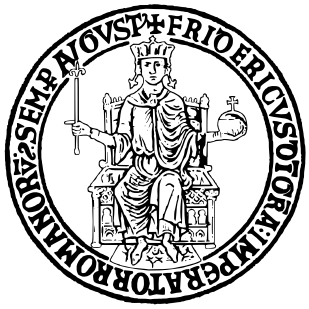
Università degli studi di Napoli Federico II



Scuola politecnica e delle scienze di base

Dipartimento di ingegneria elettrica e tecnologie dell'informazione

Tesi di laurea magistrale in ingegneria dell'automazione e robotica

Whole-Body Nonlinear Model Predictive Control for Humanoid Locomotion and Obstacle Negotiation

Relatore
Prof. Vincenzo Lippiello

Correlatore
Riccardo Aliotta

Candidato
Alberto Guglielmo
P38/261

Anno accademico 2024/2025

(A mamma e papà, il mio punto fermo)

Abstract

This thesis investigates the design of a whole-body Nonlinear Model Predictive Control (NMPC) framework for the Unitree H1-2 humanoid robot, with the goal of enabling stable locomotion and obstacle negotiation in challenging and realistic environments.

The study begins with an analysis of modern control and optimisation techniques for non-linear, high-dimensional robotic systems, leading to the adoption of MuJoCo MPC (MJPC) as the most suitable platform for model-based control.

Among the available optimisation technique, the iterative Linear Quadratic Gaussian (iLQG) method demonstrated superior stability, robustness and repeatability in locomotion tasks.

In MuJoCo, a detailed simulation model of the robot was implemented, including position-controlled actuators and a set of proprioceptive and exteroceptive sensors. Among these, a LiDAR-inspired perception system was integrated to enable environment reconstruction and support task execution.

Custom cost terms were designed to shape locomotion behaviour, regulate posture and balance, and manage interaction with the environment. These components were then extended to implement obstacle-overcoming strategies and integrated into a unified framework in which visual perception triggers transitions between locomotion and obstacle negotiation modes.

The experimental results show that the proposed NMPC controller achieves stable walking, reliable clearance of obstacles up to 30 cm, and robust operation on uneven terrain. The findings highlight the potential of whole-body NMPC for humanoid control and outline the computational conditions required for future deployment on the real hardware.

Contents

Chapter 1

Introduction

Humanoid robotics represents one of the most complex and ambitious technological frontiers of the twenty–first century. Unlike industrial manipulators or wheeled platforms, humanoid robots must operate in environments designed for humans, where locomotion, balance, manipulation, and perception are deeply interconnected. Their ability to move autonomously in unstructured spaces, overcome obstacles, and adapt to external disturbances requires control strategies capable of coordinating dozens of degrees of freedom while satisfying nonlinear dynamic constraints.

In recent years, the rapid evolution of high–fidelity physics simulators and increasingly powerful CPUs has renewed interest in model–based whole–body control approaches. Among these, Nonlinear Model Predictive Control (NMPC) has emerged as a particularly effective framework, as it enables the robot to repeatedly solve a finite–horizon optimal control problem while explicitly enforcing constraints on contacts, joint limits, and dynamic feasibility. The main limitation of NMPC, however, lies in the significant computational effort required to optimise high–dimensional nonlinear models in real time, which has historically hindered its application to humanoid robotics.

This thesis addresses these challenges by developing a whole–body NMPC framework for the Unitree H1-2 humanoid robot, currently available at the PRISMA Lab of the University of Naples Federico II. After an extensive analysis of optimisation methods and simulation tools, the MuJoCo MPC (MJPC) framework was identified as the most suitable platform for trajectory optimisation. MJPC combines a differentiable physics engine with real–time optimisation algorithms, including gradient–based methods, sampling strategies, and the iterative Linear Quadratic Gaussian (iLQG) algorithm. A central objective of this work is to determine which of these optimisation strategies is most effective for repetitive locomotion tasks and under which computational conditions real–time execution becomes feasible.

A detailed and physically consistent simulation model of the H1-2 robot was therefore developed in MuJoCo, including proprioceptive sensors, position controlled actuators and a perception layer based on a LiDAR–inspired rangefinder system. This sensing architecture enables reconstruction of the surrounding en-

vironment and online detection of obstacles, allowing the controller to generate adaptive locomotion behaviours.

The walking task was formulated through a set of residuals that encode posture regulation, balance, foot placement, heading, and velocity tracking. Additional residuals were introduced to enable the robot to negotiate obstacles by dynamically adjusting step height, foot trajectories, and whole–body posture, including the exploitation of arm–wall interaction to enhance lateral stability. The resulting control architecture was validated in three scenarios of increasing complexity: flat–ground walking, obstacle negotiation, and perception–driven locomotion inside a curved fuselage–like environment featuring an obstacle.

The proposed behaviours were evaluated in terms of centre–of–mass oscillations, distance of the CoM from the support line, gait periodicity, and trajectory regularity. A detailed computational analysis was also carried out to assess the feasibility of real–time NMPC on the physical robot. The results indicate that, while full–body optimisation with all 27 actuators remains beyond the computational capabilities of the onboard hardware, real–time execution becomes plausible when considering reduced–order models involving 13–21 actuators.

Overall, this work demonstrates that whole–body NMPC, when combined with an efficient differentiable simulator, is capable of generating dynamically consistent humanoid behaviours in real time and represents a promising direction for future sim–to–real deployment on the Unitree H1-2. The thesis contributes a complete NMPC framework, the integration of perception into the control loop, and a systematic analysis of the conditions required for real–time whole–body optimization.

The source code is available on GitHub at github.com/albertoguglielmo/mujoco-h1_2.

Chapter 2

Context and State of the Art

2.1 Background on Humanoid Robotics

The idea of constructing machines that reproduce human morphology and motion has deep historical roots. Early mechanical automata, such as Leonardo da Vinci's *Automa Cavaliere*, already demonstrated an embryonic form of anthropomorphic actuation based on levers, gears, and tendon-like ropes [9]. Although primitive and lacking sensing or autonomy, such devices represented some of the earliest systematic attempts to engineer human-inspired mechanical structures [10].

Substantial progress toward functional humanoid robots became possible only with the advent of electricity, control electronics, and digital computation during the 20th century. A major conceptual milestone was the formulation of dynamic stability criteria for bipedal locomotion, particularly the Zero Moment Point (ZMP), introduced by Vukobratović and collaborators [11]. In parallel, the first operational humanoid robot, *WABOT-1*, was developed at Waseda University in 1973, integrating motorized limbs with tactile and visual sensing and limited verbal interaction [15].

Industrial automation in the same period also produced influential robotic systems such as the *Unimate* manipulator, recognized as the first commercial industrial robot, and the SCARA [13] architecture, which introduced selective compliance for high-speed assembly tasks. While these platforms revolutionized factory automation, they did not address locomotion or interaction in unstructured environments, thereby motivating the development of humanoid robots capable of whole-body coordination.

Humanoid robots aim to operate directly within human environments by exploiting human-like proportions, locomotion patterns, and manipulation capabilities. They offer advantages such as navigating stairs, handling tools, and performing tasks in cluttered or hazardous scenarios. However, these capabilities introduce significant challenges, including high-dimensional dynamics, multi-contact interactions, the need for real-time balance control, and safe physical interaction with humans. Classical approaches such as ZMP and the Linear Inverted Pendulum Model (LIPM) [24] remain widely used, while more advanced

methods, including Model Predictive Control (MPC) and Reinforcement Learning (RL), have been explored to address complex, highly dynamic behaviors.

In the last decades, progress in sensing, actuation, and computation has accelerated the development of full-scale humanoids. Notable contributions include platforms from Honda, Boston Dynamics [50], SoftBank Robotics, PAL Robotics [51], Agility Robotics [52], Engineered Arts, and Unitree Robotics,whose H1 series represent one of the most recent examples of advanced, research-oriented humanoid systems.

2.2 Control Techniques for Humanoid Robotics

The humanoid platform used in this thesis is the Unitree H1-2, a robot equipped with 27 actuated joints (excluding the hands). Given its high number of degrees of freedom and the intrinsic dynamic complexity of humanoid locomotion, it is essential to adopt a control strategy that guarantees stability, adaptability, and coordinated full-body motion.

For the whole-body control of the robot, we selected a Nonlinear Model Predictive Control (NMPC) framework, implemented through different optimization techniques. This chapter presents an overview of the relevant scientific literature and state-of-the-art control methods that support and motivate this choice.

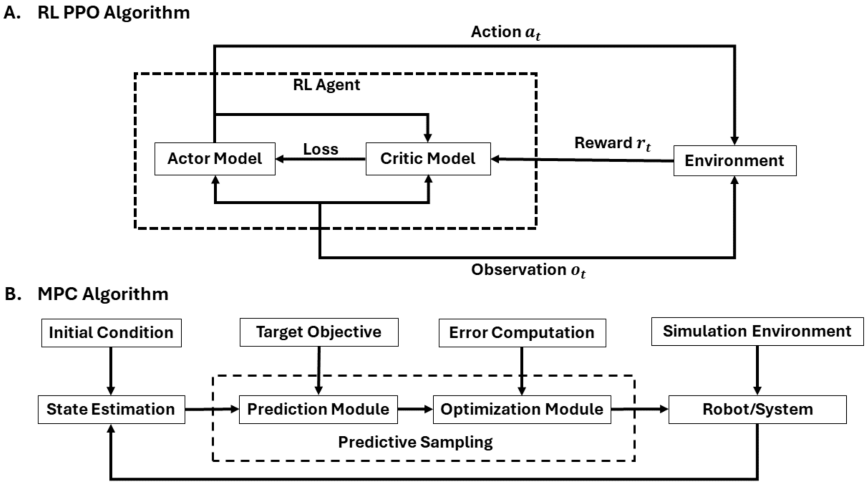


Figure 2.1: A. Reinforcement learning proximal policy optimization algorithm. B. Model predictive control framework using predictive sampling. Source: [5].

In the field of legged mobile robotics, the most widely adopted control paradigms include Reinforcement Learning (RL) [2], Model Predictive Control (MPC) [1], and hybrid methods [27] that combine both model-based and data-driven approaches. Regarding control strategies, both model-based and model-free frameworks can be employed. Model-based methods, such as MPC, rely on accurate representations of the robot's dynamics, while model-free methods use data-driven formulations, including neural networks and reinforcement learning. MPC

and RL represent two complementary paradigms for optimal control. As discussed by Bertsekas [3], both aim to minimize a cumulative cost function but differ fundamentally in their formulation and implementation. MPC operates by solving a finite horizon optimal control problem in real time using a predictive model of the system dynamics. Chapter 3 will explore this in greater depth from a theoretical perspective. Its main advantages include the explicit handling of state and input constraints, deterministic and interpretable behavior, and theoretical guarantees of stability when the model is accurate. These features make MPC particularly suitable for safety-critical and high-precision applications such as humanoid locomotion and whole-body control.

By contrast, Reinforcement Learning seeks to approximate the optimal policy or value function through data-driven interaction with the environment, often without relying on an accurate dynamic model. Its main strengths are adaptability, the ability to learn complex motor behaviors, and flexibility in unmodeled or stochastic environments. However, RL typically requires large quantities of training data and does not inherently guarantee stability or constraint satisfaction, making its direct application to physical humanoids challenging.

Bertsekas [3] states that MPC can be seen as an online approximation based on dynamic programming models, while RL is an offline implementation based on data of similar optimal control principles. In practice, these two methods can be considered complementary: MPC emphasizes prediction and constraint satisfaction through analytical models, while RL focuses on adaptation and generalization through experience.

The study by Akki et al. [5] provides a direct comparison between nonlinear Model Predictive Control (MPC) and Reinforcement Learning (RL) for legged locomotion using the Unitree Go1 quadruped robot within the MuJoCo simulation environment. The MPC controller is implemented through the MuJoCo MPC (MJPC) [37] framework using a predictive sampling algorithm, which enables real-time trajectory optimization without requiring gradient computations. In contrast, the RL controller learns its policy through direct interaction, adapting to the environment. The experiments focus on a standardized straight-walking task at constant velocity, evaluating performance in terms of disturbance rejection, energy efficiency, and terrain adaptability. The results indicate that the RL controller achieves higher disturbance rejection and lower energy consumption, while the MPC controller provides a more balanced distribution of joint control efforts, allowing effective recovery from perturbations and stable locomotion.

A recently introduced platform, HumanoidBench, provides a versatile and comprehensive simulation benchmark for humanoid robots, combining an extensive repertoire of both locomotion and whole-body manipulation tasks [34]. The benchmark is implemented using the MuJoCo physics engine and employs the Unitree H1 humanoid model equipped with dexterous Shadow Hands, thereby enabling research on complex, large action-space control problems involving full-body coordination and contact-rich interactions.

Extensive experiments carried out by the authors reveal that state-of-the-art Reinforcement Learning (RL) algorithms face substantial limitations when ap-

Figure 2.2: HumanoidBench manipulation task suite illustrating 15 whole-body manipulation scenarios of varying complexity source: [34].

plied to humanoid robots. In their analysis, they show that none of the evaluated baselines is able to achieve satisfactory performance across most tasks: the controllers particularly struggle in scenarios that demand long-horizon reasoning, whole-body coordination, and the handling of highly dimensional action spaces.

The study further highlights that these RL methods require an exceptionally large number of interaction steps even to acquire basic locomotion skills such as walking. Although agents may partially learn to stabilize when dense rewards are provided, they still fail to acquire more complex behaviors, especially in manipulation tasks.

The design of HumanoidBench places a strong emphasis on realistic dynam-

ics, coordinated full-body motion, and a diverse set of human-inspired tasks. As a result, the benchmark is not only suited to evaluating learning-based controllers, but also provides a valuable testbed for analytical and model-based approaches. Indeed, the authors explicitly state that the framework supports generic controller architectures, encompassing both learning-driven and model-driven strategies [34]. This observation motivates the investigation of advanced model-based techniques such as nonlinear Model Predictive Control (NMPC)

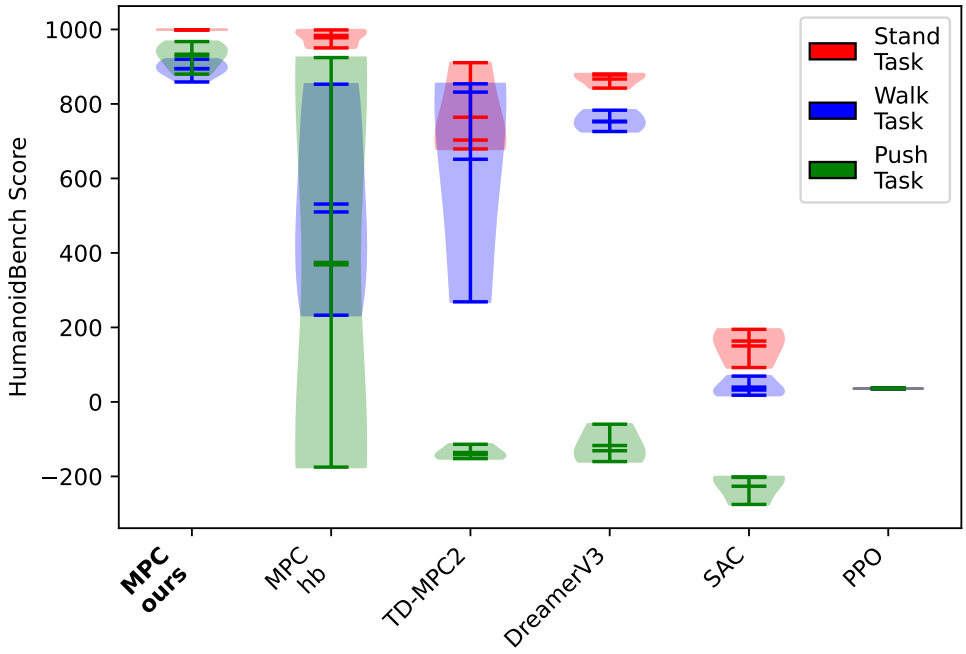


Figure 2.3: Performance comparison between the proposed controller MPC-ours that leverages shaped reward functions and the baselines:MPC-hb, which uses the original HumanoidBench reward function, and the RL baselines. The y-axis shows the HumanoidBench Score, defined as the cumulative sum of rewards over a trajectory (maximum of 1000 per task). For MPC methods, the iLQG planner is used for the Stand and Walk tasks, while a Sampling-based planner is employed for the Push task, which involves complex contact interactions. Results are averaged over six runs for each MPC method. RL baseline results are imported from [37], using the best policy from three runs per algorithm. MPC-ours significantly outperforms all baselines across tasks. Source: [7].

Building upon this motivation, Meser et al. [7] extended the HumanoidBench framework by integrating it with the MuJoCo Model Predictive Control (MJPC), enabling the evaluation of humanoid control through real-time trajectory optimization rather than offline policy learning. In their work, the authors compared two MPC configurations: MPC-hb, which directly employs the original Hu-

manoidBench reward functions, and MPC-ours, a variant enhanced with carefully designed reward shaping and stability regularization terms. These additional terms penalize excessive joint velocities, promote balance and posture maintenance, and provide feedback signals to guide the optimization process. The study demonstrates that while the standard HumanoidBench rewards (MPC-hb) often lead to unstable and physically implausible behaviors, the shaped-reward variant (MPC-ours) produces smooth and human-like trajectories. Quantitative evaluations show that MPC-ours consistently achieves higher HumanoidBench Scores across locomotion and manipulation tasks, outperforming both the MPC baseline and all reinforcement learning baselines (PPO, SAC, TD-MPC2, DreamerV3) originally reported in the benchmark, as illustrated in Fig. 2.3.These results emphasize the importance of control architectures and objective design in humanoid robotics, and motivate a broader examination of the relative strengths and limitations of model-based and learning-based approaches.

In parallel with these experimental results, a detailed theoretical study analyzing various humanoid control techniques has been conducted by Sun et al. [17], who review both model-based and learning-based locomotion approaches, including Model Predictive Control (MPC), Zero Moment Point (ZMP) control, reinforcement learning combined with imitation learning, and hierarchical hybrid control architectures.

For MPC [1], the authors emphasize that its predictive structure allows the controller to anticipate future system states and adjust inputs in advance. This feature generally leads to stable behavior and good performance when several constraints must be handled simultaneously. Its main limitation, however, is the computational burden associated with solving an optimization problem at every control step, which makes real-time execution difficult on high-DoF humanoid systems.

ZMP [20] control represents a well-established approach for bipedal walking and is appreciated for its robustness in structured settings, where foot placement and contact interactions remain predictable. By keeping the center of mass within the support polygon. However, its effectiveness decreases in highly dynamic or irregular environments, where contact transitions are harder to anticipate.

Reinforcement learning combined with imitation learning provides high adaptability to unstructured and varying environments by learning policies either through trial-and-error or from human demonstrations. A representative example of this paradigm is the DeepMimic framework, which leverages motion-capture data to guide policy learning and enables physically simulated characters to acquire complex, realistic, and robust motor skills. Through the combination of imitation objectives and task-oriented reinforcement learning, DeepMimic [21] achieves highly dynamic behaviors such as flips, kicks, and agile locomotion, while also adapting to perturbations, novel environments, and different character morphologies. Despite its flexibility, the approach requires a significant amount of interaction data, making the training process computationally and time intensive.

Hierarchical hybrid control architectures merge high-level policy planning with low-level optimization or feedback control, offering modularity and robust performance across different locomotion tasks. The main drawback lies in the complexity of system integration and synchronization across hierarchical layers, which can introduce implementation challenges in real humanoid platforms. Given the complexity of humanoid control, especially when combining predictive optimization, contact dynamics, and learning-based components, high-fidelity simulation environments play a crucial role in development, validation, and benchmarking of algorithms before deployment on real hardware.

2.3 Physical simulators for robotics

In modern robotics, the most widely adopted platforms for integrating software and hardware components are the Robot Operating System (ROS) and its latest iteration ROS 2. Before deploying control algorithms on physical robots, it is common practice to tune and validate their parameters in simulation.

For simulating the physical dynamics of robotic systems and their interactions with the environment, several simulation environments have become prominent in recent years. Among the most widely used are Gazebo, MuJoCo [26], and NVIDIA Isaac Sim. Each of these platforms offers different trade-offs between physical accuracy, computational performance, and ease of integration with robotics middleware such as ROS or machine learning toolchains.

Gazebo is an open-source simulation environment that has become a standard tool in academic and research contexts. It supports multiple interchangeable physics engines, including ODE, Bullet, and DART, providing flexibility in balancing physical realism and computational speed [32]. Gazebo offers native integration with ROS and ROS 2, supports multi-robot systems, and provides a wide range of ready-to-use models and plugins. However, its physical accuracy and numerical stability can vary significantly depending on the chosen physics backend and the simulation time step configuration.

MuJoCo (Multi-Joint dynamics with Contact) has been designed with a focus on high-fidelity simulation of articulated rigid-body systems. It employs an analytic approach for contact and friction modeling, resulting in high numerical stability and precision even with small time steps [4]. This makes MuJoCo particularly suitable for model-based control. Experimental evaluations have demonstrated that MuJoCo achieves the highest accuracy among common physics engines, while maintaining good linear stability, though it may become unstable with larger time steps or highly dynamic multi-contact scenarios.

NVIDIA Isaac Sim represents a new generation of simulators based on the PhysX engine and the Omniverse platform. It leverages GPU acceleration to deliver realistic rendering, physics computation, and parallelized multi-instance simulations [38]. This design makes Isaac Sim suitable for large-scale industrial environments, synthetic data generation, and AI model training [4]. Nevertheless, it requires powerful hardware and offers limited flexibility in CPU-only

configurations.

Recent comparative studies provide valuable insights into the performance of these simulators. Liao et al. [4] systematically evaluated five physics engines: ODE, Bullet, DART, MuJoCo, and PhysX, using complex multi-joint robot models. Their findings indicate that MuJoCo achieves the best linear stability and accuracy, while PhysX excels in angular stability and DART in friction simulation effectiveness. Similarly, Körber et al. [32] compared Gazebo and MuJoCo under reinforcement learning scenarios, highlighting that MuJoCo achieves higher real-time factors and precision for small time steps, whereas Gazebo remains advantageous for its integration with ROS and its accessibility to novice users.

In summary, Gazebo, MuJoCo, and Isaac Sim represent three distinct philosophies in robotic simulation. Gazebo prioritizes flexibility and open-source integration, MuJoCo focuses on analytical precision and control research, and Isaac Sim emphasizes GPU-based scalability and photorealism. The choice among them depends on the specific research or industrial requirements, balancing between accuracy, computational efficiency, and system integration. As also discussed by Zhang et al. [33], several previous works have investigated model-based control frameworks for legged locomotion.

Outside the MuJoCo ecosystem, Pinocchio [31] has established itself as a fundamental computational backbone for rigid-body dynamics in robotics, offering efficient evaluation of system dynamics and their analytical derivatives. This capability has enabled advanced optimal-control toolchains for contact-rich, legged behaviors

Among these, OCS2 [30] provides an open-source framework for quadruped motion generation with predefined contact phases, while Crocoddyl [23] and Aligator [19] have been used to realize whole-body control for both quadruped and humanoid robots.

However, these works rely on standalone optimal-control pipelines and do not leverage commercially supported simulation platforms commonly adopted in the robotics community, which can make them less accessible for initial experimentation. In contrast, Drake [36] provides advanced modeling and simulation tools capable of accurately capturing contact and friction dynamics for verification purposes, although their computational performance currently remains insufficient for real-time control on complex humanoid systems.

As summarized in Table 2.1, taken from Zhang et al. [33], their MuJoCo-based approach bridges this gap by combining the advantages of real-time predictive control with full-body collision detection and an interactive graphical interface, all implemented using a fast, ready-to-use simulator.

Table 2.1: A comparison of model-based control tools for contact-rich tasks. source: [33]

	[36]	[23], [19]	[30]	**[33]**
Off-the-shelf simulator	Yes	No	No	Yes
Whole-body collision detection	Yes	No	No	Yes
No fixed contact mode	N/A	No	No	Yes
Real-time control	No	Yes	Yes	Yes
Interactive GUI	No	No	No	Yes

Isaac Lab is NVIDIA's latest GPU-accelerated simulation framework, extending Isaac Sim and inheriting the massively parallel architecture of Isaac Gym [18]. By combining the PhysX engine with the Omniverse RTX renderer, it provides high-fidelity rigid-body dynamics, realistic contact modeling, and photorealistic sensor simulation (RGB, depth, LiDAR), all executed entirely on the GPU. This design enables large-scale parallel simulation, allowing thousands of environments to run simultaneously and dramatically accelerating reinforcement learning (RL) data collection compared to CPU-based simulators such as MuJoCo or Bullet. Thanks to direct integration with deep learning frameworks, Isaac Lab supports batched observations, rewards, and low-latency policy execution on the GPU, making it well-suited for training high-dimensional locomotion and perception policies in simulation before deployment on real systems.

In the context of humanoid robotics, however, applying RL to the full Unitree H1-2 model introduces practical challenges related to dimensionality rather than simulator limitations: although the official robot URDF can be converted to USD, controlling all 27 actuated joints significantly increases the search space and makes policy learning unstable or computationally expensive.

As a result, both Unitree's proprietary controller and recent academic studies adopt reduced control spaces; for example, Unitree limits its locomotion policy to lower-body joints, and Xie *et al.* [8] similarly operate their controller on only 21 of the robot's 27 degrees of freedom, omitting the wrist joints of the hands. Their work, which trains the policy in Isaac Gym and validates it through a sim-to-sim evaluation in MuJoCo, shows that excluding joints not essential for locomotion simplifies learning while preserving walking performance.

Although Isaac Lab represents a state-of-the-art platform for training neural network policies in robotics, its physics fidelity is generally lower than that of MuJoCo [35]. Shao et al. [39] evaluated their learned policies using domain randomization for sim-to-sim transfer from Isaac Gym to MuJoCo, reporting significant robustness improvements for quadruped locomotion. In contrast, for the humanoid, domain randomization offered no noticeable benefit: the policy transferred successfully to MuJoCo even without additional perturbations. This suggests that, in their setup, the humanoid walking task was sufficiently well-posed that extensive randomization was unnecessary for achieving cross-simulator generalization.

Together, these findings highlight that, while Isaac Lab (and its Isaac Gym predecessor) can simulate the H1_2 humanoid effectively, full-body RL remains computationally demanding due to the high-dimensional control problem. Consequently, this reinforces a broader trade-off in modern legged robotics between model-free and model-based strategies: model-free methods such as deep RL offer strong generalization capabilities but require large-scale data collection, whereas model-based controllers rely on accurate dynamics models and optimization (e.g., MPC) to produce predictable, interpretable behaviors without extensive training phases [37]. Embedding prior physical knowledge into control design reduces data requirements and increases reliability, though learned controllers may still provide advantages when addressing unmodeled dynamics or sim-to-real discrepancies.

The combination of high simulation fidelity, fast computational performance, and the availability of built-in optimization algorithms in the MuJoCo MPC (MJPC) framework motivates our choice of MuJoCo as the simulation environment to implement model-based control.

Chapter 3

Theoretical methods and tools

3.1 The Unitree H1-2 robot

The present work aims to contribute to the control of complex, high-dimensional systems through whole-body, model-based control strategies. To this end, the humanoid robot H1-2, developed by Unitree Robotics, was adopted as the experimental platform. The H1-2 is a full-sized humanoid robot derived from the earlier H1 model, featuring an updated mechanical design and an increased number of degrees of freedom. As shown in Fig. 3.1, it represents one of the most advanced humanoid systems currently available at the PRISMA Lab of the University of Naples Federico II, where it is employed for research on perception, locomotion, and whole-body control.

Figure 3.1: The Unitree H1-2 humanoid robot

This section provides an overview of the main technical specifications of the robot, with particular attention to its mechanical architecture, actuators, and integrated sensory systems.

3.1.1 Mechanical Architecture and Actuators

The H1-2 robot has a total mass of approximately 70 kg and a height of about 1.78 m. It is equipped with a total of 27 actuated joints, distributed as follows:

- 6 for each leg (hip $\times 3$, knee $\times 1$, ankle $\times 2$),
- 7 for each arm (shoulder $\times 3$, elbow $\times 1$, wrist $\times 3$),
- 1 in the torso (waist).

All joints are driven by Unitree M107 motors, featuring a PMSM (Permanent Magnet Synchronous Motor) architecture with a high-speed, low-inertia inner rotor. These actuators are designed to provide high torque, fast response, and good thermal dissipation, enabling smooth motion and precise control of both force and position.

The main mechanical characteristics of the joints are reported in the official Unitree documentation and on the H1 robot's web page [40].

- maximum torque at the knee: approximately 360 N·m,
- maximum torque at the hip: approximately 220 N·m,
- maximum torque at the torso (waist): approximately 220 N·m,
- maximum torque at the ankle: approximately 75 N·m per axis,
- peak torque at arm joints: shoulder and elbow approximately 120 N·m, wrist approximately 30 N·m,
- arm payload capacity: peak 21 kg, nominal 7 kg.

Power is supplied by a swappable battery with a capacity of 15 Ah (0.864 kWh) and a maximum voltage of 67.2 V. The declared walking speed is below 2 m/s, while the operating autonomy depends on the mode of operation and computational load.

Although these represent the official technical specifications provided by the manufacturer, the H1-2 simulation model uses slightly more conservative force limits to ensure greater numerical stability and more realistic dynamic behavior compared to the performance of the physical motors.

3.1.2 Control Architecture

The humanoid robot H1-2 adopts a distributed control architecture implemented on an integrated and modular computational platform. In its standard configuration, the system includes two computing units: a Motion Control Computing Unit, based on an Intel Core i5 processor, which is a dedicated and non-accessible module responsible for real-time locomotion and balance control; and a Development Computing Unit, equipped with an Intel Core i7-1265U processor, intended for user development and high-level processing tasks.

Optionally, the system can be equipped with an Intel Core i7 processor or up to three NVIDIA Jetson Orin NX modules for advanced computing, artificial intelligence, or vision-based applications [41].

The robot can be operated via a dual-hand joystick controller or through a dedicated mobile application: the input commands specify the desired linear and angular velocities, while locomotion is handled autonomously by the onboard controller and does not involve the use of the arms for stabilization [43]. In controlled walking mode, the robot reacts in real time to external disturbances by adjusting its posture and performing compensatory steps to prevent falls.

For developers, Unitree provides a complete software ecosystem and dedicated tools for robot programming and control. The second computing unit (Intel Core i7) serves as an open platform for custom development: through the official Unitree SDK2 [44], it is possible to access both low-level and high-level control interfaces, enabling the integration of proprietary algorithms into the robot's control loop. The system is compatible with the ROS2 framework and includes official descriptions in URDF and XML formats, useful for integration with simulation environments or external control applications [42].

3.1.3 Sensors and Perception Systems

The H1-2 robot integrates both proprioceptive and exteroceptive sensors:

- high-resolution rotary encoders embedded in each joint, providing accurate joint position feedback for state estimation, balance control, and locomotion;
- an Intel RealSense D435 depth camera for three-dimensional vision and obstacle perception;
- a Livox MID-360 LiDAR mounted on the head, offering 360° omnidirectional scanning and high-resolution point-cloud acquisition.

The primary external perception sensor of the robot is the Livox MID-360 LiDAR. It operates based on the Light Detection and Ranging (LiDAR) principle, which measures object distance through laser pulses and computes the Time-of-Flight (ToF) of the reflected signal. By knowing the speed of light and the direction of the laser beam, the system determines the distance point by point, constructing a three-dimensional representation of the environment.

Unlike multi-beam mechanical LiDARs, the Livox MID-360 employs a non-repetitive optical scanning technology. Inside the sensor, a single laser emitter is deflected by an optical system consisting of a rotating prism, which spins around the vertical axis ensuring full 360° horizontal coverage, and a bi-directional mirror oscillating at high frequency along two axes.

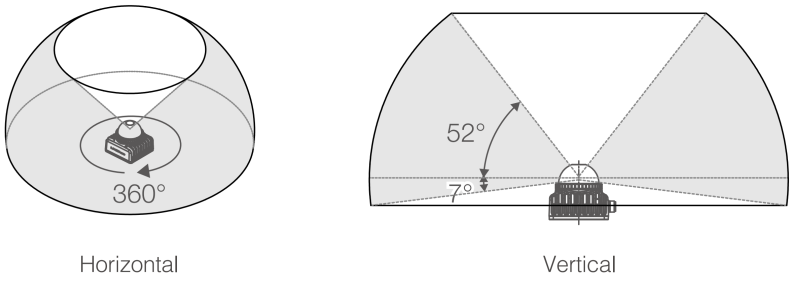


Figure 3.2: Effective FOV of the Mid-360. Source: [49].

The interaction between these two motions generates a pseudo-random pattern that never repeats the same trajectory. This principle, known as non-repetitive scanning, allows the Field of View (FOV) coverage to increase progressively over time: the longer the integration time, the more points are acquired. After approximately 0.5 s of integration, the average coverage exceeds 70% and tends to approach 100% with longer exposure times, as reported in the manual [49]. This approach enables high point density even under static conditions, avoiding complex mechanical components and improving overall system reliability.

The operating specifications of the Livox MID-360 include a horizontal field of view of 360° and a vertical field of view ranging from -7° to +52°, resulting in a total vertical span of 59°. The maximum detection range depends on the reflectivity of the surface. The sensor generates approximately 200,000 points per second at a typical frequency of 10 Hz, with a measurement accuracy of about 2-3 cm and an angular error below 0.15°.

The MID-360 also integrates a six-axis IMU (accelerometers and gyroscopes) with an output frequency of 200 Hz, aligned with the sensor reference frame and used for motion compensation and point cloud deskewing.

The output data include Cartesian coordinates (x, y, z), radial distance, elevation and azimuth angles, reflected signal intensity, and an 8-bit status code indicating point quality.

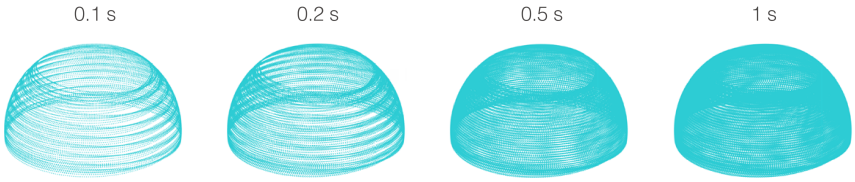


Figure 3.3: Point cloud patterns of Livox Mid-360 accumulated over different integration times. Source: [49].

In static scenarios, integrating multiple consecutive frames (between 0.2 s and 1.0 s) increases the density of the three-dimensional map and improves obstacle avoidance performance. As illustrated in Fig. 3.4, the Livox Mid-360 achieves a significantly wider and more uniform field of view compared to traditional mechanically scanned LiDARs.

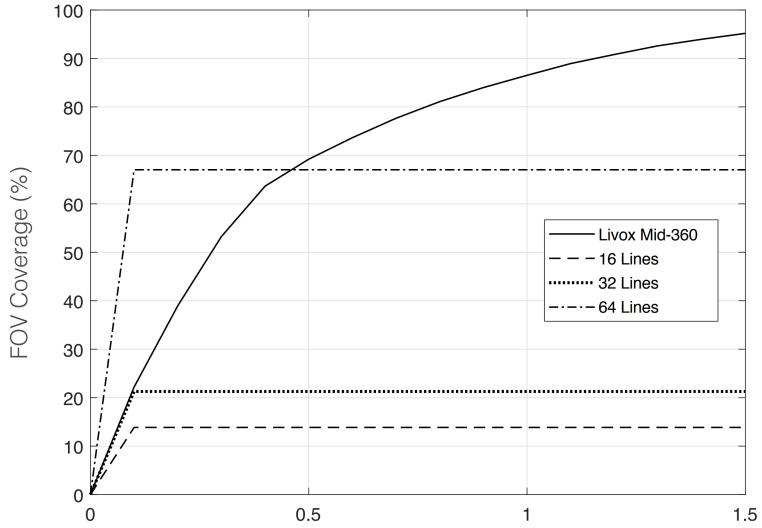


Figure 3.4: The FOV coverage of Livox Mid-360 and non-Livox LiDAR sensors using common mechanical scanning methods. The 16-line non-Livox LiDAR sensor has a vertical FOV of 30°, the 32-line sensor has 41°, and the 64-line sensor has 27°. Source: [49].

3.1.4 Joint Limits

The Unitree H1-2 humanoid robot is equipped with 27 active joints. Fig. 3.5 illustrates the naming and arrangement of the main joints, which include the shoulders, elbows, wrists, hips, knees, and ankles. Accurate knowledge of the joint limits is essential for motion planning, trajectory generation, and the avoidance of mechanically unreachable configurations. Table 3.1 reports the joint limits of the H1-2 robot, as specified in the official documentation provided by Unitree [42]. The values are expressed in radians and represent the admissible ranges of motion for each joint.

The wide range of motion of the shoulder and hip joints enables the robot to perform natural movements, whereas the more limited ranges of the wrist and ankle joints provide stability and fine control. These constraints must be respected in the design of inverse-kinematics controllers and dynamic models, in order to avoid collisions or excessive mechanical stress on the actuators. Given the mechanical complexity and the presence of multiple contact interactions, accurate simulation is essential for designing and validating whole-body control strategies. For this reason, the MuJoCo physics engine is adopted to model the robot dynamics and test the proposed control algorithms.

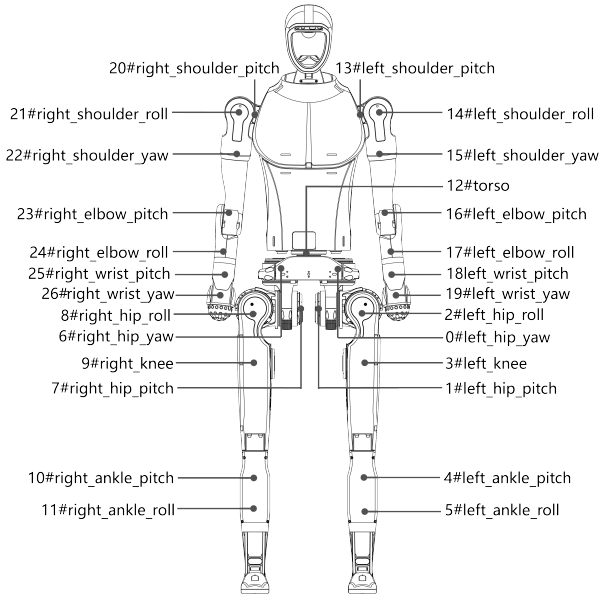


Figure 3.5: Unitree H1-2 humanoid robot with all joints labeled. Source: [42].

Joint ID	Joint Name	Limits (radians)
0	left_hip_yaw	$-0.43 \sim +0.43$
1	left_hip_pitch	$-3.14 \sim +2.50$
2	left_hip_roll	$-0.43 \sim +3.14$
3	left_knee	$-0.26 \sim +2.05$
4	left_ankle_pitch	$-0.89 \sim +0.52$
5	left_ankle_roll	$-0.26 \sim +0.26$
6	right_hip_yaw	$-0.43 \sim +0.43$
7	right_hip_pitch	$-3.14 \sim +2.50$
8	right_hip_roll	$-3.14 \sim +0.43$
9	right_knee	$-0.26 \sim +2.05$
10	right_ankle_pitch	$-0.89 \sim +0.52$
11	right_ankle_roll	$-0.26 \sim +0.26$
12	torso	$-3.14 \sim +1.57$

Joint ID	Joint Name	Limits (radians)
13	left_shoulder_pitch	$-3.14 \sim +1.57$
14	left_shoulder_roll	$-0.38 \sim +3.40$
15	left_shoulder_yaw	$-3.01 \sim +2.66$
16	left_elbow_pitch	$-2.53 \sim +1.60$
17	left_elbow_roll	$-2.96 \sim +2.967$
18	left_wrist_pitch	$-0.47 \sim +0.349$
19	left_wrist_yaw	$-1.01 \sim +1.012$
20	right_shoulder_pitch	$-1.57 \sim +3.14$
21	right_shoulder_roll	$-3.40 \sim +0.38$
22	right_shoulder_yaw	$-2.66 \sim +3.01$
23	right_elbow_pitch	$-1.60 \sim +2.53$
24	right_elbow_roll	$-2.96 \sim +2.96$
25	right_wrist_pitch	$-0.47 \sim +0.34$
26	right_wrist_yaw	$-1.01 \sim +1.01$

Table 3.1: Joint limits of the humanoid robot

3.2 The MuJoCo Simulator

MuJoCo (Multi-Joint dynamics with Contact) is a physics engine specifically designed for robotics and control applications, with particular emphasis on accurate and efficient simulation of articulated systems and contact dynamics [26]. Beyond its role as a general-purpose simulator, MuJoCo has gained widespread adoption in the optimal control community and forms the foundation of the *MuJoCo MPC* framework, which enables real-time model predictive control using differentiable rigid-body dynamics.

In this work, MuJoCo is employed to compute the evolution of the robot's dynamic state by numerically integrating the equations of motion in discrete time. The notation used in this section is summarized in Table 3.2.

Table 3.2: Summary of notation used in this section

q	generalized position vector
v	generalized velocity vector
$M(q)$	inertia matrix in generalized coordinates
$b(q, v)$	bias forces (Coriolis, centrifugal, gravity, springs)
τ	external or applied forces
J_E	Jacobian of equality constraints
v_E^*	desired velocity in equality constraint coordinates
f_E	impulse due to equality constraints
J_C	Jacobian of active contacts
v_C	velocity in contact coordinates
f_C	impulse due to contacts
h	time step size

In discrete form, the equations of motion can be expressed as:

$$M(q_t)v_{t+h} = s(q_t, v_t, \tau_t) + J_E^T f_E + J_C^T f_C \tag{3.1}$$

$$J_E v_{t+h} = v_E^* \tag{3.2}$$

$$J_C v_{t+h} = v_C \tag{3.3}$$

where the auxiliary vector s is defined as:

$$s(q_t, v_t, \tau_t) = M(q_t)v_t + h\, b(q_t, v_t) + h\, \tau_t \tag{3.4}$$

The objective of this formulation is to compute the next state of the system, that is, the pair (v_{t+h}, q_{t+h}), given the current state (v_t, q_t), the applied forces τ_t, and the dynamics of the system. In other words, MuJoCo solves a constrained dynamic problem at each simulation step, determining how the system evolves over time while satisfying both equality and contact constraints.

The robot model and its physical parameters are provided to the simulator through an `XML` file written in the *MJCF* (MuJoCo XML format). This file

defines the robot's kinematic structure, joint types, mass and inertia properties, contact geometries, actuators, and any equality or friction constraints.

Once the model is defined, the simulator computes the next state through five main computational steps. For clarity, we briefly summarize the algorithmic structure below:

1. **Forward Kinematics and Collision Detection.**
   Compute the Cartesian positions and orientations of all rigid bodies through forward kinematics. Detect potential collisions and construct the Jacobians J_E and J_C corresponding to equality constraints and active contacts, respectively.

2. **Inertial and Bias Computation.**
   Compute the inertia matrix $M(q)$ using the Composite Rigid Body Algorithm and the bias forces $b(q, v)$ using the Recursive Newton–Euler algorithm.

3. **Projection onto the Equality Constraint Subspace.**
   Express the equality constraint impulse f_E as a function of the (still unknown) contact impulse f_C. In MuJoCo, contact interactions are formulated as regular convex optimization problems with high impedance parameters, as if a virtual rigid spring were placed between the two contact points (e.g., between the foot and the ground). The system dynamics are then projected onto the subspace tangent to the equality constraint manifold. Constraint stabilization is applied by enforcing the desired constraint velocity v_E^*, ensuring numerical consistency and suppression of drift in equality constraints.

4. **Resolution of Contact Impulses.**
   Further project the dynamics into contact coordinates and solve for the contact impulse f_C together with the resulting contact velocity v_C. This step enforces non-penetration, complementarity, and frictional constraints within the contact manifold through a convex optimization or iterative solver.

5. **Numerical Integration.**
   Once the next-step velocity v_{t+h} has been obtained, integrate the system state to compute q_{t+h}. According to the official documentation, MuJoCo supports four types of integrators: three single-step integrators and one multi-step 4th-order Runge–Kutta integrator (*Euler*, *implicit*, *implicitfast*, and *RK4*). These options provide a trade-off between accuracy and computational cost, depending on the simulation requirements.

Although the next-state computation in MuJoCo results from (3.1) and a numerical integrator, in the following chapters we will adopt the compact discrete-time representation

$$x_{t+h} = f(x_t, u_t), \tag{3.5}$$

where x_t denotes the system state at time t, u_t the control input, and $f(\cdot, \cdot)$ the nonlinear state-transition map induced by the dynamics.

It is, however, important to note that the meaning of the control input u depends on the type of controller employed. Specifically, the control input can be expressed in torque, velocity, or position form. This distinction affects only the physical interpretation of the input but does not alter the dimensionality of the problem, since the vector u always has a dimension equal to the number of actuators.

Since we consider mobile robotics, the state includes both the floating base and the articulated joints. For the H1-2 humanoid with 27 actuated joints we use

$$x = \begin{bmatrix} q \\ v_{\text{ext}} \end{bmatrix},$$

with the following definitions and dimensions:

$$q = \begin{bmatrix} p_b^\top & q_b^\top & q_j^\top \end{bmatrix}^\top \in \mathbb{R}^{7+27} = \mathbb{R}^{34} \tag{3.6}$$

where $p_b \in \mathbb{R}^3$ is the base position, $q_b \in \mathbb{R}^4$ is the unit quaternion (base orientation), and $q_j \in \mathbb{R}^{27}$ are the joint positions.

$$v_{\text{ext}} = \begin{bmatrix} v_b^\top & v \end{bmatrix}^\top \in \mathbb{R}^{6+27} = \mathbb{R}^{33} \tag{3.7}$$

where $v_b = \begin{bmatrix} v_{\text{lin}}^\top & \omega^\top \end{bmatrix}^\top \in \mathbb{R}^6$ represents the linear and angular base velocities, and $v \in \mathbb{R}^{27}$ are the joint velocities.

$$u \in \mathbb{R}^{27} \tag{3.8}$$

where u represents the joint inputs.

The discrete-time model (3.5) evolves a 67-dimensional state ($\dim q = 34$, $\dim v_{\text{ext}} = 33$) under 27 inputs, with the floating-base orientation represented by a unit quaternion.

A key strength of MuJoCo lies in its efficient formulation of constraints, which leads to exceptional computational performance. Contact constraints are solved via convex optimization rather than brute-force combinatorial search, avoiding exponential complexity. By exploiting problem structure and using optimized solvers, MuJoCo achieves real-time simulation speeds for complex robots on standard hardware. It was explicitly engineered for online control and is implemented in C/C++ with careful low-level optimizations, including optional multi-threading.

This enables simulation rates of thousands of physics steps per second for moderately complex models. For example, Todorov *et al.* [26] demonstrate that their system can perform on the order of 4×10^5 dynamics evaluations per second on a 12-core CPU when simulating a humanoid model with 18 degrees of

freedom and multiple active contacts These values refer solely to the dynamic computation, excluding the optimization involved in MPC.

The engine also provides deterministic results given fixed timestep and solver settings, which is essential for debugging and reproducibility in research.

Another distinctive feature of MuJoCo is its native differentiability, which makes it particularly suitable for optimal control and planning. The simulator can compute analytic derivatives of the dynamics, providing Jacobians of state evolution with respect to states and control inputs, as well as inverse dynamics and contact sensitivities. This capability stems from MuJoCo's fluid and differentiable contact model and its internal analytical calculations, which enable derivative calculations for gradient-based and Hessian-based optimisers. Tassa et al. [22] leveraged MuJoCo's analytic derivatives to implement MPC on a humanoid robot via iLQG, achieving complex behaviors such as getting up and balancing in near real time. In their work, MuJoCo's fast derivative computations were crucial to accelerate the optimization of control sequences at each time step. This makes MuJoCo a preferred choice for model-predictive and optimization-based control of humanoids, where accurate and efficient gradient information is required.

In addition to its speed and stability, MuJoCo provides fine-grained control over simulation parameters and outputs. Researchers can easily access joint forces, contact points and normals, and other internal physics data, which facilitates controller design and analysis.

In conclusion, the need for accurate, differentiable, and computationally efficient dynamics makes MuJoCo the most suitable platform for implementing whole-body model-based control on the Unitree H1-2.With the dynamic model defined in the MuJoCo environment, we now introduce the nonlinear model predictive control framework that will be used to compute whole-body control actions.

3.3 Nonlinear Model Predictive Control (NMPC)

The problem of automatic control consists in determining a control law that, based on the measurements of the state of a dynamical system, produces inputs that drive the system's evolution toward a desired behavior. In the simplest cases, for linear and stable systems, control can be designed using classical methods based on linear feedback and frequency analysis. However, when constraints on inputs and states must be satisfied or when the desired behavior varies over time, it becomes necessary to solve an optimization problem to determine the most suitable control action.

Model Predictive Control (MPC) arises as a response to this need. Introduced in the late 1960s and developed during the 1970s in industrial applications, MPC was among the first model-based control schemes to use predictions of the system evolution to select the current input [1]. In early developments, a linear model of the process and a quadratic cost were assumed.This led to solving at each instant

a Linear–Quadratic problem with linear constraints on inputs and states. This formulation achieved great success in industrial applications thanks to its ability to incorporate constraints that are difficult to handle with classical controllers.

Over time, the idea was extended to cases in which either the dynamics or the cost function is nonlinear, as occurs in robotics and in more complex mechanical systems. In this context, we consider discrete-time state dynamics of the form

$$x_{i+1} = f(x_i, u_i), \tag{3.9}$$

with state $x_i \in \mathbb{R}^n$ and input $u_i \in \mathbb{R}^m$. Given an initial state x_0, the predictive controller determines the sequence of control inputs

$$U = \{u_0, u_1, \ldots, u_{N-1}\}$$

that minimizes the finite-horizon cumulative cost,

$$J(x_0, U) = \sum_{i=0}^{N-1} \ell(x_i, u_i) + \ell_f(x_N), \tag{3.10}$$

where ℓ is the stage cost and ℓ_f is the terminal cost.

We consider a stage cost

$$\ell(x, u) \;=\; \sum_{i=1}^{M} w_i \, n_i\big(r_i(x, u)\big), \tag{3.11}$$

where each term in the summation includes:

- a non-negative weight w_i defining its relative importance;
- a norm $n_i(\cdot)$, twice differentiable and with a minimum at 0;
- a *residual* $r_i(x, u)$, whose norm is small when the task is satisfied.

The principle underlying MPC is that of the *receding horizon*: at each instant k, problem (3.10) is solved starting from the measured state x_k, obtaining a sequence of control inputs $U_k^* = \{u_k^*, \ldots, u_{k+N-1}^*\}$. Only the first command u_k^* of this sequence is applied, then the new state x_{k+1} is measured, and the procedure is repeated on the new horizon. This structure generates a feedback control capable of adapting to model variations, external disturbances, and complex constraints.

In linear systems with quadratic costs and without constraints, the problem admits an analytical solution and can be reduced to the classical LQR control. When either the dynamics or the cost function is nonlinear, the problem becomes non-convex and must be solved numerically, often with approximations that balance accuracy and computation time, as discussed by Rawlings, Mayne, and Diehl [1]. This is of particular interest in applications such as humanoid robot locomotion, where the dynamics and cost functions are strongly nonlinear.

It is therefore necessary to solve, in real time, a finite-horizon optimal control problem. The goal of MPC is not to compute the entire optimal control law, but to provide a numerical approximation

$$u_k^*(x_k),$$

that is, the first command of the optimal sequence for the current state x_k. This approximation is affected by several sources of error.

As discussed by Rawlings *et al.* [1], the feedback law derived from the optimal control problem is generally only approximate. This is a consequence of multiple factors, such as:

(i) the plant model inevitably differs from the true system dynamics;

(ii) the optimization horizon must be truncated to a finite length;

(iii) continuous-time dynamics require discretization before prediction;

(iv) numerical solvers introduce errors because the problem cannot be solved exactly.

It is neither possible nor always desirable to solve (3.10) with extremely high precision. The objective is instead to obtain an updated control command rapidly, ensuring that the numerical error remains smaller than the other uncertainties present in the model and in the overall problem formulation.

In linear systems with quadratic costs, methods based on Riccati equations provide the exact solution in very short computation times. In nonlinear systems, numerical optimization methods are employed instead, typically based on first- or second-order linearizations, or on derivative-free methods, which determine an approximate solution.

3.4 Optimization Techniques for MPC

Algorithmic choices have a decisive impact on the controller's performance: different methods can exhibit very different computational costs and reliability, even when achieving the same level of accuracy. In nonlinear MPC, it is therefore essential to employ methods that converge rapidly to an acceptable solution, even without fully refining it, in order to preserve real-time responsiveness. MuJoCo provides several optimisation algorithms, among which iLQG, gradient-based methods, and predictive sampling are the most commonly used in the literature. Their mathematical formulation is analysed below.

3.4.1 Gradient Descent

Given the nominal control sequence computed at the previous MPC step and the currently observed state x_0, the system dynamics are simulated forward to obtain the corresponding nominal trajectories.

$$X^{\text{old}} = \{x_0^{\text{old}}, x_1^{\text{old}}, \dots, x_N^{\text{old}}\}, \qquad U^{\text{old}} = \{u_0^{\text{old}}, u_1^{\text{old}}, \dots, u_{N-1}^{\text{old}}\}.$$

In the gradient descent method proposed by Taylor *et al.* (as in [37]), the sequence of control commands is not optimized directly; instead, it is described by a parametric function, typically a spline:

$$U(t) = \Pi(t; \theta) \tag{3.12}$$

where the vector θ collects the control policy parameters, i.e., the values (or coefficients) of the spline at the temporal knots.

Splines are piecewise polynomial functions that provide a compact and smooth representation of the temporal evolution of the control inputs. The knots are the points at which the polynomial segments meet, and by specifying the command values at these knots, the spline interpolates or approximates the entire sequence of controls. In this way, instead of optimizing each individual input u_t, the optimization acts only on a reduced number of parameters $\theta = \{\theta_0, \theta_1, \ldots, \theta_K\}$, in this way decreasing the dimensionality of the optimization problem and producing smoother control actions.

Given a cost functional $J(\theta)$, the goal is to minimize J with respect to the parameter vector θ.

To apply gradient descent, one must compute

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial U} \frac{\partial U}{\partial \theta} \tag{3.13}$$

where the derivative $\partial U / \partial \theta$ depends on the analytical form of the spline, while $\partial J / \partial U$ is obtained by solving the adjoint equations derived from Pontryagin's Maximum Principle through a backward pass:

$$\lambda_t = \frac{\partial l(x_t, u_t)}{\partial x_t} + \left(\frac{\partial f(x_t, u_t)}{\partial x_t} \right)^T \lambda_{t+1} \tag{3.14}$$

$$\frac{\partial J}{\partial u_t} = \frac{\partial l(x_t, u_t)}{\partial u_t} + \left(\frac{\partial f(x_t, u_t)}{\partial u_t} \right)^T \lambda_{t+1} \tag{3.15}$$

Once the gradient has been computed, the policy parameters are updated iteratively according to the rule

$$\theta^{(k+1)} = \theta^{(k)} - \alpha \frac{\partial J}{\partial \theta} \tag{3.16}$$

where $\alpha > 0$ is the step size. In practice, at each iteration it is common to perform a parallel *line search* over several values of α and select the one that yields the greatest cost reduction.

This method requires only first-order derivatives and performs the search in a reduced-dimensional parameter space thanks to the use of splines, rather than over the entire control sequence U. This approach enables fast optimization, although it generally achieves lower performance in terms of control quality.

Algorithm 1: MPC with Gradient Descent on policy parameters θ

Input: measured initial state x_0, previous nominal control sequence U^{old}
Roll out U^{old} from x_0 to compute the nominal state trajectory X^{old}.
Represent the nominal control sequence U^{old} through a set of policy parameters θ
Compute the gradient via chain rule:

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial \Pi} \frac{\partial \Pi}{\partial \theta},$$

where $\frac{\partial J}{\partial \Pi}$ is obtained from Pontryagin's Maximum Principle backward pass and $\frac{\partial \Pi}{\partial \theta}$ from the policy parametrization.
Generate a candidate set of step sizes $\{\alpha_1, \alpha_2, \dots\}$;
for *each* α_j *(in parallel)* **do**
  Form the candidate parameters using the gradient step:

$$\theta^{(\alpha_j)} \leftarrow \theta - \alpha_j \frac{\partial J}{\partial \theta}.$$

Select the best step $\alpha^\star = \arg\min_{\alpha_j} J^{(\alpha_j)}$ and update:

$$\theta \leftarrow \theta - \alpha^\star \frac{\partial J}{\partial \theta}.$$

Recover the new control sequence $U = \Pi(t; \theta)$
Updated nominal plan and apply the first control input u_0 to the real system;

3.4.2 iLQG Algorithm

Given the nominal control sequence computed at the previous MPC step and the currently observed state x_0, the system dynamics are simulated forward to obtain the corresponding nominal trajectories.

$$X^{\text{old}} = \{x_0^{\text{old}}, x_1^{\text{old}}, \dots, x_N^{\text{old}}\}, \qquad U^{\text{old}} = \{u_0^{\text{old}}, u_1^{\text{old}}, \dots, u_{N-1}^{\text{old}}\}.$$

The iLQG optimization proceeds in two main phases:

1. Linearization and Backward Pass. To analyze how the cost-to-go changes around the nominal trajectory $\{x_i^{\text{old}}, u_i^{\text{old}}\}$, we introduce the local deviations

$$\delta x_i = x_i - x_i^{\text{old}}, \qquad \delta u_i = u_i - u_i^{\text{old}}.$$

According to the principle of dynamic programming (Bellman), at time i, the optimal control minimizes

$$V(x_i) = \min_{u_i} \big[\, \ell(x_i, u_i) + V(f(x_i, u_i),\, i+1) \,\big], \tag{3.17}$$

that is, the immediate cost plus the future cost starting from the next state.

We therefore define the function

$$\begin{aligned} Q(\delta x_i, \delta u_i) = \ell\big(x_i^{\text{old}} + \delta x_i, u_i^{\text{old}} + \delta u_i,\, i\big) - \ell\big(x_i^{\text{old}}, u_i^{\text{old}},\, i\big) \\ + V\big(f(x_i^{\text{old}} + \delta x_i, u_i^{\text{old}} + \delta u_i),\, i{+}1\big) \\ - V\big(f(x_i^{\text{old}}, u_i^{\text{old}}),\, i{+}1\big). \end{aligned} \tag{3.18}$$

The function Q measures the variation of the total cost when, at time i, starting from $(x_i^{\text{old}}, u_i^{\text{old}})$, a perturbation $(\delta x_i, \delta u_i)$ is applied.

Expanding (3.18) in a second-order Taylor series yields

$$Q(\delta x_i, \delta u_i) \;\approx\; \frac{1}{2} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix}^{\top} \begin{bmatrix} 0 & Q_x^{\top} & Q_u^{\top} \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix}. \tag{3.19}$$

The coefficients of the expansion are given by

$$Q_x = \ell_x + f_x^{\top} V_x', \tag{3.20}$$

$$Q_u = \ell_u + f_u^{\top} V_x', \tag{3.21}$$

$$Q_{xx} = \ell_{xx} + f_x^{\top} V_{xx}' f_x + V_x' \cdot f_{xx}, \tag{3.22}$$

$$Q_{uu} = \ell_{uu} + f_u^{\top} V_{xx}' f_u + V_x' \cdot f_{uu}, \tag{3.23}$$

$$Q_{ux} = \ell_{ux} + f_u^{\top} V_{xx}' f_x + V_x' \cdot f_{ux}. \tag{3.24}$$

Here, $\ell_x, \ell_u, \ell_{xx}, \ell_{uu}, \ell_{ux}$ denote the first and second derivatives of the stage cost $\ell(x, u, i)$, while $f_x, f_u, f_{xx}, f_{uu}, f_{ux}$ are the derivatives of the system dynamics $f(x, u)$. V_x' and V_{xx}' represent the first and second derivatives of the value function $V(x, i{+}1)$ evaluated at the next time step.

In *iterative LQR* (iLQG) methods, tensor terms such as $V_x' \cdot f_{xx}$, $V_x' \cdot f_{ux}$, and $V_x' \cdot f_{uu}$ are typically neglected. This simplification corresponds to a Gauss–Newton approximation. In contrast, Differential Dynamic Programming (DDP) retains these terms, resulting in a descent direction that better captures the full curvature of the problem.

Minimizing the quadratic function (3.19) with respect to δu_i, for a fixed δx_i, yields the optimal correction law.

Indeed, considering $Q(\delta x_i, \delta u_i)$ as a quadratic function in δu_i,

$$Q(\delta x_i, \delta u_i) = Q_u^{\top} \delta u_i + \tfrac{1}{2}\, \delta u_i^{\top} Q_{uu}\, \delta u_i + \delta x_i^{\top} Q_{xu}\, \delta u_i + (\text{terms independent of } \delta u_i),$$

the stationarity condition for a minimum is

$$\frac{\partial Q}{\partial \delta u_i} = Q_u + Q_{uu}\, \delta u_i + Q_{ux}\, \delta x_i = 0.$$

Solving for δu_i gives

$$\delta u_i^* = \arg\min_{\delta u_i} Q(\delta x_i, \delta u_i) = -\, Q_{uu}^{-1}\big(Q_u + Q_{ux}\, \delta x_i\big), \tag{3.25}$$

which explicitly shows that the local optimal control depends linearly on the deviation δx_i. By separating the constant and proportional terms in δx_i, we introduce the feedforward and feedback gains:

$$k_i = -\,Q_{uu}^{-1}\,Q_u, \tag{3.26}$$

$$K_i = -\,Q_{uu}^{-1}\,Q_{ux}, \tag{3.27}$$

which are used in the forward pass to update the control plan.

2. Forward Pass. Given the measured initial state $\hat{x}_0$, the new trajectory is computed as

$$\hat{u}_i = u_i^{\text{old}} + \alpha\, k_i + K_i\left(\hat{x}_i - x_i^{\text{old}}\right), \tag{3.28}$$

$$\hat{x}_{i+1} = f\left(\hat{x}_i,\, \hat{u}_i\right), \tag{3.29}$$

where $0 < \alpha \leq 1$ is the *line-search* parameter, used to scale the control update so as to ensure a decrease in the cost when generating the new control sequence.

Algorithm 2: MPC with iLQG and Line Search

Input: measured initial state x_0, previous nominal control sequence U^{old}

Roll out U^{old} from x_0 to compute the nominal state trajectory X^{old}.

Linearize the system dynamics and the cost function around the nominal trajectory $(X^{\text{old}}, U^{\text{old}})$.

Compute the local quadratic approximation of the value function and propagate backward from the final stage to obtain:

$$k_i, \quad K_i, \qquad i = 0, \ldots, N-1,$$

where k_i is the feedforward term and K_i is the feedback gain.

Generate a candidate set of step sizes $\{\alpha_1, \alpha_2, \ldots\}$;

for *each* α_j *(in parallel)* **do**

Form the candidate control sequence:

$$\hat{u}_i^{(\alpha_j)} = u_i^{\text{old}} + \alpha_j\, k_i + K_i\big(\hat{x}_i^{(\alpha_j)} - x_i^{\text{old}}\big), \quad i = 0, \ldots, N-1.$$

Propagate the system dynamics forward to obtain its cost $J^{(\alpha_j)}$.

Select the best step size:

$$\alpha^\star = \arg\min_{\alpha_j} J^{(\alpha_j)}.$$

Using $\alpha^\star$, update the control input:

$$\hat{u}_i = u_i^{\text{old}} + \alpha^\star k_i + K_i\big(\hat{x}_i - x_i^{\text{old}}\big), \quad i = 0, \ldots, N-1.$$

Updated nominal plan and apply the first control input u_0 to the real system;

It is important to point out that, although this algorithm is typically referred to as iLQG in the MuJoCo community, Howell *et al.* [37] clarify in a footnote that the procedure used in this setting is effectively identical to iLQR. In particular, the authors remark that the noise–sensitivity component that distinguishes iLQG from iLQR, originally introduced in [25], is not employed in their formulation. For historical reasons, however, they retain the name "iLQG".

3.4.3 Predictive Sampling

Given the nominal control sequence computed at the previous MPC step and the currently observed state x_0, the system dynamics are simulated forward to obtain the corresponding nominal trajectories.

$$X^{\text{old}} = \{x_0^{\text{old}}, x_1^{\text{old}}, \dots, x_N^{\text{old}}\}, \qquad U^{\text{old}} = \{u_0^{\text{old}}, u_1^{\text{old}}, \dots, u_{N-1}^{\text{old}}\}.$$

The control inputs are represented in parametric form by means of a spline function, whose nodal parameters are collected in the vector θ. Predictive sampling provides a derivative-free optimization method that explores a local policy space by sampling variations around the nominal control, perturbing the control variables with Gaussian noise.

Given a standard deviation $\sigma > 0$ and a predefined number of rollouts, a set of candidate control actions is generated. For each candidate, the system dynamics are simulated as

$$x_{t+1}^{(i)} = f\big(x_t^{(i)}, u_t^{(i)}\big),$$

the finite-horizon cost is evaluated, and the best control sequences are selected by weighting them according to their rollout cost-trajectories with lower cost contribute more significantly. At the end of the iteration, a new control sequence U is obtained, approximating the set of optimal future actions at the current step.

Because this approach is inherently sample-based and easily parallelizable, it proves particularly effective in the presence of discontinuities or strong nonlinearities arising from contact dynamics. In practical applications, it often shows superior interaction capabilities in complex or cluttered environments compared to deterministic, derivative-based methods such as iLQG. However, due to its stochastic nature, the resulting control may lack regularity and periodicity in tasks that require cyclic and repeatable behaviors (for instance, walking on flat terrain), where gradient-based methods tend to produce more stable and consistent gaits.

The main advantage of predictive sampling lies in the absence of derivatives, which generally makes it faster than other optimization algorithms. The possibility of parallelizing the computation of future trajectories plays a crucial role, particularly in this algorithm, where MuJoCo's simulation speed and CPU-level parallelization allow the evaluation of a large number of rollouts per iteration. GPU-based simulators, such as *Isaac Gym/Sim*, have demonstrated the potential

to further scale the number of rollouts by exploiting parallelization on graphical accelerators.

The MPC implementation currently distributed with MJPC uses the classic CPU-based MuJoCo simulation engine. A more recent version, named MJX (DeepMind, 2024), reimplements MuJoCo to leverage GPU acceleration for faster simulation. This variant is currently employed mainly for training neural networks and reinforcement learning policies, but it has not yet been integrated into MJPC for online MPC applications.

A further aspect discussed in the community concerns the numerical discrepancies observed between the original MuJoCo and MJX. Yuval Tassa, one of MuJoCo's principal developers, pointed out publicly (April 2024) that the collision–handling routines in MJX rely on a different convex–collision implementation compared with MuJoCo. He also emphasized that, even if identical algorithms were used, differences would still emerge because GPU computations typically rely on single–precision arithmetic, whereas CPU implementations use double precision.

This remark underscores that GPU-based simulation affects not only computational speed but also numerical behavior, due both to variations in algorithmic implementation and to the reduced floating-point precision on GPUs (float32 instead of float64).

Algorithm 3: Predictive Sampling MPC (derivative-free)

Input: measured initial state x_0, previous nominal control sequence U^{old},noise scale σ

Roll out U^{old} from x_0 to compute the nominal state trajectory X^{old}.

Generate N candidate control sequences by adding Gaussian noise to the nominal plan U^{old}.

for *each rollout* $i = 1$ *to* N *(in parallel)* **do**

  Simulate the system dynamics forward from x_0 using candidate control sequence $U^{(i)}$;

  Compute the finite-horizon cost for this simulated trajectory;

Select the candidate control sequences with the lowest costs and set them as the new nominal plan U^{old};

Apply the first control input of the updated plan to the real system;

3.5 Risk-Sensitive Optimization in MJPC

Equations (3.10)–(3.24) described earlier are derived for a standard stage cost

$$\ell(x, u)$$

as in (3.11). However, the MJPC framework does *not* directly use the base cost function $\ell(x, u)$, but rather a *risk-sensitive* transformation of the cost, denoted $c(x, u)$. This modification replaces $\ell(x, u)$ with $c(x, u)$ everywhere, thereby altering both the gradient and the Hessian.

The stage cost in MJPC is defined as

$$c(x, u, R) \;=\; \frac{e^{\,R\,\ell(x,u)} - 1}{R}, \tag{3.30}$$

where the scalar parameter R determines the degree of risk sensitivity:

$$R < 0 \;\text{ (risk-seeking)} \qquad R = 0 \;\text{ (neutral)} \qquad R > 0 \;\text{ (risk-averse)}$$

This transformation enjoys several useful properties: it is smooth for all R, it reduces to ℓ as $R \to 0$, it is monotonic in ℓ, and for $R < 0$ it is upper-bounded as $c(x, u, R) < -1/R$. These characteristics ensure continuity with the classical case when $R \to 0$.

From definition (3.30), denoting $\ell_x = \partial\ell/\partial x$ and $\ell_u = \partial\ell/\partial u$, the gradient is obtained as:

$$c_x(x, u) = \frac{\partial c}{\partial x} = e^{R\,\ell(x,u)}\;\ell_x(x, u), \tag{3.31}$$

$$c_u(x, u) = \frac{\partial c}{\partial u} = e^{R\,\ell(x,u)}\;\ell_u(x, u). \tag{3.32}$$

If the base function ℓ has the form (3.11) then its derivatives are

$$\ell_x = \sum_{i=1}^{M} w_i\,\frac{\partial n_i}{\partial r_i}\,\frac{\partial r_i}{\partial x}, \qquad \ell_u = \sum_{i=1}^{M} w_i\,\frac{\partial n_i}{\partial r_i}\,\frac{\partial r_i}{\partial u},$$

where the derivatives of the residuals r_i with respect to x, u are computed analytically within MJPC.

An important point highlighted by Taylor Howell *et al.* [37] is that the computational complexity of computing Jacobians via finite differences grows with the dimensionality of the input (x, u), but not with that of the output. In other words, adding additional sensors contributing to the residuals r_i does not significantly increase computation time. This result is particularly relevant for the integration of exteroceptive sensors, since reconstructing a point cloud requires adding a large number of sensory rays.

Naturally, the Hessian terms are also affected by this transformation. Using the Gauss–Newton approximation, all higher-order terms are neglected except for:

$$c_{xx}(x, u) \approx e^{R\,\ell(x,u)}\left[\sum_{i=1}^{M} w_i\,\frac{\partial r_i}{\partial x}^{\top}\frac{\partial^2 n_i}{\partial r_i^2}\frac{\partial r_i}{\partial x} + R\,\ell_x^{\top}\ell_x\right], \tag{3.33}$$

$$c_{uu}(x, u) \approx e^{R\,\ell(x,u)}\left[\sum_{i=1}^{M} w_i\,\frac{\partial r_i}{\partial u}^{\top}\frac{\partial^2 n_i}{\partial r_i^2}\frac{\partial r_i}{\partial u} + R\,\ell_u^{\top}\ell_u\right], \tag{3.34}$$

$$c_{xu}(x, u) \approx e^{R\,\ell(x,u)}\left[\sum_{i=1}^{M} w_i\,\frac{\partial r_i}{\partial x}^{\top}\frac{\partial^2 n_i}{\partial r_i^2}\frac{\partial r_i}{\partial u} + R\,\ell_x^{\top}\ell_u\right]. \tag{3.35}$$

The only difference from the classical case (based on ℓ) is the multiplicative factor $e^{R\ell}$ and the additional terms, which represent the effect of the exponential risk transformation.

Since the gradient and Hessian of $c(x, u)$ largely preserve the structure and computational complexity of those of $\ell(x, u)$, the computational impact of adding risk sensitivity is limited.

From a practical standpoint, the parameter R modulates the optimizer's behavior with respect to states or actions that yield high costs.

- For $\boldsymbol{R > 0}$ (the *risk-averse* regime), the transformation (3.30) amplifies the values of $\ell(x, u)$: for the same residual, the cost $c(x, u)$ is greater than in the classical case. Consequently, the optimizer penalizes large residuals more strongly, favoring trajectories that reduce the risk of entering costly or unstable states.

- For $\boldsymbol{R = 0}$ (the *risk-neutral* regime), the cost coincides with the standard $\ell(x, u)$, and the optimization behaves identically to the classical case.

- For $\boldsymbol{R < 0}$ (the *risk-seeking* regime), the transformation (3.30) attenuates the effect of large $\ell(x, u)$ values, which remain bounded from above. The optimizer thus becomes more tolerant of large residuals, favoring aggressive and more exploratory strategies. However, this property may be undesirable in the presence of very large residuals, since the planner is no longer strongly encouraged to reduce them.

Although the risk–sensitivity parameter R was actively used during controller tuning to explore different planner behaviors, for the final simulation results the parameter was set to zero, corresponding to a risk–neutral regime.

Chapter 4

Experimental setup

In this chapter, we describe the configuration of the *MuJoCo Model Predictive Control* (MJPC) framework adopted to evaluate the proposed strategies for humanoid locomotion and obstacle negotiation. The chapter provides a comprehensive overview of the simulation environment, the robot modeling choices, and the methodological foundations that support the experimental analysis.

We begin by presenting the dynamic model of the Unitree H1-2 humanoid robot and its actuation scheme, which define the basis for the subsequent control design. The following sections describe the sensing configuration implemented in the simulator to reproduce realistic proprioceptive and exteroceptive feedback, as well as the modeling of environmental geometries and obstacles. Particular attention is devoted to the representation of mesh complexity and collision handling, which are crucial to ensuring both physical consistency and real-time performance within MuJoCo.

Subsequently, we introduce the cost function formulations used to define locomotion and obstacle-negotiation tasks, highlighting the differences between standard walking behaviors and extended scenarios that incorporate visual perception. Finally, we will examine selected studies from the recent literature that have used sim-to-real via MJPC.

4.1 MuJoCo Setup

4.1.1 Unitree H1 Robot Model in Simulation

The humanoid Unitree H1-2 is simulated in *MuJoCo* using the model introduced in Chapter 3, where joint limits and physical constraints are defined. The model is based on the official Unitree MuJoCo repository, which provides the file `unitree_robots/h1_2/h1_2_handless.xml` containing accurate kinematic and dynamic specifications, including meshes, link masses, inertia tensors, and joint definitions [46].

All robot joints are actuated using position-controlled actuators defined via the `<position>` elements. In the *MuJoCo* simulation framework, such actuators

apply torques according to a proportional-derivative (PD) control law:

$$\tau = K_p(q_{\text{ref}} - q) - K_v\dot{q} \tag{4.1}$$

where K_p and K_v are the proportional and derivative gains, respectively. This type of actuator allows each joint to behave as a compliant position servo.

The selection of the actuator gains K_p and K_v was guided by the need to achieve stable and responsive joint behavior consistent with the robot's dynamics. The final set of control gains was adopted from an open-source model available online [47], which provides physically consistent control parameters.

Proximal joints such as the hips and knees were assigned higher proportional gains (kp = 200–300) to support body weight and maintain upright posture, whereas distal joints (ankles and wrists) were assigned lower gains (kp = 40–100) to ensure smoother, more compliant motion and avoid excessive oscillations. Derivative gains (kv = 2–6) were chosen to achieve near-critical damping, minimizing overshoot and ensuring stable transient responses during locomotion and manipulation tasks.

Each body segment in the model contains an `<inertial>` tag that specifies its mass, center of mass offset, and inertia tensor. This enables MuJoCo to perform accurate forward and inverse dynamics computations. Consequently, the Unitree H1-2 model reproduces the rigid-body dynamics and actuation properties of the physical robot with high fidelity, ensuring that the simulated behaviors remain physically consistent and dynamically plausible. The configuration of simulated sensors and task residuals was inspired by the work of Meser *et al.* [7] and Howell *et al.* [37]. The model integrates a set of proprioceptive and exteroceptive sensors, including joint position and velocity sensors and multiple virtual rangefinder sensors (like LiDAR) distributed around the head to simulate environmental perception. The inclusion of these sensing modalities enables a more realistic perception action loop, facilitating both balance control and navigation within the simulation environment.

4.1.2 LiDAR and other sensors

MuJoCo does not provide a native LiDAR sensor model. To emulate this functionality, a custom sensing system was implemented using multiple `rangefinder` sensors, each representing a single laser beam directed along its local positive z-axis. Each rangefinder measures the distance between its origin (defined `site`) and the nearest object in the scene along that direction. By placing multiple rangefinder sites around the robot's head with different orientations, it is possible to approximate a LiDAR-like perception system capable of producing a sparse 3D point cloud of the surrounding environment [48].

In practice, each rangefinder was associated with a dedicated `<site>` element in the XML model. The orientation of each site was defined through Euler angles, which specify the local rotation required to align the site's positive z-axis with the desired sensing direction. Mathematically, the direction of each beam in

the world frame is obtained from the third column of the rotation matrix R corresponding to the site:

$$\mathbf{P} = \mathbf{O} + d \cdot \hat{\mathbf{z}}, \tag{4.2}$$

where $\mathbf{O}$ is the site's position, d is the measured distance, and $\hat{\mathbf{z}} = [R_{0,2},\ R_{1,2},\ R_{2,2}]^\top$ is the unit vector representing the local z-axis expressed in world coordinates. This formulation allows each sensor to project its measurement into the global frame, enabling the reconstruction of a point cloud from all active rangefinders.

However, the LiDAR sensor described in Chapter 3 (Livox Mid-360) cannot be faithfully reproduced in MuJoCo for two main reasons. First, the real sensor generates a non-repetitive point cloud pattern: its scanning geometry changes continuously according to an internal motion law that is not publicly documented. Second, the real device can acquire up to approximately 200,000 points per second [49], a data rate far beyond the capabilities of the MuJoCo simulation engine. For this reason, a simplified representation was adopted, in which the perception module generates a reduced and fixed set of range measurements, resulting in a narrower field of view compared to the real LiDAR. Nevertheless, this abstraction preserves the spatial structure of the sensor's perception while maintaining real-time simulation performance.

Figure 4.1 shows the simulated Unitree H1-2 equipped with the virtual rangefinder array in MuJoCo, while Figure 4.2 illustrates the corresponding reconstructed point cloud obtained from the simulated vision system, operating at the same update frequency as the controller (approximately 67 Hz).

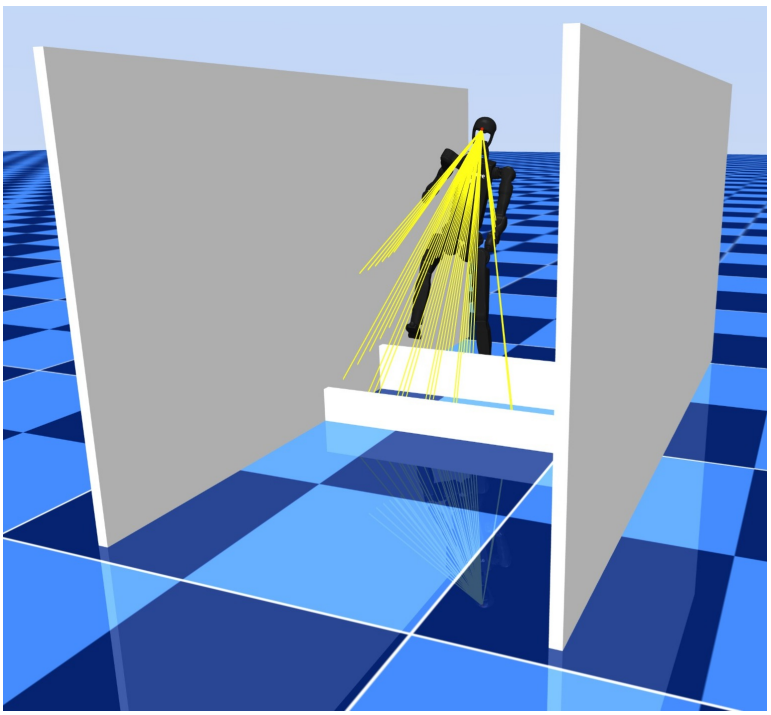


Figure 4.1: Simulated Unitree H1-2 model equipped with rangefinder-based LiDAR sensors in MuJoCo.

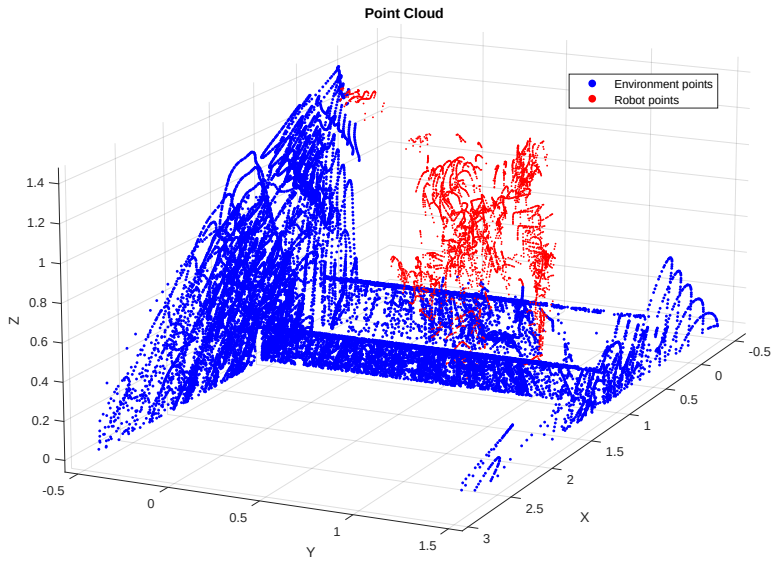


Figure 4.2: Point cloud reconstruction from the simulated LiDAR perception system. The system operates synchronously with the controller at approximately 67 Hz.

This approach provides a computationally efficient method for modeling exteroceptive perception in MuJoCo. It maintains sufficient perceptual realism for testing control and navigation algorithms while remaining compatible with real-time execution.

In addition to the rangefinder-based LiDAR system described above, several proprioceptive sensors have been defined in the MuJoCo model to provide information on the robot's internal state. These sensors are used to estimate kinematic and dynamic quantities such as body positions, orientations, and velocities, which are essential for feedback control and state estimation within the simulation framework. Table 4.1 summarizes all sensor types implemented in the model, along with their physical meaning and associated robot components.

Table 4.1: Summary of proprioceptive sensors used in the Unitree H1-2 MuJoCo model.

Sensor type	**Description**	**Measured object**
`framepos`	Measures the 3D position of a body or site in the world frame.	Torso, pelvis, feet, head
`framezaxis`	Returns the local z-axis unit vector (typically representing the "up" direction).	Pelvis, feet
`framexaxis`	Returns the local x-axis unit vector (typically representing the "forward" direction).	pelvis, feet
`framelinvel`	Measures the linear velocity of a body in world coordinates.	right and left foot
`subtreecom`	Computes the position of the center of mass of a subtree.	Torso
`subtreelinvel`	Computes the linear velocity of a subtree's center of mass.	Pelvis, torso

These sensors collectively provide both exteroceptive and proprioceptive feedback, enabling the simulation to capture the dynamic behaviour of the humanoid robot and allowing residuals to be calculated. In addition to these sensors, to solve the NMPC, the model also includes joint-level sensors that provide direct measurements of the position and velocity of each joint. These sensors emulate the encoder available on the physical robot and are essential for state estimation.

4.1.3 Meshes in MuJoCo

Before presenting the task formulation and the associated cost functions, it is important to clarify that the robot is not the only element within the simulation environment. While flat-ground locomotion requires only a planar surface, obstacle–overcoming scenarios introduce interactions with additional bodies, such as those visible in Figure 4.1. In particular, for tasks involving contact with walls through arm support, simple geometric primitives (e.g., `box` geometries) were included in the XML model. MuJoCo natively provides a set of elementary shapes that can be directly embedded in the scene and used for collision simulation.

Beyond the meshes defining the robot's kinematic structure, it is sometimes necessary to incorporate more complex geometries derived from CAD models. These may represent real-world environments or obstacles the robot must traverse. To evaluate locomotion in uneven and constrained industrial environments, we considered the interior of an aircraft fuselage, featuring horizontal beams acting as obstructions. The oval cross-section of the fuselage created a valuable challenge to assess balance, foot placement, and trajectory stability.

As highlighted by Todorov *et al.* [26], however, non-convex meshes, although they can be displayed, are not suitable for collision detection; the authors note that users should instead break such geometries into convex components. Following this guideline, we experimented with importing STL models representing the fuselage interior, discovering two fundamental limitations. First, MuJoCo did not correctly recognize the interior as an empty volume, resulting in undesired collisions between the humanoid and the environment. Second, the high complexity of the mesh substantially increased the rendering effort, forcing the simulator to drop the real-time factor to approximately 0.025, thus slowing the simulation by nearly a factor of forty.

To overcome these issues, the fuselage geometry was reconstructed as an *obround* cross–section, where the circular segments were approximated through a set of small, appropriately oriented `box` primitives. This decomposition enabled an accurate and computationally efficient approximation of the fuselage shape, allowing reliable collision detection and maintaining real-time simulation performance. A representation of the fuselage approximation is shown in Figure 4.3. In this figure, the points extracted from the `STL` model are compared with the reconstruction obtained by means of parallelepiped blocks. The reconstruction consists of 62 rectangular blocks. The cross-section of the model can be interpreted as two semicircles with a radius of 1.4 m connected by a flat region of 0.68 m in length.

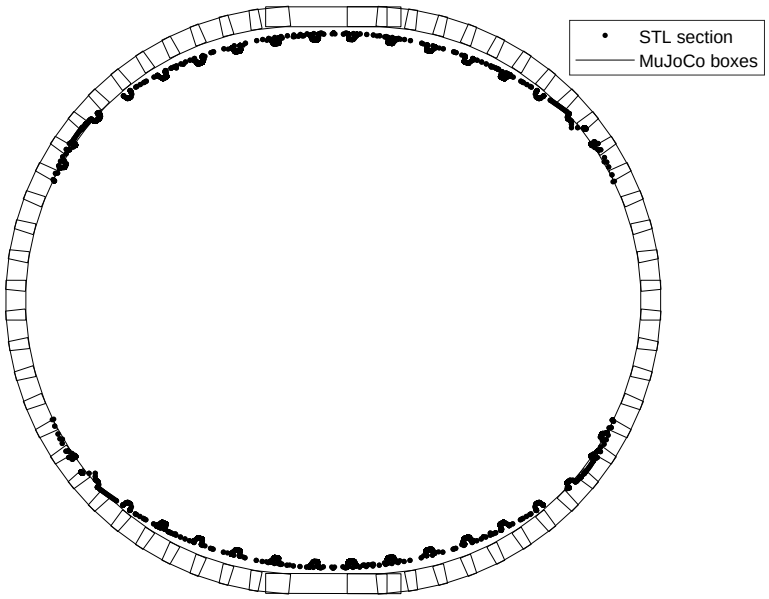


Figure 4.3: Comparison between the `STL` model points and the reconstruction using 62 parallelepiped blocks.

While this solution proved effective, it highlights a broader limitation: the integration of complex environments in MuJoCo remains challenging, particularly when accurate collision handling and real-time performance are required.

Overall, three simulation scenarios were developed: (i) locomotion on flat ground, (ii) obstacle overcoming with contact using the upper limbs, and (iii) locomotion inside a fuselage reconstructed with parallelepiped blocks.

4.2 Walking Task

With the simulation environment and robot model fully defined, the next step is to specify the control objectives that govern the humanoid's behavior. In the following sections, we formulate the cost functions and residual terms that define the locomotion task.

4.2.1 Residual Definitions for the Humanoid Walking Task

The following residuals are used to define the instantaneous cost function in (3.11), where each term $r_i(x, u)$ represents a specific objective or physical constraint of the humanoid walking behavior. Residuals r_0–r_8 are derived from the *Humanoid benchmark* of Meser *et al.* [7], while r_9–r_{11} are extensions I introduced to improve and make walking more natural.

r_0 *Torso height:* Measures the deviation of the torso's vertical position from a desired reference value

$$r_0 = z_{\text{torso}} - z_{\text{goal}}.$$

r_1 *Pelvis–feet:* This term encourages lifting the feet while walking, keeping them spatially close to the pelvis.

$$r_1 = \tfrac{1}{2}(z_{\text{left foot}} + z_{\text{right foot}}) - z_{\text{pelvis}} - 0.2.$$

r_2 *Balance:* This residual penalizes the displacement of the *capture point* from the support segment defined by the two feet. The capture point represents the predicted location on the ground where the humanoid's center of mass (CoM) would need to move to come to rest given its current velocity, and it is computed as

$$p_{\text{capture}} = p_{\text{CoM}} + \tau\, v_{\text{CoM}},$$

where p_{CoM} and v_{CoM} are respectively the CoM position and velocity, and τ is a time constant (set to 0.3 s in this implementation).

The point p_{cp} denotes the orthogonal projection of p_{capture} onto the line segment connecting the right and left foot centers. This projection effectively identifies the nearest point on the base of support.

The scalar coefficient s, referred to as the *standing factor*, scales the importance of the balance term according to the robot's torso height

$$s = \max\left[0,\ \frac{z_{\text{torso}}}{\sqrt{z_{\text{torso}}^2 + 0.45^2}} - 0.4\right], \tag{4.3}$$

This coefficient increases as the humanoid stands upright, activating the balance penalty only when the agent is in a near-standing configuration and reducing its effect when crouched.

The residual is therefore expressed as

$$r_2 = s\left(p_{\text{capture}} - p_{\text{cp}}\right),$$

so that any deviation of the capture point outside the base of support increases the balance cost, promoting stable center-of-mass placement above the feet.

The mathematical interpretation of this residual is not straightforward. For a clearer understanding of its geometric meaning, the graphical interpretation of the balance residual is illustrated in Figure 4.4.

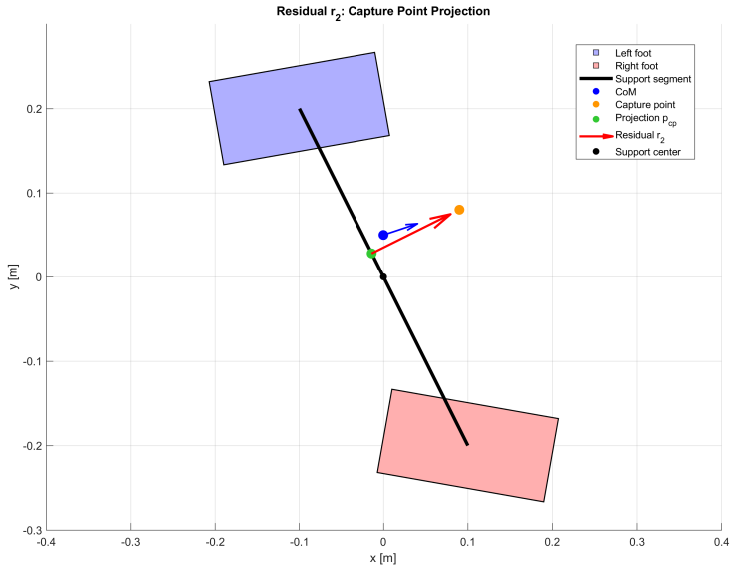


Figure 4.4: Visualization of the *balance residual* r_2. This term penalizes deviations that would lead to loss of balance, encouraging the controller to maintain the center of mass motion consistent with stable foot placement.

r_3 *Upright orientation:* This residual enforces a vertical alignment of the torso, pelvis, and both feet with respect to the global upward direction $e_z = [0,\, 0,\, 1]^\top$.

$$r_3 = \begin{bmatrix} (u_{\text{torso},z} - 1) \\ 0.3\,(u_{\text{pelvis},z} - 1) \\ 0.1\,s\,(u_{\text{right foot}} - e_z) \\ 0.1\,s\,(u_{\text{left foot}} - e_z) \end{bmatrix},$$

where u_{torso}, u_{pelvis}, $u_{\text{right foot}}$, $u_{\text{left foot}} \in \mathbb{R}^3$ denote the local "up" vectors of each corresponding body segment, and s is the standing factor defined in (4.3). The first two scalar terms ensure that the torso and pelvis remain upright, while the last two 3D terms encourage the feet to align their normal vectors with the global vertical axis.

r_4 *Posture:* This residual penalizes the deviation of the current joint configuration from a reference posture defined by a stored keyframe

$$r_4 = q_{\text{pos}} - q_{\text{key}}.$$

Here, q_{pos} represents the vector of the current joint positions, while q_{key} corresponds to the nominal standing configuration of the humanoid, in

which the robot is upright with both arms positioned close to the torso. This term promotes the maintenance of a natural and symmetric pose during locomotion, reducing unwanted joint deviations from the reference stance.

r_5 *Facing direction:* This residual encourages the humanoid to face toward the desired walking direction, promoting consistent body orientation during locomotion

$$r_5 = s\,(f_{\text{current}} - f_{\text{goal}}),$$

where $f_{\text{goal}} = [\cos\theta_{\text{goal}}, \sin\theta_{\text{goal}}]^\top$ is the unit vector pointing toward the target direction in the horizontal plane. The vector f_{current} represents the average forward orientation of the humanoid, computed as the normalized sum of the torso, pelvis, and both feet forward vectors. This formulation ensures that the entire body aligns with the intended heading rather than only a single segment.

r_6 *Velocity:* This residual drives the velocity of the humanoid's center of mass (CoM) along the current heading to match a desired forward speed.

$$r_6 = s\big(v_{\text{com}} \cdot f_{\text{current}} - v_{\text{goal}}\big),$$

where v_{com} is the planar velocity of the CoM, f_{current} is the current unit heading vector of the body, and v_{goal} is the desired forward velocity. The standing coefficient s (4.3) modulates the residual according to the upright posture of the humanoid, ensuring that the velocity objective is emphasized only when the agent is in a stable standing configuration.

r_7 *Feet motion:* This residual penalizes incoherent or asymmetric motion of the feet relative to the motion of the center of mass. It ensures that the feet move consistently with the overall locomotion of the body, and is formulated as

$$r_7 = s\big(v_{\text{com}} - 0.5(\,v_{\text{left foot}} - \,v_{\text{right foot}})\big),$$

where $v_{\text{left foot}}$ and $v_{\text{right foot}}$ are the planar linear velocities of the left and right feet, respectively. The standing coefficient s again scales the term to reduce its influence when the humanoid is crouched or in flight.

r_8 *Control :* This residual regularizes the control commands applied to the actuators, penalizing deviations from a nominal joint reference posture. It is expressed as

$$r_8 = q_{\text{ctrl}} - q_{\text{key}},$$

where q_{ctrl} denotes the vector of position-controlled actuator setpoints, and q_{key} represents the target joint configuration from the standing keyframe. This term encourages smooth and efficient control actions by limiting excessive command magnitudes and maintaining proximity to a natural reference pose.

r_9 *Foot clearance:* This residual enforces a desired vertical separation between the two feet during walking, promoting proper foot lifting during the swing phase

$$r_9 = h_{\text{clearance}} - \left| z_{\text{left foot}} - z_{\text{right foot}} \right|.$$

By maintaining an adequate height difference between the feet, the humanoid avoids dragging or scuffing, ensuring a smoother and more realistic step pattern. This term must be carefully tuned, since when the desired walking speed is zero the residual should remain constant, ensuring that both feet stay in contact with the ground in standing conditions.

r_{10} *Feet distance (3D):* This residual maintains a desired three-dimensional distance between the two feet, preventing crossing or excessive overlap of the legs during gait

$$r_{10} = \| p_{\text{right foot}} - p_{\text{left foot}} \| - d_{\text{goal}}.$$

Its inclusion improves balance and spatial consistency between steps, helping to preserve a natural stride width.

r_{11} *Knee posture:* This residual keeps both knees close to a nominal flexion angle, encouraging a slightly bent-leg stance that enhances control and yields a more human-like walking pattern

$$r_{11} = \left| q_{\text{knee,L}} - 0.15 \right| + \left| q_{\text{knee,R}} - 0.15 \right|.$$

Although not strictly necessary, this term prevents the legs from reaching fully extended joint limits, improving stability and the overall quality of motion.

Each residual $r_i(x, u)$ contributes to the overall cost in (3.11) through its weighted norm $w_i\, n_i(r_i)$, balancing stability, posture, and motion smoothness. Table (4.2) summarizes the residuals used in the walking task together with their analytical expressions and the chosen weight values.

Table 4.2: Summary of residual terms and corresponding weights for the humanoid walking task.

Name	**Formula**	**Weight** w_i
Torso height	$r_0 = z_{\text{torso}} - z_{\text{goal}}$	5.0
Pelvis–feet	$r_1 = \frac{1}{2}(z_{\text{left foot}} + z_{\text{right foot}}) - z_{\text{pelvis}} - 0.2$	1.0
Balance	$r_2 = s\,(p_{\text{capture}} - p_{\text{cp}})$	5.0
Upright orientation	$r_3 = \begin{bmatrix} (u_{\text{torso},z} - 1) \\ 0.3\,(u_{\text{pelvis},z} - 1) \\ 0.1\,s\,(u_{\text{right foot}} - e_z) \\ 0.1\,s\,(u_{\text{left foot}} - e_z) \end{bmatrix}$	5.0
Posture	$r_4 = q_{\text{pos}} - q_{\text{key}}$	1.0
Facing direction	$r_5 = s\,(f_{\text{current}} - f_{\text{goal}})$	15.0
Velocity	$r_6 = s\,(v_{\text{com}} \cdot f_{\text{current}} - v_{\text{goal}})$	2.0
Feet motion	$r_7 = s\,(v_{\text{com}} - 0.5(\,v_{\text{left foot}} - \,v_{\text{right foot}}))$	0.625
Control	$r_8 = q_{\text{ctrl}} - q_{\text{key}}$	0.1
Foot clearance	$r_9 = h_{\text{clearance}} - \lvert z_{\text{left foot}} - z_{\text{right foot}} \rvert$	0.7
Feet distance (3D)	$r_{10} = \lVert p_{\text{right foot}} - p_{\text{left foot}} \rVert - d_{\text{goal}}$	2.0
Knee posture	$r_{11} = \lvert q_{\text{knee,L}} - 0.15 \rvert + \lvert q_{\text{knee,R}} - 0.15 \rvert$	0.2

4.2.2 Parameters and Extensions

The humanoid walking task relies on a set of scalar parameters that define the nominal posture, motion objectives, and spatial constraints used by the residual functions. The parameters and their reference values are summarized in Table 4.3.

Table 4.3: Parameter values used for the humanoid walking task.

Parameter name	**Value**
Torso height goal z_{goal}	1.25
Speed goal v_{goal}	$0.5 - 1$
Direction goal θ_{goal}	$0 - 360$
Feet distance goal d_{goal}	0.40
Desired clearance h_{goal}	0.10

With the walking behaviour implemented above, the locomotion task demonstrates a high level of stability: however, while changes in direction are followed faithfully, speed variations are not perfect, since the controller prefers stability to fast walking. This robustness allows for further extensions and integration.

The walking direction can be adjusted either through the API or directly within the control code. In the latter case, direction updates can be defined through a sequence of waypoints, switching to the next one when the humanoid reaches a given proximity threshold. This allows for circular or polygonal tra-

jectories. Alternatively, a pure velocity-based control strategy can be employed, although in practice a position feedback term, e.g., derived from odometry, should always be included for improved trajectory accuracy.

In one of the experimental applications, an additional control mechanism was tested in which the robot walks toward a target object, stops upon reaching a close distance, and activates a new residual to perform a reaching motion with one arm. This auxiliary term is defined as:

$$r_{12} = \| p_{\text{wrist}} - p_{\text{target}} \|_2,$$

where p_{wrist} is the 3D position of the robot's wrist and p_{target} is the position of the object to be reached. This formulation can be seamlessly integrated into the existing cost structure, allowing the humanoid to transition naturally between locomotion and manipulation phases.

4.3 Obstacle Negotiation Task

To enable the use of the humanoid robot in industrial contexts, it is not sufficient for it to simply walk; it must also be capable of safely overcoming obstacles that may appear along its path. The idea is that, as the robot approaches an obstacle (detected through sensors such as the LiDAR or the depth camera), it should enter an *alert phase*, during which both the controller weights and the residual objectives are adapted. This section describes these modifications in detail.

The extension for obstacle avoidance builds upon the walking framework described above, introducing several modifications to adapt locomotion control to a cyclic task. These adjustments enable the robot to lift its feet and maintain stability while overcoming obstacles. The overall residual structure remains unchanged; however, residual r_{11} was replaced, and the adaptive parameters were tuned to ensure consistent and rhythmic walking behavior.

The most relevant change is the introduction of a temporal gait cycle. Rather than computing the residuals from a static configuration at each instant, the obstacle task defines a periodic function that determines the current phase of the step. This function, characterized by a total cycle period, is modeled as a trapezoidal waveform with parameters Δ (ramp duration) and H (plateau duration). The gait variable $g(t)$ therefore alternates smoothly between 0 and 1, producing a continuous transition between stance and swing phases according to time. As a result, the control objectives are no longer constant but evolve with the gait phase, allowing the robot to alternate between double-stance and obstacle-overcoming phases in a physically plausible and dynamically consistent manner. The desired walking velocity and the target foot clearance are both modulated by this gait signal. Specifically, when $g(t) = 0$, the controller enforces a zero-velocity condition with both feet on the ground, ensuring full static stability. Conversely, when $g(t) = 1$, the controller activates the stepping behavior, driving the robot to raise one leg and move forward to clear the obstacle. Note that the code does not specify which leg to lift first; this decision is left to the optimizer or control

policy to maintain flexibility and adaptivity.

The residual responsible for foot clearance is reformulated accordingly. In the standard walking task, clearance is a static difference between the heights of the two feet. However, this formulation can lead to undesired situations when the robot maintains a standing position directly on top of the obstacle once $g(t)$ returns to zero. To prevent such unstable configurations, an additional penalty term is introduced, discouraging the controller from keeping the feet elevated when contact should be re-established with the ground. This ensures that feet are lifted only during the appropriate swing phase ($g(t) = 1$) and return to the ground when $g(t) = 0$.

A further modification is applied to the posture residual, where a selective mask is introduced in the joint configuration error. This mask reduces the residual weight by a factor of ten for the posture of specific joints, particularly the `hip_pitch` and `knee` (see Fig. 3.5), which play a crucial role in overcoming obstacles, as will also be demonstrated experimentally.

Table 4.4: Summary of modified residual terms for the humanoid obstacle-overcoming task.The changes are highlighted in bold.

Name	**Formula**	**Weight** w_i
Torso height	$r_0 = z_{\text{torso}} - z_{\text{goal}}$	20.0
Pelvis–feet	$r_1 = \frac{1}{2}(z_{\text{left foot}} + z_{\text{right foot}}) - z_{\text{pelvis}} - 0.2$	1.0
Balance	$r_2 = s\,(p_{\text{capture}} - p_{\text{cp}})$	20.0
Upright orientation	$r_3 = \begin{bmatrix} (u_{\text{torso},z} - 1) \\ 0.3\,(u_{\text{pelvis},z} - 1) \\ 0.1\,s\,(u_{\text{right foot}} - e_z) \\ 0.1\,s\,(u_{\text{left foot}} - e_z) \end{bmatrix}$	2.0
Posture (masked)	$r_4 = \mathbf{mask} \odot (q_{\text{pos}} - q_{\text{key}})$	0.25
Facing direction	$r_5 = s\,(f_{\text{current}} - f_{\text{goal}})$	5
Velocity (cyclic)	$r_6 = s\,(v_{\text{com}} \cdot f_{\text{current}} - v_{\text{goal}}\,\mathbf{g(t)})$	6.0
Feet motion	$r_7 = s\,(v_{\text{com}} - 0.5(\,v_{\text{left foot}} - v_{\text{right foot}}))$	0.625
Control	$r_8 = q_{\text{ctrl}} - q_{\text{key}}$	0.1
Foot clearance	$r_9 = \big\lvert h_{\text{clearance}}\,\mathbf{g(t)} - \lvert z_{\text{left foot}} - z_{\text{right foot}}\rvert\big\rvert + \big\lvert h_{\text{clearance}}\,\mathbf{g(t)} - (z_{\text{left foot}} + z_{\text{right foot}})\big\rvert$	5.0
Feet distance **(2D)**	$r_{10} = \lVert p_{\text{right foot}} - p_{\text{left foot}}\rVert - d_{\text{goal}}$	5.0
hand distance **(new)**	$r_{11} = \sqrt{(y_{\text{hand}} - y_{\text{goal}})^2 + (z_{\text{hand}} - z_{\text{goal}})^2}$	5.0

In this way, the robot is encouraged to take advantage of these joints rather than others to effectively overcome the obstacle. This adjustment allows these joints to move more freely, enabling greater flexibility during the obstacle-crossing phase while preserving overall balance and posture control.

In addition, a new residual is introduced to model the spatial relationship between the robot's right hand and a designated target position. In a realistic

scenario, such a position would be estimated from environmental perception, for instance through LiDAR-based reconstruction. However, in this simplified task, the environment is assumed to be perfectly known, and the hand target is defined geometrically rather than perceptually. Specifically, the hand is guided to make contact with an imaginary line lying on a plane parallel to the x–z plane, representing an idealized surface of the obstacle. This residual thus drives the hand toward a predefined location corresponding to a stabilizing or supportive contact during obstacle negotiation. Enabling the robot to make contact with a wall or nearby surface significantly increases the success rate of obstacle traversal.

Finally, the controller weights have been substantially tuned to improve stability and consistency during these complex maneuvers. The changes are highlighted in bold in Table 4.4.

4.4 Integration of Walking and Obstacle Negotiation

In this section, we describe the integration of walking and obstacle-negotiation tasks within a unified control framework, where visual perception enables smooth phase transitions between locomotion and obstacle-overcoming behaviors. The primary objective is to extend the humanoid's capabilities to operate in irregular and realistic environments, such as the interior of an aircraft fuselage introduced in Section 4.1.3. In this scenario, the robot must overcome a 20 cm high obstacle while maintaining its balance on an inclined surface.

Since the fuselage floor is not perfectly flat, the ground inclination introduces an additional challenge for the controller. Nevertheless, the control strategy proved robust under the weight configuration adopted for this scenario. A perception module based on simulated LiDAR sensing continuously scans the area in front of the robot, detecting the presence of obstacles and identifying their height and position. Whenever a new point higher than previously detected is observed, the system updates the stored obstacle height and location accordingly.

When the robot approaches sufficiently close to the stored obstacle position, the controller increases the weight associated with the clearance residual by a factor of four and updates the desired clearance value to match the detected obstacle height plus a small safety margin of 18 cm. This adaptive update ensures that the swing foot lifts high enough to safely clear the obstacle.

After the robot has completely stepped over the obstacle, the controller automatically restores all weights and parameters to their nominal walking values, allowing the robot to resume standard locomotion while continuing to monitor the environment for new obstacles. This process repeats cyclically, assuming a static environment in which the geometry of the obstacles does not change over time.

For simplicity, this experiment assumes that the obstacle lies orthogonally to the robot's direction of motion and that its height is physically surmountable by

the robot. This formulation represents an intermediate step toward more general obstacle-negotiation scenarios. The main goal is to demonstrate the feasibility of performing such adaptive behaviors in simulation, while the integration of more advanced computer-vision algorithms for precise obstacle and surface detection would further enhance performance and robustness in complex environments.

4.5 Challenges in Sim-to-Real Transfer

Once the optimization strategy is defined and the residuals governing the controller behavior are weighted accordingly, it is natural to question whether such a control scheme, designed entirely in simulation, is truly valid and deployable on real robotic systems. The remarkable work by Zhang et al. [33] provides an affirmative answer, demonstrating that, under specific conditions, model predictive control (MPC) based on iLQR and MuJoCo dynamics can be surprisingly effective when transferred from simulation to the real world [33].

Figure 4.5 shows the system architecture that enables execution of the MPC policy, computed in simulation, on physical hardware. The authors show that iLQR policies obtained in MuJoCo are successfully deployed on both a Unitree quadruped (Go1/Go2) and a full-sized humanoid robot (Unitree H1), without requiring complex adaptation mechanisms or domain randomization techniques. Despite the inherent discrepancies between simulated and real-world dynamics, the system operates robustly on both platforms, with satisfactory performance in terms of tracking and stability.

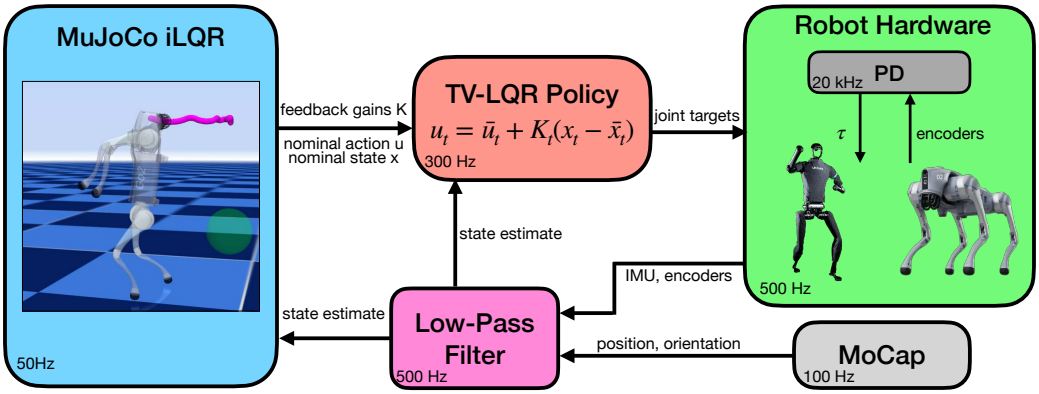


Figure 4.5: System diagram for deploying the MuJoCo iLQR policy to the Unitree Quadruped and Humanoid robots. The iLQR algorithm provides control, state, and time-varying LQR (TV-LQR) feedback gain trajectories at 50 Hz. The TV-LQR feedback policy can then be updated at 300 Hz and passed to a joint-level PD controller. (source: [33]).

As indicated in the diagram, the control architecture operates at multiple synchronized frequencies. The iLQR planner, running at approximately 50 Hz, continuously computes updated optimal trajectories and corresponding time-varying feedback gains. Between two consecutive iLQR updates, a lightweight TV-LQR (Time-Varying Linear Quadratic Regulator) controller runs at 300 Hz to provide fast feedback stabilization around the nominal trajectory, compensating for small disturbances and state estimation noise. Information from the joint encoders, reflecting the current state of the system, is sampled at 500 Hz. In this way, the controller does not simply apply the first control input of the NMPC solution; instead, it exploits the initial segment of the optimal state and control trajectories, thereby compensating for the slower simulation rate relative to the robot's real dynamics.

A central challenge in the sim-to-real transition lies in the accurate modeling of contact interactions. MuJoCo implements a so-called soft contact model, a convex approximation of the inherently non-convex contact dynamics. This approach allows for slight interpenetration between bodies to maintain numerical smoothness, continuity of derivatives, and stability in gradient-based optimization. While this formulation is computationally efficient and well-suited for real-time control, it can produce contact behaviors that are not fully physical when using default parameters.

In particular, the default contact configuration may lead to undesired foot slippage in simulation, causing discontinuous or oscillatory joint torques that would be unsafe to reproduce on real hardware. To mitigate this issue, Zhang et al. [33] modified the contact solver settings by increasing the parameter `impratio`. According to the MuJoCo documentation [48], values of impratio greater than 1 make the frictional constraints "stiffer" than the normal ones, effectively preventing slip without altering the nominal friction coefficient. In other words, higher impratio values bias the solver to resist tangential motion more strongly than penetration, resulting in a firmer contact response.

This simple but effective adjustment significantly reduces non-physical foot slippage and produces smoother, more realistic control signals, at the cost of introducing computational overhead, approximately doubling the iLQG resolution time per iteration (from approximately 10 ms to 20 ms in their experiments). According to the authors, the simplifications in the simulation model do not negatively affect performance on real hardware.

Another key element facilitating sim-to-real transfer is the use of a real-time interactive graphical interface (GUI), which allows users to monitor both the simulated and physical states of the robot and to dynamically adjust MPC parameters during execution. Through this interface, one can modify target positions, torso heights, and cost weights in real time. This interactive tuning capability enables the controller to be refined if necessary.

Beyond the work of Zhang et al. [33], recent research has also explored the deployment of dynamically consistent controllers designed in simulation onto physical legged robots, with particular attention to whole-body control, real-time execution, and robustness to modeling inaccuracies. Alvarez-Padilla et al. [29]

propose a real-time whole-body control system capable of producing complex locomotion and manipulation behaviors directly on hardware. Their approach leverages Model-Predictive Path Integral Control (MPPI) combined with massively parallel simulation in MuJoCo to evaluate full-body trajectories online. Unlike offline-trained policies or handcrafted contact strategies, this method enables emergent behaviors, such as body pushes, unexpected footholds, and impulsive interactions, without explicitly specifying contact modes. The controller is validated on a Unitree Go1, showing robust locomotion on uneven terrain. In this formulation, the optimization operates at the level of joint positions, while the corresponding torques are generated by an on-board PD tracking controller.

Similarly, Xue et al. [28] adopt a sampling-based MPC paradigm, but in this case the optimization is performed directly over joint torques. The method, referred to as DIAL-MPC (Diffusion-Inspired Annealing for Legged MPC), introduces a diffusion-style annealing process to refine torque sequences online. The system demonstrates precise and agile behaviors on a Unitree Go2, including jumping and obstacle climbing, and represents one of the first real-time full-order torque-level MPC frameworks for legged robots.

Taken together, these approaches illustrate the increasing viability of training-free, model-based control pipelines for real-world legged robotics, as well as the diversity of strategies available for bridging the simulation-to-reality gap.

In this thesis, we adopt a methodology similar to [33], applying it to the H1-2 humanoid robot, the successor to the H1 model. However, all experiments are performed exclusively in the MuJoCo simulator, without hardware implementation. Nevertheless, following the structure of the control scheme shown in Fig. 4.5, we propose a multi-frequency approach illustrated in Fig. 4.6. In this scheme, the MoCap block is removed for the locomotion task, and the walking simulation frequencies are set in the 20–50 Hz range—results that will be derived and discussed later using a reduced model.

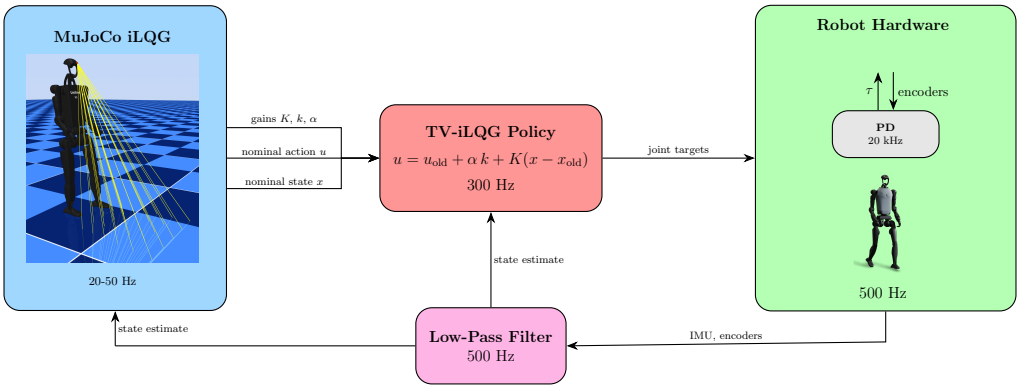


Figure 4.6: Multi-frequency control scheme inspired by Zhang et al. [33] and adapted to our case study involving the Unitree H1-2 humanoid robot with an iLQG-based locomotion controller.

Chapter 5

Results and analysis

This chapter presents the results obtained from the humanoid robot tasks within the simulation environment. The results are organized by scenario: walking, obstacle crossing, and the integrated locomotion framework. The evaluation includes stability parameters, walking frequency, residual analysis, and the percentage of successfully overcome obstacles.

For the walking scenario, different optimization and control techniques are also compared, namely the iterative Linear Quadratic Gaussian (iLQG), sampling-based, and gradient-based methods. In addition, the computing times required for online NMPC control calculations are examined in order to assess the feasibility of real-time implementation. Finally, the integration of the walking and obstacle-overcoming modules is discussed, highlighting the transition mechanisms and overall system performance.

5.1 Walking Analysis

The analysis focuses on the walking task on flat terrain. Three algorithms for nonlinear optimisation are compared, and the performance of the best-performing one, iLQG, is examined in greater detail.

5.1.1 Optimisation Techniques

The mathematical formulation of the algorithms was discussed in Chapter 3, while the experimental configuration was presented in Chapter 4. For completeness, a brief summary of the optimisation techniques used by NMPC is provided below:

- **Gradient-based**: relies on the local derivatives of the cost function to iteratively update control parameters in the direction of steepest descent, seeking a locally optimal solution.
- **iLQG**: the iterative Linear-Quadratic-Gaussian method linearises the system dynamics and quadraticises the cost function around a nominal trajectory, computing locally optimal feedforward and feedback control laws.

- **Sampling-based**: explores the control or trajectory space by generating multiple candidate solutions through stochastic sampling and selecting the sequence that minimises the objective function.

Although each case presented here corresponds to a single simulation, representative runs were chosen that exhibit average behaviour among multiple trials.

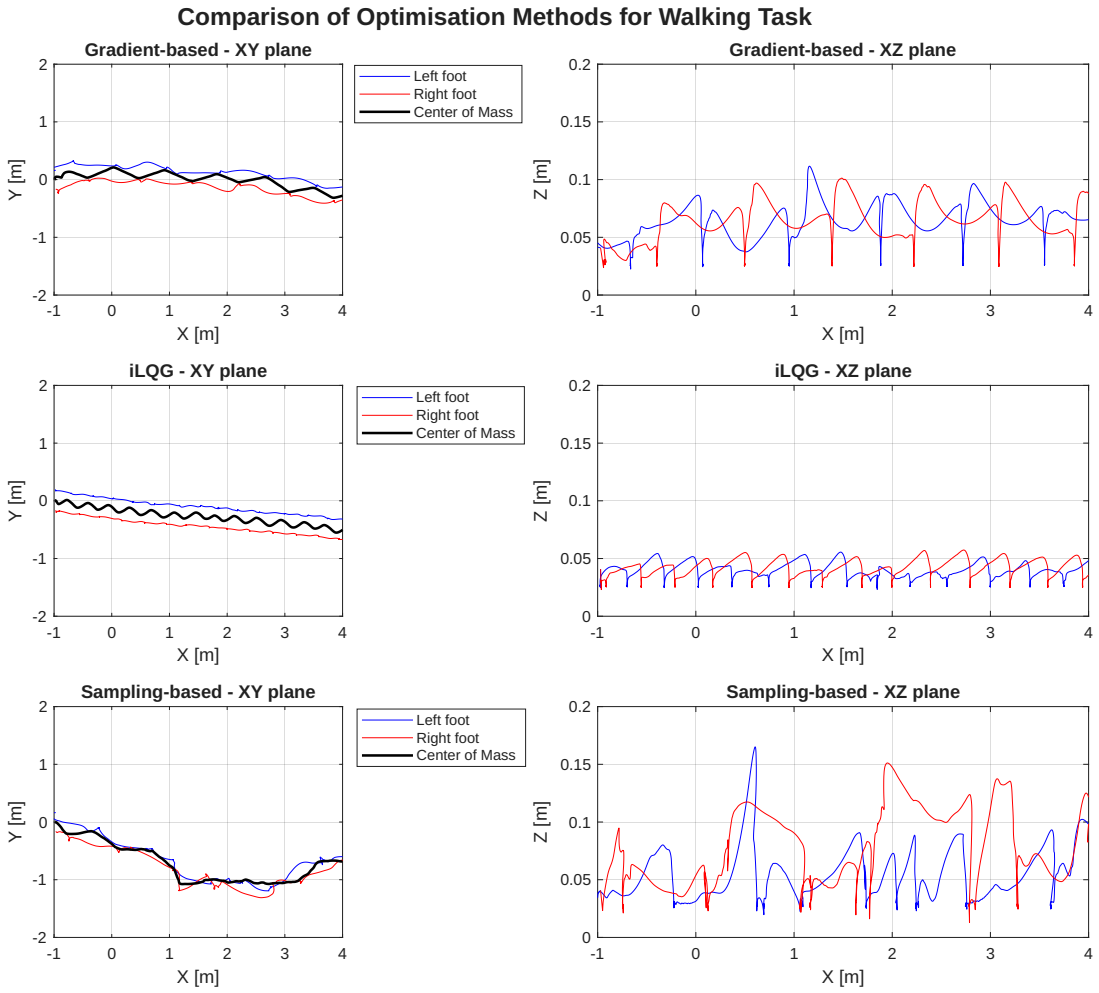


Figure 5.1: Comparison of optimisation techniques used for the walking task. Each panel shows the trajectories of the feet and the Center of Mass (CoM) obtained using different optimisation methods, with the corresponding residual weights reported in Table 5.1.

Table 5.1: Comparison of residual weights w_i for different optimisation techniques in the walking task.

Method	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
Gradient-based	20.0	1.0	20.0	5.0	1.0	15.0	20.0	0.625	0.1	0.7	2.0	0.2
iLQG	5.0	1.0	5.0	5.0	1.0	15.0	2.0	0.625	0.1	0.7	2.0	0.2
Sampling-based	10.0	1.0	10.0	5.0	1.0	15.0	5.0	0.625	0.1	0.7	2.0	0.2

Using the same weight configuration, the controllers' performance proved to be extremely poor. For this reason, it was considered more appropriate to recompute the weights, while keeping the residual functions unchanged, to better evaluate the different optimization techniques.

As shown in Fig. 5.1, the iLQG method demonstrates significantly greater stability. Throughout the entire walking sequence, the center of mass remains well within the trajectory defined by the two feet, maintaining a satisfactory safety margin.

In all flat-terrain simulations performed with the parameters reported in Table 5.1, the robot was able to walk stably without losing balance. Furthermore, the iLQG method proved to be the most robust approach: even when simulating walking for several minutes, the robot never fell.

Regarding the Sampling algorithm, its overall behavior is not entirely incorrect. However, since this technique relies on a stochastic exploration strategy, it tends to generate less harmonic gait patterns and cannot guarantee long-term stability. As illustrated in Figure 5.1, the robot does not fall, but may develop entirely undesired behaviors, such as crossing its legs during walking. Indeed, when the simulation was extended over longer durations, the robot lost balance and fell. It should nevertheless be noted that the optimization time for the sampling-based method is considerably shorter, and that, in principle, adjusting the prediction horizon could lead to improved results. However, the aim of this comparative analysis was to keep as many factors as possible unchanged; therefore, no such tuning was performed.

Finally, the gradient-based method achieved acceptable performance, further confirming the strengths of derivative-based optimization techniques; however, its frequency indices, stability metrics, and computation time are all worse than those of iLQG. Sampling-based and gradient-based methods are nevertheless included for completeness, given their frequent use in the MuJoCo MPC framework [37] and their continued relevance in control research.

5.1.2 Computational Time

Since the MPC algorithm is computed online, it is essential to perform a comparative evaluation of the computational performance of the controller. This section reports the computation times required to complete the walking task under several configurations, in order to assess the efficiency and feasibility of real-time execution.

- **With and without video rendering:** As the onboard computer of the robot does not perform any graphical rendering during operation, this comparison aims to quantify how the absence of video rendering improves overall computational performance.

- **Number of actuators (13, 21, and 27):** In the literature [8], [33], walking tasks for humanoid robots are often modelled using only the lower limb and torso actuators (13 actuators) or with simplified configurations that

exclude the wrist joints (21 actuators). In this work, different actuation setups are tested to evaluate how the number of active actuators affects the computational load and the responsiveness of the controller.

- **Optimization methods (iLQG, Gradient-based, Sampling-based):** Under identical horizon lengths and step times, the average optimization times of the three methods are compared. It should be noted that, as discussed in the previous section, their control performance is not equivalent; thus, this analysis focuses purely on computational aspects.

- **Different real-time factors:** The real-time factor defines the simulation speed relative to real-time execution (1.0 corresponds to real time, while values below 1.0 indicate slower-than-real-time performance). By varying this parameter, we simulate the effect of different computational architectures with greater processing capabilities.

The results are compared in terms of efficiency, temporal consistency, and average CPU time. MuJoCo provides detailed profiling data for each sub-task at every time step, allowing the mean CPU utilization and execution times to be accurately computed.

Since these experiments were conducted on an *ASUS ROG Strix G533QR* laptop rather than on the onboard computer of the Unitree H1-2 robot, it is necessary to briefly compare the hardware architectures to assess code portability and to estimate real-time feasibility on the target platform. The simulation was performed using an AMD Ryzen 9 5900HX processor (8 cores/16 threads, 45 W TDP, 4.9 GHz boost), while the robot incorporates an Intel Core i7-1265U CPU (10 cores/12 threads, 15 W TDP, 4.8 GHz boost). Although both are modern laptop-class processors, the Ryzen 9 5900HX provides significantly higher sustained computational throughput, outperforming the Core i7–1265U by approximately 67% in aggregate benchmark results [45]. This difference, mainly due to the higher thermal envelope and frequency scaling of the Ryzen architecture, suggests that the real-time performance observed during experimentation represents an upper bound with respect to what can be expected on the robot's onboard computer.

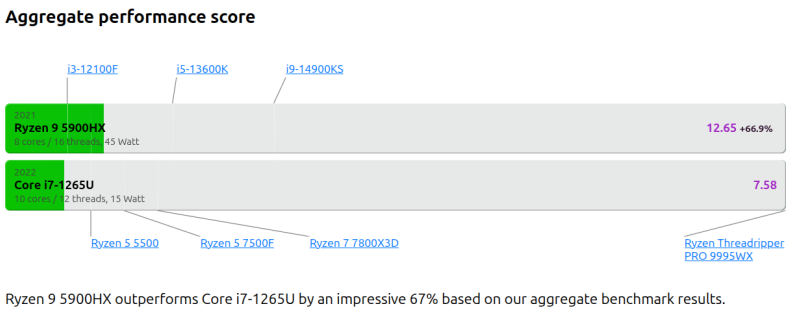


Figure 5.2: Aggregate performance comparison between the AMD Ryzen 9 5900HX and the Intel Core i7–1265U processors. Source: [45].

Table 5.2: Summary of simulation configurations and computational performance.
Note: The CPU usage (%) values are measured on the *ASUS ROG Strix G533QR* laptop. The acronym TSCT (ms) indicates the *Total Step Computation Time*, i.e., the time required to compute a single MPC step.

ID	Method	Actuators	Real-Time Factor	Rendering	Task Performed	CPU (%)	TSCT(ms)
1	iLQG	27	1.0	Enabled	no, loss of balance after 6 m	54.2	65.2
2	iLQG	27	0.5	Enabled	Yes	55.5	57.1
3	iLQG	27	0.5	Disabled	Yes	55,4	53.4
4	iLQG	27	1.0	Disabled	no, loss of balance after 10 m	54.5	57.3
5	iLQG	21	1.0	Enabled	Yes	59,6	40.5
6	iLQG	13	1.0	Enabled	Yes	57,4	19.7
7	Gradient-based	27	0.5	Enabled	no, loss of balance after 2 m	49.2	69.0
8	Gradient-based	27	0.25	Enabled	yes	43.2	67.0
9	Sampling-based	27	1.0	Enabled	no, immediate loss of balance	53.8	7.1
10	Sampling-based	27	0.5	Enabled	Yes	59.1	7.6

Before discussing the results presented in Table 5.2, some operational clarifications are necessary. To record CPU utilization during the simulations, the command `"mpstat -P ALL 1 > cpu_log.txt"` was used. Moreover, during the acquisition of data, only MJPC was employed; the use of ROS 2 would have further increased CPU load.

Throughout all experiments, several key variables were continuously monitored and logged for subsequent analysis. Since file I/O operations introduce a non-negligible computational overhead, a buffering mechanism was implemented to minimize performance interference. Specifically, a buffer of 100 elements was used: the data were temporarily stored in memory and written to file every 100 simulation steps. This detail is important to note, as for simulations performed without graphical rendering it is necessary to log internal state data in order to evaluate whether the walking task was successfully executed.

From the analysis of the results reported in Table 5.2, several considerations arise regarding the computational feasibility of real-time control on the robot's onboard system.

As previously discussed, the CPU usage values refer to the *ASUS ROG Strix G533QR* development system. Considering the performance gap between the two processors discussed earlier, a CPU utilization of approximately 60% on the development computer can be regarded as equivalent to a full computational load on the robot's onboard processor.

This equivalence enables a direct estimation of the system's portability: all configurations in 5.2 showing an average CPU usage below 60% can be considered realistically executable on the robot, ensuring real-time feasibility.

A closer examination of the data reveals that, with the full configuration of 27 actuators and 33 degrees of freedom, the robot fails to complete the walking task when the real-time factor is set to 1.0, as it loses balance after a few meters. Stable walking can only be achieved by reducing the real-time factor to 0.5. Therefore, this configuration is not physically implementable on the real robot, as it requires computational resources beyond those available on the onboard processor.

Further tests were conducted by disabling graphical rendering, thereby elim-

inating the computational overhead associated with visualization. Although this modification led to a measurable improvement in performance, it was still insufficient to sustain a real-time factor of 1.0, and the robot eventually lost balance, albeit after covering a longer distance.

A more substantial improvement was obtained by reducing the number of controlled joints. With 21 actuators, the average computation time for each MPC phase was approximately 40 ms, whereas with 13 actuators it decreased to about 20 ms. This latter result is particularly noteworthy, as it aligns with findings reported in the literature for simplified MPC implementations on real Unitree H1 hardware (which employs the same CPU as the H1-2), as discussed by Zhang et al. [33].

Further tests were carried out using gradient-based and sampling-based optimization methods, keeping the prediction horizon unchanged. Both approaches succeeded in generating walking behavior, but none achieved real-time operation with any configuration tested.

Among the methods evaluated, iLQG demonstrated the highest stability and control performance and with the proposed model simplifications, the resulting computational complexity is compatible with real-hardware deployment. Under these configurations, the walking motion remained stable, smooth, and dynamically consistent, confirming that real-time whole-body control can be achieved on the physical robot within the proposed framework.

MJPC provides timers implemented within the planner to estimate the computation time of each sub-task required to solve a single MPC iteration. Considering case 2 in Table 5.2, the computational phases corresponding to the iLQG algorithm described in Chapter 3.4.2 can be analyzed in detail.

In the NMPC controller based on the iterative iLQG algorithm, each control cycle is composed of several computational stages, whose execution times are shown in Fig. 5.3. The *Nominal* term corresponds to the forward simulation of the current policy, which generates the reference trajectory used for local approximations. The *Model Derivative* term represents the computation of the local linearization of the system dynamics with respect to the states and inputs, while the *Cost Derivative* refers to the computation of the cost function derivatives along the nominal trajectory.

As the dynamic model derivative computation is the most computationally expensive among all sub-tasks, reducing the model order (as shown in cases 5 and 6 of Table 5.2) leads to a significant reduction in the overall optimization time. The *Rollout* phase includes additional forward simulations of candidate trajectories to evaluate possible policy updates. The *Backward Pass* stage corresponds to the dynamic programming step of the iLQG algorithm, in which the value function is propagated backward in time to compute the optimal feedback corrections to be applied to the control inputs. Finally, the *Policy Update* stage applies these corrections using Eq. 3.28 to update the control law before the next iteration. The *Total* time thus represents the sum of all these contributions, quantifying the overall computational cost of one MPC update step.

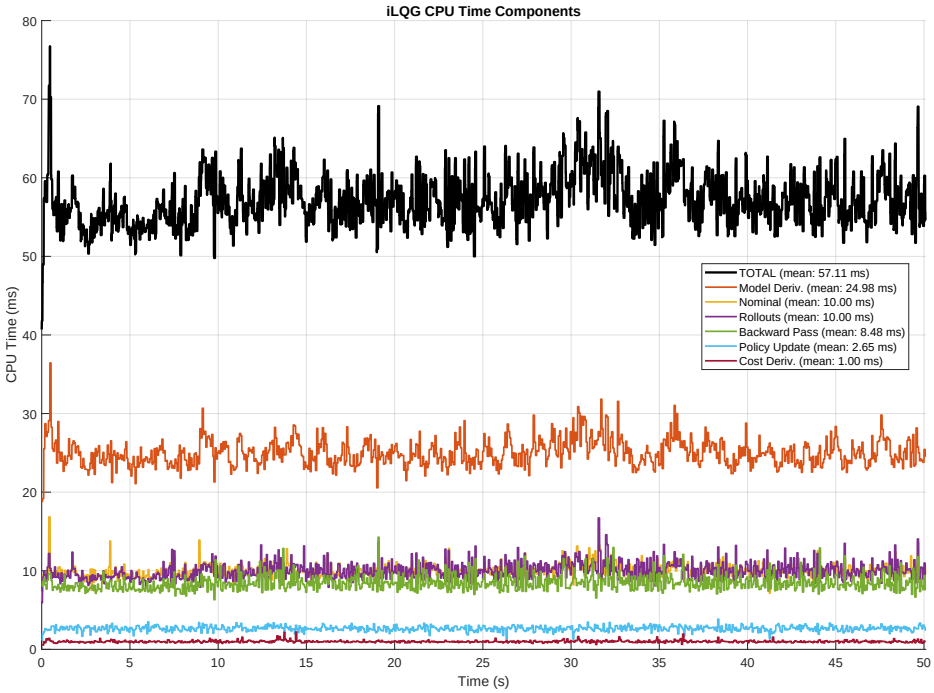


Figure 5.3: CPU time allocation for iLQG sub-tasks. Each line represents the computation time of the corresponding sub-process within a single MPC iteration. The average value of the sub-tasks is shown in the legend.

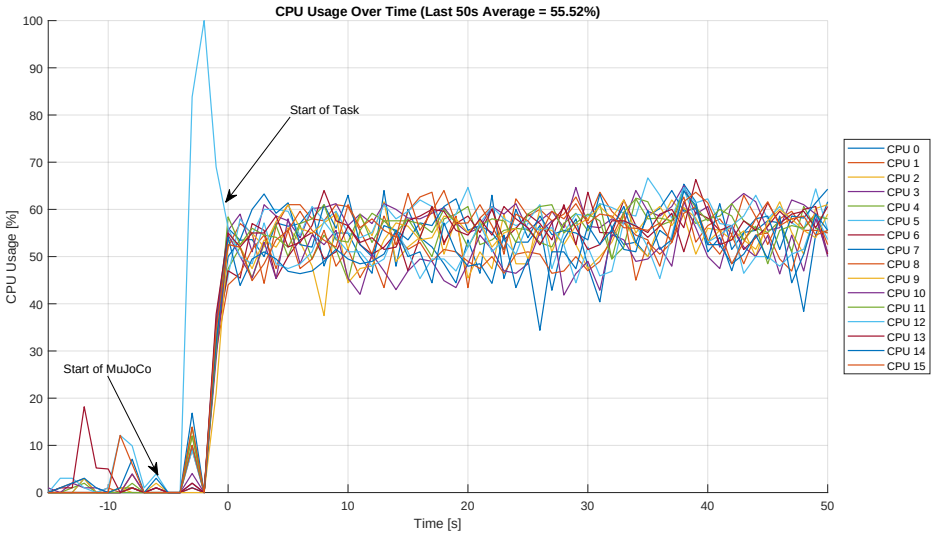


Figure 5.4: CPU usage over time for all 16 cores during the experiment. Vertical markers indicate the start of MuJoCo simulation and the start of the task. The average CPU load over the last 50 seconds was approximately 55.5%.

Fig. 5.4 shows the CPU usage of all 16 threads of the *ASUS ROG Strix G533QR* laptop during the same experiment. The timeline was synchronized so that the time origin ($t = 0$) corresponds to that of Fig. 5.3. It can be observed that the execution of the MuJoCo MPC framework engages all CPU threads

(referred to as “CPUs” in the Linux interface), hence the original notation was preserved. The purpose of this figure is to demonstrate not only that the average CPU load reaches approximately 55.5% (as reported in 5.2, case 2), but also that the workload is evenly distributed among the available threads.

5.1.3 Performance and Residual

We now undertake an analysis of the whole-body walking performance achieved using the iLQG controller within the complete humanoid configuration (33 DoF, 27 actuators), simulated with a real–time factor of 0.5. The MPC horizon is set to 0.35 s with a control step of 0.015 s and 10 rollouts. Residual weights and task parameters are taken from 4.2.

We first analyse straight–line walking, following the same scenario shown in Fig. 5.1. To evaluate gait stability, a set of indices commonly adopted in the literature is considered.

A first metric is the *perpendicular distance between the CoM and the line joining the feet*, as proposed by Jánoš et al. [14]. Figure 5.5 shows this distance over time. As visible in the plot, the CoM deviation from the foot–line axis never exceeds approximately 9 cm, indicating stable lateral behaviour. Such performance surpasses that observed with sampling-based (10 cm) and gradient-based (20 cm) techniques.

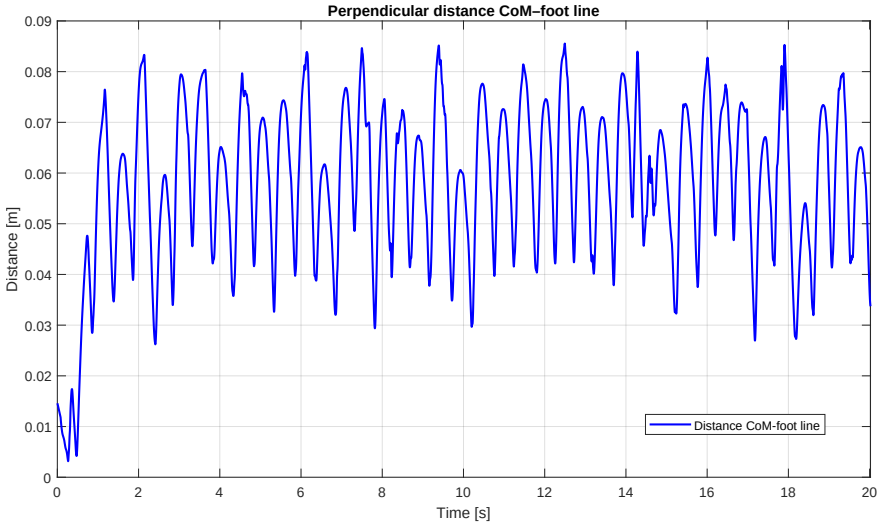


Figure 5.5: Perpendicular distance between the CoM and the line joining the feet during straight–line walking. Lower deviation indicates improved lateral stability.

Another widely used stability index in the literature is the distance of the ZMP from the support polygon. However, in our case MuJoCo does not provide direct access to the CoM acceleration, and differentiating the velocity introduces significant noise, making this metric less reliable than the CoM–to–foot–line distance in this context.

A further meaningful performance indicator often used in human gait studies is the frequency–domain analysis of step periodicity [12]. We adopt a similar approach here to evaluate the harmonic structure of the generated walking motion.

The vertical position of each foot was recorded for a duration of 20 s. The DC component (mean value) was removed, and a discrete FFT was computed. The resulting spectra, shown in Fig. 5.6, present two dominant harmonics, demonstrating a highly regular gait. The dominant frequency is approximately 1 Hz for both legs, consistent with typical human walking cadence [16]. Note that this frequency corresponds to the swing frequency of a single leg; thus, the overall step frequency is 2 Hz.

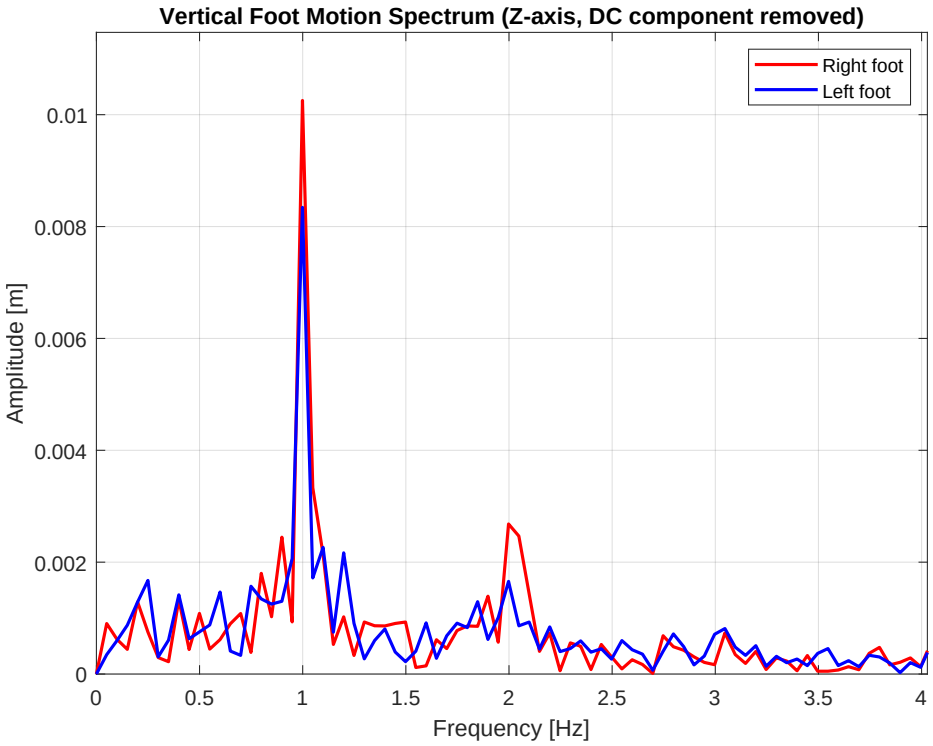


Figure 5.6: Frequency spectrum of the vertical motion of both feet during walking (Z–axis, DC component removed). The peaks indicate the main gait frequencies for the left and right foot.

Continuing from the straight–line walking analysis, we now evaluate the residual signals during locomotion on flat terrain. These residuals correspond to those listed in Table 4.2; however, it is important to emphasize that here they are shown *before* being multiplied by their respective weights and *before* the norm operator of Eq. 3.11 is applied. Therefore, the values can appear positive or negative and do not represent the final contribution to the cost, but rather the raw deviations from the reference behaviours.

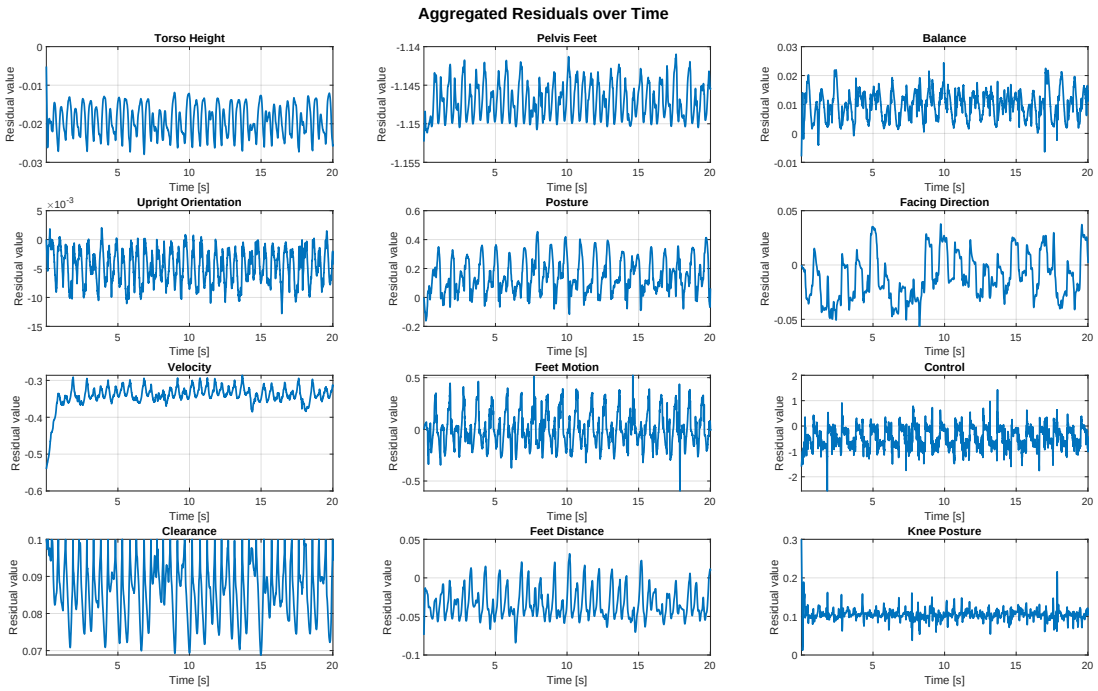


Figure 5.7: Aggregated residuals of various walking terms over time, including height, posture, balance, and foot clearance, indicating deviations from the expected reference values during flat–ground locomotion.

The residual plots provide several interesting insights. First, the velocity tracking residual exhibits a non–zero mean even during steady walking. This indicates that the controller prioritizes balance and stability over perfect velocity tracking when necessary. Similarly, the orientation residual r_5 oscillates around zero, meaning the robot roughly maintains the desired heading. However, as visible in Fig. 5.1, the CoM trajectory does not remain perfectly aligned with the world y–axis. After approximately 5 m of forward motion, a lateral drift of about 0.5 m accumulates, corresponding to a deviation angle slightly above 7°. This behaviour suggests that specifying only a desired speed vector may introduce slow orientation drift.

To address this, we adopt a waypoint–based heading strategy: the optimizer tracks only the desired forward speed magnitude and torso yaw, while the direction of motion is defined by the angle between the torso and the next waypoint in the world frame. Once the robot approaches a waypoint, the target switches to the following one. We tested this mechanism by commanding circular and square trajectories. The corresponding paths are shown in Fig. 5.8 and Fig. 5.9.

In both cases, the robot successfully follows the intended path and executes abrupt heading changes (up to 90°) without compromising stability. The maximum deviation from the reference trajectory remains below 40 cm, and can be reduced further by increasing the waypoint resolution. These results highlight both the reliability of the locomotion controller and the robot's ability to rapidly adapt its heading while maintaining balance.

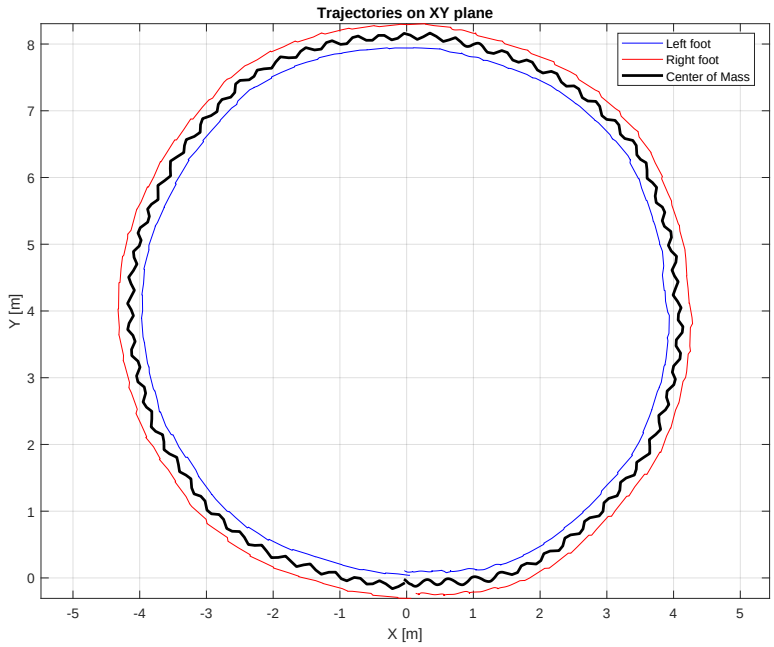


Figure 5.8: Trajectories of the feet and center of mass during circular walking. The robot continuously updates its heading based on successive waypoints, maintaining balance while changing direction.

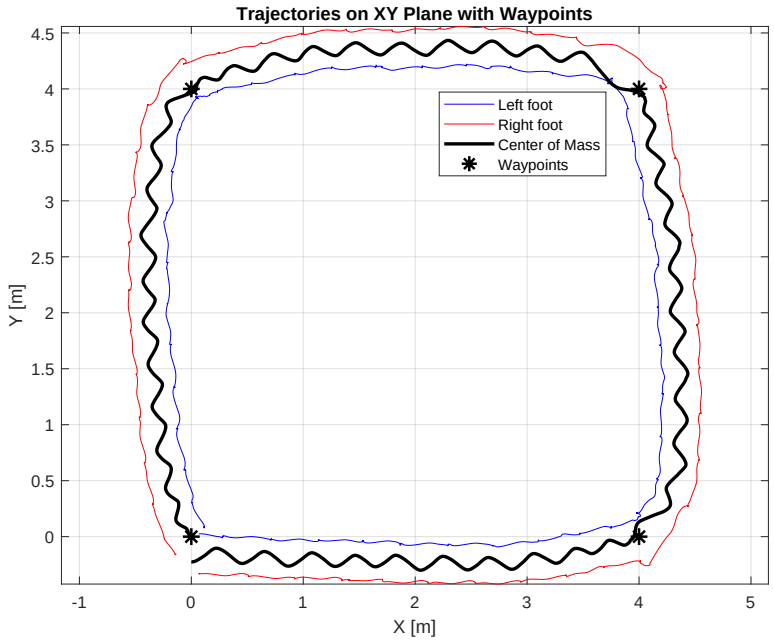


Figure 5.9: Foot and center of mass trajectories along a square path. Waypoints are shown as markers, and the robot successfully performs 90° heading changes without loss of stability.

5.2 Obstacle Negotiation Analysis

This section presents the results obtained during the obstacle-overcoming experiments performed in a simulated environment analogous to that shown in Fig. 4.1. A representative case of a 30 cm obstacle is first analyzed, focusing on the evolution of the residuals, the trajectories of the feet and the center of mass, and the clearance profile. Subsequently, the outcomes of a campaign of 80 trials are reported to evaluate the controller's capability to overcome obstacles of different heights.

5.2.1 Performance and Residual

The controller performance was assessed through the residuals and weights listed in Table 4.4. The wall on the robot's right side has a known position, and the controller allows the robot to use it as a support surface during task execution. This interaction is managed by residual r_{11}, which penalizes the hand–wall distance once contact is established. The main differences from the nominal locomotion task were discussed in Section 4.3.

The gait cycle lasts 3 s and includes a transition phase (0.3 s), an active phase (0.6 s), and a recovery phase (1.8 s). This timing scales with the desired foot clearance, whose trapezoidal profile is shown in Fig. 5.10. For the 30 cm obstacle, a clearance of 0.48 m was adopted, providing a sufficient safety margin. As illustrated in Fig. 5.11, the foot trajectories follow the desired reference, while the forward-motion residual, modulated by the gait variable, activates walking during the locomotion phase.

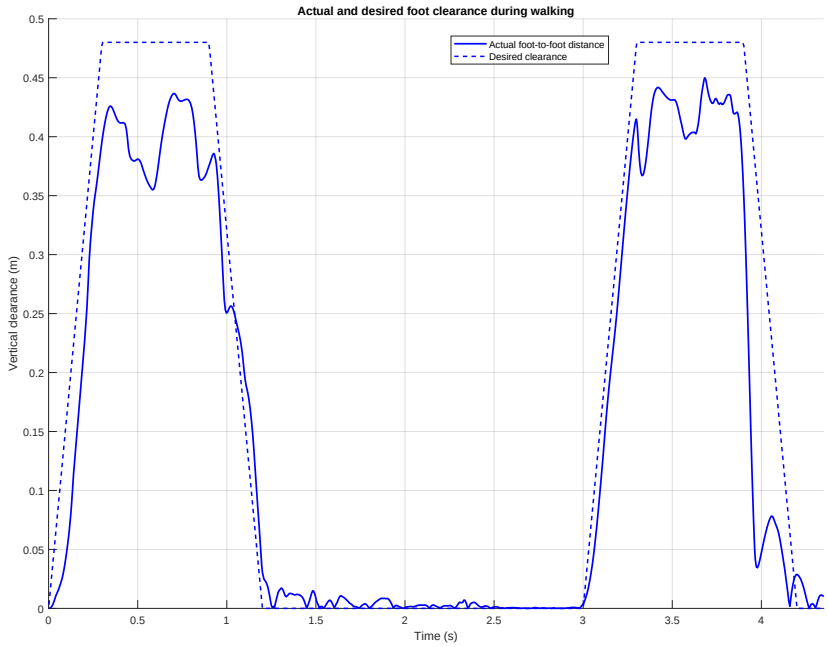


Figure 5.10: Comparison between the actual foot-to-foot vertical distance and the desired clearance profile during walking.

The NMPC configuration is identical to that used in the previous scenario, with a prediction horizon of 0.35 s and a simulation step of 0.015 s. The optimization problem is solved via the iterative Linear Quadratic Gaussian (iLQG) method. These parameters facilitate integration with the locomotion task, although the real-time factor decreases to 0.33.

The sequence begins with both feet in contact with the ground and the arms aligned along the body. Two steps are executed in approximately 4.2 s. As seen in Fig. 5.11, the right leg clears the obstacle and lands roughly 60 cm ahead of the left one. This asymmetry increases the foot-distance residual r_{10}. During the subsequent double support phase, the left foot slides forward 20 cm to restore a balanced position. Reducing the weight of r_{10} did not improve stability, this behavior is not critical and is expected to vanish in the integrated locomotion–obstacle task. The effect is mainly due to the initial setup, where the robot was placed farther from the obstacle to make the test conditions more general.

Another key aspect is the hand–wall interaction governed by residual r_{11}. The right hand moves toward a support line on the wall, located 1.3 m above the ground. Initially distant, it progressively approaches the wall. During the right-leg step, however, the center of mass shifts away from the wall (Fig. 5.11), causing a temporary separation of up to 70 cm. The controller promptly re-establishes contact before the left-leg step, when wall support becomes crucial for maintaining stability.

Residual analysis for the 30 cm obstacle shows that torso height and balance remain bounded, though slightly higher than in nominal walking. The overall residual magnitude is comparable, except for r_3 (upright orientation), which increases as the torso and legs tilt during obstacle crossing.

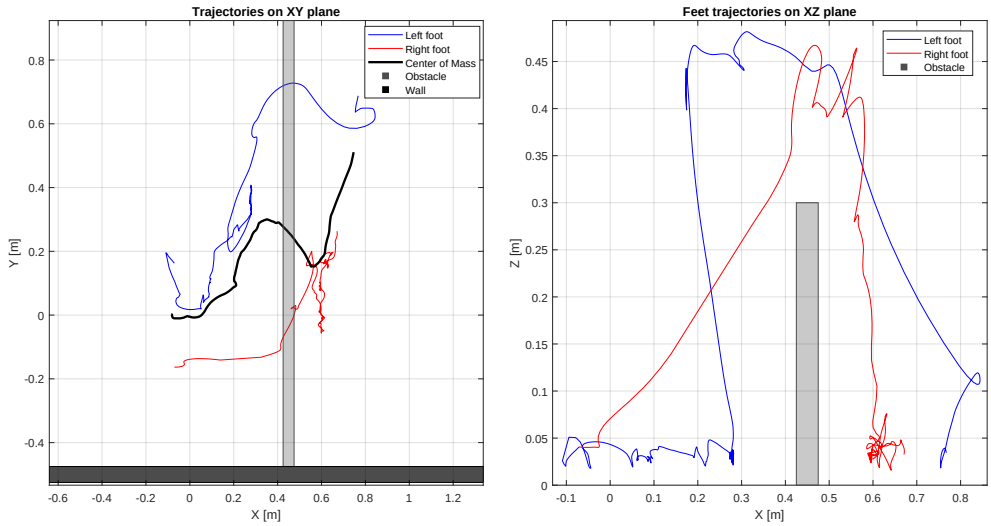


Figure 5.11: Trajectories of the robot's center of mass and feet during the obstacle crossing task, including the obstacle and supporting wall.

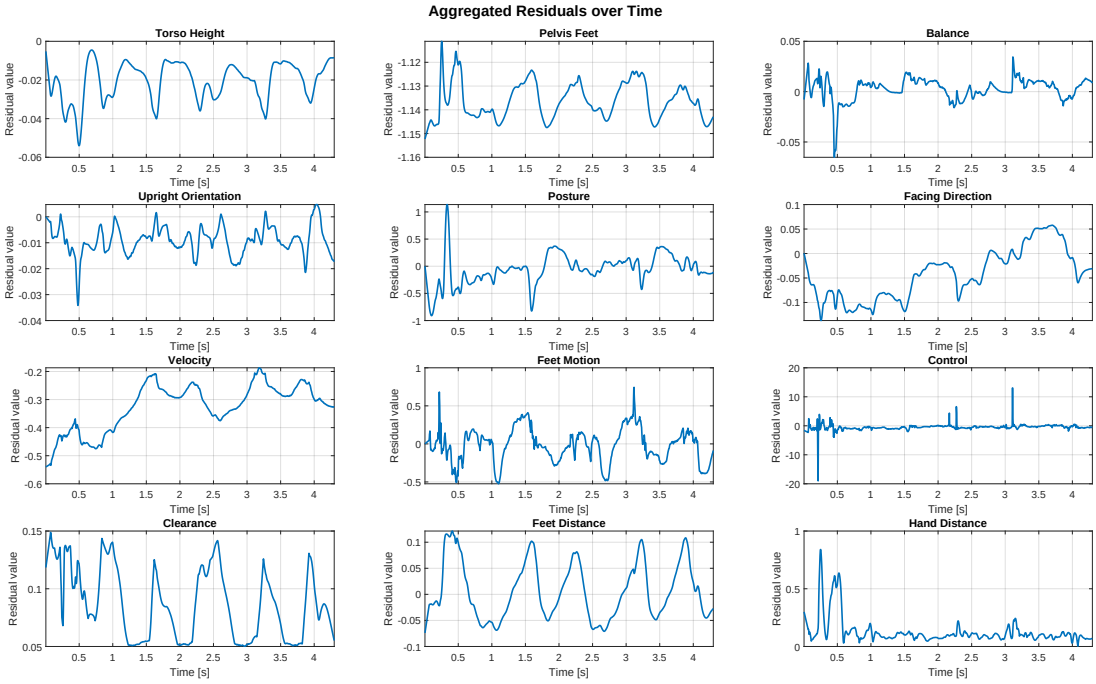


Figure 5.12: Aggregate residuals of various terms related to overcoming obstacles over time, including height, posture, and balance, indicating deviations from expected reference values during overcoming.

The previously analyzed case corresponds to one of the successful experiments. To evaluate the overall success rate, a set of 20 trials was conducted for each obstacle height of 10, 20, 30, and 40 cm. All control parameters were kept fixed across the experiments, with the sole exception of the desired clearance height, which was set to the obstacle height plus a safety margin of 18 cm.

The results indicate high reliability for smaller obstacles and a gradual degradation of performance as the obstacle height increases:

- **10 cm:** success in 100% of the trials;
- **20 cm:** success in 85% of the trials;
- **30 cm:** success in 60% of the trials;
- **40 cm:** success in 5% of the trials.

Failed attempts are mainly attributed to inadequate or excessive clearance values that caused the robot to lose balance. Typically, the first foot successfully cleared the obstacle, but failures occurred when the trailing foot or ankle collided or got stuck on the edge of the obstacle. For obstacles higher than 30 cm, the crossing task can therefore be considered infeasible.

The percentages reported in Fig. 5.13 represent a statistically significant sample. However, the success rate does not imply that the task was always completed in two steps on the first attempt. The *Partial Success* category in the figure corresponds to non-nominal robot behaviors-cases where the robot managed to

overcome the obstacle but with undesired contacts or postures, such as excessive impacts on the obstacle or wall, or the knee briefly touching the ground.

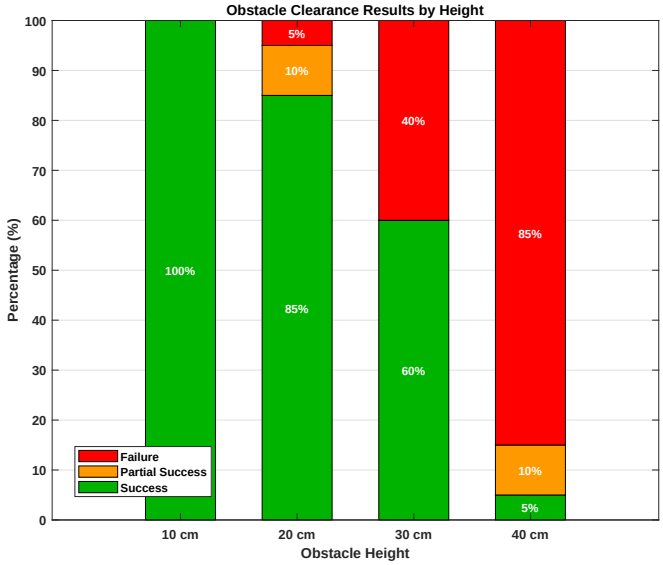


Figure 5.13: Statistical results of the robot's performance in overcoming obstacles. Evaluated over 20 episodes for each obstacle height. The graph shows the percentage of *Success*, *Partial Success*, and *Failure* for different obstacle heights.

5.3 Module Integration

This section presents the results of the integrated walking and obstacle-overcoming task, in which visual perception enables transitions between locomotion and obstacle negotiation behaviors.

As described in Section 4.4, the vision system spatially detects the obstacle. When the robot approaches the obstacle 30 cm before reaching it, the clearance parameter is updated to adapt the foot trajectory accordingly. After the obstacle has been crossed, the clearance is restored to its nominal value of 10 cm.

The walking motion is performed on a curved surface resembling the floor of an aircraft fuselage. The ground inclination affects the robot's stability, making this scenario more challenging than locomotion on flat terrain. The residuals used in this integrated activity are the same as those used for standard locomotion; however, their weights and parameters have been updated, as summarized in Table 5.3. In addition, the same activation mask applied to the posture residual during the obstacle-crossing task was also employed here to ensure more natural movement of the knee and hip.

The trajectory shown in Fig. 5.14 exhibits noticeable differences compared to the flat-terrain walking trajectory iLQG (see Fig. 5.1). In particular, the walking step length remains consistent, while the curved terrain and obstacles increase the oscillations of the center of mass. As shown in Fig. 5.14, the robot successfully clears the obstacle using both legs.

Table 5.3: Summary of residual terms and corresponding weights for the integrated task.

Residual Name	**Weight** w_i
Torso height (r_0)	20.0
Pelvis–feet (r_1)	1.0
Balance (r_2)	20.0
Upright orientation (r_3)	5.0
Posture (r_4)	1.0
Facing direction (r_5)	15.0
Velocity (r_6)	5.0
Feet motion (r_7)	0.625
Control (r_8)	0.1
Foot clearance (r_9)	$1.2 - 4.8$
Feet distance (3D) (r_{10})	5.0
Knee posture (r_{11})	0.2

The temporal evolution of the main residuals, reported in Fig. 5.15, clearly highlights the obstacle-crossing phase. Around the fourth second, the robot approaches the obstacle, as evidenced by the increase in the foot-clearance residual (r_9). Of particular interest is the knee-posture residual (r_{11}), which indicates that knee flexion becomes essential to achieve sufficient clearance. In parallel, variations in the vertical-orientation residual (r_3) reveal a progressive inclination of the feet, pelvis, and torso frames.

Moreover, the residual distance of feet (r_{10}) exhibits a periodic behavior when the robot is far from the obstacle. However, as the robot enters the crossing phase, this periodicity is disrupted: the residual increases to enforce a longer step, thereby reflecting the additional cost associated with safely negotiating the obstacle.

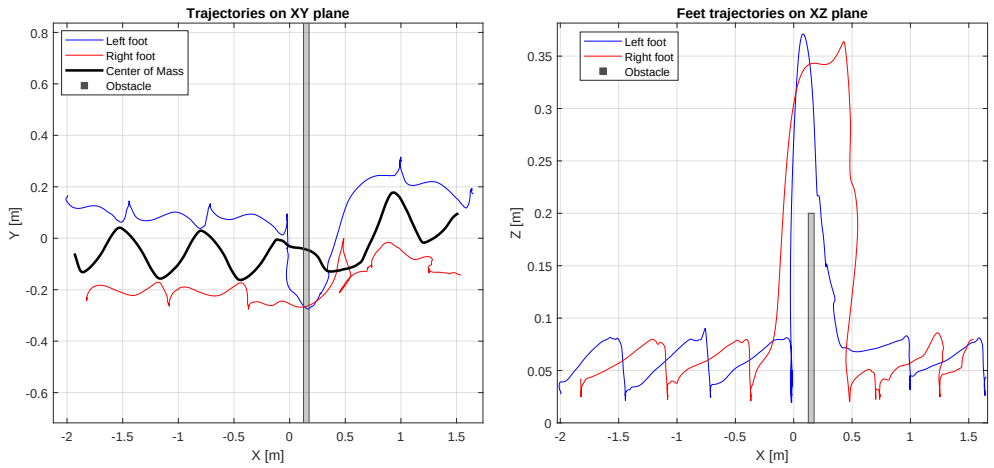


Figure 5.14: Trajectories of the robot's center of mass and feet during the movement in the fuselage.

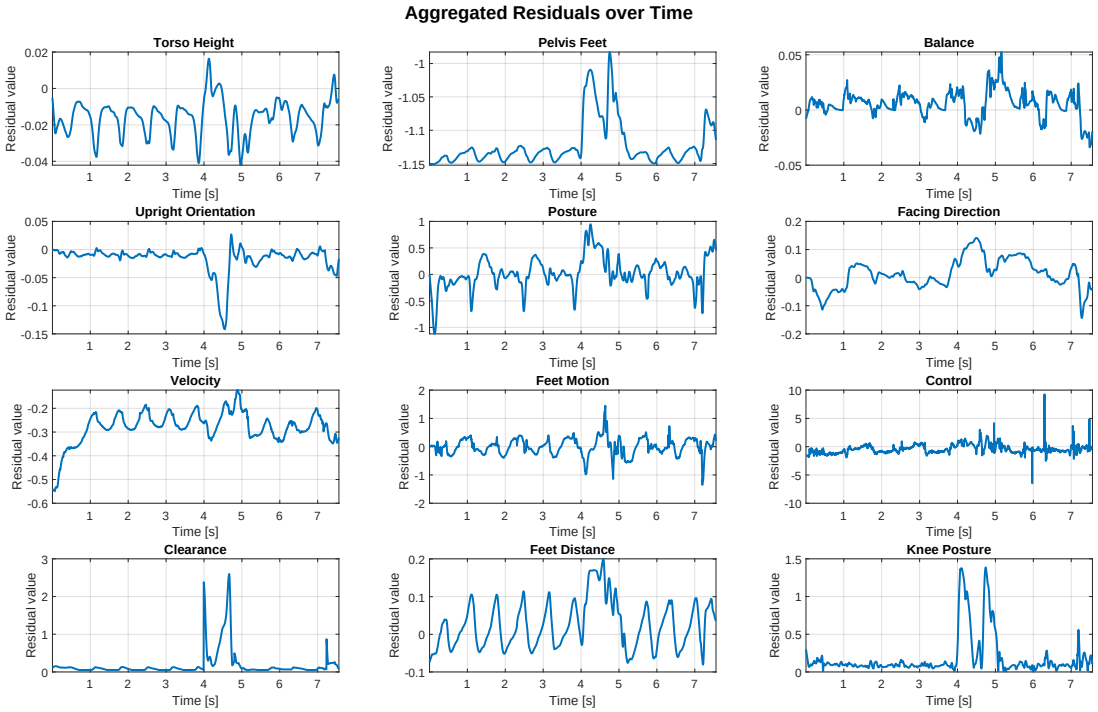


Figure 5.15: Aggregate residuals of various terms related to walking and negotiating obstacles over time, including height, posture and balance, indicating deviations from expected reference values.

The integrated task was successfully performed multiple times, demonstrating the repeatability and robustness of the proposed framework. The Unitree humanoid robot was able to complete the entire sequence without falling, consistently overcoming the obstacle and maintaining balance throughout the trials.

Chapter 6

Conclusions and Future Developments

In this thesis, a whole-body control architecture was developed for the Unitree H1-2 humanoid robot, currently available at the PRISMA Lab. After an extensive analysis of the software tools suitable for implementing a Nonlinear Model Predictive Control, MuJoCo MPC (MJPC) was identified as the most appropriate framework, despite its lack of native integration with ROS 2.

The MJPC framework provides several online optimisation methods. Among the algorithms evaluated—gradient-based, sampling-based and iLQG—the latter proved to offer the most stable performance for repetitive locomotion tasks such as walking.

Since NMPC requires real-time computation, an analysis was carried out to identify the conditions under which the MuJoCo simulator can reduce the computation time of each optimisation step. The results showed that, when considering all 27 joints of the robot, real-time locomotion is not achievable with the computing architecture used in this work. Nonetheless, by reducing the model to 21 or 13 joints, stable and computationally feasible locomotion can be achieved.

A comparison with the literature further confirmed the validity of the obtained results. To date, only one research group at Carnegie Mellon University [33] has successfully applied NMPC through MJPC to a real humanoid robot, achieving comparable performance in simulation and in the sim-to-real transition. Their findings also highlight that a multi-frequency control architecture, with position feedback running at 300 Hz and trajectory optimization at 50 Hz, provides robust performance for walking.

The work was subsequently extended to an industrial-inspired scenario representing the interior of an aircraft fuselage, characterised by an inclined floor and the presence of an obstacle. A “Divide et impera” strategy was adopted: the walking behaviour was first developed, then the obstacle-overcoming strategy, and finally the full integration of both, with autonomous switching enabled by a LiDAR-based perception module. The simulation results were overall satisfactory, confirming the effectiveness of the proposed methodology.

Future developments will focus on the integration of the MuJoCo simulator with ROS 2, with the aim of transferring the control framework to the real robot hardware or enabling sim-to-sim evaluation in environments such as Gazebo. The walking performance could also be further improved by allowing the humanoid to reach higher locomotion speeds. Indeed, while the manufacturer states that the robot can reach almost 2 m/s, the current implementation does not exceed 1 m/s. Although achieving high speed was not the primary objective of this thesis—whose focus was on stability—this remains an interesting direction for future work. Similarly, obstacle negotiation performance could be refined to improve adaptability and robustness.

Another possible extension concerns the integration of real-to-sim functionalities. By processing LiDAR and camera data to dynamically incorporate detected objects into the simulation environment, it becomes possible to generate a more realistic and continuously updated world model. Such a capability would allow the MPC to perform predictions over a dynamic and more accurate representation of the surroundings, thereby increasing the reliability of the control behaviour while keeping the computational load of the optimisation algorithm under control.

Finally, Nonlinear Model Predictive Control remains a highly active and promising research field, where new real-time optimisation algorithms and increasingly efficient control strategies can be explored and evaluated, potentially pushing the capabilities of humanoid robots even further.

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine al mio Relatore, il professor Lippiello, per la guida attenta e preziosa con cui mi ha accompagnato lungo tutto il percorso di tesi. Nei momenti di entusiasmo come in quelli più impegnativi, la sua sapienza nell'orientarmi con equilibrio e disponibilità è stata fondamentale, così come la sua costante presenza al Prisma Lab, che ha rappresentato per me un punto di riferimento sicuro.

Un ringraziamento altrettanto sentito va al mio Correlatore Riccardo Aliotta, che con pazienza e professionalità ha contribuito in modo determinante alla realizzazione del lavoro, guidandomi con chiarezza e metodo. La mia riconoscenza si estende a tutto il team del laboratorio: il confronto quotidiano, i consigli pratici e lo spirito collaborativo che ho trovato al Prisma Lab sono stati una parte fondamentale della mia crescita, non solo professionale ma anche personale. In questo ambiente ho imparato a lavorare in squadra, a mettermi in discussione e ad affrontare le difficoltà con approccio responsabile e consapevole.

Un pensiero speciale va alla mia famiglia, che rappresenta le radici di ogni mio passo. I miei genitori, Antonella e Gianluca, ai quali dedico questa tesi, sono il mio punto fermo, il porto sicuro: quella presenza solida sulla quale costruire la mia vita. Mi hanno trasmesso il valore della famiglia, l'amore per lo studio e il rispetto profondo per il prossimo. Accanto a loro, i nonni, gli zii e i cugini hanno sempre aggiunto sostegno, affetto e quel senso di appartenenza che rende più gioioso ogni traguardo e sopportabile ogni fatica.

Gli amici rappresentano un altro pilastro di questo percorso: quelle persone che scelgono di esserci, condividendo tanto le difficoltà quanto la spensieratezza. Tra le persone che hanno avuto un ruolo fondamentale in questi anni, un posto particolare spetta ad Antonio José e Pietro, che sono stati rispettivamente il fratello maggiore e il gemello che non ho mai avuto. La loro presenza silenziosa ma costante mi ha dato sostegno e leggerezza in molti momenti, senza bisogno di troppe parole.

Grazie anche ad Alessandro, Alfonso, Carlo, Francesco, Giovanni A., Giovanni P.,Giulia, Jacopo, Lorenzo, Marco, Mario e Paolo, con i quali ho costruito legami fatti di notti interminabili, conversazioni, risate e cazzate che hanno dato forma a ciò che siamo oggi. Un ringraziamento altrettanto affettuoso va al gruppo "Noi 9" di cui fanno parte Alessia, Claudia, Fabrizia, Gea e Matteo C., con cui ogni momento si trasforma in una vacanza.

Un grazie sentito va anche ai miei amici Claudia, Gianmarco, Fiorella, Roberto, Chiara e William, che hanno reso tutto questo possibile. I progetti affrontati in-

sieme, gli esami superati spalla a spalla e i viaggi condivisi hanno reso questo cammino molto più facile e divertente. Senza di voi, questo cammino non avrebbe avuto lo stesso sapore.

Infine, ma non per ultimo, un ringraziamento speciale a Franco, Maiello, Margherita, Mariachiara e Giovanni A., che mi hanno accompagnato durante la scrittura di questa tesi, sostenendomi nei momenti più stressanti e ricordandomi ogni giorno il valore della pazienza e dell'amicizia. Grazie per aver condiviso con me anche i piccoli grandi traguardi lungo il percorso.

A tutti voi va la mia più profonda gratitudine e non vedo l'ora di condividere nuovi passi, viaggi e obiettivi che sono certo, affronteremo ancora una volta insieme.

Bibliography

[1] D: Q. Mayne J. B. Rawlings and M. M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. 2nd. Madison, Wisconsin, USA: Nob Hill Publishing, 2017.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT Press, 2018.

[3] D.P.Bertsekas. "Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming". In: *IFAC-PapersOnLine* 58.18 (2024), pp. 363–383. DOI: 10.1016/j.ifacol.2024.09.056.

[4] C.Liao et al. "Performance Comparison of Typical Physics Engines Using Robot Models With Multiple Joints". In: *IEEE Robotics and Automation Letters* 8.11 (2023), pp. 7520–7526. DOI: 10.1109/LRA.2023.3320019.

[5] S. Akki and T. Chen. "Benchmarking Model Predictive Control and Reinforcement Learning Based Control for Legged Robot Locomotion in MuJoCo Simulation". In: *arXiv preprint arXiv:2501.16590* (2025). DOI: 10.48550/arXiv.2501.16590.

[6] S. H. Bang, C. Arribalzaga Jové, and L. Sentis. "RL-augmented MPC Framework for Agile and Robust Bipedal Footstep Locomotion Planning and Control". In: *arXiv preprint arXiv:2407.17683* (2024). DOI: 10.48550/arXiv.2407.17683.

[7] M. Meser et al. "MuJoCo MPC for Humanoid Control: Evaluation on HumanoidBench". In: *arXiv preprint arXiv:2408.00342* (2024). DOI: 10.48550/arXiv.2408.00342.

[8] W.Xie et al. "Humanoid Whole-Body Locomotion on Narrow Terrain via Dynamic Balance and Reinforcement Learning". In: *arXiv preprint* (2025). DOI: 10.48550/arXiv.2502.17219.

[9] M.E. Moran. "The da Vinci Robot". In: *Journal of Endourology* 20.12 (2006), pp. 986–990. DOI: 10.1089/end.2006.20.986.

[10] E. Cerveró-Meliá, S.F. Capuz-Rizo, and P. Ferrer-Gisbert. "Leonardo da Vinci's Contributions from a Design Perspective". In: *Designs* 4.3 (2020). DOI: 10.3390/designs4030038.

[11] T.Fukuda, P.Dario, and G.Yang. "Humanoid robotics History, current state of the art, and challenges". In: *Science Robotics* 2 (2017), eaar4043. DOI: 10.1126/scirobotics.aar4043.

[12] N. Abhayasinghe and I. Murray. "Human Gait Modeling, Prediction and Classification for Level Walking Using Harmonic Models Derived from a Single Thigh-Mounted IMU". In: *Sensors* 22.6 (2022). DOI: 10.3390/s22062164.

[13] Hiroshi Makino. "Development of the SCARA". In: *Journal of Robotics and Mechatronics* 26.1 (2014). DOI: 10.20965/jrm.2014.p0005.

[14] R. Jánoš et al. "Stability and Dynamic Walk Control of Humanoid Robot for Robot Soccer Player". In: *Machines* 10.6 (2022). DOI: 10.3390/machines10060463.

[15] Qiang Huang et al. "Historical Developments of BHR Humanoid Robots". In: *Advances in Historical Studies* 8 (2019), pp. 79–90. DOI: 10.4236/ahs.2019.81005.

[16] F. Danion et al. "Stride variability in human gait: The effect of stride frequency and stride length". In: *Gait & posture* 18 (2003), pp. 69–77. DOI: 10.1016/S0966-6362(03)00030-4.

[17] S. Shilong, H Haodong, and Chiyao Li. "Advancements in humanoid robot dynamics and learning-based locomotion control methods". In: *Mini-invasive Surgery* 5.3 (2025). DOI: 10.20517/ir.2025.32.

[18] V. Makoviychuk et al. "Isaac gym: High performance gpu-based physics simulation for robot learning". In: *arXiv preprint arXiv:2108.10470* (2021).

[19] W. Jallet et al. "PROXDDP: Proximal Constrained Trajectory Optimization". In: *IEEE Transactions on Robotics* 41 (2025), pp. 2605–2624. DOI: 10.1109/TRO.2025.3554437.

[20] Miomir Vukobratovic and Branislav Borovac. "Zero-Moment Point - Thirty Five Years of its Life." In: *I. J. Humanoid Robotics* 1 (2004), pp. 157–173. DOI: 10.1142/S0219843604000083.

[21] Xue Bin Peng et al. "DeepMimic: example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions on Graphics* 37 (2018). DOI: 10.1145/3197517.3201311.

[22] Y.Tassa, T. Erez, and E.Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012)*. Vilamoura, Portugal, 2012, pp. 4906–4913. DOI: 10.1109/IROS.2012.6386025.

[23] C. Mastalli et al. "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2020. DOI: 10.48550/arXiv.1909.04947.

[24] S. Kajita et al. "The 3D linear inverted pendulum mode: A simple modeling for a biped walking pattern generation". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the Next Millennium (Cat. No. 01CH37180)*. Vol. 1. IEEE. 2001, pp. 239–246. DOI: 10.1109/IROS.2001.973365.

[25] E. Todorov and W. Li. "A Generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems". In: *Proc. American Control Conference (ACC)*. IEEE. 2005, pp. 300–306. DOI: 10.1109/ACC.2005.1469949.

[26] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[27] S. H. Bang, C. Arribalzaga Jové, and L. Sentis. "RL-augmented MPC Framework for Agile and Robust Bipedal Footstep Locomotion Planning and Control". In: *Proceedings of the 2024 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2024, pp. 607–614. DOI: 10.1109/Humanoids59832.2024.10630538.

[28] H. Xue et al. *Full-Order Sampling-Based MPC for Torque-Level Locomotion Control via Diffusion-Style Annealing*. 2024. arXiv: 2409.15610. URL: https://arxiv.org/abs/2409.15610.

[29] J. Alvarez-Padilla et al. *Real-Time Whole-Body Control of Legged Robots with Model-Predictive Path Integral Control*. 2024. arXiv: 2409.10469 [cs.RO]. URL: https://arxiv.org/abs/2409.10469.

[30] F. Farshidian et al. *OCS2: An Open-Source Library for Optimal Control of Switched Systems*. Accessed on October 26, 2025. 2024. URL: https://github.com/leggedrobotics/ocs2?tab=readme-ov-file.

[31] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. *Pinocchio: fast forward and inverse dynamics for poly-articulated systems*. 2015–2021. URL: https://stack-of-tasks.github.io/pinocchio.

[32] M. Körber et al. *Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning*. 2021. DOI: 10.48550/arXiv.2103.04616.

[33] J.Z. Zhang et al. *Whole-Body Model-Predictive Control of Legged Robots with MuJoCo*. 2025. arXiv: 2503.04613. URL: https://arxiv.org/abs/2503.04613.

[34] C. Sferrazza et al. *HumanoidBench: Simulated Humanoid Benchmark for Whole-Body Locomotion and Manipulation*. 2024. DOI: 10.48550/arXiv.2403.10506.

[35] M. Kaup et al. *A Review of Nine Physics Engines for Reinforcement Learning Research*. 2024. URL: https://arxiv.org/abs/2407.08590.

[36] R. Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics*. 2019. URL: https://drake.mit.edu.

[37] T.Howell et al. *Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo*. 2022. URL: https://arxiv.org/abs/2212.00541.

[38] NVIDIA et al. *Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning*. 2025. URL: https://arxiv.org/abs/2511.04831.

[39] J. Shao, C. Zhu, and Y. Hou. *Sim2Sim on Legged Robots: Domain Randomization for Cross-Simulator Policy Transfer*. Tech. rep. Final Report,Accessed on October 19, 2025. Stanford University, Department of Mechanical Engineering, 2025. URL: https://cs224r.stanford.edu/projects/pdfs/CS224R_Final_Report.pdf.

[40] Unitree Robotics. *Unitree H1 - Product Specifications*. Ultimo accesso: 10 ottobre 2025. URL: https://www.unitree.com/h1.

[41] Unitree Robotics. *Unitree H1-2 Overview*. Accessed on 10 Oct 2025. URL: https://www.unitree.com/h1.

[42] Quadruped Docs. *H1-2 Technical Overview*. Accessed on 10 Oct 2025. URL: https://www.docs.quadruped.de/projects/h1/html/h1-2_overview.html.

[43] Unitree Robotics. *Unitree H1 User Manual*. Accessed on 10 Oct 2025. URL: https://support.unitree.com/home/en/H1_developer/About_H1-2.

[44] Unitree Robotics. *unitree_sdk2: Unitree Robot SDK Version 2*. Accessed on 10 Oct 2025. URL: https://github.com/unitreerobotics/unitree_sdk2/tree/main.

[45] technicalcity. *i7-1265U vs Ryzen 9 5900HX*. Accessed on 10 Oct 2025. URL: https://technical.city/en/cpu/Ryzen-9-5900HX-vs-Core-i7-1265U.

[46] Unitree Robotics. *MuJoCo Repository*. Accessed on October 19, 2025. URL: https://github.com/unitreerobotics/unitree_mujoco/tree/main.

[47] GitHub user "badinkajink". *MuJoCo MPC – HumanoidBench Branch*. Accessed on October 19, 2025. URL: https://github.com/badinkajink/mujoco_mpc/tree/humanoidbench.

[48] DeepMind Technologies. *MuJoCo XML Reference Documentation*. Accessed on October 19, 2025. URL: https://mujoco.readthedocs.io/en/stable/XMLreference.html.

[49] Livox Technology Company. *Livox MID-360 User Manual*. Accessed on October 19, 2025. URL: https://www.sachtleben-technology.com/wp-content/uploads/sites/5/2024/07/LivoxMid-360UserManual.pdf.

[50] Boston Dynamics. *Atlas Robot*. Accessed on December 03, 2025. URL: https://bostondynamics.com/atlas/.

[51] PAL Robotics. *TALOS Robot*. Accessed on December 03, 2025. URL: https://pal-robotics.com/robot/talos/.

[52] Agility Robotics. *Agility Robotics Official Website*. Accessed on December 03, 2025. URL: https://www.agilityrobotics.com/.