Learning to Search: Functional Gradient Techniques for Imitation Learning

Nathan D. Ratliff
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ndr@ri.cmu.edu

David Silver
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
dsilver@ri.cmu.edu

J. Andrew Bagnell
Robotics Institute and Machine Learning
Carnegie Mellon University
Pittsburgh, PA 15213
dbagnell@ri.cmu.edu

Abstract

Programming robot behavior remains a challenging task. While it is often easy to abstractly define or even demonstrate a desired behavior, designing a controller that embodies the same behavior is difficult, time consuming, and ultimately expensive. The machine learning paradigm offers the promise of enabling "programming by demonstration" for developing high-performance robotic systems. Unfortunately, many "behavioral cloning" (Bain & Sammut, 1995; Pomerleau, 1989; LeCun et al., 2006) approaches that utilize classical tools of supervised learning (e.g. decision trees, neural networks, or support vector machines) do not fit the needs of modern robotic systems. These systems are often built atop sophisticated planning algorithms that efficiently reason far into the future; consequently, ignoring these planning algorithms in lieu of a supervised learning approach often leads to myopic and poor-quality robot performance.

While planning algorithms have shown success in many real-world applications ranging from legged locomotion (Chestnutt et al., 2003) to outdoor unstructured navigation (Kelly et al., 2004; Stentz, 2009), such algorithms rely on fully specified cost functions that map sensor readings and environment models to quantifiable costs. Such cost functions are usually manually designed and programmed. Recently, a set of techniques has been developed that explore learning these functions from expert human demonstration. These algorithms apply an *inverse optimal control* approach to find a cost function for which planned behavior mimics an expert's demonstration.

The work we present extends the Maximum Margin Planning (MMP) (Ratliff et al., 2006a) framework to admit learning of more powerful, non-linear cost functions. These algorithms, known collectively as LEARCH (LEArning to seaRCH), are simpler to implement than most existing methods, more efficient than previous attempts at non-linearization (Ratliff et al., 2006b), more naturally satisfy common constraints on the cost function, and better represent our prior beliefs about the function's form. We derive and discuss the framework both mathematically and intuitively, and demonstrate practical real-world performance with three applied case-studies including legged locomotion, grasp planning, and autonomous outdoor unstructured navigation. The latter study includes hundreds of kilometers of autonomous traversal through complex natural environments. These case-studies address key challenges in applying the algorithm in practical settings that utilize state-of-the-art planners, and which may be constrained by efficiency requirements and imperfect expert demonstration.

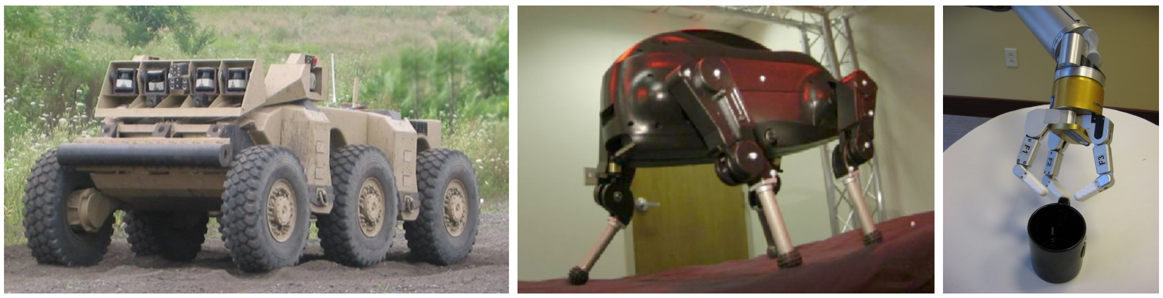


Figure 1: Imitation learning applies to a wide variety of robotic platforms. This figure shows a few of the robots on which the imitation learning algorithms discussed here have been implemented. From left to right, we have (1) an autonomous ground vehicle build by the National Robotics Engineering Center (NREC) known as Crusher, (2) Boston Dynamics's LittleDog quadrupedal robot, and (3) Barrett Technologies's WAM arm, wrist, and 10-DOF hand.

1 Introduction

Encoding a desired behavior into a robot is a difficult task. Programmers often understand intuitively how they want a robot to behave in various situations, but uncovering a set of parameters for a modern system which embodies that behavior can be a time consuming and expensive process. When programming a robot's behavior, researchers often adopt an informal process of repeated guess-and-check. For a skilled practitioner, this process borders on algorithmic. Imitation learning studies the algorithmic formalization for programming behavior by demonstration. Since many robot control systems are defined in terms of optimization (such as those designed around optimal planners), imitation learning can be modeled as finding optimization criteria that make the expert look optimal. This intuition is formalized by the maximum margin planning (MMP) framework which defines a convex objective that measures the suboptimality of an expert example policy.

MMP arises through a reduction of imitation learning to a form of machine learning called structured prediction. Structured prediction studies problems in which making multiple predictions simultaneously can improve accuracy. The term "structure" refers to the relationships among the predictions that make this possible. For instance, attemping to predict independently whether individual states occur along an optimal path in a graph has little hope of success without accounting for the connectivity of the graph. Robotics researchers naturally exploit this connectivity when developing efficient planning algorithms. By reducing imitation learning to a structured prediction problem, we leverage this body of work within learning to capture the problem's structure and improve prediction. Algorithms that solve MMP imitate by learning to predict the entire sequence of actions that an expert would take toward a goal.

In this paper, we build on and extend the techniques of (Ratliff et al., 2006a; Ratliff et al., 2006b) and present a family of function gradient-based algorithms, which we term LEArning to seaRCH (LEARCH), for solving MMP. This family includes a novel exponentiated functional gradient method that generalizes the exponentiated gradient algorithm (Kivinen & Warmuth, 1997) to function spaces in a way analogous to gradient boosting (Mason et al., 1999; Friedman, 1999a). LEARCH has performed successfully on a wide range of real-world structured prediction and imitation learning problems including programming by demonstration (Ratliff et al., 2006a; Silver et al., 2008), LADAR classification (Munoz et al., 2008; Ratliff et al., 2007a), sequence prediction (Ratliff et al., 2007a), and heuristic learning (Ratliff et al., 2006b). Figure 1 depicts three robotic platforms on which these algorithms have been applied.

LEARCH addresses a common difficulty researchers and engineers face with machine learning algorithms: designing and selecting appropriate features. Most algorithms are highly sensitive to the choice of features. We present "boosting" variants of our algorithms that provide first steps toward addressing this problem directly. The techniques provide more flexibility in the type of cost functions that can be generated than previous work using a linear combination of features, and consequently reduces the engineering burden and increases the applicability of imitation learning- and structured prediction generally- to real-world problems. Examining these algorithms also provides some insight into the intuitive process of feature generation that engineers perform when manually tuning control algorithms.

1.1 A short history of imitation learning

Before formally defining our imitation learning framework, we review prior work in the field and map out a progression of algorithms that form the foundation for this work. A full survey of the numerous approaches to imitation learning studied in the literature is beyond the scope of this paper; we point the interested reader to (Argall et al., to appear) for a detailed overview. In particular, we do not discuss the well known techniques of Schaal and Atkeson (Schaal & Atkeson, 1994; Atkeson et al., 1995), whose algorithms are not designed to generalize across differing problems.

The first notable examples of imitation learning came in the form of action prediction: predict the best action to take now given features of the current state of the environment and robot. Perhaps the best known system built using this paradigm was ALVINN (Pomerleau, 1989). Pomerleau et. al. trained a neural network to predict the desired steering angle of a vehicle given an image of the road ahead. More recently, (LeCun et al., 2006) implemented a similar reactive system using convolutional neural networks and demonstrated the learned controller on a small four wheeled robot known as DAVE.

Unfortunately, such a system is inherently myopic. This form of imitation learning assumes that all information necessary to make a decision at the current state can be easily encoded into a single vector of features that enables prediction with a relatively simple, learned function. However, this is often particularly difficult because the features must encode information regarding the future long-term consequences of taking an action. Under this framework, much of the work that we would like learning to handle is pushed onto the engineer in the form of feature extraction.

Further, as traditional supervised learning tools don't reason about the structure of making sequential decisions, it is often the case that small errors in prediction can translate to large errors in performance. Even a classifier trained to achieve very high accuracy agreement with example decisions is likely to choose a series of actions that doesn't lead to a valid path from a start to a distant goal. To avoid these problems, rather than minimizing a short-sighted loss over individual actions, a successful algorithm must learn to imitate entire sequences of actions.

In recent years, a new approach based on leveraging existing optimal control and planning algorithms has been developed for addressing the imitation learning problem. The concept is to leverage the older notion of *inverse optimal control* (Kalman, 1964; Boyd et al., 1994; Anderson & Moore, 1990): imitation learning is achieved by first recovering a cost function by observing an expert perform a task, and then executing new tasks using an optimal controller/planner to minimize that learned cost function. Inverse optimal control was first proposed– and solved for one dimensional inputs– by Kalman (Kalman, 1964) and solutions of increasing generality were developed over the ensuing years culminating in the work of (Boyd et al., 1994).

Linear systems theory is not sufficient to handle the generality of decision making problems considered in robotics, however. In this work, we consider the very general framework of Markov Decision Processes (MDPs). MDPs model the environment as a set of states connected by an action model, which defines the probability of arriving at any new state s' given that the agent started in state s and applied action a. This probability is Markovian: that is, it does not depend on previous states visited. Associated with an MDP is a reward (or cost) function, often defined over state-action pairs, which specifies quantitatively how much reward is received when an agent takes a particular action from a given state.

The first approaches to recovering a cost function for general, discrete MDPs by observing a demonstrated behavior were suggested in (Ng & Russell, 2000; Abbeel & Ng, 2004). In this formulation, it is assumed that the agent observed is truly acting in the MDP and the goal is to find a policy that accrues the same long-term cumulative reward as the demonstrated behavior. Unfortunately, as the authors point out, the problem is fundamentally ill-posed. We can see one example of this by noting that, for instance, given any MDP whose reward function is everywhere zero, all policies accrue the same long-term cumulative reward, namely zero. In (Abbeel & Ng, 2004), however, the authors note that if the reward function is a linear combination of features, an alternative, more practical, condition known as "feature count matching", can be posed to avoid this fundamental problem. Consider, for simplicity, the case of deterministic planning. We first define f^{sa} as the feature vector over a state-action pair (s, a), and choose the corresponding reward to be a linear combination of these features $c^{sa} = w^T f^{sa}$. Straightforward calculation shows that if the sum of features (i.e. $\sum_{(s,a)\in\xi} f^{sa}$) along a path ξ taken by the planner agrees with the same sum taken over the demonstrated path, by linearity both must achieve the same reward.

Unfortunately, while this work has provided an important demonstration that the inverse optimal control problem can be efficiently solved, it has a number of significant short-comings. First, the feature matching requirement is, in a sense, simultaneously over-constrained and ill-posed. It is over-constrained in the sense that there is almost never an optimal policy that achieves demonstrated feature counts on real problems– the experts demonstrating the desired behavior are not actually acting in the MDP we use to define our control problem, and even were they to be, they are rarely optimal. As such, we are left with an ill-posed problem, for which (Abbeel & Ng, 2004) offers specific algorithmic approaches for finding one of an infinite number of randomized policies (in the specific case actually a mixture of policies where we choose randomly which controller to execute on the initial step) that achieve the desired feature counts. Such policies can differ wildly in the behavior they actually manifest. Recently, (Ziebart et al., 2008) resolved these difficulties in the feature count matching approach by connecting the inverse optimal control problem to probabilistic modeling and the Maximum Entropy Method (Jaynes, 2003) and provided a unique, optimal, stochastic policy.

Further, while defining inverse reinforcement learning as the problem of matching cumulative feature counts is sensible when the reward function is a linear combination of features, the condition is no longer relevant for more general nonlinear reward functions. Many of the imitation learning algorithms that we discuss below are nonlinear and consequently require a fundamentally different problem definition. We propose a more general definition of imitation learning by inverse optimal control that attempts to directly mimic behavior according to a loss function measuring deviation from the behavior we desire (and is agnostic as to whether the demonstrating agent is truly trying to minimize a reward function in an MDP), provides approximation and regret (Ratliff et al., 2007a) guarantees on the resulting solutions, and admits a particular class of non-linear loss functions. While LEARCH has close connections with the notion of achieving similar feature counts when we specialize to linear reward functions, it returns only a single, deterministic policy for solving the problem.

We have applied the general LEARCH framework in settings ranging from quadrupedal locomotion to long-range outdoor autonomous navigation. These results developed over the preceedings three years (Ratliff et al., 2006a; Ratliff et al., 2006b; Ratliff et al., 2007a; Silver et al., 2008) are, to the best of our knowledge, the first practical applications of inverse optimal control algorithms to robotic systems. Sections 5 and 6 present some of these case studies; more recently, extensions to hierarchical learning problems for quadrupedal locomotion (Kolter et al., 2008) have additionally supported inverse optimal control's status as a preferred approach to imitation learning.

1.2 An implementational introduction to LEArning to seaRCH

The LEArning to seaRCH framework for imitation learning builds on the algorithms discussed in this section. The basic approach is sufficiently intuitive that in the next section we describe it first as it might be practically implemented.

Let $\mathcal{D} = \{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ denote a set of examples, each consisting of an MDP $\mathcal{M}_i$ (excluding a specific reward function) and an example trajectory ξ_i between start and goal points. Figure 2 visually depicts the type of training data we consider here for imitation learning problems. Often, we can think of an example as a path between a pair of end points. Each MDP is imbued with a feature function that maps each state-action pair (s,a) in the MDP to a feature vector $f_i^{sa} \in \mathbb{R}^d$. This feature vector represents a set of d sensor readings (or derived quantities) that distinguish one state from the next.

For clarity, we consider here only the deterministic case in which the MDP can be viewed as a directed graph (states connected by actions). Planning between a pair of points in the graph can be implemented efficiently using combinatorial planning algorithms such as Dijkstra or A*. In this setting, it is common to consider costs rather than rewards. Intuitively, a cost can be thought of simply as a negative reward; the planner must minimize the cumulative cost of the plan rather than maximize the cumulative reward. For

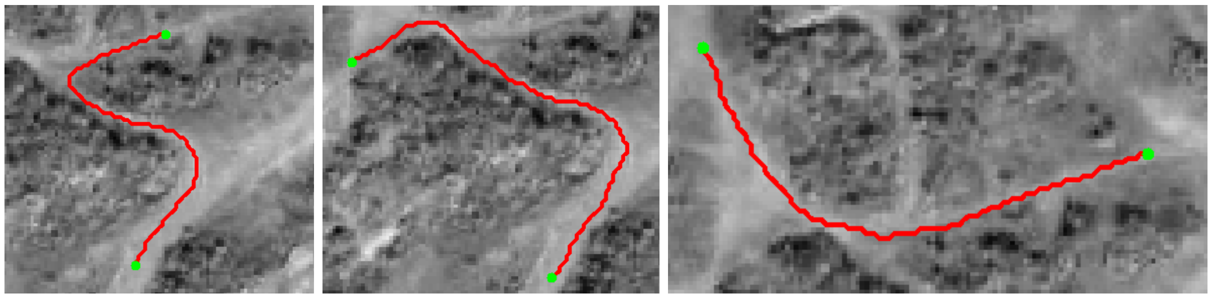


Figure 2: This figure demonstrates the flavor of the training data considered here in the context of imitation learning. In this case, a human expert specified by hand examples of the path (red) that a mobile robot should take between pairs of end points (green) through overhead satellite images. These example paths demonstrate the form of training data used for the outdoor navigational planning experiment discussed in section 6.

the moment, we ignore positivity constraints on the cost function required by many combinatorial planners such as A*, but we will address these constraints formally later on.

Intuitively, the maximum margin planning algorithm iteratively refines the cost function $c : \mathbb{R}^d \rightarrow \mathbb{R}$ in order to make the example trajectories optimal. Since there is a feature vector associated with each state-action pair, a cost map (i.e. a mapping from state-action pairs to costs) can be generated for each $\mathcal{M}_i$ by evaluating the cost function at each state-action feature vector f_i^{sa}. Given the cost map, any black box deterministic planning algorithm can be run to determine the optimal path. Since the example path ξ_i is a valid path, the minimum cost path returned by the planning algorithm will usually have lower cost. In essence, the goal of the learning algorithm is to find a cost function for which the example path is the minimum cost path. Therefore, we can use the gap between the cost of the example path and the cost of the minimum cost path as a measure of suboptimality.

During each iteration, the maximum margin planning algorithm suggests local corrections to the cost function to progress toward minimizing the gap. It does this by suggesting that the cost function be increased in regions of the feature space encountered along the planned path, and decreased in regions of the feature space encountered along the example path.[1]

Specifically, for each example i, the algorithm considers two paths: the example path ξ_i, and the planned path $\xi_i^* = \arg\min_{\xi\in\Xi_i} \sum_{(sa)\in\xi} c(f_i^{sa})$. In order to decrease the difference between the example path's cost and the planned path's cost, the algorithm needs to modify the cost function so that the cost of the planned path increases and the cost of the example path decrease. For each path, the path cost is simply a sum of state-action costs encountered along the way, which are each generated by evaluating the cost function at the feature vector f_i^{sa} associated with that state-action pair. The algorithm can, therefore, raise or lower the cost of this path incrementally simply by increasing or decreasing the cost function at the feature vectors encountered along the path.

Many planning algorithms, such as A*, require strictly positive costs in order to ensure the existence of an admissible heuristic. We can accommodate these positivity constraints by making our modifications to the log of the cost function and exponentiating the resulting log-costs before planning. Intuitively, since the exponential enforces positivity, decreasing the log-cost function in a particular region simply pushes it closer toward zero.

We can write this algorithm succinctly as depicted in Algorithm 1. We will see in Section 4 that this rather intuitive algorithm implements an exponentiated variant of functional gradient descent. Figure 3 depicts an iteration of this algorithm pictorially. The final step in which we raise or lower the cost function (or the log-cost function) in specific regions of the feature space is intentionally left vague at this point. The easiest way to implement this step is to find a regression function that is positive in regions of the feature space where we want the cost function to increase, and negative in regions where we want the function to

[1]Below we show that the accumulation of these corrections minimizes an objective function that measures the error on our current hypothesis.

Algorithm 1 LEARCH intuition

```
1: procedure LEARCH( training data $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$, feature function $f_i$ )
2:     while not converged do
3:         for each example $i$ do
4:             Evaluate the cost function at each state-action feature vector $f_i^{sa}$ for MDP $\mathcal{M}_i$ to create the
                   cost map $c_i^{sa} = c(f_i^{sa})$.
5:             Plan through the cost map $c_i^{sa}$ to find the minimum cost path $\xi_i^* = \arg\min_{\xi \in \Xi_i} \sum_{(s,a)\in\xi} c_i^{sa}$.
6:             Increase the (log-)cost function at the points in the feature space encountered along the min-
                   imum cost path $\{f_i^{sa} \mid \forall(s,a) \in \xi_i^*\}$, and decrease the (log-)cost function at points in the
                   feature space encountered along the example path $\{f_i^{sa} \mid \forall(s,a) \in \xi_i\}$.
7:         end for
8:     end while
9: end procedure
```

decrease. We can find such a function by specifying for each feature vector f_i^{sa} under consideration a label of either $+1$ or -1, indicating whether we want the function to be raised or lowered in that region. Given this data set, we can use any of a number of out-of-the-box regression algorithms to learn a function with the desired property.

1.3 Loss-augmentation

Most students can attest that learning on difficult problems makes simple problems easier. In this section, we use this intuition to devise a simple modification to the algorithm discussed in section 1.2 that greatly improves generalization both in theory and in practice. Section 2 provides a formal interpretation of the resulting algorithm in terms of margin-maximization. Indeed, if we break from the traditional view, all margin-based learning techniques, such as the support vector machine, have a similar interpretation.

Surprisingly, a simple modification to the cost map inserted immediately before the planning step is sufficient to inject a notion of margin into the algorithm. Intuitively, this cost map augmentation makes it more difficult for the planning algorithm to return the example path by making alternative paths look more desirable. Applying this handicap during training forces the algorithm to continue updating the cost function until the demonstrated path ξ_i appears significantly more desirable than alternative paths. Specifically, we lower the cost of undesirable state-action pairs to make them more likely to be chosen by the planner during training. With this augmentation, even if the example path is currently the minimum cost path through the actual cost map, it may not be the minimum cost path through the augmented cost map.

In order to solidify this concept of undesirable state-action pair, we define what we call a *loss field*. Each pair consisting of an MDP $\mathcal{M}_i$ and an example trajectory through that MDP ξ_i has an associated loss field which maps each state-action pair of $\mathcal{M}_i$ to a nonnegative value. This value quantifies how bad it is for an agent to end up traversing a particular state-action pair when it should be following the example path. The simplest example of a loss field is the hamming field which places a loss of 0 over state-action pairs found along the example path and a loss of 1 over all other pairs. In our experiments, we typically use a generalization of this hamming loss that increases more gradually from a loss of 0 along the example path to a loss of 1 away from the example path. This induces a quantitative notion of “almost correct” which is useful when there is noise in the training trajectories. In what follows, for a given state-action pair (s,a), we denote the state-action element of the loss field as l_i^{sa}, and the state-action element of the cost map as $c_i^{sa} = c(f_i^{sa})$.

The cost map modification step is called the *loss-augmentation*. During this step, the algorithm subtracts this loss field from the cost map element-wise. This amounts to defining the *loss-augmented* cost as $\tilde{c}_i^{sa} = c_i^{sa} - l_i^{sa}$. For state-action pairs that lie along the example path ξ_i, the loss is zero, and the cost function therefore remains untouched. As we venture away from the example path, the state-action loss values become

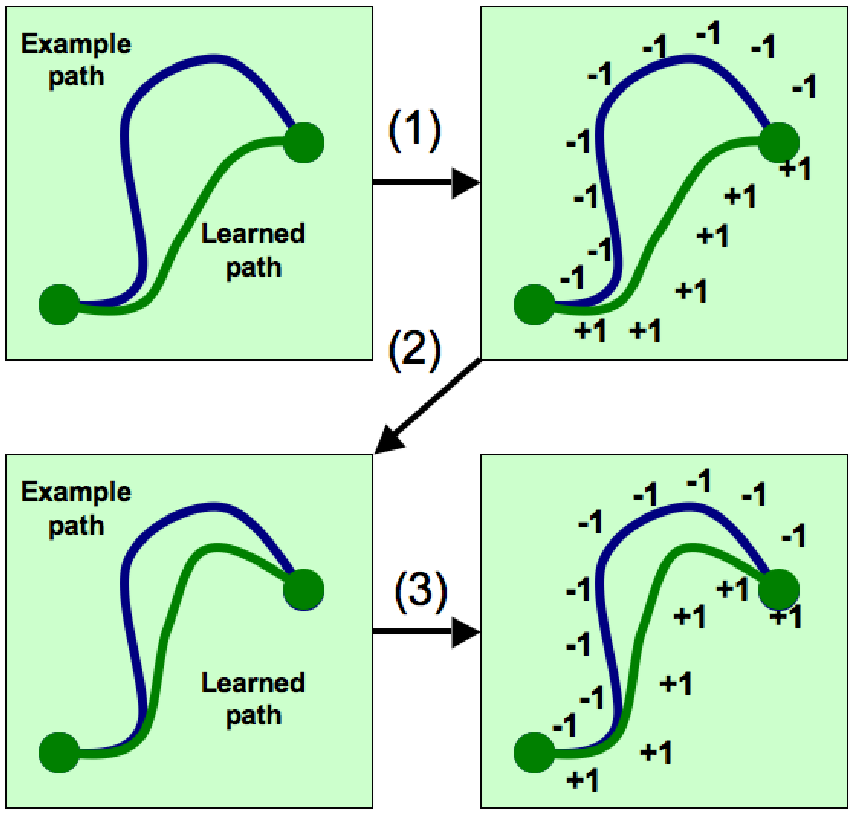


Figure 3: This figure visualizes an iteration of the algorithm discussed in section 1.2. Arrow (1) depicts the process of determining at which points in the feature space the function should be increased or decreased. Points encountered along the example path are labeled as -1 to indicate that their costs should be lowered, and points along the planned path are labeled as $+1$ to indicate that their costs should be raised. Along arrow (2) we generalize these suggestions to the entire feature space in order to implement the cost function modification. This incremental modification slightly improves the planning performance. We iterate this process, as depicted by arrow (3), until convergence.

increasingly large, and the augmentation step begins to lower the cost values substantially.

Intuitively, while the original algorithm discussed in section 1.2 cares only that the example path be the minimum cost path through the final cost map, the loss-augmentation step forces the algorithm to continue making updates until the cost of the example path is smaller than that of any other path by a margin that scales with the loss of that path. If the loss of a path is low, then the path is similar to the example path and the algorithm allows the costs to be similar. However, if the loss is large, the two paths differ substantially, and the loss-augmented algorithm tries to find a cost function for which the example path looks significantly more desirable than the alternative. In full, the algorithm becomes that given in Algorithm 2.

The next two sections develop the formal framework for Maximum Margin Planning and derive the algorithm discussed here as a novel functional gradient optimization technique applied to the convex objective functional that encapsulates MMP. Section 4 then presents the final form of this algorithm in Algorithm 5 and discusses practical considerations of using this framework in real-world robotic systems.

Algorithm 2 Loss-augmented LEARCH intuition

```
1: procedure Loss-AugLEARCH( training data $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$, loss function $l_i$, feature function $f_i$ )
2:     while not converged do
3:         for each example $i$ do
4:             Evaluate the cost function at each state-action feature vector $f_i^{sa}$ for MDP $\mathcal{M}_i$ to create the
                   cost map $c_i^{sa} = c(f_i^{sa})$.
5:             Subtract the loss field from the cost map to create the loss-augmented cost map $\tilde{c}_i^{sa} = c_i^{sa} - l_i^{sa}$.
6:             Plan through the loss-augmented cost map $\tilde{c}_i^{sa}$ to find the loss-augmented path $\xi_i^* =$
                   $\arg\min_{\xi \in \Xi_i} \sum_{(s,a)\in\xi} \tilde{c}_i^{sa}$.
7:             Increase the (log-)cost function at the points in the feature space encountered along the loss-
                   augmented path $\{f_i^{sa} \mid \forall(s,a) \in \xi_i^*\}$, and decrease the (log-)cost function at points in the
                   feature space encountered along the example path $\{f_i^{sa} \mid \forall(s,a) \in \xi_i\}$.
8:         end for
9:     end while
10: end procedure
```

2 Maximum margin planning

Maximum margin planning (MMP) is a framework for imitation learning based on maximum margin structured prediction techniques (Taskar et al., 2003; Taskar et al., 2006). We initially introduced the framework in (Ratliff et al., 2006a) for linear hypothesis spaces in conjunction with an efficient optimization procedure based on the subgradient method for convex optimization (Shor, 1985). This linear setting is well understood theoretically. In particular, contrary to many competing imitation learning frameworks, this linear margin-based formulation allows the straightforward derivation of strong regret and generalization bounds (Ratliff et al., 2007a). Recent experiments demonstrate that this algorithm outperforms most competing structured prediction techniques on variety of problems (Ratliff et al., 2007a). We later generalized the algorithm to nonlinear hypothesis spaces using a form of gradient boosting (Ratliff et al., 2006b). Since the publication of these algorithms, we have developed a newer, simpler, and experimentally superior nonlinear variant known as LEARCH (LEArning to seaRCH). This variant, which we presented informally in section 1.3, is currently our method of choice.

For completeness, we first review the reduction from imitation learning to maximum margin structured classification in section 2.2 for linear hypotheses. To strengthen intuition for the nonlinear generalization, section 2.3 additionally discuss the linear subgradient algorithm for optimizing the resulting finite-dimensional convex objective function. Section 3 then discusses the functional gradient-based optimization framework of (Mason et al., 1999) and generalizes the technique by deriving a novel exponentiated form of functional gradient descent. Finally, section 4 applies this algorithm to to a functional form of the MMP objective. LEARCH arises through the application of these functional gradient-based optimization procedures to optimizing the maximum margin planning objective functional. We begin by reviewing some definitions required to describe the problem.

2.1 Preliminaries

We model an environment as a Markov Decision Process (MDP). Throughout this document, we denote the set of states by $\mathcal{S}$, the set of possible actions by $\mathcal{A}$, and the combined set of state-action pairs by $\mathcal{M} = \mathcal{S} \times \mathcal{A}$. Each MDP has a transition function, denoted $\mathcal{T}_{sa}^{s'}$, which defines the probability of transitioning to state s' when taking action a from state s. The set of transition probabilities defines the dynamics of the MDP.

Following (Ratliff et al., 2006a), we denote policies using the dual flow. Intuitively, a policy, when run either infinitely or to a predefined horizon, visits each state-action pair an expected number of times. We denote the vector of these state-action frequency counts by $\mu \in \mathbb{R}_+^{|\mathcal{S}||\mathcal{A}|}$. The elements of these vectors adhere to a particular set of flow constraints (see (Ratliff et al., 2006a) for details). The constraints solidify our intuition of flow by specifying that the expected flow into a given state equals the expected flow out of that state, except at the start state which acts as a source, and at the goal state (should one exist) which acts as a sink. This notation is simply a matter of convenience for describing the algorithm; there is a one-to-one correspondence between the set of stationary Markovian policies and the set of feasible flow vectors (Puterman, 1994). The constraints can, therefore, be satisfied simply by invoking a generic MDP solver (i.e. a planning algorithm). We denote the set of all feasible flow vectors for a given MDP as $\mathcal{G}$.

At a high level, we define the imitation learning problem as the task of training a system to generalize from demonstrated behavior. Each training example i consists of an MDP (states, actions, and dynamics) and an expert policy. Each state-action pair has an associated fully observed feature vector $f^{sa} \in \mathbb{R}^d$ that concisely describes distinguishing qualities of that pair. These feature vectors are collected in a feature matrix which we denote $F \in \mathbb{R}^{d\times|\mathcal{S}||\mathcal{A}|}$. We are also given the set of possible flow vectors $\mathcal{G}_i$ (i.e. the set of all policies), and denote the expert policy by $\mu_i \in \mathcal{G}_i$. Finally, each example has an associated loss function $\mathcal{L}_i(\mu)$ which quantifies how bad a given policy μ is with respect to the desired policy μ_i. We require that the loss function decompose over state-action pairs so that the loss of a policy can be defined as $\mathcal{L}_i(\mu) = l_i^T\mu$, where $l_i \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. As in section 1.3, we call this vector the *loss field* and refer to each element l_i^{sa} of the loss field as a loss element. Each loss element intuitively describes the loss that a policy accrues by traversing that particular state-action pair. Using this notation, we can write the data set as $\mathcal{D} = \{(\mathcal{M}_i, F_i, \mathcal{G}_i, \mu_i, l_i)\}_{i=1}^N$.

The feature vectors in this problem definition transfer information from one MDP to another. By defining a policy implicitly in terms of the optimal controller for a cost function created from a sest of features, the policy can be applied to generalize to new MDPs outside the training set.

Of particular interest to us is the special case for which the dynamics of the MDP are deterministic and the problem is goal oriented. In this case, we can reduce the set of flow vectors $\mathcal{G}_i$ to only those that denote deterministic acyclic paths from the start state to the goal state. Each $\mu \in \mathcal{G}_i$ then becomes simply an indicator vector denoting whether or not the policy traverses a particular state-action pair along its path to the goal. Many combinatorial planning algorithms, such as A*, return only policies from this reduced set.

We assume that we have access to planner. Given a cost vector $c \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ that assigns a cost c^{sa} to each state action pair, the MDP solver returns an optimal policy. Formally, the planner solves the following inference problem:

$$\mu^* = \arg\min_{\mu\in\mathcal{G}} c^T\mu, \tag{1}$$

where $c^T\mu$ is the cumulative cost of policy $\mu \in \mathcal{G}$.

It is sometime useful to overload the notation $\mathcal{M}_i$ to denote the set of all state-action pairs in the i^{th} MDP. Additionally, we often denote the (s,a)th element of a vector $v \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ (defined over all state action pairs) as v^{sa}.[2]

2.2 Reducing imitation learning to maximum margin structured classification

We discussed informally in section 1.3 that the goal of imitation learning can be formalized as the problem of learning a cost function for which the example policy has lower expected cost than each alternative policy by a margin that scales with the loss of that policy. Intuitively, if a particular policy is very similar to the example policy as quantified by the loss function (i.e. the policy has low loss), then the margin for that policy is very small and the algorithm requires that the cost of the example policy be only slightly smaller. On the other hand, if the policy is very different from the example policy (i.e. the policy has high loss), then the margin for that policy will be large and the algorithm will want the cost of that policy to greatly exceed that of the example policy. Essentially, the margin adapts to each policy based on how bad that policy is relative to the example policy.

[2]When the path is deterministic, each element of μ is either 1 if the path traverses that particular state-action pair, or 0 if it does not. The path cost expression in this case reduces to simply the sum of the state-action costs over only the state-action pairs found in the path. This observation often eases implementation.

We can formalize this intuition using a form of machine learning known as maximum margin structured classification (MMSC) (Taskar et al., 2006; Ratliff et al., 2007a). Taking the cost function to be linear in the features, we can write the cost of a policy $\mu \in \mathcal{G}_i$ under MDP i as $c(\mu) = w^T F_i \mu$ for any real valued weight vector $w \in \mathbb{R}^d$. Using this notation, MMSC formalizes the above intuition through a set of constraints enforcing that for each MDP $\mathcal{M}_i$ and for each policy $\mu \in \mathcal{G}_i$,

$$w^T F_i \mu_i \leq w^T F_i \mu - l_i^T \mu. \tag{2}$$

These constraints, known in the structured prediction (MMSC) literature as structured margin constraints, explicitly state that the cost of the example policy $w^T F_i \mu_i$ should be lower than the cost of the alternative policy $w^T F_i \mu$ by an amount (i.e. a margin) that scales with the loss $l_i^T \mu$. If the loss term $l_i^T \mu$ is small, then we require the example policy μ_i to have cost only slightly less than μ. Alternatively, if the loss $l_i^T \mu$ is large, then the constraints require that the example policy's cost should be much smaller than that of μ.

At face value, equation 2 specifies an exponential number of constraints since the number of policies $|\mathcal{G}|$ is exponential in the number of state-action pairs $|\mathcal{S}||\mathcal{A}|$. However, following the logic originally introduced in (Taskar et al., 2003; Taskar et al., 2005) we note that, for a given example i, the left-hand-side of equation 2 is constant across all policies $\mu \in \mathcal{G}_i$. Therefore, if the constraint holds for the single policy that minimizes the right-hand-side expression then it holds for all policies. In other words, we need only worry about the constraint corresponding to the particular minimizing policy

$$\mu_i^* = \arg\min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} \quad = \quad \arg\min_{\mu \in \mathcal{G}_i} \{(w^T F_i - l_i^T)\mu\}. \tag{3}$$

If we were to remove the loss function term in equation 3, then the resulting expression would represent the traditional planning problem (see equation 1). With the presence of the additional loss term, it becomes what we call a *loss-augmented* planning problem. As in section 1.3, we refer to the vector $w^T F_i - l_i^T$ as the loss-augmented cost map. From this expression we can see that the loss-augmented planning problem can be solved simply by sending the loss-augmented cost map to the planner as described in 1.3.

This manipulation allows us to rewrite the constraints in equation 2 in a more compact form:

$$\forall i, \quad w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \left\{ w^T F_i \mu - l_i^T \mu \right\}. \tag{4}$$

While these new constraints are no longer linear, they remain convex[3]. Importantly, this transformation will allows us to derive a convex objective function for the imitation learning problem that can be efficiently optimized using the subgradient method.

These constraints, themselves, are not sufficient to characterize the desired solution. If the example policy's cost is even only a small $\epsilon > 0$ less than the cost of another policy μ, then a simple scaling of the vector w (i.e. a scaling of the cost function) can make the cost gap between the two policies arbitrarily large. With no additional constraints on the size of the weight vector w, this observation trivializes the structured margin criterion. Consequently, in order to make the margin term meaningful, we want to find the smallest weight vector w for which the constraints in equation 4 are satisfied. Moreover, since there may not be a weight vector that uniformly satisfies all of the constraints, much less a small one that exactly satisfies the constraint, we introduce a set of slack variables, one for each example $\{\zeta_i\}_{i=1}^N$, that allow constraint violations for a penalty.

These additional criteria suggest the following constrained convex optimization problem:[4]

$$\min_{w \in \mathcal{W}, \zeta_i \in \mathbb{R}_+} \frac{1}{N} \sum_{i=1}^{N} \zeta_i + \frac{\lambda}{2} \|w\|^2$$
$$\forall i, \quad w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \left\{ w^T F_i \mu - l_i^T \mu \right\} + \zeta_i$$

[3]The term on the right-hand-side of the inequality is a min over affine functions and is, therefore, concave.

[4]This optimization problem is a generalization of the traditional support vector machine (SVM). If we restrict $\mathcal{G}_i$ (which is formally interpreted as the (exponentially large) set of classes in a structured prediction problem (Taskar et al., 2006)), to contain only two elements and choose the loss function to be the hamming loss, then the convex program reduces to the one typically seen in the SVM literature.

where $\lambda \geq 0$ is a constant that trades off the penalty on constraint violations with the desire for small weight vectors. This optimization problem tries to find a simple (small) hypothesis w for which there are few constraint violations.

Technically, convex programming problems of this sort require nonnegativity constraints on the slack variables. However, in our case, the example policy is an element of the collection of all policies $\mu_i \in \mathcal{G}_i$, so the difference between the left and right sides of the cost constraints can never be less than zero ($w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i}\{w^T F_i \mu - l_i^T \mu\} \geq 0$). The slack variables will, therefore, always be nonnegative independent of explicit nonnegativity constraints.

Since the slack variables are in the objective function, the minimization drives the slack variables to be as small as possible. In particular, at the minimizer the slack variables will always exactly equal the constraint violation. The following equality condition, therefore, holds at the minimizer

$$\xi_i = w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i}\{w^T F_i \mu - l_i^T \mu\}. \tag{5}$$

This observation allows us to move the constraints directly into the objective function by replacing the slack variables with the expression given in equation 5. Doing so leads us to the following objective function which we term the Maximum Margin Planning (MMP) objective:

$$R(w) = \frac{1}{N}\sum_{i=1}^{N}\left(w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i}\{w^T F_i \mu - l_i^T \mu\}\right) + \frac{\lambda}{2}\|w\|^2. \tag{6}$$

This convex[5] objective function takes the form of a regularized risk function (Rifkin & Poggio, 2003); its two terms trade off data fit with hypothesis complexity.

We emphasize that this objective function forms an upper bound on our structured loss function $\mathcal{L}(\mu_i, \mu) = l_i^T \mu$ in a way that generalizes the upper bound formed on the zero-one loss by the support vector machine's binary binary hinge-loss. This structured loss function measures the difference in behavior between the learner and demonstrating expert. Optimizing equation 6, therefore, minimizes an upper bound on the the desired non-convex loss.

2.3 Optimizing the maximum margin planning objective

The maximum margin planning objective function given in equation 6 can be optimized in a number of ways. In this section, we discuss a general tool for convex optimization known as the subgradient method (Shor, 1985). This technique has attractive convergence guarantees, no regret online extensions (Zinkevich, 2003; Ratliff et al., 2007a), and, in the context of maximum margin planning, manifests itself as a particularly simple and intuitive iterative algorithm in which the planner is trained through repeated execution.

2.3.1 Subgradients and the subgradient method for convex optimization

The subgradient method is a generalization of gradient descent to functions that are convex but not necessarily differentiable. Formally, a subgradient of a convex function h at a point $w \in \mathcal{W}$ is any vector g which can be used to form an affine function that lower bounds h everywhere in $\mathcal{W}$ and equals h at w. Mathematically, we can write this condition as

$$\forall w' \in \mathcal{W},\ h(w') \geq h(w) + g^T(w' - w).$$

The expression on the right side of the inequality is the affine function. When $w' = w$, the rightmost term vanishes and the affine function equals $h(w)$. The inequality requires that the affine function lower bound h on the entire domain $\mathcal{W}$. In general, there could be a continuous set of vectors g, denoted $\partial h(w)$, for which this condition holds. However, at points of differentiability, the gradient is the single unique subgradient.

[5]Again, as was the case in the constraints given in equation 4, the term $\min_{\mu \in \mathcal{G}_i}\{w^T F_i \mu - l_i^T \mu\}$ is a min over affine functions which is known to be concave; its negative is therefore convex.

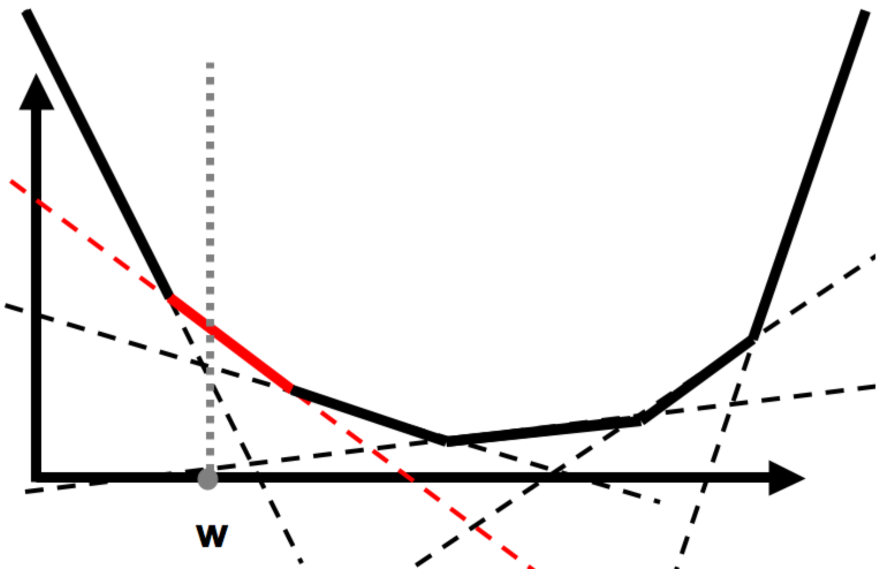


Figure 4: This figure visualizes the subgradient of a max over affine functions. Each affine function is denoted as a dashed line, and the surface of the resulting convex function is delineated in bold. The subgradient a point $w \in \mathcal{W}$ is simply the subgradient of the affine function that forms the surface at that point, shown here in red.

The subgradient method simply follows the negative subgradient at each time t. This update rule can be expressed as $w_{t+1} = w_t - \alpha_t g_t$, where g_t is a subgradient at w_t and α_t is a predefined step size sequence.[6] Additionally, since $\mathcal{W}$ can be any convex set, and since a step of fixed size α_t may result in a point outside of $\mathcal{W}$, the update typically includes a projection back onto the feasible set:

$$w_{t+1} = \mathcal{P}_{\mathcal{W}}\left[w_t - \alpha_t g_t\right]. \tag{7}$$

The projection operator $\mathcal{P}_{\mathcal{W}}[\cdot]$ can be approximate. The only requirement is that the projected point be closer to all other points in the convex set $\mathcal{W}$ than the original point was. This condition is typically expressed as

$$\forall w' \in \mathcal{W}, \quad \|\mathcal{P}_{\mathcal{W}}[w] - w'\| \leq \|w - w'\|. \tag{8}$$

By this definition, the projection has no effect on points already in the set $\mathcal{W}$. Thus, this modified algorithm proceeds without projection until a step takes the hypothesis outside the feasible set. At that point, the hypothesis is projected back onto $\mathcal{W}$. In our case, the set $\mathcal{W}$ is the set of weight vectors for which all cost elements in the provided examples are greater than a predefined positive constant. An approximate projection operator can be implemented by iteratively projecting onto the most violated constraint. See (Ratliff et al., 2006a) for details.

2.3.2 Computing the subgradient for linear MMP

The terms $w^T F_i \mu_i$ and $\frac{\lambda}{2}\|w\|^2$ are differentiable and, therefore, their unique subgradients are the gradients $F_i\mu$ and w, respectively. The term $-\min_{\mu \in \mathcal{G}_i}\{w^T F_i \mu - l_i^T \mu\}$ is only slightly more complicated. The surface

[6]To guarantee convergence, it is import to choose a step size sequence whose elements individually decrease to zero, but sum to infinity. We often use $\alpha_t = \frac{r}{\sqrt{t+m}}$ or $\alpha_t = \frac{r}{t+m}$ in our experiments, where r is some nonnegative "learning rate" scalar and m is a nonnegative integer specifying where along the progression the sequence starts. Intuitively, the steps reduce aggressively in the beginning but increasingly slow as time progresses. The parameter m, therefore, designates how quickly the step size shrinks in the beginning.

of this convex function is formed as a max over a set of affine functions.[7] The subgradient at w is, therefore, the gradient of the affine function forming the surface at that point. We can find this surface affine function simply by solving the loss-augmented inference problem given in equation 3. Using that notation, the affine function forming the surface at w is $(w^T F_i - l_i^t)\mu_i^*$, and we know the subgradient of that function to be $F_i\mu_i^*$.[8] Figure 4 visualizes the process of evaluating the subgradient of a max over affine functions.

Using the above results, we can write the subgradient of the maximum margin planning objective given in equation 6 as

$$\begin{aligned}\nabla R(w) &= \frac{1}{N}\sum_{i=1}^{N} F_i(\mu_i - \mu_i^*) + \lambda w \\ &= \frac{1}{N}\sum_{i=1}^{N} F_i \Delta\mu_i^* + \lambda w, \end{aligned} \tag{9}$$

where μ_i^* is the solution to the loss-augmented inference problem from equation 3 for example i. We denote $\Delta\mu_i^* = \mu_i - \mu_i^*$ to emphasize that this component of the gradient is constructed by transforming the difference in frequency counts between the example policy μ_i and the loss-augmented policy μ_i^* into the space of features using the matrix F_i. This term singles out the feature vectors at states for which the frequency counts differ substantially. If the example policy visits a particular state more frequently than the loss-augmented policy, the subgradient update rule will suggest a modification that will decrease the cost of that state. On the other hand, if the example policy visits a state less frequently, the update will want to increase the cost of that state.

The maximum margin planning algorithm, therefore, iterates the following update rule until convergence

$$w_{t+1} = \mathcal{P}_{\mathcal{W}}\left[w_t - \alpha_t\left(F_i(\mu_i - \mu_i^*) + \lambda w\right)\right]. \tag{10}$$

When both the environment and the policies are deterministic, the vector matrix product $F_i\mu$ can be implemented efficiently using sparse multiplication. For instance, many of our experiments use the A* algorithm to find an optimal deterministic policy (i.e. a path) through the environment. We compute the product $F_i\mu$ simply by accumulating the feature vectors encountered while traversing the path represented by μ.

3 Nonlinear techniques for optimization

In this section, we discuss generalizations of gradient based methods to function spaces, in preparation for discussing the exponentiated functional gradient algorithm in section 3.2. Section 4 covers in detail how these algorithms apply to the maximum margin planning framework developed formally in section 2.

3.1 Generalizing gradient descent to function spaces

In this section, we briefly review the argument presented in (Ratliff et al., 2006b) for generalizing gradient descent to the functional setting. That argument follows and slightly generalizes the argument originally provided by (Mason et al., 1999).

Consider objective functionals defined over a function $c \in \mathcal{L}^2$ as a sum of terms, each of which depend on $c(\cdot)$ through its evaluation at some x. These objective functionals can be written

$$F[c] = \sum_{i=1}^{k} \phi(c(x_i)). \tag{11}$$

[7]For clarity, we use the transformation $-\min_i h_i(w) = \max_i\{-h_i(w)\}$ to simplify the argument.

[8]There is a slight subtlety at points of nondifferentiability. At such points, two or more affine functions intersect and the loss-augmented inference problem has more than one solution (i.e. there are multiple optimal policies through the loss-augmented cost map). Since the subgradient algorithm requires only that one of the possible subgradients be followed at each time step, at these points we can choose any optimizer to construct the subgradient.

Evaluation of a function is a linear operation which can be defined as the L_2 inner product between the function $c(\cdot)$ and the Dirac delta function $\delta_x(\cdot)$ centered at the point x in question. Specifically,

$$c(x) = \langle \delta_x,\ c \rangle = \int_{\mathcal{X}} \delta_x(x')c(x')dx'. \tag{12}$$

As a linear functional, the gradient of the evaluation functional is simply given by

$$\nabla_f c(x) = \nabla_f \langle \delta_x,\ c \rangle = \delta_x. \tag{13}$$

Here we denote the functional gradient by ∇_f to distinguish it from the parametric gradient of section 2.3.

The functional gradient of objective functionals of the form given in equation 11 is therefore

$$\nabla_f F[c] = \sum_{i=1}^{k} \nabla_f \phi(c(x_i)) = \sum_{i=1}^{k} \phi'(c(x_i))\delta_{x_i}. \tag{14}$$

In the final expression, we use the common abbreviation $\phi'(c(x)) = \frac{d}{du}\phi(u)|_{c(x)}$.

While equation 14 provides a mathematical expression for the functional gradient of the objective functional, the functional gradient is otherwise intangible. The delta functions essentially provide only impulse information regarding how the current hypothesis function should be perturbed at a specific discrete set of points $\{x_i\}_{i=1}^k$. In order to make use of the impulse information provided by the L_2 functional gradient, the functional gradient must be projected onto a predefined set of tangible hypothesis functions $\mathcal{H}$. We often call this set of hypothesis functions the direction set. As we will see, the boosting algorithm finds the direction $h^* \in \mathcal{H}$ which best correlates with the functional gradient $g = \nabla_f F[c]$ defined by $h^* = \arg\max_{h\in\mathcal{H}} \langle h,\ g \rangle$.

For objective functionals of the form specified in equation 11, this inner product optimization can be written

$$h^* = \arg\max_{h\in\mathcal{H}} \left\langle h,\ \sum_{i=1}^{k} \phi'(c(x_i))\delta_{x_i} \right\rangle = \arg\max_{h\in\mathcal{H}} \sum_{i=1}^{k} \langle h,\ \phi'(c(x_i))\delta_{x_i} \rangle \tag{15}$$

$$= \arg\max_{h\in\mathcal{H}} \sum_{i=1}^{k} \phi'(c(x_i))h(x_i). \tag{16}$$

It is often the case that this maximization problem can be implemented efficiently (though approximately) using traditional binary classification or regression algorithms from the machine learning literature. For instance, in the most common application of this algorithm, the direction set is defined as a set of binary functions $\mathcal{H} \subset \{h \mid h(x) \in \{-1, 1\} \forall x \in \mathcal{X}\}$. If we define $\alpha_i = |\phi_i'(c(x_i))|$ and $y_i = \text{sign}(\phi_i'(c(x_i)))$, then the optimization problem in equation 16 can be rewritten as

$$h^* = \arg\max_{h\in\mathcal{H}} \sum_{i=1}^{k} \alpha_i y_i h(x_i). \tag{17}$$

This is a weighted binary classification problem, and can be solved using any of a variety of classification algorithms.

Alternatively, as discussed in (Friedman, 1999a), we can re-define our correlation measure as a weighted least squares classification problem. Using the same notation as above, we replace equation 17 with

$$h^* = \arg\min_{h\in\mathcal{H}} \sum_{i=1}^{k} \alpha_i (y_i - h(x_i))^2. \tag{18}$$

There are a number of reasons why this correlation measure may be preferred over the inner product given in equation 16. Friedman notes that the estimator is consistent (Friedman, 1999a). Some straightforward

Algorithm 3 Functional gradient intuition

```
1: procedure FunGrad( objective functional $F[\cdot]$, initial function $h_0$ )
2:     while not converged do
3:         Evaluate the functional gradient $g_t$ of the objective functional at the current hypothesis
               $f_t = -\sum_{\tau=0}^{t-1} \gamma_\tau h_\tau$.
4:         Find the element $h_t = h^* \in \mathcal{H}$ which best correlates with the functional gradient $g_t$ by maximizing
               either equation 16 or equation 18.
5:         Take a step in the negative of that direction forming $f_{t+1} = f_t - \gamma_t h_t = -\sum_{\tau=0}^{t} \gamma_\tau h_\tau$.
6:     end while
7: end procedure
```

calculations show that we can rewrite this correlation measure as:

$$\arg\min_{h\in\mathcal{H}} \sum_{i=1}^{k} \alpha_i (y_i - h(x_i))^2 = \arg\max_{h\in\mathcal{H}} \left(\sum_{i=1}^{k} \phi_i'(c(x_i))h(x_i) - \frac{1}{2}\sum_{i=1}^{k} |\phi_i'(c(x_i))| h(x_i)^2 \right). \tag{19}$$

This expression is essentially the same as equation 16, except there is an extra regularization type term which penalizes the size of the function h at each point x_i proportionally to how large the weight on the corresponding gradient component is at that point.

For the purposes of finding the most correlated search direction, this new correlation definition is equivalent to that defined in equation 16 when the direction set is the set of all classification algorithms (i.e. $h(x) \in \{-1, 1\}$, $\forall x$). Indeed, the quadratic second term of the argmax on the right hand side of equation 19 is constant with respect to h because $h(x)^2 = 1$ since $h(x)$ itself is either 1 or -1. However, for more general direction sets, this modified definition of correlation constrains the size of function at an output level. The original linear MMP algorithms suffered from a sensitivity to the relative scaling of its features, a problem common to margin based learning formulations including alternative IRL formulations as discussed in (Neu & Szepesvari, 2007).[9] Section 4.3 demonstrates a solution to this problem that arises naturally from this regression-based correlation measure when used with a hypothesis space of linear functions.

The functional gradient descent algorithm proceeds as presented in Algorithm 3.

Implementationally, a functional gradient can be viewed as a regression data set. This data set suggests where and how strongly the function should be increased or decreased at a collection of points in the feature space. For instance, let $\mathcal{D} = \{f_i, y_i\}_{i=1}^N$ be such a data set. Each y_i is a real number suggesting how strongly the function should be modified. If y_i is strongly positive, then this data point suggests a strong increase in the function at f_i. Alternatively, if y_i is strongly negative, then the function should be strong decreased at that point. Moreover, if y_i is zero, or close to zero, the data point indicates that the function should not be modified at all. Black box regression algorithms implement these suggestions and return to us a tangible function that we can add to our hypothesis. Figure 5 demonstrates pictorially the intuitive affect of functional gradient modification. On the left is a plot of the original function with the functional gradient suggestions superimposed. Implementationally, these modifications result in the figure on the right.

Unfortunately, it is difficult to apply functional gradient descent directly to optimizing the maximum margin planning objective since the planning algorithm, such as A*, often requires the costs to be positive. In the linear case, we satisfied the positivity constraints using an approximate projection step (Ratliff et al., 2006a) which iteratively projected onto the most violated constraint. It is more difficult, however, to apply such a projection in the functional setting. In prior work (Ratliff et al., 2006b), we developed a technique that utilized the linear MMP algorithm in the inner loop of the functional gradient algorithm to ensure adherence to the positivity constraints. That algorithm, however, is both more difficult to implement and tends to converge more slowly in practice.

[9] Intuitively, since we typically require the weight vector to lie within a Euclidean ball, a feature whose range is 10 times larger than another will likely dominate the hypothesis since a tiny weight on that feature can produce the same amount of cost variation as a large weight on the other.

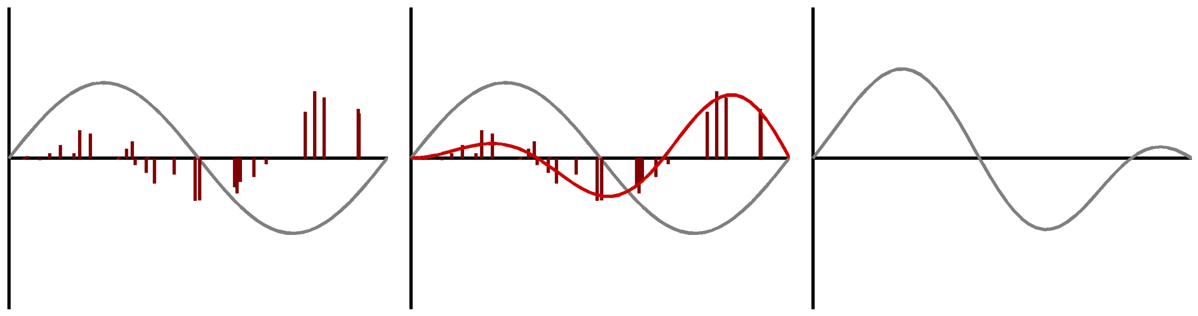


Figure 5: This figure shows a pictorial representation of the action of a functional gradient data set. The left plot shows the original function in gray along with a functional gradient data set indicated where and to what extent the function should be modified. The second plot includes a nonlinear regressor that generalizes from that data set to the rest of the domain. Finally, the right-most image shows the result of taking a functional gradient step: the nonlinear regressor in the second plot is simply added to the original function effectively implementing the discrete set of suggested modifications.

In the next few sections, we develop an alternative approach to utilizing functional gradient which ensures that the learned function is positive everywhere. The algorithm, exponentiated functional gradient descent, is a generalization of exponentiated gradient descent to function spaces. Section 3.2 derives the exponentiated functional gradient descent algorithm by analogy to the parametric variant, and section 4 utilizes this algorithm within the context of MMP and derives the LEARCH framework.

3.2 Generalizing exponentiated gradient descent to function spaces

In this section, we use the same intuition that brought the finite-dimensional gradient descent algorithm to the functional setting in section 3.1 to additionally generalize the finite-dimensional exponentiated gradient descent algorithm. We prove that the resulting update in function space has positive inner product with the negative functional gradient.

3.2.1 Exponentiated functional gradient descent

We can characterize the traditional Euclidean gradient descent update rule as a minimization problem. At our current hypothesis w_t, we create a linear approximation to the function (in the case of convex functions, this approximation is also a lower bound), and we minimize the approximation while regularizing back toward w_t. Mathematically, we write this as

$$w_{t+1} = \arg\min_{w \in \mathcal{W}} f(w_t) + g_t^T(w - w_t) + \frac{\lambda_t}{2}\|w - w_t\|^2, \tag{20}$$

where $g_t = \nabla f(w_t)$ is the gradient (or subgradient) at w_t. Analytically solving for the minimizer by setting the gradient of the expression to zero derives the Euclidean gradient descent rule: $w_{t+1} = w_t - \alpha_t g_t$ with step size $\alpha_t = 1/\lambda_t$. Thus, the gradient descent rule naturally encourages solutions that have a small norm in the sense of $\|\cdot\|_2$ (see (Zinkevich, 2003)). A similar procedure derives the update rule for exponentiated gradient descent as well. Replacing the Euclidean regularization term in equation 20 with an unnormalized KL-divergence regularization of the form $\text{uKL}(w, w_t) = \sum_j w^j \log \frac{w^j}{w_t^j} - \sum_j w^j + \sum_j w_t^j$ and analytically solving for the minimizer results in the exponentiated gradient update rule (Kivinen & Warmuth, 1997):

$$w_{t+1} = w_t e^{-\alpha_t g_t} = e^{-\sum_{\tau=0}^{t} \alpha_\tau g_\tau} \tag{21}$$

$$= e^{u_t} \tag{22}$$

Algorithm 4 Exponentiated functional gradient intuition

```
1: procedure ExpFunGrad( objective functional $F[\cdot]$, initial function $h_0$ )
2:     while not converged do
3:         Evaluate the functional gradient $g_t$ of the objective functional at the current hypothesis
               $f_t = e^{-\sum_{\tau=0}^{t-1} \gamma_\tau h_\tau}$.
4:         Find the element $h_t = h^* \in \mathcal{H}$ which best correlates with the functional gradient $g_t$ by maximizing
               either equation 16 or equation 18.
5:         Take a step in the negative of that direction forming $f_{t+1} = f_t e^{-\gamma_t h_t} = e^{-\sum_{\tau=0}^{t} \gamma_\tau h_\tau}$.
6:     end while
7: end procedure
```

where $\{\alpha_t\}_{t=1}^{\infty}$ is a sequence of step sizes (sometimes called learning rates). For simplicity, we assume in what follows that w_0 is the vector of all ones, i.e. $w_0 = e^z$ where z is the zero vector. This assumption is not necessary, but it lightens the notation and clarifies the argument.

In the final expression of equation 21, we denote $u_t = -\sum_\tau \alpha_\tau g_\tau$ in order to point out the relationship to the Euclidean gradient descent algorithm. The quantity u_t is the hypothesis that would result from an objective function whose gradients were g_t at each x_t. The only difference between the gradient descent algorithm and the exponentiated gradient descent algorithm is that in the gradient descent algorithm we simply evaluate the objective and its gradient using u_t, while in the exponentiated gradient algorithm, since the objective is a function of vectors from only the positive orthant, we first exponentiate this vector $w_t = e^{u_t}$ before evaluating the objective and its gradient. Essentially, the exponentiated gradient update rule is equivalent to the gradient descent update rule, except we exponentiate the result before using it.

In addition to the immediate benefit of only positive solutions, the key benefit enjoyed by the exponentiated gradient algorithm is a robustness to large numbers of potentially irrelevant features. In particular, powerful results (Kivinen & Warmuth, 1997; Cesa-Bianchi & Lugosi, 2006) demonstrate that the exponentiated gradient algorithm is closely related to the growing body of work in the signal processing community on sparsity and $\|\cdot\|_1$ regularized regression. (Tropp, 2004; Donoho & Elad, 2003) Exponentiated gradient achieves this by rapidly increasing the weight on a few important predictors while quickly decreasing the weights on a bulk of irrelevant features. The unnormalized KL prior from which it is derived encourages solutions with a few large values and a larger number of smaller values. In the functional setting, the weights are analogous to the hypothesized function evaluated at particular locations in feature space. We believe this form of regularization– or prior, taking a Bayesian view– is natural for many planning problems in robotics where there is a very large dynamic range in the kind of costs to be expected.

This work generalizes the connection between these two optimization algorithms to the functional setting. A formal derivation of the resulting the algorithm is quite technical and beyond the scope of this paper. The resulting algorithm, however, is straightforward and and analogous to the finite-dimensional setting. We derive this algorithm informally in what follows.

In functional gradient descent, we evaluate the functional gradient of the objective function, find the search direction from a predefined set of hypothesis functions that correlates best with the functional gradient, and take a step in the negative of that direction. In the exponentiated functional gradient algorithm we will essentially perform the same update (take a step in the negative direction of the projected functional gradient), but before evaluating the objective function or the functional gradient we will exponentiate the current hypothesis.

Explicitly, the exponentiated functional gradient descent algorithm dictates the procedure presented in Algorithm 4.

3.2.2 Theoretical results

We show here that when the functional gradient can be represented explicitly the exponentiated functional gradient descent algorithm produces a modification to the hypothesis that has a positive inner product with negative functional gradient. Therefore, on each iteration, there always exists a finite step length interval for which the algorithm necessarily decreases the desired objective while preserving the natural sparsity and dynamic range of exponentiated costs.

Let $g_t(x)$ be the functional gradient of a functional at hypothesis $f_t(x)$. Under the exponentiated functional gradient algorithm, $f_t(x) = e^{h_t(x)}$ for some log-hypothesis $h_t(x)$. Thus, we need only consider positive hypotheses: $f(x) > 0$ for all x. Our update is of the form $f_{t+1}(x) = f_t(x)e^{-\lambda_t g(x)}$, where $\lambda_t > 0$ is a positive step size. Therefore, we can write our update offset vector as

$$\begin{aligned} v_t(x) &= f_{t+1}(x) - f_t(x) = f(x)e^{-\lambda g(x)} - f(x) \\ &= f_t(x)\left(e^{-\lambda g_t(x)} - 1\right). \end{aligned}$$

We suppress the dependence on t in what follows for convenience.

Theorem 3.1: *The update directon $v(x)$ has positive inner product with the negative gradient. Specifically,*

$$-\int_{\mathcal{X}} g(x)v(x)dx = \int_{\mathcal{X}} g(x)f(x)\left(1 - e^{-\lambda g(x)}\right)dx > 0.$$

Proof. We first note that $\phi(u) = \frac{1}{u}(e^u - 1)$ is continuous and everywhere positive since u and $e^u - 1$ always have the same sign.[10] We can rewrite our expression as

$$\begin{aligned} \int_{\mathcal{X}} g(x)f(x)\left(1 - e^{-\lambda g(x)}\right)dx &= \int_{\mathcal{X}} g(x)f(x)\left(\lambda g(x)\phi(-\lambda g(x))\right)dx \\ &= \lambda \int_{\mathcal{X}} g(x)^2 f(x)\phi(-\lambda g(x))dx. \end{aligned}$$

The integrand is everywhere nonnegative, and when our functional gradient is not the zero function, there exist measureable regions over which the integrand is strictly positive. Therefore, $-\int_{\mathcal{X}} g(x)v(x)dx > 0$. □

The next section derives the LEARCH class of algorithms by applying our exponentiated functional gradient descent algorithm to the maximum margin planning framework.

4 Exponentiated functional gradient descent for maximum margin planning

Initial work into MMP focused on the linear setting in which the cost function is assumed to be a linear function of a predefined set of features. The optimization algorithms in this setting are just a simple variant on parametric gradient descent; unfortunately, while the linear setting is more easily understood theoretically (see (Ratliff et al., 2006a; Ratliff et al., 2007a)), feature extraction for linear models is often difficult.

We are now prepared to generalize the maximum margin planning framework. In the functional setting, the MMP objective takes on essentially the same form as equation 6, but with each policy cost term $w^T F_i \mu$ replaced by the more general term $\sum_{(s,a)\in\mathcal{M}_i} c(f_i^{sa})\mu^{sa}$:

$$R[c] = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{(s,a)\in\mathcal{M}_i} c(f_i^{sa})\mu_i^{sa} - \min_{\mu\in\mathcal{G}_i}\{\sum_{(s,a)\in\mathcal{M}_i}(c(f_i^{sa}) - l_i^{sa})\mu^{sa}\}\right). \tag{23}$$

As before, this functional sums over all examples the difference between the cumulative cost of the i^{th} example policy $\sum_{(s,a)\in\mathcal{M}_i} c(f_i^{sa})\mu_i^{sa}$ and the cumulative cost of the (loss-augmented) minimum cost policy $\min_{\mu\in\mathcal{G}_i}\{\sum_{(s,a)\in\mathcal{M}_i}(c(f_i^{sa}) - l_i^{sa})\mu^{sa}\}$. Since the example policy is a valid policy, the minimum cost policy

[10]Technically $\phi(u)$ has a singularity at $u = 0$. However, since the left and right limits at that point both equal 1, without loss of generality we can define $\phi(0) = 1$ to attain a continuous function.

Algorithm 5 Exponentiated functional gradient descent for maximum margin planning

```
 1: procedure LEARCH( training data $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$, loss function $l_i$, feature function $f_i$ )
 2:     Initialize log-costmap to zero: $s_0 : \mathbb{R}^d \to \mathbb{R}, s_0 = 0$
 3:     for $t = 0, \ldots, T-1$ do
 4:         Initialize the data set to empty: $\mathcal{D} = \emptyset$
 5:         for $i = 1, \ldots, N$ do
 6:             Compute the loss-augmented costmap $c_i^l = e^{s_t(F_i)} - l_i^T$ and find the minimum cost loss-
                    augmented path through it $\mu_i^* = \arg\min_{\mu \in \mathcal{G}_i} c_i^l \mu$
 7:             Generate positive and negative examples: $\forall (s,a) \in \mathcal{M}_i, \mathcal{D} = \{\mathcal{D}, (f_i^{sa}, 1, \mu_i^{*sa}), (f_i^{sa}, -1, \mu_i{}^{sa})\}$
 8:         end for
 9:         Train a regressor or classifier on the collected data set $\mathcal{D}$ to get $h_t$
10:         Update the log-hypothesis $s_{t+1} = s_t + \alpha_t h_t$
11:     end for
12:     return Final costmap $e^{s_T}$
13: end procedure
```

will always be smaller. Each example's objective term (the i^{th} term) is, therefore, always nonnegative. It represents the degree to which the example policy is suboptimal under the hypothesized cost function.

The linear setting typically includes an explicit L_2 regularization term; we remove the regularization term in this expression to simplify the requirements on the corresponding optimization algorithm. Boosting-type functional gradient descent procedures often admit regularization path arguments of the type discussed in (Rosset et al., 2004). These arguments state that the number of boosting steps executed determines the effective size or complexity of the model class under consideration. Early stopping, therefore, plays a similar role to regularization, and we need not explicitly represent the regularization term.[11]

This section discusses the algorithm that arrises when the exponentiated functional gradient decent algorithm we presented in section 3.2 is applied to optimizing this functional. We demonstrate that the resulting class algorithms is that discussed intuitively in section 1.2. Moreover, section 4.3 presents a specific instantiation of the class that has a similar efficient representation to the original linear MMP algorithm. However, we demonstrate experimentally that the algorithm can significantly outperform the older variant.

4.1 General setting

Using the tools described in section 3.1, we can derive the L_2 functional gradient of the maximum margin planning objective functional (see equation 23) as

$$\nabla_f R[c] = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{(s,a) \in \mathcal{M}_i} \mu_i^{sa} \delta_{f_i^{sa}} - \sum_{(s,a) \in \mathcal{M}_i} \mu_i^{*sa} \delta_{f_i^{sa}} \right). \tag{24}$$

In this expression, we denote $\mu_i^* = \arg\min_{\mu \in \mathcal{G}_i} \{\sum_{(s,a)\in\mathcal{M}_i} (c(f_i^{sa}) - l_i^{sa})\mu^{sa}\}$; we call this quantity the optimal loss-augmented policy.

The functional gradient has the same form as that considered in section 3.1: it is a weighted sum of delta (impulse) functions $\sum_j \gamma_j \delta_{x_j}$. In this case, magnitude of a given weight is determined by the frequency count at that state-action pair, and the sign of the weight is determined by whether it comes from the loss-augmented policy or the example policy.

[11] Additionally, boosting relies implicitly on the class of weak learning algorithms to induce generalization and limit hypothesis complexity. Strongly regularized weak learners induce slower complexity growth.

4.2 Intuition

Algorithm 5 details the LEARCH algorithm. This listing demonstrates explicitly how to implement the operation of finding a direction function that correlates well with the functional gradient. Intuitively, the functional gradient can be viewed as a weighted classification or regression data set, where weights come from the magnitude of the delta function coefficients in the gradient term, and the label comes from the sign of these coefficients.

At each iteration, the exponentiated functional gradient algorithm starts by finding a direction function, defined over the feature space, that correlates well with the negative functional gradient. Intuitively, this means that the function is positive in regions of the feature space where there are many positive delta functions (in the negative gradient) and negative in regions where there are many negative delta functions. It then adds this direction function to the log of the previously hypothesized cost function with a small scalar step size α_t. (The step size may decrease toward zero over time as discussed in section 2.3.) Adding the direction function to the cost function effectively increases and decrease the hypothesis as dictated by the impulse signals found in the negative functional gradient. Finally, the algorithm exponentiates the modified log-hypothesis to arrive at a valid positive cost function.

Intuitively, the negative functional gradient places negative impulses at feature vectors found along state-action pairs seen while executing the example policy so that the cost function is decreased in those regions. Conversely, it places positive impulses at feature vectors found along state-action pairs encountered while executing the loss-augmented policy so that the cost function is increased in those regions. In both cases, the magnitude of each impulse is proportional to the frequency with which the relevant policy traverses the state-action pair. If the distribution of feature vectors seen by both the example policy and the loss-augmented policy coincide, then the positive and negative impulses cancel resulting in no net suggested update. However, if the distributions diverge, then the algorithm will decrease the cost function in regions of the feature space where the example policy dominates and increase the cost function in regions where the loss-augmented policy (erroneously) dominates.

We have already seen this algorithm in section 1.3, where we motivated it from a practical standpoint for the specific case of deterministic planning. In some problems, we do not require the cost function to be positive everywhere. For those cases, we may simply apply the more traditional non-exponentiated variant (i.e. gradient boosting (Mason et al., 1999)). Section 5 describes experiments using both exponentiated and non-exponentiated variants of the algorithm on two imitation learning problems from the field of robotics.

4.3 A log-linear variant

The mathematical form of the cost function learned under the LEARCH framework is dictated by the choice of the regressor or classifier used to implement the functional gradient projection step. In this section, we look at the simplest case of applying linear regression to approximate the functional gradient– this often represents a simple, efficient, and effective starting point even when additional non-linear functional gradient approximations are to be applied.

Since a linear combination of linear functions is also a linear function, the final cost function has an efficient log-linear representation

$$f_k(x) \;=\; e^{\sum_{t=1}^k \alpha_t h_t(x)} \;=\; e^{\sum_{i=1}^k \alpha_t u_t^T x} \;=\; e^{w_t^T x},$$

where $w_k = \sum_t \alpha_t u_t$. Moreover, exponentiating the linear function creates a hypothesis space of cost functions with substantially higher dynamic ranges for a given set of features than our original linear alternative which we presented in section 2.3. We find that this log-linear variant demonstrates empirically superior performance.

4.3.1 Deriving the log-linear variant

We derive this variant of LEARCH simply by choosing a set of linear functions $h(x) = w^T x$ as the direction set. The following theorem presents the update rule resulting under the least-squares functional gradient correlation estimator.

Theorem 4.1: *Let w_t be the hypothesized weight vector at the t^{th} time step. Then the update rule with step size η_t under the least-squares correlation estimator of equation 18 takes the form $w_{t+1} = w_t - \eta_t C_t^{-1} g_t$ where g_t is the parametric Euclidean gradient given in equation 9 and*

$$C_t = \sum_{i=1}^{N} F_i \ \mathrm{diag}\left(\mu_i + \mu_i^*\right) \ {F_i}^T.$$

Specifically, our hypothesis at time T takes the form $c^T_{\mathcal{M}_i}(\mu) = e^{-\left(\sum_{t=1}^{T} \eta_t C_t^{-1} g_t\right)^T F_i \mu}$.

Proof. We prove the theorem for the general objective discussed in section 3.1. Applying this result to equation 23 completes the proof. Given a linear hypothesis space, the least-squares fucntional gradient correlation estimator induces the following quadratic objective function:

$$\begin{aligned}\left\langle h_w, \ \sum_{j=1}^{k} \alpha_j x_j \right\rangle - \frac{1}{2}\sum_{j=1}^{k} |\alpha_j| h_w(x_j)^2 \quad &= \quad \sum_{j=1}^{k} \alpha_j w^T x_j - \frac{1}{2}\sum_{j=1}^{k} |\alpha_j| (w^T x_j)^2 \\ = w^T \sum_{j=1}^{k} \alpha_j x_j - \frac{1}{2}\sum_{j=1}^{k} |\alpha_j| w^T (x_j x_j^T) w \quad &= \quad w^T \sum_{j=1}^{k} \alpha_j x_j - \frac{1}{2} w^T \left(\sum_{j=1}^{k} |\alpha_j| x_j x_j^T \right) w.\end{aligned}$$

Since this expression is quadratic, we can solve for the optimal update direction by setting its gradient to zero:

$$\begin{aligned}\nabla \left(w^T \sum_{j=1}^{k} \alpha_j x_j - \frac{1}{2} w^T \left(\sum_{j=1}^{k} |\alpha_j| x_j x_j^T \right) w \right) &= \sum_{j=1}^{k} \alpha_j x_j - \left(\sum_{j=1}^{k} |\alpha_j| x_j x_j^T \right) w = 0 \\ \Rightarrow w = C^{-1} \sum_{j=1}^{k} \alpha_j x_j.\end{aligned}$$

where $C = \sum_{j=1}^{k} |\alpha_j| x_j x_j^T$. Since each α_j is implicitly a function of w, C is also a function of w, and we can therefore view C as an adaptive whitening matrix. □

One may view this modified search direction as the parametric gradient taken under the Riemannian metric C_t. Under the MMP functional, this Riemannian metric adapts to the current combined distribution of feature vectors included by the example and loss-augmented policies. Importantly, using the modified correlation criterion removes the feature scaling problem discussed in (Neu & Szepesvari, 2007) referenced in section 3.1.

4.3.2 Log-linear LEARCH vs linear MMP

Figure 6 depicts a straightforward example of where the log-linear LEARCH algorithm is able to substantially outperform linear MMP. The leftmost panel shows an overhead satellite image depicting a test region, held out from the training set, which we use to evaluate both algorithms. The feature set for this problem consisted solely of Gaussian smoothings of the original grayscale overhead satellite images. We purposefully chose these features to be simple to emphasize the performance differences. The linear MMP algorithm failed to generalize the expert's behavior to the test region (rightmost panel). The best linear combination of features found by linear MMP defined a cost function with very small dynamic range and implemented a naïve minimum distance policy. However, when allowed to exponentiated the linear combination of features, within 6 iterations, the log-linear LEARCH algorithm (center panel) successfully converged to an expressive cost function that generalized the behavior well.

Additionally, figure 7 depicts the difference in validation performance between log-linear LEARCH and linear MMP on a more realistic problem using a stronger feature set, including color class, multispectral, and texture features. For this experiment, we optimized the linear MMP objective using functional gradient boosting of linear hypotheses and satisfied cost-positivity constraints by truncating the costs to a minimum value. Log-linear LEARCH significantly outperforms linear MMP on this problem because of the increased dynamic range in its hypothesis space.

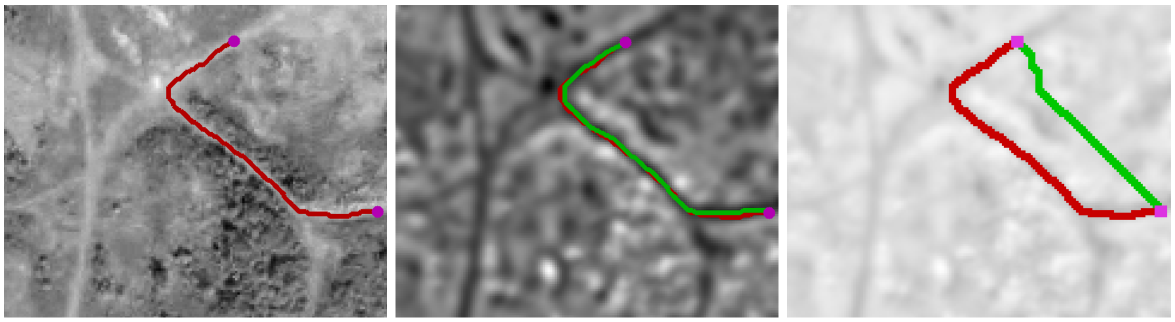


Figure 6: The LEARCH framework suggests a log-linear algorithm which can be used as an alternative to linear maximum margin planning (MMP). The cost functions in the log-linear variant's hypothesis space generally achieve higher dynamic ranges for a given feature set and, therefore, tend to show empirically superior performance. This figure compares the two algorithms on a simple application using the holdout region shown in the leftmost panel. The rightmost panel shows the planning performance on the best linear combination of features achieved by linear MMP, and the center panel shows best exponentiated linear combination of features found by the log-linear LEARCH algorithm. The log-linear algorithm generalizes the expert's behavior well and clearly outperforms linear MMP on this problem.

4.4 Implementation with Different Planners

At a high level, the implementation of LEARCH is straight forward, and simply follows the procedure described in Algorithm 5. However, in practice there are additional details related to the choice of algorithms for regression and planning, as well as the exact form and quality of human training input available. Often, these details are not free parameters, but rather are constraints imposed by a specific problem or platform.

LEARCH requires determining the frequency counts for each state for both the example plan and plan under the current cost function. For some planners, this is relatively straightforward; for example, with a basic A* planner, each state along each path is visited once. However, for other planners, it is not so simple. Since the goal of this work is to learn the proper cost function *for a specific planner*, planner details must be taken into account.

In motion planning, planners often apply a configuration space expansion to an input cost map before generating a plan, to account for the physical dimensions of a robot. When such an expansion does take place, it must also be taken into account by LEARCH; the cost function must be learned such that *after* expansion, the desired behavior is reproduced. For example, if the planner takes the average cost of all states within a window around the current state, then the frequency count of all the states within the same window must be incremented. If the expansion applies different weightings to different states, then this weighting must also be applied when incrementing the frequency counts. Finally, if the expansion applies a weighting to a set of states that is a function of the underlying cost values (for example, the max of the set) this same weighting must still be applied; however, in this case LEARCH will no longer be convex.

More complicated planners may also require non-unit changes to the frequency counts, regardless of an expansion. Fundamentally, LEARCH works on a discrete state space; even if the state space itself is not discretized by the planner, in practice it must be discretized in order to assign features to each state. For planners that operate in a continuous state space, attention must be paid during discretization of the plan. For example the interpolated A* algorithm (Ferguson & Stentz, 2006) used in Section 6 generates paths that are interpolated across 2D cell boundaries. Therefore, the distance traveled through a cell may be anywhere in the interval $(0, \sqrt{2}]$ (in cells). As the cost of traveling through a cell under this scheme is proportional to the distance traveled through the cell, the frequency counts must be incremented proportional to the distance traveled through the cell.

Many planner implementations mandate a specific minimum cost. Since LEARCH ensure positivity, costs can often be scaled to satisfy these contraints. When computing the proper scaling is not possible for

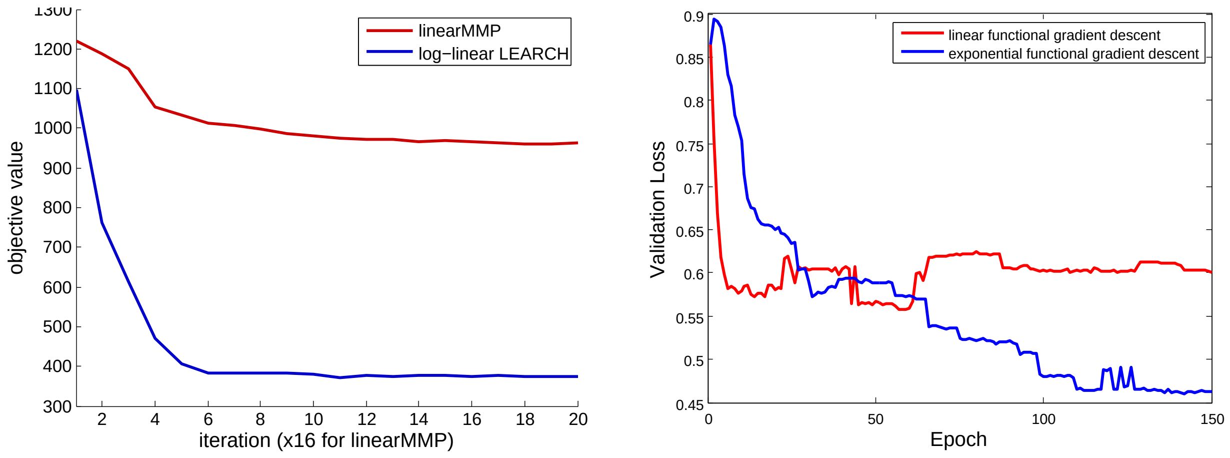


Figure 7: **Left:** Objective values obtained for the problem depicted in figure 6 under log-linear LEARCH (blue) and linear MMP (red), the latter optimized using the subgradient method with approximate projection (see section 2.3.1). The linear MMP plot is scaled to fit on the graph, although it represents 300 iterations of the algorithm (16 iterations per point). The log-linear LEARCH algorithm converged to a substantially better objective value within 20 iterations. **Right:** Performance on a validation set of log-linear LEARCH vs. linear MMP for a similar overhead navigation problem with more realistic features (see section 6). Linear MMP was optimized using functional gradient boosting of linear hypothesis.

practical reasons (for example, in systems with replanning (Ferguson & Stentz, 2006)), truncating the cost function serves as a straightforward alternative.

4.5 Regressor Selection

The choice of regressor for approximating the functional gradient can have a large impact on performance. Simple regressors may generalize better, and complex regressors may learn more complicated cost functions. This tradeoff must be effectively balanced. As in standard classification task, one way to accomplish this is to observe the performance of multiple regressors on an independent validation set of example paths.

Linear regressors offer multiple advantages in this context, as discussed in Section 4.3. Aside from simplicity and good generalization, an extra advantage to linear regressors is computational. Since all hypothesis cost functions are linear, cost evaluations within the LEARCH inner loop do not increase in computational cost. Further, inexpensive cost evaluations are very important if the cost function is to be applied on a real-time robotic system.

We have previously investigated a feature inducing algorithm in the LEARCH framework ((Ratliff et al., 2006b)). The LEARCH algorithm presented here is both simpler and reliably converges faster. More importantly, it also tends to converge on superior solutions. The concept of a feature learning phase can be easily applied to LEARCH. Linear regressors can be used until there is no further progress, at which point a single nonlinear regression is performed. However, instead of adding this nonlinear regressor directly to the cost function, it is added as a new feature for future linear regressors. In this way, future iterations can tune the contribution of these occasional nonlinear steps, and the ammount of nonlinearity in the final cost function is tightly controlled.

4.6 Unachievable Examples

Derivations up to now have operated under the assumption that example paths are either optimal or are just slightly sub-optimal, with the error being taken into the slack variables in the optimization problem.

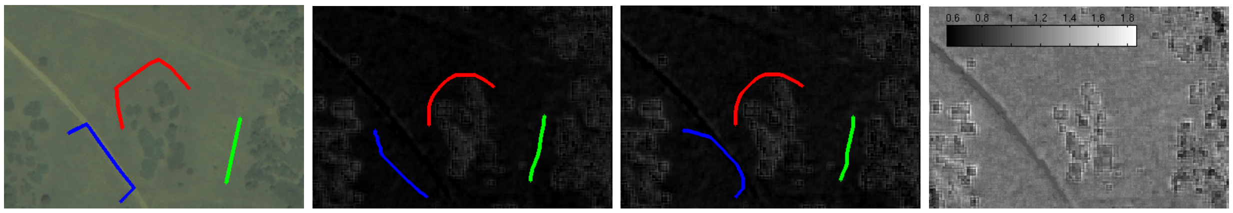


Figure 8: **Left:**The red path is an unachievable example path, as it will be less expensive under any cost function to cut more directly accross the grass. **Center Left:** With standard weighting, the unachievable example forces down the cost of grass, and prevents the blue example from being achieved. **Center Right:** Normalized weighting prevents this bias, and the blue path is achieved **Right:** The ratio of the normalized to the standard costmap. The cost of grass is higher in the normalized map.

However, in practice a good deal of example demonstration is sufficiently sub-optimal as to negatively impact generalization. The next two sub-sections present techniques that can provide increased robustness to sub-optimal example behavior.

In practice, it is often possible to generate an example path such that no consistent cost metric in the hypothesis space will make the example optimal. Figure 8 provides a simple example of such a case. The red example path takes a very wide berth around some obstacles. The features of all the terrain the path travels through are essentially the same, and so will have similar cost. If the spacing the path gives the obstacles is greater than the size of the configuration space expansion of the planner, no cost function exists under which the example is optimal; it will always be lower cost to cut more directly around the obstacle.

Such a case occurs in a small way on a high percentage of human provided examples. People rarely exhibit perfectly optimal behavior, and often generate examples that are a few percent longer than necessary. Further, a limited set of features and planner limitations mean that the path a human truly considers to be optimal may not be achievable using the planning system and features available to learn with.

It is instructive to observe what happens to the functional gradient with an unachievable example. Imagine an environment where all states are described by the identical feature vector f'. Under this scenario, equation 24 reduces to

$$\nabla_f R[c] = \begin{cases} \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{(s,a)\in\mathcal{M}_i}\mu_i^{sa} - \sum_{(s,a)\in\mathcal{M}_i}\mu_i^{*sa}\right) & \text{if } f = f' \\ 0 & \text{if } f \neq f' \end{cases}$$

the functional gradient depends only on the lengths of the example and current plan, under any cost function. If the paths are not of equal length, then the optimization will never be satisfied. Specifcally, if the example path is too long, there will always be an extra component of the gradient that attempts to lower costs. Intuitvely, an unachieveable example implies that certain costs should be 0, a condition that is not allowed by many planners. By continually implying that certain costs should be lowered, unachievable examples can counteract the effect of other (achievable) paths, resulting in cost functions with less dynamic range, and that result in poorer performance (see Figure 8).

This effect can be counteracted with a small modification to the procedure of approximating the functional gradient proposed in (Silver et al., 2008)[12]. Simplifying notation, define the negative gradient vector as

$$-\nabla_f F[c] = U_* - U_e$$

Where U_* and U_e are the frequency counts of the planned and example paths, respectively.

Now consider a new gradient vector that looks at the normalized feature counts

$$-G_{norm} = \frac{U_*}{||U_*||} - \frac{U_e}{||U_e||} \tag{25}$$

[12](Silver et al., 2008) propose an L_1 normalization, as opposed to the L_2 normalization derived above, with empirically similar results

In the special case mentioned above, this new gradient would be satisfied with the cost function as is. It can be shown that in all other cases, the new gradient still points in the correct direction.

Theorem 4.2: *The normalized functional gradient G_{norm} has a positive inner product with the unnormalized functional gradient, except when the frequency counts are perfectly correlated.*

Proof.

$$\begin{aligned}\langle -\nabla_f F[c], -G_{norm}\rangle &= \langle U_* - U_e, \frac{U_*}{||U_*||} - \frac{U_e}{||U_e||}\rangle \\ &= \frac{||U_*||^2 - \langle U_*, U_e\rangle}{||U_*||} + \frac{||U_e||^2 - \langle U_*, U_e\rangle}{||U_e||} \\ &= ||U_*|| + ||U_e|| - \langle U_*, U_e\rangle(\frac{1}{||U_*||} + \frac{1}{||U_e||})\end{aligned}$$

By the Cauchy-Schwarz inequality, we can bound $\langle U_*, U_e\rangle$ with $||U_*||||U_e||$

$$||U_*|| + ||U_e|| - \langle U_*, U_e\rangle(\frac{1}{||U_*||} + \frac{1}{||U_e||}) \;\geq\; ||U_*|| + ||U_e|| - ||U_*||||U_e||(\frac{1}{||U_*||} + \frac{1}{||U_e||}) \;=\; 0$$

When $\langle U_*, U_e\rangle$ is not tight against the upper bound

$$\langle -\nabla_f F[c], -G_{norm}\rangle > 0$$

□

It is worth noting from the proof that $\langle -\nabla_f F[c], -G_{norm}\rangle$ is negatively correlated with $\langle U_*, U_e\rangle$. That is, when the example and planned paths pass through states with very different features, the normalized and unnormalized functional gradients are very similar. As the features becomes more similar, the gradients diverge; when the features become identical, the gradients become orthogonal.

In terms of projecting the new gradient $-G_{norm}$ onto the set of hypothesis functions, equation 16 becomes

$$h^* = \arg\max_{h \in \mathcal{H}} \frac{\sum\limits_{y_i > 0} \alpha_i h(x_i)}{\sqrt{\sum\limits_{y_i > 0} \alpha_i^2}} - \frac{\sum\limits_{y_i < 0} \alpha_i h(x_i)}{\sqrt{\sum\limits_{y_i < 0} \alpha_i^2}} \tag{26}$$

With a similar normalization for equation 18.

The net effect of this normalization is to perform a more balanced classification or regression. As long as the learner can still seperate between examples for which the cost should be increased and decreased, the normalized result will have the same effect. When the learner can no longer distinguish between the classes, the gradient will shrink and eventually become zero, as opposed to simply moving in the direction of the more populous class (see Figure 8).

4.7 Alternate Constraints from Example Behavior

Often, individual examples are achievable, but inconsistent with other examples. When examples are inconsistent it is often by a small amount, due to the inherent noise in human provided example behavior. For example, in constrained environments human examples sometimes give obstacles too a narrow berth (with respect to the expansion of the planner), incorrectly implying the obstacle should be lower cost.

One approach to increasing robustness to this form of noise is to slightly redefine the constraints on cost functions implied by each example. Instead of considering an example path as indicative of the exact optimal plan, it can be considered instead as the approximately optimal plan. That is, instead of trying to enforce the constraint that no path is lower cost then the example, enforce the constraint that no path is lower cost than the lowest cost path that is close to the example. For example, for planning in a state space of $\mathbb{R}^n$, this alternate formulation would enforce the constraint that no path is lower cost than the lowest cost path that exists completely within a corridor (of some specified width) around the example. Formally, the functional gradient becomes

$$\nabla_f R[c] = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{(s,a)\in\mathcal{M}_i} \mu_i^{\epsilon * sa} \delta_{f_i^{sa}} - \sum_{(s,a)\in\mathcal{M}_i} \mu_i^{* sa} \delta_{f_i^{sa}} \right). \tag{27}$$

where $\mu_i^{\epsilon *}$ is defined as

$$\mu_i^{\epsilon *} = \arg \min_{\mu \in \mathcal{G}_i^{\mu_i,\epsilon}} \{ \sum_{(s,a)\in\mathcal{M}_i} (c(f_i^{sa}))\mu^{sa} \}$$

and $\mathcal{G}_i^{\mu_i,\epsilon}$ is the set of all paths close to the example. $\mathcal{G}_i^{\mu_i,\epsilon}$ can be defined in any consistent way; one natural choice is to define $\mathcal{G}_i^{\mu_i,\epsilon}$ as the set of paths with sufficiently small loss.

The result of this new set of constraints is that the gradient is not affected by small details. In effect, this acts as a smoother that can reduce noise in the examples. However, if $|\mathcal{G}_i^{\mu_i,\epsilon}|$ is too large, informative training information can be lost.The optimal threshold for defining $\mathcal{G}_i^{\mu_i,\epsilon}$ can be determined by maximizing validation set performance.

One downside of this new formulation is that the objective functional is no longer convex (Since $\mu_i^{\epsilon *}$ also contains an $\arg\min$). It is certainly possible to construct cases with a single path where this new formulation will converge to a very poor cost function. However, empirical observation has shown that as long as more than a few examples are used, all the local minima achieved are quite satisfactory (Silver et al., 2008).

More complicated versions of this alternative formulation are also possible. For instance, instead of a fixed width corridor along an example path, a specified variable width corridor could be used. This would allow examples with high importance at some sections (at pinch points) and low importance elsewhere. Another version of this formulation would involve multiple disjoint corridors. This could be used if an expert believed there were two different but equally desirable "best" behaviors from start to goal.

5 Case study: Multiclass classification

In this section, we demonstrate LEARCH on two multiclass classification problems: footstep prediction and grasp prediction. These experiments used single-hidden-layer neural networks to implement the functional gradient approximation step in line 9 of algorithm 5.[13]

5.1 Footstep prediction

Recent work has demonstrated that decomposing legged locomotion into separate footstep planners and execution controllers is an effective strategy for many problems (Chestnutt et al., 2003; Chestnutt et al., 2005). The footstep planner finds a sequence of feasible footstep across the terrain, and the execution controller finds a trajectory through the full body configuration space of the robot that successfully places the feet at those locations. The feasibility of the suggested footstep locations is crucial to the overall success of system. In this experiment, we define and train a greedy footstep planning algorithm for a quadrupedal robot using the functional imitation learning techniques discussed in section 4.1.

Our greedy footstep prediction algorithm chooses the minimum cost next footstep location, for a specific foot, given the current four-foot configuration of the robot and the patch of terrain residing directly below the hypothesized footstep location. The cost is defined to be a function of two types of features: action features and terrain features. A similar experimental setup is discussed in (Ratliff et al., 2007b); we build on those results here by including stronger terrain features designed from nonparametric models of the terrain in these experiments.

[13]In practice, we typically use ensembles of small neural networks to reduce variance. We trained the ensembles simply by training the network k times to get $\mathcal{S} = \{f_i(x)\}_{i=1}^k$ and averaging the results: $f(x) = \frac{1}{k}\sum_{i=1}^k f_i(x)$.

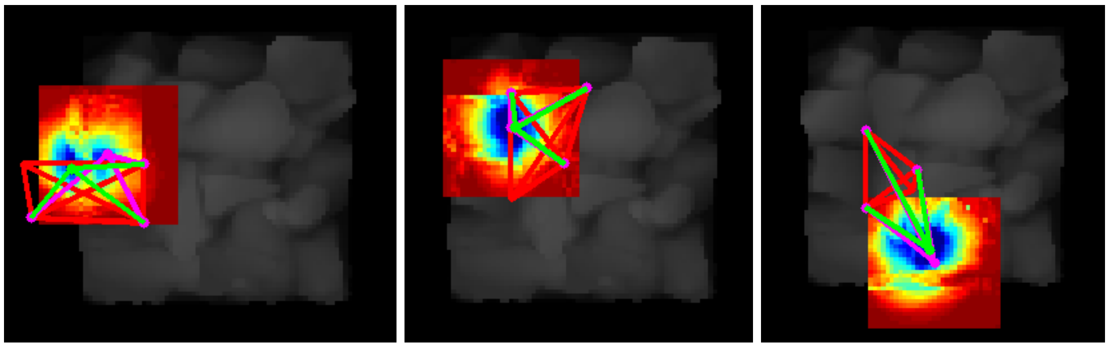


Figure 9: Validation results: Green indicates predicted next footstep, purple indicates desired next footstep, and red indicates current stance. The associated cost function is provided gradating from red at high cost regions to dark blue at low cost regions. The influence of the terrain features on the cost function is most apparent in the left-most prediction whose hypotheses straddle the border between the terrain and flat ground.

Action features encode information about the kinematic demands of the action on the robot and the stability of the stance that results. These features include quantities describing how far the robot must stretch to make the footstep and the size of the support triangle that would result after taking the step These stretch and support triangle features are depicted in figure 11.

On the other hand, the terrain features encode the local shape of the terrain residing directly below the hypothesized next footstep location. In these experiments, we used two types of terrain features. The first set was the responses of a series of Gaussian convolutions to the height map. These features present averages of the terrain heights in a local region at varying resolutions. We derived the second set from the parameters of two locally quadratic regression approximations to the terrain built at two different resolutions. The latter set of features have proven useful on the current robotic platform.

We collected examples of good footstep behavior by teleoperating the quadruped robot shown in figure 1 across the terrain shown in background of the overhead images in figure 10. We trained our footstep predictor with these examples using the non-exponentiated variant of LEARCH. For this experiment, we implemented optimal cost footstep prediction under the cost model described above using a brute force enumeration of a set of 961 feasible next footstep locations from a square region ahead of the foot in question.[14]

The loss function used for this problems was the squared Euclidean distance between the desired footstep location ν_i and the hypothesized footstep location ν: $\mathcal{L}(\nu, \nu_i) = \frac{1}{2}\|\nu - \nu_i\|^2/\sigma^2$. The increased dynamic range of the exponentiated variant of LEARCH allowed us to successfully utilize this relatively simple loss function as hypothesized in (Ratliff et al., 2007b). The experiment depicted in that paper required that the loss function to range only between 0 and 1 in order to successfully generalize under the non-exponentiated variant.

Figure 9 depicts generalization results on a validation set. For each image, the current four-foot configuration is depicted in red, and we compare the desired footstep (green) to the predicted footstep (purple).

We additionally used our trained one-step-lookahead footstep predictor to predict a sequence of footsteps to traverse both rugged and flat terrain. These results are depicted in figure 10. The top-most row shows four consecutive footsteps predicted across a rocky terrain, and the middle row renders the corresponding learned cost function. Our system successfully mimiced the expert's preference for stable cracks in the terrain that were found to induce more robust footholds. The final four images demonstrate the effect of action features alone on footstep prediction by running the predicter over flat ground. Our algorithm successfully learned the kinematic constraints represented in the data.

[14]We computed the offset defining what we mean by "ahead of" relative to the current four foot location so as to be rotationally invariant.

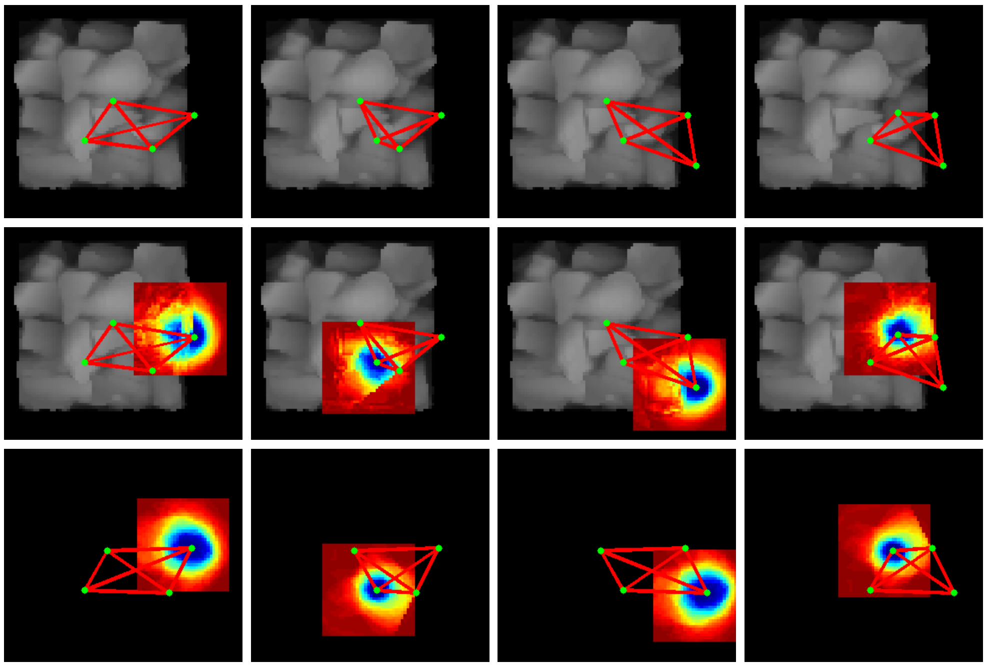


Figure 10: The first two rows show predicted footstep sequences across rough terrain both with and without the corresponding score function. The bottom row demonstrates a predicted sequence for walking across flat ground. Generalization of quadruped footstep placement. The four foot stance was initialized to a configuration off the left edge of the terrain facing from left to right. The images shown demonstrate a sequence of footsteps predicted by the learned greedy planner using a fixed foot ordering. Each prediction starts from result of the previous. The first row shows the footstep predictions alone; the second row overlays the corresponding cost region (the prediction is the minimizer of this cost region). The final row shows footstep predictions made over flat ground along with the corresponding cost region showing explicitly the kinematic feasibility costs that the robot has learned.

5.2 Grasp prediction

We modeled grasp prediction similarly to footstep prediction. Since there are many factors, including intent, that could potentially determine from which direction a robotic hand should approach a given object, we chose to assume the approach direction has already been provided by another system. The task is then to choose the best high-dimensional grasp configuration along the provided approach direction. We model this decision by discretizing the continuous space of grasp configurations, evaluating the cost of each configuration as a function features that locally describe the relationship between the hand and the object, and choosing the optimal scoring grasp. This grasp planning paradigm parallels that of the popular GraspIt! system (Miller et al., 2003). In total, our predictor chose among $2,496$ grasp candidates.

We applied the exponentiated LEARCH algorithm to generalize the grasping behavior exemplified by a set of training examples demonstrated in simulation by a human expert. The second row of figure 12 depicts select grasp demonstrations for a variety of objects taken from Princeton Shape Database[15]. Since the three fingered robot hand was almost symmetric along the palm axis in some configurations, we chose a loss function that was agnostic to the exact hand orientation. This loss function measured how well the fingertips of the predicted grasp aligned with those of the example grasp for the best matching of fingertips. We provide the exact mathematical expression for this loss function in (Ratliff et al., 2007b).

We purposefully chose relatively simple features for this problem in order to showcase the algorithm, itself, rather than a particular feature set. We derived these features from statistical summaries of distance

[15]`http://shape.cs.princeton.edu/benchmark/`

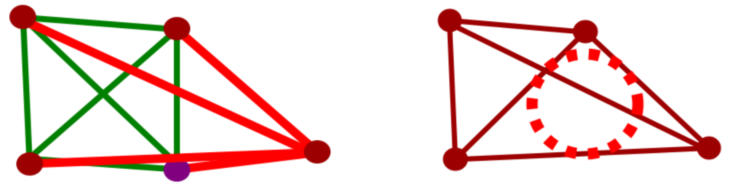


Figure 11: This figure shows some of the action features used for quadrupedal footstep prediction. In the left image, green lines delineate the initial four foot configuration; the purple dot signifies which foot is currently active. The bright red lines connecting each foot in the initial configuration to the hypothesized next foot location represent the "stretch" features. The rightmost figure shows the maximum radius inscribed circle of the support triangle that would result from taking the hypothesized step. We used this radius measure the stability of the support triangle.

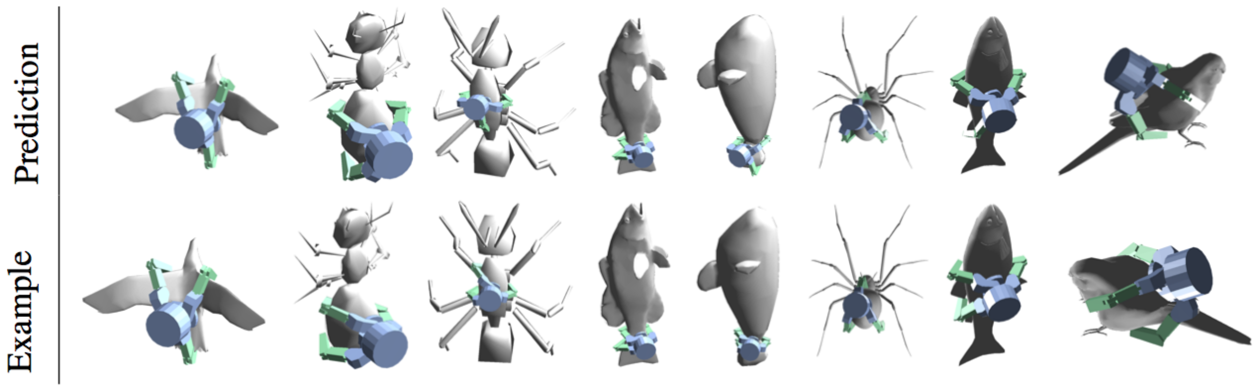


Figure 12: Grasp prediction results on ten holdout examples. The training set consisted of 23 examples in total; we generated each test result by holding out the example in question and training on the remaining 22.

and surface normal information collected by shooting cones of rays toward the object from the fingertips and the palm. These quantities encode information about the local shape of the object within the grasp.

Figure 12 shows our test predictions (from a holdout set) side-by-side with the grasp the human would have predicted for the problem. The algorithm typically does a very good job in generalizing concepts reminiscent of form closure to these new objects.

6 Case study: Structured imitation learning for outdoor navigation from overhead data

In this section, we demonstrate the application of the LEARCH algorithm to the interpretation of overhead data sources for use in long range navigation. This work is covered in detail in (Silver et al., 2008).

6.1 Outdoor Navigation from Overhead Data

This work was performed in the context of the DARPA UPI program. UPI utilizes a six wheeled, skid steer vehicle capable of traversing extreme terrains (Figure 1). With Crusher, UPI has pursued research into long range navigation through complex, unstructured environments. Crusher is equipped with laser range finders and cameras for on-board perception. A perception system processes this sensory data to produce traversability costs that are merged with any prior cost data into a single map. Due to the dynamic nature of this fused cost map, the Field D* (Ferguson & Stentz, 2006) algorithm is used for real-time global planning.

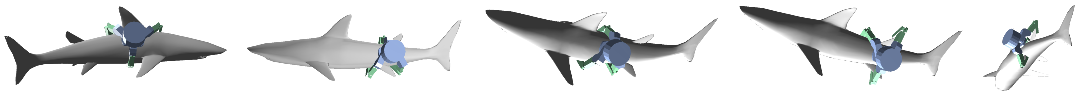


Figure 13: The first three images from the left demonstrate grasp generalization from multiple approach direction on a single object. The final two images show from two perspectives a unique grasp that arises because of the simple feature set. See the text for details.

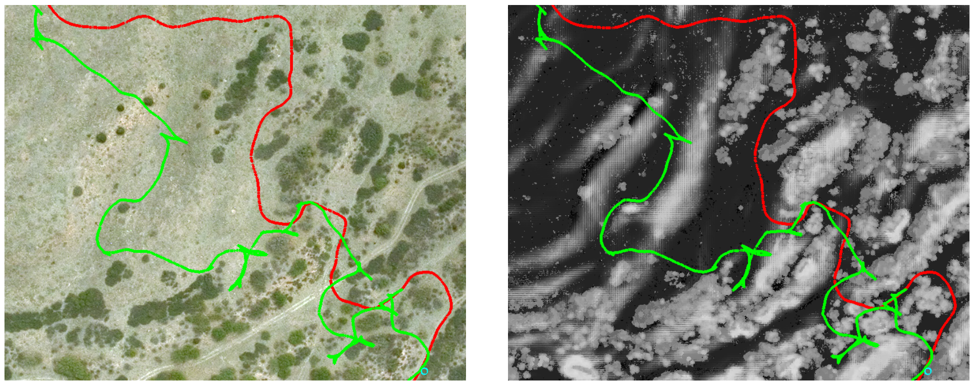


Figure 14: The path traveled during two runs of an autonomous robot moving from top-left to bottom-right. The **green** run had no prior map, and made several poor decisions that required backtracking. The **red** run had the prior costmap shown at right, and took a more efficient route. Brighter pixels indicate higher cost.

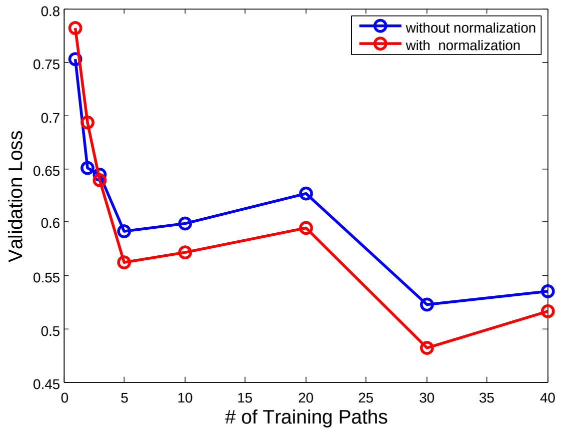
The full UPI Autonomy system is described in (Stentz, 2009).

Overhead terrain data are a popular source of prior environmental knowledge, especially if the vehicle has not previously encountered a specific area. Overhead sources include aerial or satellite imagery, digital elevation maps, and even 3-D LiDAR scans. Much of this data is freely available at lower resolutions, and is commercially available at increased resolution. Previous work (Vandapel et al., 2003) (Silver et al., 2006) has demonstrated the effectiveness of overhead data for use in route planning. Overhead imagery, elevation maps, and point clouds are processed into features stored in 2-D grids. These feature maps can then be converted to traversal costs. Figure 14 demonstrates how this approach can lead to more efficient navigation by autonomous vehicles.

Despite good results, previous work has depended on hand tuned cost functions. That is, an engineer manually determined both the form and parameters of a function to map overhead features to a single cost value, in an attempt to express a desired planning metric. This is somewhat of a black art, requiring a deep understanding of the features being used, the metric to be optimized, and the behavior of the planner. Further, there is often a desire to produce multiple costmaps from different subsets of the available data, to perform comparisons on their effectiveness. Given the variety of data and the need for validation, several days of an engineer's time are usually necessary per site.

6.2 Imitation Learning for Overhead Data

In this context, learning a costmap via LEARCH provides numerous advantages. Only one set of human input is necessary per test site; the same example paths can then be used to learn a costmap from any subset of the available features. The overhead vantage point provides a very intuitive vantage point for demonstrating behavior. A human operator can simply 'draw' an example path from start to goal on top of a visualization of the available data (e.g. an aerial or satellite image). In this way, it is also much simpler to provide examples that imply different cost metrics (Figure 16), as opposed to engineering multiple functions by hand.



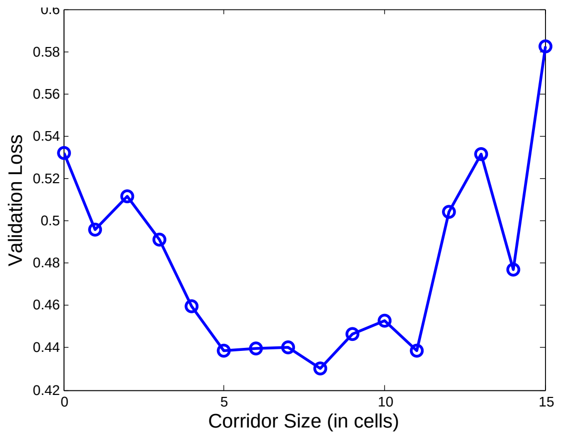


Figure 15: **Left:**Validation loss as a function of the number of training examples. Using a normalized gradient produces better validation loss. **Right:** Validation loss as a function of the corridor size. Using corridor contraints improves performance as long as the corridor is not too large.

Since global planning on UPI is achieved using Field D*, many of the details discussed in Section 4.4 came into play when applying LEARCH. Specifically, Field D* is an interpolated path planning algorithm, and so vistation counts must be computed with respect to actual distance traveled. Additionally, a configuration space expansion is performed by the planner, averaging all costs within a set radius of the current cell. Furthermore, in an attempt to reduce noise in the drawing of examples, corridor constraints were used as described in section 4.7, along with a normalized gradient as described in section 4.6. Experimental validation of these approaches is presented in Figure 15. The loss function used was a function of euclidean distance between states on the current path and the closest state on the example

$$\mathcal{L}(\mu, \mu_i) = \frac{1}{|\mu|} \sum_{s \in \mu} 1 - \exp\left(\min_{s_i \in \mu_i} [\|s - s_i\|^2]/\sigma^2\right)$$

LEARCH was implemented with 4 varieties of regressors: linear, neural networks, regression trees, and linear with a feature learning phase using regression trees (see section 4.5). Overall, the latter approach was found to provide superior all around performance. Specifically, the computational advantage of log-linear LEARCH is quite desireable when rapid training was required. Additionally, using non-linear regressors often resulted in overfitting; performance was better on the training set but worse on a validation set. In contrast a linear cost function with a handfull of simple regression trees provided good performance on both training and validation sets. It is interesting to note that at the conceptual level this approach is similar to the types of cost functions that were often generated by human engineering: a generally linear cost function, with a few special cases handled differently.

6.3 Experimental Results

Numerous offline experiments were performed to compare performance of different varieties of LEARCH applied to overhead data, and to compare its performance to the hand-tuned approach. One experiment of note compared the generalization of LEARCH with that of a hand-tuned costmap. A cost map was trained off of satellite imagery for an approximately 60 km^2 size area at 60 cm resolution. An engineered costmap had been previously produced for this same area to support UPI. A subset of both maps is shown in Figure 17. The two maps were compared using a validation set of paths generated by a UPI team member not directly involved in the development of overhead costing. The average validation loss was 0.675 with the engineered map, and 0.551 with the LEARCH map.

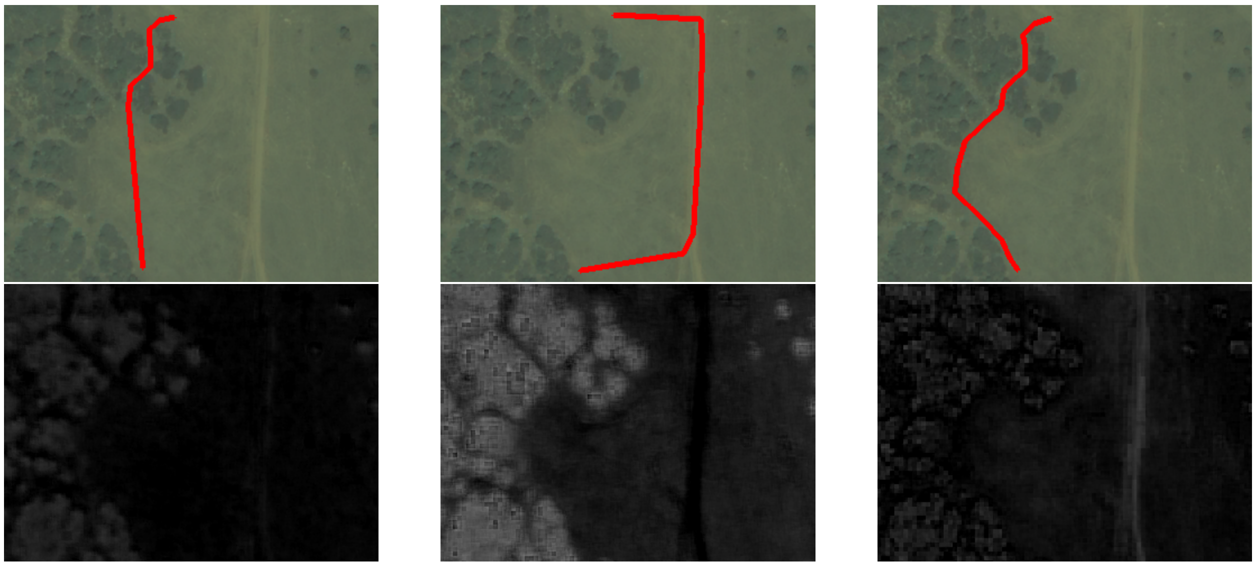


Figure 16: **Top:** Example paths that imply different metrics (From top to bottom: minimize distance traveled, stay on roads, stay near trees) **Bottom:** Learned costmaps from the corresponding examples (brighter pixels indicate higher cost). Imagery from the Quickbird satellite courtesy of Digital Globe, Inc.

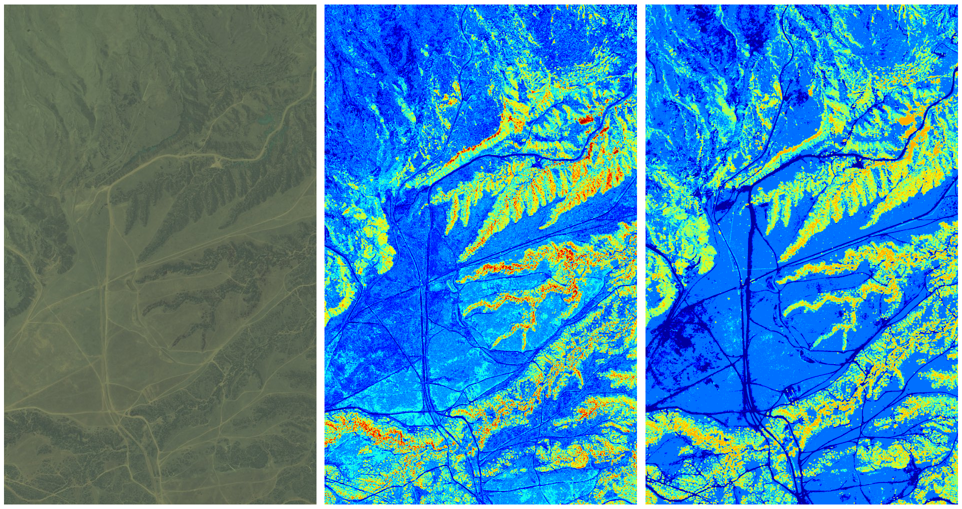


Figure 17: A 10 km^2 section of a UPI test site. From left to right: satellite imagery, LEARCH Cost, and Engineered Cost. Costmaps are Jet-Colored to highlight differences.Prominent features include roads, trails, grass, bushes, trees, and washes.

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:

Table 2: Results of experiments comparing learned to engineered prior maps. Indicated costs are from Crusher's onboard perception system.

Additional validation was also achieved during official UPI field tests. UPI field tests consist 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). Satellite Imagery and aerial fixed wing LADAR scans were used as input to multiple feature extractors and classifiers. These features were then fed into a hand tuned cost function. An initial implementation of the MMP algorithm was also field tested and demonstrated in smaller tests during 2006. During 2007 and 2008, LEARCH became the default apporach 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 testing.

In addition, two direct online comparisons were performed. These two tests were performed several months apart, at different test sites. During these experiments, the same course was run multiple times, varying only the overhead cost map given to Crusher. To score each run, Crusher's highly accurate onboard perception system was used as an estimate of ground truth mobility risk. Crusher's perception system has been engineered[16] over a period of years to enable safe autonomous navigation (even without any overhead data), and makes use of data that is far higher resolution than overhead data. The results of these experiments are shown in Table 2. In both experiments, Crusher traveled farther to complete the same course using LEARCH produced cost maps, and yet was scored with less total mobility risk by Crusher's perception system. Crusher also was able to drive faster on average, an indication of more favorable terrain. The course for Experiment 2 was also run without any overhead data; the results are presented for comparison.

7 Future directions

Our experiments demonstrate that there are a number of robotics problems to which we can apply LEARCH. However, our approach can currently only be applied to problems that can be modeled by MDPs and for which there exist efficient MDP solvers. While efficient planning algorithms often form the backbone of modern robotics systems, as the dimensionality of the configuration space increases, these planners necessarily become approximations and are often stochastic. One of our current directions for future work is to expand

[16]Crusher's perception system provides an ideal scenario for dynamic imitation learning as mentioned in Section 7

our algorithms to these settings utilizing stochastic policy models and recent advances in gradient-based trajectory optimization.

Another issue faced by robotic systems is that most environments are dynamic. In this context, any definition of optimal behavior must consider how the environment is structured and perceived over time. This added complexity makes programming the correct behavior an even greater challenge than in the static case. In the future, we hope to demonstrate the automated interpretation of a robot's onboard perception system by extending LEARCH to this context.

Additionally, we derive our algorithms by reducing the imitation learning problem to structured prediction. While we presented LEARCH particularly for the planning setting, the algorithm applies generally across the class of maximum margin structured classification problems. Efforts to extend the work of (Munoz et al., 2008) to utilize exponentiated functional gradient techniques are currently in progress. Additionally, our exponentiated functional gradient algorithm is an optimization approach that can be applied not only across imitation learning and structured prediction formulations, but also more generally as an alternative to the gradient boosting algorithm. We plan to explore where and in what cases this algorithm displays better performance. We are particularly interested in applications to density estimation which can capitalize on the positivity of the hypothesis space.

Finally, we are eager to expand the theoretical understanding of our algorithm. Theorem 3.1 demonstrates that were we able to follow the exact Euclidean gradient (e.g. of the true expected risk as discussed in (Friedman, 1999a)), then we can always make progression toward optimizing the objective. However, in practice, we can only approximate the functional gradient by projecting it onto a set of feasible direction functions. Additionally, our experiments indicate that we can often converge faster than alternative techniques; characterizing when this algorithm should perform well relative to competing techniques is an interesting direction for future work. We are working to extend some of the extensive theory on boosting techniques to our exponentiated setting.

8 Acknowledgements

The research presented in this paper was funded in part by the DARPA UPI program, the DARPA Learning Locomotion program, and Intel Research. We are grateful for fruitful discussions with Siddhartha Srinivasa, Tony Stentz, David Bradley, Joel Chestnutt, John Langford, Yucheng Low, Brian Ziebart, Martin Zinkevich, Matt Zucker, and the UPI autonomy and vehicle teams.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *ICML '04: Proceedings of the twenty-first international conference on Machine learning.*

Anderson, B. D. O., & Moore, J. B. (1990). *Optimal control : linear quadratic methods / brian d.o. anderson, john b. moore.* Prentice Hall, Englewood Cliffs, N.J. :.

Argall, B., Chernova, S., Veloso, M., & Browning, B. (to appear). A survey of robot learning from demonstration. *Robotics and Autonomous Systems.*

Atkeson, C., Schaal, S., & Moore, A. (1995). Locally weighted learning. *AI Review.*

Bain, M., & Sammut, C. (1995). A framework for behavioral cloning. *Machine Intelligence Agents.* Oxford University Press.

Boyd, S., Ghaoui, L. E., Feron, E., & Balakrishnan, V. (1994). *Linear matrix inequalities in system and control theory.* Society for Industrial and Applied Mathematics (SIAM).

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games.* New York, NY, USA: Cambridge University Press.

Chestnutt, J., Kuffner, J., Nishiwaki, K., & Kagami, S. (2003). Planning biped navigation strategies in complex environments. *Proceedings of the IEEE-RAS International Conference on Humanoid Robots.* Karlsruhe, Germany.

Chestnutt, J., Lau, M., Cheng, G., Kuffner, J., Hodgins, J., & Kanade, T. (2005). Footstep planning for the Honda ASIMO humanoid. *Proceedings of the IEEE International Conference on Robotics and Automation.*

Donoho, D. L., & Elad, M. (2003). Maximal sparsity representation via l1 minimization. *the Proc. Nat. Aca. Sci. 100* (pp. 2197–2202).

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

Friedman, J. H. (1999a). Greedy function approximation: A gradient boosting machine. *Annals of Statistics.*

Jaynes, E. (2003). *Probability: The logic of science.* Cambridge University Press.

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

Kelly, A., Amidi, O., Happold, M., Herman, H., Pilarski, T., Rander, P., Stentz, A., Vallidis, N., & Warner, R. (2004). Toward reliable autonomous vehicles operating in challenging environments. *Proceedings of the International Symposium on Experimental Robotics (ISER).* Singapore.

Kivinen, J., & Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, *132*.

Kolter, J. Z., Abbeel, P., & Ng, A. Y. (2008). Hierarchical apprenticeship learning with application to quadruped locomotion. *Neural Information Processing Systems 20.*

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

Mason, L., J.Baxter, Bartlett, P., & Frean, M. (1999). Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers.* MIT Press.

Miller, A. T., Knoop, S., Allen, P. K., & Christensen, H. I. (2003). Automatic grasp planning using shape primitives. *Proceedings of the IEEE International Conference on Robotics and Automation.*

Munoz, D., Vandapel, N., & Hebert, M. (2008). Directional associative markov network for 3-d point cloud classification. *Fourth International Symposium on 3D Data Processing, Visualization and Transmission.*

Neu, G., & Szepesvari, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. *Uncertainty in Artificial Intelligence (UAI).*

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

Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems 1.*

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming.* Wiley.

Ratliff, N., Bagnell, J. A., & Zinkevich, M. (2006a). Maximum margin planning. *Twenty Second International Conference on Machine Learning (ICML06).*

Ratliff, N., Bagnell, J. A., & Zinkevich, M. (2007a). (Online) subgradient methods for structured prediction. *Artificial Intelligence and Statistics.* San Juan, Puerto Rico.

Ratliff, N., Bradley, D., Bagnell, J. A., & Chestnutt, J. (2006b). Boosting structured prediction for imitation learning. *NIPS.* Vancouver, B.C.

Ratliff, N., Srinivasa, S., & Bagnell, J. A. (2007b). Imitation learning for locomotion and manipulation. *IEEE-RAS International Conference on Humanoid Robots.*

Rifkin, Y., & Poggio (2003). Regularized least squares classification. *Advances in Learning Theory: Methods, Models and Applications.* IOS Press.

Rosset, S., Zhu, J., & Hastie, T. (2004). Boosting as a regularized path to a maximum margin classifier. *J. Mach. Learn. Res.*, *5*, 941–973.

Schaal, S., & Atkeson, C. (1994). Robot juggling: An implementation of memory-based learning. *IEEE Control Systems Magazine*, *14*.

Shor, N. Z. (1985). *Minimization methods for non-differentiable functions.* Springer-Verlag.

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

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

Stentz, A. (2009). Autonomous navigation for complex terrain. Carnegie Mellon Robotics Institute Technical Report, manuscript in preparation.

Taskar, B., Chatalbashev, V., Guestrin, C., & Koller, D. (2005). Learning structured prediction models: A large margin approach. *Twenty Second International Conference on Machine Learning (ICML05).*

Taskar, B., Guestrin, C., & Koller, D. (2003). Max margin markov networks. *Advances in Neural Information Processing Systems (NIPS-14).*

Taskar, B., Lacoste-Julien, S., & Jordan, M. (2006). Structured prediction via the extragradient method. In *Advances in neural information processing systems 18*. MIT Press.

Tropp, J. A. (2004). Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory*, *50*, 2231–2242.

Vandapel, N., Donamukkala, R. R., & Hebert, M. (2003). Quality assessment of traversability maps from aerial lidar data for an unmanned ground vehicle. *Proceedings of the IEEE/JRS International Conference on Intelligent Robots and Systems.*

Ziebart, B., Bagnell, J. A., Mass, A., & Dey, A. (2008). Maximum entropy inverse reinforcement learning. *Twenty-third AAAI Conference.*

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. *Proceedings of the Twentieth International Conference on Machine Learning.*