FP1. Autonomous humanoid navigation in multi-floor environments

Edoardo Colonna, Silverio Manganaro, Matteo Zaramella
1697124 - 1817504 - 2025806 *

April 23, 2023

Abstract

In this report, we explain the work that has been carried out for the project of the exam of Autonomous and Mobile Robotics. Starting from recent papers novelties, we have extended the previously proposed methodologies for navigation in complex uneven terrains by developing a footstep planner based on a randomized algorithm for an autonomous humanoid robot in a multi-floor environment. This was implemented by leveraging a multi-level surface map. Tests have been carried out on three different environments, each of them with peculiar characteristics, in order to assess the correct behaviour. Finally, the results were validated with simulations on Coppellia Sim using a gait generator based on IS-MCP for computing the CoM trajectory.

1 Introduction

Recent advances in planning and control techniques for humanoid robots have demonstrated the possibility of making them navigate in complex uneven terrains. The ability of legged locomotion allows to move and reach spaces that were originally created for human motions. Although this possibility is naturally convenient, it is really complex to handle all the characteristics of a non-flat and bumpy terrain. Stairs, drops, and surfaces at different heights are situations in which standard methodologies for humanoid robots could fail. This is because some assumptions are taken, such as flat horizontal ground and CoM height constant, in order to obtain a linear approximation of the dynamic model, that is the Linear Inverted Pendulum (LIP) [1]. These constraints are lost in uneven terrain while new ones act on the robot, like the fact that one footstep should be placed entirely in an area with the same height or avoid walking near edges. Another problem the robot could encounter is in evaluating if a surface at a different height should be considered as an obstacle or as a walkable area. So, in general, extending the models used for flat worlds to 3D environments can lead, without additional considerations, to nonlinearities which can severely impact the performance and affect the success of the mobility task of a humanoid robot.

In this project, we deal with a specific kind of uneven terrain, which is called a world of stairs. This environment is composed of flat patches at different heights. So, in this case, we do not have to deal with a tilt of the floor, another possible complication of uneven environments, but only with the elevations of different areas of the map. Our purpose is to develop a planning algorithm able to act correctly in this scenario and to generate a list of feasible footsteps to reach a goal region.

*The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Italy. E-mail: *lastname.id*@studenti.uniroma1.it. We especially thank Michele Cipriano (cipriano@diag.uniroma1.it) for the help he gave us in developing this work.

This report is structured as follows: starting by talking about related works in section 2, then it will explain the formulation of the problem, the characteristics of the environment and what the agent can do in it in section 3. Then the Multi-Level surface map will be described in depth in section 4. Subsequently, the footstep planner procedure will be reported in section 5, along with the MPC-based control scheme which computes the CoM trajectory in section 6. After that, we will show the results obtained with an explanation of the tests performed in section 7. Lastly will be offered some concluding remarks in section 8.

2 Related Works

For this kind of problem, different approaches have already been proposed. One possibility is to use continuous optimization techniques, which do not restrict possible steps to a finite set [2]. With this procedure, there is a major issue with pre-computation time, since you have to find sufficiently many convex obstacle-free regions to ensure that a path through the environment can be found and this can be very time-expensive, making this technique not practical for real-world scenarios. Another approach consists in assembling a path by using a finite list of footstep displacement. This can be done by searching in the list of footsteps either deterministically or randomly. The difference between these two is essentially in how the tree is expanded.

A method that can be used is a family of algorithms based on well-known A*, which has been able to produce optimal [3] or sub-optimal [4] footstep plans. This method has the same drawbacks as A*: it is strongly reliant on the heuristic chosen to expand the nodes of the three, which is hard to find and usually works well only in that particular scenario, and so loose in generalization. Moreover, expanding a node for all its successors could be impractical in many cases, especially in uneven terrains, and clearly too expensive. In fact, it is important to recall that in this scenario the number of constraints is higher with respect to the flat-world case, and so verifying the feasibility of a footstep takes more computations.

Another possible solution is to use a family of random sampling-based algorithms that are based mostly on RRT (Rapidly-Exploring Random Tree). Some variations of RRT proposed in [5] aim to obtain a behaviour similar to A* by trying at each iteration all the possible node expansions to ensure the goal-directedness, ending in the same dimensional issue seen before. Instead, a more practical approach has been used in [6] where the authors draw randomly from a limited list of primitives to expand the tree. The primitives are defined without limiting the possibility of changing the height of the footstep with respect to the previous one, making it possible to plan efficiently in a world of stairs. In their work, they leverage a 2.5D grid map of equally-sized cells, also called an elevation map [7]. This is practical for their scenario, but it needs to be modified if we want to consider multiple floors that could overlap in the same grids of the maps. The Multi-Level surface map proposed in [8] can overcome the previous limitations and allows to store multiple surfaces in each cell of the grid.

The complexity of humanoids makes it difficult to devise control strategies based on accurate dynamic models. On flat ground, many researchers adopt a simplified model known as the Linear Inverted Pendulum (LIP) [1], in which the robot is considered a single point-mass pivoting on a foot and moving at a constant height above the ground. The linearity of this model has been exploited to design preview controllers [9] first, and — to include constraints in the formulation — Model Predictive Control (MPC) schemes [10] later. When the robot moves on uneven ground one must remove the constant height hypothesis, leading to a nonlinear inverted pendulum model. The nonlinear problem was addressed in [11] using a dual MPC scheme and, more recently, in [12] by extending capturability concepts to the 3D case. However, keeping the linearity of the system is still an attrac-

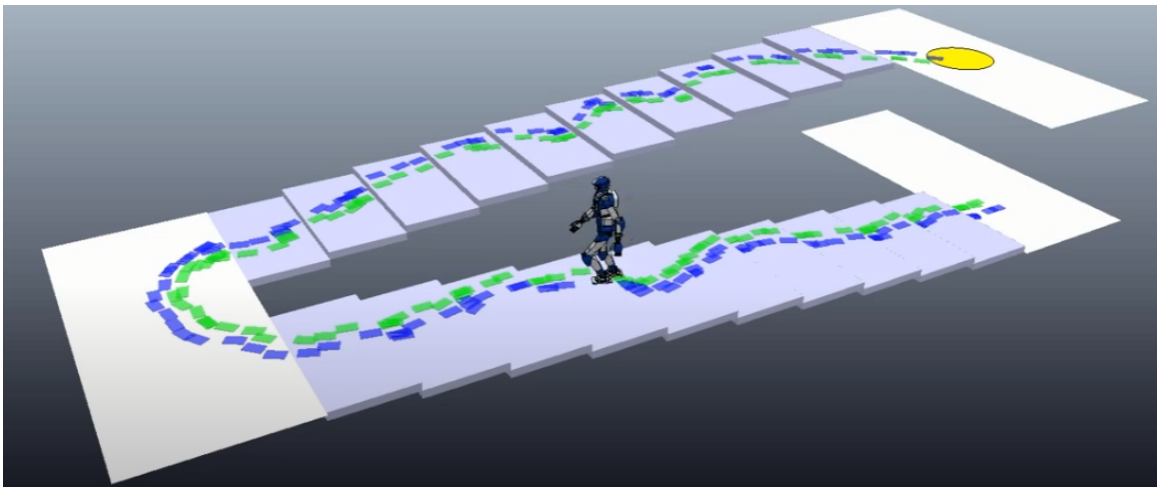

Figure 1: An instance of the task in an environment with two superimposed floors. To reach the goal region $\mathcal{G}$ (in yellow), the agent has to climb a flight of stairs, initially moving away (spatially) from the objective.

tive option in view of the simplicity of the associated controllers and, ultimately, the possibility of an efficient real-time implementation. One way to do this consists in assuming a predefined vertical trajectory of the Center of Mass (CoM), so that the robot can be described by a time-varying LIP, as done in [13, 14] and in [15] for transitions from bipedal to quadrupedal locomotion. Another, less restrictive possibility [16] is to maintain the time-invariant LIP structure by constraining the CoM vertical motion to satisfy a certain differential equation. Related approaches [17, 18] lead to a 3D model with LIP dynamics on all three axes. In [19] they have introduced an MPC scheme for gait generation incorporating a novel constraint that guarantees stable CoM trajectories regardless of the length of the prediction horizon. This scheme was then extended in [20] to the case of a 3D LIP model where the CoM height can vary under the appropriate differential constraint, thereby obtaining a scheme for walking along a given footstep sequence in a world of stairs. In this report, we integrate that gait generation scheme in a complete framework which includes a planner for generating a footstep sequence in this kind of scenario.

Our work is strongly based on [6] and follows for the implementation part most of what they have developed, modifying the necessary to act in a multi-floor environment.

3 Formulation and Approach

In Fig. 1 is represented an instance of the task that the agent has to face: 2 different large superimposed patches, connected by a flight of stairs and a landing. In one of the two patches, there is the goal region (the yellow circle). A humanoid robot should perform a walk from its starting point to the goal region. The environment is populated by elements placed at different heights and of different shapes. The knowledge of the environment is such that the agent doesn't know if an element is a walkable part of the map or if it's only an obstacle. In fact, this distinction is not well defined in general. For the robot, each place that is accessible with a step would be explored by the planning algorithm, even though it could be only an impasse.

The main goal of this task it reach a goal region which is placed at a height such that it's not

possible to access it directly from the patch of the starting point. In this way, the robot is forced to pass through areas that can even temporarily drive it away from the goal region but that allows it to modify its height on the map to go up or down. This is summarized in the task of going upstairs.

As previously mentioned, to represent this scenario it's convenient to use a *Multi-Level surface map* $\mathcal{M}_z$ which, given a cartesian point (x, y), returns a list of objects, along with their thickness, that is developed along the z-axis of that point.

To solve the task we use the same scheme proposed in [6] represented in Fig. 2. An off-line footstep planner computes a sequence of feasible footsteps $\{\boldsymbol{f}^j\}$ that leads to the goal region $\mathcal{G}$ from the starting point. Along with the sequence, it provides the foot trajectories $\{\boldsymbol{p}^j_{\text{swg}}\}$ between consecutive steps. Here $\boldsymbol{f}^j = (x^j_f, y^j_f, z^j_f, \theta^j_f)^T$ is the pose of the j-th footstep, with x^j_f, y^j_f, and z^j_f representing its position and θ^j_f its orientation. For the latter, only the yaw angle is provided since the grounds are taken all horizontal, so the other two angles are always zero, also during the execution of foot trajectories. In order to be part of the sequence, the elements of $\{\mathbf{f^j}\}$ must satisfy 3 feasibility constraints. A footstep $\boldsymbol{f}^j = (x^j_{\text{f}}, y^j_{\text{f}}, z^j_{\text{f}}, \theta^j_{\text{f}}) \in \mathcal{S}_f$ is *feasible* if it satisfies the following requirements:

R1 $\boldsymbol{f}^j$ is fully in contact within the same patch, i.e., the footprint is in every point at the same height and there are no parts not in contact with a surface. This avoids situations that can generate uncontrollable instability for the humanoid. In practice, in the code we verified that this condition held just for the perimeter of the footstep, assuming that this is enough to ensure compliance with R1. This has been done in order to save computational power.

R2 $\boldsymbol{f}^j$ is kinematically admissible form the previous foot-step $\boldsymbol{f}^{j-1}$. This is actually *stance feasibility*. The admissible region (a submanifold of $I\!R^3 \times SO(2)$) for placing $\boldsymbol{f}^j$ next to $\boldsymbol{f}^{j-1}$ is defined by the following equations:

$$-\begin{pmatrix}\Delta^-_x\\ \Delta^-_x\end{pmatrix} \leq R^T_{j-1}\begin{pmatrix}x^j_f - x^{j-1}_f\\ y^j_f - y^{j-1}_f\end{pmatrix} \pm \begin{pmatrix}0\\ 1\end{pmatrix} \leq \begin{pmatrix}\Delta^+_x\\ \Delta^+_x\end{pmatrix} \tag{1}$$

$$\Delta^-_z \leq z^j_f - z^{j-1}_f \leq \Delta^+_z \tag{2}$$

$$\Delta^-_\theta \leq \theta^j_f - \theta^{j-1}_f \leq \Delta^+_\theta \tag{3}$$

where R_{j-1} is the planar rotation matrix associated with θ^{j-1}_f and the Δ symbols are lower and upper maximum increments. A graphic visualization of this region is in Fig. 4.

R3 $\boldsymbol{f}^j$ is reachable from $\boldsymbol{f}^{j-2}$ through a collision-free motion (this is actually step feasibility).

The planner works off-line to find a *footstep plan* $\mathcal{P} = \{\mathcal{S}_f, \mathcal{S}_\varphi\}$ within the maximum number of iterations allowed. We denote by

$$\mathcal{S}_f = \{\boldsymbol{f}^1, ..., \boldsymbol{f}^n\}$$

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

$$\mathcal{S}_\varphi = \{\varphi^1, ..., \varphi^{n-2}\}$$

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

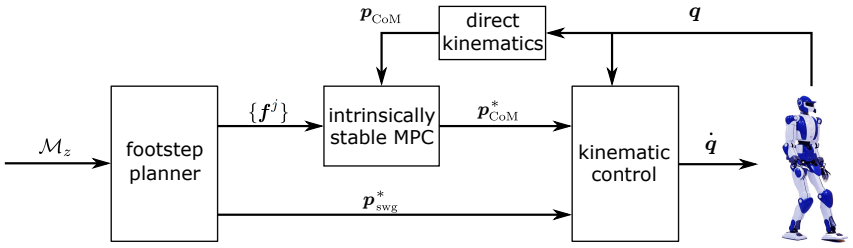

Figure 2: Block scheme of the proposed approach.

After the footstep sequence has been generated, it is passed to the online gait generation block that is based on intrinsically stable MPC, which computes a CoM trajectory $\boldsymbol{p}^*_{\text{CoM}}$ compatible with the sequence The swing foot trajectory $\boldsymbol{p}^*_{\text{swg}}$ at any instant is defined by the appropriate subtrajectory $\boldsymbol{p}^j_{\text{swg}}$. Finally, the reference trajectories $\boldsymbol{p}^*_{\text{CoM}}$ and $\boldsymbol{p}^*_{\text{swg}}$ are used in a standard pseudoinverse-based kinematic controller to compute joint commands $\dot{\mathbf{q}}$ for the robot. Proprioceptive feedback is used for both MPC and kinematic control.

In the next sections, we focus on the details of the Multi-Level map and the footstep planner.

4 Multi-Level surface map

For the purpose of accomplishing this task, a key role is played in the type of representation of the map. Elevation maps are frequently used for many cases. They are built by storing in each cell of a discrete grid the height of the surface in the corresponding area. This type of representation has an important limitation since, although compact and practical, it cannot represent multiple levels or vertical structures.

A *Multi-Level Surface* (MLS) map proposed in [8] allows us to overcome this limitation and store multiple surfaces in each cell of the grid. The Authors of the paper stated that their approach is well-suited for representing large-scale outdoor environments. In fact, they presented a procedure to build a MLS starting from point clouds acquired by sensors on the robot. For our implementation, we proceeded without using data from sensors. Instead, we developed a plug-in for CoppeliaSim that translates a scene in a data format containing position, orientation and size of each object in the environment. When a scene is finished, it looks like the examples in Fig. 7.

A multi-level surface map (MLS map) consists of a 2D grid of variable size where each cell $c_{ij}, i, j \in \mathbb{Z}$ in the grid stores a list of surface patches $P^1_{ij}, \ldots, P^K_{ij}$. A surface patch in this context is represented as a tuple (h_j, t) where h_j is the height of the top of the patch and t is the thickness, considered from h_j down. Figure 3 depicts a visualization of the MLS map for each of the environments we built. Each dot represents a patch while the segment attached to it, if present, reflects its thickness. The information in the tuple is enough to evaluate if the robot can travel through that cell, given the actual height of its position and the robot's dimensions. The cells of the map without any patch P_{ij} are empty of tuples but still useful for the RRT algorithm random point generation, as explained in the next section.

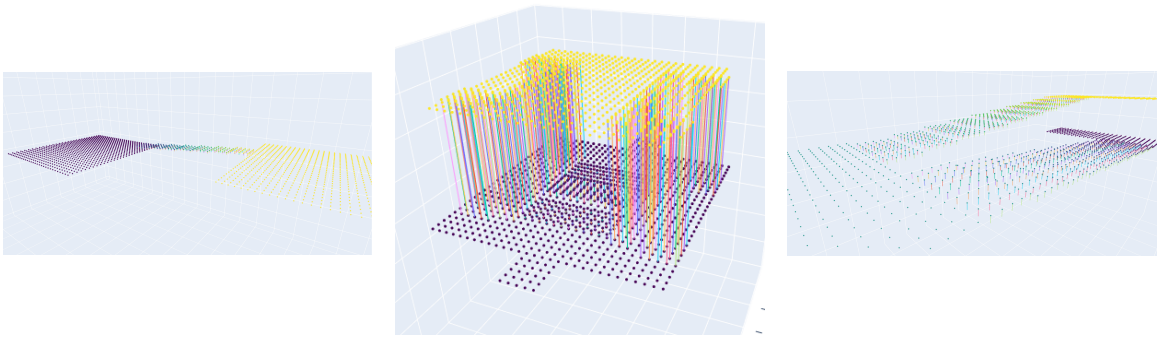

Figure 3: Visualization of the MLSM for (from left to right) *Stairway* , *Tunnel* , *Floors*

5 Footstep Planning

Once have defined the structure of the map, in this section we will explain how the footstep planner works and how it was developed. We recall that its purpose is to provide a sequence of footsteps, which are feasible under some specific constraints and that brings the humanoid from its starting position to the Goal $\mathcal{G}$. Our implementation is based on an RRT-like strategy, whose pseudo-code is given in algorithm 1. RRT allows to efficiently search spaces by randomly building a space-filling tree.

The algorithm takes as input an initial stance $\mathbf{f}$, the goal region $\mathcal{G}$, a Multi-Level map $\mathcal{M}_z$ and the maximum number of iterations allowed. The initial stance is taken as root node of the tree. In each node, it is stored the positions of both $\boldsymbol{f}_{\text{sup}}$ and $\boldsymbol{f}_{\text{swg}}$, the parent of the node, a list of children, the cost to reach the node from the root, the *id* of the swing foot and the trajectory that from the parent leads to the node's pose. Lastly, it is also stored the catalogue of remaining motion primitives (more about this later in this section). Two nodes are connected if has been found a feasible trajectory between their stances.

First of all, it is verified if both the initial stance and the goal region passed in the input are *feasible*. 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 last requirement can only be tested conservatively. In particular, we say that R3 is satisfied if:

i) There exist a collision-free swing foot trajectory $\boldsymbol{\varphi}^{j-2}$ from $\boldsymbol{f}^{j-2}$ to $\boldsymbol{f}^{j}$. The tested foot trajectory are parabolas with downward concavity and they belong to the plane that contains the sagittal axis of the pose, i.e., the mean orientation of the two footsteps. It tests 5 different parabolas with increasing height and it stops as soon as it found the first collision-free one. If θ^j is different from θ^{j-2}, the foot is considered linearly rotating during the execution of the trajectory in order to reach the new orientation, and so also the actual orientation of the foot is taken in consideration for the collision checking during the path of the parabola.

ii) A suitable volume B accounting for the maximum occupancy of the humanoid upper body at stance $(\boldsymbol{f}^{j-1}, \boldsymbol{f}^{j})$ is collision-free. B is a vertical parallelepiped with a square base around the body of the robot, starting from a distance z_b from the ground so as to not include ankles and feet.

If the initial stance or the goal region are not compliant with R1, the algorithm stops here. Otherwise, for each iteration, a random point $\boldsymbol{p}_{rand}$ is generated from a uniform distribution over

the 3 dimensions of the map. In this part, we use an *exploration/exploitation strategy*: with a 5% probability the $\boldsymbol{p}_{rand}$ is generated in the middle of $\mathcal{G}$. This allows forcing the procedure to find a path towards the goal region without losing the advantages of random methods.

The next step is to find the nearest node $\boldsymbol{v}_{near}$. The distance between $\boldsymbol{p}_{rand}$ and each node of $\mathcal{T}$ is computed with the following metric:

$$\mu(v, \boldsymbol{p}_{rand}) = ||\boldsymbol{m}(v) - \boldsymbol{p}_{rand}|| + k_\mu |\psi(v, \boldsymbol{p}_{rand})| \tag{4}$$

where $\boldsymbol{m}(v)$ represents the 3D position of the midpoint between the feet at stance v of the node, k_μ is a positive scalar and $\psi(v, \boldsymbol{p}_{rand})$ is the angle between the robot sagittal axis (whose orientation is the average of the orientations of the two footsteps) and the line joining $\boldsymbol{m}$ to $\boldsymbol{p}_{rand}$. In the end, the vertex with the lower value is selected as v_{near}.

Then it is tried to expand by drawing randomly from the catalogue U of primitives of the node. The primitives, which are depicted in Fig. 5, are chosen to automatically satisfy conditions (1-3) of requirements R2. Selecting a primitive is equivalent to creating a new candidate pose for which $\boldsymbol{f}_{\text{swg}}^{cand} = \boldsymbol{f}_{\text{sup}}^{near}$ (the new swing foot is the previous support foot, the one of $\boldsymbol{v}_{near}$) and $\boldsymbol{f}_{\text{sup}}^{cand}$ is the chosen primitive. When a primitive is picked, it is removed from the node's catalogue. By doing so, repetitions are avoided and once a node has run out of primitives, it will never be chosen as $\boldsymbol{v}_{near}$.

At this point it is necessary to assign as height z_f^{cand} of $\boldsymbol{f}_{\text{sup}}^{cand}$ the surface at a closer distance w.r.t. $\boldsymbol{f}_{\text{swg}}^{cand}$ along z-axis. If no surface is found within Δ_z, the candidate node is discarded and it moves on to the next iteration.

When the candidate's pose is complete, it is verified that it is compliant with requirements R1, R3 and condition 2 of R2. Along with them, if any of these checks fail, the candidate is discarded and it moves on to the next iteration. If not, a new node is added to the tree. It will have the candidate pose as the stance of the footsteps, $\boldsymbol{v}_{near}$ as a parent, an empty list of children, a full catalogue of primitives, the *id* of the swing foot opposite to the one of $\boldsymbol{v}_{near}$ and the first collision-free motion found by R3 as the trajectory.

The last step of the iteration is to verify if the new pose added to the tree has reached the goal region. If any point of $\boldsymbol{f}_{sup}^{j}$ (the new footstep generated) is included in G, the algorithm stops and returns a footstep plan $\mathcal{P} = \{\mathcal{S}_f, \mathcal{S}_\varphi\}$ where $\mathcal{S}_f$ is the branch of nodes that leads from the root to the newly generated node. Otherwise, a new iteration starts with a new $\boldsymbol{p}_{rand}$.

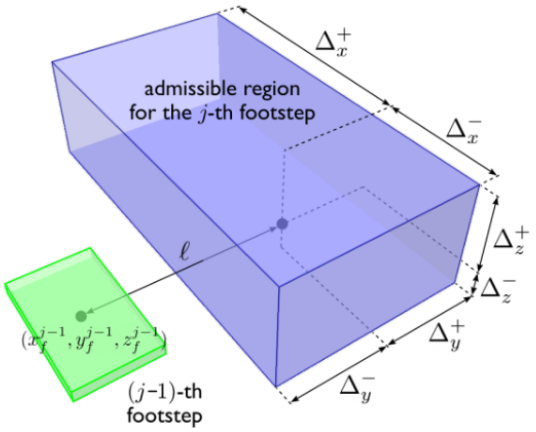

Figure 4: Admissible 3D region for a new footstep, shaped by requirement R2 constraints 1 and 2.

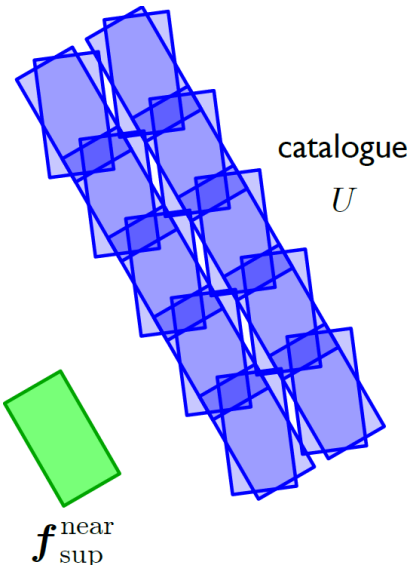

Figure 5: The catalogue U of primitives specifies the possible planar poses (in blue) of the candidate footstep $\boldsymbol{f}^{\text{cand}}$ with respect to the pose $\boldsymbol{f}_{\text{sup}}^{\text{near}}$ of the current support foot (in green). Here we have considered the case of a swinging right foot; the catalogue for a swinging left foot is specular. Once a primitive is chosen, the z coordinate of the candidate footstep will be retrieved from the elevation map $\mathcal{M}_z$.

Algorithm 1: Footstep Planner

root the tree $\mathcal{T}$ at $v_{\text{ini}} \leftarrow (\boldsymbol{f}_L, \boldsymbol{f}_R)$;
$i \leftarrow 0$;
repeat
  $i \leftarrow i + 1$;
  generate a random point $\boldsymbol{p}_{\text{rand}}$ on the ground;
  select the closest vertex v_{near} in $\mathcal{T}$ to $\boldsymbol{p}_{\text{rand}}$ according to $\gamma(\cdot, \boldsymbol{p}_{\text{rand}})$;
  randomly select from the primitive catalogue U a candidate footstep $\boldsymbol{f}^{\text{cand}}$;
  if $\boldsymbol{f}^{\text{cand}}$ is feasible w.r.t. R1–R2 **then**
    $h \leftarrow h_{\min}$;
    $\boldsymbol{p}_{\text{swg}}^{\text{cand}} \leftarrow \text{BuildTrajectory}(\boldsymbol{f}_{\text{swg}}^{\text{near}}, \boldsymbol{f}^{\text{cand}}, h)$;
    while $h \leq h_{\max}$ **and** $\text{Collision}(\boldsymbol{p}_{\text{swg}}^{\text{cand}})$ **do**
      $h \leftarrow h + \Delta h$;
      $\boldsymbol{p}_{\text{swg}}^{\text{cand}} \leftarrow \text{BuildTrajectory}(\boldsymbol{f}_{\text{swg}}^{\text{near}}, \boldsymbol{f}^{\text{cand}}, h)$;
    end
    if $h \leq h_{\max}$ **then**
      $v_{\text{new}} \leftarrow (\boldsymbol{f}^{\text{cand}}, \boldsymbol{f}_{\text{sup}}^{\text{near}})$;
      add vertex v_{new} to $\mathcal{T}$ as a child of v_{near};
      compute midpoint $\boldsymbol{m}$ between the feet at v_{new};
    end
  end
until $\boldsymbol{m} \in \mathcal{G}$ **or** $i = i_{max}$;

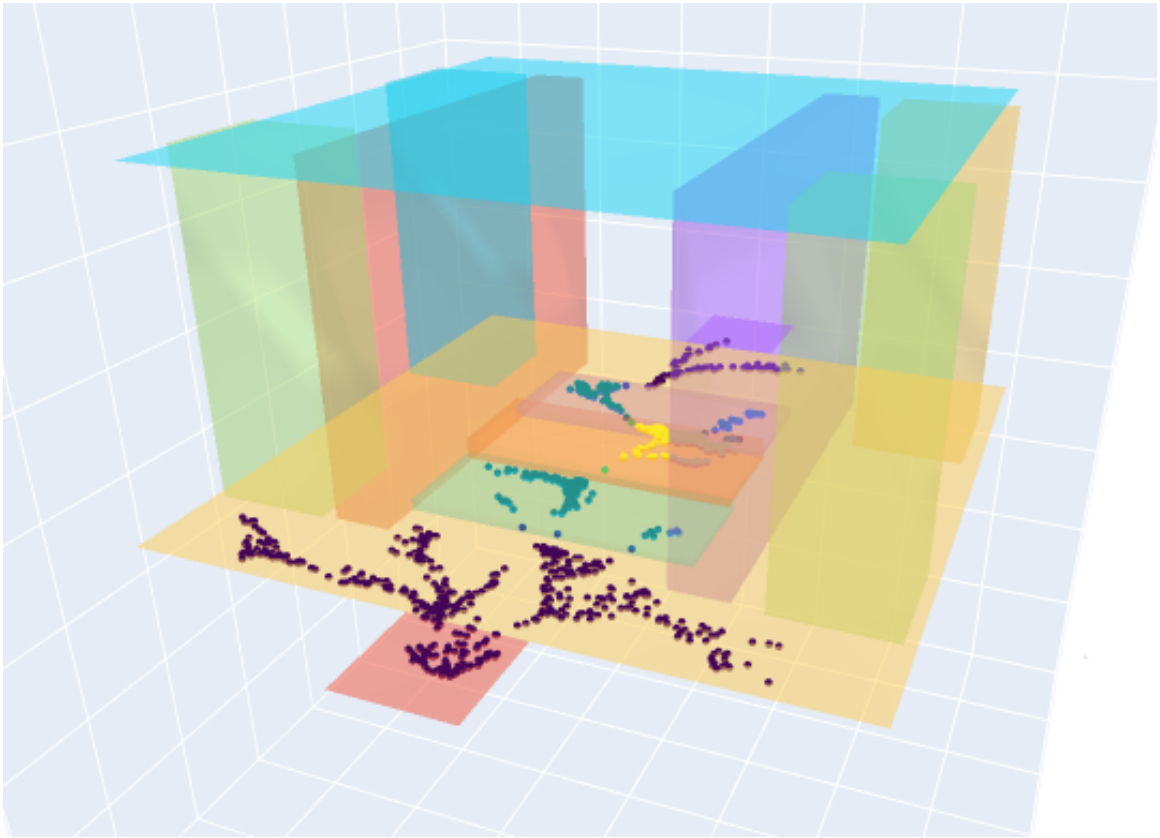

Figure 6: A tree of stances built exploring *Tunnel* map

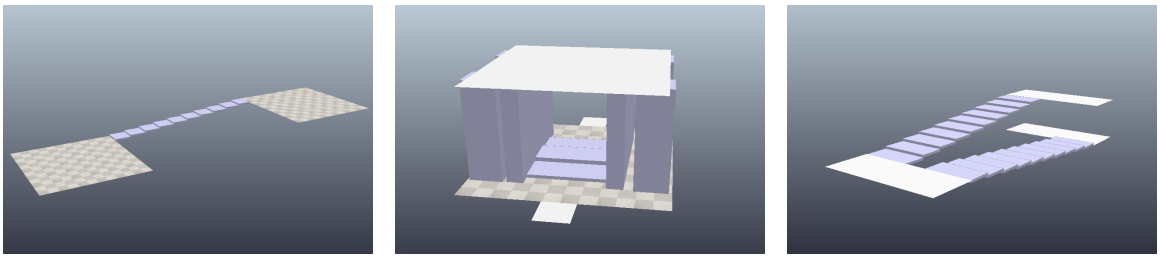

Figure 7: Scenarios developed on CoppeliaSim for test the framework. From left to right: *Stairway*, *Tunnel*, *Floors*.

5.1 Application Scenarios

After having implemented the algorithm in Python, we run it on 3 different scenarios, which are increasingly challenging and are thought to verify 2 interesting objectives that a Multi-Level surface map allows to accomplish: walking under a structure and going upstairs. The scenarios, represented in 7 are the following:

- *Stairway*: A simple structure with a straight sequence of steps that connect two small surfaces at different heights. The robot starts at the bottom one and the goal region is on the top patch. This is the easiest environment to explore and was used as ground truth.
- *Tunnel*: A short tunnel with 3 steps that separate the initial stance and the goal region. This environment has 2 characteristics that make it more difficult with respect to Stairway: the first is the presence of walls that can generate body collisions, while the second is the fact that the tunnel has a ceiling and the robot has to walk within it and the floor.
- *Floors*: Two superimposed floors of the same shape, connected with a flight of stairs with a large landing in the middle. In this scenario, there are no obstacles that can generate body collisions. The robot starts downstairs and has to go upstairs, where the goal region is located. Here the planner has to be able to explore and expand the tree in an environment that constrains the robot to first step away and then come back in the same area of the map while going up in altitude.

In all scenarios, the robot has to reach a circular goal region of radius 0.5 m using the catalogue of primitives $U = \{-0.06, 0.00, 0.06, 0.10, 0.14, 0.18, 0.23\} \times \{-0.05, 0.00, 0.05\} \times \{-0.20, 0.00, 0.20\}$. In the off-line footstep planner we have set $k_\mu = 1, k_\gamma = 0.4$, $h_{\min} = 0.02$ m, $h_{\max} = 0.24$ m, $\Delta h = 0.02$ m, $\Delta_x^- = 0.08$ m, $\Delta_x^+ = 0.24$ m, $\Delta_y^- = 0.07$ m, $\Delta_y^+ = 0.07$ m, $\Delta_z^- = 0.16$ m, $\Delta_z^+ = 0.16$ m, $\Delta_\theta^- = 0.3$rad, $\Delta_\theta^+ = 0.3$rad, $\ell = 0.25$ m, $\vartheta = 0.2$rad, $z_b = 0.3$ m, $h_b = 1.2$ m and $r_b = 0.25$ m. The Multi-Level surface map $\mathcal{M}_z$ has a resolution of 0.02 m. The used robot is HRP-4, a 1.5 m tall humanoid with 34 degrees of freedom by Kawada Robotics. In Fig. 8 are shown examples of paths found for each map.

Map	tree size (avg)	goal steps (avg)	time (sec)	total iteration (avg)
Stairway	3844	145.3	243.7	8561.6
Tunnel	1469.3	71.1	45.11	3603.2
Floors	34216.3	269.2	11768.7	59304.1

Table 1: Mean results on 10 trials for each scenario

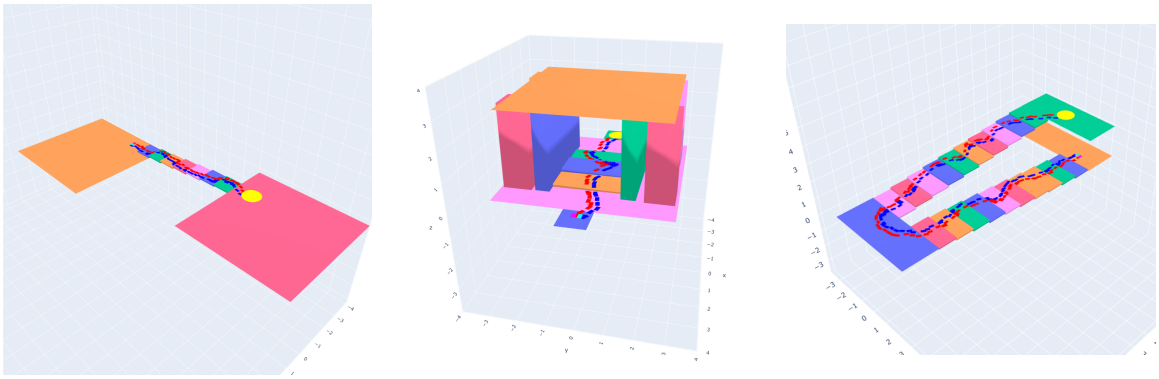

Figure 8: Examples of paths found for (from left to right) *Stairway* , *Tunnel* , *Floors*

6 Gait Generation via MPC

Once a footstep plan has been computed, it is passed to an on-line gait generation module which must provide a compatible trajectory for the humanoid CoM. This module uses a technique presented in [20] for generating stable gaits motions on uneven ground. It is based on an MPC scheme which enforces an explicit stability constraint to ensure that the resulting CoM trajectory is bounded w.r.t. the Zero Moment Point (ZMP). In this section, we briefly recall the main points of this technique.

The main feature of the proposed control scheme is to allow those vertical motions of the CoM which still lead to a Linear Time-Invariant (LTI) system. In particular, denoting with $\boldsymbol{p}_{\mathrm{c}} = (x_{\mathrm{c}}, y_{\mathrm{c}}, z_{\mathrm{c}})^T$ and $\boldsymbol{p}_{\mathrm{z}} = (x_{\mathrm{z}}, y_{\mathrm{z}}, z_{\mathrm{z}})^T$ the position respectively of the CoM and the ZMP, it has been shown that if the CoM vertical motion satisfies the differential equation

$$\ddot{z}_{\mathrm{c}} = \omega^2(z_{\mathrm{c}} - z_{\mathrm{z}}) - g \tag{5}$$

for some user-defined constant ω, then the x and y CoM motions still verify the typical LIP equations

$$\ddot{x}_{\mathrm{c}} = \omega^2(x_{\mathrm{c}} - x_{\mathrm{z}}) \tag{6}$$

$$\ddot{y}_{\mathrm{c}} = \omega^2(y_{\mathrm{c}} - y_{\mathrm{z}}). \tag{7}$$

Note that ω is a design parameter. Its value determines the equilibrium point of the z_{c} dynamics which is obtained when the difference between the z coordinates of the ZMP and CoM is equal to g/ω^2. The considered model neglects angular momentum around the robot CoM.

6.1 Motion model

In order to achieve a smoother ZMP trajectory, we use as motion model for the MPC algorithm, a third-order extension of the LIP model, i.e. we use the derivative of the ZMP $\dot{x}_{\mathrm{z}}$, $\dot{y}_{\mathrm{z}}$ and $\dot{z}_{\mathrm{z}}$ as

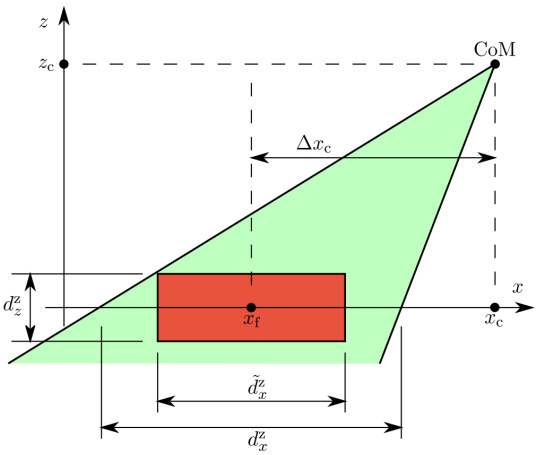

Figure 9: Side view of the polyhedral cone (green) representing the actual ZMP constraint and the box (red) used for defining approximate linear constraint.

control inputs. The model along the x-axis is

$$\begin{pmatrix} \dot{x}_c \\ \ddot{x}_c \\ \dot{x}_z \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ \omega^2 & 0 & -\omega^2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_c \\ \dot{x}_c \\ x_z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \dot{x}_z \tag{8}$$

The y component is similar while the z component only has the additional additive term in (5).

We use piecewise-constant control inputs over sampling intervals of duration δ, with a prediction horizon $T_h = N \cdot \delta$. The current time instant is t_k and the successive instants within the prediction horizon are t_{k+i}, $i = 1, \ldots, N$. A similar notation is used for all variables; e.g., the current CoM position is x_c^k and its predicted value at t_{k+i} is x_c^{k+i}.

6.2 Constraints

On uneven ground, balance is ensured when the ZMP is inside the conic polyhedron having vertex at the CoM and edges passing through the vertices of the support polygon [11]. However, the resulting balance condition would be nonlinear because the polyhedral cone has its vertex at the CoM which changes location. Using the approximation in Fig. 9, the following linear *balance constraint* is obtained

$$-\frac{1}{2}\begin{pmatrix} \tilde{d}_x^z \\ \tilde{d}_y^z \\ d_z^z \end{pmatrix} \leq R_{k+i}^T \begin{pmatrix} x_z^{k+i} - x_f^{k+i} \\ y_z^{k+i} - y_f^{k+i} \\ z_z^{k+i} - z_f^{k+i} \end{pmatrix} \leq \frac{1}{2}\begin{pmatrix} \tilde{d}_x^z \\ \tilde{d}_y^z \\ d_z^z \end{pmatrix}. \tag{9}$$

Here $\tilde{d}_x^z$ and $\tilde{d}_y^z$ are the reduced horizontal sizes of the constraint, chosen to be compatible with the vertical size d_z^z which is a design parameter. R_{k+i}^T is the rotation matrix associated with θ_f^{k+i}. Since x_z^{k+i} can be linearly expressed in terms of the decision variables $\dot{x}_z^{k+j}$ for $j = 0, \ldots, i-1$, the constraint is linear in the decision variables. Similarly for the other coordinates y_z^{k+i} and z_z^{k+i}.

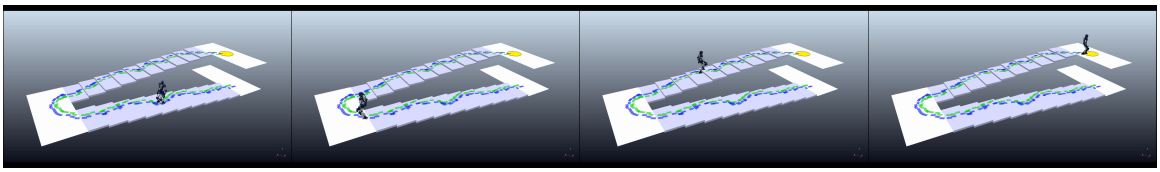

Figure 10: Frames of the execution of the framework in *Floors* environment, temporally ordered from left to right

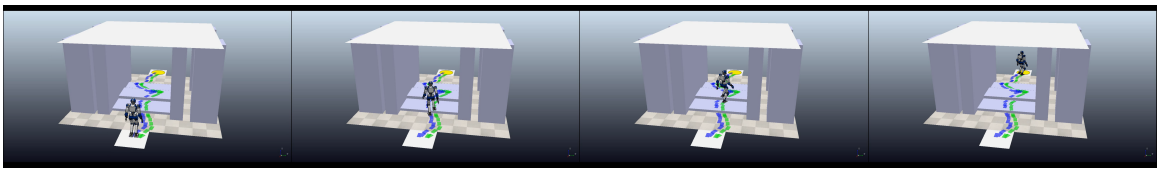

Figure 11: Frames of the execution of the framework in *Tunnel* environment, temporally ordered from left to right

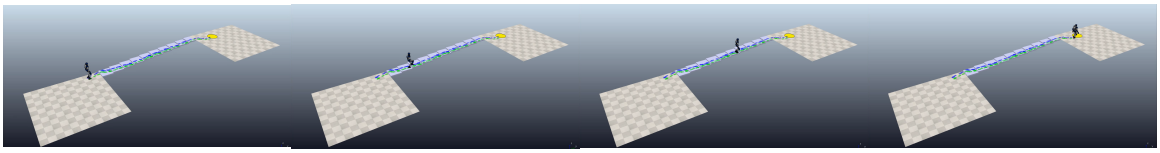

Figure 12: Frames of the execution of the framework in *Stairway* environment, temporally ordered from left to right

To ensure boundedness of the resulting CoM trajectory w.r.t. the ZMP, we enforce the *stability constraint* [19],

$$\boldsymbol{p}_{\mathrm{c}}^{k} + \frac{\dot{\boldsymbol{p}}_{\mathrm{c}}^{k}}{\omega} + \frac{\boldsymbol{g}}{\omega^2} = \omega \int_{t_k}^{\infty} e^{-\omega(\tau - t_k)} \boldsymbol{p}_{\mathrm{z}}(\tau) d\tau, \tag{10}$$

where $\boldsymbol{g} = (0, 0, -g)^T$. For the specifically chosen structure of the decision variables, (10) reduces to

$$\frac{1}{\omega} \frac{1 - e^{-\delta\omega}}{1 - e^{-N\delta\omega}} \sum_{i=0}^{N-1} e^{-i\delta\omega} \dot{x}_{\mathrm{z}}^{k+i} = x_{\mathrm{c}}^{k} + \frac{\dot{x}_{\mathrm{c}}^{k}}{\omega} - x_{\mathrm{z}}^{k}. \tag{11}$$

along the x-component (similarly for the y-component). Note that the integral in (10) extends to infinity while we only predict for the horizon T_{h}. This is handled by *infinitely replicating* the decision variables outside the prediction horizon. For the z-component we assume $\dot{x}_{\mathrm{z}}^{j} = 0$ after the prediction horizon; the stability constraint then becomes

$$\frac{1 - e^{-\delta\omega}}{\omega} \sum_{i=0}^{N-1} e^{-i\delta\omega} \dot{z}_{\mathrm{z}}^{k+i} = z_{\mathrm{c}}^{k} + \frac{\dot{z}_{\mathrm{c}}^{k}}{\omega} - z_{\mathrm{z}}^{k} - \frac{g}{\omega^2}. \tag{12}$$

6.3 QP and algorithm

At a generic time t_{k}, IS-MPC solves the following QP problem

$$\min_{\dot{\boldsymbol{p}}_z^k,\ldots,\dot{\boldsymbol{p}}_z^{k+N-1}} \sum_{i=0}^{C-1} \left\| \dot{\boldsymbol{p}}_z^{k+i} \right\|^2 + \beta \left\| \boldsymbol{p}_z^{k+i} - \boldsymbol{p}_{\mathrm{mc}}^{k+i} \right\|^2$$

subject to:

1. stability constraint
2. ZMP position constraint

The cost function minimizes the decision variables (ZMP derivative) as a regularization term, and attempts to bring the ZMP close to the center of the moving box, which is typically beneficial as it produces a more robust walking pattern. β is a weight on the second term. In typical MPC fashion, the first sample of the ZMP velocity $\dot{p}_z^k$ is used to integrate the 3D LIP dynamics. The resulting CoM trajectory p_{c} is sent, together with the swing foot trajectory generated by the footstep planner and sampled at time t_{k}, to the kinematic controller.

The MPC algorithm provides the reference CoM trajectory, while a reference trajectory for the swing foot is a direct outcome of the planning phase. Both these trajectories are then tracked on the humanoid robot by kinematic control.

7 Simulations and Results

In the end, we performed simulations on the HRP-4 humanoid robot on CoppeliaSim for each environment. We assessed the validity of the whole framework, which demonstrated to be able to complete the task correctly. The trees generated by the algorithm result in correctly and deeply exploring the scenarios, as can be seen in Fig. 6 where is reported a tree built in *Tunnel*. Despite this, the paths that are chosen to reach the goal region seem to be sub-optimal. This can be stated by looking at Fig. 8: here are shown 3 paths, found for each of the maps. It is particularly evident in *Tunnel* (scene 2 in the image) how, while the path starts to descend the steps, it performs a large variation in orientation that involves as a consequence an increase in the number of steps needed to complete the task. This can also be observed in *Floors* and *Stairway* (scenes 3 and 1 in the image, respectively) where the paths turn out to be quite twisting. These examples prove something that we already expected since we implemented a simple RRT path planner, which doesn't take into consideration the cost of nodes or steps optimality when outputting a path. Since RRT is a randomized algorithm, the quality of the results can vary in every run.

In order to have a general idea of the performances of the path planner, we run tests for each scenario. The average results over 10 runs are presented in Tab. 1: *tree size* is the number of steps added to the tree, *goal steps* is the number of steps to reach the goal, *time* is the total time of a run and *total iterations* the number of iterations executed until a path was found. For each test, the algorithm is limited to 100.000 iterations but they were more than enough to find feasible paths. In fact, the highest average number of iterations was 59304.1 for *Floors*. It is possible to notice how the average time needed to find a path is strongly proportional to the dimension of the maps and less to the difficulty of the scenario since it took less time in *Tunnel* than in *Stairway*. Finding a path in *Floors* needs significantly more time (two orders of magnitude). Performances are limited by simplification that we adopted in order to streamline the procedure, but apart from the slowness

in performing the entire walk, with respect to [6] for example, the agent acts coherently with the task constraints and formulation.

Simulations allowed us to verify that the gait generation model acts correctly in each environment, approaching steps both in ascending and descending walks. In Fig. 12 are reported different frames of the robot moving in *Floors*. From left to right, it is possible to see the robot accomplish the first flight of stairs, then bend over the landing, change direction while going over the second flight and finally reach the goal region. In the following figures, Fig.11 and Fig.12, the same representation is reported for the other two scenarios, for which the paths are more straightforward. We invite the reader to watch the accompanying videos here, which includes clips related to described simulations.

By achieving the goal correctly in all scenarios, given the different challenges of each map, we can confirm that with our framework the agent is able to move in a multi-surface environment, leveraging the advantages of an MLS map. We do not show any explicit comparison with other existing footstep planners, since the aim of our project is to introduce a possible approach to execute humanoid-legged locomotion MLS scenarios. Our results do not aim to reach S.O.T.A levels either in time execution or in optimality and efficiency of the planner. An analysis of the video clips of the simulations reveals that the generated motions are suboptimal in terms of efficiency.

8 Conclusions

We presented a planning and control framework for humanoid locomotion in a Multi-Level environment. The MLS map has been shown to be able to correctly represent environments such as multi-floor areas or tunnels, allowing to do footstep planning in such scenarios. The offline RRT planning module has proven to be capable of generating suitable footstep sequences given enough amount of time. Then the MPC module utilizes them to generate a 3D gait in real time. The overall architecture has been tested by simulations in CoppeliaSim with the HRP4 humanoid, and in such a way we confirmed its effectiveness. Both in *Tunnel* and *Floors* there are useless parts in the sequence of steps we produced, in which essentially the robot seems to take some steps remaining in the same spot. This is because there is no optimization stage in our footstep planner.

A convenient possible improvement could be to extend the planning algorithm by introducing a *Rewiring* phase to modify the edges of the tree to connect more efficiently newly generated nodes. Moreover, the code could be implemented in C++ to speed up its execution. Another possible improvement for reaching computationally faster results would be to leverage more advanced tree structures, like *k-d tree*. This would allow to speed up significantly the search of the $\boldsymbol{v}_{near}$ at each RRT iteration. The procedure we proposed is also naturally extensible to an RRT* approach, that expands the standard RRT and returns the shortest possible path among the nodes of the built tree. This requires defining a cost function that expresses the cost of reaching a node. An RRT* implementation needs adding 2 more steps to the algorithm of the planner we described. These steps consist in verifying if, for a newly generated node, there are already possible cheaper parent nodes in the tree and if it can be used as a cheaper parent for other nodes. Although, in order to be efficient, is fundamental to find a meaningful cost function, which takes into consideration the aspect of the Multi-Levels environment. Eventually, a proper step ahead would also be to verify the robustness of this procedure by implementing it in a real-world scenario and conducting tests with a real robot.

References

[1] S. Kajita and K. Tani, "Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode," in *1991 IEEE Int. Conf. on Robotics and Automation*, 1991, pp. 1405–1411.

[2] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS Int. Conf. on Humanoid Robots*, 2014, pp. 279–286.

[3] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep planning for the Honda ASIMO humanoid," in *2005 IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 629–634.

[4] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *2012 IEEE-RAS Int. Conf. on Humanoid Robots*, 2012, pp. 674–679.

[5] Z. Xia, G. Chen, J. Xiong, Q. Zhao, and K. Chen, "A random sampling-based approach to goal-directed footstep planning for humanoid robots," in *2009 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics*, 2009, pp. 168–173.

[6] P. Ferrari, N. Scianca, L. Lanari, and G. Oriolo, "An Integrated Motion Planner/Controller for Humanoid Robots on Uneven Ground," in *2019 18th European Control Conference (ECC)*. Naples, Italy: IEEE, June 2019, pp. 1598–1603. [Online]. Available: https://ieeexplore.ieee.org/document/8796196/

[7] W. Burgard, M. Hebert, and M. Bennewitz, "World modeling," in *Springer handbook of robotics*. Springer, 2016, pp. 1135–1152.

[8] P. P. R. Triebel and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[9] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *2003 IEEE Int. Conf. on Robotics and Automation*, 2003, pp. 1620–1626.

[10] P.-B. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *6th IEEE-RAS Int. Conf. on Humanoid Robots*, 2006, pp. 137–142.

[11] S. Caron and A. Kheddar, "Dynamic walking over rough terrains by nonlinear predictive control of the floating-base inverted pendulum," in *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2017, pp. 5017–5024.

[12] S. Caron, A. Escande, L. Lanari, and B. Mallein, "Capturability-based pattern generation for walking with variable height," *arXiv:1801.07022*, 2018.

[13] A. Herdt, "Model predictive control of a humanoid robot," Ph.D. dissertation, Ecole Nationale Supérieure des Mines de Paris, 2012.

[14] M. A. Hopkins, D. W. Hong, and A. Leonessa, "Humanoid locomotion on uneven terrain using the time-varying divergent component of motion," in *14th IEEE-RAS Int. Conf. on Humanoid Robots*, 2014, pp. 266–272.

[15] T. Kamioka, T. Watabe, M. Kanazawa, H. Kaneko, and T. Yoshike, "Dynamic gait transition between bipedal and quadrupedal locomotion," in *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2015, pp. 2195–2201.

[16] K. Terada and Y. Kuniyoshi, "Online gait planning with dynamical 3d-symmetrization method," in *7th IEEE-RAS Int. Conf. on Humanoid Robots*, 2007, pp. 222–227.

[17] S. C. Luo, P. H. Chang, J. Sheng, S. C. Gu, and C. H. Chen, "Arbitrary biped robot foot gaiting based on variate CoM height," in *13th IEEE-RAS Int. Conf. on Humanoid Robots*, 2013, pp. 534–539.

[18] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-dimensional bipedal walking control based on divergent component of motion," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 355–368, 2015.

[19] N. Scianca, M. Cognetti, D. De Simone, L. Lanari, and G. Oriolo, "Intrinsically stable MPC for humanoid gait generation," in *16th IEEE-RAS Int. Conf. on Humanoid Robots*, 2016, pp. 601–606.

[20] A. Zamparelli, N. Scianca, L. Lanari, and G. Oriolo, "Humanoid gait generation on uneven ground using intrinsically stable MPC," *IFAC-PapersOnLine*, vol. 51, pp. 393–398, 2018.