Trot or Rot: Survival of the Fittest Hill-Climber Algorithm for Real Robots

Maksym Bondarenko
mb5018@columbia.edu

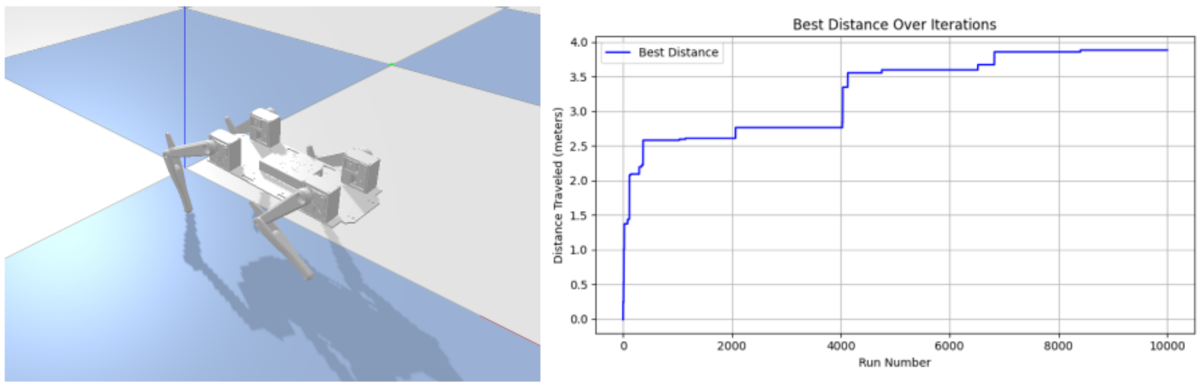


Figure 1. **Hill Climbing for Quadrupeds.** We show how a simple evolutionary algorithm can learn effective walking gaits for real quadruped robots using minimal computational resources.

Abstract

This report analyzes the use of a simple hill-climber algorithm for learning locomotion in custom quadruped robots, focusing on accessibility, basic performance optimization, and practical implementation. We show that even simple evolutionary methods can yield effective walking gaits in real robots, making them suitable as a "version 0" solution for robotics enthusiasts and beginners. We provide insights into hyperparameter tuning and guidance for practical use, supported by open-source code. This work aims to lower the barrier to entry for research and experimentation in resource-constrained, real-world robotics. Our code will be made publicly available at GitHub and later at a project website: `https://github.com/Lenguist/urdf-assembly`

1. Introduction

Achieving reliable locomotion in quadruped robots is challenging, especially for those without extensive computational resources or experience. While reinforcement learning and other advanced techniques exist, they can be complicated and resource-heavy. In contrast, a simple hill-climber algorithm provides a direct, beginner-friendly method to find basic walking gaits without large compute budgets or complex infrastructure.

Our primary contributions are:

1. We analyze the hill-climber algorithm's effectiveness in generating locomotion strategies for quadruped robots, focusing on what hobbyists and students can realistically achieve.
2. We provide a complete, open-source implementation, along with practical tips and parameter-tuning guidelines. This lowers the entry barrier, making it easier for newcomers to start experimenting.
3. We discuss future directions for improvements, including ways to further optimize and scale the approach.

By prioritizing simplicity and real-world feasibility, we hope to encourage more people to engage with robotics locomotion research, bridging the gap between theory and hands-on building.

2. Related Work

Hill climbing and evolutionary algorithms have emerged as powerful techniques for optimization across various domains, particularly in robotics and combinatorial problems. It is relatively simple, has existed for a long time, and is highly reliable.

In the domain of quadruped robotics, Kim et al. [3] demonstrated the potential of evolutionary computation for gait optimization. Their work introduced foot placement perturbation techniques, showing that evolutionary approaches can significantly improve locomotion strategies. Similarly, Jimenez's thesis [2] compared hill climbing with policy gradient algorithms for quadruped locomotion, revealing that hill climbing can outperform more complex approaches in certain scenarios.

Parallel computing research has substantially expanded the algorithm's applicability. Yelmewad et al. [5] developed a Parallel Iterative Hill Climbing (PIHC) algorithm for the Traveling Salesman Problem (TSP), achieving remarkable speedups of up to 279× on GPU architectures. O'Neil and Burtscher [4] further demonstrated the potential of GPU-based implementations, achieving up to 3× performance improvements over existing solvers.

Control selection studies have also validated the algorithm's effectiveness. Davies et al. [1] found that hill climbing can provide significantly improved search efficiency, particularly in high-dimensional control spaces. Zhang et al. [6] proposed a step-size adaptive local search algorithm that automatically adjusts mutation parameters, addressing one of hill climbing's inherent limitations.

Despite these advances, challenges remain. The algorithm's tendency to get trapped in local optima and its sensitivity to initial conditions are well-documented limitations. As noted in previous literature, hill climbing is inherently "greedy," focusing on immediate improvements without guaranteeing global optimization.

Our work builds upon these foundational studies by focusing specifically on quadruped robot locomotion. We contribute to the existing body of knowledge by:

- Demonstrating a practical application of a hill climber algorithm for learning quadruped gaits
- Providing an analysis of its performance in robotic locomotion on resource-constrained systems
- Offering an accessible, beginner-friendly approach to evolutionary robotics

3. Methodology

3.1. Algorithm Design and Implementation

We implement a hill climbing algorithm to solve qudruped locomotion problem by fitting a gait function. The algorithm outline follows.

Algorithm 1 Hill Climbing for Quadruped Robot

```
1: *Input parameters:*
2: $N_{\text{max}}$: Maximum iterations
3: $M$: Number of parallel candidates
4: $\epsilon$: Mutation scale factor
5: $\mathbf{P} = [\omega, \mathbf{a}, \mathbf{b}, \mathbf{c}]$: Gait parameters vector
6: $\mathbf{P}_{\text{best}} \leftarrow \mathcal{U}(\mathbf{P}_{\text{min}}, \mathbf{P}_{\text{max}})$ {Random initial parameters}
7: $d_{\text{best}} \leftarrow$ evaluate($\mathbf{P}_{\text{best}}$) {Distance walked}
8: **for** $i \leftarrow 1$ to $N_{\text{max}}/M$ **do**
9:     // *Mutation Phase*: $t_{\text{mut}}$
10:    **for** $j \leftarrow 1$ to $M$ **do**
11:        $\delta \sim \mathcal{U}(-\epsilon\mathbf{P}_{\text{best}}, \epsilon\mathbf{P}_{\text{best}})$
12:        $\mathbf{P}^j_{\text{new}} \leftarrow \mathbf{P}_{\text{best}} + \delta$
13:    **end for**
14:    // *Evaluation Phase*: $t_{\text{eval}}$
15:    $\mathbf{d}_{\text{new}} \leftarrow$ parallel_evaluate($[\mathbf{P}^1_{\text{new}}, ..., \mathbf{P}^M_{\text{new}}]$)
16:    // *Selection Phase*: $t_{\text{sel}}$
17:    **if** $\max(\mathbf{d}_{\text{new}}) > d_{\text{best}}$ **then**
18:        Find $k$ where $\mathbf{d}_{\text{new}}[k] = \max(\mathbf{d}_{\text{new}})$
19:        $\mathbf{P}_{\text{best}} \leftarrow \mathbf{P}^k_{\text{new}}$
20:        $d_{\text{best}} \leftarrow \mathbf{d}_{\text{new}}[k]$
21:    **end if**
22: **end for**
23: **return** $(\mathbf{P}_{\text{best}}, d_{\text{best}})$
```

Where:

- $\mathbf{P}$ represents gait parameters:
  - ω: Gait frequency (rad/s)
  - $\mathbf{a}$: Amplitude of sine wave
  - $\mathbf{b}$: Phase offset
  - $\mathbf{c}$: Vertical offset
- d measures forward distance traveled
- $t_{\text{mut}}, t_{\text{eval}}, t_{\text{sel}}$ track computation time
- $\mathcal{U}(a, b)$ denotes uniform random distribution
- M parallel candidates share total iterations N_{max}

The algorithm consists of three main components:

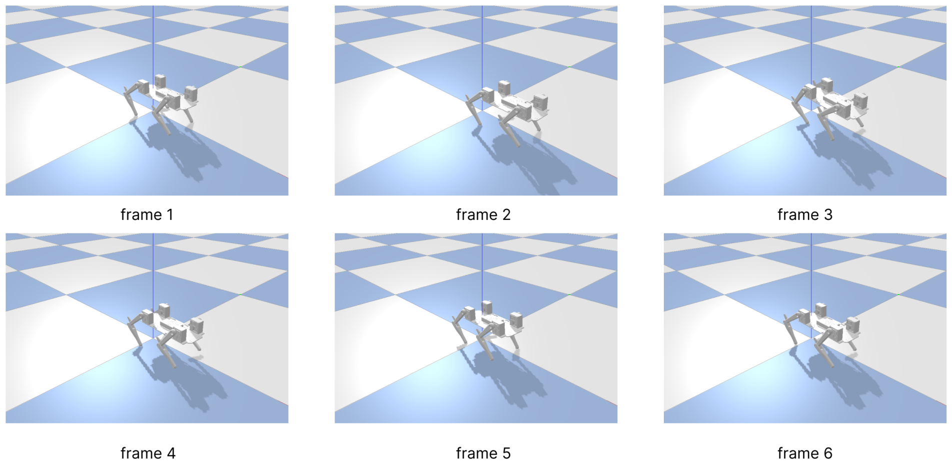
3.1.1. Mutation

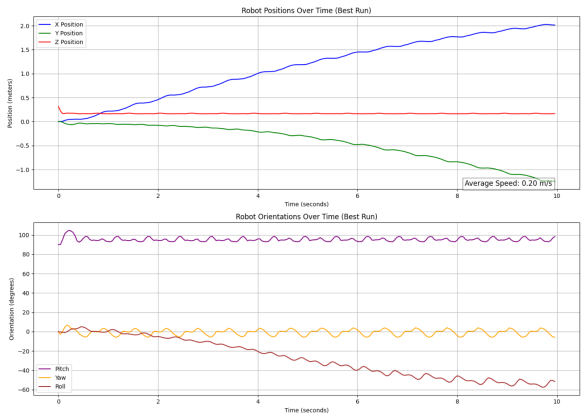
The mutation phase modifies gait parameters including:

- ω (frequency): Controls movement speed
- a, b, c (amplitudes): Define joint motion ranges
- Parameters are mutated with probability $mutation\_rate$

3.1.2. Evaluation

Each candidate gait is evaluated in PyBullet simulation with:





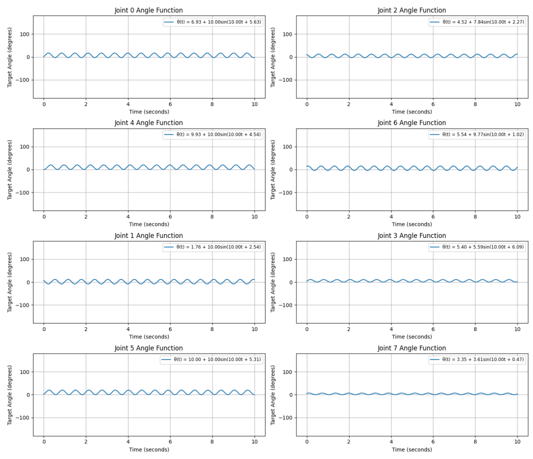


Figure 2. **Successful Run Visualization.** A representative sequence of robot snapshots showing the learned gait. The robot begins from rest and quickly converges to a stable walking pattern. (Figure 2)

- Fixed simulation duration (10 seconds)
- Settling time (0.5 seconds) before measurement
- Distance traveled as fitness metric

3.1.3. Selection

The selection process is greedy:

- Keep new parameters if they improve distance
- Otherwise, retain previous best parameters

3.2. Simulation Setup

We use PyBullet for the simulation. The robot is placed on flat ground and allowed to walk forward for a fixed time (e.g., 10 seconds) at 240 Hz simulation steps. After a short settling period, we measure forward displacement. This distance serves as the fitness metric.

4. Experiments and Results

4.1. Performance Over Iterations

Increasing the number of iterations generally improves the final gait. Although returns diminish after a certain point, investing more iterations often leads to smoother, more stable walking.

4.2. Number of Candidates Per Iteration

We also explored distributing the same number of total evaluations across fewer iterations with more candidates per iteration. While not implemented in the final version, we show a hypothetical evaluation script. This approach can parallelize evaluations, potentially speeding up the search without sacrificing performance.

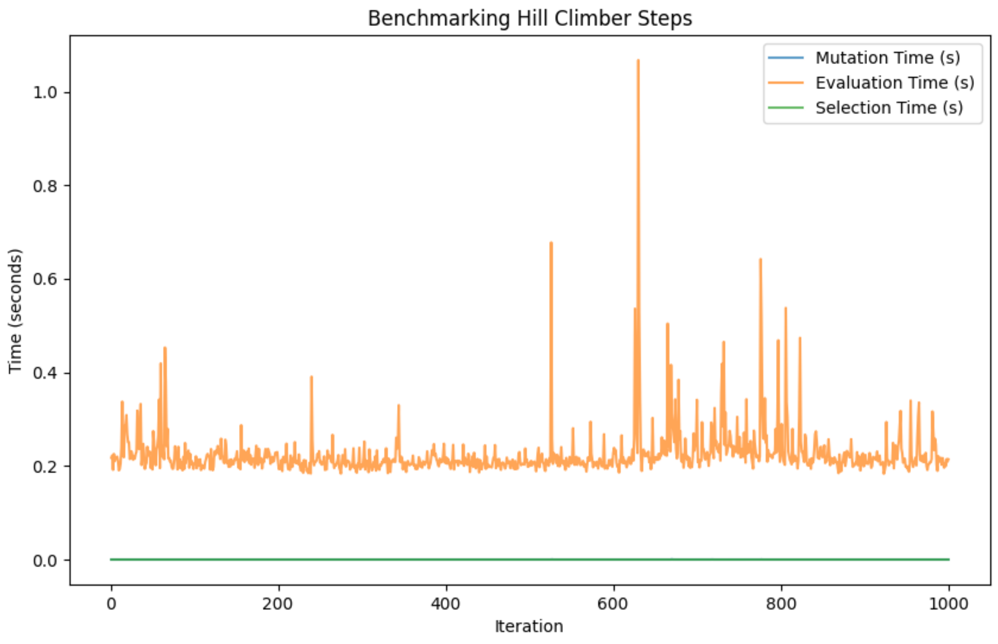


Figure 3. **Timing Breakdown.** Average time spent in mutation, evaluation, and selection phases over the course of optimization. Evaluation dominates overall runtime. (Figure 3)

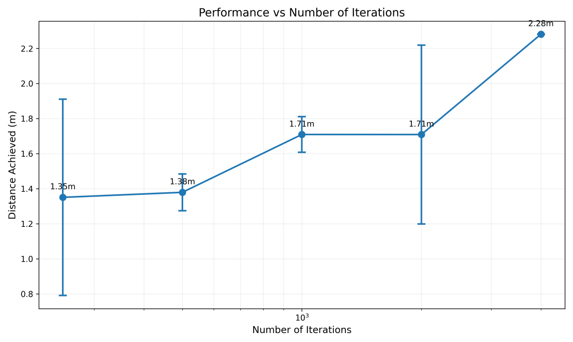


Figure 4. **Performance vs. Number of Iterations.** Distance achieved increases with more iterations, though we expect improvements slow down as we approach a plateau. More testing is required to observe this.

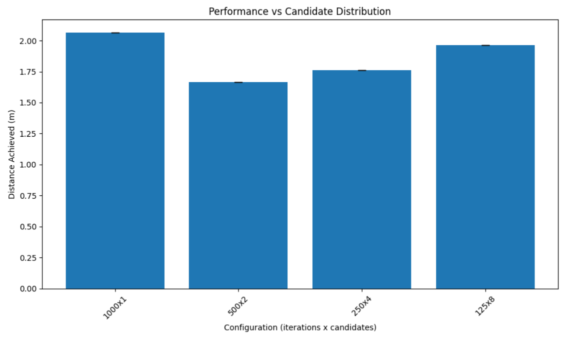


Figure 5. **Iterations vs. Candidates.** Comparison of configurations like 1000×1, 500×2, 250×4, and 125×8 (iterations × candidates). The figure shows that distributing evaluations differently can yield similar performance while potentially improving runtime due to parallelization potential.

4.3. Simulation Duration Comparison

We tested durations from 1.25 to 20 seconds and measured the resulting walking speed rather than raw distance. Short durations risk noisy evaluations, while long durations stabilize the measurement but cost more computation time. More testing is required to find the Pareto efficient duration, but we estimate it will likley be around 20s or more.

5. Discussion and Limitations

Our experiments confirm that a hill-climber algorithm can produce reasonable quadruped gaits with minimal complexity. However, this approach may struggle in highly complex environments or with non-flat terrain. The method is also prone to local optima. Future work could explore parallel evaluation strategies, hybrid methods that combine hill-climbing with more sophisticated optimization techniques,

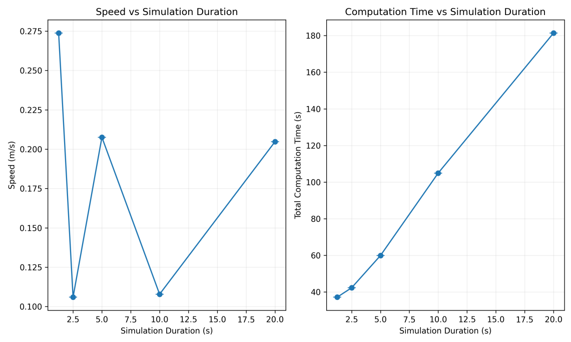


Figure 6. **Duration Comparison.** Average walking speed (m/s) and computation time as a function of simulation duration. While longer simulations improve stability of evaluation, they come at a higher computational cost.

or adaptive mutation rates.

6. Conclusions

We have shown that a simple hill-climber algorithm can learn stable quadruped gaits without requiring significant computational resources. This makes the approach accessible to hobbyists, students, and resource-constrained developers, serving as a practical "version 0" solution before considering more complex methods.

7. AI disclosure

We used ChatGPT and Claude for outlining, LaTeX formatting assistance, and suggestions. The main ideas, experiments, and analysis remain our own.

8. Acknowledgements

We thank Prof. Plancher for guidance and Justin Haddad for assistance in building the physical robot platform.

References

[1] Krispin A. Davies, Alejandro Ramirez-Serrano, Graeme N. Wilson, and Mahmoud Mustafa. Rapid control selection through hill-climbing methods. *Robotics and Autonomous Systems*, 2019. 2

[2] Antonio R Jimenez. *Policy search approaches to reinforcement learning for quadruped locomotion*. Master's thesis, Massachusetts Institute of Technology, 2004. 2

[3] Jihoon Kim, Dang Xuan Ba, Hoyeon Yeom, and Joonbum Bae. Gait optimization of a quadruped robot using evolutionary computation. *Journal of Bionic Engineering*, 18:306–318, 2021. 2

[4] Molly A. O'Neil and Martin Burtscher. Rethinking the parallelization of random-restart hill climbing: A case study in optimizing a 2-opt tsp solver for gpu execution. In *High Performance Computing Conference*, 2020. 2

[5] Pramod Yelmewad, Param Hanji, Amogha Udupa, Parth Shah, and Basavaraj Talawar. Parallel computing for iterative hill climbing algorithm to solve tsp. In *System Parallelization & Architecture Research Conference*. National Institute of Technology Karnataka, 2020. 2

[6] Wenfen Zhang, Yaohui Liu, and Hong Lv. A step-size adaptive hill-climbing algorithm for local search. *IEEE Transactions on Evolutionary Computation*, 2019. 2