PROJECT REPORT

# DESIGN AND OPTIMIZATION OF A QUADRUPED JUMPING CONTROLLER

GUILLAUME DELAMARE, TUAN LINH PHAN, PAUL BOURGOIS

LEGGED ROBOTS
EPFL

21ST OCTOBER 2025

# Contents

## 1.1 Controller Implementation

### 1.1.1 Cartesian impedance and joint damping

Each leg tracks a neutral foot position $p_i^{\text{ref}}$ set by `set_neutral_position()` defined in the `ControllerParameters` class. The neutral position is initialized from the simulator variable `hip_offsets_` with added offsets for experimentation.

We tested multiple neutral stances: laterally wide, forward-leaning, backward-leaning, and rear-biased (rear legs positioned backward by $x_{\text{offset}} = -0.07\,\text{m}$ while the front legs remain under the hip). These tests were motivated by early instability, particularly back-flip tendencies during jumping without VMC. The rear-biased stance increased pitch stability at the cost of some horizontal thrust. However, after incorporating and tuning the VMC, this stance became unnecessary. We retained a laterally wide stance to mitigate sideways toppling during side jumps.

With $h_{\text{des}} = 0.22\,\text{m}$ and lateral offsets $\pm 0.1\,\text{m}$ (depending on leg index), a Cartesian PD law

$$f_i = K_p(p_i^{\text{ref}} - p_i) + K_d(\dot{p}_i^{\text{ref}} - \dot{p}_i), \qquad \tau_i^{\text{cart}} = J_i^\top f_i,$$

provides the main stiffness. As the starting point, we use $K_p = \text{diag}(1000, 1000, 1000)$ and $K_d = \text{diag}(30, 30, 30)$. An additional joint-space damping term $\tau_i^{\text{joint}} = K_{d,j}\,\dot{q}_i$ with $K_{d,j} = \text{diag}(0.5, 0.5, 0.5)$ further smooths the impact response. This combination maintains stance stability and returns the legs to their nominal positions after landing.

### 1.1.2 Gravity compensation

Gravity torques are computed as

$$\tau_i^{\text{grav}} = J_i^\top F_i^{\text{grav}}, \qquad F_i^{\text{grav}} = [0, 0, -mg/N_{\text{stance}}]^\top,$$

where $m$ is total robot mass and $N_{\text{stance}}$ is the number of feet in contact. We apply gravity compensation only when all four feet are in contact, using `simulator.get_foot_contacts().all()`. This is an approximation assuming the center of gravity lies at the intersection of diagonals formed by the four leg contact points. Practically, this is inaccurate; however, the VMC effectively compensates for this estimation error by adjusting vertical support forces dynamically.

### 1.1.3 Impulse (force profile) controller

The jumping impulse $F_i(t) = (F_x, F_y, F_z)$ is generated by a time-parameterized `FootForceProfile`. Three modes are defined:

`FORWARD:` $(-500, 0, -800)\,\text{N}$, `SIDE:` $(80, 80, -400)\,\text{N}$, `TWIST:` $(0, 150, -300)\,\text{N}$.

These values serve as initial guesses to define reasonable ranges for later optimization. Each impulse consists of an active phase ($f_0 = 3.0\,\text{Hz}$) followed by an idle phase ($f_1 = 0.2\,\text{Hz}$). The idle phase lasts about five seconds, including the robot's flight time and balance recovery after landing.

Initially, to counter back-flip tendencies, we scaled down the front-leg force by $0.8\times$. However, after tuning VMC and the control gains ($K_p$, $K_d$) and optimizing the force-generation frequency $f_0$ and profile amplitude, this scaling was no longer required. In `SIDE` mode, $y$-forces are applied opposite to the desired motion direction, while `TWIST` applies opposite $y$-forces between front and rear legs to induce yaw rotation.

### 1.1.4 Virtual Model Control (VMC)

The formula was provided in the intruction, we start with $k_{\text{vmc}} = 1500\,\text{N/m}$.

## 1.2 Q1: Nominal foot position and CoM height

During initial tests, the robot tended to perform back-flips when using a neutral stance and no VMC. Introducing a rear-biased stance ($x_{\text{offset}} = -0.07\,\text{m}$) reduced this tendency, though it slightly reduced forward distance. After incorporating VMC, the rear bias was no longer needed, confirming that VMC stabilizes pitch effectively. We observed that lowering $h_{\text{des}}$ to 0.18 m yielded safer but lower jumps, while raising it to 0.25 m increased apex height but caused minor landing oscillations. The default $h_{\text{des}} = 0.22\,\text{m}$ offered the best balance.

## 1.3 Q2: Forward, lateral, and twist jumping performance

**Forward jumps:** The robot achieves consistent forward thrust with $(F_x, F_y, F_z) = (-500, 0, -800)\,\text{N}$. After adding VMC and correcting gravity compensation, front-leg force scaling was no longer necessary, and the robot landed with minimal pitch oscillation.

**Lateral jumps:** In `SIDE` mode, all stance legs apply the same lateral impulse $(F_x, F_y, F_z) = (0, \pm 80, -400)\,\text{N}$, where the sign of $F_y$ sets the direction: $F_y < 0$ for a left jump and $F_y > 0$ for a right jump. A laterally wide stance improved stability and reduced roll oscillations. We are a lot more conservative in both vertical leap and side force

**Twist jumps:** In `TWIST` mode, opposite $y$-forces between front and rear legs produced smooth in-place yaw rotations. It was very without much tuning.

## 1.4 Q3: Effectiveness of VMC

VMC substantially improved the consistency and robustness of both standing and jumping behaviors. Without VMC, the robot could still perform individual jumps, but stability depended strongly on precise stance geometry and careful manual tuning of the force profile parameters. This sensitivity arose mainly from two factors: (1) the simplified gravity compensation, which assumes the center of gravity lies at the intersection of diagonal contact points and therefore does not perfectly balance vertical loads, and (2) the open-loop impulse profile, which can introduce small asymmetries in takeoff forces across the legs. As a result, the robot without VMC often exhibited slight body tilting or slow reorientation after landing and required more time to regain a flat, steady posture. Achieving consistent takeoff direction also demanded a carefully tuned neutral stance, limiting the generality of the controller across different configurations.

With VMC enabled, the controller dynamically redistributed vertical support forces according to the robot's orientation, effectively compensating for gravity approximation errors and small force asymmetries. The robot remained noticeably flatter during takeoff and landing, recovered to its nominal pose faster, and tolerated greater variation in both stance and jump parameters without retuning. Overall, VMC reduced the controller's dependency on manual fine-tuning and allowed a single set of gains and stance configuration to produce stable jumps in forward, lateral, and twist modes. It therefore acts as a stabilizing layer that enhances robustness and generalization across different motion directions and stance geometries.

## 2.1 Q1: Jump Optimization (Forward, Lateral, and Twist)

To optimize for the furthest jumps, we optimized single-jump performance. To do so, we simulate a single jump of the robot and its landing by adding extra simulation time on top of the jump duration. This allows us to see if the robot is statically stable after jumping.

To reflect the issue of stability, the objective function $J$ in each direction was devised as follows:

$J =$ jumping performance $\times$ stability metric.

The jumping performance was measured as functions of the distance jumped (translational for forward/lateral jumping or angular for twist jumping), and the stability metric was defined as an indicator function equal to 1 if all four feet are in contact with the ground at the end of the trial, or 0 otherwise, as was done in (1).

For **forward jumping** the objective function is defined in world coordinates as

$$J_{\text{fwd}} = (x_{\text{final}} - x_{\text{init}}) \times (1 - \frac{|\psi_{\text{final}} - \psi_{\text{init}}|}{\pi}) \cdot 1_{\text{contact}}.$$

where $\psi$ is the yaw of the robot body in the world frame. This function aims to maximize the forwards distance jumped, while penalizing runs that result in a twisted landing positions. The optimization variables are the force profile parameters $f_0$ (frequency determining the duration of the impulse), $F_x$ and $F_z$ (the maximal forces applied in the $x$ and $z$ directions during the impulse). We fixed $f_1 = 0.5\,\text{Hz}$ and $F_y = 0\,\text{N}$.

For **lateral jumping** the objective function is defined as

$$J_{\text{lat}} = (\pm)\frac{y_{\text{final}} - y_{\text{init}}}{x_{\text{final}} - x_{\text{init}}} \cdot 1_{\text{contact}}.$$

to maximize the lateral distance jumped, while penalizing runs that jump more forwards than they do laterally. The optimization variables are $f_0$, $F_y$, and $F_z$ . We fixed $f_1 = 0.5\,\text{Hz}$ and $F_x = 0\,\text{N}$.

Finally for **twist jumping**, the objective function is

$$J_{\text{twist}} = (\pm)\frac{\text{accumulated yaw}}{(x_{\text{final}} - x_{\text{init}})^2 + (y_{\text{final}} - y_{\text{init}})^2} \cdot 1_{\text{contact}},$$

where the accumulated yaw is directly proportional to the integral of the magnitude of these quantities. This allows us to optimize for the total yaw of the jump, while the denominator penalizes runs that stray far from the original jumping position in the plane. The optimization variables are $f_0$, $F_y$, and $F_z$, with $f_1 = 0.5\,\text{Hz}$ and $F_x = 0\,\text{N}$.

| Parameter | Value | Unit |
|---|---|---|
| Nominal foot position | $[-0.05, (-1)^{\text{leg\_id}+1} \times 0.1, -0.2]$ | m |
| $f_1$ | 0.5 | Hz |

Table 1: Fixed parameter values (nominal positions in leg frame).

| Optimization Variable | Range | Unit | Forward | Lateral | Twist |
|---|---|---|---|---|---|
| $f_0$ | [1.4, 4.5] | Hz | 4.387 | 1.618 | 2.366 |
| $F_x$ | [0, 300] | N | 289.24 | – | – |
| $F_y$ | [0,300] | N | – | 125.98 | 276.37 |
| $F_z$ | [200,700] | N | 497.62 | 212.59 | 355.09 |
| $k_{\text{VMC}}$ | [500, 2000] | N/m | 500.35 | 1285.52 | 1885.32 |
| **Furthest single jump** | **–** | **–** | **1.16 [m]** | **0.11 [m]** | **137.44 [deg]** |

Table 2: Optimization variables and best-performing values among 5 optimization runs.

## 2.2 Q2: Stability Measures

For the optimal solutions to avoid falling, the first step was to act on the nominal position of the feet. The chosen configuration results in a larger support polygon for the robot than with the feet directly under the hips, with feet spread wider along the $y$ direction.

Additionally, the optimization objective includes the stability metric to automatically disqualify unstable runs. This metric works for simple jumps (where static equilibrium is reached between jumps). For dynamic hopping, the metric must evolve to accommodate continuous motion, for example by simulating multiple consecutive jumps so that unstable controllers are filtered out naturally.

## 2.3 Q3: Optimizing continuous forwards hopping

To optimize for the fastest continuous hopping controller, we optimized multi-jump performance. To do so, we simulate multiple successive jumps of the robot (we chose $n_{jumps} = 10$). By increasing the number of simulated jumps, we select only the ones that result in stable gaits, and eliminate those that fall over after a few jumps. Moreover, we must only select the gaits that result in the robot hopping in a straight line along the $x$-direction, to mitigate long-term drift when the controller is applied for long periods of time.

In this case, the objective function is modified from the previous forwards optimization as follows, to maximize average $x$-direction speed while penalizing drifts in yaw:

$$J_{contfwd} = \frac{x_{final} - x_{init}}{t_{final} - t_{init}} \times (\text{accumulated yaw})^{-1} \cdot 1_{\text{contact}},$$

The optimization variables are the force profile parameters $f_0$ (frequency determining the duration of the impulse), $f_1$ (frequency determining the duration between impulses), $F_x$ and $F_z$. We fixed $F_y$ = 0 N by using the system's symetries.

The result of the optimization was an average forwards speed of 0.37 $m.s^{-1}$, with a sideways ($y$-direction) average velocity of only 0.001 $m.s^{-1}$, resulting in a drift of only 0.48 $deg$ in yaw after 20 jumps.

## 2.4 Q4: From Simulation to Hardware

Transitioning from simulation to real hardware introduces several practical challenges:

| Optimization Variable | Range | Unit | Best performing value |
|---|---|---|---|
| $f_0$ | [2, 4.5] | Hz | 3.297 |
| $f_1$ | [0.1, 0.6] | Hz | 0.561 |
| $F_x$ | [100, 300] | N | 162.51 |
| $F_z$ | [200,500] | N | 275.39 |
| $k_{\text{VMC}}$ | [500, 2000] | N/m | 522.76 |

Table 3: Optimization variables and best-performing values among 5 optimization runs. The nominal positions of the feet in the leg frames are the same as in Table 1.

- **Model inaccuracies:** The simulated model is only an approximation. Real robots differ in masses, inertias, and joint limits, and even small differences can affect control performance.
- **Hardware imperfections:** Motors exhibit nonlinearities, friction, backlash, and delays, while sensors suffer from noise, limited sampling rates, and imperfect calibration, effects often idealized in simulation.
- **Safety and stability:** Unlike simulation, hardware failures can cause damage. Safety mechanisms such as joint and torque limits, soft stops, and emergency shutdowns are essential.
- **Parameter tuning:** Controller gains optimized in simulation rarely transfer directly to hardware and must be re-tuned to match physical dynamics.
- **Computational constraints:** Onboard computation may be limited, requiring control law optimization or simplification.

**Running the optimization on hardware:**

Optimization should be incremental and safety-focused:

1. Initialize with the best parameters from simulation.
2. Test at low power or small motion amplitudes to verify stability.
3. Collect real-world data (errors, torques, sensor feedback).
4. Gradually refine parameters ($f_0$, $f_1$, $F$), avoiding simultaneous tuning of many variables.
5. Continuously monitor safety constraints throughout all tests.

## 2.5 Q5: Additional Challenges for Walking

Designing a walking controller adds significant complexity compared to hopping:

- **Gait coordination:** Walking involves asynchronous leg motions and phase differences (stance vs. swing), unlike symmetric hopping.
- **Balance and stability:** The robot often has only partial ground contact, making center-of-mass balance harder to maintain.
- **Dynamic transitions:** Walking requires smooth handling of transitions between phases or speeds, such as from walking to running.
- **Ground interaction:** Uneven terrain, slipping, and variable contact forces introduce uncertainty.
- **Synchronization:** Accurate timing and coordination between legs are crucial to avoid tripping or instability.

# GENERATIVE AI DECLARATION

**1) Use of Generative AI**

We used generative AI tools as support throughout the project:

- To resolve setup issues and environment configuration errors.
- To clarify technical concepts beyond the teaching assistant's explanations.
- For code completion and debugging once the main logic was in place.
- To refine comments and improve report clarity.
- To correct grammar and improve the fluency of this report.

AI was employed as a *support tool* for understanding and refinement; all design and implementation decisions were made independently.

**2) Most Useful Aspects**

AI was most useful for clarifying complex concepts, resolving specific technical issues, and identifying LaTeX syntax errors during report writing.

**Example:**

> **Prompt:** "How to reduce the margin of this document"
>
> **AI Answer:** "Place this after classicthesis: [a4paper, left=3.5cm, right=2.5cm, top=3cm, bottom=3cm]geometry"

**3) Least Useful Aspects**

AI was less effective when debugging incomplete or unclear code logic. In such cases, responses often proposed unrelated redesigns rather than targeted fixes.

**Example:**

> **Prompt:** "The simulation diverges after 10 seconds — can you fix the controller code?"
>
> **AI Answer:** "Replace your PD controller with a reinforcement learning agent."
>
> This was unhelpful as it ignored the intended control structure.

**4) Overall Impact**

AI made the project smoother by accelerating debugging, improving documentation, and enhancing the report's clarity and readability.

**5) Learning Impact**

AI positively contributed to our understanding of the underlying algorithms and control strategies, enabling deeper comprehension of the code logic rather than mere application.

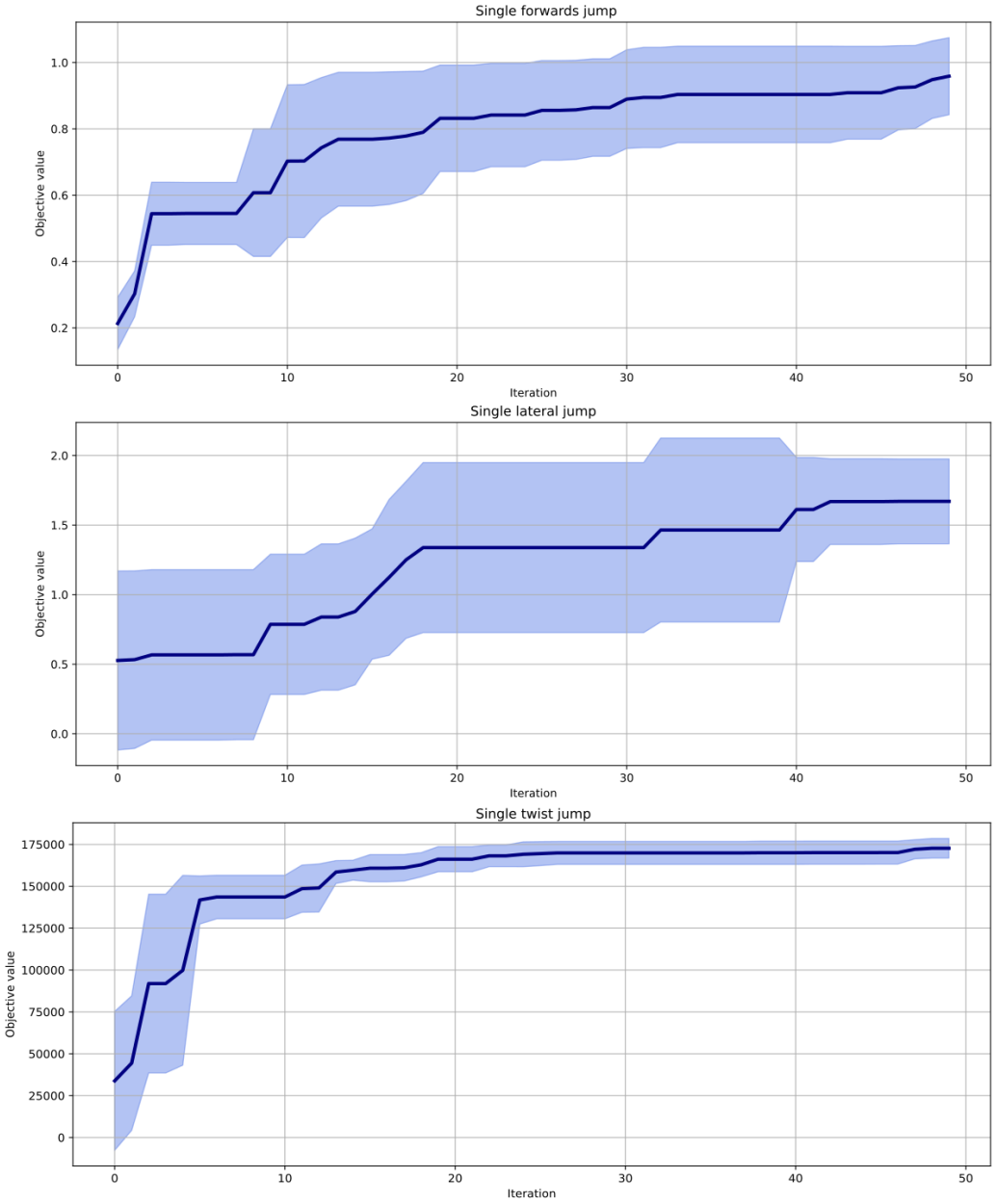


Figure 1: Single jump optimization: training average (solid line) and standard deviation (blue area) across 5 runs for forward (top), lateral (middle), and twist (bottom) jumps. All runs achieve successful jumping.

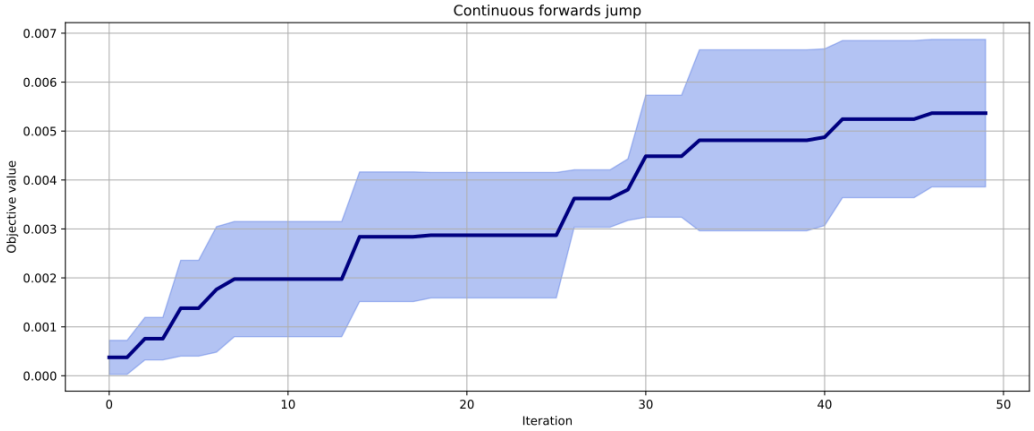


Figure 2: Continuous forward hopping: training average (solid line) and standard deviation (blue area) across 5 runs. All runs achieve stable hopping.

G. Bellegarda, M. Shafiee, M. E. Özberk, and A. Ijspeert, "Quadruped-frog: Rapid online optimization of continuous quadruped jumping," 2024. (Cited on page 5.)