MMP framework implicitly assumes that one or more cost functions exist under which demonstrated behavior is near optimal. Generally this is not the case, as there will always be noise in human demonstration. Further, multiple examples possibly collected from different environments and different experts may be inconsistent with each other (due to inconsistency in human behavior, a different concept of what is desirable, or an incomplete perceptual description of the environment by the robot.) Finally, sometimes experts are flat out wrong, and demonstrate behavior that is not even close to desirable.

While the MMP framework is robust to poor training data, it does suffer degraded overall performance and generalization, in the same way that supervised classification performance is degraded by noisy or mislabeled training data. Attempting to have an expert sanitize the training input after initial demonstration is disadvantageous for two reasons. First it creates an additional need for human involvement, eliminating some of the time savings promised by this approach. Second, it assumes that an expert can detect all errors; while this may be true for extreme cases, a human expert may be no more capa-

Algorithm 4: The dynamic LEARCH algorithm with example replanning and weight balancing

Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Cost Map $C_{\bar{\mathcal{V}}}$, Corridor width β

$C_0 = 1$;
foreach P_e^i **do**
  for τ =`firstTime`(P_e^i): $\Delta\tau$:`lastTime`(P_e^i) **do**
    $P_e^{\tau,i}$ = `extractPathSeg`(P_e^i, τ,`lastTime`(P_e^i));
    $[\mathcal{F}^{\tau,i}, \mathcal{V}^{\tau,i}]$ = `simPerception`($\mathcal{H}^i$,`firstTime`(P_e^i),τ);
for $j = 1...K$ **do**
  $U +_= U_- = \vec{0}$;
  foreach $P_e^{t,i}$ **do**
    $\mathcal{M}^{t,i}$ = `buildCostmap`($C_{i-1}, \mathcal{F}^{t,i}$,$\mathcal{V}^{t,i}$,$C_{\bar{\mathcal{V}}}$);
    $P_*^{t,i}$ = `planLossAugPath`(`start`($P_e^{t,i}$), `goal`($P_e^{t,i}$), $\mathcal{M}^{t,i}$);
    $\mathcal{M}^{t,i}_{\beta,\mathcal{V}^{t,i}}$ = `buildCorridorCostmap`($\mathcal{M}^{t,i}$,β,$\mathcal{V}^{t,i}$);
    $P_{e^*}^{t,i}$ = `replanExample`(`start`($P_e^{t,i}$), `goal`($P_e^{t,i}$), $\mathcal{M}^{t,i}_{\beta,\mathcal{V}^{t,i}}$,);
    foreach $x \in P_{e^*}^{t,i} \bigcap \mathcal{V}^{t,i}$ **do**
      $U_-(\mathcal{F}_x^{t,i}) = U_-(\mathcal{F}_x^{t,i}) + 1$;
    foreach $x \in P_*^{t,i} \bigcap \mathcal{V}^{t,i}$ **do**
      $U_+(\mathcal{F}_x^{t,i}) = U_+(\mathcal{F}_x^{t,i}) + 1$;
  $T_f = T_o = T_w = \emptyset$;
  $U = U_+ - U_-$;
  $N_+ = N_- = 0$;
  foreach $\mathcal{F}_x^{t,i}$ *such that* $U(\mathcal{F}_x^{t,i}) \neq 0$ **do**
    $T_f = T_f \bigcup \mathcal{F}_x^{t,i}$;
    $T_o = T_o \bigcup$ `sgn`$(U(\mathcal{F}_x^{t,i}))$;
    $T_w = T_w \bigcup |U(\mathcal{F}_x^{t,i})|$;
    if `sgn`$(U(\mathcal{F}_x^{t,i})) > 0$ **then** $N_+ = N_+ + 1$ **else** $N_- = N_- + 1$;
  foreach $(t_o, t_w) \in (T_o, T_w)$ **do**
    if $t_o > 0$ **then** $t_w = t_w / N_+$ **else** $t_w = t_w / N_-$;
  R_j = `trainWeightedRegressor`(T_f, T_o, T_w);
  $C_j = C_{j-1} * e^{\eta_j R_j}$;
return C_K

ble of identifying small amounts of noise than he is of preventing that noise in the first place. Even if detecting and filtering out noisy demonstration is automated (as in Section 4.2), removing all imperfect demonstration would remove a large percentage of available training data. This would greatly increase the amount of effort that must be expended to produce a viable training set; it may also remove example demonstrations from which something could still have been learned. Therefore, a practical and robust learning approach must be able to handle a reasonable amount of error in provided demonstration without significantly degraded performance. The rest of this section describes modifications to the LEARCH algorithm that can increase robustness and improve generalization in the face of noisy or poor expert demonstration.

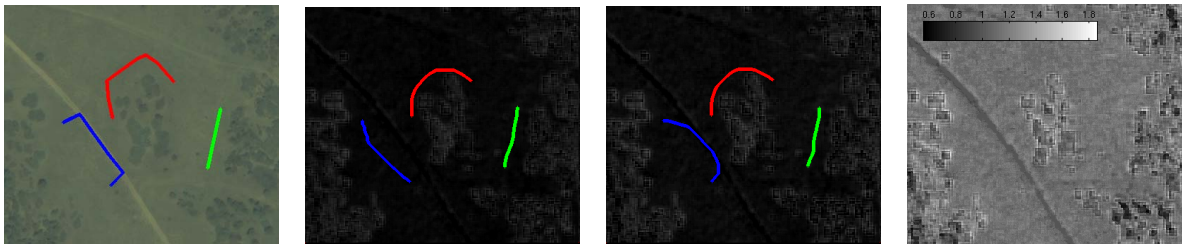
4.2.1 Unachievable Example Behaviors

Experts do not necessarily plan their example behavior in a manner consistent with a robot's planning system: this assumption is not part of the MMP framework. However, what is assumed is that there exists at least one allowable cost function that will cause the robot's planner to reproduce demonstrated behavior (by scoring said behavior as the minimum cost plan). Unfortunately, this is not always the case: it is possible for an example to be *unachievable*. An unachievable example is one such that no consistent cost function, when applied to the available perceptual feature representation, will result in the specified planning system reproducing the example demonstration. For example, an expert may give an inconsistently wide berth to obstacles, or make wider turns than are necessary. Perhaps the most intuitive case is if an expert turns left around a large obstacle, when turning right would have been slightly shorter. The result is that experts often take slightly longer routes through similar terrain than are optimal [Silver et al., 2008]; depending on planner details (such as c-space expansion and dynamic constraints) such examples are often unachievable.

It is instructive to observe what happens to the functional gradient with an unachievable example. Imagine a section of an environment where all states are described by the identical feature vector F'. Under this scenario, (10) reduces to

$$\nabla \mathcal{O}_{F'}[C] = \begin{cases} \sum_{x \in P_e} 1 \;-\; \sum_{x \in P_*} 1 & \text{if } F = F' \\ 0 & \text{if } F \neq F' \end{cases}$$

The functional gradient depends only on the lengths of the example and current plan, independent of the cost function. If the paths are not of equal length, then the optimization will never be satisfied. Specifically, if the example path is too long, there will always be an extra component of the gradient that at-



(a) 3 Example Training Paths (b) Learned costmap with unbalanced weighting (c) Learned costmap with balanced weighting (d) Ratio of the balanced to unbalanced costmaps

Figure 7: The red path is an unachievable example path, as it will be less expensive under any cost function to cut more directly across the grass. With standard unbalanced weighting (b), the unachievable example forces down the cost of grass, and prevents the blue example from being achieved. Balanced weighting (c) prevents this bias, and the blue example is achieved. Overall, grass is approximately 12% higher cost with balanced than unbalanced weighting (d)

tempts to lower costs at F'. Intuitively, an unachievable example implies that the cost of certain terrain should be 0, as this would result in any path through that region being optimal. However, since costs are constrained to $\mathbb{R}^+$, this will never be achieved. Instead an unachievable example will have the effect of unnecessarily lowering costs over a large section of the feature space, and artificially reducing dynamic range. Depending on the expressiveness of $\mathcal{R}$, an unachievable example counteracts the constraints of other (achievable) paths, resulting in poorer performance and generalization (see Figure 7).

This negative effect can be avoided by performing a slightly different regression or classification when projecting the gradient. Instead of minimizing the weighted error, the *balanced* weighted error is minimized; that is, both positive and negative targets make an equal contribution. Formally, in (19) R_* is replaced with R_*^B defined as

$$R_*^B = \arg\max_R \sum_t \left(\sum_{y_x^t>0} \frac{\alpha_x^t R(F_x^t)}{N_+} - \sum_{y_x^t<0} \frac{\alpha_x^t R(F_x^t)}{N_-} \right)$$
$$N_+ = \sum_t \sum_{y_x^t>0} \alpha_x^t = |U_+|_1 \qquad N_- = \sum_t \sum_{y_x^t<0} \alpha_x^t = |U_-|_1 \tag{20}$$

In the extreme unachievable case described above, R_*^B will be zero everywhere; the optimization will be satisfied with the cost function as is. The effect of balancing in the general case can be observed by rewriting the regression operation in terms of the planned and example visitation counts, and observing the correlation of their inputs.

$$R_* = \arg\max \langle R, U_+ - U_- \rangle$$
$$R_*^B = \arg\max \langle R, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle$$

Theorem 4.1. *The regression targets of R_* and R_*^B are always correlated, except when the visitation counts between the example and planned path are perfectly correlated.*

Proof.

$$\begin{aligned}
&\langle U_+ - U_-, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle \\
&= \frac{\langle U_+, U_+ \rangle}{N_+} - \frac{\langle U_+, U_- \rangle}{N_+} + \frac{\langle U_-, U_- \rangle}{N_-} - \frac{\langle U_+, U_- \rangle}{N_-} \\
&= \frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-}) \langle U_+, U_- \rangle
\end{aligned} \tag{21}$$

By the Cauchy-Schwarz inequality, $\langle U_+, U_- \rangle$ is bounded by $|U_+||U_-|$, and is only tight against this bound when the visitation counts are perfectly correlated, which implies

$$\begin{aligned}
\langle U_+, U_- \rangle = |U_+||U_-| &\iff U_- = \kappa U_+ \\
&\implies |U_-| = \kappa |U_+| \ , \ N_- = \kappa N_+
\end{aligned}$$

for some scalar κ. By substitution

$$\begin{aligned}
&\frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-}) \langle U_+, U_- \rangle \\
&\geq \frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-}) |U_+||U_-| \\
&= \frac{|U_+|^2}{N_+} + \frac{\kappa^2 |U_+|^2}{\kappa N_+} - (\frac{1}{N_+} + \frac{1}{\kappa N_+}) \kappa |U_+||U_+| \\
&= \frac{|U_+|^2}{N_+} + \frac{\kappa |U_+|^2}{N_+} - \frac{|U_+|^2}{N_+} - \frac{\kappa |U_+|^2}{N_+} \\
&= 0
\end{aligned}$$

When $\langle U_+, U_- \rangle$ is not tight against the upper bound

$$\langle U_+ - U_-, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle \geq 0$$

□

By (21) the similarity between inputs to the projections is negatively correlated to the overlap of the positive and negative visitation counts. When there exists clear differentiation between what features should have their costs increased and decreased, the projection inputs will be similar. As the example and current planned behaviors travel over increasingly similar terrain, the inputs being to diverge; the contribution of the balanced projection to the current cost function will level out, while that of the unbalanced projection will increase in the direction of the longer path. Finally, in a fully unachievable case, the balanced projection will zero out, while the unbalanced would drive the cost in the direction of the more dominant class. This effect is observed empirically in Section 5. The implementation of this balancing is shown in Algorithm 4.

4.2.2 Noisy Demonstration: Replanning and Corridor Constraints

A balanced regression can help to account for large scale sub-optimality in human demonstration. However, sub-optimality can also occur at a smaller scale. It is unreasonable to ever expect a human to drive or demonstrate the exact perfect path; it is often the case that a plan that travels through neighboring or nearby states would be a slightly better example. In some cases this example noise translates to noise in the cost function; in more extreme cases it can significantly affect performance (Figure 8). What is needed is an approach that smoothes out small scale noise in expert demonstration, producing a better training example.

Such a smoothed example can be derived from expert demonstration by redefining the MMP constraint: instead of example behavior being interpreted as the exact optimal behavior, it can be interpreted as a behavior that is spatially near the optimal path. The exact definition of 'close' depends on the state space; the loss function will always provide at least one possible metric. If the state space is $\mathbb{R}^n$, then Euclidean distance is a natural metric. Therefore, rather than an example defining the exact optimal path, it would define a corridor in which the optimal path exists.

Redefining the original MMP constraint in (4) in this way yields

$$\begin{aligned} &\text{minimize } \mathcal{O}[C] = \lambda \textsc{Reg}[C] \\ &\text{subject to the constraint} \\ &\sum_{x \in P_*} (C(F_x) - L_e(x)) \geq \sum_{x \in P_e^*} C(F_x) \\ &P_* = \arg\min_P \sum_{x \in P} (C(F_x) - L_e(x)) \\ &P_e^* = \arg\min_{P \in \mathcal{N}_e} \sum_{x \ in P} C(F_x) \end{aligned} \tag{22}$$

Instead of enforcing that P_e is optimal, the new constraint is to enforce that P_e^* is optimal, where P_e^* is the optimal path within some set of paths $\mathcal{N}_e$. The definition of $\mathcal{N}_e$ determines how 'close' is defined. Using the above example of a corridor in a Euclidean space, $\mathcal{N}_e$ would be defined as

$$\mathcal{N}_e = \{P \mid \forall x \in P \;\; \exists y \in P_e \quad s.t. \; ||x - y|| \leq \beta\}$$

with β defining the corridor width. In the general case, this definition can always be rewritten in terms of the loss function

$$\mathcal{N}_e = \{P \mid \forall x \in P \;\; \exists y \in P_e \quad s.t. \; L(x, y) \leq \beta\}$$

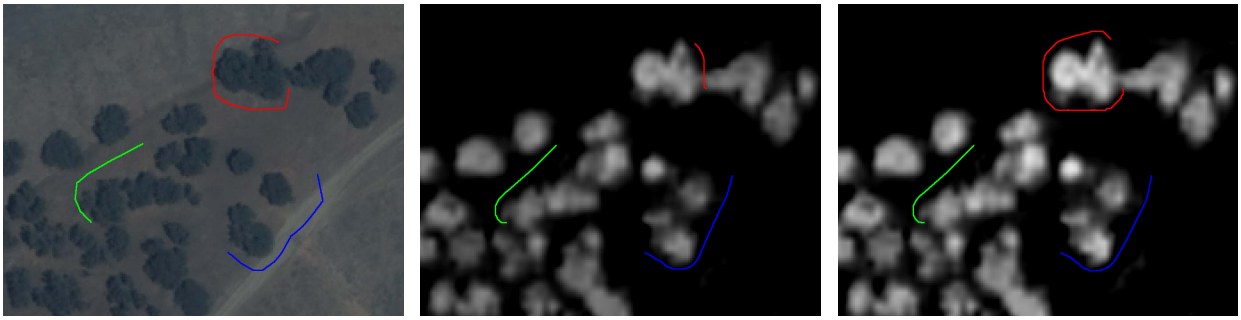
It is important to note that the definition of N_e is only dependent on individual states. Therefore, P_e^* can be found by an optimal planner, simply by only allowing traversal through states that meet the loss threshold β with respect to some state in P_e.

Reformulating (22) as an optimization problem yields the following objective

$$\begin{aligned} \text{minimize } \mathcal{O}[C] = \lambda \textsc{Reg}(C) &\\ + \min_{\hat{P}_e \in \mathcal{N}_e} &\left[\sum_{x \in \hat{P}_e} C(F_x) \right] \\ - \min_{\hat{P}} &\left[\sum_{x \in \hat{P}} (C(F_x) - L_e(x)) \right] \end{aligned} \tag{23}$$

The resulting change in the LEARCH algorithm is to carry through the extra minimization to the computation of the visitation counts. That is, at every iteration, a new, smoothed, example is chosen from with $\mathcal{N}_e$; example visitation counts are computed with respect to this path. This new step is shown in Algorithm 4.

It should be noted that as a result of this additional min term, the objective is no longer convex. It is certainly possible to produce individual examples where such a smoothing step can result in poor



(a) Example Paths (b) Planned Paths (No Replanning) (c) Planned Paths (With Replanning)

Figure 8: An example of how noisy demonstration can hurt performance. The red and green example paths in (a) are slightly too close to trees, preventing the cost of the trees from increasing sufficiently to match the red example (map (b)). However, if the paths are allowed to be replanned within a corridor, the red and green path are essentially smoothed, allowing the cost of trees to get sufficiently high (map (c)). On average, the trees achieve three times the cost in (c) as in (b).

local minima; however, it has been observed empirically that this effect is neutralized when using multiple examples. The experimental performance of this smoothing step is presented in Section 5.

When operating with dynamic and partially unknown perceptual data, this replanning step provides another important side effect. Rewriting (17) with this additional min term yields

$$\text{minimize } \mathcal{O}[C] = \lambda \text{REG}(C) \tag{24}$$
$$+ \sum_t \min_{\hat{P}_e^t \in \mathcal{N}_e^t} \left(\sum_{x \in \hat{P}_e^t \cap \mathcal{V}^t} C(F_x^t) + \sum_{x \in \hat{P}_e^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right)$$
$$- \sum_t \min_{\hat{P}^t} \left(\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) + \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right)$$

Where $\mathcal{N}_e^t$ is the set of paths near P_e^t. However, as before it should be noted that the $C_{\bar{\mathcal{V}}}$ terms will have no affect on the functional gradient. Therefore, the definition of $\mathcal{N}_e^t$ does not need to consider states in $\bar{\mathcal{V}}$. This yields the following general definition of $\mathcal{N}_e^t$

$$\mathcal{N}_e^t = \{P^t \mid \forall x \in P^t \bigcap \mathcal{V}^t \;\; \exists y \in P_e^t \bigcap \mathcal{V}^t \;\; s.t.\; L(x,y) \leq \beta\}$$

the result is that $\mathcal{N}_e^t$ only defines closeness over $\mathcal{V}^t$. Behavior outside of $\mathcal{V}^t$ does not directly affect the gradient, but does affect the objective value (the difference in cost between the current (replanned) example and planned behavior). Therefore, by performing a replanning step (even with $\beta = 0$), example behavior can be made consistent with $C_{\bar{\mathcal{V}}}$ without compromising its effectiveness as an example within $\mathcal{V}^t$. This notion of consistency proves to have meaningful value.

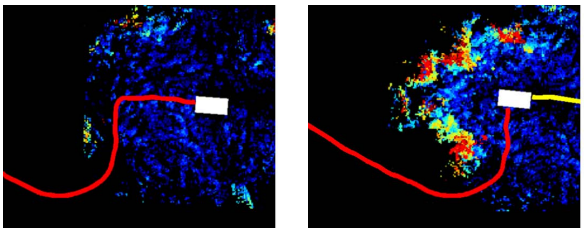


Figure 9: Recorded example behavior from time t (left) and $t+1$ (right), overlayed on a single perceptual feature (obstacle height). Future behavior is inconsistent at time t, but makes sense at time $t+1$ given additional perceptual data.

4.2.3 Filtering for Inconsistent Examples

One fundamental issue with expert demonstration is consistency. A human demonstrator may act approximately according to one metric during one example, and a slightly different metric during another example. While each individual example may be near-optimal with respect to some metric, the two examples together may be inconsistent; that is, there is no consistent cost function that would define both behaviors as optimal.

The possibility of an expert interpreting unknown terrain in a different manner is a potentially large source of inconsistency. This is especially true when attempting to learn an online cost function, as it is very likely that the demonstrator will have implicit prior knowledge of the environment that is unavailable to the perception system. However, by always performing a replanning step as previously discussed, example demonstration can be made consistent with

the robot's interpretation of the environment in unobserved regions.

With consistency in unobserved regions accounted for, there remain four primary sources of inconsistent demonstration

- Inconsistency between multiple experts
- Expert error (poor demonstration)
- Inconsistency between an expert's and the robot's perception in observed regions
- A mismatch between an expert's planned and actual behavior

This last issue was alluded to in Section 4.1: while an expert example should consist of an expert's *plan* at time t from the current state to the goal, what is recorded is the expert's *behavior* from time t to the goal. Figure 9 provides a simple example of this mismatch: at time t, the expert likely planned to drive straight, but was forced to replan at time $t+1$ when the cul-de-sac was observed. This breaks the assumption that the expert behavior from time t onward matches the expert plan; the result is that the discretized example at time t is inconsistent with other example timesteps.

However, the very inconsistency of such timesteps provides a basis for their filtering and removal. Specifically, it can be assumed that a human expert will plan in a fairly consistent manner during a single example traverse[7]. If the behavior from a single timestep or small set of timesteps is inconsistent with the demonstrated behavior at other timesteps, then it is safe to assume that this small set of timesteps does not demonstrate correct behavior, and can be filtered out and removed as training examples. This does not significantly affect the amount of training data required to train a full system., as by definition an inconsistent timestep is unlikely to provide an example of an important concept.

Inconsistency can be quantitatively defined by observing each timestep's contribution to the objective functional (its slack penalty). In (5) this penalty is explicitly defined as a a measurement of by how much a constraint remains violated. If the penalty at a single timestep of an example behavior is a statistical outlier from the distribution of slack penalties at all other timesteps, it indicates that single timestep implies constraints that remain violated far more than others. That is, the constraints at an outlier timestep are inconsistent with those implied by the rest of a demonstration.

[7]If this assumption does not hold, then the very idea of learning from said expert's demonstration is flawed

Therefore, the following filtering heuristic is proposed as a pre-processing step. First, attempt to learn a cost function over all timesteps of a single example behavior and identify statistical outliers (according to slack penalties). During this step, a more complex hypothesis space of cost functions should be used than is intended for the final cost function (i.e use more complex regressors). As these outlier timesteps are inconsistent within an overly complex hypothesis space, there is evidence that the inconsistency is in the example itself, and not for lack of expressiveness in the cost function. Therefore, these timesteps should be removed. This process can be repeated for each example behavior, with only remaining timesteps used in the final training.

Aside from filtering out inconsistency due to plan/behavior mismatch, this approach will also filter timesteps due to other sources of inconsistency. This is beneficial, as long as the timesteps truly are inconsistent. However, the possibility always remains that the example itself was correct; it may only appear inconsistent due to the fidelity of perception or planning. In this case, filtering is still beneficial, as the examples would not have been learnable (with the current set of perceptual features and the current planning system); instead, the small subset of filtered examples can be examined by a system expert, who may then identify a necessary additional component level capability. Experimental results of this filtering approach are presented in Section 5.

4.3 Application to Mobile Robotic Systems

Before LEARCH (in either its static or dynamic forms) can be applied to the task of learning a terrain cost function for a mobile robotic system, there are still some practical considerations to address. It is important to remember the specific task for which LEARCH is intended: it is designed to select a cost function from a defined hypothesis space $\mathcal{C}$, such that expert demonstration is recreated when the cost function is applied to the *specific* perception and planning systems for which it was trained. There are several hidden challenges in that statement, such as defining $\mathcal{C}$, and ensuring LEARCH is producing a cost function for the correct planning system.

4.3.1 Selecting a Cost Function Hypothesis Space

The cost function hypothesis space $\mathcal{C}$ is implicitly defined by the regressor space $\mathcal{R}$. In turn, $\mathcal{R}$ is defined by design choices relating to the family and allowable

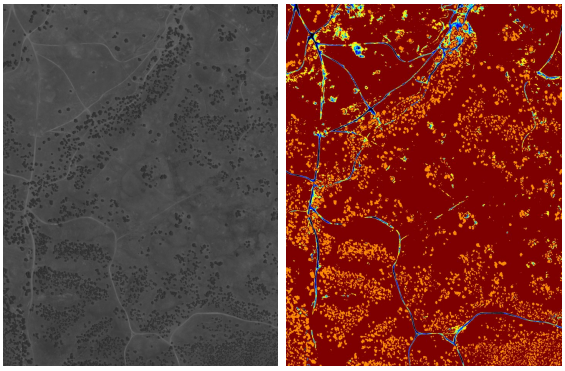


Figure 10: Example of a new feature (right) learned automatically from panchromatic imagery (left) using only expert demonstration (there are no explicit class labels).

complexity of regressors. For example, if single layer neural networks with at most H hidden are chosen as the class of regressors, than $\mathcal{C}$ consists of all cost functions that are a weighted sum of such networks. In this way, cost functions of almost arbitrary complexity can be allowed.

However, as with most machine learning techniques, there is a design tradeoff between expressiveness and generalization. Complex regressors are capable of expressing complex costing rules, but are more prone to overfitting. In contrast, simpler regressors may generalize better, but are also limited in the costing rules they can express. This tradeoff must be effectively balanced to ensure both sufficient expressiveness and generalization. Fortunately, as regression is a core machine learning task, there are well developed and understood approaches for achieving this balance. In practice, validation with an independent holdout set of demonstrated behaviors can help quantify this tradeoff. This allows a range of regressor types and complexities to be automatically evaluated; the one with the best holdout performance can be selected with minimal additional human interaction.

Another issue with respect to the definition of $\mathcal{C}$ is computational cost. In a scenario where perceptual features are static, this concern is not as important, as cost evaluations are only performed once. However, in an online and dynamic setting, cost evaluations are performed continuously; computational complexity may be a much larger issue. As the final learned cost function is a weighted combination of K regressors, computational complexity of the final cost function is linear in K. Again, this creates a design tradeoff; limiting K will limit the computational cost per evaluation, but will also limit the accuracy of the cost function (as the final steps of a gradient descent operation fine tune the solution).

One solution is to define $\mathcal{R}$ as the space of linear functions. Since the weighted sum of linear functions is another linear function, using this definition of $\mathcal{R}$ would result in the final complexity of C_K being constant with respect to K. However using linear regressors with LEARCH results in almost the same solution as would be produced by the linear MMP algorithm (but not identical due to exponentiated functional gradient descent). As the fundamental advantage of the LEARCH approach was to allow non-linear cost functions, this would seem to imply a necessary tradeoff. However, the addition of a feature learning phase [Ratliff et al., 2007, Silver et al., 2008, Silver et al., 2009a] to LEARCH can potentially solve this problem. During most learning iterations, linear regressors are used. However, when it is detected that the objective error is no longer decreasing, a single, simple non-linear regression is performed. This single, non-linear step would better discriminate between terrains that are difficult for a purely linear regressor. However, rather than add this new regressor directly into the cost function, it is instead treated as a new feature. Figure 10 provides a simple example of a learned feature with only greyscale imagery as input.

4.3.2 Planner Interpretation of Cost Maps

A common approach in mobile robotics is to treat the environment as a 2.5D space; this allows the state space $\mathcal{S}$ for high level path planning to be $\mathbb{R}^2$. This results in terrain costs defined over a a discretized 2D grid. However, different planning systems may interpret a 2D cost grid differently. Since the goal is to learn a cost function to recreate behavior with a specific planning system, these details must be taken into account to learn the cost function correctly.

Perhaps the simplest cost-aware planner one might use for a mobile robot would be 4-connected A* (or other grid planner). Such a planning system would incur a cost of $C(x)$ whenever a cell x was traversed. Now consider the case of an 8-connected A*. Many 8-connected implementations treat traversing a cell diagonally as higher cost than traversing the same cell axis-aligned; this allows the planner to take into account the extra distance traversed through a grid. This usually takes the form of incurring a cost of $C(x)$ when traversing the cell axis-aligned, and a cost of $\sqrt{2}C(x)$ when traversing the cell diagonally.

Since cells traversed diagonally incur more cost, this must be taken into account by LEARCH when computing the projection of the functional gradient.

With respect to the final cost of a path, a cell traversed diagonally will have a $\sqrt{2}$ greater affect than one traversed axis-aligned; therefore it is $\sqrt{2}$ times more important to get the sign right on the projection. The solution is to increment the visitation count of a cell by 1 when traversing it axis-aligned, and by $\sqrt{2}$ when traversing diagonally. In the general case, a planner may incur a cost of $dC(x)$ when traversing a distance d through cell x; the visitation count of state x should then be incremented by d.

Another issue that must be considered is that of configuration space expansion. Motion planners for mobile robotic systems often apply a c-space expansion to an input cost map before generating a plan, in order to account for the physical dimensions of the robot. The result is that the cost the planner assigns for traversing distance d through state x is no longer $dC(x)$, but rather something along the lines of

$$d \sum_{y \in \mathcal{N}} W(x, y) C(y) \tag{25}$$

where $\mathcal{N}$ is a set of states sufficiently close to x, and W is a weighting function. Common choices for W include a constant, or a linear falloff based on $||x - y||$. As before, this weighting must be captured by the visitation counts: if distance d is traversed through cell x, then all cells $y \in \mathcal{N}$ must have there visitation counts incremented by $dW(x, y)$. A further complication arises if W depends not only on the locations of x and y, but also their (or other states) cost values. For instance, if a c-space expansion defined the cost of traversing d through state x as

$$d \max_{y \in \mathcal{N}} C(y)$$

then only the state y with the maximum cost in $\mathcal{N}$ should have its visitation count incremented by d; all other states would not affect the planner perceived cost of traversing x under the current C. Unlike (25), this form results in non-convexity in the LEARCH algorithm (in addition to non-convexity from replanning), as different initial cost functions C_0 may produce significantly different results.

4.3.3 Planners with Motion Constraints

A common architecture for mobile robot planning systems is to utilize a hierarchy of planners, ranging from long-range low resolution planners to short-range high resolution planners. The simplest form of this architecture utilizes a long-range, unconstrained 'global' planner, and a short-range, kinematically or dynamically constrained 'local' planner [Kelly et al., 2006, Stentz et al., 2007]. Usually, a local planner does not plan all the way to the goal; instead it produces a set of feasible short-range actions, and utilizes the global planner to produce a path from the end of the action to the goal. Local planner generated plans are not actually followed to completion, but instead are replanned at a high rate. In this way, the local planner is not actually optimal, but instead performs something akin to a greedy search.

If LEARCH is to be used for an onboard perception system, it is important that they be learned with respect to the planning system that will be used onboard the robot. If a hybrid architecture is used, LEARCH must therefore be implemented using the same configuration. It is important that the decisions the planner is tasked with making during training are exactly those that will be required during operation. For example, in the case of a greedy local planning system, P_*^t at each iteration should be the direct output of the planning system, not the concatenation of multiple planning cycles (even though this is how the robot's behavior is determined online). This is necessary for credit assignment; if at time t the expert swerved to avoid an obstacle, then not only must the cost of the obstacle be sufficiently high, but the cost of the terrain swerved over must be sufficiently low. Even if subsequent actions reduce the distance traveled during the swerve, the planner must be willing to perform a large turn at time t to begin that maneuver.

Related to this issue is one of planner fidelity. Depending on discretization resolution and the level of vehicle modeling, it is unlikely that a kinematically or dynamically constrained planning system will be able to exactly recreate the same behavior as executed during demonstration. Inability to recreate demonstrated behavior can introduce noise into the learned cost function, of magnitude proportional to the degree of mismatch. Possible solutions for this issue are discussed in the next two sections.

5 Experimental Results

Learning cost functions by demonstration was applied to the Crusher autonomous system (Figure 1). The Crusher vehicle is capable of traversing rough, complex, and unstructured terrain; as such understanding the relative benefits and tradeoffs of various terrain types is of paramount importance to its autonomous operation. On Crusher, terrain data comes from two primary sources. The first is prior, static, overhead data sources (satellite or aerial imagery, aerial LiDAR, etc.). Overhead data is processed via a set

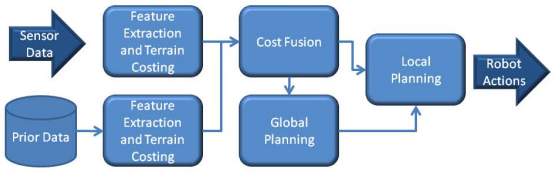


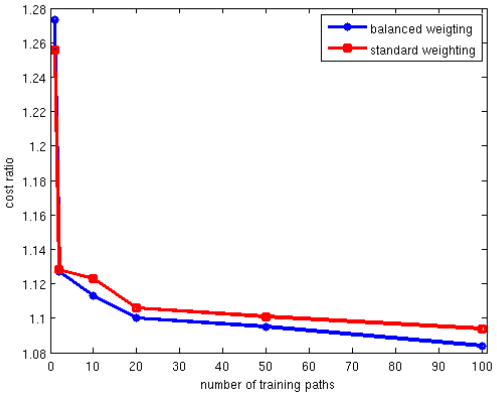
Figure 11: A high level block diagram of the Crusher Autonomy system

of engineered feature extractors into a set of feature maps, which are then mapped into a single, static costmap for an entire area of operation. The second source of terrain data comes from the onboard perception system. The onboard perception system processes local data from onboard camera images and LiDAR into a dynamic stream of features. At a high data rate, these features are continuously mapped to costmaps over a local area.

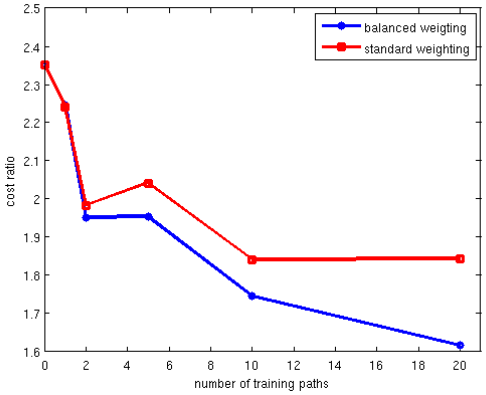
Costs from both sources are continuously fused into a single consistent costmap, which is then passed to Crusher's motion planning system. Fusing prior and onboard perceptual data at the cost level allows for Crusher to continuously plan a path all the way from its current position to the goal. Due to the dynamic nature of this cost data, the Field D* algorithm is utilized [Ferguson and Stentz, 2006]. In order to determine feasible local motion commands for Crusher, a variant of the RANGER system [Kelly, 1995, Kelly et al., 2006] is applied, utilizing the Field D* plan for global guidance. This architecture is shown is Figure 11.

Early in Crusher's development, the task of interpreting both static and dynamic perceptual data into costs was performed through manual engineering, and was a large source of frustration. This led to the application of learning by demonstration to constructing cost functions. This was first applied in the static perceptual case (overhead data) utilizing the Field D* planner, and was next applied to the dynamic case (onboard perceptual data) utilizing the Field D* guided RANGER local planner. The remainder of this section describes these experiments, along with offline results from each task.

Two metrics are used for evaluating offline performance. The first is the average loss along a path. As the loss function is constructed to encode the similarity between two paths, the loss between an example path and the corresponding planned path (over the interval [0,1]) is a measure of how accurately expert behavior has been reproduced. A second measure is the cost ratio, defined as the cost of an example path divided by the cost of the corresponding planned path. As this ratio approaches 1, it indicates the current cost function hypothesis is approaching consistency with the expert demonstration. The cost ratio as opposed to cost difference is used to account for the affect of scaling on the cost function (with the cost difference, simply scaling the costs closer to zero would improve the metric, without improving the relative cost difference).



(a) Simulated Examples



(b) Expert Drawn Examples

Figure 12: Learning simulated and expert drawn example paths. Test set performance is shown as a function of number of input paths

5.1 Learning to Interpret Overhead Data

In order to verify LEARCH under ideal conditions, tests were first run on simulated examples. A known (arbitrary) cost function was used to generate a cost map over a single environment from its overhead features; this cost map was used to produce paths between random waypoints. Different numbers of these paths were then used as input for LEARCH, and the performance measured on a large independent vali-

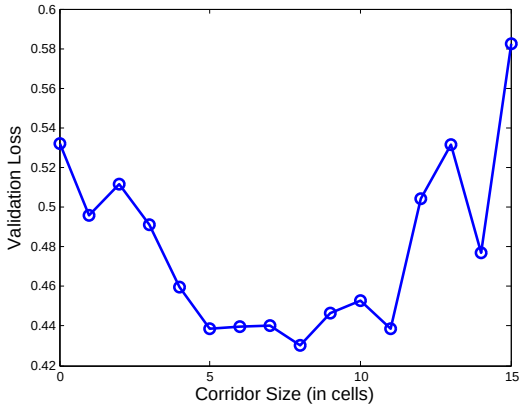


Figure 13: Validation loss as a function of the corridor size. Using corridor constraints improves performance as long as the corridor is not too large

dation set of paths (generated in the same manner).

Figure 12(a) shows the results using both the balanced and standard weighting schemes (Section 4.2.1). As the number of training paths is increased, the test set performance continues to improve. Each input path further constrains the space of possible cost functions, bringing the learned function closer to the desired one. However, there are diminishing returns as additional paths overlap to some degree in their constraints. Finally, the performance of the balanced and standard weighting schemes is similar. Since all paths for this experiment were generated by a planner, they are by definition optimal under some metric, and therefore both achievable and consistent with each other.

Next, experiments were performed with expert examples (both training and validation) drawn on top of overhead data maps. Figure 12(b) shows the results of an experiment of the same form as that performed with simulated examples. Again, the validation set cost ratio decreases as the number of training examples increases. However, with real examples there is a significant difference between the two weighting schemes; the balanced weighting scheme achieved significantly better performance. This demonstrates both how human demonstration can suffer from large scale non-optimality, and how LEARCH can be made robust to this fact through a balanced regression.

Another series of experiments were performed to determine the effect of performing replanning with corridor constraints (Section 4.2.2). For these experiments, the performance of learning was measured with validation loss, to indicate how well the planned paths matched the examples. When measuring loss on the validation set, no replanning was performed. Therefore, in order to provide a smoother metric, the specific loss function used was a radial basis function between states on the current path P_* and the closest state on the example P_e, with a scale parameter σ^2

$$L(P_*, P_e) = \frac{1}{|P_*|} \sum_{x \in P_*} [1 - \exp(\min_{x_i \in P_e} [\|x - x_i\|^2]/\sigma^2)]$$

Using a loss function of this form provides a more analog metric than a Hamming style loss as previously described. Figure 13 shows the results on the validation set as a function of the corridor size (in cells). Small corridors provide an improvement over no corridor, demonstrating how small scale smoothing can improve generalization. However, as the corridor gets too large, this improvement disappears; large corridors essentially over-smooth the examples and begin to miss critical information.

Finally, experiments were performed in order to compare the offline performance of learned costmaps with engineered ones. A cost map was trained off of satellite imagery for an approximately 60 km^2 size environment. An engineered costmap had been previously produced for this same test site to support Crusher operations. This engineered map was produced by performing a supervised classification of the imagery, and then manually determining a cost for each class [Silver et al., 2006]. A subset of both maps is shown in Figure 15. The two maps were compared using a validation set of paths generated by a Crusher team member not directly involved in the development of overhead costing. The average validation loss using the LEARCH map was 23% less than the engineered map (Figure 14), thus demonstrating superior generalization of the learned approach.

Online validation of the learned costmaps was also achieved during Crusher field tests. These tests consisted of Crusher autonomously navigating a series of courses, with each course defined as a set of widely spaced waypoints. Courses ranged in length up to 20 km, with waypoint spacing on the order of 200 to 1000 m. These tests took place at numerous locations across the continental U.S., each with highly varying local terrain characteristics, and sizes ranging from tens to hundreds of square kilometers.

During 2005 and 2006, prior maps were primarily generated as described in [Silver et al., 2006]. An initial implementation of the LEARCH algorithm was also field tested and demonstrated in smaller tests during 2006. During 2007 and 2008, LEARCH became the default approach for producing cost maps. Overall, LEARCH maps were used during more than 600 km of sponsor monitored autonomous traverse, plus hundreds of kilometers more of additional field

Experiment	Total Net Distance(km)	Avg. Speed(m/s)	Total Cost Incurred	Max Cost Incurred
Experiment 1 Learned	6.63	2.59	11108	23.6
Experiment 1 Engineered	6.49	2.38	14385	264.5
Experiment 2 Learned	6.01	2.32	17942	100.2
Experiment 2 Engineered	5.81	2.23	21220	517.9
Experiment 2 No Prior	6.19	1.65	26693	224.9

Table 1: Results of experiments comparing learned to engineered prior maps. Indicated costs are from the vehicle's onboard perception system.

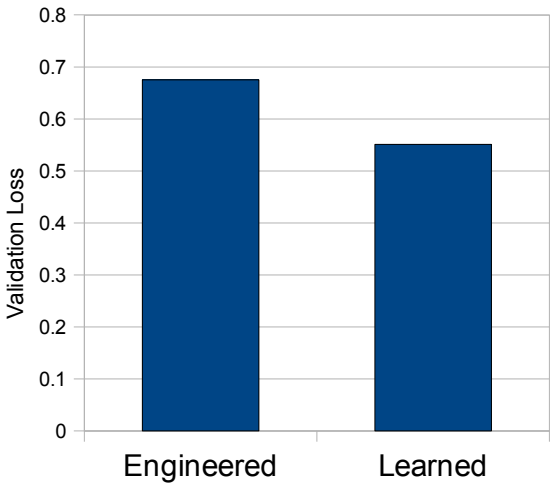


Figure 14: Performance comparison between a learned and engineered prior costmap. The learned map produced behavior that better matched an independent validation set of examples.

testing. This demonstrates that a learned cost function was sufficient for use online a complex robotic system.

In addition, two direct online comparisons were performed. These two tests were performed several months apart, at different test sites. During each experiment, the same course was run twice, varying only the prior cost map given to the vehicle between runs. The purpose of these experiments was to demonstrate that learning a cost function not only generalized better with respect to initial route planning, but also with respect to dynamic replanning online.

Each run was scored according to the total cost incurred by the vehicle according to its onboard perception system. At the time of these experiments, the perception system made use of a manually engineered cost function. However, this function was shown through numerous experiments [Stentz et al., 2007] to result in a high level of autonomous performance; therefore it is a valid metric for scoring the safety of a single autonomous run.

The results of these experiments are shown in Table 1. In both experiments, the vehicle traveled farther to complete the same course using learned prior data, and yet incurred less total (online) cost. Over both experiments, with each waypoint to waypoint section considered an independent trial, the improvement in average cost and average speed is statistically significant at the 5% and 10% levels, respectively. This indicates that the terrain the vehicle traversed was on average safer when using the learned prior map, according to its own onboard perception system. This normalization by distance traveled is necessary because the learned prior and engineered perception cost functions do not necessarily agree in what they consider unit cost. Additionally, the maximum cost incurred at any point along an experiment is also provided; for both terrains, the maximum is significantly lower when using the learned prior data. The course for Experiment 2 was also run without any prior data; the results are presented for comparison.

In addition to improving vehicle performance, using learned cost functions also reduced the necessary amount of human interaction. When preparing for a Crusher test using engineered costmaps, performing a supervised classification and tuning the cost function would take on the order of 1-2 days. In contrast, when using learned costmaps drawing example paths

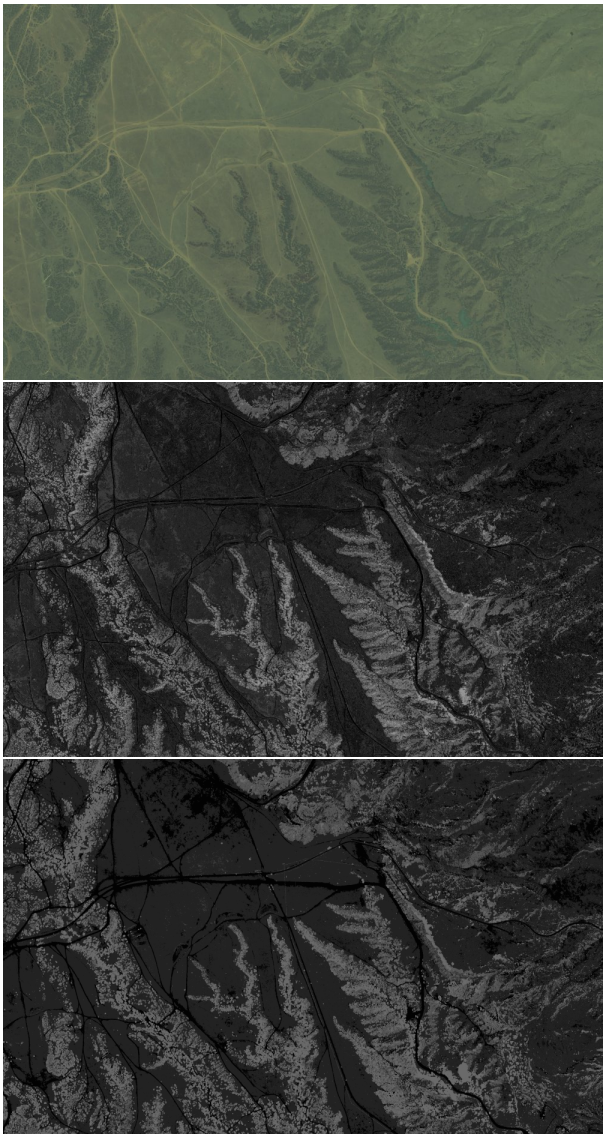


Figure 15: A 10 km^2 section of a Crusher test site. From top to bottom: Quickbird imagery, Learned Cost, and Engineered Cost

would require on the order of 1-2 hours[8]. In a timed head-to-head 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. On this same experiment, the learned costmap had a validation loss of 0.43, compared to 0.56 for the engineered map. This demonstrates that the learned approach

[8]Neither of these timings include the necessary effort to process raw overhead data into feature maps, as this process is a shared precursor to both approaches to costing

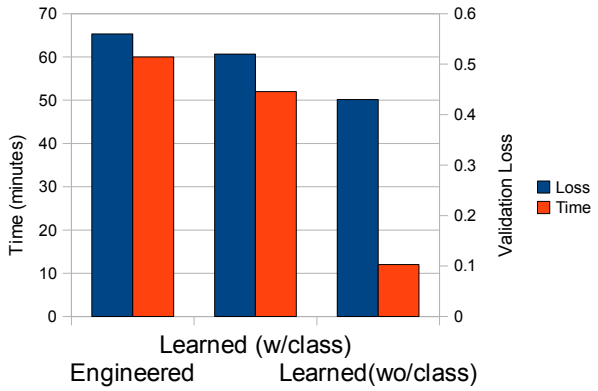


Figure 16: Performance comparison between 3 approaches to generating prior costmaps. LEARCH not only performs better and faster at the task of determining costs of different semantic classes, it also does a better job at interpreting raw data (for the purpose of costing) than semantic classification.

produced superior performance, with less human interaction time (Figure 16).

An additional test was performed in which the same training set of example paths was used to learn a cost function only from the results of the supervised classification; in this case the learned map had a validation loss of 0.52. This demonstrates two important points. The first is that even when the problem of learning a cost function was reduced to solely a low dimensional parameter tuning problem (in this case 5 dimensions), the automated approach was able to perform better than manual tuning, and with less human interaction. The second point is that reducing the task to a lower dimensional problem (labeling for the supervised classification) required additional interaction, and that this feature space compression resulted in a loss of useful information (as the validation loss was better when learning from the full feature space as opposed to the compressed one).

5.2 Learning to Interpret Online Perceptual Data

Next, dynamic LEARCH was applied to the task of learning a cost function for Crusher's onboard perception system (Figure 17). Training data in the form of expert example behaviors was gathered by having Crusher's safety operator RC the vehicle through sets of waypoints. Different training examples were collected over a period of months in varying locations and weather conditions, and with 3 different operators at one time or another. During data collec-

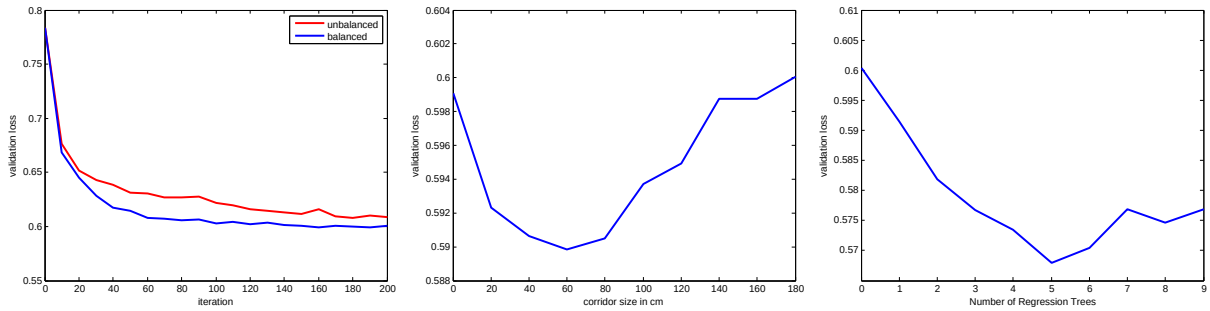


Figure 18: Results of offline experiments on logged perception data. **Left:** Validation Loss during learning for the balanced and standard weighting **Center:** Validation Loss as a function of the replanning corridor size **Right:** Validation Loss as a function of the number of regression trees.

tion, all raw sensor data was logged along with the example path. Perceptual features were then produced offline by feeding the raw sensor data through Crusher's perception software. In this way, the base perception system and its features could be modified and improved without having to recollect new training data; the raw data is just reprocessed, and a cost function learned for the new features.

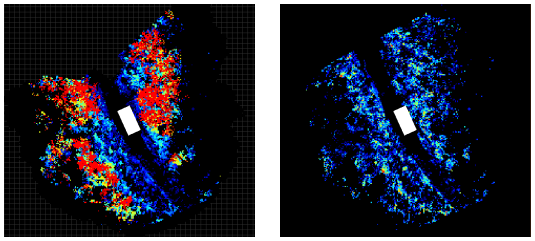
This set of examples was first used to perform a series of offline experiments to validate the dynamic extension of LEARCH. Loss on a validation set was used as the metric, to measure how well planned behavior recreated example behavior. These results are presented in Figure 18. The left graph again demonstrates the effectiveness of performing balanced as opposed to unbalanced regression, as the balanced version has superior validation performance. The center graph further demonstrates the utility of replanning with corridor constraints. With a small corridor size, the algorithm is able to smooth out some of the noise in example human behavior, and improve generalization. As the corridor size increases, the algorithm begins to over-smooth, resulting in decreasing validation performance. This also demonstrated how validation data can be used to automatically determine the optimal corridor size.

An experiment was also performed to assess the effectiveness of filtering out inconsistent timesteps. A single expert behavior was used to learn a cost function, first with no filtering, and then with approximately 10% of its timesteps automatically filtered. As would be expected, the performance of the remaining 90% of the training set improved after filtering (Figure 19). However, performance on the validation set also improved slightly. This demonstrates that filtering out inconsistent timesteps non only improves performance on examples for which the filtered timesteps were inconsistent, it also improves generalization to unseen examples.

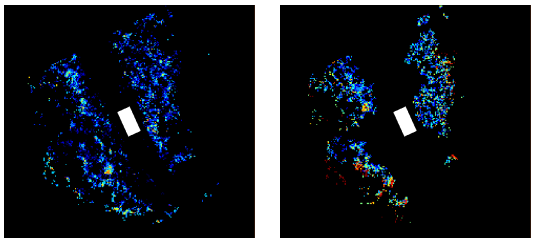
As cost evaluations in an onboard perception system must be performed in real time, the computational cost of an evaluation is an important consideration. As described in (Section 4.3.1), using only linear regressors is beneficial from a computational standpoint, and feature learning can be used to improve the complexity of the cost function if necessary. Figure 18 (**Right**) shows validation loss as a function of the number of added features learned using simple regression trees. At first, additional features improves the validation performance; however, eventually too many features can cause overfitting.

Next, the collected training set was used to learn a cost function to run onboard Crusher. Originally, Crusher used an engineered perception cost function. During more than 3 years of Crusher development, this cost function was continually redesigned and retuned, culminating in a high performance system [Stentz et al., 2007]. However, this performance came at a high cost of human effort. Version control logs indicate that 145 changes were made to just the structure of the model mapping perceptual features to costs; additionally more than 300 parameter changes were checked in (untold more were tried at one point or another, requiring verification on logged data or actual vehicle performance). 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 time to collect the final training set for learning by demonstration required only a few hours of human time (spread over several months). This seemingly small amount of human demonstration is sufficient due to the numerous constraints implied by each example behavior: the approximately 3 kilometers of demonstration provides hundreds of thousands of examples of states to traverse, and millions more

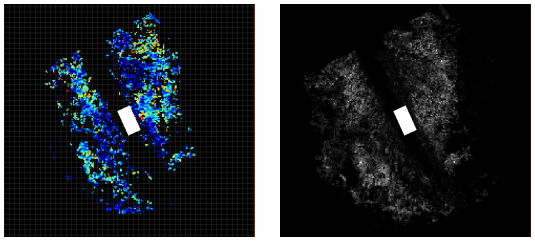
(a) Left Camera Image (b) Right Camera Image



(c) Max Object Height (d) Density in Wheel Region



(e) Density in Hull Region (f) Density above Vehicle



(g) Solid Classification (h) Learned Cost

Figure 17: An example of learned perception cost from a simple scene, depicted in (a),(b). Processing of raw data results in perceptual features (a subset of which are shown in (c) - (g)) that are mapped into cost (h) by a learned function.

Figure 19: Comparison of performance with and without filtering. Filtering out inconsistent timesteps improves performance on both training example which were inconsistent with the filtered examples, as well as on an independent validation set.

examples of states that were avoided.

Crusher's perception system actually consists of two modules: a local perception system that produces high resolution cost data out to 20m, and a far range perception system that produces medium resolution cost data out to 60m. The far range system utilizes near-to-far learning as described in [Sofman et al., 2006] for learning a far range cost function from near range cost data. Therefore, when the system learns online from scratch, the near range cost function implicitly defines the far range cost function. For this reason, learning by demonstration was only used to learn the near range function; the far range function would automatically adapt to whatever near range function was utilized.

As Crusher utilizes a hybrid local/global planning architecture, it was necessary to learn a cost function with respect to the lowest level of motion planning (see Section 4.3.3); otherwise the learned cost function would not actually recreate demonstrated behaviors when applied online. However, the local planner suffers from low fidelity; while the concatenation of multiple planning cycles can produce behavior of surprising complexity, individual planning cycles consider only a relatively small subset of possible actions. This approach has proven quite effective on Crusher, as well as in previous off road mobile systems [Singh et al., 2000, Kelly et al., 2006, Biesiadecki and Maimone, 2006, Carsten et al., 2009]. Unfortunately, as previously stated this inability for the planner to recreate demonstrated behavior in a single planning cycle can

produce noise in a learned cost function.

If there were some way to map a demonstrated behavior to the appropriate planner action at each cycle, then the selection of that action could serve as the proper termination condition. Unfortunately, collecting this information during demonstration by an expert would be extremely tedious, requiring an expert selection at every planning cycle. Instead, a heuristic approach to approximating this decision was implemented [Silver et al., 2009a]. Essentially, this technique seeks to project the expert's example behavior onto the space of possible planner actions. This was performed by first learning a perception cost function for Crusher's global planner (as this planner is not kinematically constrained, it can better reproduce demonstrated paths). Such a cost function will generally underestimate the cost necessary for the equivalent kinematically constrained planner. Therefore, each potential constrained planner plan is scored by its *average* cost instead of total cost. A plan with low average cost can not be said to be optimal, but it traverses desirable(low cost) terrain. An additional penalty based on path length is added to bias scores towards those that make progress towards the goal[9]. 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 that is feasible to the local planner.

The performance of different perception cost functions was compared through more than 150 km of comparison trials. The final results comparing 4 different cost functions are presented in Table 2. In addition to the engineered cost function, 3 learned cost functions were compared: one using the global planner, one using the local planner, and one using the local planner with the initial heuristic replanning. Unfortunately, as the cost function itself is the variable being tested, cost can not be used as a metric for comparing the relative safety of each systems behavior. Therefore, various proprioceptive statistics were recorded to offer an approximate quantitative description of safety. The number of safety related e-stops was also recorded. The statistics for each learned system were then compared on a waypoint by waypoint basis to the engineered system, to test for statistical significance. While it is not possible to convert these numbers into a single quantitative metric for comparison[10], it is possible to make a broad relative comparison between systems by observing individual statistics.

In comparison to the engineered system, the cost function learned for the global planner resulted in overly aggressive performance. As discussed in Section 4.3.3, learning in this manner does not result in sufficiently high costs for Crusher to avoid obstacles online. The empirical result is that Crusher drives faster, turns less, and backs up less, while suffering from increased mobility risk (in the form of twice the rate of safety e-stops). In contrast, the cost function learned for the local planner resulted in performance very similar to that of the already high performance engineered system; The only significant difference was the learned system turned slightly less (as indicated by lower average angular velocity and lateral speed/acceleration).

Adding an initial heuristic replanning step to learning a cost function for the local planner resulted in a system that also maintained a seemingly equal level of safety to the engineered system; the difference in safety e-stops was not statistically significant, but there was a significant decrease in the wear on Crusher in the form of lower motor current draw and less suspension travel. However, this equal safety was achieved with seemingly more aggressiveness than the engineered cost function. This is indicated by a statistically significant decrease not only in angular velocity and lateral movement, but also in the amount of backing up, with a significant increase in average speed. The effect of the initial replanning stage is to reduce noise in the cost function; the reduction of this noise allows the vehicle to alter its behavior less in the face of false positive high cost regions, while still avoiding true obstacles. The overall result is a slight performance improvement, achieved with orders of magnitude less human effort.

6 Conclusion

This paper addresses the task of interpreting perceptual data for use in autonomous navigation. We have shown the applicability of learning from expert demonstration 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 intervention. This automated approach not only reduces development and validation time, but can produce a more efficient and robust system than manual approaches.

Most importantly, this approach provides a formal-

[9] the weight of this penalty can be automatically tuned by optimizing performance on a validation set, preventing any hand tuning

[10] doing so would require a manually tuned and essentially arbitrary weighting, the very problem this work seeks a solution to

System	Avg Dist 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)
Engineered	130.7	3.24	6.56	0.181	0.107	7.53
Global	123.8*	3.34*	4.96*	0.170*	0.081*	7.11*
Local	127.3	3.28	5.93*	0.172*	0.100	7.35
Local w/replan	124.3*	3.39*	5.08*	0.170*	0.082*	7.02*

System	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	4.06	2.21	0.696	0.997	0.239	0.027
Global	4.02	2.22	0.710*	0.966*	0.237	0.054*
Local	4.06	2.22	0.699	0.969*	0.237	0.034
Local w/replan	3.90*	2.18	0.706*	0.966*	0.234*	0.030

Table 2: Averages over 295 different waypoint to waypoint trials per perception system, totaling over 150km of traverse. Statistically significant differences (from Engineered) denoted by *

ism for how a cost function should be defined; it is the simplest function that meets constraints implied by example behavior. As opposed to a hand tuned approach (which can easily overfit) this simplicity provides a theoretical basis for achieving robustness and generalization. Further, this approach naturally results in an automated framework for both initial and continuing cost function validation.

Future work will explore the application of this approach to learning cost functions over full state-action pairs. The behavior of an autonomous navigation system is defined not only by which terrain it prefers, but by which motions it prefers (e.g. minimum curvature). Improper modeling of these preferences is just as significant a detriment to robot performance as improper modeling of terrain preferences; it also contributes to the inability of some planning systems to properly recreate demonstrated behavior. Previous work [Abbeel et al., 2008] has briefly investigated this challenge independently of the equivalent perception problem. However, by learning all preferences at once from human demonstration, we hope to further improve the robustness of autonomous navigation, while further reducing the effort involved in deployment and validation of such systems.

Acknowledgments

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. Additionally, the authors would like to thank the many members of the Crusher vehicle and autonomy teams who made this work possible.

References

[Abbeel et al., 2008] Abbeel, P., Dolgov, D., Ng, A. Y., and Thrun, S. (2008). Apprenticeship learning for motion planning with application to parking lot navigation. In *International Conference on Intelligent Robots and Systems*.

[Abbeel and Ng, 2004] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine learning*.

[Angelova et al., 2007] Angelova, A., Matthies, L., Helmick, D., and Perona, P. (2007). Learning and prediction of slip from visual information. *Journal of Field Robotics*, 24(3):205–231.

[Bagnell et al., 2010] Bagnell, J., Bradley, D., Silver, D., Sofman, B., and Stentz, A. (2010). Learning rough-terrain autonomous navigation. *IEEE Robotics and Automation Magazine*.

[Bajracharya et al., 2008] Bajracharya, M., Tang, B., Howard, A., Turmon, M., and Matthies, L. (2008). Learning long-range terrain classification for autonomous navigation. In *IEEE International Conference on Robotics and Automation*, pages 4018–4024.

[Balakirsky and Lacaze, 2000] Balakirsky, S. and Lacaze, A. (2000). World modeling and behavior gen-

eration for autonomous ground vehicle. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 2, pages 1201–1206.

[Biesiadecki and Maimone, 2006] Biesiadecki, J. and Maimone, M. (2006). The mars exploration rover surface mobility flight software driving ambition. In *IEEE Aerospace Conference*.

[Bradley et al., 2007] Bradley, D., Unnikrishnan, R., and Bagnell, J. A. (2007). Vegetation detection for driving in complex environments. In *International Conference on Robotics and Automation*.

[Buehler, 2006] Buehler, M. (2006). Summary of dgc 2005 results. *Journal of Field Robotics*, 23:465–466.

[Carsten et al., 2009] Carsten, J., Rankin, A., Ferguson, D., and Stentz, A. (2009). Global planning on the mars exploration rovers: Software integration and surface testing. *Journal of Field Robotics*, 26:337–357.

[Dahlkamp et al., 2006] Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., and Bradski, G. (2006). Self-supervised monocular road detection in desert terrain. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA.

[Dima et al., 2004] Dima, C., Vandapel, N., and Hebert, M. (2004). Classifier fusion for outdoor obstacle detection. In *International Conference on Robotics and Automation*, volume 1, pages 665 – 671. IEEE.

[Ferguson and Stentz, 2006] Ferguson, D. and Stentz, A. (2006). Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics*, 23(2):79–101.

[Hadsell et al., 2009] Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., and LeCun, Y. (2009). Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26:120–144.

[Halatci et al., 2007] Halatci, I., Brooks, C., and Iagnemma, K. (2007). Terrain classification and classifier fusion for planetary exploration rovers. In *IEEE Aerospace Conference*.

[Hamner et al., 2006] Hamner, B., Singh, S., and Scherer, S. (2006). Learning obstacle avoidance parameters from operator behavior. *Journal of Field Robotics*, 23(1):1037–1058.

[Happold and Ollis, 2007] Happold, M. and Ollis, M. (2007). Using learned features from 3d data for robot navigation. *Autonomous Robots and Agents*, 76:61–69.

[Helmick et al., 2009] Helmick, D., Angelova, A., and Matthies, L. (2009). Terrain adaptive navigation for planetary rovers. *Journal of Field Robotics*, 26:391–410.

[Howard et al., 2007] Howard, A., Turmon, M., Matthies, L., Tang, B., Angelova, A., and Mjolsness, E. (2007). Towards learned traversability for robot navigation: From underfoot to the far field. *Journal of Field Robotics*, 23:1005–1017.

[Howard et al., 2005] Howard, A., Werger, B., and Seraji, H. (2005). A human-robot mentor-protege relationship to learn off-road navigation behavior. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 430–435 Vol. 1.

[Huertas et al., 2005] Huertas, A., Matthies, L., and Rankin, A. (2005). Stereo-based tree traversability analysis for autonomous off-road navigation. In *Seventh IEEE Workshops on Application of Computer Vision*, volume 1, pages 210–217.

[Iagnemma et al., 1999] Iagnemma, K., Genot, F., and Dubowsky, S. (1999). Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In *IEEE Internation Conference on Robotics and Automation*.

[Iagnemma et al., 2004] Iagnemma, K., Kang, S., Shibly, H., and Dubowsky, S. (2004). Online terrain parameter estimation for wheeled mobile robots with application to planetary rovers. *IEEE Transactions on Robotics*, 20(5):921–927.

[Kalman, 1964] Kalman, R. (1964). When is a linear control system optimal? *Trans. ASME, J. Basic Engrg.*, 86:51–60.

[Karlsen and Witus, 2007] Karlsen, R. and Witus, G. (2007). Terrain understanding for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 895–900.

[Kelly, 1995] Kelly, A. (1995). *An Intelligent Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem*. PhD thesis, Robotics Institute, Carnegie Mellon University.

[Kelly et al., 2006] 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., and

Warner, R. (2006). Toward reliable off road autonomous vehicles operating in challenging environments. *International Journal of Robotics Research*, 25(5-6):449–483.

[Kim et al., 2006] Kim, D., Sun, J., Oh, S. M., Rehg, J., and Bobick, A. (2006). Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 518–525.

[Lacaze et al., 2002] Lacaze, A., Murphy, K., and Delgiorno, M. (2002). Autonomous mobility for the demo iii experimental unmanned vehicles. In *in Assoc. for Unmanned Vehicle Systems Int. Conf. on Unmanned Vehicles (AUVSI 02*.

[Lalonde et al., 2006] Lalonde, J.-F., Vandapel, N., Huber, D., and Hebert, M. (2006). Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(10):839–861.

[LeCun et al., 2006] LeCun, Y., Muller, U., Ben, J., Cosatto, E., and Flepp, B. (2006). Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems 18*. MIT Press.

[Manduchi et al., 2005] Manduchi, R., Castano, A., Talukder, A., and Matthies, L. (2005). Obstacle detection and terrain classification for autonomous off-road navigation. *Autonomous Robots*, 18:81–102.

[Mason et al., 2000] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (2000). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA. MIT Press.

[Ng and Russell, 2000] Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*.

[Olin and Tseng, 1991] Olin, K. and Tseng, D. (1991). Autonomous cross-country navigation: an integrated perception and planning system. *IEEE Expert*, 6(4):16–30.

[Ollis et al., 2007] Ollis, M., Huang, W. H., and Happold, M. (2007). A bayesian approach to imitation learning for robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Pomerleau, 1989] Pomerleau, D. (1989). Alvinn: an autonomous land vehicle in a neural network. *Advances in neural information processing systems 1*, pages 305 – 313.

[Procopio et al., 2007] Procopio, M., Mulligan, J., and Grudic, G. (2007). Long-term learning using multiple models for outdoor autonomous robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3158–3165.

[Rasmussen, 2002] Rasmussen, C. (2002). Combining laser range, color, and texture cues for autonomous road following. In *IEEE Conference on Robotics and Automation*.

[Ratliff et al., 2006] Ratliff, N., Bagnell, J. A., and Zinkevich, M. (2006). Maximum margin planning. In *International Conference on Machine Learning*.

[Ratliff et al., 2007] Ratliff, N., Bradley, D., Bagnell, J. A., and Chestnutt, J. (2007). Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA. MIT Press.

[Ratliff et al., 2009] Ratliff, N. D., Silver, D., and Bagnell, J. A. (2009). Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*.

[Seraji and Howard, 2002] Seraji, H. and Howard, A. (2002). Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation*, 18(3):308–321.

[Shneier et al., 2008] Shneier, M., Chang, T., Hong, T., Shackleford, W., Bostelman, R., and Albus, J. (2008). Learning traversability models for autonomous mobile vehicles. *Autonomous Robots*, 24:69–86.

[Silver et al., 2008] Silver, D., Bagnell, J. A., and Stentz, A. (2008). High performance outdoor navigation from overhead data using imitation learning. In *Proceedings of Robotics Science and Systems*.

[Silver et al., 2009a] Silver, D., Bagnell, J. A., and Stentz, A. (2009a). Applied imitation learning for autonomous navigation in complex natural terrain. In *Field and Service Robotics*.

[Silver et al., 2009b] Silver, D., Bagnell, J. A., and Stentz, A. (2009b). Perceptual interpretation for autonomous navigation through dynamic imitation

learning. In *International Symposium on Robotics Research*.

[Silver et al., 2006] Silver, D., Sofman, B., Vandapel, N., Bagnell, J. A., and Stentz, A. (2006). Experimental analysis of overhead data processing to support long range navigation. In *Proceedings of the IEEE/JRS International Conference on Intelligent Robots and Systems*.

[Singh et al., 2000] Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., and Schwehr, K. (2000). Recent progress in local and global traversability for planetary rovers. *IEEE Conference on Robotics and Automation*.

[Sofman et al., 2006] Sofman, B., Ratliff, E. L., Bagnell, J. A., Cole, J., Vandapel, N., and Stentz, A. (2006). Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, 23(12).

[Stavens and Thrun, 2006] Stavens, D. and Thrun, S. (2006). A self-supervised terrain roughness estimator for off-road autonomous driving. In *In Proc. of Conf. on Uncertainty in AI*, pages 13–16.

[Stentz et al., 2007] Stentz, A., Bares, J., Pilarski, T., and Stager, D. (2007). The crusher system for autonomous navigation. In *AUVSIs Unmanned Systems*.

[Stentz and Hebert, 1995] Stentz, A. and Hebert, M. (1995). A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2).

[Sun et al., 2007] Sun, J., Mehta, T., Wooden, D., Powers, M., Rehg, J., Balch, T., and Egerstedt, M. (2007). Learning from examples in unstructured, outdoor environments. *Journal of Field Robotics*, 23:1019–1036.

[Talukder et al., 2002] Talukder, A., Manduchi, R., Rankin, A., and Matthies, L. (2002). Fast and reliable obstacle detection and segmentation for cross-country navigation. In *IEEE Intelligent Vehicles Symposium*, pages 610–618.

[Thrun et al., 2006] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692.

[Urmson et al., 2006] Urmson, C., Ragusa, C., Ray, D., Anhalt, J., Bartz, D., Galatali, T., Gutierrez, A., Johnston, J., Harbaugh, S., ldquo, H., Kato, Y., Messner, W., Miller, N., Peterson, K., Smith, B., Snider, J., Spiker, S., Ziglar, J., Whittaker, W. L., Clark, M., Koon, P., Mosher, A., and Struble, J. (2006). A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508.

[Wellington et al., 2006] Wellington, C., Courville, A., and Stentz, A. (2006). A generative model of terrain for autonomous navigation in vegetation. *The International Journal of Robotics Research*, 25(1):1287 – 1304.