2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

An Optimization Approach to Rough Terrain Locomotion

Matt Zucker J. Andrew Bagnell Christopher G. Atkeson James Kuffner

The Robotics Institute, Carnegie Mellon University
{mzucker,dbagnell,cga,kuffner}@cs.cmu.edu

***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 deeper 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.**

I. Introduction

Legged locomotion has been a key area of robotics research since nearly the inception of the field [1]; however, robots that can skillfully and deliberatively traverse rough terrain have yet to be fielded 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 [2], [3]. 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.

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.

In this paper, we present our planning and control software for LittleDog. Our software has been used successfully throughout the final phase of Learning Locomotion in order to traverse a broad range of terrains.

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

II. System Overview

Our approach to system design is strongly motivated 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 can be crucial. Other participants in the Learning Locomotion project

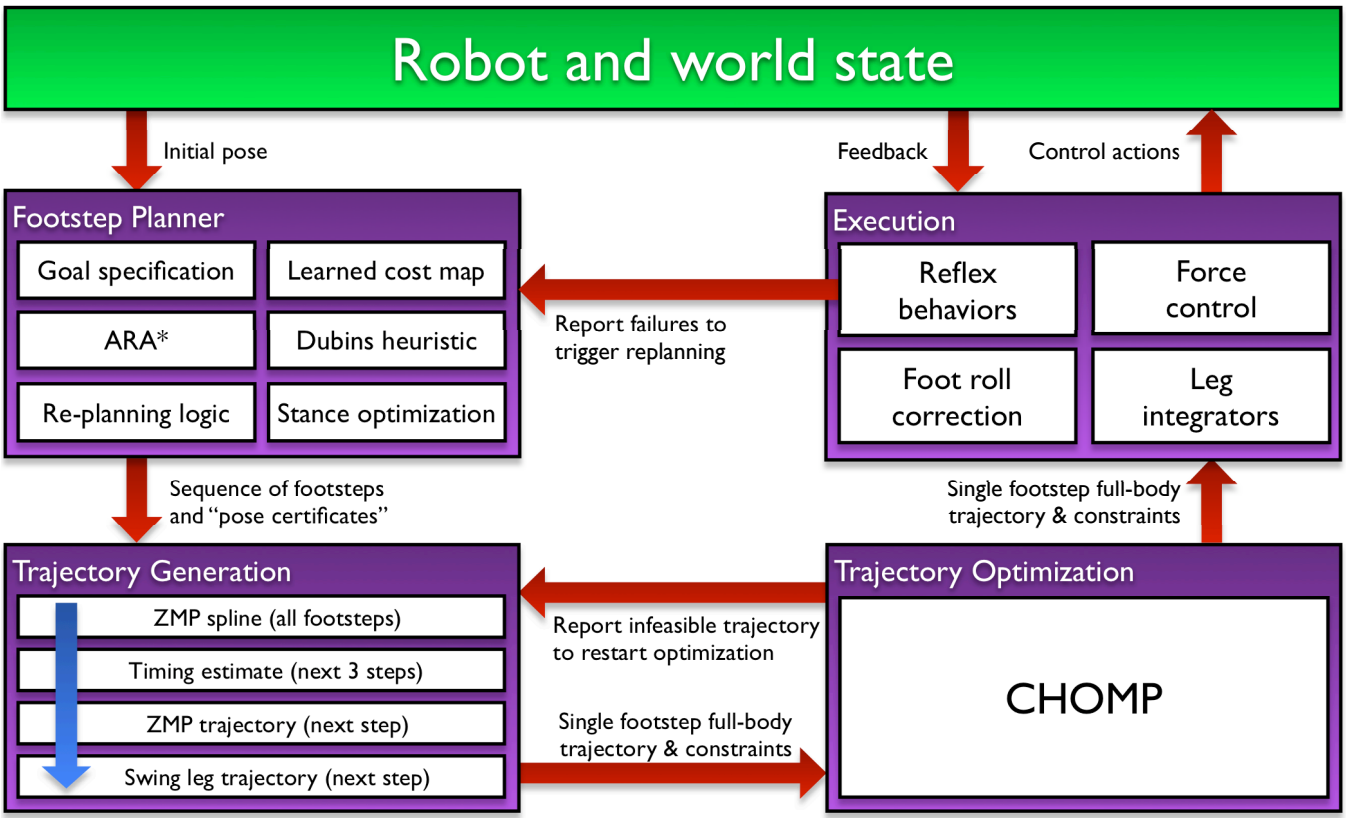

Fig. 2. System block diagram. A hierarchy of planners and optimizers generates complex full-body locomotion behavior.

have chosen to plan a path for the robot's trunk at the lowest level [4]–[6]. 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.

The system presented here features only a subset of a wide variety of techniques developed for this project at the Robotics Institute. An upcoming paper will discuss the entire CMU Learning Locomotion effort in greater detail [**?**].

III. 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 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 [7]. 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.

A. 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 possible actions defined relative to the current state.[1] In

[1]We use a local search scheme similar to the one described in [8] to improve output and avoid artifacts from discretizing actions.

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 [9].

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 (forward-only) car with a fixed turning radius [10]. 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.

B. 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. In previous work, Maximum Margin Planning was used to solve the IOC problem for a LittleDog footstep planner [11], [12]. Here, we present a faster, simpler IOC formulation that is effective in practice.

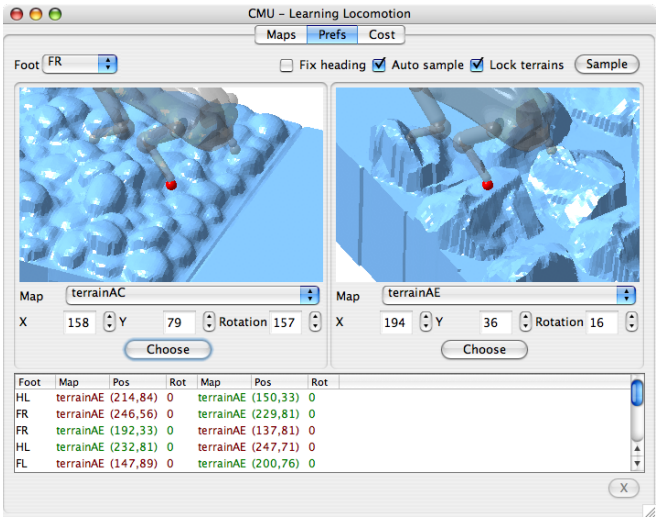

Fig. 3. Tool for terrain cost learning. An expert expresses preferences over pairs of terrain samples, and the system learns a cost function based on the underlying ranking problem.

In our system, the terrain cost function emerges implicitly from a set of preferences given by a knowledgeable expert. 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 3).

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 [13]. 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 [14].

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.

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. 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 material. 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.

C. 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.

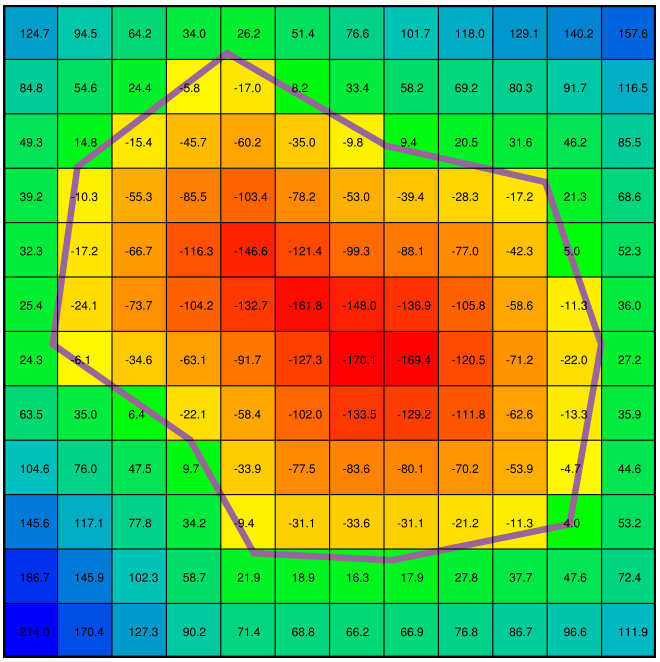

Fig. 4. Signed distance field representation of a 2D polygonal obstacle. Space is discretized into cells. Cells storing a negative distance are inside the obstacle; cells with a positive distance are outside. Representing the 3D terrain works in exactly the same manner.

D. Pose certificate search

Searching for a pose certificate is a local, gradient-based method that accumulates a number of translational and angular displacements to the robot 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 x, and it's orientation to be the rotation matrix R. At the beginning of each iteration, we initialize the translational displacement Δx and rotational displacement ω 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, Δx and ω are adjusted in order to translate and rotate the foot towards the foot location given in the input state, a procedure described as "reachability control" in [15].

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 increasing distance. We represent the terrain as a signed distance field (see figure 4), which allows for very fast distance queries, and which also gives useful gradient information even when the robot lies in collision with the terrain.

Finally, the displacements Δx and ω are applied to obtain a new position and orientation x and R. The process is

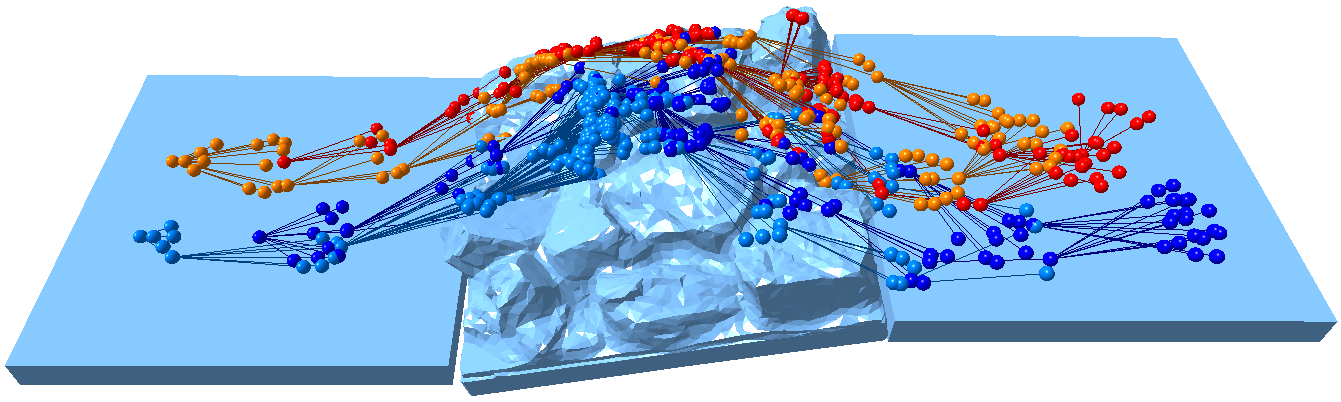

Fig. 5. Visualization of ARA* search for footstep planning. Each state visited during planning consists of a set of four footstep locations. The set of all states visited form a search tree. In the picture above, the 8-dimensional search tree is projected down to 2D so that the parent/child relationships 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.

iterated until no displacements are necessary, or until a preset number of iterations (25 in our system). Pose certificate search is discussed in more detail in our upcoming paper [?].

E. 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. 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 [16]. 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.

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.

Terrain	States	Initial	Ratio
Large rocks (Figs 1,3)	23,973	4.40s	1.67
Round rocks (Fig 3)	33,063	1.24s	2.48
Sloped rocks (Figs 1,5)	46,768	1.23s	1.35
Logs (Fig 1)	41,028	1.09s	1.42
Gap	28,232	1.88s	1.58

TABLE I. 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.

IV. 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.

A. Initial trajectory

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 [17]. 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

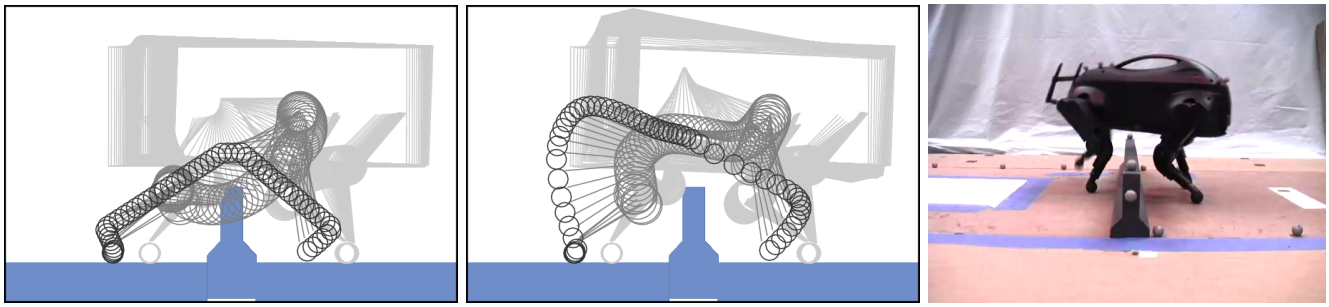

Fig. 6. Optimizing a footstep over a barrier with CHOMP. *Left:* Initial trajectory has severe knee collisions. *Center:* After optimization, collisions have been resolved by lifting the leg higher and tilting the body. *Right:* Execution on robot.

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 6. 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.

B. Optimization

Trajectory optimization is accomplished via the Covariant Hamiltonian Optimization and Motion Planning (CHOMP) algorithm. The bulk of our trajectory optimization approach is described in prior work [18]; here, we describe a few additional details. CHOMP is a local search algorithm for producing smooth, collision-free trajectories. It represents trajectories as sequences of samples taken at regular time intervals.

[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.

Previous systems for generating footstep trajectories have primarily reasoned about collisions for the swing foot alone [19]. 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.

Besides the smoothness and collision-free objective criteria described in prior work, our software also attempts to optimize stability and kinematic reachability. The stability and reachability gradients are handled analogously to collision; after they are computed as described in section III-D, they 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).

CHOMP is a gradient-based optimizer that extends naturally to become a probabilistically complete planner by using the Hamiltonian Monte Carlo procedure [18]. In this work, we omit the stochastic Hamiltonian Monte Carlo step because it is usually unnecessary in practice. Nevertheless, despite certificates, there may exist no feasible trajectory to execute the footstep, in which case the additional stochastic search would become stuck for a long time in deep and broad local minima. For all of these reasons, we believe further refinement would be unnecessary for this application.

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.

V. Execution

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.

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 similar to [15].

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

$$\begin{aligned} \Delta &\leftarrow \Delta + \gamma\,(b - \bar{b}) \\ \theta &\leftarrow \mathrm{IK}(b + \Delta) \end{aligned}$$

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.

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.

VI. 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. 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.

Acknowledgments

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

References

[1] M. Raibert, *Legged robots that balance*. MIT Press, 1986.
[2] U. Saranli, M. Buehler, and D. Koditschek, "RHex: A simple and highly mobile hexapod robot," *The International Journal of Robotics Research*, vol. 20, no. 7, p. 616, 2001.
[3] M. Buehler, R. Playter, and M. Raibert, "Robots step outside," in *Proc. International Symposium on Adaptive Motion in Animals and Machines*, 2005.
[4] J. Kolter and A. Ng, "Learning omnidirectional path following using dimensionality reduction," in *Proc. Robots, Science and Systems*, 2007.
[5] J. Kolter, M. Rodgers, and A. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2008, pp. 811–818.
[6] 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.
[7] J. Chestnutt, M. Lau, K. 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.
[8] J. Chestnutt, "Navigation Planning for Legged Robots," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2007.
[9] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
[10] 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*, vol. 79, 1957.
[11] N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt, "Boosting Structured Prediction for Imitation Learning," in *Advances in Neural Information Processing Systems 19*, B. Scholkopf, J. Platt, and T. Hofmann, Eds. Cambridge, MA: MIT Press, 2007.
[12] N. Ratliff, J. Bagnell, and S. Srinivasa, "Imitation Learning for Locomotion and Manipulation," in *IEEE-RAS International Conference on Humanoid Robots*, December 2007.
[13] 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)*, vol. 1, 1999.
[14] J. Kivinen, A. Smola, and R. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.
[15] M. Stolle, "Finding and Transferring Policies Using Stored Behaviors," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2008.
[16] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in Neural Information Processing Systems*, vol. 16, 2004.
[17] 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.
[18] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, May 2009.
[19] J. Kolter and A. Ng, "Task-Space Trajectories via Cubic Spline Optimization," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2009.