Legged Robots Mini-Project1 Report

Xinran Wang, Hsu-Li Huang, Dacheng Zhou

October 2025

1 Introduction

In this mini project, we implemented and optimized a quadruped jumping controller to achieve forward, lateral, and twist jumps. In particular, we discussed the necessary conditions for stable landing of lateral jumps and provided a feasible solution for its implementation.

2 Assignment

2.1 Controller (Question Set 1)

2.1.1 Forward Jumping Parameter

Before jumping, we first need to decide where to place the quadruped's feet. For the following reasons, the feet should be positioned directly under each hip:
Quadruped robots place their nominal foot positions directly under the hips because it is the **most balanced and efficient stance**.

- **Statics:** Keeps the center of mass centered in the support polygon, ensuring stability.
- **Kinematics:** Legs are centered in their range of motion, providing maximum reach in all directions.
- **Torques:** Vertical alignment minimizes joint torques and energy use.
- **Control:** Simple, symmetric, and well-conditioned for inverse kinematics and gait control.

In short, this configuration represents the **neutral, energy-efficient, and stable default posture**.

When designing the robot for forward jumping, the setup differs from previous practicals. The leg is no longer modeled in 2D; instead, we consider a full 3D world frame. Each leg has an additional motor at the hip that controls the lateral (y-axis) offset of the leg. This offset influences the quadruped's roll stability. In general, a wider stance improves stability, but the optimal offset depends on the robot's mechanical design. We must ensure that stability is maintained without violating the robot's original geometry or compromising its mobility.

In the `simulation.py` file, the function `_build_hip_offsets` sets the hip offsets according to the A1 robot's URDF model. In our implementation, we use the default y-axis offset values provided by this function.

For forward jumping, we aim for a **low center of mass (CoM)**. A lower CoM provides a more stable stance (lower potential energy) and requires less torque to balance. It also allows a larger leg extension range, enabling the robot to generate more upward thrust and achieve a stronger jumping impulse.

In summary: A high center of mass reduces jumping power and makes the robot less stable, increasing the likelihood of tipping over during jumps. Therefore, maintaining a low CoM offers both improved stability and better jumping performance.

2.1.2 Controllers For Different Kinds of Jumping

Our omnidirectional jumping controller is fundamentally designed according to the motion principles of quadruped locomotion. The initial controller parameters are set around the mid-range values suggested in [1]. The specific **controller designs** and **influences of force profile parameters** for different types of jumping are summarized as follows:

- **Forward Jump:** For forward jumping, the objective is to generate a positive displacement along the x-axis. To achieve this, the lateral force component is constrained as $F_y = 0$ to avoid side disturbances and heading drift, which also prevents the accumulation of orientation errors over multiple jumps. Under this constraint, F_x and F_z are tuned jointly: F_x provides the forward driving impulse, while F_z ensures sufficient vertical lift for take-off. Increasing F_x enhances horizontal displacement but may lead to forward pitch or slipping, whereas increasing F_z improves jump height and airtime at the cost of higher impact forces on landing.

The stance-phase frequency f_0 also affects jump dynamics; higher f_0 shortens stance duration and increases take-off speed, while lower f_0 yields smoother but less energetic motion. With stabilization strategies such as Virtual Model Control (VMC) and gravity compensation, and by applying identical force profiles to all four legs, the quadruped achieves a smooth and stable forward jump.

- **Lateral Jump (Left/Right):** For lateral jumping, the controller applies an upward vertical force F_z (250–450 N) for lift-off and a lateral thrust F_y (30–150 N) to generate sideward motion. A positive F_y drives the robot to the right, and a negative F_y to the left. Too small a vertical force results in insufficient lift, reducing observable lateral displacement. A small empirical forward force $F_x = -18$ N is added to counteract the forward-leaning tendency induced by lateral motion, ensuring postural stability during take-off and landing.

  Parameter sweeps show that increasing F_y improves lateral displacement up to the friction limit, beyond which foot slippage occurs. Higher F_z values yield greater airtime and stability at take-off but increase impact energy during landing. The stance frequency f_0 also influences the motion: higher f_0 produces faster, sharper hops, while lower f_0 results in smoother but less pronounced lateral movement.

- **Twist Jump (Clockwise/Counterclockwise):** For the twist jump, a yaw torque is generated by applying opposite lateral forces on the fore and hind legs, with $F_x = 0$. The front legs apply a positive F_y and the hind legs a negative F_y for clockwise twist, forming a lateral force couple that produces rotation about the vertical axis.

  Parameter sweeps show that moderate F_y $(60 - 100$ N) achieves stable rotation, while smaller values yield insufficient torque and larger values cause body oscillation and unstable landings. The vertical force F_z $(250 - 350$ N) controls airtime for rotation—too small limits turning, too large leads to unstable landing. The stance frequency f_0 affects torque timing: moderate values $(1.1 - 1.4$ Hz) enable smooth rotation, whereas excessively low or high frequencies reduce stability.

2.1.3 Virtual Model Control

Virtual Model Control [2] is a motion control language that generates forces by simulating virtual mechanical elements, and the forces are applied as **real joint torques** to the robot. It should be noted that the virtual force added to the controller of the robot is **opposite in direction** to the force exerted by the imagined spring. To be specific, if a corner is higher than the neutral plane, the force of the virtual spring should be along the direction of $-z$, which means that we should reduce the supporting force from the ground (along the $+z$ direction) to achieve it. As a result, based on Newton's third law, the value of the force exerted by the robot's foot on the ground (along the $-z$ direction) should accordingly decrease. In other words, the change of the force exerted by the robot is along the $+z$ direction, so we need to add an positive virtual force (along the $+z$ direction) to the desired force of the robot.

In our simulation, the effect of virtual model control is obvious. The quadruped robot can return to a steady state quickly after lands on the ground. Although the robot's base exhibits a considerable tilt during the flight phase, it rapidly returns to a stable posture after landing, thus having minimal impact on the next jump. In comparison, without VMC, the robot does not recover as effectively, leading to accumulated instability over multiple jumps.

In this simulator, a larger gain of VMC generally leads to better performance, as higher gains enable the robot to restore stability more rapidly. However, in the real world, increasing the gain indefinitely does not necessarily improve control performance. This is mainly due to the following two factors:

- **Motor response:** Real motors do not respond instantaneously to pulse commands; instead, their output exhibits oscillatory behavior. When excessively large VMC gains are applied, these oscillations can be significant, potentially destabilizing the overall control of the robot.

- **Motor resonance:** Real motors are not perfectly linear systems and tend to exhibit strong nonlinearities near their resonance frequencies. To achieve optimal control performance, it is necessary to avoid operating near these resonance points. Consequently, the optimal gain of VMC is influenced by the motor's resonance characteristics.

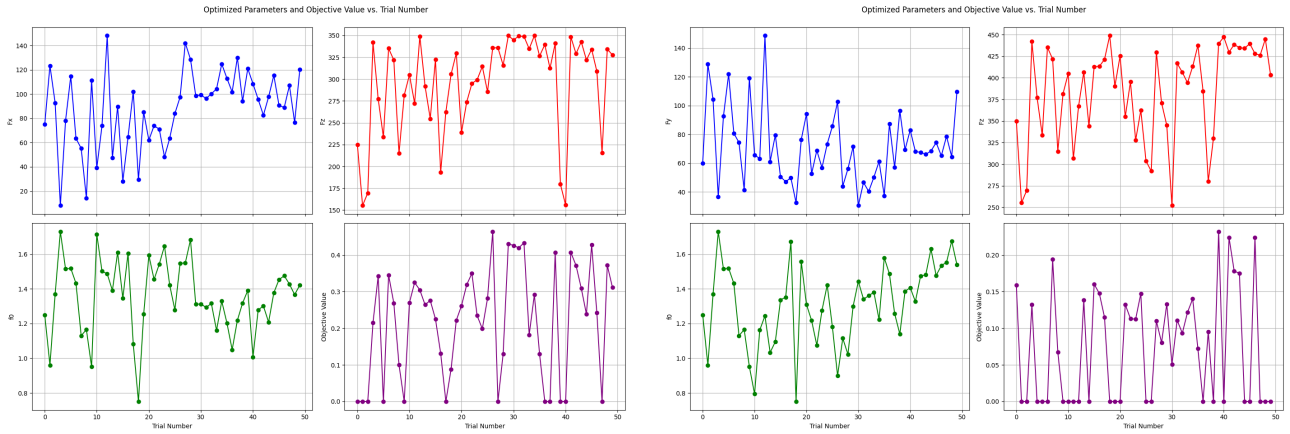
To maintain the physical realism of our simulation, an excessively large k_{VMC} was avoided. We use $K_{\text{VMC}} = 200$ for jumping tasks, which is big enough to keep the balance of the robot in most of scenarios.

2.2 Optimization (Question Set 2)

2.2.1 Optimization for the Furthest Jump

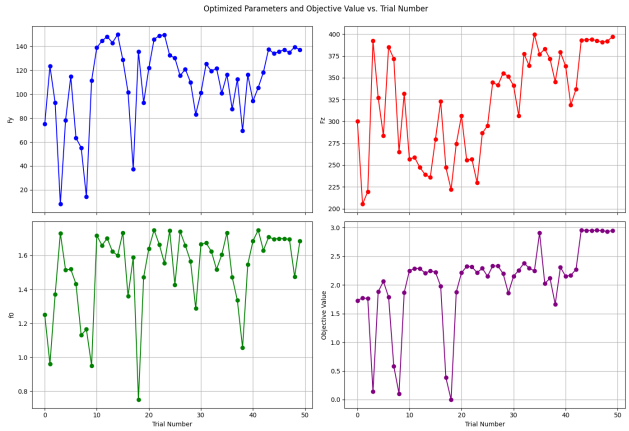
We optimized the elements in force profile to get the furthest jumping distance. To get a better result for continuous jump, we use the average jumping displacement of the first n jumps as the objective function. To

balance the stability of the optimization results and the efficiency, we set $n = 5$. The Bayesian Optimzer used in this project is Optuna [3].

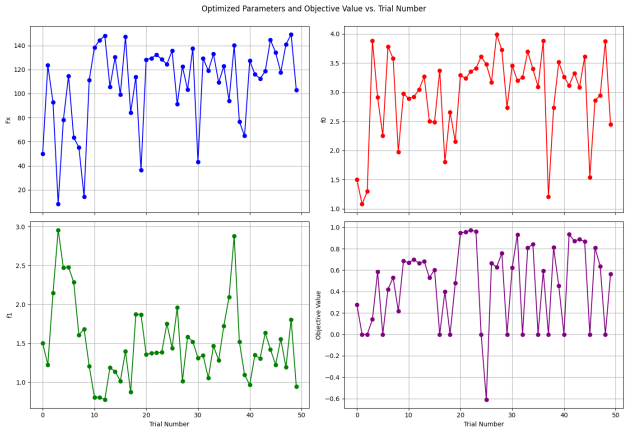


(a) Furthest Forward Jump (F_z Range: 150-350)

(b) Furthest Lateral Jump



(c) Largest Twist Jump



(d) Fastest Forward Hopping

Figure 1: Parameters and objective value in optimization process for different jumping motions.

- **Forward Jump:** The optimization variables for forward jump are f_0, F_x, F_z, and the cost function is defined as the average displacement in x direction of the first n jump.

$$J_\mathrm{fwd} = \frac{x_\mathrm{n} - x_0}{n} \tag{1}$$

  Fig. 1(a) shows the optimization process of the furthest forward jump, and the best variable values are $f_0 = 1.55s^{-1}$, $F_\mathrm{x} = 97.6N$, $F_\mathrm{z} = 335.7N$, and the largest average forward jumping distance is $x = 0.46m$.

- **Lateral Jump:** The optimization variables for lateral jump are f_0, F_y, F_z, and the cost function is defined as the average displacement in y direction of the first n jump .

$$J_\mathrm{lat} = (\pm)\frac{y_\mathrm{n} - y_0}{n} \tag{2}$$

  The sign is (+) for jumping left, or (-) for jumping right.

  Fig. 1(b) shows the optimization process of the furthest lateral jump, and the best variable values are $f_0 = 1.47s^{-1}$, $F_\mathrm{y} = 78.0N$, $F_\mathrm{z} = 428.2N$, and the largest average lateral jumping distance is $y = 0.30m$.

- **Twist Jump:** The optimization variables for twist jump are f_0, F_y, F_z, and the cost function is defined as the average rotation angle around yaw axis of the first n jump .

$$J_\mathrm{twist} = (\pm)\frac{\psi_\mathrm{n} - \psi_0}{n} \tag{3}$$

  The sign is (+) for jumping counterclockwise, or (-) for jumping clockwise.

  Fig. 1(c) shows the optimization process of the largest twist jump, and the best variable values are $f_0 = 1.71s^{-1}$, $F_\mathrm{y} = 127.7N$, $F_\mathrm{z} = 392.8N$, and the largest average yaw angle is $\psi = 2.96rad$.

2.2.2 Optimal Solution for Stable Jumping & Falling Avoidance

To ensure stable jumping and prevent the quadruped from falling, several measures were incorporated into the optimization process.

First, during optimization, it is crucial to determine whether the robot has fallen. If a fall is detected, the objective value is set to zero. This allows the Bayesian optimization to automatically adjust the parameters to avoid configurations that lead to instability or falling.

However, identifying a fall after just one jump is challenging. Even if the robot lands successfully on all four legs, it may still lose balance and fall after a few seconds due to unstable posture recovery. To address this, we introduced **consecutive jumping tests**. In each trial, the quadruped performs five continuous jumps. If the robot falls at any point within these five jumps, it cannot complete the sequence, and the objective value is again set to zero. This ensures that only parameter sets that produce consistently stable jumps are rewarded by the optimizer.

A second key measure is the **penalty function** based on the robot's roll and pitch angles. During each jump, the system continuously monitors the roll and pitch of the base frame. If the roll exceeds 80° or the pitch exceeds 90°, the objective value is set to zero. This indicator function helps the optimizer reject solutions that result in excessive body inclination, which typically indicates instability or imminent falling.

The threshold values were chosen based on both **biological inspiration and physical constraints**. Studies on biological and robotic frogs suggest that an optimal forward jumping motion has a take-off angle of around 60°. Therefore, setting the falling pitch threshold to 90° provides a clear distinction between normal jumping dynamics and an unrecoverable posture. Similarly, the roll threshold of 80° accounts for lateral instability, as beyond this angle the robot's physical structure can no longer maintain balance.

Through these measures—fall detection, consecutive jump testing, and angle-based penalties—the optimization process is guided toward parameters that yield stable, repeatable jumping behaviors without falling.

2.2.3 Fastest Continuous Hopping Controller

To get the fastest forward continuous jump, we use the velocity in x direction of the robot as the objective function.

$$J_{\text{fast}} = \frac{x_{\text{n}} - x_0}{n}(\frac{0.5}{f_0} + \frac{0.5}{f_1})^{-1} \tag{4}$$

In the design of the fastest forward hopping controller, four parameters are considered potentially influential: the forward thrust F_x, the vertical lift force F_z, and the stance and flight frequencies f_0 and f_1. However, optimizing all four parameters simultaneously makes convergence difficult within a limited budget of 50 trials. Empirical tests show that fixing f_1 often leads to suboptimal solutions biased toward non-aggressive hopping behaviors. In contrast, setting F_z to a relatively large constant value yields a more consistent optimization landscape and facilitates the emergence of faster locomotion policies. Moreover, the joint tuning of f_0 and f_1 provides a more reasonable coordination between the stance and flight phases. Therefore, the optimization focuses on the parameter combination $\{F_x, f_0, f_1\}$ to balance convergence efficiency and forward velocity performance.

Fig. 1(d) shows the optimization process of the fastest forward jump, and the best variable values are $f_0 = 2.44s^{-1}$, $f_1 = 0.94s^{-1}$, $F_{\text{x}} = 132.4N$, and the largest forward jump velocity is $v = 0.97m \cdot s^{-1}$.

2.2.4 Sim-to-Real Transfer

- **Pre-Deployment Setting:** Before transferring the controller from simulation to hardware, several precautions must be taken to ensure safe and reliable operation. All torque, velocity, acceleration, and joint position limits should be strictly enforced to prevent mechanical or thermal overload. Precise time synchronization among sensors such as IMU, encoders, and force sensors is essential to maintain consistent state estimation across the control loop. Safety mechanisms including command saturation, output clipping, and an emergency stop should be implemented to immediately halt operation under abnormal conditions. Furthermore, the simulation environment must be carefully aligned with the real-world setup by matching key parameters such as ground friction, terrain compliance, and control loop frequency.

- **Sim-to-Real Training**: Domain randomization is then applied to physical parameters such as terrain properties, friction coefficients, and joint damping, enabling the policy to generalize across varying environments and learn robust control behaviors. Realistic physical constraints, including actuator torque limits and thermal capacity, are incorporated during training. Furthermore, stability-related terms are added to the objective function to encourage smooth and dynamically consistent behaviors when the policy is transferred to real hardware.

- **Optimization on Hardware**: The hardware optimization process begins with a warm-start from the best policies/parameters obtained in simulation under different random seeds and terrain adaptations. Before each trial, the quadruped starts from a stable and well-controlled standing posture to ensure consistent

initial conditions. A safe and constrained optimization strategy is employed, where the parameter updates between iterations are restricted within a trust region to prevent abrupt changes that could lead to loss of balance. Each parameter set is executed for a small number of repeated trials to obtain a reliable estimate of the objective value with low measurement noise. Safety constraints are enforced throughout the process, including joint torque, body orientation, foot slip ratio, and motor temperature limits. The optimization objective combines performance and stability, and a Safe BO scheme is used to iteratively propose new parameters within safe bounds, allowing gradual improvement while preserving hardware integrity.

2.2.5 Extra Challenges for Designing Walking Controller

Designing a walking controller for the Unitree A1 introduces several additional challenges compared to the hopping controller. While hopping involves symmetric leg motion and a simple alternation between flight and full-ground contact phases, walking requires the coordination of asynchronous leg movements across multiple gait phases. This leads to a more complex contact schedule, where the robot must handle varying combinations of stance and swing legs while maintaining stability. The Zero Moment Point (ZMP) and Center of Pressure (CoP) also change continuously as the support polygon shifts during walking, requiring dynamic balance control and real-time adjustment of body posture. Furthermore, each leg must follow a carefully planned swing trajectory to ensure smooth motion, proper ground clearance, and stable foot placement. Walking also introduces the need for compliant control to manage uneven terrain and external disturbances, as different legs experience different ground reaction forces. In addition, achieving energy-efficient motion and enabling smooth transitions between gaits (such as walking, trotting, and running) demand more advanced control and optimization strategies. Finally, walking control relies more heavily on accurate state estimation from onboard sensors, since continuous feedback on contact states, body orientation, and slippage is critical for maintaining balance and trajectory tracking. Overall, walking control presents a significantly higher level of complexity than hopping, as it requires precise multi-contact coordination, dynamic stability management, and adaptive motion planning in real time.

3 Lateral Jump

3.0.1 Footwork for landing

For lateral jumping, the main challenge lies in the landing phase. During a typical landing, the robot operates in the f_1 phase of the force profile, where the additional foot force is zero and the feet remain at their nominal positions. For forward and twisting jumps, this configuration is sufficient because the Zero Moment Point (ZMP) remains within the support polygon defined by the nominal foot positions. However, during a lateral jump, maintaining nominal foot positions causes the ZMP to shift outside the support polygon, as illustrated in Figure 2.

This instability arises from two factors. First, when landing laterally, the quadruped tilts to one side, and the gravitational force—always perpendicular to the ground—pulls the ZMP toward the landing side. Second, the robot's lateral acceleration in reference frame during the jump generates an inertial force along the y-axis, further pushing the ZMP in the same direction. Together, these effects move the ZMP beyond the lower edge of the support polygon, leading to a loss of balance and eventual falling.

To counteract this, the robot must actively extend its legs outward along the y-axis using the hip actuators. This leg extension enlarges the support polygon and keeps the ZMP within its boundaries during landing. As shown in Figure 2, once the legs are stretched laterally, the ZMP remains inside the support polygon, ensuring a stable landing.

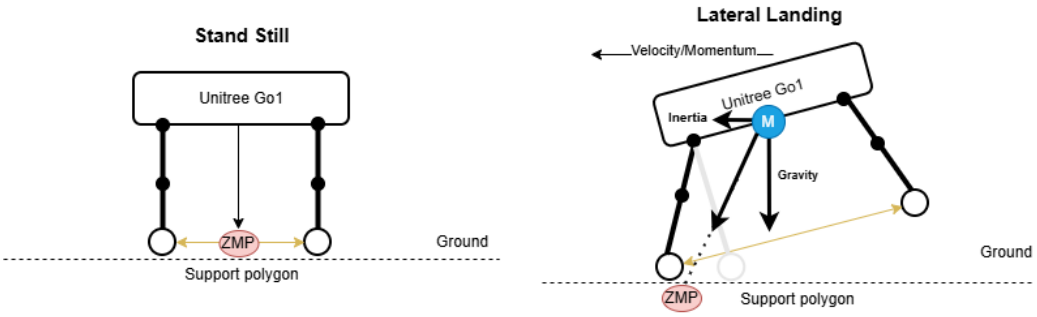


Figure 2: Dynamics of lateral jumping.

A Necessary Code Explanation For Lateral Jump

This part mainly contain the code explanation, which is typically not included in academic paper, thus categorizing in Appendix.

```
foot_contacts = simulator.get_foot_contacts()  # contact state for all 4 feet
if all(foot_contacts):
    airborne = 0
if not any(foot_contacts):
    airborne = 1
if airborne == 1:
    tau += nominal_position_fly(simulator, Kp, Kd, Kd_point)
elif airborne == 0:
    tau += nominal_position_ground(simulator, Kp, Kd, Kd_point)
```

To achieve reliable performance with a simple implementation, a Boolean variable `airborne` is introduced. Its initial value is set to zero. When no leg is in contact with the ground, `airborne` is set to 1, indicating that the robot is in flight. In this state, the nominal leg positions are adjusted with an extended offset along the y-axis to improve landing stability. Once all four legs have made ground contact again, `airborne` is reset to 0, restoring the nominal standing configuration. The stretching factor along the y-axis is set to 1.5, corresponding to a 50% increase over the nominal lateral offset.

B Generative AI Declaration

B.1 Which aspects of the project did you use generative AI for?

We used generative AI tools primarily to clarify basic control concepts mentioned in the course materials (e.g. traditional control approaches referenced in the lectures), to assist in implementing non-core utility functions such as visualization scripts, and to refine the academic writing style of the report. In general, I use AI when learning new theoretical concepts, implementing routine or tool-level code that does not involve deep reasoning, or proofreading written text. However, I deliberately avoid using AI for the core components related to robotics control and learning algorithms, which I completed independently.

B.2 In which aspects was generative AI the most useful and least useful?

AI was most useful for understanding completely new concepts at the beginning of the project. For example, "Can you explain what different quadruped gaits correspond to in intuitive examples?", the response provided clear explanations and examples that helped me build an intuitive understanding before diving into equations and implementation. AI also suggested related academic papers for further reading, which accelerated my conceptual learning.

B.3 Has the use of AI made the project smoother to complete?

Yes. By automating repetitive tasks and assisting with non-technical components such as visualization and documentation, AI significantly reduced the time spent on peripheral work, allowing me to focus more on the analytical and experimental aspects of the project.

B.4 Has the use of AI impacted your ability to learn and understand the concepts behind this project?

Yes, positively. AI helped me quickly grasp new concepts by offering clear explanations and guiding me toward key references. This allowed me to form an initial understanding faster and to deepen it later through experimentation and reading original papers.

References

[1] G. Bellegarda, M. Shafiee, M. E. Özberk, and A. Ijspeert, "Quadruped-frog: Rapid online optimization of continuous quadruped jumping," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1443–1450, 2024.

[2] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *The International Journal of Robotics Research*, vol. 20, no. 2, pp. 129–143, 2001.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, ACM, 2019.