A Data-Driven Approach to High Level Planning

Matt Zucker

January 2009

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Abstract

Motion planning for complex systems such as legged robots and mobile manipulators has proven to be a difficult task for a variety of reasons: not only must planning software consider a high-dimensional configuration space, but it must reason dynamically about how to apply forces to the real world. Accomplishing such planning in real-time is harder still. One promising strategy is to take a high-level approach to planning by reasoning about sequences of discrete behavior primitives. For many plaforms this has proven far more tractable than global search from start to goal in the full configuration space of the robot. A successful example from the field of legged robots is footstep planning, which reasons over sequences of footsteps.

High-level planning, however, introduces significant tradeoffs. Although the space of discrete behavior primitives is frequently more efficient to search, the planner depends on an underlying controller or policy to faithfully and effectively execute the primitives the planner has selected. Hence, the planner must have some model of the capabilities of the controller. Encoding such capability models can be a tedious and error-prone task: if the capability model is too conservative, the high-level planner may fail to find solutions for difficult problems; if the model is too liberal, the robot may fail to execute the selected behavior. The situation is compounded if the planner must reason about heterogeneous behaviors—imagine adding hopping and sliding behavior primitives to an existing footstep planner. It may not be clear how to translate estimates of the risk and cost of diverse actions into a common currency.

This thesis aims to develop a general system for quickly and effectively selecting among heterogeneous behaviors for high-dimensional robotic navigation and manipulation. The central idea of the system is to store pre-planned and/or previously executed actions in a behavior library, which is then analyzed and queried via machine learning techniques. This data-driven approach can aid high-level planning in a number of ways. First and foremost, planning is accelerated by re-using the results of previous computation. In addition, the high-level planner can adapt its capability model as well as heuristic cost-to-go estimates over time in order to better reflect the capabilities of the system.

The proposed high-level planning system will be demonstrated and evaluated on the Boston Dynamics Inc. LittleDog quadruped robot, as well as on another high-dimensional robotic platform.

Contents

Figure 1: Very capable robots for locomotion and manipulation. Left to right: Stanford University PR1, Boston Dynamics Inc. BigDog, and AIST HRP-2.

1 Introduction

Robotic systems are becoming far more capable than they once were. Complex tasks such as locomotion over rough terrain and dextrous manipulation in human environments are finally coming within reach of today's robots. However, as robotic capability increases, motion planning can become a daunting task.

Motion planning can be broadly defined as the task of finding a collision-free path or trajectory through the configuration space, or set of all possible poses, of a robotic system in order to reach a particular goal. Typically, some effort is made to ensure that the path returned by the planner is of acceptably low cost, expressed in terms of quantities such as smoothness, energy consumption and distance to obstacles.

For relatively simple robotic systems, resolution complete algorithms such as Dijkstra's algorithm, A*, and its dynamic real-time variant D*, can be used to find collision-free paths and trajectories for robots [19, 49]. As the number of degrees of freedom of a robot increases (generally above three or four DOF), these algorithms may become unusable due to the so-called *curse of dimensionality*: an exponential increase in the time required for search. Today's predominant workhorses in motion planning for high-DOF robots are sampling-based planners such as PRM and RRT [31, 34]. which use randomness to more rapidly discover the connectivity of high-dimensional configuration spaces. Although sampling-based planners can solve some complex search problems far more quickly than their resolution-complete predecessors, for many problem domains their success is dependent upon the use of ad-hoc heuristic sampling strategies. Furthermore, such planners produce suboptimal output which must be optimized to remove jerky or unnecessary motion; such optimization is itself an active field of research.

Some high-DOF robots such as TU Delft's Flame robot [23], or the Boston Dynamics Inc. BigDog quadruped [11], use carefully tuned controllers and policies to coordinate their motion without an explicit "motion planning" module. While these controllers and policies may be robust enough to overcome significant deviations in terrain, they fall short of exploiting the full capabilities of their respective platforms. In particular, if careful reasoning about footstep locations is required, these policies fail because they only treat terrain features as a disturbance to the robots' walking motion.

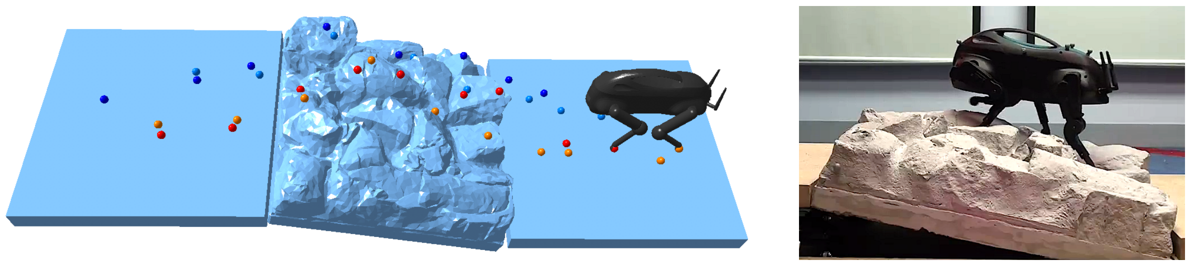


Figure 2: *Left:* Footstep plan over tilted rocky terrain. *Right:* Execution on Boston Dynamics Inc. LittleDog robot.

1.1 High Level Planning

Rather than solving the seemingly intractable task of global planning from start to goal in the full configuration space of the robot, my aim is to create a general *high level planner* which reasons in the space of discrete behavior primitives, such as "step to location S" or "jump from point A to point B". A successful example of high level planning from the domain of legged robots is footstep planning, which attempts to find low-cost sequences of footsteps [13].

I have already begun exploring footstep planning through Carnegie Mellon University's participation in the DARPA Learning Locomotion project. The project's goal is to design planning and control algorithms to guide the LittleDog quadruped robot, designed and manufactured by Boston Dynamics Inc., over rough terrain.

In my current implementation of the LittleDog control software, a footstep planner is used to find a good sequence of footsteps from from start to goal. Once a sequence of footsteps is chosen, a trajectory generation module outputs a naïve initial guess of the full-body trajectory to execute each footstep. The estimated footstep trajectory is refined in a trajectory optimization module before execution. The optimizer balances criteria such as smoothness, obstacle avoidance, and dynamic stability in order to produce good output trajectories.

Although the current combination of footstep planning and optimization is quite effective, it is too slow to use in a real-time setting, and it only handles a single type of behavior primitive. Fast performance is important in order to effectively re-plan in case of execution errors. As the difficulty of the LittleDog terrains increases, adding additional behavior modalities such as hopping and sliding will become increasingly important.

High level planning holds much promise because reasoning at the level of discrete behaviors decomposes the overall motion planning problem into more easily digestible chunks. A particularly thorny issue surrounding high level planning, though, is the degree of coupling between the planner, which selects behavior primitives, and the underlying controllers or policies which execute the behaviors. Managing this coupling can be a difficult task for a programmer because the planner must maintain a good model of the robot's capabilities. If the capability model is too conservative, the planner may fail to find solutions to difficult problems; too liberal, and the robot may fail to execute the selected behaviors. Another aspect of this coupling is that the planner should have a good estimate of the heuristic cost-to-go for a problem instance in order to speed up search, which may also be difficult for a programmer to encode. Introducing heterogeneous behaviors complicates matters further for the programmer, because the capability model and heuristic estimates must reflect capabilities of diverse modalities of actions.

1.2 Thesis Statement

This thesis aims to develop a general system for quickly and effectively selecting among heterogeneous behaviors for high-dimensional robotic navigation and manipulation. The central idea of the system is to store pre-planned and/or previously executed actions in a *behavior library*, which is then analyzed and queried via machine learning techniques.

Work on the thesis will demonstrate three main benefits of this data-driven approach to high level planing. First, the behavior library should accelerate planning by re-using previous computation. Second, the system should adapt its capability model over time so that the planner gradually improves its ability to estimate what behaviors can be executed, as well as the behaviors' relative costs. Finally, it will also adaptively improve its heuristic cost-to-go estimates in order to reflect the costs and capabilities that it learns.

In order to evaluate whether the system in fact confers these benefits, learned functionality will be compared to a baseline implementation that represents a best effort towards hand-designed functionality. The system can be evaluated both in terms of success rate (how often did the robot slip or fall) as well as in terms of computation effort. Some effects may come in terms of tradeoffs; for example, taking a small decrease in overall optimality in order to achieve increased generality.

The generality of the proposed work will be demonstrated by using the system to coordinate heterogeneous behaviors on the LittleDog quadruped robot, as well as on another high-DOF robotic system.

2 Related Work

2.1 High Level Planning

As previously stated, footstep planning is a notable and relevant example of high level planning for legged robots [13]. Earlier work on quadrupeds phrased the problem more in terms of trajectory optimization [52].

In the computer graphics community, high level planning is used to select sequences of behaviors for animated characters. In the so-called "motion graphs" approach, a library of recorded behaviors can be connected into sequences based on an automated stitching technique [33]. Motion graphs have been used more recently in a real-time approach with precomputed search trees [36], and work has been done to ensure optimal search on interpolated graphs [47].

2.2 Behavior and Trajectory Libraries

The concept of using a trajectory library to store previously executed behaviors was used in [4] to solve a marble maze. The trajectory library stores both trajectories observed by watching a skilled human, as well as additional trajectories created during learning by the robotic system. A similar approach is taken again on the marble maze, and also for controlling a LittleDog robot in [50].

2.3 Learned Heuristics

Recent work at Carnegie Mellon University has investigated Maximum Margin Planning (MMP), which learns heuristics for optimal planners in by phrasing the problem as maximum-margin structured prediction [44, 46]. The goal of MMP is to learn a weighting of features to create a costmap that yields optimal paths similar to those specified by a human expert. First used to compute cost maps for an autonomous ground vehicle, the research was later extended to grasping and also to footstep selection for the LittleDog robot [42].

Also related is the concept of Inverse Reinforcement Learning (IRL), developed at Stanford [39, 1]. The motivation is similar to MMP: learn a policy that matches the behavior of a human expert. IRL has been used for autonomous helicopter maneuvers, as well as on simulated driving platforms.

2.4 Trajectory Optimization

The field of trajectory optimization is closely tied to sampling-based planning because such planners produce output that often contains jerky or redundant motion which must be eliminated before execution on a robot. Perhaps the most prevalent method of path optimization is the so-called "shortcut" heuristic, which picks pairs of configurations along the path and invokes a local planner to attempt to replace the intervening sub-path with a shorter one [30, 12]. "Partial shortcuts" as well as medial axis retraction have also proven effective [16].

Another approach used in elastic bands or elastic strips planning involves modeling paths as mass-spring systems: a path is assigned an internal energy related to its length or smoothness, along with an external energy generated by obstacles or task-based potentials. Gradient based methods are used to find a minimum-energy path [40, 9].

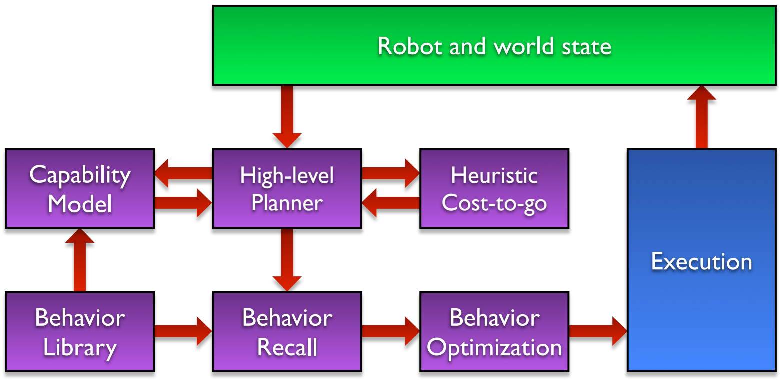
2.5 Locomotion Over Rough Terran

A wide variety of work examines the task of locomotion over rough terrain. Work at Stanford has investigated rock climbing robots [8]. Other research at that institution has investigated using sampling-based planners to produce walking motion for humanoid robots without a regular gait [20]. Similar work explores grasp-based motion planning for animated characters [29].

Perhaps the most prominent example of robust robot locomotion is Boston Dynamics' BigDog quadruped [11], which recently autonomously traversed 12.8 miles without human intervention [6].

3 Technical Approach

As stated, the core idea of the system proposed here is to store pre-planned and/or previously executed behaviors in a *behavior library*, which is then analyzed and queried via machine learning techniques.



What follows is a brief outline of the functionality of the various modules of the system: Let $x \in \mathcal{X}$ denote the state of the robot. The task of the high-level planner is to select the best behavior $u \in U(x)$ to advance the robot towards its goal. To do so, it consults a *capability model* which provides not only the set $U(x)$ of available behaviors given the robot state, but also state transition functions $x' = f(x, u)$ and one step cost functions $c(x, u)$ for the actions in $U(x)$. The high-level planner is also able to consult a *heuristic cost-to-go* module which estimates a lower bound $h(x)$ on the total cost to get to the goal.

Once a behavior is selected by the high-level planner, the task of the *behavior recall* module is to select a trajectory or control sequence from the behavior library which closely matches the selected behavior. Since the behavior recalled from the library is not necessarily optimal for the current robot and world state, it is passed through a *behavior optimization* module prior to execution (similar to the existing trajectory optimization module used in the LittleDog software described in section 4.2.1).

The following sections describe the modules in more detail. Particular techniques for implementing the modules are discussed in section 5.

3.1 High Level Planner

The task of the high level planner is to pick the best next behavior for the robot to execute given the current state and some specification of the robot's goals. Ideally, this could be implemented via a number of deterministic heuristic search algorithms. One obvious candidate is to use A* search.

In the case of A* search, the planner needs to be able to enumerate the behaviors available for a particular state as well as their relative costs. This information is provided by the capability model. The planner must also have access to a heuristic cost-to-go function, which will be provided by a dedicated module. Finally, in order to implement A*, the high level planner must be able to determine if a state has been visited before. A simple hashing scheme based on binning the various robotic degrees of freedom should suffice here.

The appeal of the system as described here is that it is amenable variety of other search algorithms, including N-step-lookahead search or anytime variants such as ARA* [38].

3.2 Behavior Library

The behavior library is intended to store pre-planned and/or previously executed actions of the robot. At its core it is simply a collection of records. At minimum, each record in the behavior library should store the initial state of the robot, the type of behavior primitive used, and the parameters used to instantiate the behavior.

For behaviors that require a non-trivial trajectory generation and optimization such as the current LittleDog footsteps (see section 4.2.1), the records should also store the full-body trajectory used to control the robot throughout the behavior. If optimization is used to generate trajectories, objective function values for the optimizer at the start and end of optimization may also be useful to store.

Actions that have actually been executed on the robot (as opposed to merely pre-planned) should also store the measured state of the robot from execution, along with sensor values (such as force sensor values) which may be helpful for diagnostic purposes.

Major questions that will have to be answered are how frequently to insert records into the library, and whether all actions, good or bad, should be candidates for insertion. Since the consumers of the behavior library will be attempting to evaluate the risk and cost of various actions, it may in fact be useful to have both positive and negative examples of behaviors in the library.

3.3 Capability Model

The task of the capability model is to provide the set $U(x)$ of behaviors available for a given robot state x, as well as the relative cost $c(x, u)$ of various behaviors. Here u includes not only the type of the behavior (if heterogeneous behaviors are supported) but also the parameters needed to instantiate the behavior.

The task of assigning cost to actions can be phrased in machine learning terms as regression or function approximation. The input to the learning system is training data consisting of the state x and the behavior parameters u. In a supervised setting, the output of the module would be trained to match particular labels of the (x, u) pairs. In an unsupervised setting, the robot could use reinforcement learning techniques to learn the cost based on success or failure of its own actions. Note that failure in this setting need not be anything as drastic as the robot falling over; it could also include data such as increased error in trajectory following or unexpected force sensor readings.

Although the behaviors themselves may be heterogeneous, there is no reason that a single regressor needs to compute costs for more than one modality of behavior. As long as their relative estimates are consistent overall, separate regressors may compute the cost-to-go for diverse behaviors.

For many types of actions in certain environments (footsteps on level ground, for example), there may be a large number of near-equally valid parameter values for the behavior. In such situations, it would be ideal if the capability model could cluster the behavior into a succinct number of representative alternatives. A similar problem of picking a “representative set” of trajectory snippets for a wheeled vehicle is studied in [7], and the greedy algorithm proposed in the work may be applicable towards this goal.

3.4 Heuristic Cost-to-go

Search-based planners such as A* require a heuristic estimate $h(x)$ of the cost-to-go for a state x in order to evaluate the relative merit of states in the search tree. The heuristic cost-to-go module is responsible for providing this information. Like the capability model, this can be formalized in terms of regression or function approximation.

This may well be the most complicated component of the system because it needs to estimate the cost-to-go considering all modalities of behaviors (unlike the capability model which can have a one-to-one mapping from regressors to behavior types). On the other hand, learning for the cost-to-go module can always be conducted in a supervised manner: for a given trajectory, its job is simply to try to match the sum of costs returned by the capability model.

The difficulty of the task is also mitigated by the fact that the estimate $h(x)$ need not be perfectly accurate—if it were so, the heuristic would make redundant the entire planner! Instead, the estimated cost-to-go should be a useful lower bound.

3.5 Behavior Recall and Optimization

Although many behaviors can be efficiently implemented as real-time policies or controllers, some behaviors (such as LittleDog footsteps in the current framework) require extensive trajectory generation and optimization before they can be executed on the robot. In this sense, trajectory optimization can be viewed as a “local planner” of the type commonly referred to in the PRM literature.

Given a state action pair (x, u) from the high-level planner, the aim of the behavior recall module is to search for the best matching stored behavior in the library to the requested pair. If the stored behavior sufficiently resembles the one selected by the high level planner it may be possible to greatly speed convergence of optimization by starting near a local optimum.

Behavior recall may not be necessary for behaviors that are implemented via simple controllers or policies.

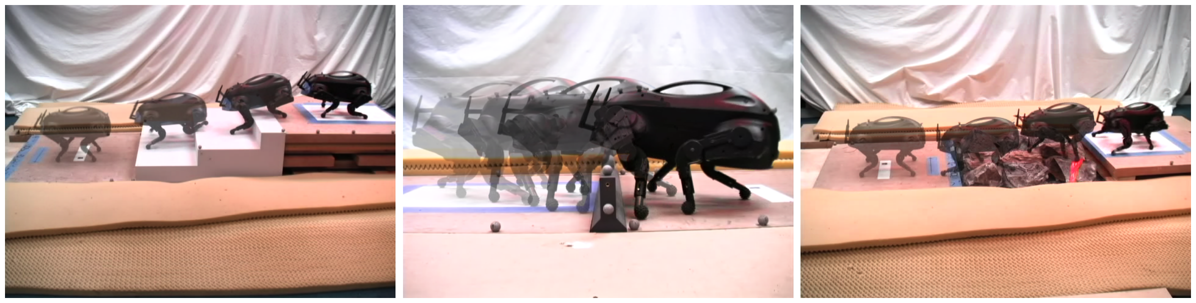


Figure 3: Multiple-exposure photos of LittleDog traversing various terrains from Phase II of the DARPA Learning Locomotion project. Left to right: steps, Jersey Barrier, and rocks.

4 Work to Date

4.1 LittleDog Quadruped Robot

CMU is one of five institutions participating in the DARPA Learning Locomotion project. The goal of the project is to produce robust and fast walking behavior for the Boston Dynamics LittleDog robot across rough terrain. Testing is performed by having the robot walk across a variety of predetermined "terrain boards" which simulate rocks, barriers, slopes, and other obstacles. In the current phase of the project, the robot is required to be capable of traversing obstacles over 10 cm (0.65 leg lengths) in height, at speeds of over 7.2 cm/s (roughly 0.5 leg lengths per second). The Learning Locomotion project has provided a rich backdrop for developing robot functionality. Since I have joined the project, I have developed a framework consisting of over 28,000 lines of C/C++ code for robot planning, control, visualization, and analysis. I have also logged hundreds of hours of runtime on the LittleDog robot.

The LittleDog robot has 4 legs with three degrees of freedom each (two at the hip and one at the knee) for a total of 12 actuators. All of LittleDog's joints are controlled via geared motors. The hip degrees of freedom are jointly controlled through a differential, and the knee is independently driven. LittleDog's feet are attached to the lower legs via a spring which is constrained to move along the axis of the leg. This introduces an extra, passive, degree of freedom for each leg which we generally have not modeled.

The LittleDog robot is equipped with encoders for each of the 12 joints, as well as three-axis force sensors on each spherical foot. Forces parallel to the leg are measured via a linear potentiometer in series with the spring; perpendicular (lateral/shear) forces are measured using strain gauges in the shin. The position and orientation of the dog, as well of all the terrain boards, are measured by a Vicon motion capture system, which gives sub-millimeter position sensing at very low latency.

4.1.1 Conventions and coordinate frames

In the rest of the LittleDog section, the following notations will be adopted:

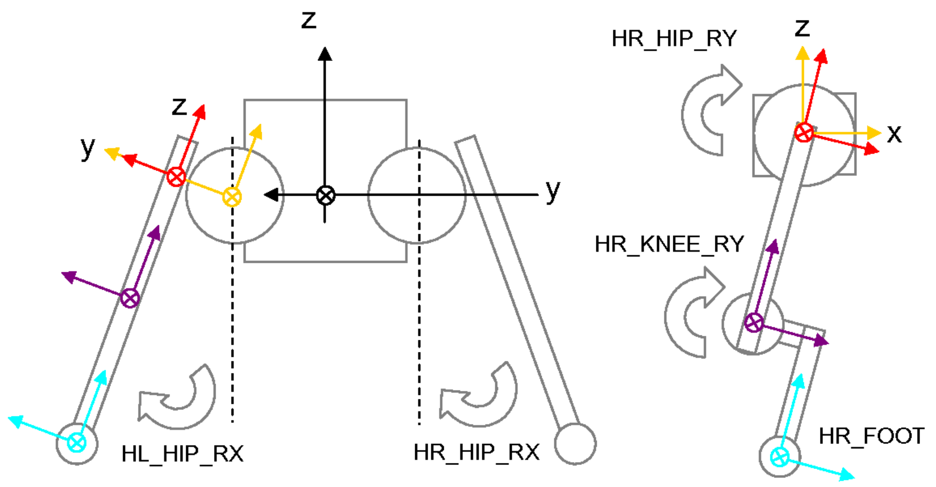


Figure 4: Kinematic coordinate systems for LittleDog, reproduced from Boston Dynamics documentation.

$\mathbf{x}$	Robot position - refers to center point of hip joints
$\mathbf{R}$	Robot rotation
ω	Robot angular rates or differential rotation
$\mathbf{f}$	Foot position
$\mathbf{p}$	Arbitrary reference point on robot
$\mathbf{q}$	Joint angles for leg

Subscripts W and B are used to denote either *world* or *body* frame for vectors, i.e. $\mathbf{p}_W$. If subscripts are omitted, the point is assumed to be in the world frame. Transforming from body to world frame is accomplished via the transformation

$$\mathbf{p}_W = \mathbf{R}\,\mathbf{p}_B + \mathbf{x}$$

And inversely,

$$\mathbf{p}_B = \mathbf{R}^T\,(\mathbf{p}_W - \mathbf{x})$$

Finally, I use $[\mathbf{u}]_\times$ to denote the skew-symmetric matrix such that $[\mathbf{u}]_\times\,\mathbf{v}$ is equal to the cross product of $\mathbf{u}$ and $\mathbf{v}$.

4.1.2 LittleDog Kinematics

Each LittleDog leg uses three separate motors to control the position of a spherical rubber foot. Since the motors cannot independently control the position and orientation of the foot, the legs are underactuated; hence, LittleDog cannot generate arbitrary forces and torques where the foot touches the ground.

The default forward and inverse kinematics routines supplied by Boston Dynamics consider the end effector for the leg to be located on the surface of the sphere at negative leg Z. I find a more natural description of the foot position refers to the center of the foot, which allows the software to quickly compute the height of the foot above flat ground independently of rotation.

The most frequent use of inverse kinematics is to solve for the set of joint angles which place the trunk at a given position and orientation, given a set of desired foot positions. Although there are as many as four distinct solutions to place the foot at a given body-relative position (knee forward/back, hip up/down), there is generally only one preferred solution in the parts of the workspace that are typically used.

The LittleDog robot has roughly 3 kg of mass, 2.25 kg of which is in the trunk. Most of the leg mass is concentrated in the upper legs. The kinematic origin of the robot lies at the centroid of all four hip joints. Although the actual center of mass of the body is about a centimeter above the kinematic origin, due to leg mass, the actual CoM of the robot is close enough to the kinematic origin to use as a CoM for stability computation.

4.1.3 Full-body IK for Arbitrary Reference Points

Assume there is a reference point $\mathbf{p}$ associated with a particular frame for some link on the robot, as shown in figure 5. The stance leg is controlled so that for any position and orientation $(\mathbf{x}, \mathbf{R})$ of the trunk, the foot remains at position $\mathbf{f}$. If the reference point $\mathbf{p}$ is in collision with the environment, we want to know how to change the trunk pose to bring the leg out of collision. This requires solving for the two Jacobian matrices

$$\mathbf{J}_{\mathbf{p}}(\mathbf{x}) = \left[\frac{d\mathbf{p}}{d\mathbf{x}}\right], \quad \text{and} \quad \mathbf{J}_{\mathbf{p}}(\mathbf{R}) = \left[\frac{d\mathbf{p}}{\omega}\right].$$

To accomplish this, we first need to solve a simpler problem. How does the reference point $\mathbf{p}_B$ change in body coordinates as we move the foot $\mathbf{f}_B$? The answer is the matrix

$$\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B) = \left[\frac{d\mathbf{p}_B}{d\mathbf{f}_B}\right]$$

The differential kinematics for the current joint angles $\mathbf{q}$ yield straightforward solutions for the matrices $\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})$ and $\mathbf{J}_{\mathbf{f}_B}(\mathbf{q})$. Then the chain rule tells us that

$$\begin{aligned}
\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B) &= \left[\frac{d\mathbf{p}_B}{d\mathbf{f}_B}\right] \\
&= \left[\frac{d\mathbf{p}_B}{d\mathbf{q}}\right]\left[\frac{d\mathbf{q}}{d\mathbf{f}_B}\right] \\
&= \mathbf{J}_{\mathbf{p}_B}(\mathbf{q})\,\mathbf{J}_{\mathbf{f}_B}(\mathbf{q})^{-1}
\end{aligned}$$

Since this step requires matrix inversion, we must be careful to make sure that the Jacobian for the foot is invertible. Also, note that the matrix $\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})$ may be rank-deficient: it will have zeros in one or more right-hand columns when the reference point is not attached to the frame of the shin. In the case of a reference point on the trunk, $\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})$ is simply the zero matrix.

Now we can tackle the original problem of finding out how $\mathbf{p}$ moves as we change the position

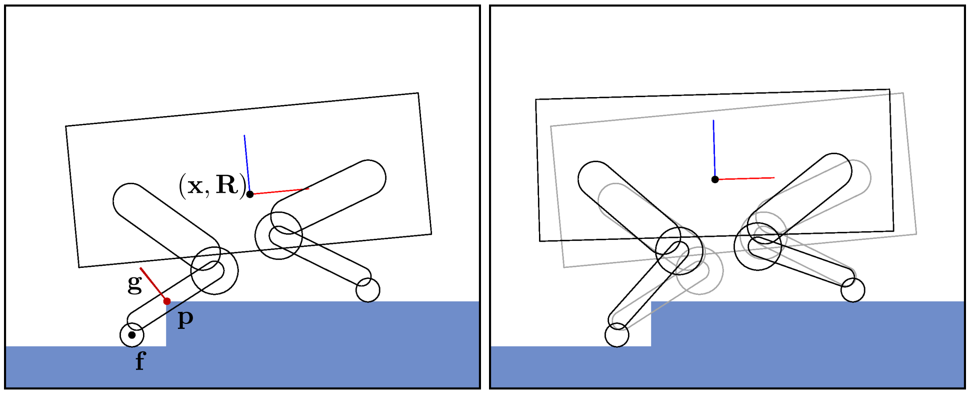


Figure 5: *Left:* a collision is detected at point $\mathbf{p}$ along the shin. How should we change the pose of the trunk $(\mathbf{x}, \mathbf{R})$ to move point $\mathbf{p}$ along the gradient $\mathbf{g}$ of the terrain? *Right:* full-body IK translates and rotates the trunk to correct the shin collision.

and orientation of the trunk. It turns out that

$$\begin{aligned} \mathbf{J_p}(\mathbf{x}) &= \left[\frac{d\mathbf{p}}{d\mathbf{x}}\right] \\ &= \mathbf{I} - \mathbf{R}\,\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B)\,\mathbf{R}^T \end{aligned}$$

where $\mathbf{I}$ is the 3-by-3 identity matrix, and

$$\begin{aligned} \mathbf{J_p}(\mathbf{R}) &= \left[\frac{d\mathbf{p}}{\omega}\right] \\ &= \mathbf{R}\left(\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B)\,[\mathbf{f}_B]_\times - [\mathbf{p}_B]_\times\right) \end{aligned}$$

By inspection, it is easy to verify that these solutions are correct when the reference point is at the foot, in which case $\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B) = \mathbf{I}$, and also when the reference point is attached to the trunk frame, in which case $\mathbf{J}_{\mathbf{p}_B}(\mathbf{f}_B) = [\mathbf{0}]$.

To find out a differential body translation $\Delta\mathbf{x}$ and rotation ω, we use Jacobian transpose control. If we want to move point $\mathbf{p}$ in the direction of $\mathbf{g}$, we set

$$\begin{aligned} \Delta\mathbf{x} &\propto \mathbf{J_p}(\mathbf{x})^T\,\mathbf{g} \\ \omega &\propto \mathbf{J_p}(\mathbf{R})^T\,\mathbf{g} \end{aligned}$$

4.1.4 Footstep Planning for LittleDog

The goal of footstep planning is to find low-cost sequences of footsteps to move LittleDog from its start position to a predefined goal region. The cost of each footstep is expressed as the sum of a *location cost* which specifies the relative difficulty of stepping on various terrain patches, and a *pose cost* which considers kinematic reachability, support polygon size, and proximity to obstacles.

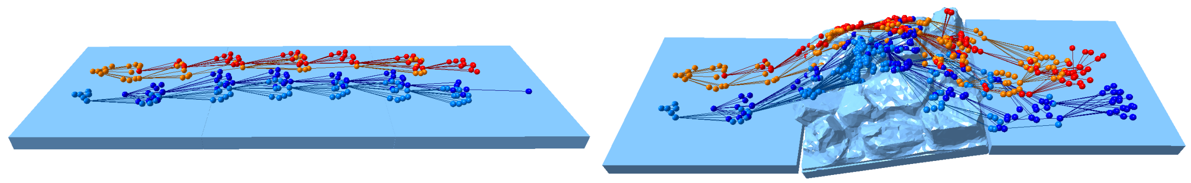


Figure 6: A* search trees from footstep planning. *Left:* the footstep planner finds a near-optimal path over flat ground with just over 200 expansions. *Right:* over a more difficult slanted rocks terrain, the planner must expand more than 700 nodes to find a reasonable solution.

In the current LittleDog software, the location cost depends only on the location of the current swing foot. Hence, it can be precomputed into a cost map for a terrain before execution. Pose cost depends on the joint location of all four feet before the step is taken as well as the new location for the swing foot. It must be computed online during footstep planning.

Footstep planning for LittleDog is accomplished via the A* algorithm or an anytime variant, the ARA* algorithm [38]. In order to use these algorithms, not only must the cost of a footstep be defined, but the planner must have access to a heuristic cost-to-go estimate. The heuristic estimate used in the LittleDog planning is to model the robot as a so-called "Dubins car" that can only move forward, with a minimum turning radius. Given starting and goal configurations, it is possible to solve for the minimum distance for such a car to travel [48]. To convert the distance estimate to a cost estimate, I compute an estimated number of footsteps per unit distance and assume flat ground (unit cost per step).

The robot may begin a footstep sequence by moving either of the hind feet. Thereafter, an ordered crawl gait is enforced: the front leg on the same side of the robot is moved, followed by the opposite hind leg, and so on. Computation of location cost and pose cost are summarized in the following sections.

4.1.5 Location Cost

To build a cost map for the terrain, a feature vector is extracted for each (x, y) location. The features are based on characteristics of a locally weighted quadratic regression of the heights $z(x, y)$ around each sample of terrain:

$$z(x,y) \;=\; [\,x\;y\,]\begin{bmatrix} k_{xx} & k_{xy} \\ k_{xy} & k_{yy}\end{bmatrix}\begin{bmatrix} x \\ y\end{bmatrix} + [\,k_x\;k_y\,]\begin{bmatrix} x \\ y\end{bmatrix} + k_z$$

For each local regressor, the mean x and y values are subtracted off, as is the weighted mean of z so that all data is zero-centered. The coefficients k are chosen to minimize the weighted residual

$$\begin{aligned} k &= \arg\min \|W\,(A\,k - z)\|^2 + \lambda\|k\|^2 \\ &= (A^T W A + \lambda I)^{-1}\,A^T\,W\,z \end{aligned}$$

where λ is a small regularization constant and I is the identity matrix. The coefficient vector k, data matrix A and observations z are given by:

$$\begin{aligned} k &= \begin{bmatrix} k_{xx} & k_{yy} & k_{xy} & k_x & k_y & k_z \end{bmatrix}^T \\ A &= \begin{bmatrix} x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ & \vdots & & & \vdots & \\ x_n^2 & y_n^2 & x_n y_n & x_n & y_n & 1 \end{bmatrix} \\ z &= \begin{bmatrix} z_1 & \dots & z_n \end{bmatrix}^T \end{aligned}$$

The weights are simply Gaussian in x and y:

$$\begin{aligned} w_i &= \exp\left(-\frac{(x_i^2 + y_i^2)}{2\sigma^2}\right) \\ w &= \begin{bmatrix} w_1 & \dots & w_n \end{bmatrix}^T \\ W &= \operatorname{diag}(w) \end{aligned}$$

From the coefficients k, five features are extracted for use as input to the cost map computation:

$$\begin{aligned} f &= \begin{bmatrix} e_1 & e_2 & s & \varepsilon & d \end{bmatrix}^T \\ (e_1, e_2) &= \operatorname{eig}\left(\begin{bmatrix} k_{xx} & k_{xy} \\ k_{xy} & k_{yy} \end{bmatrix}\right), \quad e_1 \leq e_2 \\ s &= \sqrt{k_x^2 + k_y^2} \\ \varepsilon &= \sqrt{\frac{\sum_i w_i \left(z(x_i, y_i) - z_i\right)^2}{\sum_i w_i}} \end{aligned}$$

Here e_1 and e_2 are the eigenvalues of the matrix of second-order coefficients (always real because the second-order matrix is a symmetric real matrix). Two positive eigenvalues correspond to a bowl-like indentation of terrain; negative eigenvalues correspond to a dome-like feature. Both positive and negative eigenvalues indicate a saddle point. The magnitude of the terrain slope is given by s, with higher values indicating steeper slope. The weighted error ε of the regression is also used as a feature. Finally, another feature d related to the distance of each terrain sample to the edge of terrain boards is extracted.

This set of features has a number of advantages. First, it is rotationally invariant, so that the same terrain board will yield the same features independent of its yaw. Second it allows for easy identification of hazards for LittleDog – the robot should avoid edges, steep slopes, and peaked areas of the terrain. Finally, the zero vector corresponds to flat ground far from terrain board edges, which is a known good support for LittleDog.

To convert from feature vectors to location cost, a utility function $U(f)$ is created which has a multiplicative relationship to cost:

$$C(f) = \exp\left(U(\vec{0}) - U(f)\right)$$

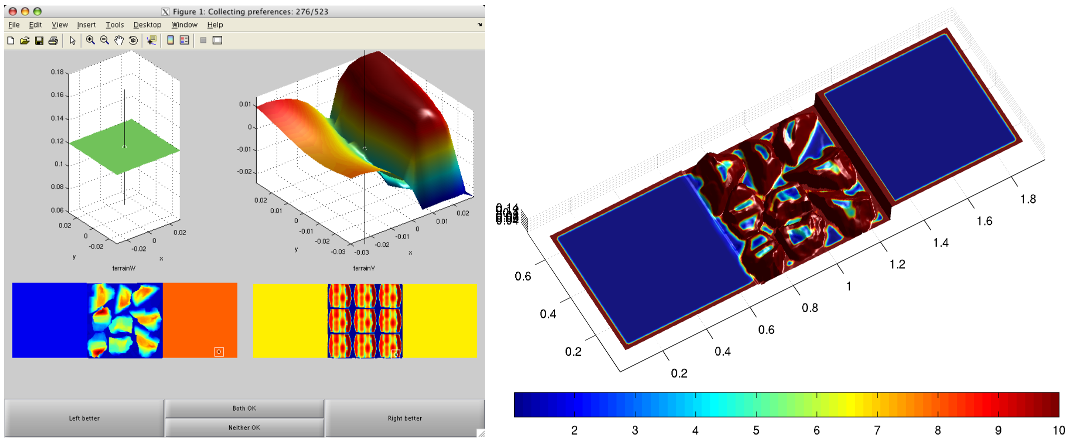


Figure 1: Collecting preferences: 276/523
File Edit View Insert Tools Desktop Window Help
terrainW
terrainY
Left better
Both OK
Neither OK
Right better

Figure 7: *Left:* interface for learning cost maps via support vector ranking. An expert chooses expresses preferences over pairs of randomly sampled terrain patches. *Right:* learned costmap for a rocky terrain. Low costs in blue, high costs in red.

Here, the cost $C(f)$ associated with a feature vector f is computed by exponentiating the negative utility of the features $U(f)$. As mentioned, flat ground far from terrain board edges corresponds to the zero feature vector. Hence, the cost function is normalized so that flat ground has unit cost (adding $U(\vec{0})$ in the equation above). As utility decreases linearly, cost increases multiplicatively: if $C(f_1)$ is greater than $C(f_2)$ by a factor of m, then $U(f_1)$ must be less than $U(f_2)$ by a difference of $\log m$.

The utility function $U(f)$ is learned from preferences input by a human expert (see figure 7). The learning algorithm is based on the notion of support vector ranking [22], which enforces the expert's preferences as constraints on the utility function in a maximum-margin fashion. The utility function is expressed as a sum of radial basis functions centered on the training data elements f_i:

$$U(f) \quad = \quad \sum_i \alpha_i \exp\left(-\frac{\|f - f_i\|^2}{2\sigma^2}\right)$$

When the human expert expresses a preference over two features f^+ and f^- (better and worse, respectively), the constraint

$$U(f^+) \quad \geq \quad U(f^-) + m$$

is enforced. Instead of solving the quadratic programming problem that corresponds to finding the weights a_i of the training data points, a subgradient-based method similar to the NORMA family of algorithms is used [32].

4.1.6 Pose Cost

The first step of assigning a pose cost to a footstep is to examine the size of the support polygon for the *next* footstep. Say the front left leg is being moved. Then the next leg to move is the hind

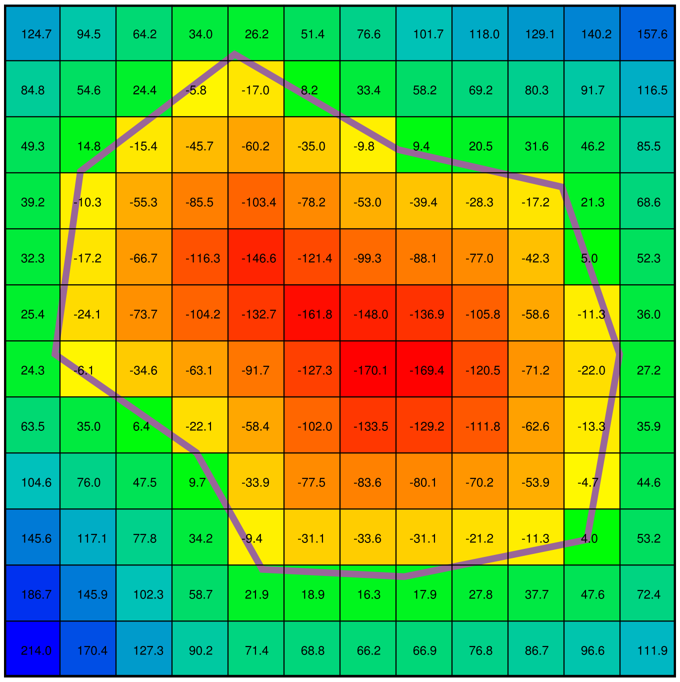


Figure 8: Signed distance field representation of a closed 2D polygon. Cells with negative values lie inside the polygon; positive values lie outside.

right leg (as dictated by the order of the crawl gait). If the incircle radius of the support triangle formed by the hind left, front left, and front right foot locations is lower than a predefined threshold, then the footstep is discarded. Very small triangles that are just bigger than the threshold are penalized slightly as well. Self-collisions of the robot are also discouraged by penalizing poses where the swing foot moves very close to a stance foot.

The next step to assigning the pose cost is to associate the footstep with a pose of the robot's trunk. The trunk position and rotation are initialized to heuristic values based on the positions of the three stance feet and the position of the new swing foot, and IK is used to find solutions for the leg joint angles to place the feet. At this point a *stance optimization* module refines the trunk position and orientation in order to ensure reachability, guarantee stability, and maximize distance from obstacles.

Each terrain board used in LittleDog trials is supplied as a 3D triangle mesh. The Vicon motion capture system gives the pose of each terrain board in terms of position and orientation. To support fast collision detection and obstacle distance queries, the terrain is converted into a Signed Distance Field (SDF), $d(x)$, which stores the distance from a point $x \in \mathbb{R}^3$ to the boundary of the nearest obstacle. Values of $d(x)$ are negative inside obstacles, positive outside, and zero at the boundary.

To construct the SDF representation of the terrain, we begin by scan-converting triangle mesh models of the terrain into a voxelized grid. Then we compute the Euclidean Distance Transform (EDT) of the voxel grid and its logical complement. The signed distance field is then simply given by the difference of the two EDT values. Computing the EDT is surprisingly efficient: for a lattice of K samples, computation takes time $O(K)$ [15].

We set the grid resolution for the SDF to 5 mm. The resulting SDFs usually require about 10-20 megabytes of RAM to store. The scan-conversion and computation of the SDF is created as a preprocessing step before optimization, and usually takes under 5 seconds on commodity hardware.

Collision checking and distance queries approximate the geometry of the robot robot as a "skeleton" of spheres and capsules, or line-segment swept spheres. For a sphere of radius r with center x, the distance from any point in the sphere to the nearest obstacle is no less than $d(x) - r$. An analogous lower bound holds for capsules. There are a few key advantages of using the SDF to perform collision checking and distance queries. Queries are very fast, taking time proportional to the number of voxels occupied by the robot's "skeleton". Since the SDF is stored over the entire workspace, computing its gradient via finite differencing is a trivial operation.

The stance optimization module computes for each link of the robot (aside from the supporting feet) the point of closest distance to the terrain. If the distance is smaller than a preset threshold, the full-body inverse kinematics (as defined in section 4.1.3) are used to generate a differential translation $\Delta \mathrm{x}$ and rotation ω to move the robot away from the terrain. Additional translation and rotation components are generated in order to ensure that footholds are reachable given the robot's kinematics, and to make sure the robot's center of mass lies above the support polygon for the footstep.

Stance optimization iterates in this manner until convergence, or a preset number of iterations. Components are added to the pose cost based on final distance to obstacles, kinematic reachability, and distance from the center of mass to the support polygon. The trunk poses found during stance optimization are passed along to the walking trajectory generator as "hints" of where to place the robot during execution. In practice, using this optimization during footstep planning allows high performance because it carefully reasons about what poses LittleDog should or should not be able to attain.

4.2 Covariant Gradient Trajectory Optimization

Along with Nathan Ratliff and others, I have developed CHOMP, a novel method for continuous path refinement that uses covariant gradient techniques to improve the quality of sampled trajectories [43]. Like previous approaches [41, 9], CHOMP formalizes trajectory optimization as energy minimization via functional gradient descent. However, it offers two significant advantages over previous approaches. First, because we use SDF representations of workspace obstacles, the initial trajectory input to our method need not be strictly collision free. Second, convergence is accelerated through the use of a Covariant gradient approach.

The goal of CHOMP is to find a smooth, collision-free, trajectory through the configuration space $\mathbb{R}^m$ between two prespecified end points $q_{\text{init}},\ q_{\text{goal}} \in \mathbb{R}^m$. Let $q(t) : [0, 1] \mapsto \mathbb{R}^m$ be the trajectory in configuration space that we seek to optimize, with $q(0) = q_{\text{init}}$ and $q(1) = q_{\text{goal}}$.

Let $\mathcal{B}$ denote the set of points comprising the robot body. When the robot is in configuration q, the workspace location of the element $u \in \mathcal{B}$ is given by the function

$$x(q, u) : \mathbb{R}^m \times \mathcal{B} \mapsto \mathbb{R}^3$$

A trajectory for the robot is then collision-free if for every configuration q along the trajectory and for all $u \in \mathcal{B}$, the distance from $x(q,u)$ to the nearest obstacle is greater than $\varepsilon \geq 0$. If we use a signed distance field to store $d(x)$, the distance from a point $x \in \mathbb{R}^3$ to the boundary of the nearest obstacle, then we can create an obstacle potential function

$$c(x) = \max\big(\varepsilon - d(x), 0\big)$$

that repels the robot when it is closer than ε to an obstacle. Now, the functional that we seek to optimize is

$$\begin{aligned}
\mathcal{U}[q] &= f_{prior}[q] + f_{obs}[q] \\
f_{prior}[q] &= \frac{1}{2}\sum_{d=1}^{D} w_d \int_0^1 \|q^{(d)}(t)\|^2 dt \\
f_{obs}[q] &= \int_0^1 \int_u c\Big(x\big(q(t),u\big)\Big) \left\| \frac{d}{dt} x\big(q(t),u\big) \right\| du\, dt
\end{aligned}$$

Here, the prior term is simply a weighted sum of squared derivatives of the trajectory that encourages smoothness. The obstacle term is an arclength integration of the obstacle potential function $c(x)$ over every point of the robot as it sweeps through space. The reason for the arclength parameterization of this integral will become clear once we take the functional gradient of f_{obs}:

$$\bar{\nabla} f_{obs} \;=\; \int_u J^T \left[\|x'\| \left(\left(I - \hat{x}'\hat{x}'^T\right) \nabla c - c\kappa \right) \right] du$$

where κ is the curvature vector [14] defined as

$$\kappa = \frac{1}{\|x'\|^2} \left(I - \hat{x}'\hat{x}'^T\right) x''$$

and J is the kinematic Jacobian $\frac{\partial}{\partial q} x(q,u)$. To simplify the notation we have suppressed the dependence of J, x, and c on integration variables t and u. We additionally denote time derivatives of $x(q(t),u)$ using the traditional prime notation, and we denote normalized vectors by $\hat{x}$.

As written above, because the f_{obs} functional is written as an arclength parameterization, its functional gradient has the effect of taking the gradient of the obstacle potential, ∇c, and projecting away the components tangent to the motion of the robot before transforming it by the robot Jacobian. In practice, this has the effect that the obstacle potential function only induces motion perpendicular to the current trajectory, and never tries to alter the velocity profile along the trajectory.

To run the optimization, we represent the curve $q(t)$ as the trajectory as a discrete set of n waypoints representing samples spaced at even time intervals:

$$\xi = [\, q_1^T, \ldots, q_n^T \,]^T$$

In this form, we can map the functional gradient $\nabla\mathcal{U}[q]$ to the normal gradient $\nabla\mathcal{U}(\xi)$. Continuous derivatives are simply replaced by finite differencing operators. When we do so, we notice that given suitable finite differencing matrices K_d for $d = 1, \ldots, D$, we can represent the prior term $f_{prior}(\xi)$ as a sum of terms

$$f_{prior}(\xi) = \frac{1}{2}\sum_{d=1}^{D} w_d \left\|K_d\,\xi + e_d\right\|^2 ,$$

where e_d are constant vectors that encapsulate the contributions from the fixed end points. For instance, the first term ($d = 1$) represents the total squared velocity along the trajectory. In this case, we can write K_1 and e_1 as

$$K_1 = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ -1 & 1 & 0 & \ldots & 0 & 0 \\ 0 & -1 & 1 & \ldots & 0 & 0 \\ & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ldots & -1 & 1 \\ 0 & 0 & 0 & \ldots & 0 & -1 \end{bmatrix} \otimes I_{m\times m}$$

$$e_1 = \left[-q_0^T, 0, \ldots, 0, q_{n+1}^T\right]^T .$$

where $\otimes$ denotes the Kronecker (tensor) product. We note that f_{prior} is a simple quadratic form:

$$f_{prior}(\xi) = \frac{1}{2}\xi^T A\,\xi + \xi^T b + c$$

for suitable matrix, vector, and scalar constants A, b, c. We note that A is symmetric positive definite for all d.

The gradient update rule to optimize ξ is then

$$\xi_{k+1} = \xi_k - \alpha A^{-1} g_k$$

where α is a step size parameter and $g_k = \nabla\mathcal{U}(\xi)$. This update rule is a special case of a more general rule known as Covariant gradient descent [2, 58], in which the matrix A need not be constant. In our case, it is useful to interpret the action of the inverse operator A^{-1} as spreading the gradient across the entire trajectory. As an example, we take $d = 1$ and note that A is a finite differencing operator for approximating accelerations. Since $AA^{-1} = I$, we see that the ith column/row of A^{-1} has zero acceleration everywhere, except at the ith entry. $A^{-1}g_k$ can, therefore, be viewed as a vector of projections of g_k onto the set of smooth basis vectors forming A^{-1}.

We gain additional insight into the computational benefits of the covariant gradient based update by considering the analysis tools developed in the online learning/optimization literature and especially [57, 21]. Near a local optimum, f_{obs} is convex. Therefore under these conditions, the overall CHOMP objective function is *strongly convex* [45]– that is, it can be lower-bounded over the entire region by a quadratic with curvature A. [21] shows how gradient-style updates can be understood as sequentially minimizing a local quadratic approximation to the objective

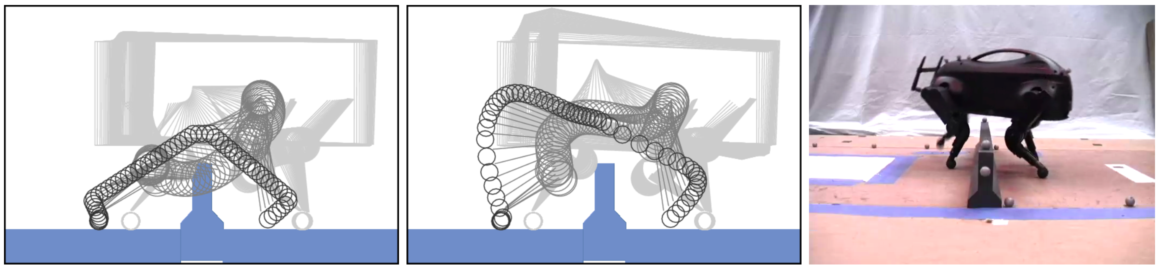


Figure 9: Optimizing a footstep over a barrier. *Left:* Initial trajectory estimate has severe knee collisions. *Center:* After optimization, collisions have been resolved by lifting the leg higher and tilting the body. *Right:* Execution on robot.

function. Gradient descent minimizes an uninformed, isotropic quadratic approximation while more sophisticated methods, like Newton steps, compute tighter lower bounds using a Hessian. In the case of CHOMP, the Hessian need not exist as our objective function may not even be differentiable, however we may still form a quadratic lower bound using A. This is much tighter than an isotropic bound and leads to a correspondingly faster minimization of our objective– in particular, in accordance with the intuition of adjusting large parts of the trajectory due to the impact at a single point we would generally expect it to be $O(n)$ times faster to converge than a standard, Euclidean gradient based method that adjusts a single point due an obstacle.

4.2.1 CHOMP for LittleDog

Footstep trajectories for LittleDog consist of a stance phase, where all four feet have ground contact, and a swing phase, where the swing leg is moved to the next support location. During both phases, the robot can independently control all six degrees of trunk position and orientation via the supporting feet. Additionally, during the swing phase, the three degrees of freedom for the swing leg may be controlled. For a given footstep, we run CHOMP as coordinate descent, alternating between first optimizing the trunk trajectory ξ_T given the current swing leg trajectory ξ_S, and subsequently optimizing ξ_S given the current ξ_T on each iteration. The initial trunk trajectory is given by a Zero Moment Point (ZMP) preview controller [28], and the initial swing leg trajectory is generated by interpolation through a collection of knot points intended to guide the swing foot a specified distance above the convex hull of the terrain.

When running CHOMP with LittleDog, we exploit domain knowledge by adding a prior to the workspace potential function $c(x)$. The prior is defined as penalizing the distance below some known obstacle-free height when the swing leg is in collision with the terrain. Its effect in practice is to add a small gradient term that sends colliding points of the robot upwards regardless of the gradient of the SDF.

For the trunk trajectory, in addition to the workspace obstacle potential, the objective function includes terms which penalize kinematic reachability errors (which occur when the desired stance foot locations are not reachable given desired trunk pose) and which penalize instability resulting from the computed ZMP straying towards the edges of the supporting polygon. Penalties from the additional objective function terms are also multiplied through A^{-1} when applying the gradient, just as the workspace potential is.

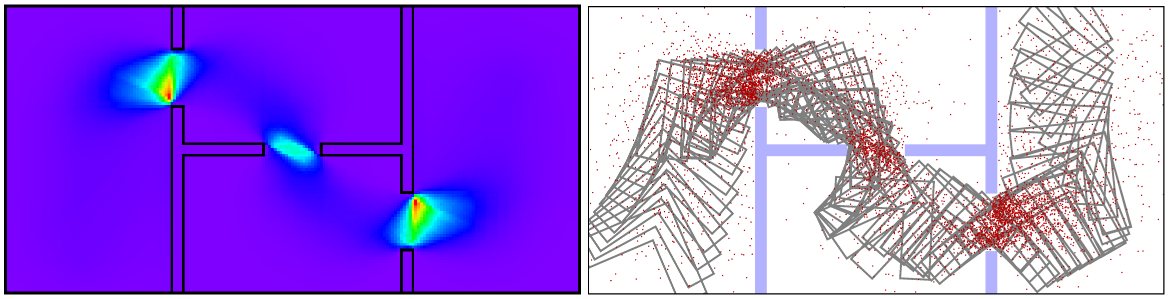


Figure 10: Workspace-biased sampling distribution used to plan paths for a planar rotating and translating *L*-beam. *Left:* automatically learned probability distribution. *Right:* samples drawn from distribution (red) shown along with robot path (gray). Although the scene pictured above has three doorways, the distribution was learned from examples containing only a single doorway, demonstrating ability to generalize across environments.

Although we typically represent the orientation of the trunk as a unit quaternion, we represent it to CHOMP as an exponential map vector corresponding to a differential rotation with respect to the "mean orientation" of the trajectory. The exponential map encodes an (axis, angle) rotation as a single vector in the direction of the rotation axis whose magnitude is the rotation angle. The mean orientation is computed as the orientation halfway between the initial and final orientation of the trunk for the footstep. Because the amount of rotation over a footstep is generally quite small (under 30°), the error between the inner product on exponential map vectors and the true quaternion distance metric is negligible.

Timing for the footstep is decided by a heuristic which is evaluated before the CHOMP algorithm is run. Typical footstep durations run between 0.6 s and 1.2 s. We discretize the trajectories at the LittleDog host computer control cycle frequency, which is 100 Hz. Currently, trajectories are pre-generated before execution because in the worst-case, optimization can take slightly longer (by a factor of about 1.5) than execution. We have made no attempt to parallelize CHOMP in the current implementation, but we expect performance to scale nearly linearly with the number of CPUs.

As shown in figure 9, the initial trajectory for the footstep is not always feasible; however, the CHOMP algorithm is almost always able to find a collision-free final trajectory, even when the initial trajectory contains many collisions.

4.3 Learned Heuristics

During earlier work with LittleDog, I became interested in using RRTs to generate trajectories for difficult footsteps. Although the current software no longer uses RRTs, that line of investigation produced some interesting results on learning sampling distributions for randomized planners [59]. For a probabilistic motion planner, the sampling distribution can be viewed as comparable to heuristic functions used by deterministic planners; the correct one can make a crucial difference in performance.

Although probabilistically complete, in practice, sampling-based planners commonly suffer from the so-called "narrow passage problem": the relatively low probability of sampling states that extend the graph through small gaps in free configuration space [26]. Much effort has been dedicated to developing sampling strategies for improved planning, based on either configuration space or workspace features.

Configuration space samplers include the bridge test sampler to identify samples within narrow passages [25], approximate medial axis sampling [55], and rejection of samples within the Voronoi region of near-colliding states [56]. Characteristics of specific robotic platforms such as manipulability can be used as well [37]. For dynamic re-planning applications, the sampling distribution can be biased towards previously useful states [10], prompting the use of nearest-neighbor classifiers to search among many past queries for relevant information [27]. Since many sampling strategies themselves take as input samples from an underlying distribution, their strengths can be combined by setting up chains of dependent samplers [51].

Workspace biasing methods include the Gaussian sampler for sampling near workspace obstacles [5], as well as the workspace medial axis approach [24]. More recent methods have focused on geometric analysis to identify workspace narrow passages [35, 53].

Despite all the effort invested in developing sampling strategies, experiments have shown that there is no “one-size-fits-all” sampler that gives optimal performance across all classes of planning problems [17, 18]. The sampling strategy here plays a role reminiscent of the learned heuristic functions in [46, 54]: an informed one can be crucial for effective planning. A key idea in our work is that the sampling strategy adopted by a probabilistic sample-based planner can be understood as a stochastic policy in the sense common in the field of Reinforcement Learning[3].

Our framework produces a workspace-biased distribution that yields good performance on a given class of planning problems based on a weighting of workspace features. Aside from a sampling-based planner, our approach requires: a discretization of the workspace into a set $\mathcal{X}$ of finite elements; a feature vector $f(x) : \mathcal{X} \mapsto \mathbb{R}^K$ evaluated at each $x \in \mathcal{X}$; a distribution $p(y|x)$ over robot configurations given a particular $x \in \mathcal{X}$; and finally a reward function $r(\xi)$ which assigns rewards based on planning performance given the samples $\xi = \{x^{(1)}, \ldots, x^{(T)}\}$. Since the general problem formulation is not specifically targeted at a particular feature set or randomized planner, we note that the methods presented here should generalize across a variety of probabilistic planning algorithms and features beyond those used here.

The overall course of the approach is as follows: Invoke sampling-based planner on an instance of the problem class. To generate samples, the planner should first sample an element $x \sim q(\theta, x)$ based on a weighting θ of the workspace features $f(x)$, and then sample a configuration y from $p(y|x)$. At the end of a planning episode, assign a reward $r(\xi)$. Finally, estimate the gradient of the expected reward with respect to the weights θ, take a gradient step, and start again.

We create a workspace-biased probability distribution $q(\theta, x)$ that samples in a discretization $\mathcal{X}$ of the workspace into finite elements. The distribution $q(\theta, x)$, a member of the exponential family of distributions, uses the weight vector θ to bias the salience of various features in the workspace. Our current implementation uses a uniform voxel discretization of the workspace, but non-uniform representations of the workspace (i.e. quadtrees, octrees, KD-trees) are in practice equally viable. For each workspace element $x \in \mathcal{X}$, we compute a feature vector $f(x) \in \mathbb{R}^K$.

Now we define the distribution $q(\theta, x)$ as

$$\begin{aligned} q(\theta, x) &= \frac{1}{Z(\theta)} \exp\left(\theta^T f(x)\right) \\ Z(\theta) &= \sum_{x_i \in \mathcal{X}} \exp\left(\theta^T f(x_i)\right) \end{aligned}$$

Note that $Z(\theta)$ serves as a normalizing term so that the probabilities sum to one. The distribution $q(\theta, x)$ is a Gibbs distribution, and has several useful properties: When $\theta = 0$, $q(\theta, x)$ becomes the uniform distribution. As θ_j approaches ∞, the distribution holds nonzero probabilities only where $f_j(x)$ reaches its maximum value. Given a uniform discretization over the workspace, as the cell size approaches zero, $q(\theta, x)$ approaches a continuous probability density function.

The distribution for a sampling-based planner can be viewed as a stochastic policy. We are interested in the expected reward for a given planning episode:

$$\eta(\theta) = E_q[r(\xi)]$$

Given that the reward is assigned at the end of the episode, and given that all samples are drawn from q, we can approximate the policy gradient with respect to θ by multiplying the reward for the planning episode by the average score ratio over all samples:

$$\widehat{\nabla} \eta(\theta) = \frac{r(\xi)}{T} \sum_{t=1}^{T} \frac{\nabla q(\theta, x^{(t)})}{q(\theta, x^{(t)})}$$

In the case of the distribution $q(\theta, x)$ described above, the score ratio is the difference between the feature vector at a particular element and the expectation of the feature vector under the distribution:

$$\begin{aligned} \frac{\nabla q(\theta, x)}{q(\theta, x)} &= f(x) - E_q[f] \\ &= f(x) - \sum_{x_i \in \mathcal{X}} f(x_i)\, q(\theta, x_i) \end{aligned}$$

The update rule for θ is then

$$\theta_{new} = \theta_{old} + \alpha\, \widehat{\nabla} \eta(\theta)$$

where α is a small positive constant.

By writing the reward function to maximize number of problems solved per unit time, we observed a $7\times$ speedup over an unbiased planner when planning a 3D translation and rotation "piano mover" type problem (example shown in figure 11).

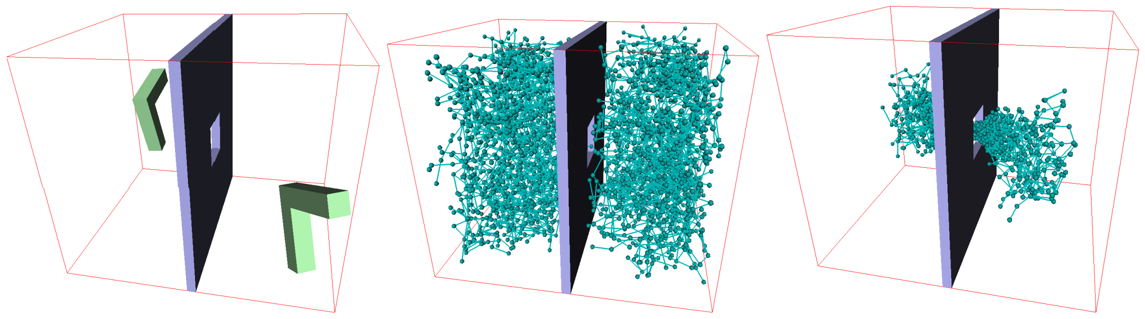


Figure 11: Comparing search trees for the 3D (6DOF) L-beam planning problem. *Left:* start and goal configurations for beam. *Center:* tree built with uniform sampling (2924 nodes). *Right:* tree built with adaptive workspace biasing (404 nodes).

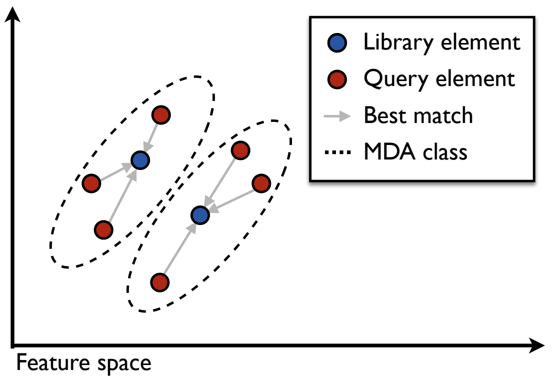


Figure 12: Illustration of labeling scheme for training MDA. Library elements x_i are shown in blue. The corresponding query elements y_j are shown in red. MDA will correctly separate the data, but PCA will not.

5 Proposed Work

5.1 Behavior Library and Behavior Recall

The current LittleDog footstep planner is not quite capable of running in real-time, mostly due to the amount of time spent in trajectory generation and optimization. I believe that efficient behavior recall will help move the current footstep planner towards a real-time implementation by finding initial trajectories that are much closer to optimal than our current heuristic guesses (pictured at left of figure 9).

Behavior recall can be formalized as efficient nearest neighbor classification. For a given state-action pair (x, u), provided by the planner, the behavior recall module should select the nearest behavior in the library.

As discussed in [50, 4], behavior matching of this type can be either global or local. Global matching would compare states by reference to absolute coordinates on a particular terrain board, whereas local matching extracts features $f(x)$ from the vicinity of the robot and tries to compare them to novel environments. Although local matching is more complicated to implement due to the need to encode context, it promises to be more general than a global matching approach.

If the dimension of the state-action pair is high (for instance, storing a representation of the terrain around the robot), dimensionality reduction techniques may be utilized. Multiple Discriminant Analysis (MDA) could be very useful in this context. MDA is a supervised classification algorithm that identifies a low-dimensional basis that maximizes the ratio of between-class variance to among-class variance. MDA can significantly outperform other dimension reduction algorithms such as principal component analysis (PCA) in classification tasks. In the behavior recall system, the "class" of a query state-action pair (x, u) indicates the action that we wish the behavior recall module would have selected for a the pair.

A possible algorithm for labeling data for input to MDA in the footstep trajectory domain might go as follows: We are given a library that consists of the extracted features $f(x_i)$ and footstep trajectories ξ_i for every element $i \in \{1, \ldots, n\}$ of the library. We also generate a set of query states $y_1, \ldots, y_m$ to match against the library. For each state y_j we find in the library the state $x^*(y_j)$ that corresponds to the trajectory ξ^* which minimizes the objective function of the trajectory optimizer when given as input. Then the class of an element x_i is the set of states $\{y_j | x_i = x^*(y_j)\}$. See figure 12 for an illustration.

Obviously, since global matching is much easier than local matching, it would be good to verify that global behavior recall works before commencing work on local matching.

5.2 Capability Model

We want high level planners to be able to adapt their estimates of action costs to reflect the true capabilities of the robot. A reasonable first step in this direction would be to come up with a quantitative metric of robot performance that can be computed to discriminate between successful and poor actions. Quantities to consider might include catastrophic failures (slips and falls) as well as subtler metrics—quality of trajectory tracking, proximity to obstacles, extreme accelerations or impacts, etc. If these measures of execution quality are combined into a single scalar value, various function approximation or regression schemes could be used to map state-action pairs.

Say the scalar quality metric represents probability of failure; then logistic regression may well be sufficient to produce reasonable cost values. Like behavior recall, the local vs. global issue becomes salient here as well. Again, some effort needs to be devoted to finding a suitable representation of features to use as input for the regressor.

Since the planner will tend to execute low-cost actions in preference of high-cost actions, developing this module recalls the "exploration vs. exploitation" dilemma familiar to those in the field of reinforcement learning. The robot risks missing out on very good actions if they are mistakenly labeled as high-cost during the learning process.

One possibility is to enforce some exploration by selecting actions stochastically, weighted by their cost:

$$p(u|x) \propto e^{-\lambda\, c(x,u)}$$

This stochastic action selection would be enabled for learning purposes only (not for high-performance execution), and it would have the benefit of occasionally selecting what seem like poor actions in order to confirm their high cost. A natural learning scheme in this framework is a REINFORCE-style algorithm similar to that used in [59].

5.3 Heuristic Cost-to-go

In order to make the system adaptive from top to bottom, the last step is to learn a heuristic cost-to-go estimate. Maximum margin planning (MMP) is a promising starting candidate. To implement MMP for footstep planning, I would probably start by using a 3D (x, y, θ) grid-based planner along with terrain features similar to those discussed in section 4.1.5. Using A* to train the MMP should be very fast in this reduced-dimension space.

5.4 Heterogeneous Behaviors

In order to overcome the more difficult terrain boards in the current phase of the DARPA Learning Locomotion project, it is likely that additional behavior modalities beyond a simple crawl gait will be required. Possibilities include trotting, hopping, or sliding. Perhaps the most major impact of introducing heterogeneous behaviors is to change the dedicated footstep planner into a general, high level planner.

This would also require updating the behavior recall, capability model and heuristic modules.

5.5 Additional Robotic Platform

In order to demonstrate generality beyond a single robotic platform, I intend to use the system to plan for another high-DOF robot in the domain of legged locomotion and/or mobile manipulation. My research group has access to Sarcos and HRP-2 humanoid robots, as well as the Intel personal robotics research platform, which consists of a WAM arm with a Barrett hand, mounted on a Segway base.

However the second platform ends up being selected, the important aspect of the research is to demonstrate that the proposed system can be adapted to another robot without undue effort.

5.6 Evaluation

Before delving too deeply into developing functionality, I will formalize the criteria for evaluating the success of the proposed system beyond those outlined in section 1.2. In particular, I will identify the aspects of the system likely to provide the best value in terms of improvement over the current state of the art.

The final system will be compared against a best-effort, hard-coded implementation which implements comparable functionality without the adaptive components. Quantitative measurements will likely include computation effort (including storage overhead) and measurements of robot performance: slips and falls, tracking errors, execution time and quality.

Beyond quantitative measurement, a more subtle evaluation criterion is ease of use—how general is the system? Is it prohibitively difficult to add new behaviors or modify existing ones?

6 Conclusion

The proposed work should contribute a novel approach to high level planning which selects among heterogeneous behaviors for complex robotic tasks. The work will overcome limitations of existing high level planning approaches by using machine learning techniques to adapt to the capabilities of the robot over time.

The generality of the data-driven high level planning system will be demonstrated through implementation on the LittleDog quadruped robot as well as on another high-DOF robotic platform.

Schedule of Work

A rough schedule of work follows:

- **Jan 2009** – Proposal.
- **Feb 2009** – Formalize evaluation criteria (section 5.6).
- **Mar 2009** – Behavior recall with global matching (section 5.1).
- **Apr 2009** – Behavior recall with local feature-based matching using MDA (section 5.1).
- **May 2009** – End of DARPA Learning Locomotion project. Identify an additional robot platform to demonstrate generality (section 5.5). Demonstrate capability model based on logistic regression (section 5.2).
- **Jun 2009** – Implement MMP to provide heuristic estimates to high-level planner (section 5.3).
- **Jul-Sep 2009** – Begin porting functionality to second robotic platform (section 5.5).
- **Sep 2009** – Investigate REINFORCE-style algorithm for adaptively learning action costs (section 5.2). Finish developing LittleDog functionality.
- **Oct-Nov 2009** Finish developing functionality for second robotic platform section 5.5).
- **Dec 2009-Mar 2010** Evaluation (section 5.6) and dissertation writeup.
- **Mar 2010** – Thesis defense.

References

[1] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM New York, NY, USA, 2004.

[2] J. Andrew Bagnell and Jeff Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August 2003.

[3] J. Baxter and PL Bartlett. Direct gradient-based reinforcement learning. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 3, 2000.

[4] Darrin Bentivegna. *Learning from Observation Using Primitives*. PhD thesis, Georgia Institute of Technology, 2004.

[5] V. Boor, M.H. Overmars, and A.F. van der Stappen. Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 2, pages 1018–1023, 1999.

[6] Boston Dynamics Inc. BigDog sets new world record. Press release, Oct 2008.

[7] M.S. Branicky, R.A. Knepper, and J.J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *IEEE International Conference on Robotics and Automation*, pages 1359–1364, 2008.

[8] T. Bretl, S. Rock, J.C. Latombe, B. Kennedy, and H. Aghazarian. Freeclimbing with a multi-use robot. In *International Symposium on Robotics Research*. Springer, 2004.

[9] O. Brock and O. Khatib. Elastic Strips: A Framework for Motion Generation in Human Environments. *The International Journal of Robotics Research*, 21(12):1031, 2002.

[10] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, Switzerland, October 2002. An earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.

[11] M. Buehler, R. Playter, and M. Raibert. Robots step outside. In *Proceedings of the International Symposium on Adaptive Motion in Animals and Machines (AMAM),(Ilmenau, Germany), September*, 2005.

[12] PC Chen and YK Hwang. SANDROS: a dynamic graph search algorithm for motion planning. *Robotics and Automation, IEEE Transactions on*, 14(3):390–403, 1998.

[13] Joel Chestnutt. *Navigation Planning for Legged Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2007.

[14] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentice-Hall, 1976.

[15] P. Felzenszwalb and D. Huttenlocher. Distance Transforms of Sampled Functions. Technical Report TR2004-1963, Cornell University, 2004.

[16] R. Geraerts and M. Overmars. Creating High-quality Roadmaps for Motion Planning in Virtual Environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4355–4361, 2006.

[17] R. Geraerts and M.H. Overmars. A comparative study of probabilistic roadmap planners. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2004.

[18] R. Geraerts and M.H. Overmars. Sampling Techniques for Probabilistic Roadmap Planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 10–13, 2004.

[19] PE Hart, NJ Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[20] K. Hauser, T. Bretl, and J.C. Latombe. Non-gaited humanoid locomotion planning. *Humanoids, Tsukuba, Japan*, 2005.

[21] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. In *In COLT*, pages 499–513, 2006.

[22] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, 1, 1999.

[23] D. Hobbelen, T. de Boer, and M. Wisse. System overview of bipedal robots Flame and TUlip: Tailor-made for Limit Cycle Walking. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2486–2491, 2008.

[24] C. Holleman and LE Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 2, 2000.

[25] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 3, 2003.

[26] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, pages 141–153. AK Peters, Ltd. Natick, MA, USA, 1998.

[27] X. Jiang and M. Kallmann. Learning humanoid reaching tasks in dynamic environments. *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1148–1153, 2007.

[28] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, 2003.

[29] Maciej Kalisiak and Michiel van de Panne. A grasp-based motion planning algorithm for character animation. *The Journal of Visualization and Computer Animation*, 12(3):117–129, 3001.

[30] L. Kavraki and J.C. Latombe. Probabilistic roadmaps for robot path planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, 53, 1998.

[31] L. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, 1996.

[32] J. Kivinen, AJ Smola, and RC Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.

[33] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Transactions on Graphics (TOG)*, 21(3):473–482, 2002.

[34] J.J. Kuffner and S.M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, San Francisco, CA, April 2000.

[35] H. Kurniawati and D. Hsu. Workspace importance sampling for probabilistic roadmap planning. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, volume 2, 2004.

[36] Manfred Lau and James J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 299–308, September 2006.

[37] P. Leven and S. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *Robotics and Automation, IEEE Transactions on*, 19(6):1020–1026, 2003.

[38] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, 16, 2004.

[39] A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

[40] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *IEEE International Conference on Robotics and Automation*, pages 802–807, 1993.

[41] Sean Quinlan. *The Real-Time Modification of Collision-Free Paths*. PhD thesis, Stanford University, 1994.

[42] N. Ratliff, J. Bagnell, and S. Srinivasa. Imitation Learning for Locomotion and Manipulation. In *IEEE-RAS International Conference on Humanoid Robots*, 2007.

[43] N. Ratliff, M. Zucker, D. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (submitted)*, 2009.

[44] Nathan Ratliff, J. Andrew Bagnell, and Margin Zinkevich. Maximum margin planning. In *Twenty Second International Conference on Machine Learning (ICML06)*, 2006.

[45] Nathan Ratliff, James (Drew) Bagnell, and Martin Zinkevich. (online) subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*, March 2007.

[46] Nathan Ratliff, David Bradley, J. Andrew Bagnell, and Joel Chestnutt. Boosting structured prediction for imitation learning. In *NIPS*, Vancouver, B.C., December 2006.

[47] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), August 2007.

[48] P. Soueres and JD Boissonnat. Optimal Trajectories for Nonholonomic Mobile Robots. *Lecture Notes in Control and Information Sciences*, pages 93–170, 1998.

[49] A. Stentz. The focussed D* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

[50] Martin Stolle. *Finding and Transferring Policies Using Stored Behaviors*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2008.

[51] S. Thomas, M. Morales, X. Tang, and N.M. Amato. Biasing Samplers to Improve Motion Planning Performance. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1625–1630, 2007.

[52] N. Torkos and M van de Panne. Footprint-based quadruped motion synthesis. In *Graphics Interface '98*, June 1998.

[53] J.P. van den Berg and M.H. Overmars. Using Workspace Information as a Guide to Non-uniform Sampling in Probabilistic Roadmap Planners. *International Journal of Robotics Research*, 24(12):1055, 2005.

[54] Y. Xu, A. Fern, and S. Yoon. Discriminative learning of beam-search heuristics for planning. In *Proceeding of the International Joint Conference on Artifical Intelligence*, 2007.

[55] Y. Yang and O. Brock. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 5, 2004.

[56] A. Yershova, L. Jaillet, T. Simeon, and SM LaValle. Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3856–3861, 2005.

[57] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 928–936, 2003.

[58] Mark Zlochin and Yoram Baram. Manifold stochastic dynamics for bayesian learning. *Neural Comput.*, 13(11):2549–2572, 2001.

[59] M. Zucker, J. Kuffner, and D. Bagnell. Adaptive workspace biasing for sampling based planners. In *IEEE International Conference on Robotics and Automation*, 2008.