

Quadruped robot obstacle negotiation via reinforcement learning

Abstract—Legged robots can, in principle, traverse a large variety of obstacles and terrains. In this paper, we describe a successful application of reinforcement learning algorithm to the problem of negotiating obstacles with a quadruped robot. Our algorithm is based on a two-level hierarchical decomposition of the task, in which the high-level controller selects the sequence of foot-placement positions, and the low-level controller generates the continuous motions to move each foot to the specified positions. The high-level controller uses a value function approximation to guide its search, and we describe a novel algorithm to learn the value function; the low-level controller is obtained via policy search. We demonstrate that our robot can successfully climb over a variety of obstacles which were not seen at training time.

I. INTRODUCTION

While wheeled vehicles (such as cars and trucks) are very fuel efficient, legged robots can, in principle, climb over larger obstacles (relative to the size of the robot) and thereby access significantly more difficult terrain. In this paper, we apply reinforcement learning to develop a novel controller for a quadruped robot capable of negotiating a wide variety of obstacles, including ones that had not been previously seen during training.

In related work, other authors have focused on mechanical design of legs that automatically adapt to terrains, without requiring closed-loop control [1], [2]. Bai et al. proposed a rule-based algorithm to generate free gaits in 3D terrain [3], but only demonstrated successful stepping over a small and simple obstacle. Also there has been work for adaptive gait for a quadruped robot on 3d path planning using biologically-inspired "central pattern generator" [4]–[9], but they are more focused on walking on terrain of medium degree of irregularity rather than climbing over obstacles. Buehler et al.'s hexapod (six-legged) RHex Robot is capable of climbing up steps, but does so using clever mechanical design in which large, circular, "wheel-like" legs flail at the obstacles, and thus avoiding the need for careful balance and coordination [10], [11]. There are also several robots that rely on hopping to climb up steps, e.g. Raibert's biped robot [12] and Poulakakis et al.'s galloping robot [13]. However, these hopping gaits are not stable enough for traversing rugged terrains or climbing irregular obstacles. Latombe et al.'s vertical climbing robot has successfully applied motion planning algorithm to the robot climbing problems [14]. Their algorithms relied on computing statically stable poses for vertical climbing, and do not extend to more general obstacle negotiation tasks. Learning algorithms have been successfully applied to a few legged locomotion problems, for example [15]–[17]. But these

methods have been demonstrated to work only on flat terrain in which the same (periodic) gait could be repeated without taking into account possible obstacles in the path. Kim et al. [18] proposed a gait planning algorithm that carefully calculates a trajectory of a quadruped robot using geometry, but this method is limited to boundaries of two planar surfaces, and thus not generalizable to wide range of obstacle shapes. Chestnutt et al. [19] used hierarchical planning to deal with obstacle avoidance, but considered only 2D path planning and 3D obstacle avoidance ("stepping" over small obstacles), but not the more difficult task of climbing over obstacles.

Our method is based on hierarchical reinforcement learning [20]–[23] using a two-level hierarchical decomposition of the task. Given an obstacle (such as a step) that the robot needs to climb over, the high-level planner selects a sequence of "target foot placement positions" for the robot, one foot at a time. It is then the low-level controller's task to move the feet to these targets in order, while keeping the robot balanced and preventing the legs from hitting any obstacles.

The low-level controller operates on a very short temporal scale (about 1-3 seconds per footstep), in a problem which requires fine coordination and balance, but does not require careful multi-step reasoning. Policy search algorithms [24] with simple parameterized policies apply well to this setting, and is used to learn the low-level controller.

In contrast, the high-level controller has to plan a sequence of foot placement positions, and must reason on significantly longer timescales. We build a high-level controller using value function approximation and beam search. More precisely, we learn a value function via an algorithm that we call **max-margin reinforcement learning**, and apply beam search with the learned value function to find the sequence of foot placements that maximizes the reward. This sequence is then executed by the low-level controller.

Hierarchical reinforcement learning algorithms have been applied to a number of other problems, for example various grid-world variants (e.g., [23]). But to our knowledge, it has not yet been successfully applied to any continuous state-space problems of comparable size and complexity to ours.

II. QUADRUPED ROBOT MODEL

The quadruped robot used in this work is shown in Figure 1. The robot is approximately 9.5 cm tall, 27.0 cm wide, and 11.1 cm long, and weighs 500 grams. Each leg has 3 servomotors, two for rotating the upper-leg forward/backward, up/down, and one for rotating the lower-leg inward/outward. The servos have a maximum torque 0.03 kg-m, and may fail to move to the

specified by \tilde{g}_t .¹

This idea needs couple of additional refinement. Naively following the gradient of the potential function will only work if there are no local minima aside from the one at the goal position. However, near the front of obstacles, it is possible for the attractive goal potential and the repulsive surface potential to cancel, causing valleys to form in the potential function. To deal with this problem, we use a vortex field [27]. The vortex field is a vector field that goes around the obstacles. It is motivated by the simple observation that the moving foot must not only avoid colliding with obstacles, but also go around obstacles in order to reach the goal position. So near the front of obstacles where the attractive goal potential and the repulsive surface potentials cancel, the vortex field will cause the moving foot to move upwards over the obstacle. Details on the vortex field can be found in the Appendix.

Another problem is keeping the three supporting feet stationary while moving one foot. At every step, the robot can be thought of as trying to change its state in the direction of \hat{g}_t , where \hat{g}_t is given by the potential and vortex fields. In general, following \hat{g}_t exactly would also change the position of the supporting (non-moving) feet $u_t^{s_1}$, $u_t^{s_2}$, and $u_t^{s_3}$, where s_1 , s_2 , and s_3 are the indices of the robot’s supporting feet. In order to avoid this (because we want to move only one leg at a time), we project \hat{g}_t into the subspace of motions which keeps the positions of the supporting feet constant. More precisely, we define $\Phi_t \in \mathbb{R}^{18 \times 9}$ to be the matrix whose columns are the gradients of the components of $u_t^{s_1}$, $u_t^{s_2}$, and $u_t^{s_3}$ with respect to the 18 state variables. The change in the positions of the supporting feet due to a small change in the state variables $\delta\Omega_t$ is approximately $\Phi_t^T \cdot \delta\Omega_t$. Hence, in order to keep the supporting feet stationary, we should only move in directions that are in the nullspace of Φ_t^T . Now, let \hat{g}_t^* be the projection of \hat{g}_t onto the nullspace of Φ_t^T . Finding \hat{g}_t^* can be posed as the following minimization problem:

$$\begin{aligned} \min \quad & \|\hat{g}_t^* - \hat{g}_t\|_2 \\ \text{s.t.} \quad & \Phi_t^T \cdot \hat{g}_t^* = 0 \end{aligned} \quad (4)$$

The closed form solution for \hat{g}_t^* is:

$$\hat{g}_t^* = (I - \Phi_t \cdot (\Phi_t^T \cdot \Phi_t)^{-1} \cdot \Phi_t^T) \cdot \hat{g}_t \quad (5)$$

Thus, we change the joint angles in the direction of \hat{g}_t^* .

In choosing the form of the potential function, we were thus able to encode a significant amount of prior knowledge about the task of moving a single foot. Specifically, minimizing the combination of the three potential functions will tend to cause the robot to move the foot toward the goal while maintaining balance and avoiding collisions with obstacles or with the ground. Further, the projection onto the nullspace of Φ_t prevents the supporting feet from moving. All this prior knowledge, encoded into the low-level controller’s policy class, helps it to solve the high-dimensional search problem

¹Only 12 of the components of \tilde{g}_t correspond to robot joint angles. The other 6 components are ignored.

associated with moving a single foot to a new position.² That we can encode so much prior knowledge into the policy parameterization is one of the strengths of policy search. However, the policy class still has many free parameters which are difficult to choose by hand. For example, even though it is easy to specify the form of a potential that repulses a foot from an obstacle, it is hard to decide by hand exactly what the *magnitude* of this repulsion should be. In our approach, we learn the parameters of the potential functions automatically, as discussed in the next section.

A. Policy search

The low-level controller is parameterized by 20 numbers that govern various trade-offs in its attractive and repulsive potentials. To learn these parameters, we began by building a dynamical simulator³ of the robot following the specifications of the real robot. Using the simulator, we apply PEGASUS policy search [24], implemented with locally greedy hill climbing, to optimize the potential function parameters so as to maximize the expected reward. In our experiments, the reward function is simply $R(s) = -1$ in each state until a footstep is completed (or a large negative reward if the robot falls over), so that the total reward simply measures the total duration of a footstep. The low-level controller was trained on a fixed set of tasks, each of which involved moving a single foot to a new location. The learned parameters were then fixed, and the resulting low-level controller used henceforth by the high-level planner.

IV. HIGH-LEVEL PLANNER

Given the low-level controller for moving a single leg, we now need a high-level planner for generating the sequence of target foot positions. This is a difficult search problem because a bad choice of foot position early in the sequence could lead to the robot being stuck in an impossible position at a later stage. Performing exhaustive search is computationally expensive due to the high branching factor and large search depth involved.

In order to avoid the expense of exhaustive search, there needs to be a way of measuring how “good” a state is. For example, in A^* search, this role is fulfilled by the heuristic function which estimates the total future cost to the goal. However, specifying a heuristic function is often difficult to do in complex problems because it requires estimating the entire sequence of unknown future costs. In reinforcement learning [28], the “goodness” of a state s is measured by the value function $V(s)$, which is defined as the “expected total rewards” starting from the state s . However, unlike the heuristic function in A^* , the value function in reinforcement learning is not directly specified, but rather learned automatically. A large number of RL algorithms already exist for learning the

²The action space used by the low-level controller is 12d and not 3d, since even if the current step requires only moving foot 1, we may still wish to move the servomotors in the other legs to maintain balance.

³We used Open Dynamics Engine(ODE) for simulation. See <http://ode.org> for more detail.

value function. However, many of these algorithms have only been successfully demonstrated on simple problems with small state-spaces, while others suffer from long running times and lack of convergence guarantees. To our knowledge, there has been no previous successful application of RL to a problem as complex as robot obstacle climbing. In this paper, we present a novel RL algorithm that was able to quickly learn a value function that enabled our quadruped to climb a variety of obstacles.

A. Reinforcement learning preliminaries

In reinforcement learning, we define a set of states S and a set of actions A . The reward function $R(s, a)$ is defined for each state-action pair (s, a) . The optimal value function satisfies the Bellman equation

$$V(s) = \max_a R(s, a) + \gamma V(s'), \quad (6)$$

where s is the current state, s' is the state reached upon taking action a in state s ⁴, and γ is a discount factor for future rewards ($0 \leq \gamma \leq 1$). In other words, the maximum total rewards starting from the current state is found by maximizing the sum of the immediate one-step reward and the future rewards from the next state. The optimal action at state s is simply the action that achieves the above maximum

$$a_{opt} = \arg \max_a R(s, a) + \gamma V(s'), \quad (7)$$

In our high-level planner, taking an action a corresponds to choosing a new position for one of the feet. The reward function we used gave positive rewards for (1) movement of the robot's CG towards the goal; small negative rewards for (2) the time it took to execute the movement; and very large negative rewards for (3) failure of low-level controller to execute the action.

B. Value function approximation

For large continuous state spaces, it is generally not possible to obtain $V(s)$ exactly. In our problem, we approximate the value function $V(s)$ as a linear function of the following features of the state⁵: (1) Distance from robot's CG to the goal; (2) Distance from average foot position to the goal (3) Orientation of the body ($1 - \cos(\omega_1), 1 - \cos(\omega_2), 1 - \cos(\omega_3)$); (4) Maximum knee angle; (5) Surface roughness⁶; (6) Surface curvature⁷; (7) Difference between maximum and minimum height of the foot positions; (8) Area of supporting triangle; (9) Radius of the largest circle that inscribes the supporting triangle; (10) The distances between each pair of feet. This

⁴We are assuming deterministic dynamics in our description while the general RL framework is most often applied to problems with stochastic dynamics.

⁵Implementation detail: We fixed the order of moving foot as in the conventional trot gait, so we mean a *state* as *extended state* which includes 18d state space representation of the robot along with information about the moving foot.

⁶Surface roughness is defined as a gaussian weighted sum of square of gradient of $z(x, y)$ over a small patch centered at each foot on the ground.

⁷Surface roughness is defined as a gaussian weighted sum of laplacian of $z(x, y)$ over a small patch centered at each foot on the ground.

gives us our feature vector $\phi(s)$. We approximate the value of the state as $V(s) = \phi(s)^T \theta$, where θ parameterizes our linear value function approximator. Hence, the problem of learning the value function in our problem reduces to the problem of learning θ .

C. Max-margin reinforcement learning

We developed a new method called **max margin reinforcement learning**. This algorithm optimizes a value function by maximizing a *margin* between every pair of optimal and suboptimal actions (target foot positions), as well as fitting the value function into a reinforcement learning framework. Here the *margin* means a minimum difference between two labeled groups (optimal and sub-optimal actions in our case) of data. So a positive value-function margin means that the value function is able to discriminate optimal and sub-optimal actions at least by that margin. In this formulation, we specify pairs of optimal action a_{opt}^t and sub-optimal action a_{subopt}^t given an optimal state s_{opt}^t for each time step t , then optimize θ by maximizing the "future reward" margin between the optimal and suboptimal actions with the reinforcement learning fixed-point constraint.⁸

$$\begin{aligned} \max \quad & \delta - \beta \sum_i \|V(s_{opt}^t) - \gamma V(s_{opt}^{t+1}) - R(s_{opt}^t, a_{opt}^t)\|_1 \\ \text{s.t.} \quad & \gamma V(s_{opt}^t) + R(s_{opt}^t, a_{opt}^t) - \gamma V(s_{opt}^{t'}) - R(s_{opt}^t, a_{subopt}^t) \geq \delta \\ & \|\theta\|_1 \leq c \end{aligned} \quad (8)$$

where $s_{opt}^t, s_{opt}^{t'}$ are the states followed by taking action a_{opt}^t, a_{subopt}^t at state s_{opt}^t respectively, and β, c are constants. In this optimization, we want to maximize the margin between pairs of optimal and sub-optimal actions, while minimizing the deviation of value function from the "fixed point of reinforcement learning". So the objective has some penalty term regarding this violation of the fixed-point constraint. Since we approximate the value function $V(s)$ as a linear function of the features, the problem above becomes a linear program (LP). The advantage of this algorithm is that we can quickly solve this optimization using LP (in a few seconds), and also easily modify the features for the value function. In our experiments, we generated a training set consisting of 400 pairs of optimal-suboptimal target foot positions (actions) throughout a sequence of 16 footsteps for climbing a single 2.6cm step, and then performed the above optimization to learn θ .

We also tried solving our problem using temporal difference (TD) learning [28], one of the best known traditional reinforcement learning algorithms. Unlike our new algorithm, TD learning does not make use of human knowledge in the training process.⁹ The TD learning algorithm failed to produce

⁸Meaning that we satisfy the Bellman equation given by Eq. 6

⁹Although we specify additional inputs from human, making a training examples is not difficult. This is because we only need to make a reasonably good sequence of footsteps as optimal actions first, and then specify sub-optimal actions as the neighbor points of the optimal target foot positions. Also we do not need to explicitly specify value function for that each foot positions.



Fig. 3. Simulated robot executing a sequence of controls to climb up a single 4 cm step.

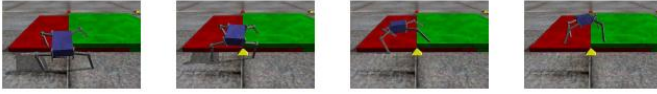


Fig. 4. Simulated robot climbing up an irregular step; the right portion of the step is 6cm and the left portion of the step is 3cm.

a good value function even after running for more than 10 hours.

Based on this learned θ , we perform a motion planning using a beam search to look multiple steps ahead. In our implementation, the high-level planner uses a deterministic simulator for the robot, and using $V(s)$ as the heuristic function with which to guide beam search, attempts to find a sequence of foot placements that take the robot all the way to the goal. Beam search proceeds in a discretized space of actions a (possible foot placements), and has a branching factor of 15. We used a beam width of 10.

V. EXPERIMENTAL RESULTS

All learning was done in simulation. After learning for both the low- and the high-level controllers completed, we tested the resulting hierarchical policy on a variety of obstacles. In the first experiment, we ask the robot to climb up a step of height (4 cm). Figure 3 shows the resulting sequence of robot motions. We also test the robot on an irregular step. (See Figure 4.) Here, the right half portion of the step is 6 cm high, and the left portion is 3 cm high. The

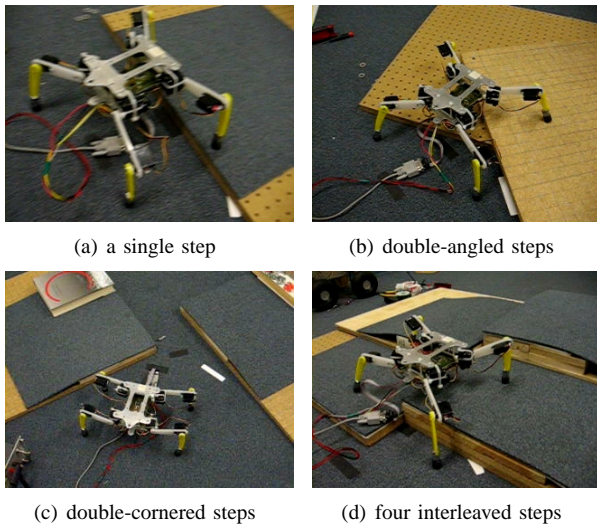


Fig. 5. Real robot executing learned sequence of controls to climb up obstacles.

step height difference and the right portion's high step makes the climbing task more challenging than the first experiment, since a highly asymmetric (and, we find, somewhat non-intuitive) gait is now required. In addition to these examples, the learned controller appears quite robust, and is able to climb a large variety of different obstacles, including steps at various heights (0-4cm); ones oriented at different angles relative to the robot; and irregular steps of different heights. We have also successfully demonstrated some of these gaits on the real robot. For example, Figure 5 shows the real robot executing the gait shown in Figure 3, and other gaits for different obstacles. Videos of the simulated and actual robot executing the learned gaits are available online, at

http://youtube.com/profile_videos.php?user=icra2006.

Note that we have used a training set (optimal and sub-optimal target foot positions) for climbing only a 2.6cm single step. So our results demonstrate that high-level planner algorithm is generalizable to wide range of obstacles *unseen* at training set. We know of no other algorithms that can make a quadruped similarly negotiate obstacles that were not seen at training time. For comparison, we also implemented the Rapidly exploring Random Trees (RRT) algorithm [29]. This is a commonly-used nonholonomic motion planning algorithm, that in principle can be applied to tasks such as ours. We spent significant effort adjusting RRT's parameters, but in all versions, even after running for more than twenty hours, it was never able to find control sequences for negotiating a single 4cm step. In robotics, one commonly used measure for how large an obstacle a robot can climb is the ratio of the obstacle height to the robot's ground clearance. Our algorithm is able to negotiate 4cm obstacles, which is 42% of its ground clearance. This is comparable to the best quadruped robots that we are aware of; for example, [13] used a jumping gait that was able to leap onto an obstacle 40% of its height.

VI. CONCLUSIONS

We have described a two-level hierarchical controller that successfully makes a quadruped robot with a 36 dimensional state-space negotiate a number of non-trivial obstacles. We believe that hierarchical reinforcement learning holds rich promise for negotiating obstacles and rough terrain with biped, quadruped and hexapod robots.

REFERENCES

- [1] S. Hirose, A. Nagabuko, and R. Toyama, "Machine that can walk and climb on floors, walls, and ceilings," in *International Conference on Advanced Robotics*, Pisa, Italy, 1991.
- [2] S. Hirose, K. Yoneda, and H. Tsukagoshi, "Titan vii: Quadruped walking and manipulating robot on a steep slope," in *International Conference on Robotics and Automation*, Albuquerque, New Mexico, USA, 1997.
- [3] G. S. Shaoping Bai, K. H. Low and T. Zielinska, "A new free gait generation for quadrupeds based on primary/secondary gait," in *International Conference on Robotics and Automation*, 1999, pp. 1371-1376.
- [4] H. Kimura and Y. Fukuoka, "Adaptive dynamic walking of the quadruped on irregular terrain - autonomous adaptation using neural system model," in *International Conference on Robotics and Automation*, 2000, pp. 436-443.

[5] Y. H. H. Kimura, Y. Fukuoka and K. Takase, “Three-dimensional adaptive dynamic walking of a quadruped - rolling motion feedback to cpgs controlling pitching motion,” in *International Conference on Robotics and Automation*, 2002, pp. 2228–2233.

[6] Y. H. Y. Fukuoka, H. Kimura and K. Takase, “Adaptive dynamic walking of a quadruped robot ‘tekken’ on irregular terrain using a neural system model,” in *International Conference on Robotics and Automation*, 2003, pp. 2037–2042.

[7] I. Hiroshi and K. Masayoshi, “Adaptive gait for a quadruped robot on 3d path planning,” in *International Conference on Robotics and Automation*, 2003, pp. 2049–2054.

[8] C. L. S. Peng and G. Cole, “A biologically inspired four legged walking robot,” in *International Conference on Robotics and Automation*, 2003, pp. 2024–2030.

[9] Y. F. Z.G. Zhang and H. Kimura, “Adaptive running of a quadruped robot on irregular terrain based on biological concepts,” in *International Conference on Robotics and Automation*, 2003, pp. 2043–2048.

[10] E. Z. Moore and M. Buehler, “Stable stair climbing in a simple hexapod,” in *International Conference on Climbing and Walking Robots*, Karlsruhe, Germany, 2001, pp. 603–610.

[11] E. Z. Moore, D. Campbell, F. Grimmering, and M. Buehler, “Reliable stair climbing in the simple hexapod rhex,” in *International Conference on Robotics and Automation*, vol. 3, Washington, D.C., U.S.A, 2002, pp. 2222–2227.

[12] M. H. Raibert, *Legged Robots that Balance*. Cambridge, MA: MIT Press, 1986.

[13] I. Poulakakis, J. A. Smith, and M. Buehler, “Experimentally validated bounding models for scout ii quadrupedal robot,” in *International Conference on Robotics and Automation*, New Orleans, LA, USA, 2004, pp. 825–830.

[14] T. Bretl, S. Rock, J. Latombe, B. Kennedy, and H. Aghazarian, “Free-climbing with a multi-use robot,” in *9th Int. Symp. on Experimental Robotics*, Singapore, 2004.

[15] J. Morimoto and C. Atkeson, “Minimax differential dynamic programming: an application to robust biped walking,” in *Neural Information Processing Systems Conference*, 2002.

[16] J. Bagnell, S. Kakade, A. Ng, and J. Schneider, “Policy search by dynamic programming,” in *Neural Information Processing Systems Conference*, Vancouver, Canada, 2003.

[17] N. Kohl and P. Stone, “Machine learning for fast quadrupedal locomotion,” in *National Conference on Artificial Intelligence*, San Jose, California, USA, 2004.

[18] H. Kim, T. Kang, V. G. Loc, and H. R. Choi, “Gait planning of quadruped walking and climbing robot for locomotion in 3d environment,” in *International Conference on Robotics and Automation*, 2005.

[19] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami, “Planning biped navigation strategies in complex environments,” in *International Conference on Humanoid Robots*, 2003.

[20] R. Sutton, D. Precup, and S. Singh, “Intra-option learning about temporally abstract actions,” in *International Conference on Machine Learning*, 1998, pp. 556–564.

[21] M. Hauskrecht, N. Meuleau, C. Boutilier, L. P. Kaelbling, and T. Dean, “Hierarchical solution of markov decision processes using macroactions,” in *Conference on Uncertainty in Artificial Intelligence*, 1998.

[22] R. Parr and S. Russell, “Reinforcement learning with hierarchies of machines,” in *Neural Information Processing Systems Conference*, 1998.

[23] T. G. Dietterich, “Hierarchical reinforcement learning with the maxq value function decomposition,” *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 1999.

[24] A. Y. Ng and M. I. Jordan, “Pegasus: A policy search method for large mdps and pomdps,” in *Conference on Uncertainty in Artificial Intelligence*, 2000, pp. 406–415.

[25] J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd ed. Addison-Wesley Longman Publishing, 1989.

[26] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.

[27] C. D. Medio and G. Oriolo, “Robot obstacle avoidance using vortex fields,” in *Advances in Robot Kinematics*, S. Stifter and J. Lenarcic, Eds. Springer-Verlag, 1991, pp. 227–235.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 1998.

[29] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” in *International Conference on Robotics and Automation*, Detroit, Michigan, USA, 1999.

APPENDIX LOW-LEVEL PLANNER DETAILS

We describe the potential functions and the vortex field in detail here. Our coordinate system is defined with the robot’s direction of travel being the x direction. Our notation is defined below

t	is the time since the start of the task (of moving a single leg)
Ω_t	is the state vector at time t
i	is the index of the moving foot
u_t^j	is the position of foot j at time t
u_g	is the goal position for the moving foot
N_{obs}	is the number of obstacles in the environment. Right now, we only permit obstacles that are rectangular boxes. Note, the ground is also considered as an obstacle for the function definitions below.

$$U_t(\Omega_t, u_t^i, u_g) = U_t^g(u_t^i, u_g) + U_t^s(u_t^i, u_g) + U_t^c(\Omega_t) + U_t^o(\Omega_t) \quad (9)$$

where $U_t^g(\cdot)$, $U_t^s(\cdot)$, $U_t^c(\cdot)$, and $U_t^o(\cdot)$ represent goal, floor, CG, and orientation potential.

A. Goal Potential

If $C \leq \theta_2^{\text{center}}$ or $t \leq \theta_1^{\text{center}}$, then $U_t^g = 0$. Otherwise,

$$U_t^g = \theta_1^g \|u_t^i - u_g\|_2 - \theta_2^g e^{-\theta_3^g \|u_t^i - u_g\|_2} + \theta_4^g (u_{t,0}^i - u_{g,0})^2$$

B. Surface Potential

If $C \leq \theta_2^{\text{center}}$ or $t \leq \theta_1^{\text{center}}$, then $U_t^s = 0$. Otherwise,

$$U_t^s = (1 - e^{-\theta_3^s \|u_t^i - u_g\|_2^2}) \sum_{j=1}^{N_{\text{obs}}} \theta_1^s e^{-\theta_2^s d_j}$$

where d_j is the distance from the moving foot to obstacle j .

C. Posture Potential

The posture potential is defined as the sum of 3 other potentials

$$U_t^p = U_t^c + U_t^o + U_t^f$$

where U_t^c , U_t^o and U_t^f are the CG, orientation, and support feet constraint potentials respectively.

1) CG Potential:

If $C \geq \theta_3^{\text{center}}$ and $t > \theta_1^{\text{center}}$, then $U_t^c = 0$. Otherwise,

$$U_t^c = e^{-\theta_2^{\text{CG}} \min(C, \theta_1^{\text{CG}})}$$

Where C is a measure of how far the center of mass of the robot is inside the triangle formed by the 3 supporting legs. It is calculated by projecting the positions of the 3 supporting feet and the center of mass into the xy plane and measuring the minimum distance between projection of the center of mass to

the sides of the triangle formed by the projected feet positions. C is positive if the CG is within the triangle, and negative if it is outside the triangle.

2) *Orientation Potential:*

The orientation potential discourages large roll and yaw of the robot body and is defined as

$$U_t^o = \theta_1^o(1 - \cos(\phi_x)) + \theta_2^o(1 - \cos(\phi_z))$$

Where ϕ_x and ϕ_z are respectively the roll and yaw of the robot body.

3) *Constraint Potential:*

The constraint potential tries to keep the supporting feet at the same position as they were at the beginning of the task and is defined as

$$U_t^f = \theta_1^f \sum_{j=1, j \neq i} \|u_t^j - u_0^j\|_2^2$$

Although it may seem that the constraint potential is redundant since it accomplishes the same purpose as the nullspace projection described by 4, it is nonetheless necessary because of errors in the linearization used by the nullspace projection.

D. *Vortex Field*

Each obstacle induces a vortex field around it. The total vortex field is the sum of the vortex field of all the obstacles. At a point u , the direction of the vortex field induced by obstacle j is given by

$$\tilde{v}_j = \frac{\theta_1^v}{1 + \exp(\theta_2^v(\|u - n_j\|_2 - \theta_3^v))} \cdot \frac{\text{sign}(u_0 - u_{g,0})\vec{v}_j}{\|\vec{v}_j\|_2}$$

Where n_j is the nearest point on obstacle j to the point u , and \vec{v}_j is the vector cross product

$$\vec{v}_j = (u - n_j) \times \hat{y}$$

We note that \tilde{v} is given in terms of the 3d coordinates of the moving foot and we need to convert it into the 18d state space of the robot. This can be done simply by multiplying by the Jacobian made up of the partial derivatives of the components of the moving foot's position with respect to state variables.