École polytechnique fédérale de Lausanne

Master of Robotics

Legged Robots

Design and Optimization of a Quadruped Jumping Controller

October 21, 2025

Produced by

LUYET Maxime,
KUGATHASAN Ashvin
BAUER Arthur,

Group 04

Professors
Auke Jan Ijspeert

Table of contents

1 Introduction

This project focuses on the design and optimization of a 3D jumping controller for a quadruped robot, extending the principles of single-leg control in 2D. The controller combines foot force profiles, Cartesian impedance control, and Virtual Model Control (VMC) to generate stable and efficient jumps.
The work is divided into two parts: implementing and tuning the jumping controller for various jump types, and optimizing its parameters using Bayesian optimization with Optuna [1] to improve performance and stability.
The simulation is launched via terminal with the command "`python quadruped_jump.py <jump_type>`". If no argument is provided, the program prompts the user to select a jump type.The optimization file works in a similar way.

2 Basic controler

2.1 Description

This section presents the operation of the jump controllers. This first stage was carried out without optimizing the force profile or control gains (K_p, K_d, K_{VMC}). A stable forward jump configuration was selected using the gain values from the reference paper [2], with slight adjustments to K_d to improve damping and stability for higher jumps. The Z-axis damping factor is slighltly differnt in order to better absorb impacts and reduce oscillations.
The default stable parameters were:

```
force_profile = FootForceProfile(f0=1, f1=0.5, Fx=130, Fy=0.0, Fz=250)
k_vmc = 200
Kp_cartesian = np.diag([400]*3)
Kd_cartesian = np.diag([40, 40, 60])
```

2.2 Effect of foot position

The goal of this section is to determine the most stable foot position for the robot. This distance is expressed as a negative value since the reference frame is fixed to the robot's body. The pitch angle was used as the main stability criterion for the front jump. By plotting it against the X-displacement (both averaged over five jumps), upper and lower limits for the Z foot position were identified. As shown in Figure 1, stability decreases below –0.25 m, while positions above –0.10 m risk ground contact. A compromise value of –0.15 m was therefore chosen, offering both good stability and sufficient ground clearance.

2.3 Jumps implementation

The omnidirectional jumping controller is organized as a finite state machine, where each jump type corresponds to a distinct mode with its own force profile. The stabilization and damping gains (K_p, K_d, and K_{VMC}) are shared across all modes to ensure consistent behavior.
The "*Front_jump*" mode applies forces only along the X and Z axes.
The "*Side_jump_L*" and "*Side_jump_R*" modes generate lateral motion along the Y axis, with identical force magnitudes in Y and Z but opposite Y signs depending on direction.
For the "*Cw_twist*" and "*Ccw_twist*" modes, the controller applies opposite Y forces between front and rear legs to produce rotational motion. The forces are negative on the front legs for clockwise rotation and negative on the rear legs for counterclockwise rotation.

The force profile parameters (F_x, F_y, F_z, f_0, f_1) were manually tuned to study the robot's behavior in simulation. This empirical approach identified stable parameter ranges later used as initial values for optimization.

2.4 Effect of K_{vmc}

The virtual model control gain K_{vmc} has a significant impact on the stability of the jumps. The first part of Figure 2 shows the evolution of the roll and pitch angles over time during six consecutive jumps.
For a zero K_{vmc} value, it can be observed that the pitch angle does not stabilize sufficiently and exhibits a small overshoot around 4.5 s. This instability propagates to the next jump, leading to an imbalance that ultimately causes the roll angle to diverge and the robot to fall. A possible solution would be to increase the resting time between two jumps, allowing the PD controller to stabilize the robot more effectively before the next jump. However, this adjustment would reduce the robot's forward velocity along the X-axis.
In the subsequent test with $K_{vmc} = 50$, better stability was achieved and the robot did not fall. Nevertheless, using the default gain values introduced in Section 2.2 resulted in improved performance in terms of distance covered for a same force profile as shown in the second part of the Figure 2.
From these observations, it can be concluded that K_{vmc} has a clear influence on both the robot's stability and jumping performance.

3 Optimisation

In this section, we detail the procedure used to optimize the different parameters in order to achieve a stable and high-performing robot. All optimizations were performed using a bayesian optimisation Optuna library [1] with a by. By specifying parameter ranges and an objective function to maximize, Optuna iteratively searches for the parameter set that maximizes the objective over a given number of iterations. It is also possible to minimize some terms by assigning them negative weights in the objective function. Each optimization was run for 50 iterations to ensure convergence toward the best possible performance.

3.1 Controller optimisation

To optimize the different jumping behaviors, we first selected a baseline controller. As mentionned before, we first used similar gains to the study [2], namely $K_{dxy} = 40$, $K_{dz} = 60$, $K_p = 400$, and $K_{vmc} = 200$. We then attempted to optimize the parameters f_0, f_1, F_x, F_y, and F_z for the forward jump. However, we quickly noticed that the controller itself was causing instability issues. Therefore, we decided to first optimize and tune the controller before addressing the optimization of the individual jumping behaviors.

To optimize the controller, we created a new jump that acts only along the Z axis, using the parameters $f_0 = 1.5$, $f_1 = 0.8$, $F_x = 0$, $F_y = 0$, and $F_z = 120$.

We parameterized the values of K_p, K_d, and K_{vmc}, and further separated K_p and K_d into two distinct components: one for the X, Y directions and one for the Z direction. This resulted in five variables to optimize: K_{pxy}, K_{pz}, K_{dxy}, K_{dz}, and K_{vmc}.

We then defined a objective function that evaluates only the stability of the model:

$$\begin{aligned} stab\_score = -\,(\,&20 \cdot max\_roll + 20 \cdot max\_pitch + 10 \cdot mean\_roll \\ &+ 10 \cdot mean\_pitch + 15 \cdot roll\_std + 15 \cdot pitch\_std + 5 \cdot dist\_y\,) \end{aligned}$$

To assess stability, we minimize the maximum and average roll and pitch angles. In order to minimize oscillations, we also included the standard deviation of the roll and pitch angles. Additionally, we minimize the displacement along the Y direction.
After several optimization runs, we obtained the following results: $K_{pxy} = 505.1$, $K_{pz} = 599.57$, $K_{dxy} = 69.91$, $K_{dz} = 79.92$, and $K_{vmc} = 465.27$.
We observed that the controller's stability mainly depends on the ratio between K_p and K_d. As shown in Figure 3, with $K_p = 80$, the optimal ratio $\frac{K_p}{K_d}$ is approximately 7.3. The optimization resulted in a ratio of 7.22 for $\frac{K_{pxy}}{K_{dxy}}$ and 7.5 for $\frac{K_{pz}}{K_{dz}}$. Therefore, the optimized values are very close to the ideal ratio.
As shown in section 2.4, a higher K_{vmc} value also leads to increased stability.

3.2 Jumps optimisation

Now that a stable and robust controller has been obtained, we proceeded to optimize the different jumping behaviors. To do so, we used the controller gains obtained in the previous section. The parameters to be optimized are therefore only the foot force profile parameters. The following sections present the optimization steps for each type of jump, including the foot force profiles, the objective functions, the specific characteristics of each jump and the obtained results.

3.2.1 Front jump optimisation

The parameters to be optimized are: f_0, f_1, F_x, and F_z, while F_y is fixed at 0.
The objective function is defined as:

$$\text{score} = dist_x - 3 \cdot |dist_y| - 3 \cdot max\_pitch$$

This objective function maximizes the distance traveled along the X axis, while penalizing the displacement along the Y axis and the maximum pitch angle.
We also introduced a bias to make the front legs push harder than the back legs, since the front of the robot is heavier. This bias increases the applied force on the front legs by approximately 20% compared to the back legs.
Through the Optuna optimization, we obtained $f_0 = 1.67$ and $f_1 = 0.7$. We then ran a second optimization with f_0 and f_1 fixed in order to refine the values of F_x and F_z. The final optimized values were $F_x = 291.69$ and $F_z = 393.87$.
After 10 jumps, the robot covered a distance of $10.8\,\text{m}$ along the X axis and $0.01\,\text{m}$ along the Y axis.

3.2.2 Side jump optimisation

The parameters optimized for the side jump are f_0, f_1, F_y, and F_z, while F_x is fixed to 0 to ensure purely lateral motion.

$$\begin{aligned} \text{score} = 70 \cdot |\text{dist}_y| + 25 \cdot \frac{\text{flight\_stab}}{\text{flight\_stps}} + 10 \cdot \frac{\text{smoothness\_score}}{\text{tot\_steps}} \\ + \text{orientation\_bonus} - 10 \cdot \min\left(\frac{|\text{dist}_x|}{0.2}, 1.0\right) \end{aligned}$$

This objective function emphasizes lateral displacement along the Y-axis (70 points), with a minimum threshold of 0.12 m; if not reached, the score is set to -5. Stability and smoothness are also rewarded:

Flight stability (25 pts): fraction of airborne steps with $|\text{roll}|, |\text{pitch}| < 0.3$ rad.
Smoothness (10 pts): rewards steps where Δroll and Δpitch are below 0.05 rad.
Orientation bonus (5 pts): 5 pts for $\text{pitch}_{\max} < 0.15$ rad, 3 pts $\text{pitch}_{\max} < 0.3$ rad.
Forward drift penalty (up to 10 pts): discourages X-axis movement greater than 0.2 m.

An empirical adjustment, "Y_foot_offset", was made by introducing an outward foot offset, to improve lateral stability during takeoff and landing. This modification increases the support base, reducing roll oscillations during side jumps. A side bias was introduced to increase the lateral force on the legs on the pushing side. This bias raises the applied force by 125% compared to the opposite side, enhancing sideways propulsion and overall jump performance. We attempted to introduce a diagonal bias to prevent the robot from gradually turning in circles after multiple jumps. Although the robot remained roughly on the intended axis, the resulting behavior was suboptimal, creating a poor trade-off. Therefore, we decided not to include it. Additionally, we tried a two-phase optimization approach to first tune K_{vmc}, K_{dxy} and K_{pxy} separately, but this did not yield any conclusive improvements. Through the Optuna optimization, we obtained the following parameters for the left side jump : $f_0 = 1.7$, $f_1 = 0.8$, $F_y = 100$, and $F_z = 170$. After five consecutive jumps using the optimized parameters, the robot achieved a lateral displacement of 0.776 m along the Y-axis, while drifting about 0.109 m along the X-axis.

3.2.3 Twist jump optimization

The parameters to be optimized are as follows: f_0, f_1, F_y, and F_z, while F_x is fixed at 0.
The objective function is defined as:

$$\text{score} = \text{Total rotation angle} - 30 \cdot |dist_x| - 30 \cdot |dist_y| - 20 \cdot \text{max pitch angle}.$$

This objective function maximize the robot's rotation while ensuring that it does not deviate from its initial position and remains stable.
To further improve stability we used an outward foot position offset, Y_foot_offset. As shown in Figure 4, increasing the foot offset improves the objective function result. A wider stance provides a larger support area, which increases resistance to tipping and better distributes the forces during landing.
We then observed that regardless of the values of F_y and F_z, the robot remained stable. To keep the results physically realistic, we imposed an upper bound of 400 on both F_y and F_z.

Through the Optuna optimization, we obtained : $f_0 = 2.46$, $f_1 = 0.66$, $F_y = 359.11$, and $F_z = 394.07$. After 10 jumps, the robot achieved a total rotation angle of 42.27 rad, corresponding to 6 full rotations and 262°, with a deviation of 0.06 m along the X-axis and 0.03 m along the Y-axis.

3.3 Measures to ensure stability

To maintain a stable robot, the most critical step was to design a stable and robust controller. This process is detailed in Section 3.1. A high K_{vmc} value, for instance, provides significant stability.
During the optimization process, a rotation-related component was included in the objective

functions to enhance stability. Specifically, each objective function penalized excessive pitch angles through a negative term proportional to max_pitch, encouraging the optimization to favor foot force profiles that maintain the robot in stable configurations.
For the side and twist jump, the introduction of the "Y_foot_offset" also contributed to increased stability.

3.4 Fastest forward continuous hopping controller

To achieve a force profile for maximum forward speed, we based the controller on the front jump controller. The gains K_p, K_d, and K_{vmc} were not modified, as satisfactory results were obtained by optimizing only the foot force profile parameters.
The objective function was adjusted to incorporate a notion of speed:

$$\text{score} = \frac{10000 \cdot dist_x}{nb\_tot\_steps} - 3 \cdot |dist_y| - 6 \cdot max\_pitch$$

Here, nb_tot_steps represents the total number of simulation steps, providing a discrete measure of time. Dividing the traveled distance $dist_x$ by nb_tot_steps gives a quantity proportional to the robot's average forward velocity. The function thus aims to maximize forward speed while minimizing lateral deviation and maximum pitch.
Initially, the penalty for max_pitch was set to 3, which led to semi-stable behaviors. Increasing the penalty to 6 and doubling the number of jumps per trial (from 10 to 20) ensured long-term stability.

Through the Optuna optimization, we obtained : $f_0 = 2.25$, $f_1 = 1.71$, $F_x = 229.16$, and $F_z = 222.44$. After 20 jumps, the robot achieved a distance of in the direction X of 18.71m in 10.28 s thus acheving an average speed of 1.82 m/s. With deviation of 0.08 m along the Y axis.

4 Discussion

4.1 Hardware Implementation

Before deploying the controller on hardware, clear safety limits must be defined to prevent actuator damage, including torque and motion constraints with automatic shutdowns. Risk assessments such as HARA [3] or FMEA [4] can guide the implementation of safety functions for both the robot and the operator. Simulation parameters should be calibrated to match real-world dynamics, and a CI/CD pipeline can be used to partially automate safe testing and optimization. On hardware, parameters should be tuned incrementally within predefined safety bounds, starting from the best simulated values.

4.2 Design of Walking Controller

Designing a walking controller for a quadruped is more challenging than a hopping one because it requires precise coordination between legs, stable transitions between support phases, and continuous balance control. Unlike hopping, walking involves cyclic gait patterns and contact timing management, which make the controller more complex and sensitive to parameter tuning.

A Apendix

A.1 Graphs

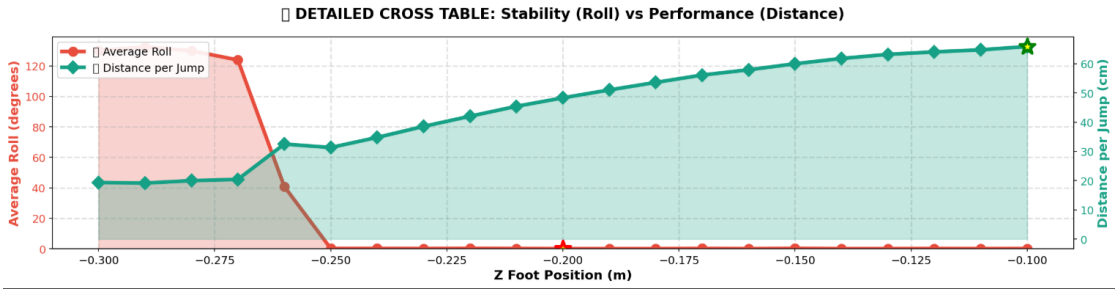

Figure 1: Cross table of the roll pitch angle and X distance averaged on 5 jumps in function of Z foot position

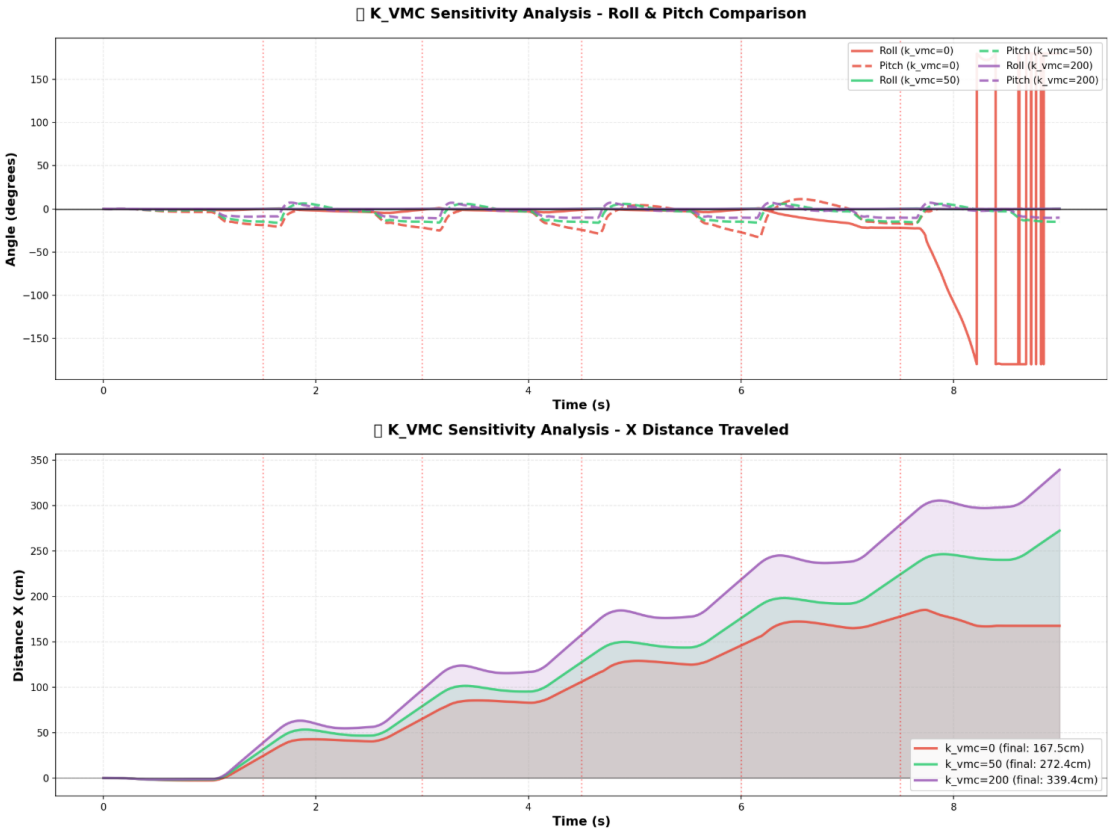

Figure 2: Graph showing the influance of VMC gain on stability and jump distance.

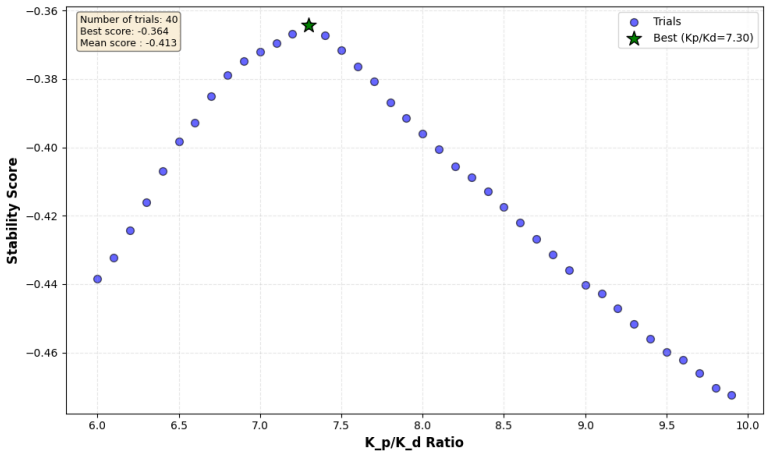

Figure 3: Graph showing the relationship between the ratio $\frac{K_p}{K_d}$ and the stability.

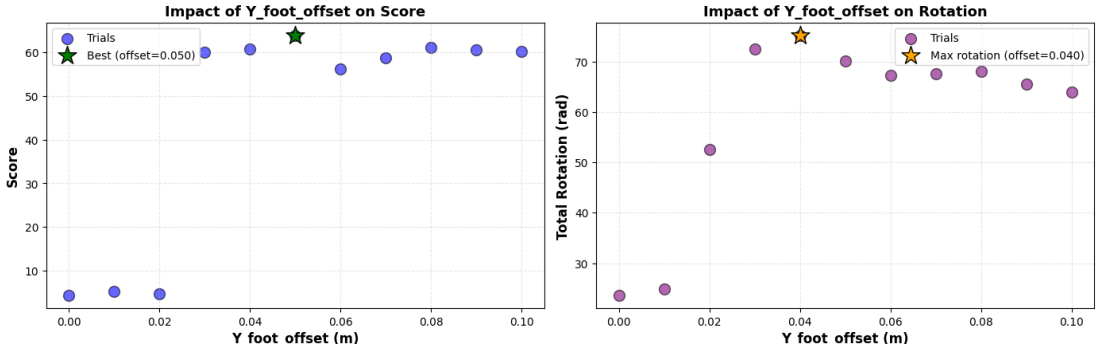

Figure 4: Graph of the twist jump performance as a function of Y_foot_offset.

A.2 Results

Parameter	Front jump	Speed hop	Side jump	Twist jump
Nb jumps	10	20	5	10
f0	1.67	2.25	1.70	2.46
f1	0.70	1.71	0.80	0.66
Fx	291.69	229.16	0	0
Fy	0	0	100	359.11
Fz	393.87	222.44	170	394.07
Dist x [m]	10.80	18.72	0.11	0.06
Dist y [m]	0.01	0.08	0.77	0.03
Speed [m/s]	1.07	1.82		
Rotation [rad]				42.27

Table 1: Summary of jump performance results

B Generative AI Declaration

During this project, our team made use of two generative AI tools, ChatGPT-5 and Claude Sonnet 4.5 via github copilot in VSCode. Both tools were used for different purposes, and their usage was consistent among team members.

B.1 Aspects of the project where generative AI was used

- ChatGPT-5 was used for *correcting and reformulating text* in the report, improving clarity, grammar, traducing and technical phrasing.
- Claude Sonnet 4.5 was used for *code-related tasks*, including generating graphic functions and visualizations. By specifying parameters, it could generate well-structured plotting functions ready to be integrated into the code. It was also useful to introduce comments in the code.

B.2 Most useful aspects

- ChatGPT-5:
  Prompt: "Correct and improve the clarity of this paragraph about foot position testing in the context of a legged robot report."
  Result: A concise, accurate, and well-structured paragraph that preserved technical meaning while improving readability.
- Claude Sonnet 4.5:
  Prompt: "Generate a Python function to plot X-displacement against pitch angle for our robot, with parameters adjustable from outside the function."
  Result: Fully functional plotting functions that could be directly integrated into our scripts.

B.2.1 Least useful aspects

- ChatGPT-5: generating code from scratch often required manual adaptation to fit the project context.
- Claude Sonnet 4.5: text manipulations were less effective and required editing.

B.2.2 Impact on project completion

The use of generative AI made the project smoother, mainly by improving the quality and readability of the report. The plots generated by Claude Sonnet 4.5 were visually appealing and allowed us to focus more on designing effective cost functions and tuning parameters. While the AI-generated code generally provided good skeletons, manual modifications and reviews were often necessary.

B.2.3 Impact on learning and understanding

The use of AI positively impacted understanding, particularly for structuring text and clearly communicating technical ideas. All critical points were carefully reviewed and adjusted to ensure they accurately reflected our understanding and design choices in the project.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. ACM, 2019.

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

[3] EPFL Graph Search. Évaluation des risques (risk evaluation). `https://graphsearch.epfl.ch/fr/concept/219072`, 2024. Accessed: 2025-10-21.

[4] EPFL Graph Search. Failure mode and effects analysis (fmea). `https://graphsearch.epfl.ch/fr/concept/981631`, 2024. Accessed: 2025-10-21.