Optimization and Learning for Rough-Terrain Legged Locomotion

Matt Zucker[1], Nathan Ratliff[2], Martin Stolle[3], Joel Chestnutt[4],
J. Andrew Bagnell[1], Christopher G. Atkeson[1], James Kuffner[1]

[1]The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{*mzucker, dbagnell, cga, kuffner*}*@cs.cmu.edu*

[2] Toyota Technical Institute
6045 S. Kenwood Avenue, Chicago IL 60637
ndr@tti-c.org

[3] Google, Inc.
Brandschenkestrasse 110
8002 Zürich, Switzerland
mstoll@google.com

[4] Digital Human Research Center
National Institute of Advanced
Industrial Science and Technology
2-41-6, Aomi, Koto-ku, Tokyo, 135-0064, Japan
joel.chestnutt@aist.go.jp

Abstract

We present a novel approach to legged locomotion over rough terrain that is thoroughly rooted in optimization. This approach relies on a hierarchy of fast, anytime algorithms to plan a set of footholds, along with the dynamic body motions required to execute them. Components within the planning framework coordinate to exchange plans, cost-to-go estimates, and “certificates” that ensure the output of an abstract high-level planner can be realized by lower layers of the hierarchy. The burden of careful engineering of cost functions to achieve desired performance is substantially mitigated by a simple inverse optimal control technique. Robustness is achieved by real-time re-planning of the full trajectory, augmented by reflexes and feedback control. We demonstrate the successful application of our approach in guiding the LittleDog quadruped robot over a variety of rough terrains. Other novel aspects of our past research efforts include a variety of pioneering inverse optimal control techniques as well as a system for planning using arbitrary pre-recorded robot behaviors.

1 Introduction

Legged locomotion has been a key area of robotics research since nearly the inception of the field (Raibert, 1986); however, robots that can skillfully and deliberatively traverse rough terrain have yet to be fielded

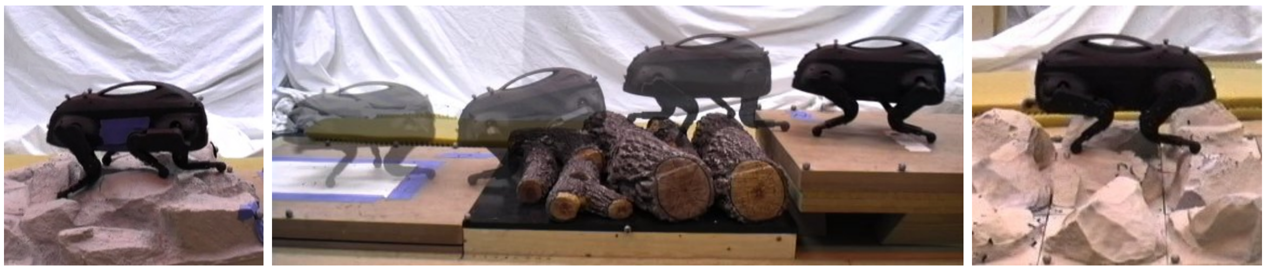


Figure 1: Boston Dynamics Inc. LittleDog quadruped robot, shown crossing several types of rough terrain.

outside of the laboratory. Some outdoor robots are beginning to address this problem by using inherent mechanical stability and/or well-tuned feedback control to overcome small- to medium-sized obstacles (Saranli et al., 2001; Buehler et al., 2005). Although both mechanism design and low-level control are vitally important, ultimately what is needed is a suite of algorithms for rough terrain locomotion, capable of reasoning about not only the path the robot should take to navigate through the world, but what actions the robot should perform to realize the path safely.

The Robotics Institute is one of six institutions participating in the DARPA Learning Locomotion project, aimed at producing algorithms for robust legged locomotion over rough terrain. Participants develop software for the LittleDog robot, designed and manufactured by Boston Dynamics, Inc., and the software is evaluated upon the speed with which the robot can cross a set of standardized terrains. Each phase of the project has presented successively more difficult terrain to traverse as well as a higher "goal" speed metric to pass. By the end of Phase III, LittleDog was required to traverse obstacles up to 10.8 cm height at speeds in excess of 7.2 cm/s. Scaled up to human size, this is analogous to moving over hip-height boulders at just below a normal walking pace.

The LittleDog robot is driven by geared motors, with two degrees of freedom at each hip, and one at the knee. Aside from onboard joint encoders and three-axis force sensors in each foot, sensing is provided by a multi-camera motion capture system that tracks the position and orientation of the robot and the terrains. The robot is controlled wirelessly by an offboard control computer, and the control software has access to detailed 3D triangle mesh models of the terrain.

1.1 Research contributions

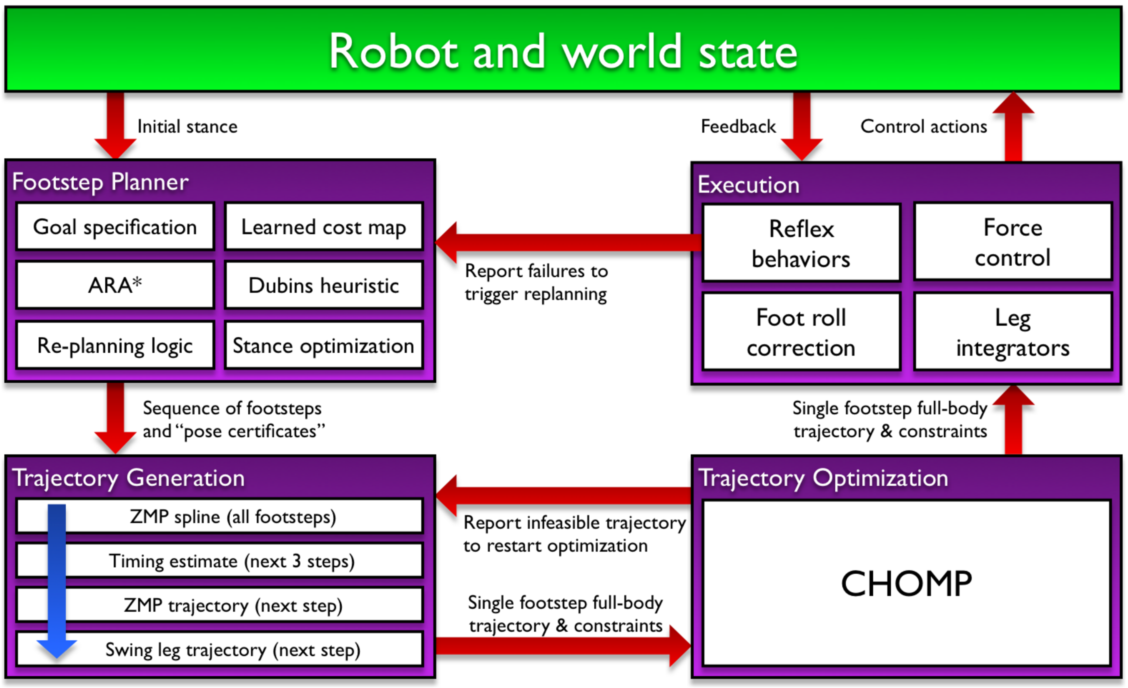
In this article, we present our planning and control software for LittleDog. Our approach to rough terrain locomotion is thoroughly rooted by optimization. Where possible, optimization is augmented by the use of machine learning techniques which serve not only to increase performance, but also to relieve system designers from the burden of tedious and error-prone tasks. Our software has been used successfully throughout the Learning Locomotion project in order to traverse a broad range of terrains and our methods constitute a number of novel achievements in the fields of planning, imitation learning, and optimization. In the first year of the program, we developed what we believe is the first ever implementation of footstep planning on a rough terrain robot Chestnutt (2007), as well as the first general solution of an Inverse Optimal Control problem on a robotic platform Ratliff et al. (2007c).

More recently, we have developed a framework to record and play back specialized behaviors and to generalize behaviors to novel situations. This "trajectory library" approach is the first of its kind to reason about full-body obstacle avoidance (Stolle, 2008). We have also developed a general preference learning framework that relieves the system designer from the burden of manually specifying a cost function for an optimal planner (Zucker, 2009). Finally, we have developed CHOMP, a numerical optimization tehnique to make sure that the trajectories commanded to the robot are smooth, stable and collision free (Ratliff et al., 2009b).

The strength of these contributions has been validated by other teams' decisions to adopt and extend our strategies in their own approaches. Most prominently, many teams ended up using a footstep planning approach to navigate the robot towards the goal (Kolter et al., 2008). Others have also extended our method of formulating preference learning of cost functions as a ranking problem by considering terrain templates as additional high-dimensional features (Kalakrishnan et al., 2009). Furthermore, our approach to trajectory optimization has been adopted as well (Kalakrishnan, 2009).

1.2 Organization of this article

In the next section, we give a general overview of the system architecture. Section 3 reviews primitive operations for kinematics and obstacle avoidance. Sections 4 and 5 detail footstep planning and footstep trajectory optimization. In section 6 we describe low-level control strategies. Specialized behaviors for non-gaited locomotion are reviewed in section 7, and additional imitation learning work is described in section 8.



Robot and world state
Initial stance
Feedback
Control actions
Footstep Planner
Goal specification
Learned cost map
ARA*
Dubins heuristic
Re-planning logic
Stance optimization
Report failures to trigger replanning
Execution
Reflex behaviors
Force control
Foot roll correction
Leg integrators
Sequence of footsteps and "pose certificates"
Single footstep full-body trajectory & constraints
Trajectory Generation
ZMP spline (all footsteps)
Timing estimate (next 3 steps)
ZMP trajectory (next step)
Swing leg trajectory (next step)
Report infeasible trajectory to restart optimization
Single footstep full-body trajectory & constraints
Trajectory Optimization
CHOMP

Figure 2: System block diagram. A hierarchy of planners and optimizers generates complex full-body locomotion behavior.

For the most part, this article describes the functionality of the current software as of the final Learning Locomotion trials; however, it is important to note that the work described in this document represents the combined effort of a number of researchers over a time frame of more than three years. Hence, not all results presented here have been combined into a single software architecture. The techniques described in sections 7.1 and 8 in particular are not implemented in the current software, but we include them here because the research is nonetheless valuable to the overarching theme of the Learning Locomotion project.

2 System Overview

Our approach to system design is strongly motivated by the desire to build an optimal controller for rough terrain locomotion. Due to the complexity of the task and the high degree of actuation of the robot, fully optimizing over the space of all possible robot motions is intractable; instead, we decompose the problem into a pipeline of smaller optimization tasks.

The components of the system are depicted in figure 2. First, the system selects a sequence of footholds on the terrain through the use of a footstep planner, which is guided by a cost map learned from expert preferences. The footstep plan is realized by generating full-body trajectories, which are passed through an optimizer in order to maximize clearance from obstacles and ensure smooth, dynamically stable motion. The system operates in real-time, capable of recovering and re-planning in the event of slips or accidental collisions with the terrain.

The theme of nesting planners within planners in a hierarchy of representations ranging from abstract to concrete arises in multiple places throughout our framework. Our footstep planner solves the task of navigation at an abstract level relative to the full-body trajectory optimizer; however, the footstep planner itself is informed by a heuristic that models the robot as a yet simpler wheeled vehicle.

Choosing the correct problem decomposition is crucial. Other participants in the Learning Locomotion project have chosen to plan a path for the robot's trunk at the lowest level (Kolter and Ng, 2007; Kolter et al., 2008; Kalakrishnan et al., 2009). After these systems make a hard commitment to a particular body path, they choose footholds along the path, and finally generate full-body trajectories to execute the footsteps. We believe our software is capable of walking over a wider range of terrains (particularly long gaps or chasms) because the body path planner at the most abstract level of our competitors' software is too conservative relative to the full range of possible footholds.

Where possible, we avoid hard commitments to decisions at an abstract level that might lead to mistakes at a deeper level of the hierarchy. We prefer instead to rate alternatives with continuous cost functions, and to "lift up" the intermediate results of abstract planners into the full problem space to minimize the need for backtracking through the pipeline. We view the use of continuous heuristics such as the wheeled vehicle heuristic as more general than making fixed choices about problem decomposition because heuristics allow the overall planning approach to retain flexibility in adapting to the terrain while still guiding the search with domain specific knowledge about the capabilities of the platform.

3 Kinematics and Obstacle Avoidance

In this section, we describe two low-level optimization routines which are used by a variety of system components in order to ensure kinematic feasability, and to maximize clearance from obstacles. Given a full 18-DOF robot configuration consisting of body position and orientation, as well as all four sets of leg joint angles, both of these routines find a nearby body position and orientation and leg joint angles that accomplish the objective (kinematic feasibility or obstacle avoidance) while leaving the four foot positions unchanged.

Another way to envision the effect of these routines is constrained gradient descent; keeping the four foot positions unchanged imposes a 12-dimensional constraint manifold in the 18-DOF configuration space. These routines identify the steepest descent directions of the objectives along the tangent of the constraint manifold.

In our current software architecture, neither routine is used directly for control of the robot; instead, both routines are designed to be run iteratively as drivers for a gradient-based search scheme. Both the "certificate" search procedure described in section 4.4 as well as the trajectory optimization described in section 5 use these low-level routines to generate gradients.

3.1 Conventions and coordinate frames

Commonly used variable notations:

$\mathbf{x}$	Robot position, defined as center point of hip joints
$\mathbf{R}$	Robot rotation matrix
ω	Angular rates or differential rotation
$\mathbf{f}$	Foot position, defined as center of spherical foot
$\mathbf{p}$	Arbitrary reference point on robot
$\mathbf{q}$	Joint angles for leg
i	Leg index

Subscripts are used to index legs, e.g. $\mathbf{q}_i$. Subscripts W and B are used to denote either *world* or *body* frame for vectors, e.g. $\mathbf{p}_W$. 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,

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

The inverse kinematics solution of the joint angles to place the foot of leg i at position $\mathbf{f}_i$ given a body pose $(\mathbf{x}, \mathbf{R})$ is written as

$$\begin{aligned} \mathbf{q}_i &= \mathrm{IK}_W(\mathbf{x}, \mathbf{R}, \mathbf{f}_i) \\ &= \mathrm{IK}_B\big(\mathbf{R}^T(\mathbf{f}_i - \mathbf{x})\big) \end{aligned}$$

If no solution is possible, the IK is assumed to return the closest reachable position. The forward kinematics to reconstruct foot position from joint angles is denoted

$$\begin{aligned} \mathbf{f}'_i &= \mathrm{FK}_W(\mathbf{x}, \mathbf{R}, \mathbf{q}_i) \\ &= \mathbf{R}\,\mathrm{FK}_B(\mathbf{q}_i) + \mathbf{x} \end{aligned}$$

Finally, we 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 $\mathbf{v}$.

3.2 Increasing kinematic reachability

At times, given the desired foot positions, the current body pose $(\mathbf{x}, \mathbf{R})$ may be kinematically infeasible in the sense that there exists no set of joint angles $\mathbf{q}_i$ for leg i that leaves the foot at the desired foot position $\mathbf{f}_i$. This could be because the body pose was determined by an initial heuristic that incorrectly positioned the body, or because other optimization objectives (such as obstacle avoidance or stability) have driven the body past the limit of kinematic reachability.

In this case, we improve reachability by imposing a virtual force on the body at the best achievable position of foot i, with direction moving it towards the desired position. Let $\mathbf{q}_i$ be the closest reachable position returned by $\mathrm{IK}_W(\mathbf{x}, \mathbf{R}, \mathbf{f}_i)$, and let $\mathbf{f}'_i$ be the forward kinematic position corresponding to $\mathbf{q}_i$. Then set $\mathbf{e}_i = \mathbf{f}_i - \mathbf{f}'_i$ to be the difference between the desired and actual foot positions. To impose the virtual force, we compute the body translation and rotation displacements $(\Delta\mathbf{x}, \omega)$ as:

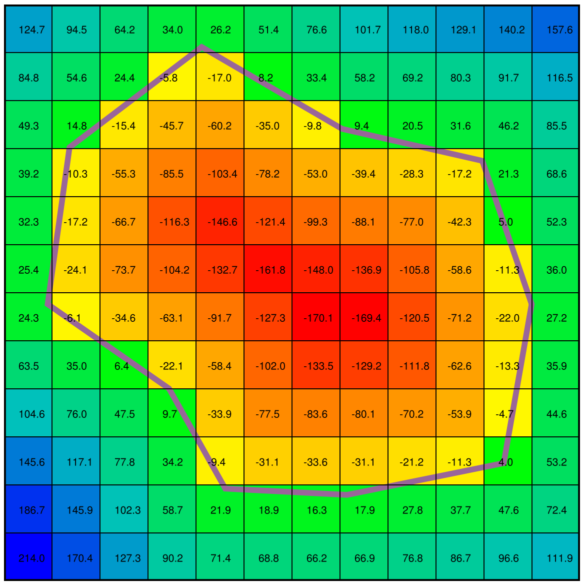
$$\Delta\mathbf{x} = \alpha \sum_i e_i, \quad \text{and} \quad \omega = \beta \sum_i \big((\mathbf{f}'_i - \mathbf{x}) \times \mathbf{e}_i\big)$$

where α and β represent step sizes for rotation and translation, respectively.

3.3 Distance query

In our framework, computing the distance between LittleDog and the terrain for a particular robot configuration is a very frequent operation. Because of their fast runtime and the ability to extract gradient information, we have found that signed distance fields are an ideal representation for distance query and collision checking computation. A signed distance field $d(\mathbf{x})$ stores the distance from any point $\mathbf{x} \in \mathbb{R}^3$ to the boundary of the nearest terrain obstacle. Values are negative inside obstacles, positive outside obstacles, and zero at the boundary.

In our software, the signed distance field is built when the software first encounters a new configuration of terrain. We first discretize the world into a uniform lattice (typically with 5 mm resolution). Next, we scan-convert the triangle-mesh representation of the environment into a boolean-valued voxel map – any voxels which intersect a triangle are assigned value 0; other voxels are assigned value 1. Then we compute the Euclidean Distance Transform (EDT) for the voxel map. Computing the EDT is surprisingly efficient: for a lattice of K samples, computation takes time $O(K)$, as shown by Felzenszwalb and Huttenlocher (2004). The EDT yields an unsigned distance field, which fails to distinguish between the inside and outside of obstacles. To augment the distance field with sign information, we use robust inside-outside testing in each voxel to negate the values which lie inside of the terrain triangle meshes. Converting the mesh to binary voxels results in some degree of aliasing in the distance field, especially near the boundary of the terrain. To mitigate this, for all voxels within one or two grid cells of the terrain, we compute the exact distance



124.7	94.5	64.2	34.0	26.2	51.4	76.6	101.7	118.0	129.1	140.2	157.6
84.8	54.6	24.4	-5.8	-17.0	8.2	33.4	58.2	69.2	80.3	91.7	116.5
49.3	14.8	-15.4	-45.7	-60.2	-35.0	-9.8	9.4	20.5	31.6	46.2	85.5
39.2	-10.3	-55.3	-85.5	-103.4	-78.2	-53.0	-39.4	-28.3	-17.2	21.3	68.6
32.3	-17.2	-66.7	-116.3	-146.6	-121.4	-99.3	-88.1	-77.0	-42.3	5.0	52.3
25.4	-24.1	-73.7	-104.2	-132.7	-161.8	-148.0	-136.9	-105.8	-58.6	-11.3	36.0
24.3	-6.1	-34.6	-63.1	-91.7	-127.3	-170.1	-169.4	-120.5	-71.2	-22.0	27.2
63.5	35.0	6.4	-22.1	-58.4	-102.0	-133.5	-129.2	-111.8	-62.6	-13.3	35.9
104.6	76.0	47.5	9.7	-33.9	-77.5	-83.6	-80.1	-70.2	-53.9	-4.7	44.6
145.6	117.1	77.8	34.2	-9.4	-31.1	-33.6	-31.1	-21.2	-11.3	4.0	53.2
186.7	145.9	102.3	58.7	21.9	18.9	16.3	17.9	27.8	37.7	47.6	72.4
214.0	170.4	127.3	90.2	71.4	68.8	66.2	66.9	76.8	86.7	96.6	111.9

Figure 3: Signed distance field representation of a 2D polygonal obstacle. Space is discretized into cells. Cells storing a negative distance are inside the obstacle; cells with a positive distance are outside. Representing the 3D terrain works in exactly the same manner.

to the nearest triangle (this distance can be computed and stored during the original voxel scan-conversion step). Computing the signed distance field for a 2 m $\times$ 0.6 m environment takes between 2 and 15 seconds, depending on the complexity of the triangle meshes involved.

To perform a distance query between the robot and the signed distance field, we use a simplified representation of the geometry of the robot. We note that the minimum distance between a sphere and any point on the terrain can be computed by taking the value of the signed distance field at the center of the sphere, and subtracting off its radius. Similarly, distance query for a capsule, or line-segment-swept sphere, can be accomplished by sampling along the line segment, taking the minimum, and subtracting off the capsule radius. Our simplified representation of the robot, therefore, consists of a collection of spheres and capsules. Where query points fall between exact cell boundaries, we use mult-linear interpolation to smooth the output of the distance field. Overall, this gives us a collision checking framework which is constant time with respect to the obstacle geometry, and which is linear time in both the number of spheres and the length of line segments in the robot collision model.

Although there is some degree of error between our model and the true robot geometry, the radii of the collision model can be selected such that the model is truly conservative, and only reports collision-free when the robot is collision free. In practice, light or glancing collisions between the robot and the terrain are acceptable (especially of course at the feet) as long as the robot does not attempt to drive *through* obstacles.

For any point $\mathbf{x}$, the signed distance field not only provides $d(\mathbf{x})$, the distance to the nearest obstacle boundary, but $\nabla d(\mathbf{x})$ the corresponding gradient, which tells which direction to move $\mathbf{x}$ in to most quickly increase the distance from obstacles. As shown in the next section, this gradient information is quite useful for resolving collisions during planning. Additionally, computing the gradient of the signed distance field using finite difference is both straightforward and fast.

3.4 Obstacle avoidance

Imagine that the current candidate robot configuration is found to be in collision with the environment at a point $\mathbf{p}$, and that the best direction to move the colliding point in order to resolve the collision is given by

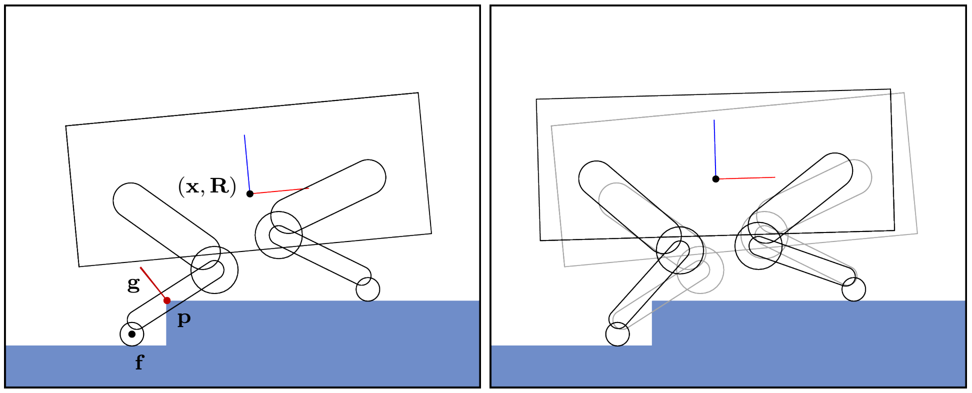


Figure 4: *Left:* during planning, a collision is detected at point $\mathbf{p}$ along the shin. How should the pose of the trunk $(\mathbf{x}, \mathbf{R})$ change to move point $\mathbf{p}$ along the gradient $\mathbf{g}$ of the terrain? *Right:* the obstacle avoidance routine translates and rotates the trunk to correct the shin collision.

the vector $\mathbf{g}$ (as shown in figure 4). How should the robot configuration be modified in order to cause the correct motion of the reference point, while still ensuring that the foot positions remain constrained?

The answer to the question depends upon which link of the robot lies in collision. For collisions with the trunk, a "virtual force" approach similar to the one outlined in the previous section is sufficient to resolve collisions. However, for collisions with hips, knees, or shins, the solution is not quite as straightforward. To accomplish this, we will solve for the two Jacobian matrices

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

which indicate how point $\mathbf{p}$ on the robot moves as $\mathbf{x}$ and $\mathbf{R}$ are changed, given that the foot position will be maintained via IK.

We begin by describing how $\mathbf{p}$ is computed as a function of $\mathbf{x}$, $\mathbf{R}$, and $\mathbf{f}_W$. Let $\mathbf{p}_B(\mathbf{q})$ specify the body-frame position of the reference point given the joint angles $\mathbf{q}$. Recall that $\mathbf{q} = \mathrm{IK}_B(\mathbf{f}_B)$ will be the IK solution of joint angles given a body-frame foot position $\mathbf{f}_B$. Also recall that the body-frame foot position can be computed from world-frame by $\mathbf{f}_B = \mathbf{R}^T(\mathbf{f}_W - \mathbf{x})$. Then expressed in world coordinates, we have

$$\mathbf{p} = \mathbf{R}\,\mathbf{p}_B\Big(\mathrm{IK}_B\big(\mathbf{R}^T(\mathbf{f}_W - \mathbf{x})\big)\Big) + \mathbf{x}$$

First let us examine the translational Jacobian. From the chain rule,

$$\left[\frac{d\mathbf{p}}{d\mathbf{x}}\right] = \mathbf{I} + \mathbf{R}\left[\frac{d\mathbf{p}_B}{d\mathbf{q}}\right]\left[\frac{d\mathbf{q}}{d\mathbf{f}_B}\right]\left[\frac{d\mathbf{f}_B}{d\mathbf{x}}\right]$$

Now we derive the three Jacobian matrices on the right hand side of the above equation. The first one is $\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})$, the Jacobian of the reference point with respect to joint angles. This is the "usual" kinematic Jacobian defined for the reference point. Note that if the reference point is anchored to the hip link, this matrix will be rank-deficient because changing the knee angle has no effect on the body-frame position of the reference point. The second matrix is the rate of change in joint angles with respect to desired foot body position. This is $\mathbf{J}_{\mathbf{f}_B}(\mathbf{q})^{-1}$, the inverse of the familiar foot Jacobian (if such an inverse exists). Finally, by inspection, the last matrix in the equation is simply $-\mathbf{R}^T$, yielding the overall equation

$$\mathbf{J}_{\mathbf{p}}(\mathbf{x}) = \mathbf{I} - \mathbf{R}\,\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})\,\mathbf{J}_{\mathbf{f}_B}(\mathbf{q})^{-1}\,\mathbf{R}^T$$

The Jacobian for the rotational component is derived as:

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

Once again, the chain rule yields the same two kinematic Jacobian matrices produced for the translational derivatives above. The last chain rule term resolves to

$$\left[\frac{d\mathbf{f}_B}{\omega}\right] = \mathbf{R}^T\ [\mathbf{f}_W - \mathbf{x}]_\times$$

The skew-symmetric matrix $[\mathbf{a}]_\times$ obeys the identity

$$[\mathbf{R}\,\mathbf{a}]_\times = \mathbf{R}[\mathbf{a}]_\times\mathbf{R}^T$$

for any rotation matrix $\mathbf{R}$ and vector $\mathbf{a}$. Therefore, the expression above can be further simplified to read

$$\mathbf{J}_\mathbf{p}(\mathbf{R}) = \mathbf{R}\left(\mathbf{J}_{\mathbf{p}_B}(\mathbf{q})\,\mathbf{J}_{\mathbf{f}_B}(\mathbf{q})^{-1}\ [\mathbf{f}_B]_\times - [\mathbf{p}_B]_\times\right)\mathbf{R}^T$$

Finally, to move the reference point $\mathbf{p}$ in the direction of $\mathbf{g}$, we set

$$\Delta\mathbf{x} = \alpha\,\mathbf{J}_\mathbf{p}(\mathbf{x})^T\mathbf{g}, \quad \text{and} \quad \omega = \beta\,\mathbf{J}_\mathbf{p}(\mathbf{R})^T\mathbf{g}$$

Once again, α and β are step sizes that specify the relative magnitude of translational and rotational terms.

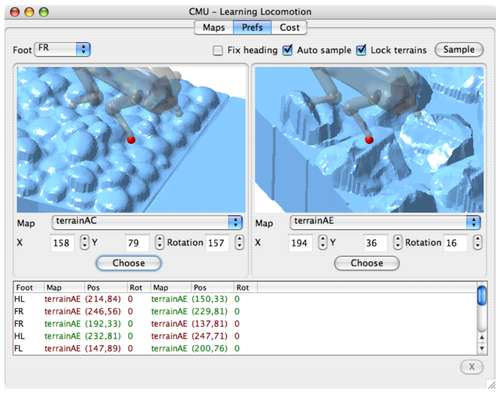
4 Footstep Planning

Discrete search, characterized by algorithms such as A*, is a powerful way to solve problems in motion planning; however, as the underlying dimension of the problem increases, runtime increases exponentially. Unfortunately, legged robots typically have many degrees of freedom (DOF). Footstep planning, as formalized by Chestnutt et al. (2005), mitigates the curse of dimensionality by combining a low-dimensional discrete search over the space of possible footstep locations with a policy or controller that coordinates full-body motion in order to follow a particular footstep path. Because the space of possible footstep locations is much smaller than the space of all full-body motions, the search problem becomes far more tractable.

One major difficulty with this hierarchical approach is the loose coupling between the footstep planner and the underlying policy that executes footsteps. If the planner is too conservative in estimating traversability, the system will underperform; too aggressive, and execution will fail. Our planning software features a novel "certificate" concept which is aimed at ensuring agreement between the planner's traversability estimates and the capabilities of the underlying footstep controller. Other highlights of our footstep planning approach are a terrain cost map learned from expert preferences, and an efficient anytime planning scheme that allows real-time re-planning.

4.1 Footstep cost and heuristic

Recall that the goal of discrete search algorithms such as A* is to find the minimum-cost path from a particular start state to a goal state. For the purposes of a footstep planner, a state is the 2D (x, y) location for each foot; a goal state is one that places the centroid of all the foot locations within a predetermined radius of a specified goal point on the terrain. Both states and actions are discretized, with a finite number of



Foot: FR
[ ] Fix heading [✓] Auto sample [✓] Lock terrains

Map: terrainAC
X: 158 Y: 79 Rotation: 157

Map: terrainAE
X: 194 Y: 36 Rotation: 16

Foot	Map	Pos	Rot	Map	Pos	Rot
HL	terrainAE	(214,84)	0	terrainAE	(150,33)	0
FR	terrainAE	(246,56)	0	terrainAE	(229,81)	0
FR	terrainAE	(192,33)	0	terrainAE	(137,81)	0
HL	terrainAE	(232,81)	0	terrainAE	(247,71)	0
FL	terrainAE	(147,89)	0	terrainAE	(200,76)	0

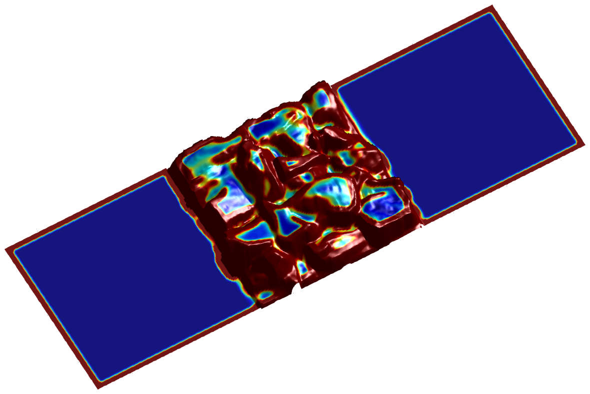


Figure 5: *Left:* Tool for terrain cost learning. An expert expresses preferences over pairs of terrain samples, and the system learns a cost function based on the underlying ranking problem. *Right:* Learned terrain cost function applied to rocky terrain. Blue areas are low cost, red areas are high cost. The system is able to distinguish between flat ground (low cost), shallow indendations (medium cost), dome shapes, slopes, and deep crevices (high cost).

possible actions defined relative to the current state.[1] In our framework, the cost of an action is determined by the sum of two factors: *terrain cost*, which indicates the relative traversability of the terrain under the footstep target, and *pose cost*, which estimates the quality of the various full-body poses that can be achieved in the low-dimensional state.

To reduce planning times, A* and its variants use a heuristic cost-to-go function $h(s)$ that estimates the remaining cost to get to the goal from a particular state s. If $h(s)$ is strictly less than the true cost-to-go, then the heuristic is said to be *admissible*, and A* returns a path guaranteed to be optimal. The running time of A* strongly depends upon the fidelity of the heuristic—a heuristic that approximates the true cost-to-go well should perform far better than an uninformed heuristic.

If optimality is not paramount, it may be desirable to use an inadmissible heuristic known to more closely approximate the true cost-to-go than an alternative, admissible heuristic. This approach can greatly reduce planning times at the expense of incurring some degree of suboptimality, bounded by the disagreement between $h(s)$ and the true cost-to-go (Pearl, 1984).

A typical approach to building a heuristic function for a footstep planner starts by collapsing the robot state down to a 2D point such as the centroid of all foot locations, and computing the Euclidean distance from that point to the goal. Unfortunately, due to kinematic restrictions, LittleDog is not particularly adept at moving sideways or turning in place without taking many steps. Therefore, a heuristic based on Euclidean distance severely underestimates the cost-to-go in certain configurations (such as when the goal lies directly off to the side of the robot).

Instead of collapsing the state to a 2D point, we convert it to a position and orientation (x, y, θ) triple. We then model the robot as a Dubins-style (forward-only) car with a fixed turning radius (Dubins., 1957). To reach the goal along a minimum-distance path, the car must first turn along an arc-shaped segment to orient towards the goal, followed by a straight-line segment to the goal. Using the Dubins heuristic prevents the planner from uselessly allocating search nodes that require large amounts of turning or side-stepping despite their relative proximity to the goal. In a typical trial on rough terrain, using the Dubins heuristic as opposed to a naïve Euclidean heuristic can result in over a twofold speedup in planning time.

4.2 Terrain cost

Terrain cost encodes the relative desirability of a particular foothold, weighing the relative merits of features such as slope, convexity/concavity, and bumpiness. It is easy to come up with broad statements relating terrain shape to cost: steep slopes are bad, dome-shaped features are bad because feet may slip off, small indentations are good because they physically register feet, large indentations are bad because shins can get stuck in them, etc. However, as with many planning and optimization problems, coming up with a precise weighting of these features can be quite difficult in practice—if bumps are worse than slopes, then by how much? And how do both compare to chasms?

Instead of burdening the system designer with the task of exactly specifying a cost function, the field of Inverse Optimal Control (IOC) seeks to learn a cost function to imitate the preferences or behavior of an expert. The first phase of the LittleDog project coincided with our development of the first general inverse optimal control technique which we call Maximum Margin Planning (MMP) (Ratliff et al., 2006). The first real-world implementation of MMP was done on the LittleDog platform. The Learning for Locomotion project inspired continued research within this area, including the development of the nonlinear variants on MMP such as MMPBoost (Ratliff et al., 2007c), leading to a generalized family of nonlinear MMP algorthms known as LEArning to seaRCH (LEARCH) (Ratliff et al., 2009a), as well as a novel form of structured prediction known as Maximum Margin Structured Regression (MMSR) (Ratliff et al., 2007a). In section 8, we describe a variety of extensions to the basic MMP algorithm in order to extend the approach of learning footstep costs for LittleDog.

MMP and LEARCH present rigorous solutions to general IOC problems with strong regret and generalization bounds; however, they rely upon a full sequence of optimal controls to maintain convexity. An expert must demonstrate not only the optimal footstep to take from a given robot state, but in fact the full sequence of optimal footsteps from the inital state to the goal. To further alleviate this burdern, inspired by the spirit of MMP, we explored an alternate cost preference learning methodology that can be understood as a degenerate IOC formulation where the user only specifies relative preferences on single footsteps. This simple IOC formulation was adopted for use in the final Learning Locomotion trials.

In our system, the terrain cost function emerges implicitly from a set of preferences given by a knowledgeable expert (Zucker, 2009). The system presents the expert with a succession of pairs of terrain samples. For each pair, the expert indicates whether he prefers one sample over the other. Then, the system learns a cost function that matches the preferences expressed by the expert (see figure 5).

Our goal is to produce a utility function $u(f)$ over terrain features f such that if the expert prefers f^+ over f^-, then the function makes the preferred feature appear better by a margin $m > 0$:

$$u(f^+) > u(f^-) + m$$

This is exactly the *support vector ranking* problem (Herbrich et al., 1999). Of course, the constraint written in the above equation can not always be met in the presence of noisy or inconsistent data; hence, analogous to support vector machines in the case of classification, we introduce slack variables that transform the hard constraint into a soft one. We solve the underlying ranking problem with an online subgradient method similar to the one described by Kivinen et al. (2004).

We use as terrain features the coefficients obtained by a local quadratic regression of the proposed foothold: that is, the coefficients obtained by the least-squares fit of the surface

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

for multiple window sizes at increasing radii from 1 cm to 6 cm. We also include derived quantities such as the eigenvalues of the second-order coefficient matrix, and the magnitude of the first-order coefficient vector.

[1]We use a local search scheme similar to the one described by Chestnutt (2007) to improve output and avoid artifacts from discretizing actions.

The utility function is defined to be a simple linear combination of all features encoded by a weight vector w, with $u(f) = w^T f$. It gets transformed into a cost function $c(f)$ through exponentiation:

$$c(f) = e^{-u(f)}$$

This ensures that cost is everywhere positive, and it also changes the additive margin m into a multiplicative margin. We believe a multiplicative margin is more natural for this setting since we often setting since we often interpret costs in terms of competitive ratios: how much would we go out of our way (an inflation factor further distance) to avoid this spot. We note that since flat ground produces the zero feature vector, footsteps on flat ground have unit cost.

Our system supports learning separate cost functions for front and hind feet, and it also is capable of extracting features in an oriented fashion, considering the overall heading of the robot as it is computed for the Dubins heuristic.

The process of learning terrain cost is amenable to multiple iterations. After learning a cost function and testing with the robot, the expert can define additional preferences over problem spots. Learning can then be re-run with the original set of preferences, augmented by the new data. Our final cost function was based on just over 200 preferences, the vast majority of which were collected in the initial preference-gathering session over the course of about 2-3 hours.

4.3 Pose cost and certificates

Terrain cost considers the shape of the terrain as it relates to a single foot. Pose cost, however, considers the overall effects of a full set of footstep locations. To ensure a stable support polygon, the planner examines the triangle formed by the three supporting feet for any given footstep. If the triangle has an incircle of less than a preferred radius, the pose cost is increased. This ensures that the robot has a sufficiently large base of support whenever possible.

Computing the remainder of the pose cost consists of searching for a “pose certificate”: a full specification of all 18 DOF of the LittleDog robot that leaves the feet at the positions given by the target state, leaves the body’s center of mass (CoM) above the support polygon, and that is free of collisions with the terrain (except of course at the feet). This is an attempt to ensure good agreement between the footstep planner and footstep execution by preventing the footstep planner from permitting footsteps that are known to be impossible to execute in practice. The pose cost is increased if the certificate puts the robot near kinematic singularities or collisions. If no valid pose certificate can be found, then pose cost becomes infinite, and the action under consideration is disallowed.

It is not uncommon to use a hierarchical planning approach (such as footstep planning) to approximate the solution to a high-dimensional planning problem by running a high-level, abstract planner in a lower dimension. To the best of our knowledge, the pose certificate system is a novel method to “lift up” the intermediate results of an abstract planner into the full problem space while the abstract plan is being constructed. Validating the work of the abstract planner before it makes a hard commitment to a particular plan makes the concrete realization of the output in terms of full-body trajectories far more likely to be feasible.

4.4 Pose certificate search

Searching for a pose certificate is a local, gradient-based method that accumulates a number of translational and angular gradient steps to the robot’s trunk pose over successive iterations. The process begins by placing the robot’s trunk in a heuristically determined position and orientation, given the current foot locations.

Denote the position of the robot’s trunk (determined as the center point of all the hip joints) to be vector $\mathbf{x}$, and it’s orientation to be the rotation matrix $\mathbf{R}$. At the beginning of each iteration, we initialize the translational and rotational gradient terms $\Delta\mathbf{x}, \omega$ to be zero. Next, inverse kinematics (IK) are used to find the joint angles for each leg to place the feet in the correct location. If no valid IK solution can be found for some leg, we add to the current gradient term the reachability gradient described in section 3.2.

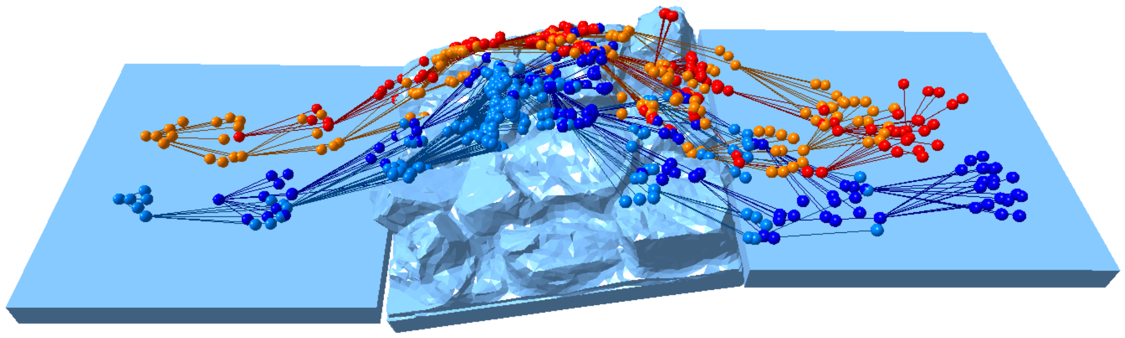


Figure 6: Visualization of ARA* search for footstep planning. Each state visited during planning consists of a set of four footstep locations. The set of all states visited form a search tree. In the picture above, the 8-dimensional search tree is projected down to 2D so that the parent/child relationships between individual footsteps are made visible. The tree shown here has over 700 states; not shown are the many states discarded during search because of collisions, kinematic infeasability, or insufficiently large support triangles.

Terrain	States	Initial	Ratio
Large rocks (Figs 1,5)	23,973	4.40s	1.67
Round rocks (Fig 5)	33,063	1.24s	2.48
Sloped rocks (Figs 1,6)	46,768	1.23s	1.35
Logs (Fig 1)	41,028	1.09s	1.42
Gap	28,232	1.88s	1.58

Table 1: Representative runs showing effect of varying terrain on footstep planner performance. At the start of each trial, the anytime planner is run for 45 seconds. *Terrain* indicates the terrain type *States* indicates the total number of candidate states explored by ARA*, *Initial* is the time it takes ARA* to return the first feasible plan from start to goal, and *Ratio* is the amount of improvement in total cost between the initial and final plans.

Next, the 2D distance between the robot's CoM and the support triangle is evaluated. If the distance is below a margin, a horizontal increment is added to Δx which nudges the body back into the triangle.

Finally, the minimum distance between the robot and the terrain is computed. If it is less than a threshold, Δx and ω are modified in order to move the point at which the minimum distance is attained in the direction of the terrain gradient, as described in section 3.4.

Finally, Δx and ω are applied to obtain a new position and orientation x and R. The process is iterated until no displacements are necessary, or for a preset number of iterations (25 in our system).

4.5 Planning in real-time

One of the most important goals for our software is the ability to plan in real-time. Although the Learning Locomotion evaluation trials allow up to 90 seconds of planning time before the start of a trial, execution errors during a trial (when the clock is running) require fast reaction. We set a target of resuming walking within 2 seconds of detecting and recovering from an execution error.

Both the state space and branching factor for our footstep planner are quite large: in a typical scenario, each foot can be placed anywhere on a 125×375 grid, yielding a state space with over 10^{18} states, and branching factors are between 12 and 25 actions. With an admissible heuristic in such an environment, 90 seconds is nowhere near enough time to compute an optimal plan, let alone the 2 seconds we allocate for re-planning after errors.

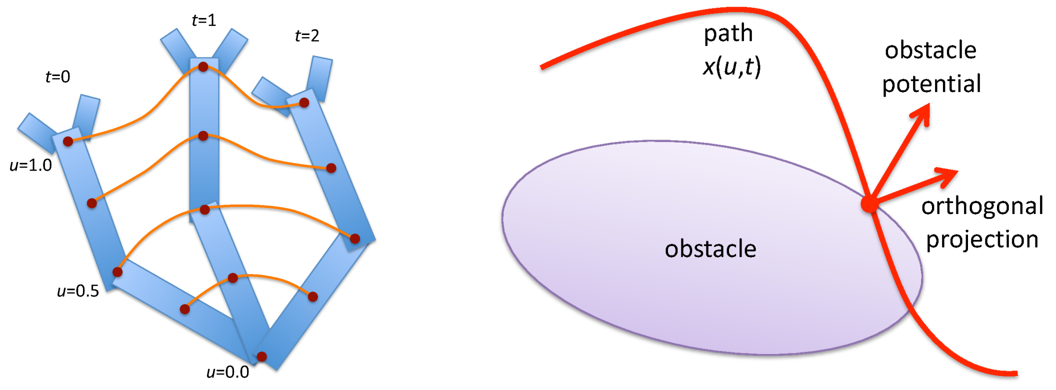


Figure 7: Trajectory optimization for a robot can be considered as minimizing a potential over a set of workspace paths which are jointly constrained by the robot kinematics. *Left:* CHOMP seeks to minimize the obstacle potential $c(x)$, where $x(u, t)$ is a workspace position indexed by time t and robot body position u. *Right:* the gradient update rule for CHOMP takes an obstacle potential acting along the obstacle normal, and projects it perpendicular to the tangent of the path through the workspace, thus ensuring that gradient updates do not merely rearrange the timing of points along the path.

Therefore, we chose to implement our planner as an anytime planner using the ARA* algorithm, which works by iteratively reducing the inflation of the A* heuristic in order to produce plans which are successively closer to optimal (Likhachev et al., 2004). ARA* efficiently re-uses vast amounts of computation between successive searches, examining a minimal set of states each time the heuristic inflation is lowered. Using ARA* allows us to make good use of the full initial planning period to produce high-quality plans, while still producing usable plans quickly after an execution error. See table 1 for a summary of ARA* planning performance.

Of course, there are many other dynamic replanning schemes for heuristic search: D*, for example, is a variant of A* that is specialized for dynamic replanning (Stentz, 1995). However, D* and related algorithms such as D*-Lite and AD* (Koenig and Likhachev, 2002; Likhachev et al., 2005) are not applicable to our problem because they search backwards from a single goal state to the start state, and our footstep planner has a very large set of possible goal states.

Precomputation is also key to our fast planning performance. At the start of a set of trials over a particular terrain, our software computes the signed distance field representation of the terrain, as well as the feature vectors for evaluating terrain cost. Even though the pose certificate computation may invoke as many as 25 full 3D collision checks against the terrain, we can still evaluate tens of thousands of candidate actions, typically 500-1,100 per second on commodity hardware.

5 Footstep Trajectory Optimization

Once the footstep planner produces a sequence of footsteps and pose certificates, it is passed along to a module which generates a trajectory for each footstep. These trajectories are optimized before being executed on the robot in order to ensure they are smooth, dynamically stable, and collision-free. In the remainder of this section we describe CHOMP, a gradient-based optimization algorithm, review how initial footstep trajectories are generated, and detail how CHOMP is used to optimize footstep trajectories.

5.1 The CHOMP algorithm

For footstep trajectory optimization, we use the Covariant Hamiltonian Optimization and Motion Planning (CHOMP) algorithm (Ratliff et al., 2009b). CHOMP is a gradient-based optimizer for creating smooth, collision-free robot trajectories. Although CHOMP extents naturally to become a probabilistically complete planner using the Hamiltonian Monte Carlo algorithm, we use only the deterministic optimization aspects of CHOMP, which we briefly summarize here.

CHOMP represents a trajectory ξ as a sequence of n robot configurations $q_t \in \mathbb{R}^m$ sampled at regular time intervals $t \in \{1, \ldots, n\}$, with endpoints q_0 and q_{n+1} fixed. The objective function $f(\xi)$ that CHOMP minimizes is the sum of two terms: a smoothness objective $f_{smooth}(\xi)$, and a workspace potential objective $f_{obs}(\xi)$. The efficiency and utility of CHOMP stem from two key geometric insights. First, a trajectory for a robot can be viewed as a collection of paths through the workspace, jointly constrained by the robot's kinematics (see figure 7). Second, using a covariant, or *natural* gradient ensures that convergence is fast independent of the number of samples used to represent the trajectory.

Now we define the two terms of the CHOMP objective function. The smoothness term is simply expressed as a sum of squared time-derivatives (e.g. velocity or acceleration) over the trajectory. Since the trajectory is represented as a sequence of samples, such differentiation can be accomplished via a straightforward finite differencing operation. If the vector ξ is a simple concatenation of the robot configurations q_t, then for any derivative, the differencing operation can always be written as $K\,\xi + e$ where K is a suitable differencing matrix with band-diagonal structure, and e is a vector that accounts for the endpoints of the trajectory. The sum of squared derivatives can therefore be expressed as

$$f_{smooth}(\xi) = \xi^T A\,\xi + \xi^T b + c$$

with positive definite $mn \times mn$ matrix $A = K^T K$, the vector $b = 2K^T e$, and scalar $c = e^T e$.

As shown in figure 7, we can consider the overall trajectory to be a collection of paths through the workspace jointly constrained by the robot kinematics. Let u index points on the robot's body. Then denote by $x(u,t)$ the function that gives the position of point u on the robot's body at time t. Now, given a workspace potential function $c(x)$ imposed by obstacles or other workspace constraints, we define the workspace potential to be the discrete analog of the functional

$$f_{obs}(\xi) = \int_u \int_t c\big(x(u,t)\big) \left\| \frac{d}{dt} x(u,t) \right\| dt\, du$$

The arc-length parameterization of f_{obs} has an important effect: the functional gradient of this objective projects the workspace potentials orthogonal to the tangent of the path $x(u,t)$ (also illustrated in figure 7). This ensures that the effect of the gradient is to always adjust the overall path, instead of merely redistributing points along it.

To run gradient descent with CHOMP, we use the update rule

$$\xi \leftarrow \xi - \alpha A^{-1} \nabla f(\xi)$$

where α is a step size, $\nabla f(\xi)$ is the normal (Euclidean) gradient of the objective function with respect to ξ, and A is the same positive-definite matrix defined above. This is the so-called "natural" gradient because it is invariant to re-parameterization (for instance, changing the number of samples used to represent the trajectory). The practical effect of multiplying the gradient by A^{-1} is to speed convergence by spread the impact of obstacles smoothly along the entire trajectory.

CHOMP confers a number of benefits over other path and trajectory optimization schemes such as elastic strips or shortcut-based methods (Brock and Khatib, 2002; Geraerts and Overmars, 2006). First and foremost, the natural gradient technique gives very fast convergence. CHOMP also considers dynamic quantities such as velocity or acceleration instead of simply minimizing path length. Furthermore, by using signed distance fields (described in section 3.3) to define the workspace potential function, we get smooth gradients even for robot configurations that are in collision with obstacles. Aside from LittleDog, CHOMP has also been used to guide robotic manipulators through cluttered workspaces (Ratliff et al., 2009b; Willow Garage, 2009).

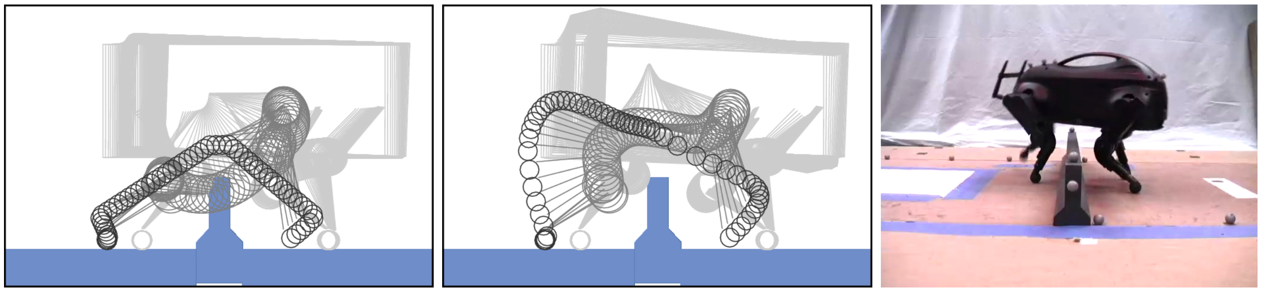


Figure 8: Optimizing a footstep over a barrier with CHOMP. *Left:* Initial trajectory has severe knee collisions. *Center:* After optimization, collisions have been resolved by lifting the leg higher and tilting the body. *Right:* Execution on robot.

5.2 Initial trajectory

CHOMP requires as its input an initial robot trajectory. The input trajectory need not be strictly collision-free, but we make some effort to ensure that it does not lie too close to a bad local minimum. To make sure that the motion of the robot is dynamically stable, we initialize the trajectory for the body to follow a path generated by a ZMP preview controller (Kajita et al., 2003). The zero moment point (ZMP), equivalent to the center of pressure, is the dynamic analogue to the center of mass (CoM) of a stationary object. If the ZMP remains above the support polygon at all times, the robot will support itself stably. Based on a 2D table-cart model, the ZMP preview controller takes as input a desired reference trajectory for the robot zero moment point, and outputs a trajectory for the CoM that matches the ZMP reference trajectory closely and minimizes extraneous motion.

The first step in building footstep trajectories is to construct a linear spline path for the ZMP to follow. The default choice for the ZMP in each footstep is the body position given by the corresponding pose certificate, because it is known to be stable, reachable, and collision-free.[2] The linear spline path is transformed into a time-referenced trajectory by considering the estimated duration for each footstep.

LittleDog's maximum walking speed is largely dictated by the maximum 7.5 rad/s hip joint velocity and 14 rad/s knee joint velocity. Therefore, footstep timing can be well approximated by computing the maximum over all joints of minimum time to move the joint from initial to final position.

Once the footstep durations for the next three footsteps are estimated, the ZMP preview controller generates the body trajectory for the next footstep to be execute. Multiple steps of lookahead are required due to the preview controller's moving preview window. The CoM trajectory output by the 2D ZMP preview controller does not specify body height or orientation, so we independently use splines connecting the certificate poses to fill in this information.[3]

For the swing leg, a spline is constructed to move smoothly from initial foot position, up to a heuristically determined "liftoff" position, over to a "dropoff" position, and down to the footstep target, as shown at the left of figure 8. Once the body and swing leg trajectories have been generated, the supporting leg trajectories are determined by IK, and the full initial trajectory is ready to be sent to the optimization module. Note that although the trajectory satisfies the 2D dynamic stability criteria, it may be infeasible in other respects: there could be kinematic reachability issues, or collisions against the terrain. Trajectory optimization is an opportunity to correct these issues before the footstep is executed.

5.3 Footstep trajectory optimization with CHOMP

Other systems for generating footstep trajectories primarily reason about collisions for the swing foot alone (Kolter and Ng, 2009). For most footsteps, this is sufficient; however, it can sometimes lead to situations where the shin or knee of a supporting leg impacts the terrain as the body moves forward. Considering full-body collisions with CHOMP goes a long way to mitigate these situations.

Trajectory optimization works on a one-footstep window. A footstep consists of an optional "shift" phase, where the robot's center of pressure is moved into the current footstep's polygon of support, and during which all four feet are on the ground, followed by a "step" phase during which one foot is lifted up, moved over its destination, and set back down. Because of this difference, there may acutally be a different number of samples in a given footstep for the body and for the current swing leg, because swing leg angles are constrained by IK during the shift phase. To optimize a footstep trajectory, we run CHOMP as coordinate descent over body and swing leg configurations. First, the body trajectory is optimized with CHOMP, given the current swing leg joint angles. Second, the swing leg joint angles are optimized with CHOMP given the current body poses.

Beyond the smoothness and collision-free objective criteria required for CHOMP, our software also attempts to optimize kinematic reachability and static stability. The reachability and stability gradients are handled analogously to collision; after they are computed as described in sections 3 and 4.4, then projected orthogonal to the current trajectory as in the original CHOMP work.

One unresolved issue in CHOMP as presented to date is the problem of allocating samples. It is difficult to determine *a priori* how many samples it takes to represent a trajectory, because the number of samples increases with the complexity of the trajectory. We chose a somewhat heuristic solution to the problem. For our LittleDog software, trajectories are always sampled at a fixed rate (half the frequency of the main control loop). The CHOMP algorithm proceeds until the trajectory is valid, or for a predetermined number of iterations. If, after CHOMP finishes, the trajectory is in collision with the terrain, or if the leg joint velocity limits are violated, the estimated duration of the footstep is increased, the initial trajectory is re-generated and CHOMP is restarted (up to a maximum number of restarts).

As mentioned in section 5.1, we omit the Hamiltonian Monte Carlo aspect of CHOMP in our LittleDog software because we found it to be largely unnecessary for this class of problems. Additionally, since there is no guarantee that a feasible trajectory exists for a given foot placement action computed by the footstep planner, the additional work performed by a stochastic search could be wasted. In the event that the final trajectory after optimization is not feasible, one possible action to take would be to throw an exception back to the footstep planner; however, in our system we chose instead to optimistically execute these known-infeasible trajectories. In the best case, they will succeed; otherwise, hopefully the execution layer can recover and re-plan after execution errors.

6 Execution and Low-Level Control

The footstep execution module maintains a trajectory buffer that operates in first-in, first-out (FIFO) fashion. The trajectory specifies not only the desired joint angles and velocities over time, but also additional information such as desired body position and orientation, corresponding velocities and accelerations, which feet should be supporting, and estimated ground interaction forces. After the first footstep, trajectory optimization occurs in parallel with execution. As the current step is being executed, the next step gets optimized and appended to the FIFO buffer.

The execution module corrects for errors at a variety of levels. The servos onboard the LittleDog robot support per-joint proportional-derivative (PD) control, as well as force control, which we use to minimize shear forces and to help soften foot impacts on touchdown.

[2] In order to achieve faster walking speeds and minimize the displacement of the trunk, we also test a "shortcut" ZMP position that reduces the overall length of the spline. If the shortcut position results in terrain collisions or other errors, the system falls back to the certificate pose using the restart procedure described later in this section.

[3] Accelerations due to vertical motion and rotation are assumed to be negligible in terms of the ZMP stability criterion.

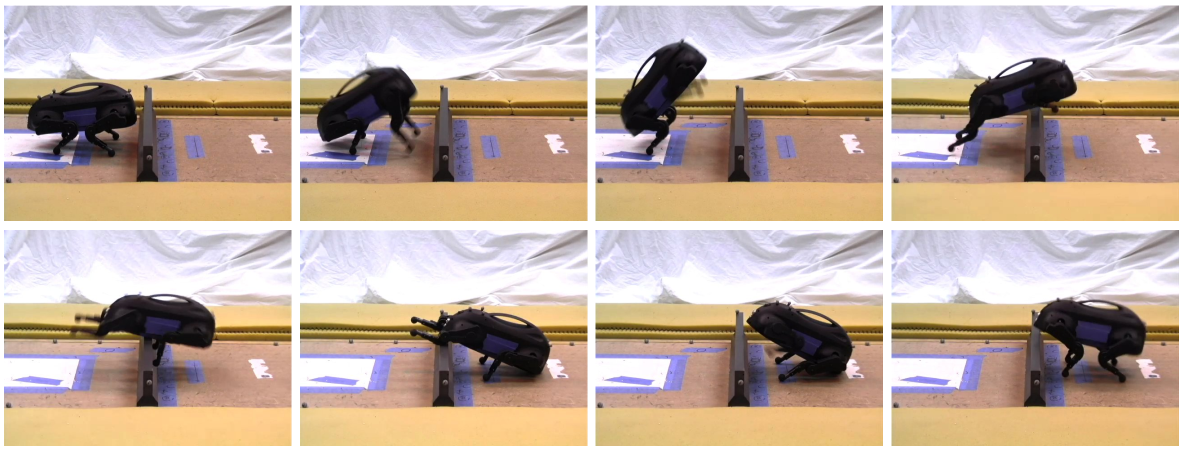


Figure 9: Specialized behavior for barrier crossing. First, the robot crouches in front of the barrier. Rearing up, it lunges forward to place its front shins on the edge of the barrier, and uses its front legs to push itself over. To gain enough clearance to move its hind legs over the barrier, the robot creates a stable tripod with its two front legs and the front of its trunk. Finally, it recovers by placing its hind feet on the ground.

Although commands to the robot are specified in terms of joint angles and velocities, it is important to track the desired trunk position and orientation in order to maintain dynamic stability and to ensure that the robot hits the planned footstep targets. Foot slip and gear backlash both contribute to errors in body pose. To combat this, we run a slow-moving integrator on each supporting leg.

The integrator works as follows: Denote the desired position vector and rotation matrix of the body by $\mathbf{x}$ and $\mathbf{R}$, respectively, and denote the desired foot position for a supporting leg in body-frame Cartesian coordinates by $\mathbf{f}_B$. Denote the foot's desired position in the world frame by $\mathbf{f}_W = \mathbf{R}\,\mathbf{f}_B + \mathbf{x}$. Say the body is observed to be at position $\bar{\mathbf{x}}$ with rotation $\bar{\mathbf{R}}$. We transform the desired world foot position into the observed body frame $\bar{\mathbf{f}}_B = \bar{\mathbf{R}}^T(f - \bar{\mathbf{x}})$. The update rule for the leg integrator Δ to compute the commanded joint angles $\mathbf{q}$ is then given by

$$\begin{aligned} \Delta &\leftarrow \Delta + \gamma\,(\mathbf{f}_B - \bar{\mathbf{f}}_B) \\ \mathbf{q} &\leftarrow \mathrm{IK}_B(\mathbf{f}_B + \Delta) \end{aligned}$$

where γ is an integrator gain, and IK is the function that solves for joint angles corresponding to given body-frame coordinates. This control law acts to counter the body error for each leg, shortening the leg if the body is too high, and lengthening it if the body is too low.

Assuming no further slippage occurs, this control will move the body towards its correct orientation and position. In order to have the feet step into their intended locations, the integrator for each foot is decayed to zero while the foot is in fight. Since the foot is no longer on the ground in this case, the foot is no longer needed to correct the body position and orientation. Assuming the stance feet succeed in correcting the body's position and orientation, the flight foot, with zeroed integration term, will step into its intended location on the terrain (Stolle, 2008).

Execution errors larger than a certain threshold trigger a reflex module which attempts to restore the robot's balance and place all four feet firmly on the ground. The reflex module can be triggered by body position or orientation errors, foot position errors, or foot force errors. After recovery stabilizes the dog, the footstep planner is re-started, and walking resumes soon after.

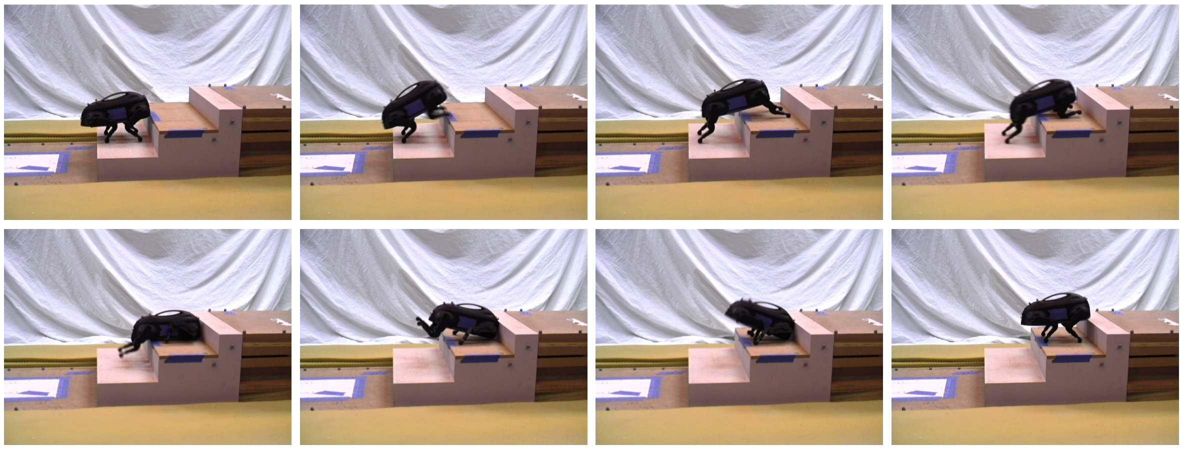


Figure 10: Specialized behavior for climbing tall steps. The robot begins by rearing up and lunging to place its front legs far forward on the next step. It then uses its front elbows to drag its body and hind feet forward all the way onto the step. Finally, it tips up into a tripod using the front of its trunk and front legs to gain enough clearance to bring its hind legs onto the step.

7 Specialized Behaviors

Although the vast majority of the terrain encountered in the Learning Locomotion program can be traversed by our walking gait, in some scenarios we elected to switch to a set of specialized behaviors designed specifcally to exploit geometric features of certain terrains.

We developed two specialized behaviors in particular to help us through the final phase of the Learning Locomotion project. These behaviors were used for climbing large steps and traversing tall barriers with heights of over 12 cm (note that the LittleDog leg is 15 cm when fully extended). They are illustrated in figures 9 and 10. The behaviors consist of a sequence of joint angles which are replayed by the robot once it has walked to an appropriate location on the terrain. In the past, we have used joystick input to create specialized behaviors, but the ones used for the barrier and step terrains were specified as a scripted program.

In the most recent phase of the Learning Locomotion project, specialized behaviors were not integrated into a larger planning structure, and were run only when the system had *a priori* knowledge of the class of terrain (barrier or steps). Despite the lack of a general planning algorithm, these behaviors are actually capable of handling a broad range of obstacle geometry.

7.1 Planning with specialized behaviors

In past learning locomotion phases, we investigated integrating these specialized behaviors into a more general planning scheme. In order to make use of these pre-recorded behaviors on new terrain, we employ a two part algorithm. In the first part, for each step, we find possibly multiple patches of terrain that are similiar to the terrain where the step was recorded. The recorded information is then transfered to the new position and orientation as well as adapted to some changes in the terrain. We also verify using swept volumes of the body that the dog will not collide with the terrain in the new location.

The second part of the algorithm is a hierarchical search, which incorporates the footstep planner and produces plans that are a combination of synthetic steps from the footstep planner as well as precorded steps. For the high level search, we generate a topological graph (see figure 11). The nodes of the graph are the start state and the goal state of the robot, as well as every step in the transferred library. Edges represent walking from the end of the pre-recorded step represented by the start node to the beginning of the pre-recorded step represented by the target node. The cost of every edge is roughly the number of additional steps that have to be taken to traverse the edge. If the foot locations at the end of the source pre-recorded

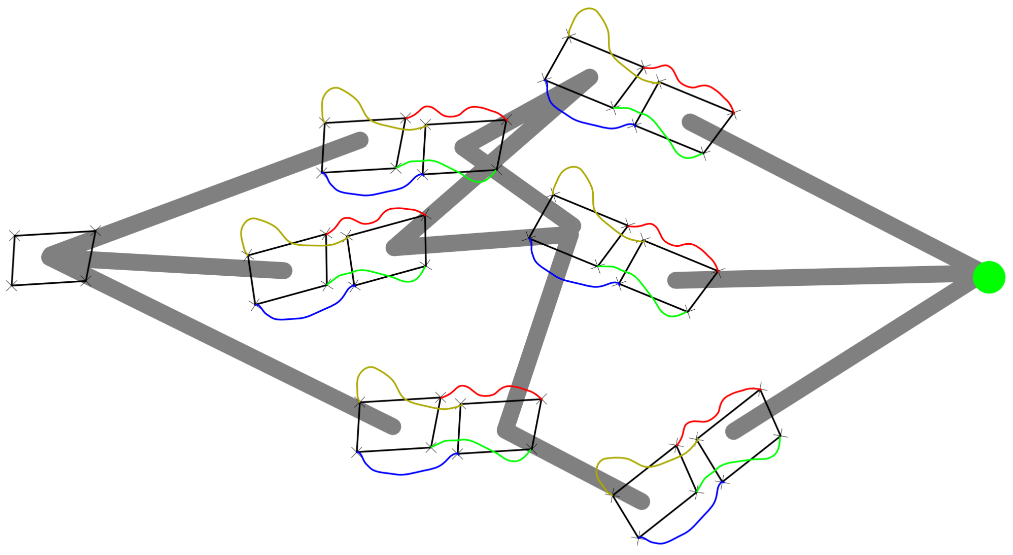


Figure 11: Illustration of topological graph after transfer of library. Each behavior is a node in the topological graph. The connections (gray lines) are made using the foot step planner.

step are close to the foot locations at the beginning of the target pre-recorded step of the edge, no additional steps are necessary. In order to know the number of additional steps, the footstep planner is used at this stage to connect the gaps between steps when we generate the topological graph. Since the steps that are output by the planner are considered risky, we assign a higher cost to planned steps. (If the planner created reliable steps, we could just use the planner to plan straight from the start to the goal.) In order to reduce the complexity of the graph, nodes are only connected to the n-nearest steps based on the sum of Euclidean foot location difference metric. We then use a best-first search through this graph to find a sequence of footstep-planner-generated and pre-recorded steps. This sequence is added to the final library. More details on planning with specialized behaviors can be found in (Stolle, 2008).

8 Additional Imitation Learning

The LittleDog platform has provided an ideal test bed for imitation learning, and has led to the development of numerous novel imitation learning techniques. To motivate the importance of imitation learning for quadrupedal locomotion, we note that to date there still exists a very difficult terrain developed for the project, known as terrain D, that cannot be reliably traversed autonomously by any team. The only robust policy that we were able to develop was one built atop a trajectory created by a human teleoperator. While traversal using existing automated technology has proven impossible, the resulting demonstrated policy was shown to be both repeatable and robust to significant modeling perturbations. This discrepancy illustrates the importance of learning by demonstration within the context of quadrupedal locomotion across rugged terrain. We have already described how IOC algorithms such as MMP and LEARCH can learn how to reproduce expert behavior. In this section, we detail two primary applications inspired the development of these techniques: heuristic learning for footstep planning, and footstep prediction.

8.1 Training a fast heuristic planner to mimic a slow exact planner

As discussed above, at a course resolution, we can view our system as decomposing into two primary subcomponents: a footstep planner, and an execution module. During the footstep planning stage, an optimal sequence of footsteps is constructed that guides the robot from a start configuration to a goal configuration, and during the execution stage, full-body trajectories are generated to take the robot from one footstep to the next along that sequence. Our learning techniques for this project focused primarily on the footstep generation phase. In this section, we assume that the costs processed by the high-dimensional footstep planner are correct, i.e. we assume that the execution stage will be able to find successful and efficient trajectories for the footsteps that result. The primary issue, then, becomes speeding up the slow high-dimensional footstep planner to achieve effective real-time performance for efficient replanning. We address the higher-level question of learning superior costs to mimic human demonstration in the next section.

Footstep planning is a complicated and high-dimensional problem. As a result, footstep planners built atop the A* algorithm, while tractable, tend to be slow. Importantly, however, the performance of A* is highly dependent on the sophistication, accuracy, and efficiency of the heuristic. Heuristics that more accurately predict the planner's cost-to-go from any given state can prune away large portions of the search space and, accordingly, facilitate fast convergence. While the sequence of footsteps is generally high-dimensional, we observed that the problem naturally decomposes into two well-defined subtasks. The most straightforward of these subtasks is the foot-placement, itself. At any given location, the footstep planner must reason about where the foot should go within the local reigon. However, at a higher level, the footstep planner must additionally reason about the most efficient two-dimensional navigational trajectory the robot should follow through the terrain. This second task significantly increases the computational difficulty of the problem. Our approach, therefore, was to extract this two-dimensional navigational behavior into a fast 2D planning algorithm which could then be used as an intelligent and highly accurate heuristic to efficiently guide the footstep planner from the start to the goal.

To generate training examples, we ran the footstep planner across the terrain between randomly selected pairs of start and goal points, and extracted the two-dimensional navigational trajectory by estimating the trajectory of the robot's center of mass. We then trained a fast 2D navigational planner to mimic this behavior using a novel nonlinear variant of MMP we call MMPBoost (Ratliff et al., 2007c). In order to obtain an informative heuristic under the trained planner, then scaled the resulting cost map using linear regression so that the cost-to-go values of the trained 2D planner best matched those of the higher-dimensional footstep planner. Details of this approach can be found in (Ratliff et al., 2007c). When used as a heuristic for a bipedal planner, we attained an average speedup of 120x in the footstep planner. The best speedup seen under the quadrupedal footstep planner, though, was only about 4x. We hypothesize that this difference in speedup can be attributed primarily to the quality of approximation provided by the 2D navigational planner. For bipedal locomotion, where the robot is much more accurately represented as a two-dimensional orientationless point, the approximation proved accurate. On the other hand, under quadrupedal locomotion, the built in action model biased the planner toward footstep sequences that avoided turning in place, roughly introducing a third, rotational, dimension to the navigational component. We hypothesize that improved speedups may be attained for quadrupedal locomotion by using a three-dimensional navigational planner that explicitly accounts for the robot's orientation. The implementation of this heuristic learning technique for the LittleDog platform was used for experimentation only; we opted not to utilize the technology during the trails.

The technique discussed above is what we called *regressed MMP*. It decomposes the problem into two phases. The first phase utilizes the MMP algorithm to learn to mimic the overall navigational behavior embodied by the footstep planner, while the second phase then scales the learned cost function to enable its use as a heuristic. In order to shortcut this two-stage decomposition and more directly train the planner to reproduce the desired cost-to-go values, this project inspired the development of a novel form of structured prediction we call Maximum Margin Structured Regression (MMSR), which we experimentally validated in (Ratliff et al., 2007a). Details of the approach can be found in both that paper and in (Ratliff, 2009). At a high level, though, this approach can be viewed as a natural generalization of Maximum Margin Structured Classification framework to the regression setting using techniques analogous to the well-understood Support Vector Regression algorithm (Smola and Schölkopf, 2003).

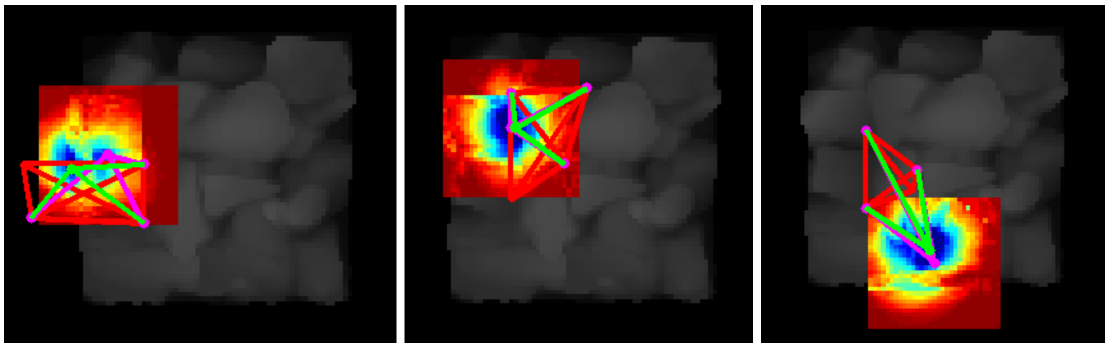


Figure 12: Validation results: Green indicates predicted next footstep, purple indicates desired next footstep, and red indicates current stance. The associated cost function is provided gradating from red at high cost regions to dark blue at low cost regions. The influence of the terrain features on the cost function is most apparent in the left-most prediction whose hypotheses straddle the border between the terrain and flat ground.

8.2 Footstep prediction using teleoperated demonstrations

Heuristic learning, discussed above, concentrated primarily on the problem of speed and implicitly assumed that the planner finds efficiently executable sequences of footsteps. Unfortunately, this assumption rarely holds in practice. The subtleties of trading off features to define a single scalar cost measuring footstep fitness is difficult in real-world systems such as the LittleDog platform. On the other hand, demonstrating desired behavior can be straightforward. Teleoperation provided a means by which we could demonstrate robust and repeatable policies for crossing difficult terrain, and the resulting trajectory across the terrain implicitly encoded important information about the desirability of candidate footstep locations. Using techniques discussed more thoroughly in (Ratliff et al., 2007b) and (Ratliff et al., 2009a; Ratliff, 2009), we trained a footstep prediction algorithm to greedily predict the next footstep as a function of terrain features and course kinematic reachability features given the current four-foot configuration. We demonstrate that the prediction algorithm can efficiently learn to trade off terrain costs with reachability costs, and in the process learn an informed action model for the robot. This work represents one of the first implementations of our novel exponentiated functional gradient algorithm for imitation learning (Ratliff et al., 2009a) which extends the work on MMPBoost disucssed above. Figure 12 depicts the performance of the prediction algorithm on few test examples, and figure 13 shows its performance on a greedy sequential prediction application across both rugged and flat terrain.

We implemented and used this learned cost function on the LittleDog platform throughout the second half of the Phase II trials of the Learning for Locomotion competition. Since the execution module was developed and tested under the footstep planner's action model, we incorporated only the terrain cost component of the footstep prediction policy into the planner by retraining the cost function without the action features. Under the new terrain costs, the footstep planner produced executable footstep sequences that embodied the behavior of the human demonstrator. Accordingly, the resulting policy qualitatively tended to step more frequently in stable local concavities and proved to be much more robust in practice than previous hand-tuned attempts.

9 Conclusions

We have described a software framework for planning and control of a rough terrain quadruped robot. Notable features of our system include a fast, anytime footstep planner guided by a cost map learned from expert preferences, novel use of high-dimensional "pose certificates" to verify a low-dimensional planner,

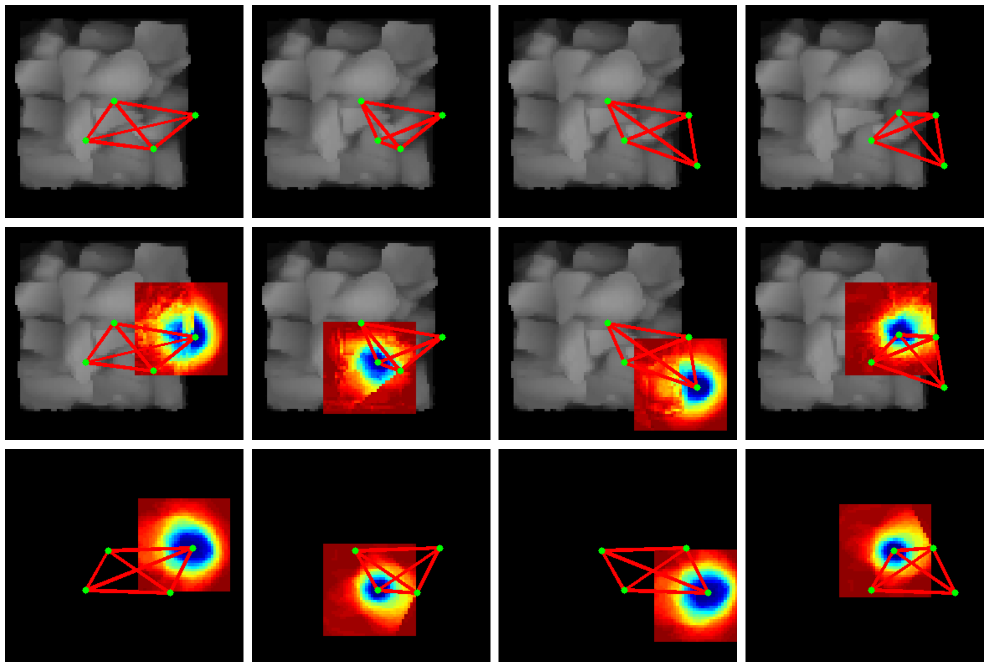


Figure 13: The first two rows show predicted footstep sequences across rough terrain both with and without the corresponding score function. The bottom row demonstrates a predicted sequence for walking across flat ground. Generalization of quadruped footstep placement. The four foot stance was initialized to a configuration off the left edge of the terrain facing from left to right. The images shown demonstrate a sequence of footsteps predicted by the learned greedy planner using a fixed foot ordering. Each prediction starts from result of the previous. The first row shows the footstep predictions alone; the second row overlays the corresponding cost region (the prediction is the minimizer of this cost region). The final row shows footstep predictions made over flat ground along with the corresponding cost region showing explicitly the kinematic feasibility costs that the robot has learned.

and a novel trajectory optimization module that smooths trajectories and resolves potential collisions in real-time.

9.1 Future Work

There remains much work to be done in order to produce a system that could be deployed on an autonomous robot "in the wild". First and foremost, the software here depends on the existence of a good map of the world; obtaining such presents a substantial challenge outside of a laboratory environment.

In addition, we would like to investigate other ways in which learning can be brought to bear on the problem of legged locomotion (and planning in general). The current work is solely focused on imitation learning, with little adaptation online and none that generalizes. Our ultimate goal is to produce a system that improves its own performance over time, avoiding repeated mistakes and exploiting plans that have worked well in past situations.

In the past, we have used stochastic policy gradient methods to create randomized planners that learn the characteristics of a particular problem domain (Zucker et al., 2008). We are interested in producing analogous work of discrete forward search.

In the rare cases where no feasible footstep trajectory can be found, how can the failure get propagated back to the planner to avoid risky behavior? We would like to use machine learning techniques to continuously monitor execution and steer the planner away from problem areas in the future.

Acknowledgments

This research was funded by the DARPA Learning Locomotion project. The authors wish to thank Hanns Tappeiner, Andrew Maas, Elliot Cuzzillo, Eric Whitman, and Alex Grubb for their valuable contributions to the CMU Learning Locomotion effort.

References

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.

M. Buehler, R. Playter, and M. Raibert. Robots step outside. In *Proc. International Symposium on Adaptive Motion in Animals and Machines*, 2005.

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

J. Chestnutt, M. Lau, KM Cheung, J. Kuffner, J. Hodgins, and T. Kanade. Footstep Planning for the Honda ASIMO Humanoid. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2005.

L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79, 1957.

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

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.

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.

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.

M. Kalakrishnan. Personal communication, November 2009.

M. Kalakrishnan, J. Buchli, P. Pastor, and S. Schaal. Learning locomotion over rough terrain using terrain templates. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2009.

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

S. Koenig and M. Likhachev. Dˆ* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

J.Z. Kolter and A.Y. Ng. Learning omnidirectional path following using dimensionality reduction. In *Proc. Robots, Science and Systems*, 2007.

J.Z. Kolter and A.Y. Ng. Task-Space Trajectories via Cubic Spline Optimization. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2009.

J.Z. Kolter, M.P. Rodgers, and A.Y. Ng. A control architecture for quadruped locomotion over rough terrain. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 811–818, 2008.

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

M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271, 2005.

J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.

Marc Raibert. *Legged robots that balance.* MIT Press, 1986.

N. Ratliff. *Learning to Search: Structured Prediction Techniques for Imitation Learning.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2009.

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

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

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

N. Ratliff, D. Bradley, J.A. Bagnell, and J. Chestnutt. Boosting Structured Prediction for Imitation Learning. In B. Scholkopf, J.C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007c. MIT Press.

N. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots, Special Issue on Robot Learning*, 2009a.

N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2009b.

U. Saranli, M. Buehler, and D.E. Koditschek. RHex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616, 2001.

A.J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical report, Statistics and Computing, 2003.

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

M. Stolle. *Finding and Transferring Policies Using Stored Behaviors.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2008.

Willow Garage. CHOMP Motion Planner. Willow Garage Blog, September 2009. URL `http://www.willowgarage.com/blog/2009/09/02/chomp-motion-planner`. Retrieved on November 1, 2009.

M. Zucker. A data-driven approach to high level planning. Technical Report CMU-RI-TR-09-42, The Robotics Institute, Carnegie Mellon University, January 2009.

M. Zucker, J. Kuffner, and J.A. Bagnell. Adaptive workspace biasing for sampling based planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2008.