Article

Not peer-reviewed version

Humanoid Motion Generation in Complex 3D Environments

Diego Marussi , Michele Cipriano , Nicola Scianca * , Leonardo Lanari , Giuseppe Oriolo

Posted Date: 19 May 2025

doi: 10.20944/preprints202505.1371.v1

Keywords: humanoid; planning; Model Predictive Control

 doi:10.20944/preprints202505.1371.v1

Article

Humanoid Motion Generation in Complex 3D Environments

Diego Marussi[1], Michele Cipriano[1], Nicola Scianca[1,*], Leonardo Lanari[1] and Giuseppe Oriolo[1]

[1] Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Italy; lastname@diag.uniroma1.it
* Correspondence: scianca@diag.uniroma1.it

Abstract: We address the problem of humanoid locomotion in a 3D environment composed of planar regions with arbitrary inclination and elevation, which includes staircases, ramps, and multi-floor layouts. The proposed framework couples an off-line randomized footstep planner with an on-line control pipeline comprising a model predictive controller and a whole-body controller which provides the robot torque commands. The planner efficiently explores the environment without predefined motion primitives and returns the highest-quality plan it can find within a user-specified time budget, while the control layer ensures dynamic balance and adequate ground friction. The complete pipeline is evaluated in MuJoCo using the JVRC1 humanoid across four scenarios of increasing complexity.

Keywords: humanoid; planning; Model Predictive Control

1. Introduction and Related Work

Humanoid robots have seen remarkable advancements in recent years, due to improvements in both design and control. They are increasingly being considered for practical applications in settings where human-like mobility is essential. Unlike wheeled robots, which are mostly limited to flat and structured environments, humanoids are designed to traverse complex human-centric environments, including stairs and ramps.

Despite recent successes demonstrating formidable agility, we have seen less impressive results in the field of autonomous navigation of complex environments. Indeed, in such environments it is necessary to couple long-term planning with effective control.

Moving in complex and cluttered environments demands the ability to plan over a long horizon, ensuring that the robot can reach a distant goal without getting stuck. To make the problem more approachable, the possible paths the robot can take are often reduced to those that can be constructed as concatenations of *primitives*, i.e., the displacement between two consecutive footsteps is selected from a catalogue [1]. The problem then becomes that of searching within this large discrete space. Approaches based on grid or graph search (e.g., A*) proved effective in relatively simple contexts, thanks to clear heuristics and systematic exploration [2–4]. However, they may suffer from combinatorial explosion due to the need of exploring a large number of branches.

The representation of the environment is a crucial aspect, not only for efficiency reasons, but also because it can allow or disallow features of the algorithm. A commonly used representation is the *elevation map*, which associates each point in the xy-plane with a corresponding elevation value. Although elevation maps are easy to reconstruct from data, they have limitations, e.g., they do not allow representing multi-floor environments. In [5,6], the authors showed efficient ways to represent the environment as a set of planar regions, and tested on-line footstep planning using an A* algorithm.

A significant boost in the search efficiency can be obtained using sample-based methods, such as those based on Rapidly-exploring Random Trees (RRT) [7]. These methods can address some of the limitations of search techniques because the expansion mechanism is biased towards exploration. RRT in its basic formulation however does not account in any way for the quality of the produced plan, and

 doi:10.20944/preprints202505.1371.v1

can lead to inefficient motions. Furthermore, the use of a primitive catalogue will result in a certain *granularity* of the final plan, as the number of possible choices is limited by the number of primitives available.

On the control side, many techniques have been proposed. While approaches based on reinforcement learning and imitation learning have proven to produce very agile motions, these are usually not designed to move in crowded environments because high-level commands are given as reference velocities, rather than a precise sequence of footstep positions. Conversely, approaches based on Model Predictive Control (MPC) can more effectively follow an input reference footstep plan. Classic methods use a Linear Inverted Pendulum (LIP) as a simplified dynamic model [8], which is unsuited to motion in complex environments due to the constant height Center of Mass (CoM) requirement. However, later studies have shown [9,10] that other similar simplified models can be used to generate variable-height CoM trajectories.

The problem becomes more complex when the surfaces on which it is possible to step have an arbitrary orientation, since the possibility of slipping of the contact feet is now significant. Therefore, it is necessary to generate interaction forces that offer sufficient friction. One possibility is to produce a sufficiently robust controller that can walk blindly and adapt to unknown slope changes [11,12]. While this has its advantages, there are cases in which the terrain information is known, and its use will undoubtedly produce better performance. [13] focused on the problem of moving across surfaces with known different orientation, but the footstep plan was given a priori.

We previously proposed an MPC-based humanoid gait generation methodology including an explicit stability constraint [14]. While the original approach used the LIP as a template model, we have showed that it can be extended in order to generate vertical motions [15], and even running [16]. In [17], we integrated our MPC controller with a footstep planner to realize motion in complex environments constituted by horizontal ground patches at different heights (a *world of stairs*). The exploration was performed by randomly expanding a tree of possible footstep sequences with the goal of reaching a given goal region. The planner was given a time budget, after which it would return the best found plan according to a specified optimality criterion using an RRT*-like strategy [18].

In this paper, we propose a comprehensive framework for footstep planning and control that addresses the unique demands posed by complex environments, including the presence of stairs, inclined planes, and multiple floors. The footstep planner is based on an off-line randomized exploration which finds the best footstep plan in a given time budget. Its goal is to find the sequence that reaches a given goal position in the minimum number of steps. The control pipeline is constituted by a Model Predictive Controller (MPC) and a Whole-Body Controller (WBC) generating torque control commands for the humanoid.

In particular our unified framework is capable of:

- planning a collision-free sequence of footsteps in a *world of ramps*, i.e., an environment constituted by planar regions with different inclination, without using motion primitives;
- optimizing said footstep plan according to a quality metric and returning the best solution found within the afforded time budget;
- providing, along with the footstep sequence, also variable height collision-free trajectories for the swing foot in order to overcome small obstacles on the ground or climb at a different height;
- generating a stable 3D CoM trajectory that allows the robot to move along the planned footstep sequence;
- computing torque commands for the humanoid robot in such a way that contact forces provide sufficient friction to avoid slipping.

We validated the footstep planner with an extensive testing campaign in which we evaluated its performance in several different simulated environments, and compared results using different time budgets. Furthermore, we tested the full framework, including the control modules, in dynamic simulations in MuJoCo.

 doi:10.20944/preprints202505.1371.v1

The rest of this article is organized as follows. In the next section, we formulate the motion generation problem while in Section 3 we give an overview of the proposed approach. A full description of the introduced footstep planner is given in Section 4 while the control architecture is presented in Section 5. We extensively test the proposed approach on increasingly complex environments through dynamic simulations in MuJoCo in Section 6. Section 7 concludes the article.

2. Problem Formulation

In the situation of interest (Figure 1), a humanoid robot moves in a *world of ramps*, a specific 3D environment consisting of planar regions that are arbitrarily placed and oriented in space. This characterization encompasses office-like scenarios, with rooms connected by staircases or ramps, possibly arranged on several floors and populated by obstacles. Depending on its elevation and/or inclination, a planar region may be accessible for the humanoid to step on, or not.

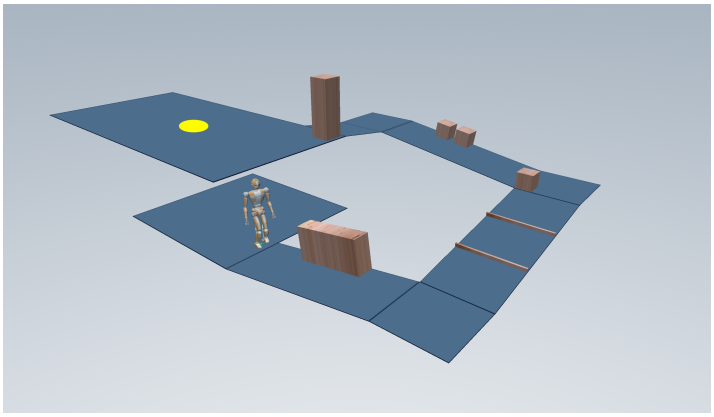

Figure 1. An instance of the considered problem. The robot must reach the goal area (in yellow) by traversing a *world of ramps*.

The mission of the robot is to reach a certain goal area $\mathcal{G}$, assumed to be contained in a single planar region. The locomotion task is accomplished as soon as the robot places a footstep inside $\mathcal{G}$.

We want to devise a complete framework enabling the humanoid to plan and execute a motion to fulfill the assigned task in the world of ramps. This requires addressing two fundamental problems: finding a 3D footstep plan and generating a variable-height gait consistent with such plan. Footstep planning consists in finding both footstep placements and swing foot trajectories; overall, the footstep plan must be feasible (in a sense to be formally defined later) for the humanoid, given the characteristics of the environment. Gait generation consists in finding a Center of Mass (CoM) trajectory which realizes the footstep plan while guaranteeing dynamic balance of the robot at all time instants. Starting from the trajectories of the CoM and the feet, a whole-body controller will generate torque commands for the robot joints that complying with the robot kinematic and dynamic limitations.

The problem will be solved under the following assumptions.

A1 Information about the environment geometry is completely known a priori and collected in a *map* $\mathcal{M}$ expressed as

$$\mathcal{M} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\},$$

where $\mathcal{R}_i$ is an arbitrarily oriented planar region in 3D space (a *ramp*).

A2 The humanoid is endowed with a localization module which, using the available sensor data, provides an estimate of the robot current state in terms of its CoM and swing foot pose.

In view of assumption A1, a complete footstep plan leading to the goal area $\mathcal{G}$ can be computed off-line, before the humanoid starts to execute the motion. While we will maintain this viewpoint in the present paper, an extension to the on-line case can be devised along the same lines of [17], i.e., by updating the footstep plan on the basis of new information gathered and added to $\mathcal{M}$ during the motion, e.g., using visual information [5,6,17].

 doi:10.20944/preprints202505.1371.v1

3. Proposed Approach

To solve the described problem, we adopt the architecture shown in Figure 2, in which the main components are the footstep planning, gait generation and whole-body control modules.

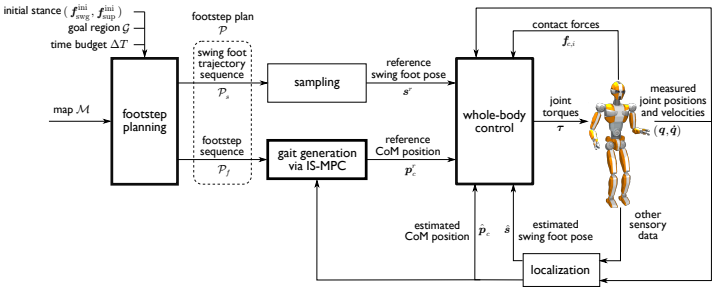

Figure 2. A scheme of the proposed solution approach. Blocks with thick countour are the subject of specific sections in the paper.

In the following, we denote by $\boldsymbol{f} = (x_f, y_f, z_f, \alpha_f, \beta_f, \gamma_f)$ the *pose* of a footstep, with x_f, y_f, z_f representing the coordinates of a representative point, henceforth collectively denoted as $\boldsymbol{p}_f$, and $\alpha_f, \beta_f, \gamma_f$ its orientation, expressed via roll-pitch-yaw angles, collectively denoted by $\boldsymbol{\phi}_f$. Moreover, a pair $(\boldsymbol{f}_{\text{swg}}, \boldsymbol{f}_{\text{sup}})$ defines a *stance*, i.e., the feet poses during a double support phase after which a step is performed by moving the swing foot from $\boldsymbol{f}_{\text{swg}}$ while keeping the support foot at $\boldsymbol{f}_{\text{sup}}$.

The footstep planner receives in input the initial humanoid stance $(\boldsymbol{f}^{\text{ini}}_{\text{swg}}, \boldsymbol{f}^{\text{ini}}_{\text{sup}})$, the goal area $\mathcal{G}$, a time budget ΔT, and the map $\mathcal{M}$ of the environment.

The time budget represents the time given to the planner to find a solution. When this time runs over, the algorithm either returns a solution or ends with a failure. Explicitly specifying this time as input to the algorithm allows us to evaluate the performance of the planning module, but also paves the way for the extension to the on-line case, where the time budget would be set equal to the duration of a step in order to meet the real-time requirement.

The planner works off-line to find, within ΔT, an optimal *footstep plan* $\mathcal{P} = \{\mathcal{P}_f, \mathcal{P}_s\}$ leading to the desired goal area $\mathcal{G}$. In $\mathcal{P}$, we denote by

$$\mathcal{P}_f = \{\boldsymbol{f}^1, \ldots, \boldsymbol{f}^n\}$$

the sequence of footstep placements, whose generic element $\boldsymbol{f}^j$ is the pose of the j-th footstep, with $\boldsymbol{f}^1 = \boldsymbol{f}^{\text{ini}}_{\text{swg}}$ and $\boldsymbol{f}^2 = \boldsymbol{f}^{\text{ini}}_{\text{sup}}$. Also, we denote by

$$\mathcal{P}_s = \{\boldsymbol{s}^1, \ldots, \boldsymbol{s}^{n-2}\}$$

the sequence of associated swing foot trajectories, whose generic element $\boldsymbol{s}^j$ is the j-th *step*, i.e., the trajectory leading the foot from $\boldsymbol{f}^j$ to $\boldsymbol{f}^{j+2}$ over a fixed duration T_s.

Once the footstep plan has been generated, the sequence of footsteps $\mathcal{P}_f$ is sent to a gait generator based on Intrinsically Stable MPC (IS-MPC), which computes in real time a variable-height CoM trajectory that is compatible with $\mathcal{P}_f$ and guaranteed to be *stable*, i.e., bounded with respect to a specific point called *Zero Moment Point* (ZMP, more on this in Sect. 5.1). In particular, we denote by $\boldsymbol{p}^r_c$ the current reference position of the CoM produced by IS-MPC. Also, let $\boldsymbol{s}^r$ be the current reference pose of the swing foot, obtained by sampling the appropriate subtrajectory in $\mathcal{P}_s$.

Finally, the reference trajectories $\boldsymbol{p}^r_c$ and $\boldsymbol{s}^r$ are passed to a whole-body controller which computes the joint torque commands τ for the robot so as to track these trajectories while ensuring that the generated contact forces are feasible.

Sensor information, including joint encoder data, is used by the localization module to continuously update the estimate of the CoM position and swing foot pose ($\hat{\boldsymbol{p}}_c$ and $\hat{\boldsymbol{s}}$, respectively) through kinematic computations. Finally, these estimates are used to provide feedback to both the gait generation and whole-body control modules.

 doi:10.20944/preprints202505.1371.v1

4. Footstep Planner

The input data for this module are the initial robot stance $(f_{\text{swg}}^{\text{ini}}, f_{\text{sup}}^{\text{ini}})$, the goal region $\mathcal{G}$, the time budget ΔT and the elevation map $\mathcal{M}_z$. Given an optimality criterion, the footstep planner returns the best footstep plan $\mathcal{P}$ leading to $\mathcal{G}$ found within ΔT.

The planning algorithm builds a tree $\mathcal{T}$, where each vertex $v = (f_{\text{swg}}, f_{\text{sup}})$ specifies a stance, and an edge between two vertexes v and $v' = (f'_{\text{swg}} = f_{\text{sup}}, f'_{\text{sup}})$ indicates a step between the two stances, i.e., a collision-free trajectory such that one foot swings from f_{swg} to f'_{sup} and the other is fixed at f_{sup}.

Each branch joining the root of the tree to a generic vertex v represents a footstep plan $\mathcal{P}$. The sequences $\mathcal{P}_f$ and $\mathcal{P}_s$ for this plan are respectively obtained by taking along the branch the support foot poses of all vertexes and the steps corresponding to all edges.

4.1. Footstep Feasibility

Footstep $f^j = (\boldsymbol{p}_f^j, \boldsymbol{\phi}_f^j) = (x_f^j, y_f^j, z_f^j, \alpha_f^j, \beta_f^j, \gamma_f^j) \in \mathcal{P}_f$ is *feasible* if it satisfies the following requirements:

R1 f^j is fully contained in a single horizontal region.

In practice, one typically uses an enlarged footprint to ensure that this requirement will still be satisfied in the presence of small positioning errors.

R2 f^j is kinematically admissible from the previous footstep f^{j-1}.

Given f^{j-1}, the *kinematically admissible region* for f^j is the submanifold of $\mathbb{R}^3 \times SO(3)$ defined as

$$\mathcal{K}(f_{j-1}) = \mathcal{K}_{\text{pos}}(\boldsymbol{p}_{j-1}, \boldsymbol{\phi}_{j-1}) \times \mathcal{K}_{\text{ang}}(\boldsymbol{\phi}_{j-1}).$$

Here, $\mathcal{K}_{\text{pos}}(\boldsymbol{p}_{j-1}, \boldsymbol{\phi}_{j-1})$ is the set of kinematically admissible positions of the j-th footstep, defined by the following constraints:

$$-\begin{pmatrix}\Delta_x^- \\ \Delta_y^- \\ \Delta_z^-\end{pmatrix} \leq \boldsymbol{R}^T(\boldsymbol{\phi}_f^{j-1}) \begin{pmatrix}x_f^j - x_f^{j-1} \\ y_f^j - y_f^{j-1} \\ z_f^j - z_f^{j-1}\end{pmatrix} \pm \begin{pmatrix}0 \\ \ell \\ 0\end{pmatrix} \leq \begin{pmatrix}\Delta_x^+ \\ \Delta_y^+ \\ \Delta_z^+\end{pmatrix}, \tag{1}$$

where $\boldsymbol{R}(\boldsymbol{\phi}_f^{j-1})$ is the rotation matrix associated with $\boldsymbol{\phi}_f^{j-1}$ and the Δ symbols denote lower and upper maximum increments, see Figure 3.

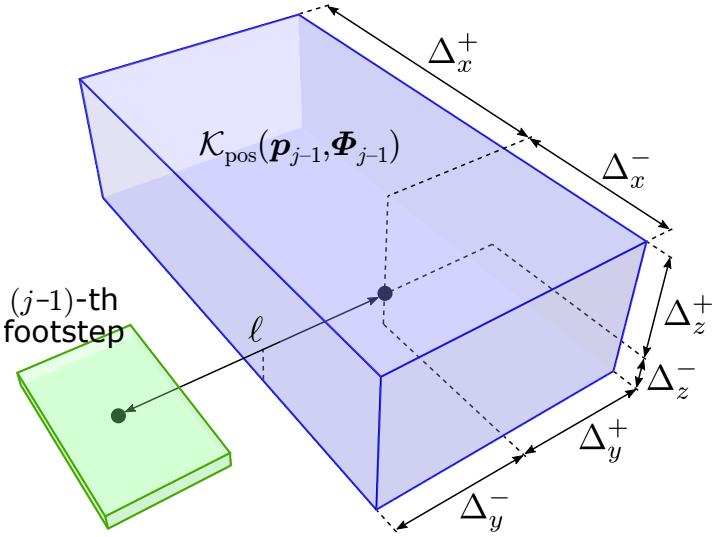

Figure 3. The kinematically admissible region $\mathcal{K}_{\text{pos}}(\boldsymbol{p}_{j-1}, \boldsymbol{\phi}_{j-1})$ for the position of the j-th footstep is defined relative to the position and orientation of the $(j-1)$-th footstep

 doi:10.20944/preprints202505.1371.v1

The set $\mathcal{K}_{\text{ang}}(\boldsymbol{\phi}_{j-1})$ of kinematically admissible orientations of the j-th footstep is defined by the following constraints:

$$\begin{aligned} -\Delta_\alpha^- &\leq \alpha_f^j \leq \Delta_\alpha^+ \\ -\Delta_\beta^- &\leq \beta_f^j \leq \Delta_\beta^+ \\ -\Delta_\gamma^- &\leq \gamma_f^j - \gamma_f^{j-1} \leq \Delta_\gamma^+. \end{aligned} \tag{2}$$

Note that the constraints on the roll and pitch angles α_f and β_f are bounds on their value, whereas the constraint on the yaw angle γ_f only limits its variation with respect to the previous footstep. This is because yaw can assume any value, whereas pitch and roll are subject to kinematic limitations that are impossible to evaluate at the planning stage, and must then be accounted for conservatively.

R3 $\boldsymbol{f}^j$ is reachable from $\boldsymbol{f}^{j-2}$ through a collision-free motion.
Since information about the whole-body motion of the robot is not yet available during footstep planning (it will only be defined in the subsequent gait generation phase), this requirement can only be tested conservatively. In particular, we say that R3 is satisfied if (*i*) a collision-free swing foot trajectory $\boldsymbol{s}^{j-2}$ from $\boldsymbol{f}^{j-2}$ to $\boldsymbol{f}^j$ can be found, and (*ii*) a suitable volume $\mathcal{B}$ accounting for the maximum occupancy of the humanoid upper body at stance $(\boldsymbol{f}^{j-1}, \boldsymbol{f}^j)$ is collision-free. More precisely, $\mathcal{B}$ is a vertical cylinder whose base has radius r_b, center at the midpoint $\boldsymbol{m}$ between the footsteps, and is raised from the ground by z_b, which represents the average distance between the ground and the hip (Figure 4).

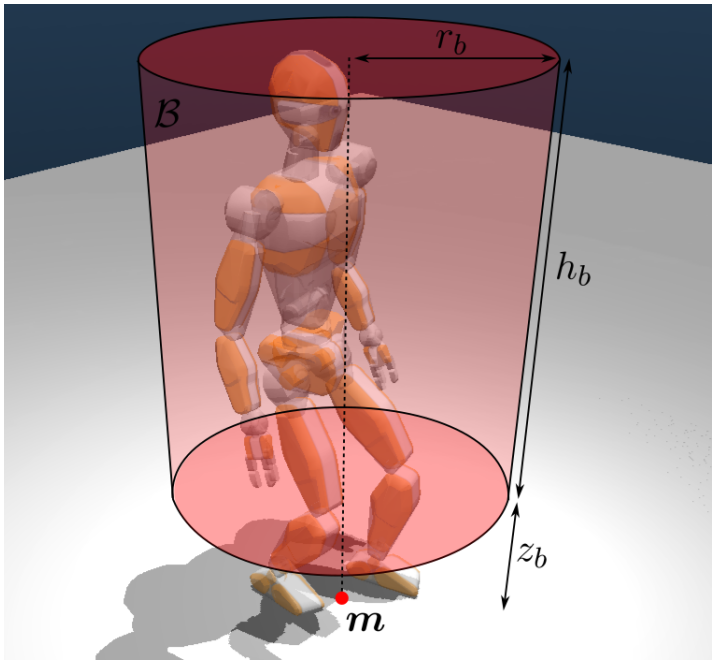

Figure 4. The robot volume occupancy (in red) used for checking requirement R3.

4.2. Algorithm

At the beginning, $\mathcal{T}$ is rooted at $(\boldsymbol{f}_{\text{swg}}^{\text{ini}}, \boldsymbol{f}_{\text{sup}}^{\text{ini}})$, the initial stance of the humanoid. Then, $\mathcal{T}$ is expanded using an RRT*-like strategy. The generic iteration consists of: selecting a vertex for expansion, generating a candidate vertex, choosing a parent for the new vertex and rewiring the tree. We will describe these individual steps below, right after some quick preliminaries.

 doi:10.20944/preprints202505.1371.v1

4.2.1. Preliminaries

We assume that an edge between two vertexes v^a and v^b of $\mathcal{T}$ has a cost $l(v^a, v^b)$. The *cost* of a vertex v is then defined as

$$c(v) = c(v^{\text{parent}}) + l(v^{\text{parent}}, v)$$

and represents the cost of reaching v from the root of $\mathcal{T}$. Then, the cost of a plan $\mathcal{P}$ ending at a vertex v is $c(\mathcal{P}) = c(v)$. In this paper, we let $l(v^a, v^b) = 1$ for all edges in $\mathcal{T}$. Correspondingly, the cost of a vertex is the length[1] of the corresponding plan in terms of number of edges (i.e., steps).

The algorithm will make use of the set of *neighbors* of a vertex $v = (f_{\text{swg}}, f_{\text{sup}})$, defined as

$$\mathcal{N}(v) = \{v' = (f'_{\text{swg}}, f'_{\text{sup}}) \in \mathcal{T} : d_1(f_{\text{sup}}, f'_{\text{sup}}) \leq \bar{d}_1\},$$

where

$$d_1(f, f') = \|\boldsymbol{p}_f - \boldsymbol{p}'_f\| + \sigma|\boldsymbol{\phi}_f - \boldsymbol{\phi}'_f|$$

is the footstep-to-footstep metric, with $\sigma \geq 0$, and $\bar{d}_1$ is a threshold distance.

4.2.2. Selecting A Vertex For Expansion

A point $\boldsymbol{p}^{\text{rand}}$ is randomly selected in $\mathbb{R}^3$, and the vertex v^{near} is identified that is closest to $\boldsymbol{p}^{\text{rand}}$ is identified according to the following vertex-to-point metric

$$d_2(v, \boldsymbol{p}) = \|\boldsymbol{m}(v) - \boldsymbol{p}\|_{\boldsymbol{W}} + |\psi(v, \boldsymbol{p})|,$$

where $\boldsymbol{m}(v)$ is the midpoint between the feet at stance v, $\boldsymbol{W}$ is a weight matrix , and $\psi(v, \boldsymbol{p})$ is the angle between the ground projection of the robot sagittal axis (whose orientation is the average of the yaw angles of the two footsteps) and the ground projection of the line joining $\boldsymbol{m}_{xy}(v)$ to $\boldsymbol{p}_{xy}$. Matrix $\boldsymbol{W}$ can be used to weigh differently vertical and horizontal displacements.

4.2.3. Generating A Candidate Vertex

After identifying the vertex $v^{\text{near}} = (f^{\text{near}}_{\text{swg}}, f^{\text{near}}_{\text{sup}})$, a candidate footstep is generated in the kinematically admissible region associated to $f^{\text{near}}_{\text{sup}}$. For simplicity, denote by $(\boldsymbol{p}^{\text{near}}, \boldsymbol{\phi}^{\text{near}})$ the pose of $f^{\text{near}}_{\text{sup}}$. We first compute the *steppable region* $\mathcal{S}$ from $f^{\text{near}}_{\text{sup}}$, i.e., the intersection between $\mathcal{K}_{\text{pos}}(\boldsymbol{p}^{\text{near}}, \boldsymbol{\phi}^{\text{near}})$ and the collection $\mathcal{M}$ of planar regions that constitute the world:

$$\mathcal{S} = \{\mathcal{K}_{\text{pos}}(\boldsymbol{p}^{\text{near}}, \boldsymbol{\phi}^{\text{near}}) \cap \mathcal{M}\}.$$

By construction, $\mathcal{S}$ itself will be a collection of planar regions. One of these regions is randomly selected, and a sample point p^{cand} is extracted from it. Also, a random yaw angle γ^{cand} is generated in $[\gamma^{\text{near}} - \Delta^-_\gamma, \gamma^{\text{near}} + \Delta^+_\gamma]$. Finally, we perform *foot snapping*, a process that correctly places the footstep at p^{cand}, with yaw angle γ^{cand}, by matching its pitch and roll angles to those of the associated planar region of $\mathcal{S}$.

Once the candidate footstep f^{cand} has been generated, it can be checked with respect to requirement R1 and R2. For the latter, note that the position constraint (1) and the last of the angular constraints (2) are automatically satisfied by the above procedure, and therefore only the pitch and roll rows of (2) must be explicitly checked.

In order to check requirement R3, we invoke an engine that generates a swing foot trajectory $\boldsymbol{s}^{\text{near}}$ from $f^{\text{near}}_{\text{swg}}$ to $f^{\text{cand}}_{\text{sup}}$. Such engine uses a Bezier curve as a parameterized trajectory; given the endpoints, such trajectory can be deformed through control points so as to change the apex height h along the motion. Once a a swing foot trajectory has been generated, it is checked for collision with the

[1] See [17] for other interesting definitions of the cost of a plan, such as the total height variation or the minimum obstacle clearance.

 doi:10.20944/preprints202505.1371.v1

environment: if a collision is detected, h is increased and the process is repeated. If a maximum height $h_{\max}$ has been reached without success, the candidate footstep is discarded.

If R1–R3 are satisfied, the candidate vertex is generated as $v^{\text{cand}} = (\boldsymbol{f}^{\text{cand}}_{\text{swg}}, \boldsymbol{f}^{\text{cand}}_{\text{sup}})$, with $\boldsymbol{f}^{\text{cand}}_{\text{swg}} = \boldsymbol{f}^{\text{near}}_{\text{sup}}$; however, adding v^{cand} to $\mathcal{T}$ is postponed, as the planner first has to choose a parent for it. If, instead, one of the requirements is violated, we abort the current expansion and start a new iteration.

4.2.4. Choosing A Parent

Although v^{cand} was generated from v^{near}, there might be a different vertex in the tree that leads to the same vertex with a lower cost. To find it, the planner checks for each vertex $v' = (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}) \in \mathcal{N}(v^{\text{cand}})$ whether setting v' as parent of v^{cand} satisfies requirements R2-R3, and whether this connection reduces the cost of v^{cand}, that is

$$c(v') + l(v', v^{\text{cand}}) < c(v^{\text{near}}) + l(v^{\text{near}}, v^{\text{cand}}).$$

The vertex $v^{\text{min}} = (\boldsymbol{f}^{\text{min}}_{\text{swg}}, \boldsymbol{f}^{\text{min}}_{\text{sup}})$ that allows to reach v^{cand} with minimum cost is chosen as its parent. If $v^{\text{min}} = v^{\text{near}}$, then v^{cand} can be added to the tree together with the edge joining it to v^{near}. However, if a different parent $v^{\text{min}} \neq v^{\text{near}}$ is chosen, the candidate vertex v^{cand} must be modified by relocating its swing footstep to the support footstep of v^{min}. To this end, a new vertex $v^{\text{new}} = (\boldsymbol{f}^{\text{new}}_{\text{swg}}, \boldsymbol{f}^{\text{new}}_{\text{sup}})$ with $\boldsymbol{f}^{\text{new}}_{\text{swg}} = \boldsymbol{f}^{\text{min}}_{\text{sup}}$ and $\boldsymbol{f}^{\text{new}}_{\text{sup}} = \boldsymbol{f}^{\text{cand}}_{\text{sup}}$ is generated and added to $\mathcal{T}$ as child of v^{min}. The edge between v^{min} and v^{new} corresponds to the swing foot trajectory $\boldsymbol{s}^{\text{min}}$.

4.2.5. Rewiring

This final step checks whether v^{new} allows to reach with a lower cost some vertex already in $\mathcal{T}$, and updates the tree accordingly. In particular, for each $v' = (\boldsymbol{f}'_{\text{swg}}, \boldsymbol{f}'_{\text{sup}}) \in \mathcal{N}(v^{\text{new}})$, the procedure checks whether setting v' as a child of v^{new} satisfies requirements R2-R3, and whether this connection reduces the cost of v', that is

$$c(v^{\text{new}}) + l(v^{\text{new}}, v') < c(v').$$

If this is the case, v' is modified (similarly to what was done when choosing a parent) by relocating its swing footstep to $\boldsymbol{f}^{\text{new}}_{\text{sup}}$, and then reconnected to $\mathcal{T}$ as a child of v_{new}. The edge between v^{new} and v' corresponds to the swing foot trajectory $\boldsymbol{s}^{\text{new}}$. Finally, for each child $v'' = (\boldsymbol{f}''_{\text{swg}}, \boldsymbol{f}''_{\text{sup}})$ of v', the swing foot trajectory $\boldsymbol{s}'$ from the relocated $\boldsymbol{f}'_{\text{swg}}$ to $\boldsymbol{f}''_{\text{sup}}$ is generated and the edge between v' and v'' is accordingly updated.

4.3. Pseudocode

For convenience, the pseudocode of the footstep planning algorithm so far described is given in Algorithm 1.

 doi:10.20944/preprints202505.1371.v1

Algorithm 1 FootstepPlanner()

Require: $f^{\text{ini}}_{\text{swg}}$, $f^{\text{ini}}_{\text{sup}}$, ΔT, $\mathcal{M}$
  Initialize($\mathcal{T}$, ($f^{\text{ini}}_{\text{swg}}$, $f^{\text{ini}}_{\text{sup}}$))
  while TimeNotExpired(ΔT) **do**
    $p^{\text{rand}} \leftarrow$ RandomPointGenerator()
    $v^{\text{near}} \leftarrow$ NearestNeighborVertex($\mathcal{T}$, p^{rand})
    $f^{\text{cand}} \leftarrow$ GenerateCandidateFootstep($f^{\text{near}}_{\text{sup}}$, $\mathcal{M}$)
    if R1(f^{cand}) AND R2(f^{cand}) **then**
      $s^{\text{near}} \leftarrow$ SwingTrajectoryGenerator($f^{\text{near}}_{\text{swg}}$, $f^{\text{cand}}_{\text{sup}}$)
      if R3(s^{near}) **then**
        $v^{\text{cand}} \leftarrow$ ($f^{\text{near}}_{\text{sup}}$, $f^{\text{cand}}_{\text{sup}}$)
        $\mathcal{N} \leftarrow$ Neighbors($\mathcal{T}$, v^{cand})
        (v^{min}, s^{min}) $\leftarrow$ ChooseParent($\mathcal{T}$, $\mathcal{N}$, v^{near}, v^{cand}, s^{near})
        $v^{\text{new}} \leftarrow$ ($f^{\text{min}}_{\text{sup}}$, $f^{\text{cand}}_{\text{sup}}$)
        $\mathcal{T} \leftarrow$ AddVertex(v^{min}, v^{new}, s^{min})
        Rewire($\mathcal{T}$, $\mathcal{N}$, v^{new})
      end if
    end if
  end while
  return

When the time budget ΔT expires, we stop tree expansion and retrieve $\mathcal{V}_{\text{goal}}$, the set of vertexes v such that $p_{f,\text{sup}} \in \mathcal{G}$. From this, we extract the vertex with minimum cost and obtain the optimal footstep plan $\mathcal{P}$ as the branch of $\mathcal{T}$ joining the root to v^*.

4.4. Footstep Planning: Numerical Results

The proposed footstep planner was implemented in C++ on a laptop equipped with an AMD Ryzen 7 6800HS CPU and an NVIDIA GeForce RTX 3050 GPU. The chosen humanoid is JVRC1, an open-source virtual robot having a total of 44 joints created for the Japan Virtual Robotics Challenge [19]. JVRC1 is similar to the HRP series humanoids in terms of mechanical design, thus representing a realistic yet convenient simulation platform.

A campaign of planning experiments took place over four scenarios of different complexity (see Figure 5):

- *Single Floor*. This environment includes elevation changes, ramps (both ascending and descending) and obstacles.
- *Multi-Floor with Stairs*. This is an environment with two floors connected by 23 steps.
- *Spiral Staircase*. A spiral staircase with 26 steps connects two floors.
- *Multi-Floor with Ramps*. In this environment, the two floors are connected by four inclined ramps and three landings.

In all scenarios, the robot has to reach a circular goal region of radius 0.3 m. In the footstep planner we have set $h_{\max} = 0.19$ m, $\Delta^-_x = 0.05$ m, $\Delta^+_x = 0.30$ m, $\Delta^-_y = 0.20$ m, $\Delta^+_y = 0.30$ m, $\Delta^-_z = \Delta^+_z = 0.12$ m, $\Delta^-_\alpha = \Delta^+_\alpha = 0.175$ rad, $\Delta^-_\beta = \Delta^+_\beta = 0.175$ rad, $\Delta^-_\gamma = \Delta^+_\gamma = 0.35$ rad, $z_b = 0.3$ m, $h_b = 1.2$ m, and $r_b = 0.25$ m.

Figure 5 shows some example of successful footstep plans obtained in the four scenarios using the proposed algorithm. Table 1 gives details about the performance of the planner in each scenario, for different values of the time budget, with each row reporting average results over 30 runs of the planner (1st Plan is the time needed by the footstep planner to generate the first plan that reaches the goal). A run is considered unsuccessful if the planner terminates without placing any footstep in the goal region. Note that increasing the time budget consistently improves the success rate, suggesting that the proposed footstep planner is probabilistically complete.

 doi:10.20944/preprints202505.1371.v1

Overall, the results highlight the effectiveness of the proposed planner in rather complex 3D scenarios, even in the presence of stairs and ramps. See the concluding section for some hints at possible improvements of our planner.

Table 1. Performance of the footstep planner in the four scenarios

Scenario	Time Budget [s]	Avg Cost	Min Cost	Max Cost	Iters	Tree Size	1st Plan [s]	Successes
Single-Floor	10	57.3	46.0	67.0	10685	7392	4.0	19/30
	30	61.5	49.0	83.0	18458	12890	11.22	28/30
	60	61.3	46.0	75.0	26370	18457	13.67	29/30
Multi-Floor with Stairs	10	94.0	86.0	104.0	11290	7026	6.27	14/30
	30	94.2	85.0	102.0	19297	12500	8.94	27/30
	60	93.3	78.0	103.0	27933	18119	12.51	29/30
Spiral Staircase	10	73.3	68.0	114.0	13316	6654	3.98	24/30
	30	72.8	59.0	108.0	21771	12291	5.37	29/30
	60	66.9	59.0	73.0	29247	17571	5.59	30/30
Multi-Floor with Ramps	10	103.7	100.0	111.0	10382	7997	8.64	3/30
	30	110.0	96.0	123.0	17765	13876	19.73	19/30
	60	107.5	94.0	130.0	25924	19695	21.88	29/30

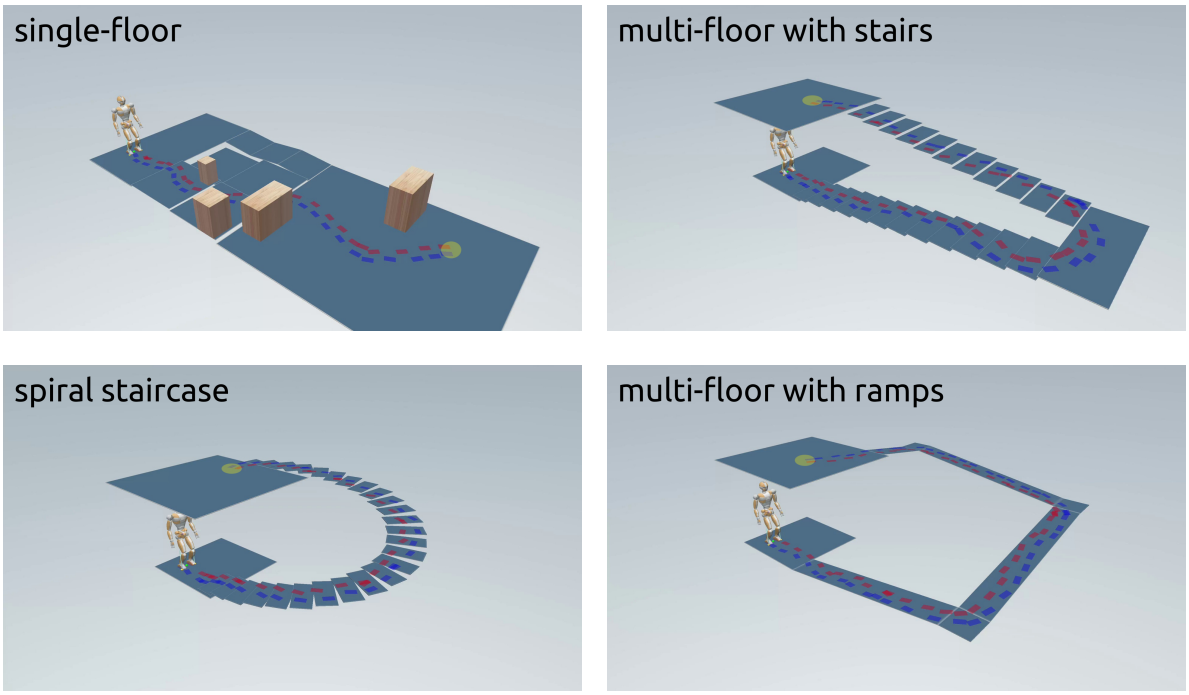

Figure 5. Examples of footstep plans found in the four considered scenarios.

5. Control Architecture

This section describes the control pipeline, which consists of the IS-MPC module followed by the whole-body controller (see Figure 2).

5.1. Is-Mpc Gait Generation

Gait generation is the process of generating a feasible trajectory for the CoM position $\boldsymbol{p}_c = (x_c, y_c, z_c)$, such that the robot realizes the assigned footstep plan and maintains balance. This last requirement can be formulated as a constraint on the position $\boldsymbol{p}_z = (x_z, y_z, z_z)$ of the Zero Moment Point (ZMP), which is the point of application of the equivalent ground reaction force.

To generate the CoM trajectory, we use Model Predictive Control (MPC), which optimizes trajectories over a certain horizon, using a model to predict the evolution of the system.

 doi:10.20944/preprints202505.1371.v1

5.1.1. Prediction Model

Our prediction model is a 3D extension of the classic Linear Inverted Pendulum (LIP), which has the ZMP as input and allows for vertical motion of the CoM [15,20]. By neglecting the internal angular momentum variation, we have

$$\begin{aligned}(z_c - z_z)\,\ddot{x}_c &= (x_c - x_z)(\ddot{z}_c - g)\\ (z_c - z_z)\,\ddot{y}_c &= (y_c - y_z)(\ddot{z}_c - g),\end{aligned}$$

where g is the gravity acceleration. To remove the nonlinear coupling between the horizontal and vertical components, we impose $(\ddot{z}_c - g)/(z_c - z_z) = \eta^2$, where η is a constant parameter. The resulting model is expressed as

$$\ddot{\boldsymbol{p}}_c = \eta^2(\boldsymbol{p}_c - \boldsymbol{p}_z) - \boldsymbol{g}, \tag{3}$$

where $\boldsymbol{g} = (0, 0, g)$ is a constant drift. This means the 3D LIP is at an equilibrium when the CoM and ZMP are vertically displaced by η/g; this should be taken into account when choosing the value of η. In our formulation, we dynamically extend (3) so that the input is the ZMP velocity $\dot{\boldsymbol{p}}_z$, rather than the position $\boldsymbol{p}_z$, in order to generate smoother trajectories.

5.1.2. ZMP constraints

In the classic LIP, the condition for balance requires the ZMP to stay within of the support polygon of the robot, which however is no more defined in a 3D setting. A sufficient condition for ensuring balance in this case is that the ZMP must belong to a pyramidal region $\mathcal{Z}$, with apex at the robot CoM and edges going through the vertexes of the convex hull of the contact surfaces. Since this condition is nonlinear, we resort to the following conservative approximation.

Consider a *moving box* $\mathcal{B}$ with constant dimensions (d_x, d_y, d_z) along the three axes. Its center $\boldsymbol{p}_{\text{mc}} = (x_{\text{mc}}, y_{\text{mc}}, z_{\text{mc}})$ moves in the following way: during single support phases it coincides with the position of the support foot, while during double support phases it performs a roto-translation from the position of the previous support foot to the position of the next. By properly choosing d_x, d_y and d_z one may guarantee that $\mathcal{B}$ is always contained in $\mathcal{Z}$ [17].

At a generic time instant t_j, a linear ZMP constraint can then be written as

$$-\frac{1}{2}\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \leq \boldsymbol{R}_j^T(\boldsymbol{p}_z^j - \boldsymbol{p}_{\text{mc}}^j) \leq \frac{1}{2}\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}, \tag{4}$$

where $\boldsymbol{R}_j$ is the rotation matrix associated to the orientation of the moving box at t_j, and the j superscript denotes the variable at t_j.

5.1.3. Stability Constraint

While keeping the ZMP inside $\mathcal{B}$ (and therefore, inside $\mathcal{Z}$) is sufficient for balance, it does not by itself guarantee that the COM trajectories are bounded with respect to the ZMP. This is due to the well-known fact that the LIP model includes one unstable mode, which can be isolated by defining the variable $\boldsymbol{p}_u = \boldsymbol{p}_c + \dot{\boldsymbol{p}}_c/\eta$.

To avoid divergence, IS-MPC enforces the following *stability condition*:

$$\boldsymbol{p}_u^k = \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z(\tau) d\tau + \frac{\boldsymbol{g}}{\eta}.$$

This is a relationship between the current value of $\boldsymbol{p}_u$ and the future of the ZMP trajectory, and is therefore noncausal. To obtain a causal condition, we split the integral in two parts, the first from t_k to t_{k+C} (the final instant of the prediction horizon) and the second from t_{k+C} to infinity. While the first integral can be expressed in terms of the MPC decision variables $\dot{\boldsymbol{p}}_z^k, \ldots, \dot{\boldsymbol{p}}_z^{k+C-1}$, the second can be

 doi:10.20944/preprints202505.1371.v1

computed by conjecturing a trajectory $\tilde{\boldsymbol{p}}_z$ of $\boldsymbol{p}_z$ (*anticipative tail*) on the basis of information contained in the footstep plan:

$$\boldsymbol{c}_u^k = \eta \int_{t_{k+C}}^{\infty} e^{-\eta(\tau - t_k)} \tilde{\boldsymbol{p}}_z(\tau) d\tau.$$

The stability constraint can then be written as

$$\boldsymbol{p}_u^k = \eta \int_{t_k}^{t_{k+C}} e^{-\eta(\tau - t_k)} \boldsymbol{p}_z(\tau) d\tau + \boldsymbol{c}_u^k + \frac{\boldsymbol{g}}{\eta}. \tag{5}$$

5.1.4. IS-MPC

At each iteration, IS-MPC solves the following quadratic program:

$$\begin{aligned} \min_{\dot{\boldsymbol{p}}_z^k, \ldots, \dot{\boldsymbol{p}}_z^{k+C-1}} \quad & \sum_{i=k}^{k+C-1} ||\dot{\boldsymbol{p}}_z^i||^2 + \lambda ||\boldsymbol{p}_z^i - \boldsymbol{p}_{\text{mc}}^i||^2 \\ \text{subject to:} \quad & \text{ZMP constraints (4), for } j = k, \ldots, k+C \\ & \text{stability constraint (5),} \end{aligned}$$

where λ is a positive weight. After solving the quadratic program, the first optimal sample $\dot{\boldsymbol{p}}_z^k$ is used to integrate the dynamically extended model (3) and obtain the next sample of the reference CoM position $\boldsymbol{p}_c^*$ and velocity $\dot{\boldsymbol{p}}_c^*$, as well as of the corresponding ZMP position $\boldsymbol{p}_z^*$. The corresponding CoM acceleration can then be computed from (3). All these terms are sent to the whole-body control module, where they are used to set up the CoM tracking task.

5.2. Whole-Body Controller

Commands to be sent to the robot joints are generated by a whole-body controller which works in two stages. A first stage determines appropriate joint accelerations and contact forces by solving a constrained quadratic program, and a second stage computes the associated joint torques via inverse dynamics.

5.2.1. Task References

The input to the whole-body controller is a set of reference trajectories, one for each task. The set of tasks is $\mathcal{T} = \{com, left, right, torso, joint\}$, which respectively represent:

- *com*: the CoM reference, generated by the IS-MPC module;
- *left/right*: foot pose references, generated by the footstep planner;
- *torso*: a reference torso orientation, typically chosen to be upright;
- *joint*: a reference joint posture (for solving kinematic redundancy).

Denote the generic task variable by $\boldsymbol{t}_i$, $i = 1, \ldots, n_t$, and its reference position, velocity and acceleration by $\boldsymbol{t}_i^*$, $\dot{\boldsymbol{t}}_i^*$ and $\ddot{\boldsymbol{t}}_i^*$, respectively.[2] The desired acceleration $\ddot{\boldsymbol{t}}_i$ is then defined as

$$\ddot{\boldsymbol{t}}_i = \ddot{\boldsymbol{t}}_i^* + k_p(\boldsymbol{t}_i^* - \boldsymbol{t}_i) + k_d(\dot{\boldsymbol{t}}_i^* - \dot{\boldsymbol{t}}_i),$$

where the reference acceleration is used as feedforward and the PD feedback is aimed at robustifying kinematic inversion. The desired acceleration $\ddot{\boldsymbol{t}}_i$ will be realized provided that the following relationship holds:

$$\boldsymbol{J}_i \dot{\boldsymbol{\nu}} + \dot{\boldsymbol{J}}_i \boldsymbol{\nu} = \ddot{\boldsymbol{t}}_i^* + k_p(\boldsymbol{t}_i^* - \boldsymbol{t}_i) + k_d(\dot{\boldsymbol{t}}_i^* - \dot{\boldsymbol{t}}_i),$$

where $\boldsymbol{J}_i$ is the Jacobian of the i-th task.

[2] With a small abuse, we are using a time derivative notation also for angular task variables.

 doi:10.20944/preprints202505.1371.v1

5.2.2. Robot Dynamics

The robot dynamics can be written as

$$\boldsymbol{M}(\boldsymbol{q})\,\dot{\boldsymbol{\nu}} + \boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu}) = \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{\tau}\end{pmatrix} + \sum_{i=1}^{n_c} \boldsymbol{J}_{c,i}^{T}(\boldsymbol{q})\boldsymbol{f}_{c,i}. \tag{6}$$

Here, $\boldsymbol{M}$ is the inertia matrix, the configuration $\boldsymbol{q} = (\boldsymbol{q}_u, \boldsymbol{q}_a)$ is the stack of the floating base (unactuated) variables and the joint (actuated) variables, $\boldsymbol{\nu} = (\boldsymbol{\nu}_u, \dot{\boldsymbol{q}}_a)$ contains the corresponding pseudovelocities, $\boldsymbol{n}$ collects Coriolis, centrifugal and gravitational terms, $\boldsymbol{\tau}$ are the joint torques, and $\boldsymbol{J}_{c,i}$ denotes the Jacobian of the point of application of the i-th contact force $\boldsymbol{f}_{c,i}$, with $i = 1, \ldots, n_c$.

Equation (6) can be partitioned to highlight the unactuated and the actuated dynamics:

$$\begin{pmatrix}\boldsymbol{M}_u\\ \boldsymbol{M}_a\end{pmatrix}\dot{\boldsymbol{\nu}} + \begin{pmatrix}\boldsymbol{n}_u\\ \boldsymbol{n}_a\end{pmatrix} = \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{\tau}\end{pmatrix} + \sum_{i=1}^{n_c}\begin{pmatrix}\boldsymbol{J}_{ci,u}^{T}\\ \boldsymbol{J}_{ci,a}^{T}\end{pmatrix}\boldsymbol{f}_{c,i}, \tag{7}$$

where we have omitted the dependence on $\boldsymbol{q}$ and $\boldsymbol{\nu}$ for compactness.

5.2.3. Constraints

The first constraint enforces the unactuated part of (7):

$$\boldsymbol{M}_u\,\dot{\boldsymbol{\nu}} + \boldsymbol{n}_u = \sum_{i=1}^{n_c} \boldsymbol{J}_{ci,u}^{T}\boldsymbol{f}_{c,i}. \tag{8}$$

We do not directly enforce the actuated dynamics (which would require adding $\boldsymbol{\tau}$ as a decision variable) because $\boldsymbol{\tau}$ will not explicitly appear in our quadratic program; therefore, it can be computed a posteriori from the inverse dynamics without affecting the optimization process.[3]

The second set of constraints guarantees stationary contacts by imposing zero acceleration to the left and/or right foot, depending on the specific support phase:

$$\boldsymbol{J}_{\text{left/right}}\,\dot{\boldsymbol{\nu}} + \dot{\boldsymbol{J}}_{\text{left/right}}\,\boldsymbol{\nu} = \boldsymbol{0}. \tag{9}$$

The third set of constraints enforces limits on joint velocities and positions:

$$\dot{\boldsymbol{q}}_{a,\min} \le \dot{\boldsymbol{q}}_a + \delta\,\ddot{\boldsymbol{q}}_a \le \dot{\boldsymbol{q}}_{a,\max} \tag{10}$$

$$\boldsymbol{q}_{a,\min} \le \boldsymbol{q}_a + \delta\,\dot{\boldsymbol{q}}_a + \tfrac{\delta^2}{2}\,\ddot{\boldsymbol{q}}_a \le \boldsymbol{q}_{a,\max}, \tag{11}$$

where δ is the duration of a sampling interval.

The last set of constraints requires each contact force $\boldsymbol{f}_{c,i}$ to be contained in the friction cone, ensuring that no slipping occurs:

$$-\mu\, f_{ci,n} \le f_{ci,x} \le \mu\, f_{ci,n} \tag{12}$$

$$-\mu\, f_{ci,n} \le f_{ci,y} \le \mu\, f_{ci,n}, \tag{13}$$

where μ is the friction coefficient, $f_{ci,n}$ is the normal component of $\boldsymbol{f}_{c,i}$, and $f_{ci,x}, f_{ci,y}$ are its tangential components along the x and y axes of the foot frame at the contact. Note that *(i)* a pyramidal approximation of the friction cone has been used to maintain linearity, and *(ii)* these constraints automatically imply $f_{ci,n} > 0$, i.e., unilaterality of the contact forces.

[3] If one wants to minimize torque effort or impose torque limits, $\boldsymbol{\tau}$ must be included in the decision variables, and then the actuated dynamics should be added as a constraint.

 doi:10.20944/preprints202505.1371.v1

5.2.4. Quadratic Program

At each iteration, the first stage of the whole-body controller solves the following QP problem:

$$\begin{aligned} \min_{\dot{\boldsymbol{\nu}}, f_1, \ldots, f_{nc}} \quad & \sum_{i=1}^{n_t} w_i \left\| \boldsymbol{J}_i \dot{\boldsymbol{\nu}} + \dot{\boldsymbol{J}}_i \boldsymbol{\nu} - \ddot{\boldsymbol{t}}_i^* - k_p(\boldsymbol{t}_i^* - \boldsymbol{t}_i) - k_d(\dot{\boldsymbol{t}}_i^* - \dot{\boldsymbol{t}}_i) \right\|^2 \\ \text{subject to} \quad & \text{unactuated dynamics (8)}, \\ & \text{stationary contact constraints (9)}, \\ & \text{joint velocity constraints (10)}, \\ & \text{joint position constraints (11)}, \\ & \text{contact force constraints (12), (13)}, \; i = 1, \ldots, n_c. \end{aligned}$$

5.2.5. Inverse Dynamics

Once $\dot{\boldsymbol{\nu}}$ and the contact forces $\boldsymbol{f}_{c,i}$ have been obtained by solving the above QP, the joint torques can be computed using the actuated dynamics in (7), i.e.,

$$\boldsymbol{\tau} = \boldsymbol{M}_a \dot{\boldsymbol{\nu}} + \boldsymbol{n}_a - \sum_{i=1}^{n_c} \boldsymbol{J}_{ci,a}^T \boldsymbol{f}_{c,i}.$$

6. Simulations

We now complement the footstep planning results of Sect. 4.4 by presenting dynamic simulations of the JVRC1 robot executing the planned motions in MuJoCo, which guarantees high physical accuracy. IS-MPC runs at 100 Hz, uses a sampling interval of 0.1 s and a control horizon of 2 s, whereas the whole-body controller runs at 1000 Hz (i.e., $\delta = 1$ ms). Contacts are modeled with 4 forces per foot, so that $n_c = 4$ or 8 depending on the support phase. Finally, all quadratic programs (QP) are solved with HPIPM [21], which requires under 1 ms to compute a solution, ensuring real-time capability.

The simulation results are shown via successive snapshots in Figures 6–9 and, in form of video clips, at https://youtu.be/RFPrQjY2tD8. As expected, the robot was able to efficiently and robustly realize all footstep plans, even in the presence of stairs or inclined ramps.

 doi:10.20944/preprints202505.1371.v1

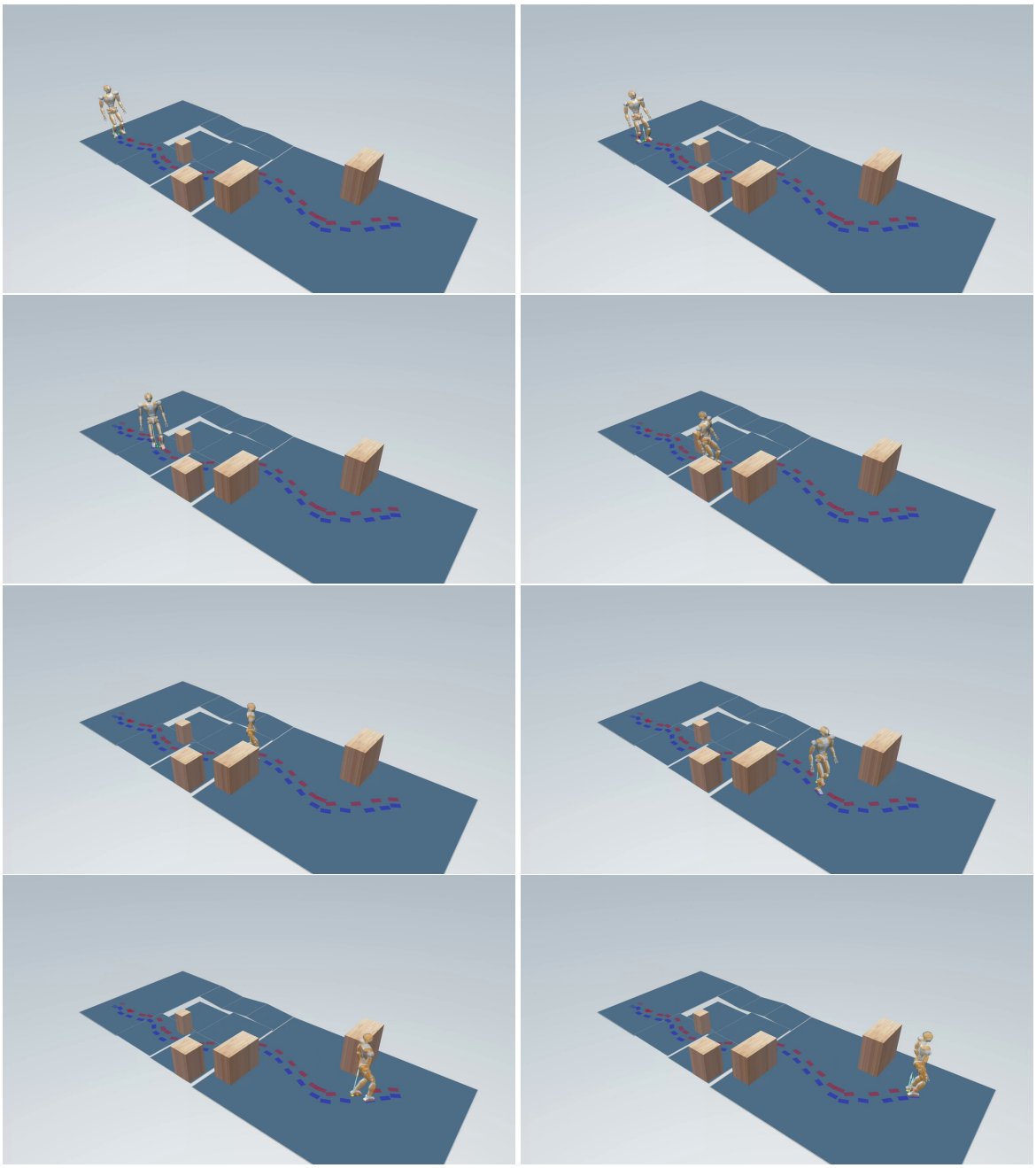

Figure 6. Single-Floor: the robot traverses some horizontal patches at different levels while treating others as obstacles due to their height.

 doi:10.20944/preprints202505.1371.v1

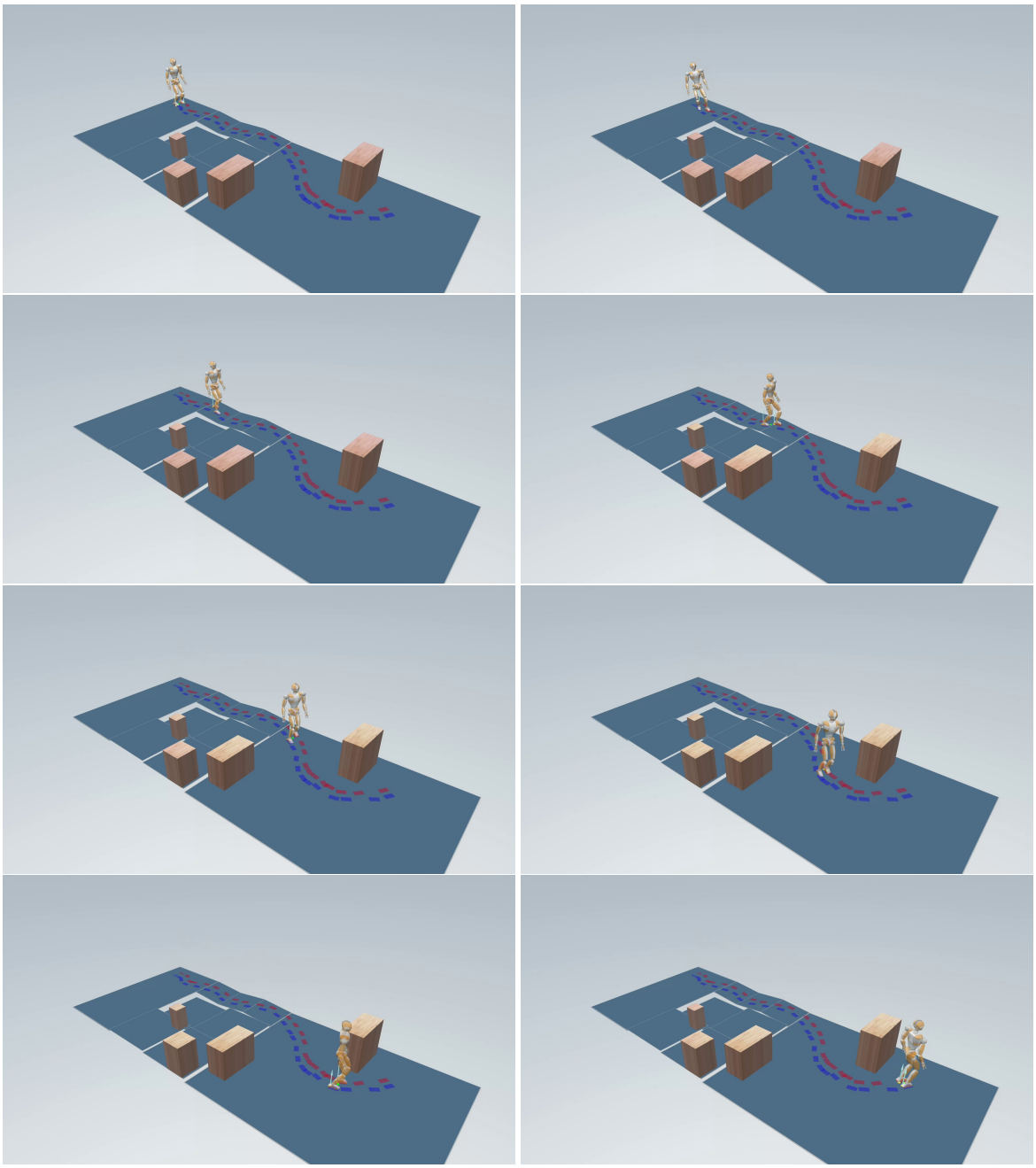

Figure 7. Single-Floor with a different start: the robot now takes a different route going through some inclined ramps.

 doi:10.20944/preprints202505.1371.v1

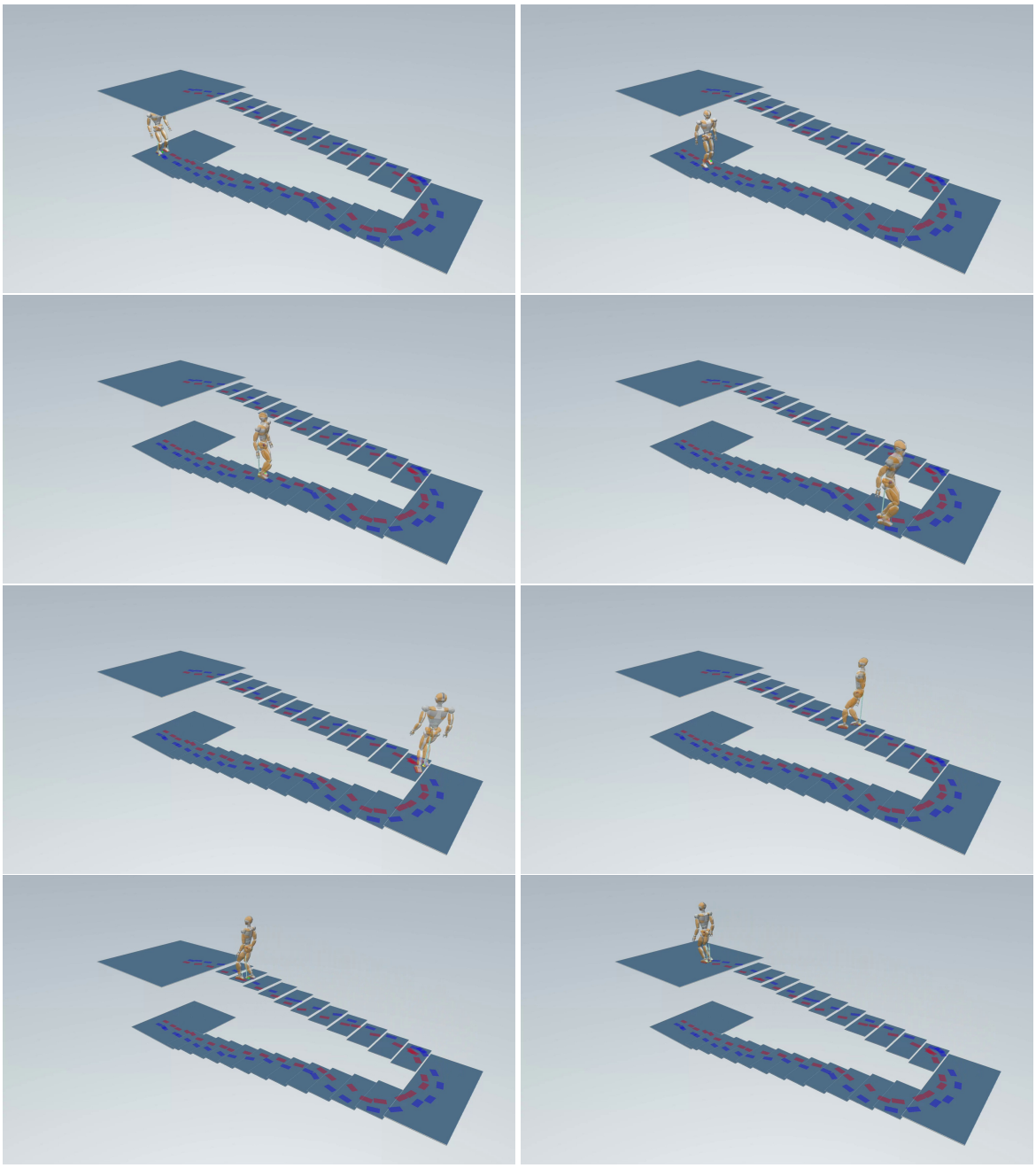

Figure 8. Multi-Floor with Stairs: since the start and the goal are on different floors the robot climbs the whole staircase.

 doi:10.20944/preprints202505.1371.v1

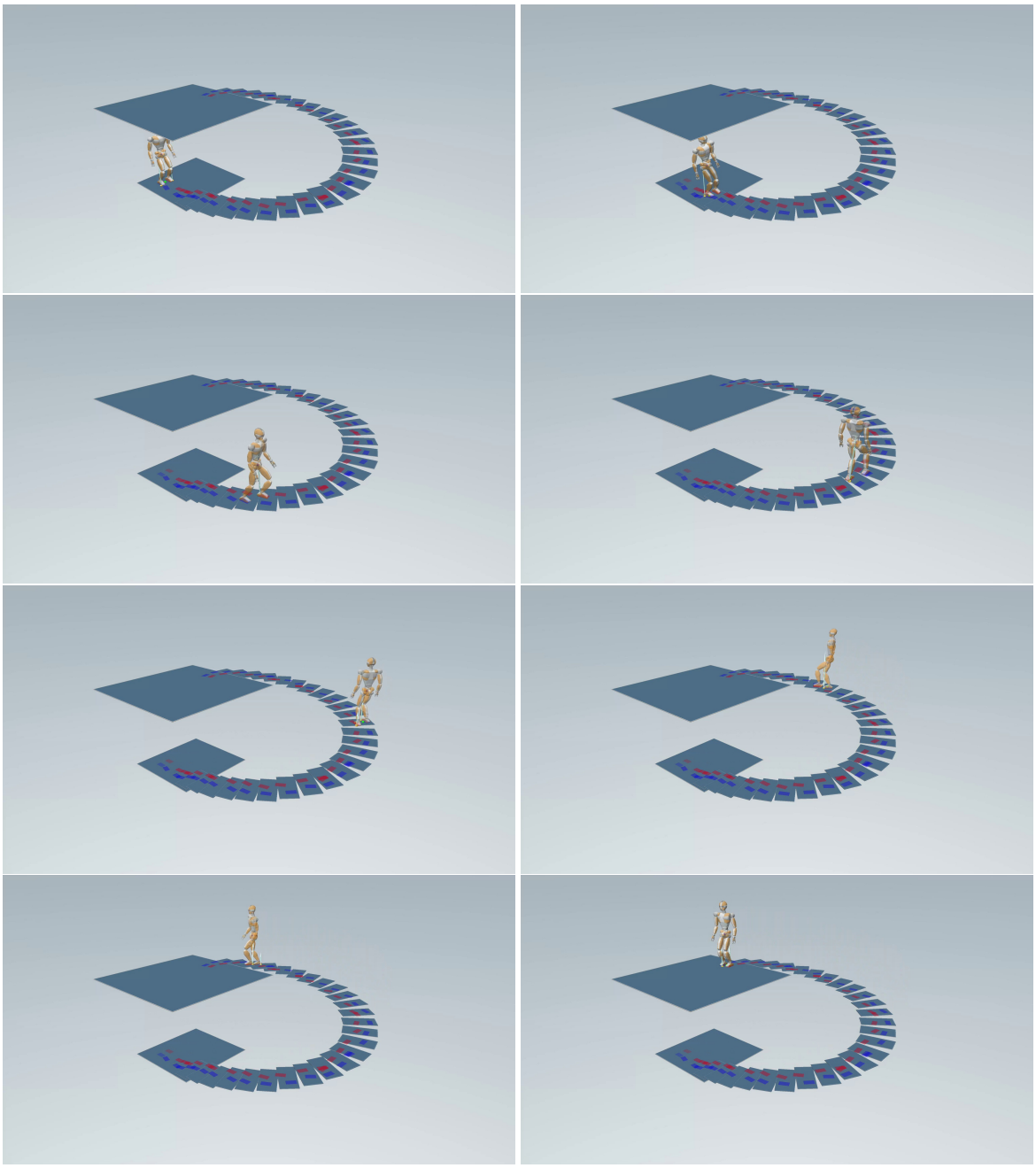

Figure 9. Spiral Staircase: as before, the robot climbs the whole spiral staircase to reach the upper floor.

 doi:10.20944/preprints202505.1371.v1

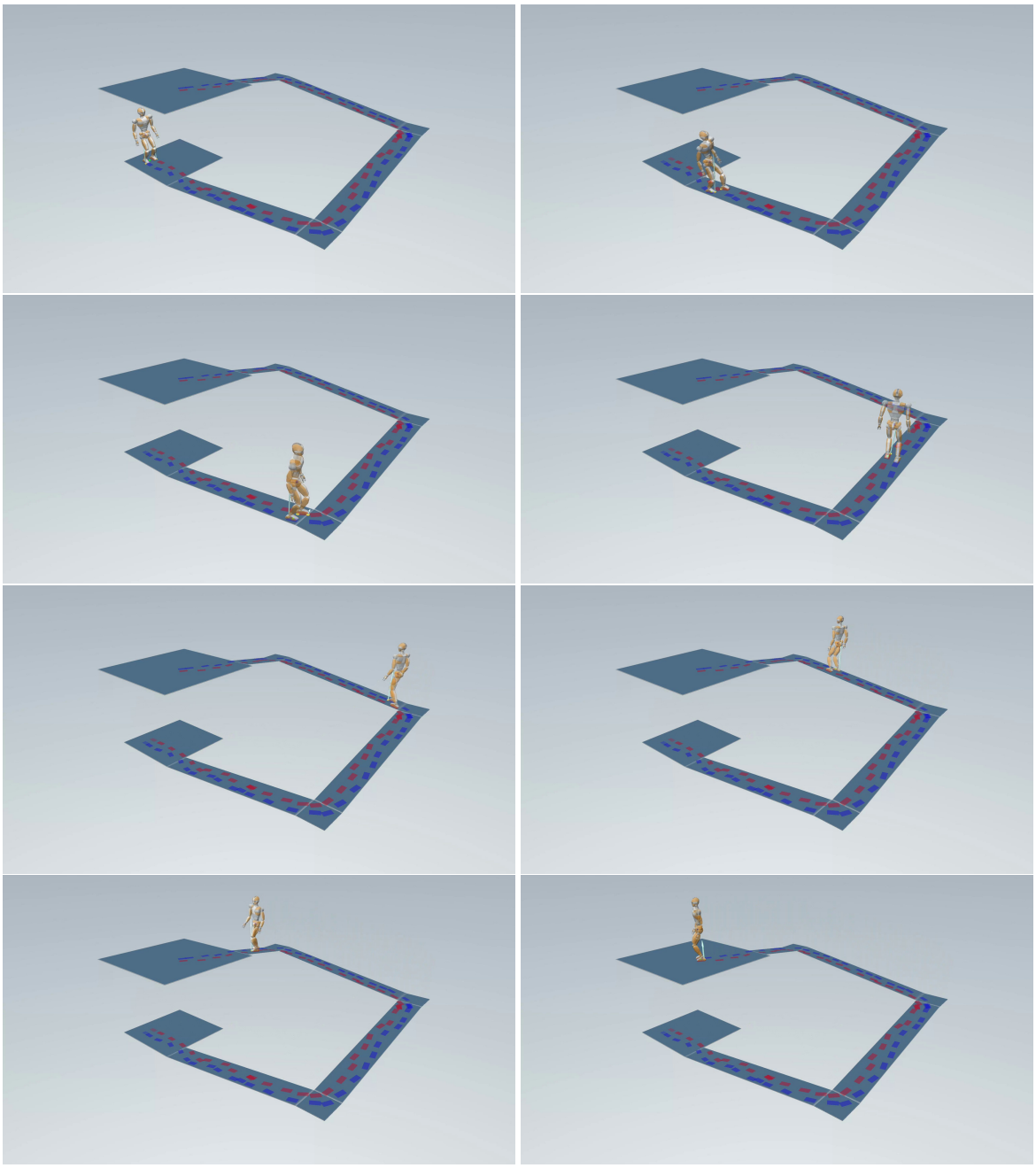

Figure 10. Multi-Floor with Ramps: in this last scenario, the robot reaches the upper floor going through a sequence of inclined ramps.

7. Conclusions

The proposed framework represents a humanoid motion generation scheme that enables navigation in complex 3D environments, including multiple floors and inclined surfaces. We ran an extensive campaign in order to validate the performance of the pipeline (footstep planner, gait generation and whole body control) in several different scenarios via dynamic simulations on a JVRC1 robot in MuJoCo. Our results show that our framework is effective and robust in very complex environments, paving the way for further promising research.

Future work will explore several directions, including:

- speed up the footstep planner by adopting a k-d tree [18] data structure and introducing waypoints (intermediate goals);
- devise an on-line version of the planner that can be used as the robot moves and gathers new information about the environment;

 doi:10.20944/preprints202505.1371.v1

- use a whole-body MPC [22] for enhancing agility and allowing more complex movements.

Author Contributions: The individual contributions for this article were as follows: Conceptualization, M.C., N.S., L.L., and G.O.; methodology, D.M., M.C., N.S., L.L., and G.O.; software, D.M., M.C., and N.S.; validation, D.M., M.C., and N.S.; formal analysis, D.M., M.C., N.S., L.L., and G.O.; investigation, D.M., M.C., N.S., L.L., and G.O.; resources, M.C., N.S., L.L., and G.O.; data curation, D.M., M.C., N.S., L.L., and G.O.; writing—original draft preparation, D.M., M.C., N.S., L.L., and G.O.; writing—review and editing, M.C., N.S., L.L., and G.O.; visualization, D.M., M.C., N.S., L.L., and G.O.; supervision, M.C., N.S., L.L., and G.O.; project administration, M.C., N.S., L.L., and G.O.; funding acquisition, L.L. and G.O. All authors have read and agreed to the published version of the manuscript.

Funding: Nicola Scianca has been fully supported by PNRR MUR project PE0000013-FAIR.

Data Availability Statement: Data will be available upon request from the corresponding authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hauser, K.; Bretl, T.; Latombe, J.C.; Harada, K.; Wilcox, B. Motion Planning for Legged Robots on Varied Terrain. *The International Journal of Robotics Research* **2008**, *27*, 1325–1349.
2. Chestnutt, J.; Lau, M.; Cheung, G.; Kuffner, J.; Hodgins, J.; Kanade, T. Footstep Planning for the Honda ASIMO Humanoid. In Proceedings of the 2005 IEEE Int. Conf. on Robotics and Automation (ICRA), 2005, pp. 629–634.
3. Gutmann, J.S.; Fukuchi, M.; Fujita, M. Real-time path planning for humanoid robot navigation. In Proceedings of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI), 2005, pp. 1232–1237.
4. Griffin, R.J.; Wiedebach, G.; McCrory, S.; Bertrand, S.; Lee, I.; Pratt, J. Footstep Planning for Autonomous Walking Over Rough Terrain. In Proceedings of the 2019 IEEE Int. Conf. on Robotics and Automation (ICRA), 2019, pp. 9–16.
5. Mishra, B.; Calvert, D.; Bertrand, S.; McCrory, S.; Griffin, R.; Sevil, H.E. GPU-accelerated rapid planar region extraction for dynamic behaviors on legged robots. In Proceedings of the 2021 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). IEEE, 2021, pp. 8493–8499.
6. Mishra, B.; Calvert, D.; Bertrand, S.; Pratt, J.; Sevil, H.E.; Griffin, R. Efficient Terrain Map Using Planar Regions for Footstep Planning on Humanoid Robots. In Proceedings of the 2024 IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids). IEEE, 2024, pp. 8044–8050.
7. Liu, H.; Sun, Q.; Zhang, T. Hierarchical RRT for Humanoid Robot Footstep Planning with Multiple Constraints in Complex Environments. In Proceedings of the 2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2012, pp. 3187–3194.
8. Wieber, P.B. Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations. In Proceedings of the 6th IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids), 2006, pp. 137–142.
9. Caron, S.; Escande, A.; Lanari, L.; Mallein, B. Capturability-based Pattern Generation for Walking with Variable Height. *IEEE Transactions on Robotics* **2019**, *36*, 517–536.
10. Englsberger, J.; Ott, C.; Albu-Schäffer, A. Three-Dimensional Bipedal Walking Control Based on Divergent Component of Motion. *IEEE Transactions on Robotics* **2015**, *31*, 355–368.
11. Han, Y.H.; Cho, B.K. Slope walking of humanoid robot without IMU sensor on an unknown slope. *Robotics and autonomous systems* **2022**, *155*, 104163.
12. Wiedebach, G.; Bertrand, S.; Wu, T.; Fiorio, L.; McCrory, S.; Griffin, R.; Nori, F.; Pratt, J. Walking on partial footholds including line contacts with the humanoid robot Atlas. In Proceedings of the 2016 IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids). IEEE, 2016, pp. 1312–1319.
13. Caron, S.; Pham, Q.C.; Nakamura, Y. ZMP support areas for multicontact mobility under frictional constraints. *IEEE Transactions on Robotics* **2016**, *33*, 67–80.
14. Scianca, N.; De Simone, D.; Lanari, L.; Oriolo, G. MPC for Humanoid Gait Generation: Stability and Feasibility. *IEEE Transactions on Robotics* **2020**, *36*, 1171–1178.
15. Zamparelli, A.; Scianca, N.; Lanari, L.; Oriolo, G. Humanoid Gait Generation on Uneven Ground using Intrinsically Stable MPC. *IFAC-PapersOnLine* **2018**, *51*, 393–398.

 doi:10.20944/preprints202505.1371.v1

16. Smaldone, F.M.; Scianca, N.; Lanari, L.; Oriolo, G. From walking to running: 3D humanoid gait generation via MPC. *Frontiers in Robotics and AI* **2022**, *9*, 876613.
17. Cipriano, M.; Ferrari, P.; Scianca, N.; Lanari, L.; Oriolo, G. Humanoid motion generation in a world of stairs. *Robotics and Autonomous Systems* **2023**, *168*, 104495.
18. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. of Robotics Research* **2011**, *30*, 846–894.
19. Okugawa, M.; Oogane, K.; Shimizu, M.; Ohtsubo, Y.; Kimura, T.; Takahashi, T.; Tadokoro, S. Proposal of inspection and rescue tasks for tunnel disasters — Task development of Japan virtual robotics challenge. In Proceedings of the 2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2015, pp. 1–2.
20. Smaldone, F.M.; Scianca, N.; Lanari, L.; Oriolo, G. From Walking to Running: 3D Humanoid Gait Generation via MPC. *Frontiers in Robotics and AI* **2022**, *9*.
21. Frison, G.; Diehl, M. HPIPM: A high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine* **2020**, *53*, 6563–6569.
22. Belvedere, T.; Scianca, N.; Lanari, L.; Oriolo, G. Joint-Level IS-MPC: A Whole-Body MPC with Centroidal Feasibility for Humanoid Locomotion. In Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024, pp. 11240–11247.