Monte-Carlo Planning for Agile Legged Locomotion

Patrick Clary, Pedro Morais, Alan Fern, Jonathan Hurst
Collaborative Robotics and Intelligent Systems Institute
Oregon State University, Corvallis, USA
{*claryp, autranep, alan.fern, jhurst*}@*oregonstate.edu*

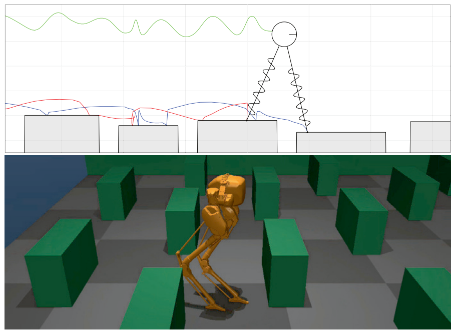

Figure 1: Top: Execution of a plan in a reduced-order planar biped domain. Colored traces show the past locations of the body and feet. Bottom: Full-order simulation of Cassie, a blind-walking robot, in an obstacle-filled environment.

Abstract

Recent progress in legged locomotion research has produced robots that can perform agile *blind-walking* with robustness comparable to a blindfolded human. However, this walking approach has not yet been integrated with planners for high-level activities. In this paper, we take a step towards high-level task planning for these robots by studying a planar simulated biped that captures their essential dynamics. We investigate variants of Monte-Carlo Tree Search (MCTS) for selecting an appropriate blind-walking controller at each decision cycle. In particular, we consider UCT with an intelligently selected rollout policy, which is shown to be capable of guiding the biped through treacherous terrain. In addition, we develop a new MCTS variant, called Monte-Carlo Discrepancy Search (MCDS), which is shown to make more effective use of limited planning time than UCT for this domain. We demonstrate the effectiveness of these planners in both deterministic and stochastic environments across a range of algorithm parameters. In addition, we present results for using these planners to control a full-order 3D simulation of Cassie, an agile bipedal robot, through complex terrain.

Introduction

Legged robots have the potential to traverse complex terrain that fundamentally limits other forms of terrestrial robot locomotion. Unfortunately, motion planning for legged robots is complicated by unstable underactuated dynamics, high dimensionality, and uncertain ground contact. Many different approaches have been tried to make this problem tractable. This paper explores a combination of robust *blind-walking* controllers and Monte-Carlo planning as a novel and compelling approach to agile legged locomotion.

A legged robot that moves around using only proprioceptive feedback about the world can be said to be *blind-walking*. This level of capability is equivalent to a blindfolded human, who can be adept at recovering if they stumble but will haplessly wander into a wall or step into a hole if allowed to do so. Early examples of robots that could competently blind-walk can be found in (Raibert 1986). More recently, the robot ATRIAS used a compliant-gait control approach to demonstrate robust and agile blind-walking that could be compared favorably with human performance (Hubicki et al. 2017). ATRIAS was shown to walk and run over rough outdoor terrain and take large kicks without falling, and had an interface through which an operator could adjust its gait to pilot it around obstacles.

The emergence of agile blind-walking robots that are pilotable by humans raises the possibility that they could by piloted by some planning algorithm. We identify Monte-Carlo planners as a promising class of candidate planners for this application (Browne et al. 2012). They place few requirements on the system to be controlled beyond the existence of a simulator. Simulation of legged systems has been considered difficult because of the complexity of contact dynamics, but work with ATRIAS showed that a compliant gait made it possible to match real behavior with a simple and generic ground model (Hubicki et al. 2017).

Monte-Carlo planners are able to find good plans with sparse sampling, which will be necessary for tractable real-time motion planning. However, they still choke on domains with very large action spaces that require deep planning horizons, which makes them a poor choice for controlling robots at the joint torque level. In this paper, we show that using a blind-walking controller below the planner allows the action space and decision complexity to be greatly reduced while still allowing for agile movement.

Competing approaches for achieving agile legged locomotion include whole-body planning and control architectures like those used in the DARPA Robotics Challenge Finals (Kuindersma et al. 2015; Feng et al. 2015; Johnson et al. 2017). These methods use reduced-order models of the robot in its environment to generate realistic control targets for sophisticated optimization-based movement controllers. They excel at performing general movement and manipulation tasks, but computational and sensory limitations have thus far kept them from displaying the agility and robustness seen in blind-walking robots like ATRIAS.

Another approach to agile locomotion is to forgo planning entirely and rely on trained deep networks to make smart decisions. This was seen in (Peng et al. 2017) and (Heess et al. 2017) for humanoid models, but has not been demonstrated on robot hardware or a realistic simulation of a physical robot. We anticipate that future agile robots may use learned feed-forward policies to accelerate decision making even when a planner is used to provide accountability.

Blind-walking robots are in need of guidance to help them avoid obstacles that cannot be handled by brute force and robust recovery. Conversely, Monte-Carlo planners are in need of a way to greatly reduce the dimensionality of controlling a complex robot. In this paper, we show that combining these elements produces a control architecture capable of agile movement through treacherous environments. We present an adaptation of a standard Monte-Carlo Tree Search for the robust blind-walking robot domain. We also present a novel type of Monte-Carlo planner, Monte-Carlo Discrepancy Search (MCDS), and demonstrate its performance in real-time scenarios with severely limited planning time. We test our algorithms both on a simplified planar biped domain and on a full-order 3D simulation of Cassie, a successor to the blind-walking robot ATRIAS (Figure 1). A supplemental video shows footage from ATRIAS and examples of MCDS in action on both the planar and 3D domains.

Blind-walking as a Planning Domain

In this section, we discuss how robotic walking using blind-walking controllers is translated into a format that Monte-Carlo planners can operate on. We examine two robot models of varying complexity. Most of our experiments use a planar model of a reduced-order biped, but we also demonstrate our planners on a full-order model of Cassie, a blind-walking robot. The parameters of the planar model, such as leg length and body mass, are chosen to resemble Cassie.

States. State is represented by the position, velocity, and orientation of the base of the robot, along with the robot's internal joint positions and velocities. Both robots have series compliance in the legs, which adds an additional position and velocity to the state for each relevant axis. Terminal states are reached when the robot is no longer upright and able to take steps. In our tests, we defined limits for the robot's orientation and height above the ground beyond which a state is considered terminal.

Dynamics. The reduced-order biped is still complex enough that its dynamics are not closed-form, so it is simulated numerically using a fixed-step fourth-order integrator and a smooth spring-damper ground contact model. The model includes body mass and inertia, foot mass, and spring-dampers in series with leg length and angle actuators. The full-order simulation of Cassie uses MuJoCo (Todorov, Erez, and Tassa 2012) as its physics engine, with modeling parameters provided by the robot's manufacturer, Agility Robotics. For tests involving a stochastic model, Gaussian noise is added to state velocities before running the simulator. This is equivalent to random impulsive forces acting on the robot and propagating through its dynamics.

Actions. Both the reduced- and full-order robot models have had robust blind-walking controllers developed for them. These controllers follow principles developed for the robot ATRIAS (Hubicki et al. 2017), combining a feed-forward clock-driven gait cycle with feedback on body velocity for choosing stabilizing footstep targets. These controllers have a number of parameters affecting the resulting gait, such as target speed, step height, and jump height. The controller for the reduced-order biped has five real-valued parameters. The controller that we have for the full-order model is more limited in the behaviors it can perform, and has four real-valued parameters.

Rewards. Planning the actions of a blind-walking robot is meant to give them some environmental awareness, allowing them to avoid obstacles that would otherwise trip them. A typical planning goal would be to make progress in a desired direction without falling, rather than to reach a specific state. To implement this type of goal, we designed reward functions, operating over complete plans rather than individual states, with the following terms:

- A reward for approximately matching the goal velocity
- A penalty for pitching the body wildly
- A large penalty for reaching a terminal state
- A penalty for shorter plans before reaching terminal states

Monte-Carlo Planning

We use Monte-Carlo planning to choose actions for a blind-walking robot that allow it to move through difficult terrain without falling. We begin with some of the details of using a Monte-Carlo planner on this domain, then present an application of UCT, a standard Monte-Carlo Tree Search. We also present MCDS, a new type of planner that exhibits better performance with limited planning time in this domain.

Domain Instantiation for Monte-Carlo Planners

Action Space. The actions considered by the Monte-Carlo planners will correspond to selecting parameter values for the blind-walking controller and letting the controller execute for a short period of time using those parameters. Thus, the naive action space is continuous and multi-dimensional, which is difficult to integrate directly into standard Monte-Carlo planners, which expect discrete action spaces. Thus, we defined a discrete set of actions by identifying a small set of parameter values (six for the planar biped, seven for the full robot) that cover the most common types of behavior. These actions represent behaviors like walking forward, taking shorter or longer steps than usual, jumping, sidestepping, and turning. Using these actions, it is always possible find a plan for traversing the terrain considered in this work if given enough planning time.

Rollout Policy. Monte-Carlo planning relies on performing rollouts to estimate the expected values of decisions. Randomly selecting actions is a common rollout policy, but doing so in this domain interferes with the self-stabilizing properties of blind-walking control, and will often cause the robot to fall over on flat ground. Instead, a better rollout policy for this domain is to repeat the previous action until the horizon or a terminal state is reached. Note that for the action space of this paper, this rollout policy corresponds to the continued execution of the blind-walking controller at a fixed set of parameter values.

Decision Cycles. To discretize the domain in time, we defined a consistent decision cycle of 0.3 seconds. The robot executes a particular action for one decision cycle, meanwhile planning the next action. We found a constant time more effective than tying actions to the gait cycle, and that this value balanced permitting frequent action changes against giving each action a sufficient amount of planning time.

Planning Horizon. A planning horizon equivalent to several steps is useful in this domain because there is a minimal number of steps needed for an underactuated walker to arbitrarily change speed and direction (Zaytsev, Hasaneini, and Ruina 2015), but in realistic scenarios stochasticity makes predicting far into the future useless. Humans look three steps ahead while walking in challenging terrain (Matthis and Fajen 2014). We found that a horizon of 3 seconds was broadly effective in the planar domain, but that longer horizons performed better for the full-order robot.

Simulation Time. Even in the reduced-order domain, planning is computationally limited by simulation speed. The maximum real-time simulation rate achievable for the full-order Cassie model on mobile hardware is approximately 100× real-time. This translates to allowing the planner to simulate a total of 30 seconds of real time during each decision cycle (of 0.3 seconds each). Unless otherwise specified, we restrict planning to this amount of simulated time to evaluate the real-time planning performance.

Planning Algorithms

UCT. We use UCT (Kocsis and Szepesvári 2006) as a benchmark Monte-Carlo Tree Search algorithm. UCT is a rollout-oriented planner that uses the UCB1 bandit criterion to choose where to expand the tree. The results of rollouts are used to refine value estimates for actions at the root of the tree. In our instantiation, rollouts continue to a fixed horizon or until a terminal state is reached, and the plan starting from the root is evaluated with our reward function and backpropagated. As simulation is a more significant bottleneck than memory in this domain, rollouts are saved in the tree. The exploration parameter C_p used in UCB1 was chosen by testing across a wide range, but was found to have little impact on performance in the real-time decision-making setting. This is because, in the real-time setting, each decision cycle only allows approximately 10 rollouts per decision cycle. With six actions to choose from, this generally corresponds to first trying each action at the root of the tree, followed by rolling out that action until the horizon. Next, a small number of

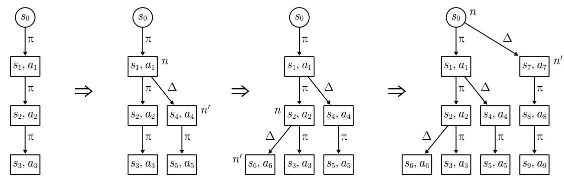

Figure 2: A sequence of search trees generated by MCDS. The leftmost tree is the result of the initial policy rollout. Each subsequent tree adds a discrepancy.

rollouts are executed using the UCB1 rule to expand untried choices at the second level of the tree.

In an online planning context, it is useful to reuse results from the previous decision cycle to warm-start the planning process. For UCT, this can be done by re-rooting the tree at the child of the root that was previously chosen. Some nodes in this subtree will already have value estimates, which can be refined with further rollouts. However, in the domains explored in this paper, re-rooting UCT has no observable benefit because the limited simulation time allowed means that there are very few value estimates for children of the chosen action. Because of this, the results presented in the results section reflect UCT without re-rooting the tree to warm-start the search.

MCDS. We designed Monte-Carlo Discrepancy Search (MCDS) to address some of the shortcomings of UCT and similar planners that we encountered in this domain. Namely, UCT does not make effective use of results from previous decision cycles to warm-start its search, and it makes few decisions using UCB1 when given severely limited planning time, making it little better than trying a single rollout from each of the actions available at the root.

In contrast to planners like UCT that estimate values of root actions, MCDS is oriented around finding and maintaining a current best path out to a receding horizon across iterations of re-planning. This path may be invalidated as the horizon recedes or by stochasticity, necessitating additional searching, but if this happens infrequently, planning costs can be amortized over multiple decision cycles. MCDS searches among *discrepancies* from its initial plan, and when the planner terminates at the end of a cycle it chooses the highest-scoring plan that it found.

Our implementation of MCDS begins a decision cycle with a tree rooted at the state predicted to result from the action being executed by the robot. The first rollout it performs follows the actions along the best plan identified in the previous cycle, with an additional action selected at the end using the rollout policy π due to the receding horizon. When a rollout completes, by reaching the horizon or a terminal state, a *discrepancy generator* Δ chooses a node in the tree to branch from and a new action to try as the start of a rollout, as shown in Figure 2. For our experiments, we select this node by first randomly selecting a depth, then choosing uniformly among all nodes at this depth. The action is selected randomly from among the actions that have not been tested from this node.

When the planning time runs out, MCDS evaluates each of the plans represented by leaf nodes in the planning tree with our reward function and returns the first action along the highest value plan, saving the remaining actions in the plan as the seed for the next decision cycle.

Comparison of MCDS and UCT. MCDS compared to UCT makes better use of limited planning time in our blind-walking domain. The biggest difference is that UCT essentially conducts a breadth first expansion of the search tree. That is, UCT will always try all actions available from a decision node at least once before enumerating the action choices of the node's children. Thus, even if UCT uses a warm-start and initializes the next decision cycle with the best trajectory (or sub-tree) found in the previous cycle, UCT will focus on trying all actions at the root that were not in the warm-start trajectory. The remaining simulation time will then be spent enumerating and rolling out all depth-two action choices under the root.

This focus on "searching at the top" by UCT has been observed to be fairly unproductive in our blind-walking domain with limited decision time. In particular, searching at the top of the tree corresponds to very frequent switching (every 0.3 seconds) between parameters of the blind-walking controller, which usually is not required or desirable. Rather, switching between control parameters is more important when the robot enters a qualitatively different type of terrain or dynamic state. This is the main motivation for allowing MCDS to introduce a new discrepancy at any depth of a trajectory currently in the tree. MCDS is effectively searching over possible switching points for control parameters. While this selection of switching points is currently random, we believe that future work that incorporates learning to inform these decisions will be able to dramatically improve the efficiency of the search.

Experimental Results

The effects of horizon length and planning time were examined for both UCT and MCDS on the reduced-order planar biped domain. All tests use a deterministic version of the model during the planning phase, but some tests use a stochastic model for advancing the state of the robot, showing how re-planning after each decision cycle affects algorithm performance in the event of an imperfect model. We also show that these planners can be used successfully without modification on a full-order robot simulation. A supplemental video shows test runs of MCDS controlling both the reduced- and full-order robot models.

Terrain. Blind-walking controllers can be very robust to rough terrain, but can not natively avoid stepping in holes or tripping on steps. To test how our planners provide these capabilities, we generated randomized terrain consisting of platforms of varying height with gaps between them. Height differences between adjacent platforms range between ± 0.1 m, platform width ranges from 0.3–1 m, and gap width is between 0.1–0.2 m. For tests with the full-order model, we set up a course with regularly spaced barriers that the robot has to weave through.

Stochasticity. Stochasticity in legged locomotion can take the form of constant noise from ground height and stiffness variations as well as large, infrequent disturbances like a foot slipping or the robot being pushed. The latter type of disturbance is handled well by blind-walking control alone,

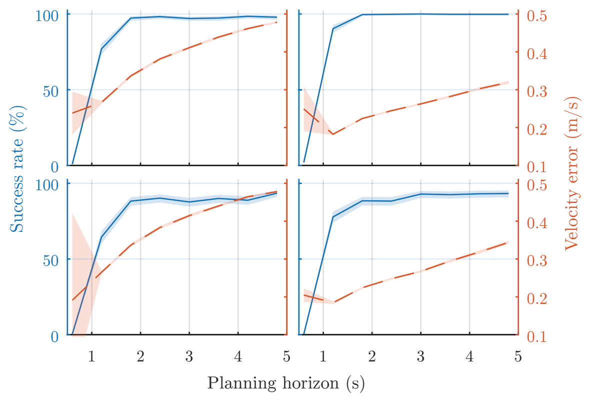

Figure 3: Success rate (solid blue) and velocity error (dashed red) for varying planning horizons with fixed simulation time. UCT on left, MCDS on right, deterministic simulation on top, stochastic simulation on bottom.

so this paper examines the former's impact on deterministic planning. Constant noise is modeled by adding a small random impulse to the robot's velocity at the beginning of each simulator call. The magnitude of the impulse is scaled such that longer plans are frequently invalidated, but the robot is not thrown irrecoverably towards a fall.

Metrics. For each configuration of planner and domain, several hundred trials were run in which the biped was directed to walk over randomly generated terrain for 30 seconds or until falling. The results are compiled into the percentage of trials that do not fall before the end of the run (*success rate*) and the root-mean-square velocity error averaged over the successful trials (*velocity error*). Plots show 95% confidence intervals. Real-world usage will generally prioritize a high success rate over low velocity error.

Varying Horizon. Figure 3 shows results from tests in which the planning horizon was varied for a fixed amount of planning time. Longer horizons have an increased ability to identify upcoming terminal states, but create sparser trees that sample fewer possible action sequences. We see that the success rate increases with horizon length but reaches a plateau after about two seconds, while velocity error increases with horizon length. MCDS outperforms UCT in terms of velocity error in all cases, and has a slightly better success rate with short horizons. Tests using a stochastic model show a reduced success rate compared to the deterministic case, but equivalent velocity error.

Varying Planning Time. Figure 4 shows results from varying the planning time available during each decision cycle. To separate the effects of stochasticity from those of plan quality on our metrics, these tests used only a deterministic model. MCDS has a notably better success rate than UCT with small planning budgets, and continues the trend of lower velocity error in all tests.

Full-order Model. Finally, UCT and MCDS were used to guide a full-order 3D model of Cassie through a grid of obstacles. The algorithm parameters were kept the same,

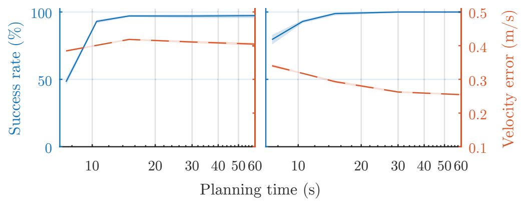

Figure 4: Success rate (solid blue) and velocity error (dashed red) for varying amounts of simulated time allowed per decision cycle. UCT on left, MCDS on right. Both cases use a deterministic simulator.

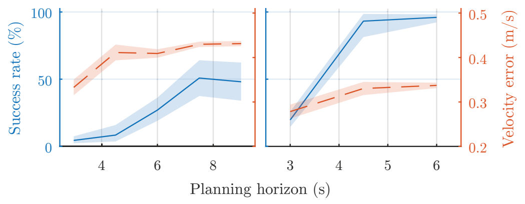

Figure 5: Results from the full-order robot model. Success rate (solid blue) and velocity error (dashed red) for varying planning horizons with fixed simulation time. UCT on left, MCDS on right.

including the simulated time budget of 30 seconds per 0.3 second decision cycle. Figure 5 shows results for different planning horizons. We found that increasing the planning horizon beyond three seconds greatly improved MCDS' success rate, and that MCDS significantly outperformed UCT on this domain. With a fixed planning time budget, MCDS reached near-100% success with a horizon of 4.5 seconds, while UCT's performance peaked near 50% at a 7.5 second horizon, requiring more planning time to reach a comparable success rate to MCDS.

Conclusions and Future Work

This paper presents an example and a strong argument for a control hierarchy that combines a self-stable blind-walking controller at the base level with a fast Monte-Carlo planner that guides it through and around obstacles. Despite having a very small action space, a planning rate of about 3 Hz, and a realistic planning time budget, our MCDS planner was able to reach a nearly 100% success rate on the planar domain and a 95% success rate on the full-order domain.

Future work will look towards a more comprehensive treatment of stochasticity. MCDS as described in this paper is appropriate for deterministic domains, but we show that it achieves passable performance on our stochastic domain through a combination of frequent re-planning and the self-stability of the underlying blind-walking controllers. With explicit consideration of stochasticity, MCDS may be suitable for a broader range of domains, such as blind-walking with underlying controllers that are less self-stable.

Elaborations of MCDS as applied to the blind-walking domain will investigate more sophisticated versions of the discrepancy generator used to start new rollouts. The experiments in this paper show reasonable performance with random selection of nodes and actions, but MCDS may be able to find good plans in even less time if learning is used to make informed decisions about where to introduce a discrepancy and which actions to sample.

Acknowledgements

This work was supported by NSF grant IIS-1619433 and DARPA contract W911NF-16-1-0002. We thank Intel for providing compute support for this work.

References

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Feng, S.; Whitman, E.; Xinjilefu, X.; and Atkeson, C. G. 2015. Optimization-based Full Body Control for the DARPA Robotics Challenge. *Journal of Field Robotics* 32(2):293–312.

Heess, N.; TB, D.; Sriram, S.; Lemmon, J.; Merel, J.; Wayne, G.; Tassa, Y.; Erez, T.; Wang, Z.; Eslami, S. M. A.; Riedmiller, M.; and Silver, D. 2017. Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286 [cs]*. arXiv: 1707.02286.

Hubicki, C. M.; Abate, A.; Clary, P.; Rezazadeh, S.; Jones, M.; Peekema, A.; Van Why, J.; Domres, R.; Wu, A.; Geyer, H.; and Hurst, J. W. 2017. Walking and Running with Passive Compliance: Lessons from Engineering a Live Demonstration of the ATRIAS Biped. *IEEE Robotics and Automation Magazine*.

Johnson, M.; Shrewsbury, B.; Bertrand, S.; Calvert, D.; Wu, T.; Duran, D.; Stephen, D.; Mertins, N.; Carff, J.; Rifenburgh, W.; Smith, J.; Schmidt-Wetekam, C.; Faconti, D.; Graber-Tilton, A.; Eyssette, N.; Meier, T.; Kalkov, I.; Craig, T.; Payton, N.; McCrory, S.; Wiedebach, G.; Layton, B.; Neuhaus, P.; and Pratt, J. 2017. Team IHMC's Lessons Learned from the DARPA Robotics Challenge: Finding Data in the Rubble: Team IHMC's Lessons Learned from the DARPA Robotics Challenge. *Journal of Field Robotics* 34(2):241–261.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Kuindersma, S.; Deits, R.; Fallon, M.; Valenzuela, A.; Dai, H.; Permenter, F.; Koolen, T.; Marion, P.; and Tedrake, R. 2015. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots* 1–27.

Matthis, J. S., and Fajen, B. R. 2014. Visual control of foot placement when walking over complex terrain. *Journal of experimental psychology: human perception and performance* 40(1):106.

Peng, X. B.; Berseth, G.; Yin, K.; and van de Panne, M. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*.

Raibert, M. H. 1986. *Legged Robots that Balance*. MIT Press.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 5026–5033. IEEE.

Zaytsev, P.; Hasaneini, S. J.; and Ruina, A. 2015. Two steps is enough: no need to plan far ahead for walking balance. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 6295–6300. IEEE.