# Efficient Optimization of Control Libraries

**Debadeepta Dey, Tian Yu Liu, Boris Sofman** and **J. Andrew Bagnell**
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

**Abstract**

A popular approach to high dimensional control problems in robotics uses a library of candidate "maneuvers" or "trajectories". The library is either evaluated on a fixed number of candidate choices at runtime (*e.g.* path set selection for planning) or by iterating through a sequence of feasible choices until success is achieved (*e.g.* grasp selection). The performance of the library relies heavily on the content and order of the sequence of candidates. We propose a provably efficient method to optimize such libraries, leveraging recent advances in optimizing sub-modular functions of sequences. This approach is demonstrated on two important problems: mobile robot navigation and manipulator grasp set selection. In the first case, performance can be improved by choosing a subset of candidates which optimizes the metric under consideration (cost of traversal). In the second case, performance can be optimized by minimizing the depth in the list that is searched before a successful candidate is found. Our method can be used in both on-line and batch settings with provable performance guarantees, and can be run in an anytime manner to handle real-time constraints.

## Introduction

Many approaches to high dimensional robotics control problems such as grasp selection for manipulation (Berenson et al. 2007; Goldfeder et al. 2009) and trajectory set generation for autonomous mobile robot navigation (Green and Kelly 2006) use a library of candidate "maneuvers" or "trajectories". Such libraries effectively discretize a large control space and enable tasks to be completed with reasonable performance while still respecting computational constraints. The library is used by evaluating a fixed number of candidate maneuvers at runtime or iterating through a sequence of choices until success is achieved. Performance of the library depends heavily on the *content and order* of the sequence of candidates.

Control libraries have been successfully used in planning for both humanoids (Stolle and Atkeson 2006) and UAVs (Frazzoli, Dahleh, and Feron 2000). Frazzoli et al. show for a planning task, a feasible trajectory can be quickly generated using a concatenation of stored trajectories in the library. This has the additional advantage that the entire concatenated result is dynamically feasible if each element in the control library is feasible. In grasp pose selection, libraries usually contain different grasp poses as elements. To choose the best pose, each element is evaluated for force closure and stability, and the one with the highest score is chosen for execution.(Berenson et al. 2007; Saut and Sidobre 1918; Chinellato et al. 2003; Ciocarlie and Allen 2008).

A fundamental open question is how such control libraries should be constructed and organized in order to maximize performance on the task at hand while minimizing search time.

This class of problems can be framed as *list optimization* problems where the ordering of the list heavily influences both performance and computation time (Sofman, Bagnell, and Stentz 2010). In the grasp selection example the system is searching for the first successful grasp in a list of candidate grasps, performance is dependent on the depth in the list that has to be searched before a successful answer can be found. In the trajectory set generation example the system comes up with a subset of trajectories such that the computed cost of traversal of the robot is minimized.

We show that list optimization problems exhibit the property of *monotone sequence submodularity* (Fujishige 2005; Streeter and Golovin 2008). We take advantage of recent advances in submodular sequence function optimization by (Streeter and Golovin 2008) to propose an approach to high-dimensional robotics control problems that leverages the online and submodular nature of list optimization. These results establish algorithms that are near-optimal (within known NP-hard approximation bounds) in both a fixed-design and no-regret sense. Such results may be somewhat unsatisfactory for the control problem we address as we are concerned about performance on future data and thus we consider two batch settings: static optimality, where we consider a distribution over training examples that are independently and identically distributed (i.i.d), and a form of dynamic optimality where the distribution of examples is influenced by the execution of the control libraries. We show that online-to-batch conversions (Cesa-Bianchi, Conconi, and Gentile 2004) combined with the advances in online submodular function maximization enable us to effectively optimize these control libraries with guarantees.

For the trajectory sequence selection problem, we show that our approach exceeds the performance of the current state of the art by achieving lower cost of traversal in a real-world path planning scenario (Green and Kelly 2006). For grasp selection (related to the MIN-SUM SUBMODULAR COVER problem) we show that we can adaptively reorganize a list of grasps such that the depth traversed in the list until a successful grasp is found is minimized. Although our approach in both cases is online in nature, it can operate in an offline mode where the system is trained using prior collected data and then used for future queries without incorporating additional performance feedback.

## Review of Submodularity and Maximization of Submodular functions

A function $f : \mathscr{S} \to [0,1]$ is monotone submodular for any sequence $S \in \mathscr{S}$ where $\mathscr{S}$ is the set of all sequences if it satisfies the following two properties:

- (Monoticity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1) \leq f(S_1 \oplus S_2)$ and $f(S_2) \leq f(S_1 \oplus S_2)$
- (Submodularity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1)$ and any action $a \in \mathscr{V} \times \mathbb{R}_{>0}$, $f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \langle a \rangle) - f(S_1)$

where $\oplus$ means order dependent concatenation of lists, $\mathscr{V}$ is the set of all available actions and $\mathbb{R}_{>0}$ is the set of non-negative real numbers denoting the cost of each action in $\mathscr{V}$.

In the online setting $\alpha$-regret is defined as the difference in the performance of an algorithm and $\alpha$ times the performance of the best expert in retrospect. (Streeter and Golovin 2008) provide algorithms for maximization of submodular functions whose $\alpha$-regret (regret with respect to proven NP-hard bounds) approaches zero as a function of time.

We review here the relevant parts of the online submodular function maximization approach as detailed by (Streeter and Golovin 2008). Throughout this paper subscripts denote time steps while superscripts denote the slot number. Assume we have a list of feasible control actions $\mathscr{A}$, a sequence of tasks $f_{1 \ldots T}$, and a list of actions of length $N$ that we maintain and present for each task. One of the key components of this approach makes use of the idea of an *expert algorithm*. Refer to survey by (Blum 1996). The algorithm runs $N$ distinct copies of this expert algorithm: $\mathscr{E}^1, \mathscr{E}^2, \ldots, \mathscr{E}^N$, where each expert algorithm $\mathscr{E}^i$ maintains a distribution over the set of possible experts (in this case action choices). Just after task $f_t$ arrives and before the correct sequence of actions to take for this task is shown, each expert algorithm $\mathscr{E}^i$ selects a control action $a_t^i$. The list order used on task $f_t$ is then $S_t = \{a_t^1, a_t^2, \ldots, a_t^N\}$. At the end of step $t$, the value of the reward $x_t^i$ for each expert $i$ is made public and is used to update each expert accordingly.

## Application: Mobile robot navigation

Traditionally, path planning for mobile robot navigation is done in a hierarchical manner with a global planner at the top driving the robot in the general direction of the goal while a local planner avoids obstacles while making progress towards the goal. At every time step the local planner evaluates a set of feasible control trajectories on the immediate perceived environment to find the trajectory yielding the least cost of traversal. The robot then moves along the trajectory, which has the least sum of cost of traversal and cost-to-goal from the end of the trajectory. This process is then repeated at each time step. This set of feasible trajectories is usually computed offline by sampling from a much larger (possibly infinite) set of feasible trajectories.

Such library-based model predictive approaches are widely used in state-of-the-art systems leveraged by most DARPA Urban Challenge, Grand Challenge teams (including the two highest placing teams for both) (Urmson and others 2008; Montemerlo and others 2008; Urmson and others 2006; Thrun and others 2006) as well as on sophisticated outdoor vehicles LAGR *e.g.* (Jackel and others 2006), UPI (Bagnell et al. 2010), Perceptor (Kelly and others 2006) developed in the last decade. A particularly effective automatic method for generating such a library is to generate the set of trajectories greedily such that the area between the trajectories is maximized (Green and Kelly 2006). As this method runs offline, it does not adapt to changing conditions in the environment nor is it data-driven to perform well on the environments encountered in practice.

Let $cost(a^i)$ be the cost of traversing along trajectory $a^i$. Let $N$ be the budgeted number of trajectories that can be evaluated during real-time operation. For a set of trajectories $\{a^1, a^2, ..., a^N\}$ sampled from the set of all feasible trajectories, we define the monotone, submodular function that we maximize using the lowest-cost path from the set of possible trajectories as $f : \mathscr{S} \to [0,1]$:

$$f \equiv \frac{N_o - min(cost(a^1), cost(a^2), \ldots, cost(a^N))}{N_o} \qquad (1)$$

where $N_o$ is the highest cost trajectory that can be expected for a given cost map.

We present the general algorithm for online selection of action sequences in Algorithm 1. The inputs to the algorithm are the number of action primitives $N$ which can be evaluated at runtime within the computational constraints and $N$ copies of experts algorithms, $\mathscr{E}^1, \mathscr{E}^2, ..., \mathscr{E}^N$, one for each slot of the sequence of actions desired. The experts algorithm subroutine can be either Randomized Weighted Majority (WMR) (Littlestone and Warmuth 1994) (Algorithm 3) or EXP3 (Auer et al. 2003) (Algorithm 2). $T$ represents the number of planning steps the robot is expected to carry out. In lines 1-5 of Algorithm 1 a sequence of trajectories is sampled from the current distribution of weights over trajectories maintained by each copy of the expert algorithm. Function $a^i = sampleActionExperts(\mathscr{E}^i)$ samples the distribution of weights over experts (trajectories) maintained by experts algorithm copy $\mathscr{E}^i$ to fill in slot $i$ ($S_t^i$) of the sequence without repeating trajectories selected for slots before the $i^{th}$ slot. The function *sampleActionExperts* in the case of EXP3 corresponds to executing lines 1-2 of Algorithm 2. For WMR this corresponds to executing line 1 of Algorithm 3. Similarly the function *updateWeight* corresponds to executing lines 3-6 of Algorithm 2 or lines 3-4 of Algorithm 3.

Function $a^* = evaluateActionSequence(\mathbf{ENV}, S_t)$ (line 6) of Algorithm 1 takes as arguments the constructed sequence of actions $S_t^i$ and the current environment around the robot **ENV**. Actions in the sequence are evaluated on the environment to find the action which has the least cost of traversal plus cost to go to the goal which is output as $a^*$.

Function $\mathbf{ENV} = getNextEnvironment(a^*, \triangle t)$ (line 7) takes as arguments the best trajectory found earlier ($a^*$) and the time interval ($\triangle t$) for which that trajectory is to be traversed. On traversing it for that time interval the robot reaches the next environment which replaces the previous environment stored in **ENV**.

In lines 8-13 each of the expert algorithms weights over all feasible trajectories are increased if the monotone submodular function $f_t$ is increased by adding trajectory $a^i_j$ at the $i^{th}$ slot.

As a side note, the learning rate $\varepsilon$ for WMR is set to be $\frac{1}{\sqrt{T}}$ where $T$ is the number of planning cycles, possibly infinite. For infinite or unknown planning time this can be set to $\frac{1}{\sqrt{t}}$ where $t$ is the current time step. Similarly the mixing parameter $\gamma$ for EXP3 is set as $min\left\{1, \sqrt{\frac{|\mathscr{A}|\ln|\mathscr{A}|}{(e-1)T}}\right\}$. For each expert algorithm the respective learning rates are set to the rates proven to be no-regret with respect to the best expert in the repertoire of experts. $T$ can be infinite as a ground robot can be run for arbitrary amounts of time with continuously updated goals. Since the choice of $T$ influences the learning rate of the approach it is necessary to account for the possibility of $T$ being infinite.

Note that actions are generic and in the case of mobile robot navigation are trajectory primitives from the control library.

WMR may be too expensive for *online* applications as it requires the evaluation of every trajectory at every slot, whether executed or not. EXP3, by contrast, learns more slowly but requires as feedback only the cost of the sequence of trajectories actually executed, and hence adds negligible overhead to the use of trajectory libraries. For EXP3 line 9 would loop over only the experts chosen at the current time step instead of $|\mathscr{A}|$.

We refer to this sequence optimization algorithm (Algorithm 1) in the rest of the paper as SEQOPT.

**Algorithm 1** Algorithm for trajectory sequence selection

```
Require: number of trajectories $N$, experts algorithms subroutine copies (Algorithms 2 or 3) $\mathscr{E}^1, \mathscr{E}^2, \ldots, \mathscr{E}^N$
1:  for $t = 1$ to $T$ do
2:      for $i = 1$ to $N$ do
3:          $a^i = sampleActionExperts(\mathscr{E}^i)$
4:          $S^i_t \leftarrow a^i$
5:      end for
6:      $a^* = evaluateActionSequence(\mathbf{ENV}, S_t)$
7:      $\mathbf{ENV} = getNextEnvironment(a^*, \triangle t)$
8:      for $i = 1$ to $N$ do
9:          for $j = 1$ to $|\mathscr{A}|$ do
10:             $reward^i_j = f_t(S^{\langle i-1\rangle}_t \oplus a^i_j) - f_t(S^{\langle i-1\rangle}_t)$
11:             $w^i_j \leftarrow updateWeight(reward^i_j, w^i_j)$
12:         end for
13:     end for
14: end for
```

SEQOPT: the approach detailed is an online algorithm which produces a sequence which converges to a greedily sequence with time. The greedy sequence achieves at least $1-1/e$ of the value of the optimal list (Feige 1998). Therefore SEQOPT is a zero $\alpha$-regret (for $\alpha = 1-1/e$ here) algorithm. This implies that its $\alpha$-regret goes to 0 at a rate of $O(1/\sqrt{T})$ for $T$ interactions with the environment.

We are also interested in its performance with respect to future data and hence consider notions of near-optimality with respect to distributions of environments. We define a

**Algorithm 2** Experts Algorithm: Exponential-weight algorithm for Exploration and Exploitation (EXP3) (Auer et al. 2003)

```
Require: $\gamma \in (0,1]$, initialization $w_j = 1$ for $j = 1, \ldots, |\mathscr{A}|$
1: Set $p_j = (1-\gamma)\frac{w_j}{\sum_{j=1}^{|\mathscr{A}|} w_j} + \frac{\gamma}{|\mathscr{A}|}$ $j = 1, \ldots, |\mathscr{A}|$
2: Randomly sample $i$ according to the probabilities $p_1, \ldots, p_{|\mathscr{A}|}$
3: Receive $reward_i \in [0,1]$
4: for $j = 1$ to $|\mathscr{A}|$ do
5:
       $$\hat{reward}_j = \begin{cases} \frac{reward_j}{p_j} & \text{if } i == j \\ 0 & \text{otherwise} \end{cases}$$
6:     $w_j \leftarrow w_j exp(\frac{\gamma \hat{reward}_j}{|\mathscr{A}|})$
7: end for
```

**Algorithm 3** Experts Algorithm: Randomized Weighted Majority (WMR) (Littlestone and Warmuth 1994)

```
Require: $\varepsilon \in (0,1]$, initialization $w_j = 1$ for $j = 1, \ldots, |\mathscr{A}|$
1: Randomly sample $j$ according to the distribution of weights $w_1, \ldots, w_{|\mathscr{A}|}$
2: Receive rewards for all experts $reward_1, \ldots, reward_{|\mathscr{A}|}$
3: for $j = 1$ to $|\mathscr{A}|$ do
4:
       $$w_j = \begin{cases} w_j(1+\varepsilon)^{reward_j} \text{ if } reward_j \geq 0 \\ w_j(1-\varepsilon)^{-reward_j} \text{ if } reward_j < 0 \end{cases}$$
5: end for
```

*statically optimal* sequence of trajectories $S_{so} \in \mathscr{S}$ as:

$$S_{so} = \underset{S}{\operatorname{argmax}} E_{d(\mathbf{ENV})}[f(\mathbf{ENV}, S)] \quad (2)$$

where $d(\mathbf{ENV})$ is a distribution of environments that are randomly sampled. The trajectory sequence $S$ is evaluated at each location. A statically near-optimal trajectory sequence $S_{so}$ thus approximately maximizes the expectation $(E_{d(\mathbf{ENV})}[f(\mathbf{ENV}, S)])$ of Equation 1 over the distribution of environments $\mathbf{ENV}$, effectively optimizing the one-step cost of traversal at the locations sampled from the distribution of the environments.

(Knepper and Mason 2009) note that sequences of trajectories are generally designed for this kind of static planning paradigm but are used in a dynamic planning paradigm where the library choice influences the examples seen and that there is little correlation in performance between good static and good dynamic performance for a sequence. Our approach bridges this gap by allowing offline batch training on a fixed distribution, or allowing samples to be generated by running the currently sampled library.

We define a *weak dynamically optimal* trajectory sequence $S_{wdo} \in \mathscr{S}$ as:

$$S_{wdo} = \underset{S}{\operatorname{argmax}} E_{d(\mathbf{ENV}|\pi)}[f(\mathbf{ENV}, S)] \quad (3)$$

where $d(\mathbf{ENV}|\pi)$ is defined as the distribution of environments that are induced by the robot following the policy $\pi$. The policy $\pi$ corresponds to the robot following the least cost trajectory within $S_{wdo}$ at each situation encountered.

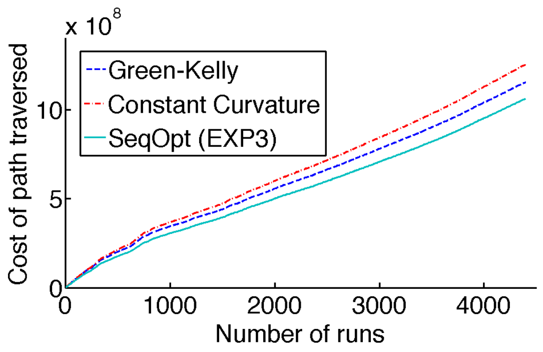


Figure 1: The cost of traversal of the robot on the cost map of Fort Hood, TX using trajectory sequences generated by different methods for sequence of 30 trajectories over 1055788 planning cycles in 4396 runs. Constant curvature trajectories result in the highest cost of traversal followed by Green-Kelly path sets. Our sequence optimization approach (SEQOPT) using EXP3 as the experts algorithm subroutine results in the lowest cost of traversal (8% lower than Green-Kelly) with negligible overhead.

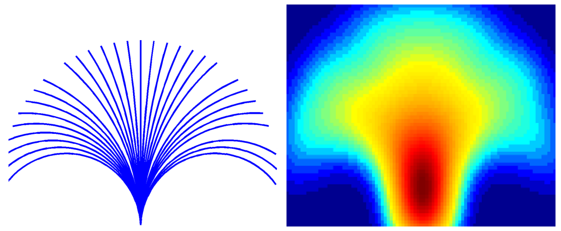
Hence a weak dynamically optimal trajectory sequence minimizes the cost of traversal of the robot at all the locations which the robot encounters *as a consequence of* executing the policy $\pi$. We define this as weak dynamically optimal as there can be other trajectory sequences $S \in \mathscr{S}$ that can minimize the cost of traversal with respect to the distribution of environments induced by following the policy $\pi$.

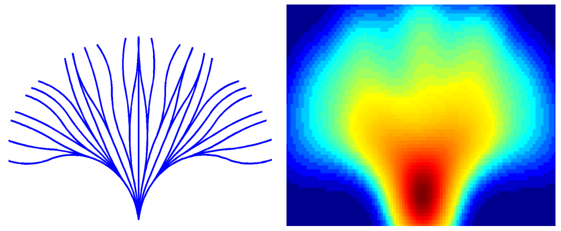
Knepper et al (Knepper and Mason 2009) further note the surprising fact that for a vehicle following a reasonable policy, averaged over time-steps the distribution of obstacles encountered ends up heavily weighted to the sides. Good earlier policy choices imply that the space to the immediate front of the robot is mostly devoid of obstacles. It is effectively a chicken-egg problem to find such a policy with respect to its own induced distribution of examples, which we address here as *weak dynamic optimality*.

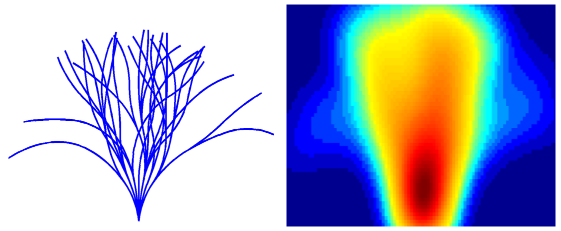
We briefly note the following propositions about the statistical performance of Algorithm 1. We elide full proofs to supplement material (Dey et al. 2011), but note that they follow from recent results of online-to-batch learning (Ross, Gordon, and Bagnell 2010) combined with the regret guarantees of (Streeter and Golovin 2008) on the objective functions we present.

**Proposition 1. (Approximate Static Optimality)** If *getNextEnvironment* returns independent examples from a distribution over environments (*i.e.*, the chosen control does not affect the next sample), then for a list $S$ chosen randomly from those generated throughout the $T$ iterations of Algorithm 1, it holds that $E_{d(\mathbf{ENV})}[(1-1/e)f(S*) - f(S)] \leq O(\frac{ln(1/\delta)}{\sqrt{(T)}})$ with probability greater then $1-\delta$.

**Proposition 2. (Approximate Weak Dynamic Optimality)** If *getNextEnvironment* returns examples by forward simulating beginning with a random environment and randomly choosing a new environment on reaching a goal, then consider the policy $\pi_{\text{mixture}}$ that begins each new trial by choosing a list randomly from those generated throughout the $T$ iterations of the Algorithm 1. By the no-regret property, such a mixture policy will be $\alpha$ approximately dynamically optimal in expectation up to an additive term $O(\frac{ln(1/\delta)}{\sqrt{T}})$ with probability greater then $1-\delta$. Further, in the (empirically typical) case where the distribution over library sequences converges, the resulting single list is (up to approximation factor $\alpha$) weakly dynamically optimal.


(a) Constant Curvature (b) Constant Curvature Density


(c) Green-Kelly (d) Green-Kelly Density


(e) SEQOPT (EXP3) Dynamic (f) SEQOPT (EXP3) Dynamic Density

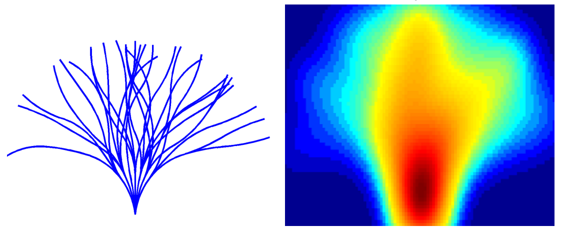

(g) SEQOPT (EXP3) Static (h) SEQOPT (EXP3) Static Density

Figure 2: The density of distribution of trajectories learned by our approach (SEQOPT using EXP3) for the dynamic planning paradigm in Figure 2e shows that most of the trajectories are distributed in the front whereas for the static paradigm they are more spread out to the side. This shows that for the dynamic case more trajectories should be put in the front of the robot as obstacles are more likely to occur to the side as pointed out by (Knepper and Mason 2009)

## Experimental setup

We simulated a robot driving over a real-world cost map generated for Fort Hood, Texas (Figure 3) with trajectory sequences generated by using the method devised by (Green and Kelly 2006) for both constant curvature (Figures 2a, 2b) and concatenation of trajectories of different curvatures (Figures 2c, 2d). The cost map and parameters for the lo-

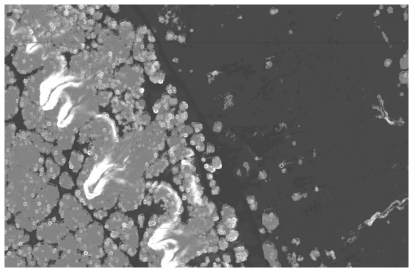


Figure 3: We used a real-world cost map of Fort Hood, TX and simulated a robot driving over the map in between random goal and start locations using trajectory sequences generated by different methods for comparision.

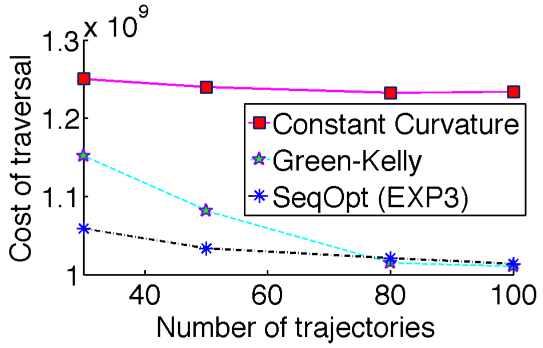


Figure 4: As the number of trajectories evaluated per planning cycle are increased the cost of traversal for trajectory sequences generated by Green-Kelly and our method drops and at 80-100 trajectories achieve almost the same cost of traversal. It is to be noted that our approach decreases the cost of traversal much faster than Green-Kelly trajectory sequences.

cal planner (number of trajectories to evaluate per time step, length of the trajectories, fraction of trajectory traversed per time step) were taken to most closely match that of the *Crusher* off-road ground vehicle described in (Bagnell et al. 2010). Please note that for purely visualization purposes, we plot the density of paths by a kernel density estimation: we put down a radial-basis-function kernel at points along the trajectories with some suitable discretization. The sum of these kernels provides a density estimate at each grid location of the plot. This is visualized using a color map over the values of the density estimate.

## Results

**Dynamic Simulation** Figure 1 shows the cost of traversal of the robot with different trajectory sets against number of runs. In each run the robot starts from a random location and ends at the specified goal. 100 goal locations and 50 start locations for every goal location were randomly selected. The set of weights for the $N$ copies of experts algorithm EXP3 were carried over through consecutive runs.

The cost of traversal of constant curvature trajectory sequences grows at the highest rate followed by Green-Kelly path set. The lowest cost is achieved by running Algorithm.1 with EXP3 as the experts algorithm subroutine. In 4396 runs there is a 8% reduction in cost of traversal between Green-Kelly and SEQOPT. It is to be emphasized that improvement in path planning is obtained with negligible overhead. Though the complexity of our approach scales linearly in the number of motion primitives and depth of the library, each operation is simply a multiplicative update and a sampling step. In practice it was not possible to evaluate even a single extra motion primitive in the time overhead that our approach requires. In 1 ms, 100000 update steps can be performed using EXP3 as the experts algorithm subroutine.

**Static Simulation** We also performed a static simulation where for each of the 100 goal locations the robot was placed at 500 random poses in the cost map and the cost of traversal of the selected trajectory $a^*$ over the next planning cycle was recorded. SEQOPT with EXP3 and Green-Kelly sequences obtained 0.5% and 0.25% lower cost of traversal than constant curvature sequences respectively. The performance for all three methods was essentially at par. This can be explained by the fact that Green-Kelly trajectory sequences are essentially designed to handle the static case of planning where trajectories must provide adequate density of coverage in all directions as the distribution of obstacles is entirely unpredictable in this case.

In the dynamic planning case on the other hand, the situations the robot encounters are highly correlated and because the robot is likely to be guided by a global trajectory, a local planner that tracks that trajectory well will likely benefit from a higher density of trajectories toward the front as most of the obstacles will be to the sides of the path. This is evident by the densities of generated trajectory sequences for each case as shown in Figure 2. Our approach naturally deals with this divide between static and dynamic planning paradigms by adapting the chosen trajectory sequence at all times. A video demonstration of the algorithm can be found at the following link:(Video 2012)

## Application: Grasp selection for manipulation

Most of the past work on grasp set generation and selection have focused on automatically producing a successful and stable grasp for a novel object, and the computational time is of secondary concern. As a result very few grasp selection algorithms have attempted to optimize the order of consideration in grasp databases. (Goldfeder et al. 2009) store a library of precomputed grasps for a wide variety of objects and find the closest element in the library for each novel object. (Berenson et al. 2007) dynamically rank pre-computed grasps by calculating a grasp-score based on force closure, robot position, and environmental clearance. (Ratliff, Bagnell, and Srinivasa 2007) employ imitation learning on demonstrated grasps to select one in a discretized grasp space. In all of these cases the entire library of grasps is evaluated for each new environment or object at run time, and the order of the entries and their effect on computation are not considered. In this section we describe our grasp ranking procedure, which uses result of trajectory planning to reorder a list of grasps, so that for a majority of environments, only a small subset of grasp entries near the front of the control library need to be evaluated.

For a sequence of grasps $S \in \mathscr{S}$ we define the submodular monotone grasp selection function $f : \mathscr{S} \rightarrow [0,1]$ as $f \equiv P(S)$ where $P(S)$ is the probability of successfully grasping an object in a given scenario using the sequence of grasps provided.

For any sequence of grasps $S \in \mathscr{S}$ we want to minimize the cost of evaluating the sequence i.e. minimize the depth in the list that has to be searched until a successful grasp is found. Thus the cost of a sequence of grasps can be defined as $c = \sum_{i=0}^{N} 1 - f(S_{\langle i \rangle})$ where $f(S_{\langle i \rangle})$ is the value of the submodular function $f$ on executing sequence $S \in \mathscr{S}$ up to $\langle i \rangle$ slots in the sequence. Minimizing $c$ corresponds to minimizing the depth $i$ in the sequence of grasps that must be evaluated for a successful grasp to be found. (We assume that every grasp takes equal time to evaluate)

The same algorithm for trajectory sequence generation (Algorithm 1) is used here for grasp sequence generation. Here each expert algorithm $\mathscr{E}_i$ maintains a set of weights for each grasp (expert) in the library. A sequence of grasps is constructed by sampling without repetition the distribution of weights for each expert algorithm copy $\mathscr{E}_i$ for each position $i$ in the sequence (lines 1-5). This sequence is evaluated on the current environment until a successful grasp $a^*$ is found (line 6). If the sucessful grasp was found at position $i$ in the sequence then in expert algorithm $\mathscr{E}_i$ the weight corresponding to the successful grasp id is updated using SEQOPT with EXP3's update rule. For WMR all the grasps in the sequence are evaluated and the weights for every expert are updated according to lines 9-12.

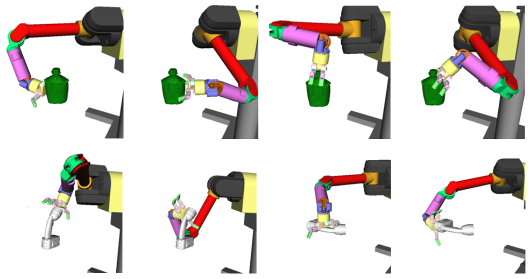


Figure 5: Example grasps from the grasp library sequence. Each grasp has a different approach direction and finger joint configuration recorded with respect to the object's frame of reference. Our algorithm attempts to re-order the grasp sequence to quickly cover the space of possible scenarios with a few grasps at the front of the sequence.

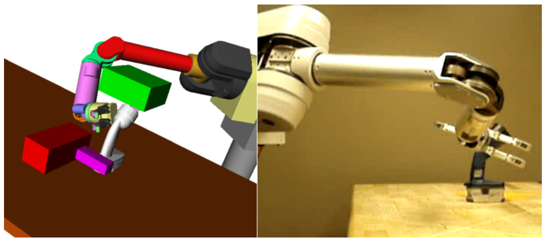


Figure 6: Executing a grasp in both simulation and real hardware. The grasp library ordering is trained in simulation, and the resulting grasp sequence can be executed in hardware without modifications.

## Experimental setup

We use a trigger-style flashlight as the target object and the OpenRAVE (Diankov 2010) simulation framework to generate a multitude of different grasps and environments for each object. The manipulator model is a Barret WAM arm and hand with a fixed base. A 3D joystick is used to control the simulated robot. Since the grasps in our library are generated by a human operator, we assume they are stable grasps and the main failure mode is in trajectory planning and obstacle collision. Bidirectional RRT (Kuffner and LaValle 2000) is used to generate the trajectory from the manipulator's current position to the target grasp position.

The grasp library consisted of 60 grasps and the library was evaluated on 50 different environments for training, and 50 held out for testing. For an environment/grasp pair the grasp success is evaluated by the success of Bi-RRT trajectory generation, and the grasp sequence ordering is updated at each time-step of training. For testing and during run-time, the learned sequence was evaluated without further feedback.

We used both EXP3 and WMR with SEQOPT, and compared performance to two other methods of grasp library ordering: a random grasp ordering, and an ordering of the grasps by decreasing success rate across all examples in training (which we call "frequency"). Since this is the first approach to predicting sequences there are no other reasonable orderings to compare against. At each time step of the training process, a random environment was selected from the training set and each of the four grasp sequence orderings were evaluated. The cost of evaluating a grasp is either 0 for a successful grasp while an unsuccessful grasp incurs cost of 0. The search depth for each test case was recorded to compute overall performance. The performance of the two naive ordering methods does not improve over time because the frequency method is a single static sequence and the random method has a uniform distribution over all possible rankings.

## Results

The performance of each sequence after training is shown in Figure 7. We can see a dramatic improvement in the performance of SEQOPT over the random and frequency methods. While random and frequency methods produce a grasp sequence that requires an average of about 7 evaluations before a successful grasp is found, SEQOPT with WMR and EXP3 produce an optimized ordering that require only about 5 evaluations which is a $\sim 29\%$ improvement. Since evaluating a grasp entails planning to the goal and executing the actual grasp, which can take several seconds, this improvement is significant. Additionally this improvement comes at negligible cost and in practice it was not possible to evaluate a single extra grasp in the extra time overhead required for our approach.

Note that a random ordering has similar performance to the frequency method. Because similar grasps tend to be correlated in their success and failure, the grasps in the front of the frequency ordering tend to be similar. When the first grasp fails, the next few are likely to fail, increasing average search depth. The SEQOPT algorithm solves this correlation problem by ordering the grasp sequence so that the grasps in the front quickly cover the space of possible configurations . A video demonstration of the algorithm can be found at the following link:(Video 2012)

## Conclusion

We have shown an efficient method for optimizing performance of control libraries and have attempted to answer the question of how to construct and order such libraries.

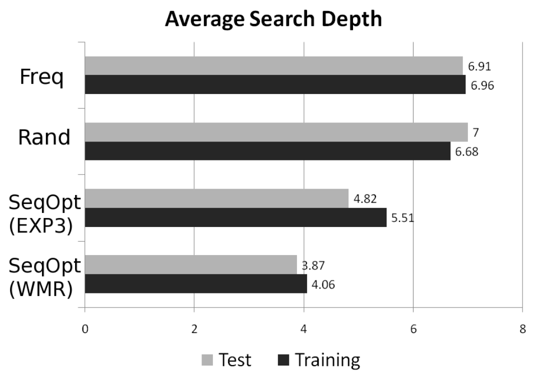


Figure 7: Average depth till successful grasp for flashlight object with 50 test environments. The training data shows the average search depth achieved at the end of training over 50 training environments. Algorithm 1 (SEQOPT) when run with EXP3 as the experts algorithm subroutine achieves 20% reduction over grasp sequences arranged by average rate of success (Freq.) or a random ordering of the grasp list (Rand.)

We aim to modify the current approach to close the loop with perception and take account of features in the environment for grasp sequence generation.

As robots employ increasingly large control libraries to deal with the diversity and complexity of real environments, approaches such as the ones presented here will become crucial to maintaining robust real-time operation.

## Acknowledgements

This work was funded by Army Research Laboratories through R-CTA and Defense Advanced Research Projects Agency through ARM-D.

## References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. 2003. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32(1):48–77.

Bagnell, J.; Bradley, D.; Silver, D.; Sofman, B.; and Stentz, A. 2010. Learning for autonomous navigation. *Robotics Automation Magazine, IEEE* 17(2):74 –84.

Berenson, D.; Diankov, R.; Nishiwaki, K.; Kagami, S.; and Kuffner, J. 2007. Grasp planning in complex scenes. In *IEEE-RAS Humanoids*.

Blum, A. 1996. On-line algorithms in machine learning. In *In Proceedings of the Workshop on On-Line Algorithms, Dagstuhl*, 306–325. Springer.

Cesa-Bianchi, N.; Conconi, A.; and Gentile, C. 2004. On the generalization ability of on-line learning algorithms. *Information Theory, IEEE Transactions on* 50(9):2050 – 2057.

Chinellato, E.; Fisher, R.; Morales, A.; and del Pobil, A. 2003. Ranking planar grasp configurations for a three-finger hand. In *ICRA*, volume 1, 1133–1138. IEEE.

Ciocarlie, M., and Allen, P. 2008. On-line interactive dexterous grasping. In *EuroHaptics*, 104.

Dey, D.; Liu, T. Y.; Sofman, B.; and Bagnell, J. A. D. 2011. Efficient optimization of control libraries. Technical Report CMU-RI-TR-11-20, Robotics Institute, Pittsburgh, PA.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. Dissertation, Carnegie Mellon University, Robotics Institute.

Feige, U. 1998. A threshold of ln n for approximating set cover. *JACM* 45(4):634–652.

Frazzoli, E.; Dahleh, M.; and Feron, E. 2000. Robust hybrid control for autonomous vehicle motion planning. In *Decision and Control, 2000.*, volume 1.

Fujishige, S. 2005. *Submodular functions and optimization*. Elsevier Science Ltd.

Goldfeder, C.; Ciocarlie, M.; Peretzman, J.; Dang, H.; and Allen, P. 2009. Data-driven grasping with partial sensor data. In *IROS*, 1278–1283. IEEE.

Green, C., and Kelly, A. 2006. Optimal sampling in the space of paths: Preliminary results. Technical Report CMU-RI-TR-06-51, Robotics Institute, Pittsburgh, PA.

Jackel, L., et al. 2006. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *JFR*.

Kelly, A., et al. 2006. Toward reliable off road autonomous vehicles operating in challenging environments. *IJRR* 25(1):449–483.

Knepper, R., and Mason, M. 2009. Path diversity is only part of the problem. In *ICRA*.

Kuffner, J.J., J., and LaValle, S. 2000. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, volume 2, 995 –1001.

Littlestone, N., and Warmuth, M. 1994. The Weighted Majority Algorithm. *INFORMATION AND COMPUTATION* 108:212–261.

Montemerlo, M., et al. 2008. Junior: The stanford entry in the urban challenge. *JFR* 25(9):569–597.

Ratliff, N.; Bagnell, J.; and Srinivasa, S. 2007. Imitation learning for locomotion and manipulation. Technical Report CMU-RI-TR-07-45, Robotics Institute, Pittsburgh, PA.

Ross, S.; Gordon, G.; and Bagnell, J. 2010. No-Regret Reductions for Imitation Learning and Structured Prediction. *Arxiv preprint arXiv:1011.0686*.

Saut, J., and Sidobre, D. 1918. Efficient Models for Grasp Planning With A Multi-fingered Hand. In *Workshop on Grasp Planning and Task Learning by Imitation*, volume 2010.

Sofman, B.; Bagnell, J.; and Stentz, A. 2010. Anytime online novelty detection for vehicle safeguarding. In *ICRA*.

Stolle, M., and Atkeson, C. 2006. Policies based on trajectory libraries. In *ICRA*, 3344–3349. IEEE.

Streeter, M., and Golovin, D. 2008. An online algorithm for maximizing submodular functions. In *NIPS*, 1577–1584.

Thrun, S., et al. 2006. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.* 23:661–692.

Urmson, C., et al. 2006. A robust approach to high-speed navigation for unrehearsed desert terrain. *JFR* 23(1):467–508.

Urmson, C., et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *JFR*.

Video, A. 2012. http://youtube.com/robotcontrol1.