SAPIENZA
UNIVERSITÀ DI ROMA

Master's Degree in Control Engineering

RL-augmented MPC for Humanoid Locomotion

Autonomous and Mobile Robotics

Professors:
Giuseppe Oriolo
Nicola Scianca

Students:
Coletta Emanuele
Sartori Giulio
Sperduto Gianluca

Academic Year 2025/2026

Contents

1 Introduction

Humanoid robots are more and more present in the robotics field. Unlike other mobile robots, humanoids are designed to interact with environments structured lived in and used by humans. These might include stairs or uneven terrain, where a human can easily move around on feet but are prohibitive for wheels. Contrary to other legged robots like quadrupeds, humanoid robots are also capable of performing manipulation tasks, even while moving. Humanoid robot design is typically very complex and such robots have a large number of Degree of Freedom (DoF), but are also underactuated. Careful controller design is thus needed to handle the complexity.

The majority of the controller nowadays are optimization based and use Model Predictive Controller (MPC) schemes based on the simplified linear dynamics of an Linear Inverted Pendulum (LIP). For dynamic balance of flat ground, the Zero Moment Point must remain inside the support polygon, that is the convex hull formed by the surfaces in contact with the ground in a given moment. The support polygon is given by the footsteps of the robot. This may be generated using a template model, for example a unicycle or an omnidirectional platform.

Other control approaches rely on machine learning methods, such as Reinforcement Learning. With these methods a policy is trained starting from data alone, in principle entirely ignoring any knowledge of the dynamics of the robot. The two approaches can also be used in conjunction: a baseline controller is obtained, possibly starting from assumptions under which the model can be simplified and made linear, and a data-driven method is then used to capture the remaining part of the dynamics, nonlinear in general, which is neglected by the simplified model.

In this work we consider the case of a humanoid robot, the HRP-4, which can move anywhere on flat ground and is controlled by means of an Intrinsically Stable Model Predictive Controller [1][2] that generates reference trajectories for the CoM, and a Whole Body Controller which computes the corresponding joint torques. Footsteps are generated offline using a unicycle as a template model and are encoded as constraints in IS-MPC. We add a residual policy to IS-MPC [3], trained using Reinforcement Learning techniques, and in particular the PPO algorithm [4]. Our policy modifies the footstep plan online in order to maintain balance in spite of external disturbance forces and terrain inclination, which are ignored in the LIP model used for prediction in IS-MPC.

This report is organized as follows: in this section we introduce our work and previous works it is based on. In Section 2 we present a general framework of how Reinforcement Learning is used in this project, and our implementation in particular. In Section 3 we describe and compare different approaches for training and action definition. In Section 4 simulations and comparison between the methods are carried out. Finally, conclusions are drawn in Section 6.

Figure 1: HRP4 Humanoid Robot presentation images.

1.1 HRP-4

The robot used in for this project is **HRP-4** humanoid robot. This robot is 151 [cm] high and weighs 39 [kg], with a total of 34 degree of freedom.

1.2 LIP Dynamics

In order to control a humanoid robot that has generally a lot of DoF and is characterized by a complex dynamics is often used the simplified Linear Inverted Pendulum (LIP) model, obtained expressing the dynamics of the CoM (*Center of Mass*) in terms of the position of the ZMP (*Zero Moment Point*), under the simplifying assumptions that the CoM height is constant at $z_c = h$ and that the Internal Angular Momentum derivative is zero $\dot{L}_c = 0$.

The overall model is a classical LTI system of the form $\dot{x} = Ax + Bu$, explicitly:

$$\frac{d}{dt}\begin{bmatrix} p_c \\ \dot{p}_c \\ p_z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \eta^2 & 0 & -\eta^2 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_c \\ \dot{p}_c \\ p_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{p}_z \tag{1}$$

with $\eta = \sqrt{\frac{g}{h}}$ and the ZMP derivative along the z axis $\dot{p}_z$ as new input according to the dynamics extension's fashion to avoid discontinuity in the ZMP trajectories.

The choice of this dynamical model assumes to deal with the motion (and consequently the angular momentum) of only one rigid body instead of taking into account the whole humanoid body.

Other kinds of approaches, like the one in the reference paper [3], focus on dynamics

that also consider quantities like the Angular Momentum about the contact point with the ground, for example the well known ALIP model. In this case the simplifying assumption is that the Angular Momentum about the CoM is zero.

Taking as state $\mathbf{x} = [x_c, y_c, L_x, L_y]^{\mathrm{T}}$, with L_x, L_y being the planar component of the Angular Momentum about the pivot, it is possible to obtain another LTI system of the form $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u_{fp}}$, where:

$$A = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{mz_H} \\ 0 & 0 & -\frac{1}{mz_H} & 0 \\ 0 & -mg & 0 & 0 \\ mg & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2}$$

with m denoting the mass of the robot, g the gravitational constant, and $z_H = h$. $u_{fp} = [u_x, u_y]^{\mathrm{T}}$ are respectively the x and y coordinates of the new foot placement positions w.r.t. the contact point.

The model used in the RL-augmented control system proposed is the LIP system, without then any informations about Pivot Angular Momentum evolution.

1.3 IS-MPC

The IS-MPC structure presented in [1], [2] is a variation of the classic MPC that add a condition to make the LIP dynamics stable, for this is called Intrinsically Stable MPC. The IS-MPC use use the LIP model described earlier for generate ZMP trajectories enforcing the stability of the unstable part , in particular the ZMP trajectory could be realizable using a divergent CoM position and making the gain unfeasible for the robot. Performing a change of coordinates to phosphatize the stable and unstable part of the dynamics:

$$\begin{aligned} x_s &= x_c - \dot{p}_c/\eta \\ x_u &= x_c + \dot{p}_c/\eta \end{aligned} \tag{3}$$

and looking at the corresponding dynamics:

$$\begin{aligned} \dot{x_s} &= -\eta(x_s - x_z) \\ \dot{x_u} &= \eta(x_u - x_z) \end{aligned} \tag{4}$$

Is possible to find a particular initial condition in witch the system remain stable. Enforcing this using a particular condition in the QP will guarantee that the trajectory of the CoM will be bounded w.r.t. the ZMP trajectory making the gait generation always feasible for the robot. An implementation is available at [5]

1.4 RL-approach

The Reinforcement learning approach is based on an Agent that must learn a Policy in order to interact with the environment maximizing rewards. In our case the Agent should learn how to displace the footsteps such that the HRP4 humanoid robot can reach the end of the plan without fall even when subject to external forces or whit sloping floor. The environment in our case is composed by the IS-MPC, HRP4 humanoid robot and the physic environment where is placed the robot. The algorithm used for learn the policy is Proximal Policy Optimization (PPO). When the robot interact with the environment get a reward based on how good the performance are, the desired performance are high level quantity such as the minimum displacement of the footsteps, good positioning of the feet, tracking of the reference etc.

1.4.1 Proximal Policy Optimization - PPO

The Proximal Policy Optimization [4] is a Trusty region on-policy reinforcement learning algorithm that optimize the policy without changing it too much each iteration based on the Advantage function computed from the reward and the estimate Value function of the policy. The algorithm use two neural network for estimate both the policy and the associated Value function performing on-policy roll-outs. Is a very stable and data efficient algorithm heavily employed not only in the robotics field.

1.4.2 Policy Structure

The policy learned is a probabilistic neural network with two hidden layer of 64 neuron each and *tanh()* activation function. For each action the neural network give the mean and the standard deviation of a Gaussian distribution from witch the action will be sampled. This approach is crucial for the exploration phase in PPO algorithm.

1.5 Disturbances

The disturbances that we wont to make the system robust are mainly the external forces and the slope of the floor. The IS-MPC controller based on the simplified LIP dynamics of the robot is not robust w.r.t. those perturbations and for this we wont to train a policy that can take care of all IS-MPC can't handle.

1.5.1 External Forces

The robustness to weak forces can be provided by the Inverse Dynamics of the robot but for stronger forces this is not enough and is necessary modify the footsteps. The agent will learn how to modify the footsteps in order to make the IS-MPC controller able to keep alive the robot with strong forces.

1.5.2 Ground Slope Variations

The slope of the floor is a critical aspect for keep balanced the robot. IS-MPC alone is not able to handle this perturbation and is necessary to modify the footsteps for find a feasible trajectory that keep the ZMP inside the support polygon. Is required that the Agent learn how to modify the footsteps creating a feasible plan for the IS-MPC. IS-MPC use equation where the gravity is assumed always acting completely on the z axis, for this the prediction on different condition will be wrong and lead the robot to fall.

1.6 Implementation

Our implementation is based on the existing IS-MPC python implementation [5] published by the original authors of [1], [2]. For the learning algorithm (PPO), we used the python library Stable-Baselines3 [6], which is compatible with Gymnasium [7] environments. For this project we wrapped the whole IS-MPC implementation, including the gait generator, MPC, inverse dynamics and Kalman filter, inside a Gymnasium environment for use in SB3, using the standard interface with *step()* and *reset()* functions. The agent receives from the environment the state of the robot and the reward gained in each step of the simulation. Much work in this project went in the definition and computation of the state of the agent, the reward function and how actions are applied to the environment. In the following we describe and compare the different approaches we tested, both in the training of the agent and in application of the actions.

1.6.1 Disturbances

The force disturbances are easily introduced applying at a random point in the vicinity of the centre of mass of the body, base or the swinging foot a random forces, all this randomicity should made the Agent learn how to react in general to forces and not to overfeet over specific situation.

For the steepness of the ground the implementation is made modifying the gravity vector of the simulation world, this modification is made randomly selecting the x and y angles around which the floor rotates, even in this case choosing randomly the angles should be enough for make the Agent generalize the information.

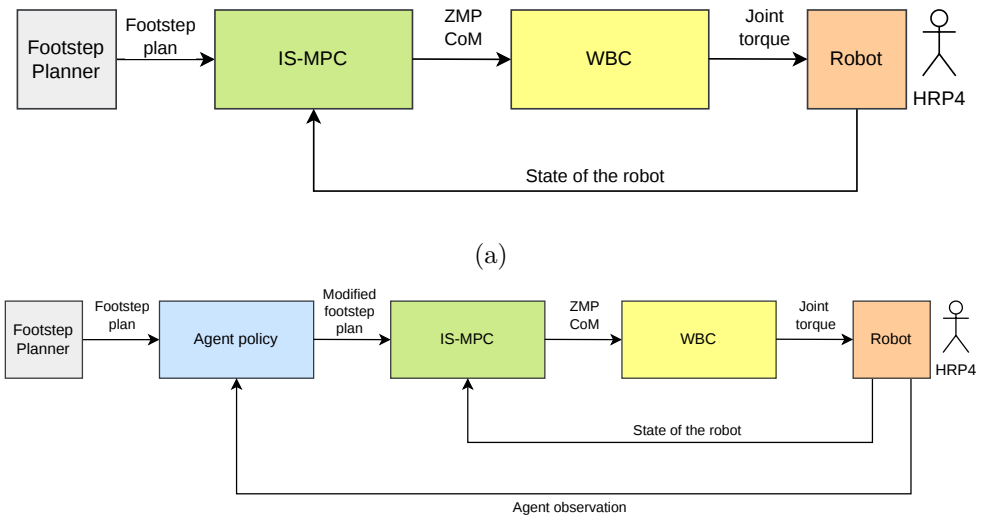


Figure 2: Comparison of the two control structures: (a) reference and (b) ours

2 General Framework

2.1 RL Augmented Control Scheme

A typical humanoid-motion controller structure is that based on a Gait Generator and Whole Body Controller, where the former produces footsteps based on a desired reference trajectory, typically for the CoM, while the latter keeps the robot in balance while having the feet track the desired gait, typically using MPC and inverse dynamics (ID). In this project we explore an extension of this structure that introduces an Agent trained via Reinforcement Learning (RL) that modifies the reference footsteps planned by the Gait Generator while the robot is moving. A comparison of the two structures is shown in Figure 2. The two feedback loops run at different frequencies: the MPC runs at higher frequencies to ensure stable predictions, while the RL Agent runs at slower frequency to ensure that changes to the plan are consistent and not jittery.

2.2 Environment: State and Episodes

Taking a *step* into the environment means that the agent *performs* an *action* and then it *observes* the resulting state and reward. After the action is applied, i.e. the footstep plan is displaced accordingly, the environment runs *MPC-frequency* cycles of IS-MPC evaluation and physics simulation of the robot. The *MPC-frequency* parameter is crucial to make the system more stable: low frequencies result in a fast modification of the plan by the Agent, but make the IS-MPC model unreliable (and often unable to find a solution that satisfies all the constraints); while with high frequencies the agent

takes actions too rarely to properly react to perturbations such as external forces.

An *episode* is a sequence of steps in the environment and is considered *terminated* in exactly three cases: the robot completes the footstep plan; the IS-MPC solver is unable to find a solution that satisfies all the constraints (which means the LIP dynamics has become unstable, i.e. the robot is falling); or the inverse dynamics solver is unable to find a solution.

The observable state given to the policy for choosing the action is a 27-dimensional vector composed as:

$$\mathcal{S} = \begin{bmatrix} \sigma_k & \Delta t_{r,k} & \mathbf{p_{c,k}} & \dot{\mathbf{p}}_{\mathbf{c,k}} & \mathbf{p_{z,k}} & \dot{\mathbf{p}}^{\mathbf{r}}_{\mathbf{c,k}} & \mathbf{p}^{\mathbf{r}}_{\mathbf{z,k}} & \mathbf{L} & \mathbf{\Delta}\sigma_{\mathbf{k+1}} & \mathbf{\Delta}\sigma^{\mathbf{r}}_{\mathbf{k+1}} \end{bmatrix}^T \in \mathbb{R}^{25} \times \mathbb{SO}^2(2) \tag{5}$$

Where $\sigma_{\mathbf{k}} \in \{\begin{bmatrix} 1 & 0 \end{bmatrix}^T, \begin{bmatrix} 0 & 1 \end{bmatrix}^T\}$ is the embedding of the support foot at time k, $\Delta t_{r,k} \in \mathbb{R}$ is the remaining time in the swing phase at time k, $\mathbf{p_{c,k}} \in \mathbb{R}^3$ is the position of the CoM relative to the support foot at time k, $\dot{\mathbf{p}}_{\mathbf{c,k}} \in \mathbb{R}^3$ is the CoM velocity at time k, $p_{z,k} \in \mathbb{R}^3$ is the position of the ZMP relative to the support foot at time k, $\dot{p}^r_{c,k} \in \mathbb{R}^3$ is the reference velocity at time k, $p^r_{z,k} \in \mathbb{R}^3$ is the reference ZMP position at time k, $L_k \in \mathbb{R}^3$ is the angular velocity of the robot relative to the support foot position at time k, $\Delta\sigma_{k+1} = \begin{bmatrix} \Delta x & \Delta y & \Delta\varphi \end{bmatrix}^T \in \mathbb{R}^2 \times \mathbb{SO}(2)$ is the relative position of the next footstep and $\Delta\sigma^r_{k+1} \in \mathbb{R}^3 \times \mathbb{SO}(2)$ is the next relative footstep position desired.

2.2.1 Reset

The environment is reset each time a new episode starts. When that happens, the Gymnasium environment calls the IS-MPC implementation to create a new instance of the *world* physics environment is created using **DartPy**, in which the HRP-4 humanoid is collocated, modelled as a chain of rigid bodies and including the footstep planner with a reference trajectory of choice, expressed in terms of desired CoM velocity. The IS-MPC is also initialized for the newly instantiated robot, as well as the Kalman filter for state observation.

2.2.2 Step

The *Step()* function connects the Agent and the environment. Each step, the action computed by the Agent policy is applied to the environment, then the new state and resulting reward are observed. In our case, the action of the Agent corresponds to a modification of the footstep plan. Different approaches to the modification have been tested for this work, and they are described in Sections 3.1 through 3.5. After the footstep plan is displaced, a number of IS-MPC evaluations and numerical integration steps are executed, corresponding to the *MPC-frequency* parameter, before the Agent policy is evaluated and apply again. If, at this point, the simulation fails for any

reason (e.g. the MPC or Inverse Dynamics solvers crash because no solutions can be found) the episode is *terminated* early and a penalty is given to the Agent. If, on the other hand, the robot is able to complete the footstep plan successfully, the episode is *truncated* early and a bonus is awarded to the agent.

2.2.3 Reward function

The reward function defines how the Agent learns. Accurate design of a reward function encourages or discourages some behaviours of the agent. For complex system like a humanoid robot, reward functions tend to contain a lot of different terms. In our case, the reward function is defined as the sum of different terms that reward some behaviours, like following the nominal footstep plan, or proper distancing of the feet to avoid self-collisions, and penalize others, like displacement magnitude or early termination. The reward function is defined as:

$$R(s_k, a_k, s_{k+1}) = r_{\text{state}} + r_{\text{phase}} + r_{\text{cp}} + r_{\text{CoM}} + r_{\text{fs}} \tag{6}$$

Where r_{state} is a bonus or a penalty based on the environment status, i.e. whether the robot is in a safe set or the episode was terminated early; r_{phase} is a term that depends on the gait phase the robot is in: single support or double support; r_{cp} is a bonus awarded for reaching checkpoints in the footstep plan, placed every third step; r_{CoM} is a bonus given for following reference CoM velocities; finally, the r_{fs} term penalizes vicinity of the two feet, which would result in self-collisions. In detail, each term is defined as follows. For exact numerical values of each coefficient, see our implementation [8].

$$r_{\text{state}} = \begin{cases} \text{alive bonus} & \text{if robot alive} \\ \text{penalty} & \text{if IS-MPC solver solution innacurate} \\ \text{penalty} \cdot 0.5 & \text{if IS-MPC solver max iters.} \\ \text{penalty} \cdot 3 & \text{if Inverse Dynamics not feasible} \\ \text{penalty} \cdot 5 & \text{if feet collide} \end{cases} \tag{7}$$

$$r_{\text{phase}} = \begin{cases} r_{\text{swing}} & \text{if is in swing phase} \\ r_{\text{ds}} & \text{if is in double support} \end{cases}$$

where:

$$r_{\text{swing}} = -w_h(z_c - h)^2 + w_a \frac{a^T a}{\Delta t_r + 10^{-3}} \tag{8}$$

and

$$r_{ds} = -w_a a^T a \tag{9}$$

The first term in (8) keeps the height to the constant value h assumed in the LIP model, while the second one penalizes actions a of higher magnitude as the end of the swing phase approaches. Δt_r is the normalized remaining time in the swing (or single support) phase of the gait. The only term in (9) simply penalizes the squared norm of actions while in the double support phase of the gait.

The term r_{cp} defines checkpoint bonuses, used to encourage the robot in following the whole plan rather then terminating early and accumulating rewards for lots of smaller episode instead of a single, longer one (so-called reward hacking):

$$r_{\text{cp}} = \begin{cases} \text{checkpoint bonus} & \text{if robot reaches multiple of 3 footsteps} \\ \text{checkpoint + end bonus} & \text{if robot reaches the end of the plan} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The last two terms in (6) reward the Agent for following the nominal footstep plan, in particular:

$$r_{CoM} = \text{Ker}_{\dot{x}_c}(e_{\dot{x}_c,\dot{x}_c^r}) + \text{Ker}_{\dot{y}_c}(e_{\dot{y}_c,\dot{y}_c^r}), \tag{11}$$

where $\{\dot{x}, \dot{y}\}_c$ are the x and y velocity of CoM and the superscript ^{r} indicates the reference, and

$$\begin{aligned} r_{fs} &= r_{fs_1} + r_{fs_2} \\ r_{fs_1} &= -\text{Ker}_f(\Delta f^2) \\ r_{fs_2} &= \text{Ker}_f(|\Delta f^r|^2) \end{aligned} \tag{12}$$

,

where r_{fs_1} penalizes footsteps that place the feet too close together (and would result in a self collision, or in general a poorly stable pose for the robot), and r_{fs_2} rewards the displaced footstep for being closed to the original one, where Δf^r is the pose of the original footstep with respect to and expressed in the reference frame of the current support foot.

Finally, the function $Ker()$ used in the above terms is defined as [3]:

$$\text{Ker}_v(e) = w_v \exp(-(e/\sigma_v)^2) \tag{13}$$

2.3 Agent Interaction

The agent interacts with the environment by giving deviations of the footstep from the reference plan. This gives it a crucial role and great control authority. The steps can be displaced in multiple ways, for example only the next footstep or the whole plan can be displaced. The displacement acted by the agent can make the robot recover after an external disturbance force is appllied, or it can help the robot in keeping

balance when the floor is not completely flat and has a slope. Other parameters which are not controlled by the Agent, like the duration of each gait phase, are also crucial for maintaining balance in the presence of external disturbances. Effect of these parameters on the agent is discussed in Section 4.

The agent have a saturation on the cumulative actions so to have a maximum displacement of each footstep. If this saturation is ignored the agent will anyway learn but much slower.

2.4 Training Routine

The training of the environment follows the classical Reinforcement Learning procedure for on-policy algorithms. First of all a *roll-out* is performed, that is a simulation of the environment with the Agent acting according the the current policy. The policy is defined as a Neural Network with two hidden layers of 64 nodes each and tanh activation functions and no bias on the second layer. The network weights are initialised at random. During a rollout, informations are stored as a tuple $D = (s_t, a, r, s_{t+1})$ composed by current state s_t, the action taken a, the reward gained r and the next state reached s_{t+1}. A single rollout collects a given number of these tuples inside a *rollout buffer*, possibly across multiple episodes if one terminates or is truncated. The buffer can also be filled by different environments running in parallel. After a rollout finishes, the policy is updated by using the rollout buffer to compute *returns* and *advantages*. The advantage is computed as

$$\begin{aligned} \hat{A}_t &= \sum_{k=0}^{T-t} (\gamma\lambda)^k \delta_{t+k} \\ \text{where} \quad \delta_t &= r_t + \gamma V(st+1) - V(s_t) \end{aligned} \tag{14}$$

where $V(\cdot)$ is the value function, $\gamma = 0.99$ is the discount factor that ensures convergence and makes the Agent "look forward". The advantage is strictly related to the reward earned during the roll-out and the value function as seen in (14). The advantage drives the update of the policy, as it contains essential information on the benefit of taking some action with respect to others. The policy update is performed using the stochastic gradient descent algorithm on a suitably computed loss functions, dividing the dataset in mini batches. For a single rollout, the network is in general updated for multiple epochs, 2 to 10 epochs. The loss function used for SGD is the sum of three losses, the *clipped loss*, the *actor loss*, and the *entropy loss*, scaled with some weights:

$$L = L^{CLIP} + w_C L^C + w_E L^E \tag{15}$$

The clipped loss is the focal point of the PPO algorithm, and it is the loss that drives the policy update. It is defined as:

$$
\begin{aligned}
L^{CLIP} &= -\hat{\mathbb{E}}\Big[\min\big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, i+\varepsilon)\hat{A}_t\big)\Big] \\
\text{where} \quad r_t(\theta) &= \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}
\end{aligned}
\tag{16}
$$

Where $A(t)$ is the advantage and $r_\theta(t)$ is the ratio between the new policy and the old one. If the ratio is too high, it means that the policy changed drastically, and it is clipped with a factor of $\varepsilon = 0.2$ to guarantee stable and reliable learning. The sign is negative because we want to maximize L_{CLIP} but loss optimizers are usually minimizers by default.

The critic loss is just the Mean Squared Error (MSE) norm of the value function. The value function indicate how good is the current policy based mainly on the reward gained, the value critic network will use this loss to improve the estimation of the policy. The critic loss is scaled in the complete loss by a weight $w_C = 0.5$ because the main goal of the algorithm is to learn a good policy and not just to accurately predict the future state of the environment.

The entropy loss is the entropy of the current distribution. The sign is flipped in order to maximize the entropy and encourage the agent to explore new actions. The entropy loss is scaled by a weight $w_E = 0.01$ but this parameter is often tuned depending on the training performances of the Agent and on the environment.

The training alternates the roll-out and learning phases for a large amount of iterations, allowing the Agent to explore the environment at large and understand which actions lead to the maximum possible reward.

3 Different approaches to footstep displacement

Consider a footstep *plan*. It is made of multiple footsteps, each described by its support foot, pose (position + orientation in space) and single and double support phases duration.

3.1 Approach 1: Only First Footstep Displaced

3.1.1 Actions

The first approach we describe is that of modifying only the *next footstep* compared to the support foot currently on the ground, i.e. displacing the target pose of the foot currently swinging in the air. As with all methods we discuss, it has advantages and disadvantages: modifying only the next footstep means that the original plan will not be significantly modified, hence the robot will better track the desired trajectory, particularly in nominal conditions. To further ensure this, a simple penalty proportional to the magnitude of each action can be added to the reward function. On the other hand, displacing only one footstep might generate constraints that are unsatisfiable by IS-MPC, and the agent has to learn how to generate satisfiable footsteps. Figure 3 shows an example of the robot following a reference straigh line trajectory for 15 footsteps, ending with five steps on the spot. Reference footsteps are drawn as filled rectangles, red for left foot and blue for right foot. Actual footsteps are drawn as unfilled rectangles on the same color. The dotted line in green shows the center of mass trajectory.

3.1.2 Rewards

The reward for this approach is the same as described in (2.2.3). Notably in this case, the Agent manages to follow the original plan reasonably well even when the behaviour is not explicitly encouraged by adding the r_{fs_2} term in (12).

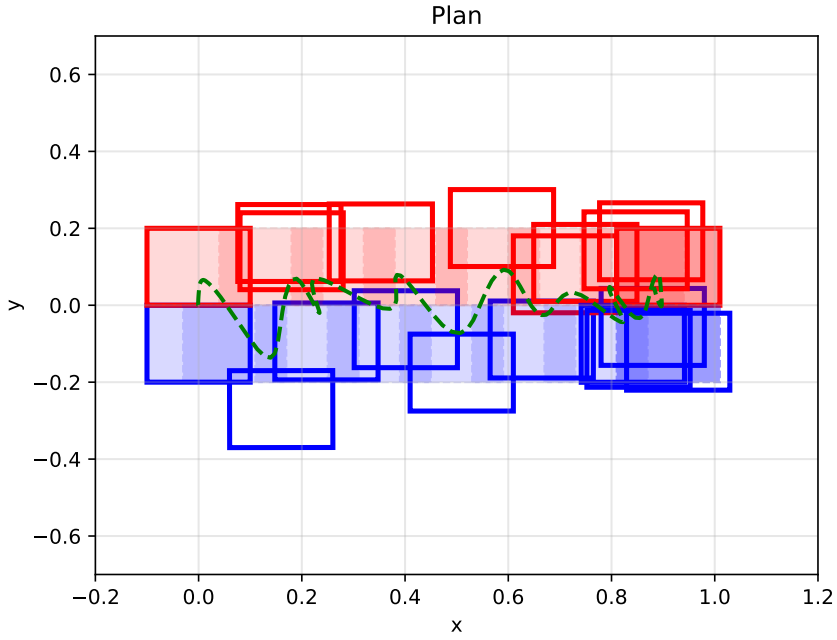


Figure 3: Example of reference and modified plan according to approach 1. The robot started with the right foot.

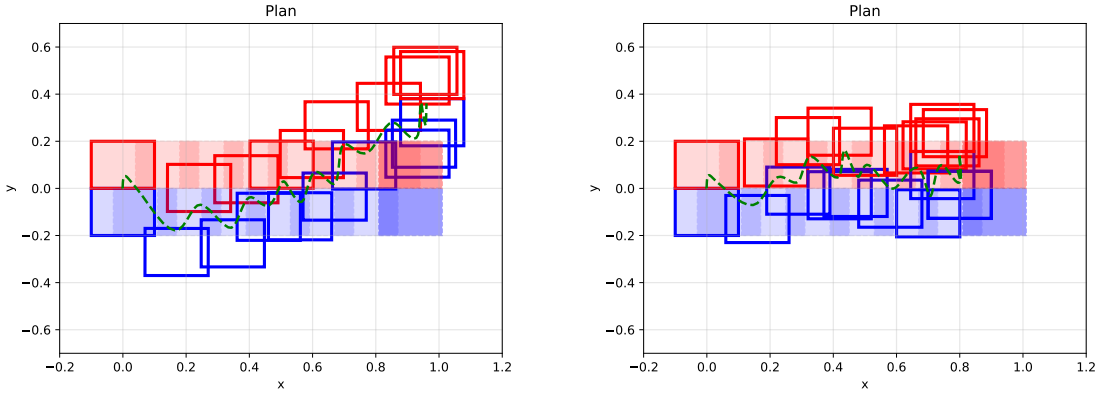
3.2 Approach 2: Full Footstep Plan Displacement

3.2.1 Actions

The second approach we described displaces the whole footstep plan by the same amount, starting from the desired pose of the foot currently in swing. This approach keeps the the plan consistent and successive footsteps feasible for MPC, as long as the reference nominal plan and the displacement are. However, successive displacements of the plan accumulate and eventually the robot strays away from the nominal plan. This can be solved by appropriately modifying the reward function.

3.2.2 Rewards

When the whole plan is displaced, as opposed to only the next footstep, successive displacements increasingly deviate the footstep plan from the nominal one. An example of this is shown in Figure 4a, where the robot follows the same straight line trajectory as Figure 3. The problem is solved by adding the r_{fs_2} term in (12) to the reward function, which rewards placing the footstep near its position in the original plan. An example of this is shown in Figure 4b.



(a) Without using r_{fs_2} term in (12)

(b) Using r_{fs_2} term in (12)

Figure 4: Example or reference and modified plan according to approach 2. The robot starts with the right foot

3.3 Approach 3: Footstep Displacements Gradually Scaled

The third approach combines the first two: footsteps are displaced gradually less as they get further from the current one. This means that the action is fully applied to the next footstep, and geometrically scaled by a factor ξ for the successive ones. The scaling factor ξ is a hyperparameter that needs to be tuned, and we found that values between 0.9 and 0.99 gives the best results. The scaling is done in such a way that the plan has a higher chance of remaining feasible for the MPC compared to the first approach, while gently guiding the robot back to the nominal plan after actions are applied. This behaviour can be further encouraged by adding appropriate terms to the reward function, like in Section 3.2.2. Footsteps are displaced according to the formula:

$$FS'_{t+i} = FS_{t+i} + \xi^{i-1}\Delta_t, \;\; i = 1, 2, ..., \Sigma \tag{17}$$

where i is the index of the next footsteps, up until the last one in the planΣ. FS'_j is the displaced pose of the modified footsteps, FS_j is the current pose before the actor action is applied, $\xi \in [0, 1]$ is the scaling factor and Δ_t is the displacement given from the actor as action at time t. In general, the current pose FS_j includes the displacements caused by previous actions of the actor.

The approaches shown in Sections 3.1 and 3.2 are degenerate cases of this one, and can be recovered by choosing $\xi = 0$, and $\xi = 1$, respectively.

3.3.1 Constant Scaler Parameter

An example of a footstep plan modified by this approach, with $\epsilon = 0.9$, is shown in Figure 5. Notice how the plan deviates from the nominal one more than in Figure 3,

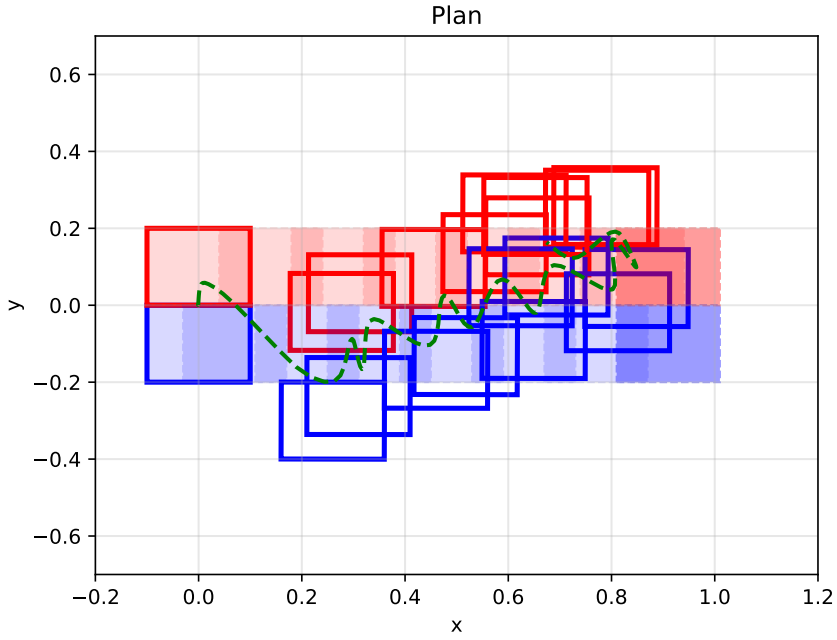


Figure 5: Example of reference and modified plan according to approach 3. The robot started with the right foot.

in particular towards the final footsteps, where the robot goes backwards instead of remaining on the spot.

3.3.2 Learned Scaler Parameter

A slight modification of this approach involves the Agent learning the scaling factor as well. Including such a parameter in the RL-framework means to delegate to the Agent to learn how the scaling of footsteps beyond the first one affects feasibility of the remaining plan. In this way the agent is urged to find a compromise between a large scaling, which entails a more rapid convergence to the original plan, and a smaller one, which can results useful in the case in which the robot has to deviate from the reference plan in order to maintain balance after a collision with an external destabilising force.

In this settings the scaling factor is completely part of the action space of the agent and changes each step. Like other actions, the scaling action is sampled from a Normal $\mathcal{N}(\mu, \sigma)$ distribution with parameters μ and σ predicted by the agent, since the action is continuous. The sampled action value is then rescaled in the range $[0, 1]$ in order to limit displacement's scaling capabilities between the two degenerate cases: $\xi = 0$, (only the first foot displaced), and $\xi = 0$ (the full plan displaced). This kind of approach extends the original problem formulation to the class of parametrized action space MDPs, in which actions tuples are made by a set of standard action with which the agent interacts with the environment, A_{int}, and a set of continuous parameters A_p

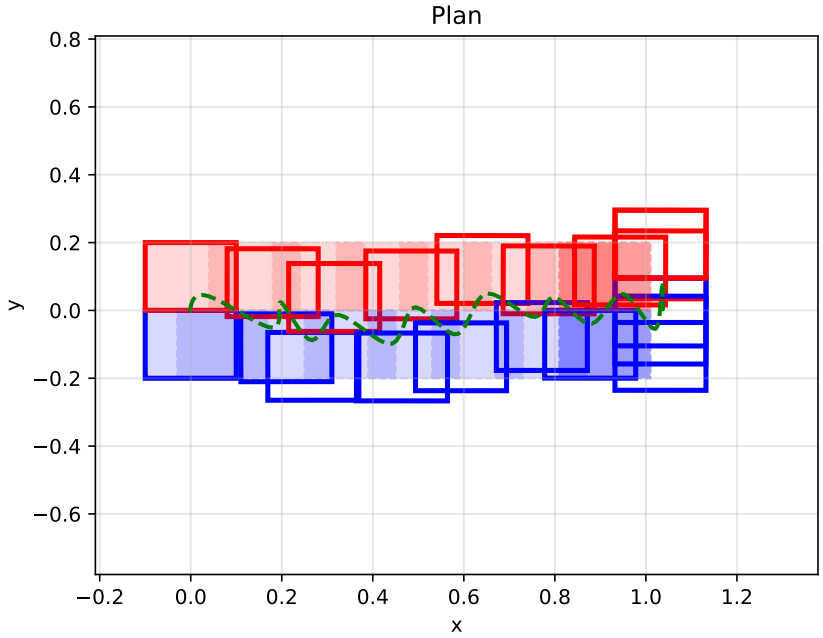


Figure 6: Example of reference and modified plan according to approach 3, with trainable scaling. The robot started with the right foot.

that scale actions $a_{int_i} \in A_{int}$ in order to regulate their intensity or magnitude. This Reinforcement Learning paradigm is extensively treated in [9].

In this case , $A_{int} = \{\Delta_x, \Delta_y, \Delta_\theta\}$, thus $dim(A_{int}) = 3$, and $A_p = \{\xi\}$, thus $dim(A_p) = 1$. An interesting behaviour is shown training an agent with this approach: as can be seen in Figure 6, in purely nominal conditions, i.e. without external disturbance forces and without any perturbation of the ground's slope, the agent tends to modify to plan scaling as much as possible the displacements of the footsteps after the first one: the result is a trend to displace significantly only the first footstep keeping the other ones to their original position, until they become one by one the first footstep to displace, leading to a behavior close to approach 1.

The list of the overall predicted scaling factor ξ for the plan executed and shown in figure 6 is represented in 7.

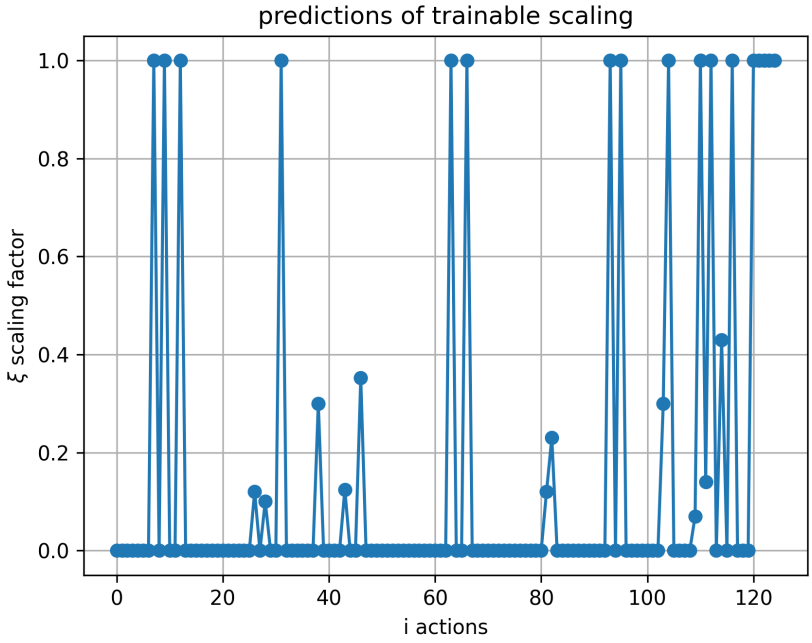


Figure 7: Evolution of the scaling factor ξ predicted in figure 6. In nominal conditions, the supremacy of the approach 1 -like actions is clear.

3.4 Approach 4: Learning angular momentum

The idea is that the agent can learn something about the angular momentum of the humanoid and use this information for a better correction. The LIP prediction model of the IS-MPC controller assumes constant angular momentum of centre of mass, as $\dot{L}_c = 0$, but when the robot is subjected to external forces or is on inclined terrain the angular momentum often changes and becomes a relevant contribution to robot dynamics, making the LIP model not accurate anymore and leading IS-MPC into instability. On the other hand, IS-MPC does not have any information about the angular momentum about the x, y coordinates of the contact point $L = [L_x, L_y]$. The naive approach, adopted in [3], it to use the ALIP model [10], in which L is considered as part of the state, as the prediction model for the MPC formulation.

We opted for a different approach: our idea is to insert information about the angular momentum in the observable state of the Agent, a give it a reward based on the difference between the actual and the desired angular momentum about the contact point, forcing the Agent to learn about this quantity and providing to it an additional information about its dynamical settings.

3.4.1 Observable State

For this approach, information about the current angular momentum and the desired one are added to the state. The desired angular momentum is obtained by running simulations in the nominal environment and saving the data. Since the agent acts

once every n IS-MPC evaluation cycles (where n being a number associated to the *MPC-frequency* parameter of Section 2), it is possible to obtain a comparison between the current and the desired angular momentum only every n IS-MPC cycles.

3.4.2 Reward Function

A reward term is also added to encourage tracking of the desired angular momentum $\mathbf{L_{des}} = [L_{xd}, L_{yd}]$, like in [3].

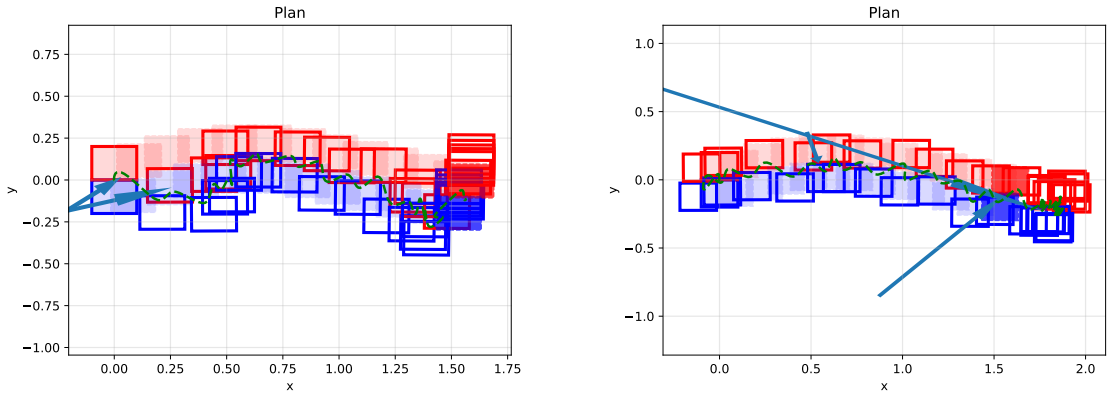
Defined the angular momentum error as $\mathbf{e_L} = \mathbf{L_{des}} - \mathbf{L}$, and the associate reward term as:

$$r_{L_{des}} = Ker_L(|\mathbf{e_L}|) \tag{18}$$

with σ_L standard deviation of the $Ker_L(\cdot)$ function sufficiently small to detect deviation of L of the order of 10^{-2}.

The drawback of this kind of approach is that it is applicable only in the cases of tasks that are feasible and fully executable in nominal conditions by the IS-MPC controller, from which we are allowed to extract a signal L_{des} to be used in the reward term. We explore in section 4 that this isn't always the case.

An example of this approach is shown in Figure 8b, where the robot tracks a slightly curved line which is feasible for IS-MPC in nominal conditions.



(a) Without learned L.

(b) With Learned L.

Figure 8: Comparison between 2 agents: one trained with approach 3 ($\xi = 0.9$, the other with approach 3 + Learned Angular momentum about the contact point.

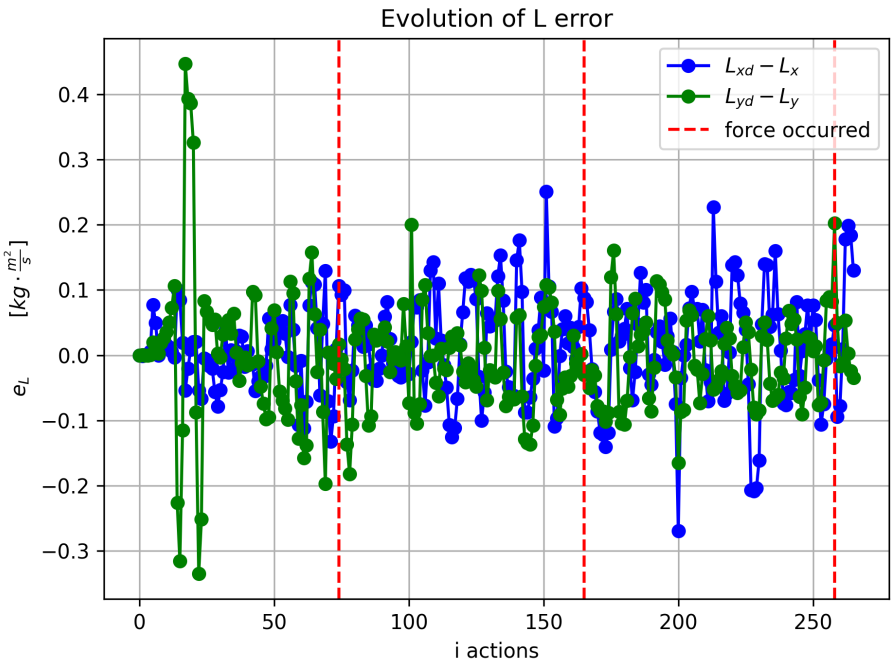


Figure 9: Evolution of the angular momentum error e_L .

As can be seen in Figure 8, the agent trained with the information about current and desired L is able to better track the reference path. Both agents are trained with a slope of the ground of 2° about both the x and y axes, and external forces. Note that IS-MPC is not able to solve this task in this conditions.

Looking at the angular momentum error e_L value over the episode shown in Figure 9, we can note that the external force that perturbs the L-trained agent creates little spikes in the error, as expected (especially the 3° one that is the strongest). Note also that the tracking L is not perfect. This is also expected, since IS-MPC controller uses the LIP model and not the ALIP.

3.5 Curriculum Learning

Curriculum Learning is a learning technique in which a model is trained on a simplified version of the task, e.g. in a simplified environment or with a reduced action set, and the difficulty is gradually increased as the agent learns. Instead of placing the agent directly in a harsh environment, with high disturbance forces and ground slope, the agent starts learning how to follow the nominal plan without any disturbances. As the agent learns, disturbance forces and ground slope are gradually increased. This technique can be applied to all three approaches presented up until now, and in general yields better performance and generalizes to different reference plans better than classic learning.

3.5.1 Levelling System

To implement curriculum learning, a levelling system that increases the slope of the floor and the magnitude of the disturbance forces needs to be implemented. At level 0 the ground is flat, which allows the agent to learn how its actions affect the system. In this phase, the agent should learn not to modify the reference plan at all, as IS-MPC is capable of maintaining stability If the robot completes the reference plan, the difficulty level is increased and with it the magnitude of the forces and the slope of the floor. Each time the difficulty level increases, the robot already has a policy that worked in simpler environments, and the PPO algorithm is designed to slowly adapt it to the new environment. In principle, this means that the agent will keep the information learnt in easier level, and in particular to follow the reference and displace footsteps the least possible, but become more robust to increasing disturbances. On the other hand, an agent trained with this technique which is capable of operating in harsher environments will also be able to operate well on the simple environments it was initially trained on.

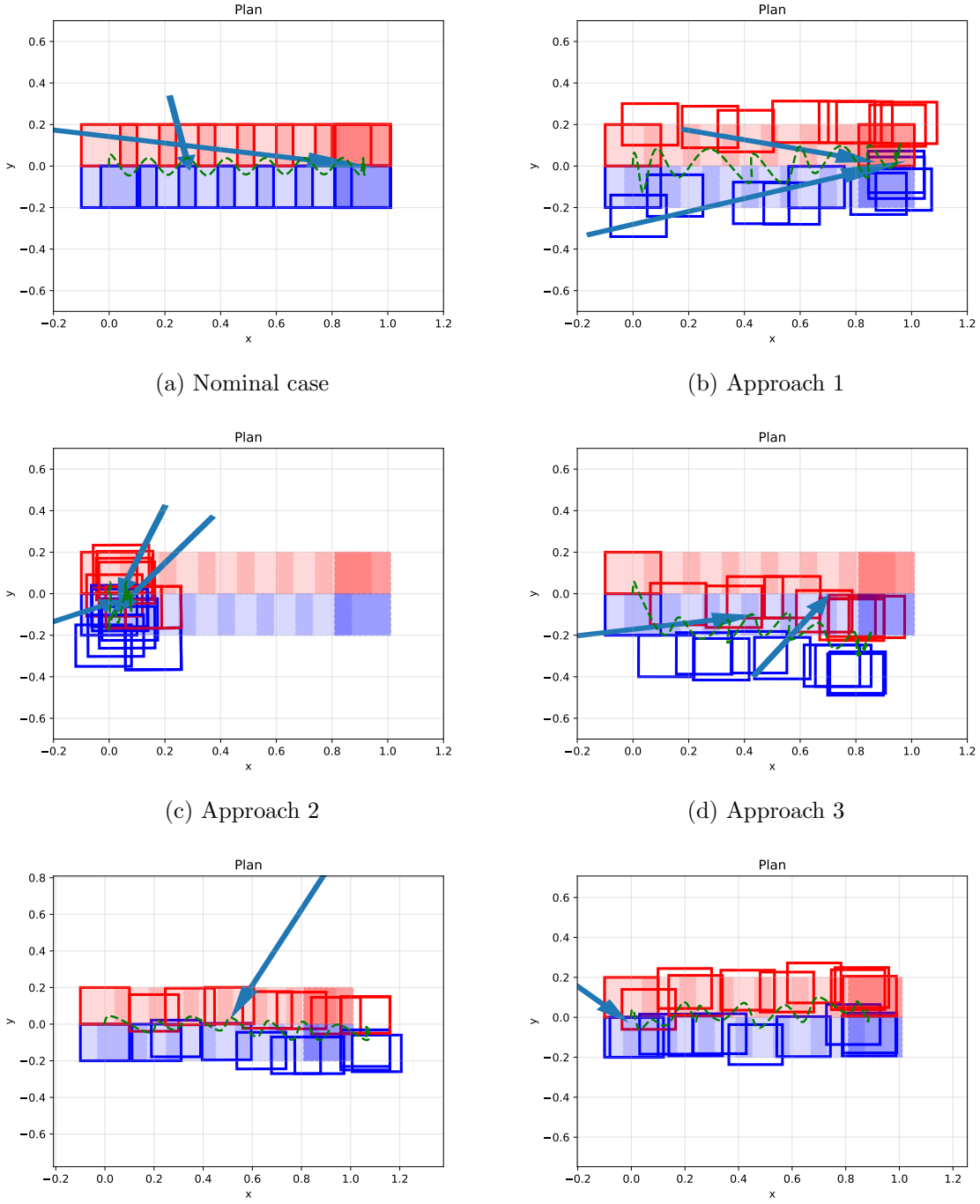
4 Simulations and Results

In this section we show some simulation results, with the robot operating in different scenarios, subject to external forces and sloped terrain. We compare the behaviour of the robot across the three different approaches described in Section 3 and with the nominal (IS-MPC only) case.

The different agents have all been trained using curriculum learning, starting from flat ground and gradually increasing to about 3° of slope. All agents start from level 20 in the system described in Section 3.5.1, and training is stopped manually when reaching level 30 or above. Until level 26, the agents are trained with the robot only following a straight line trajectory of 20 footsteps in total. After level 26, when a new episode starts a new footstep plan is chosen randomly from a pool of pre-defined plans. These include: walking on the spot, walking forwards and backwards in a straight line, and walking left and right in a straight line. We also show simulations on one validation plan the agent has never seen before, which differ from training plans both in length (i.e. number of footsteps) and in desired trajectory. Each footstep plan tested includes 5 final footsteps in which the robot walks on the spot. In all cases, the MPC prediction horizon is set to 150 steps and the sampling time to 0.01 seconds. For each footstep plan, we test four approaches in the following order: nominal (no RL agent), displacing only the next footstep (Approach 1, Section 3.1), displacing the whole plan (Approach 2, Section 3.2), and scaling the displacement (Approach 3, Section 3.3) with $\epsilon = 0.9$. Each step, a random force of magnitude between 50N and 150N is generated with a 3% of probability, and a body part selected at random between the torso, the body, and one of the feet. The force is directed towards the CoM of the selected body part, plus a small random displacement.

4.1 Simulation 1: Straight line forwards - no slope

In this simulation, the robot follows a straight line forwards, made of 15 steps in total, plus 5 more on the spot. The plots show right and left footsteps in blue and red respectively. Nominal footsteps are shown with dashed rectangles with transparent filling, while actual footsteps are shown with empty rectangles of the respective colour. Purple arrows show a top-down view of external forces, their norm is scaled by a factor of 1/100 with respect to the actual force. Since this is the trajectory that is most seen during training, especially during early levels, all approaches easily complete the plan, including in the nominal case. Notice, however, how approach 1 (displacing only the next footstep) better follows the reference plan than approaches 2 and 3. This is to be expected, because the agent is rewarded first and for most for completing the plan (that is to take as many successful footsteps as possible) and secondly for limiting the magnitude of the displacement and staying close to the nominal plan. In this context, approaches 2 and 3 in general will displace future footsteps more than approach 1.



(a) Nominal case

(b) Approach 1

(c) Approach 2

(d) Approach 3

(e) Approach 3 - trainable ξ

(f) Approach 3 - learned L.

Figure 10: Simulation 1: Straight line trajectory - Reference and actual footsteps in the presence of different external forces on flat ground.

As we can note in the graph (c) of Figure 10, scaling factor in the trainable case becomes on average near to 1.0 as soon as a disturbance force occurs. So Agent tends to recover dynamical balance trying also to displace the following footsteps and not only the first one, moving from approach 1-like to 2-like when nominal conditions are no longer valid. An overview of how ξ parameter is predicted over the episode according

the approach $3b$ is here reported in figure 11. Near, the evolution of the angular momentum error about the contact point as performed in the learned L approach 3 agent, that is the one that reproduces the reference path better.

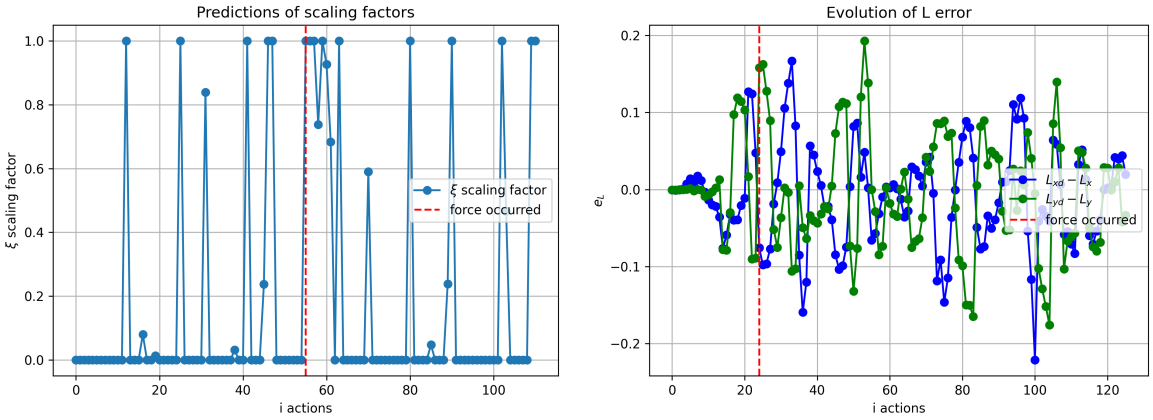
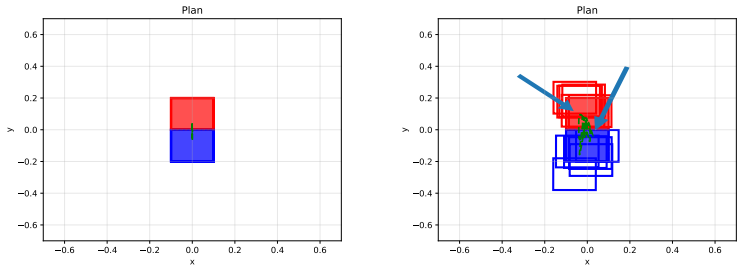


Figure 11: (a) Evolution of the scaling factor ξ predicted in figure 10 (e). When a force occurs, it happens the switch from the approach 1 -like to the 2 -like.
(b) Evolution of angular momentum error e_L for trained L agent. As in the presentation case, tracking is not perfect.

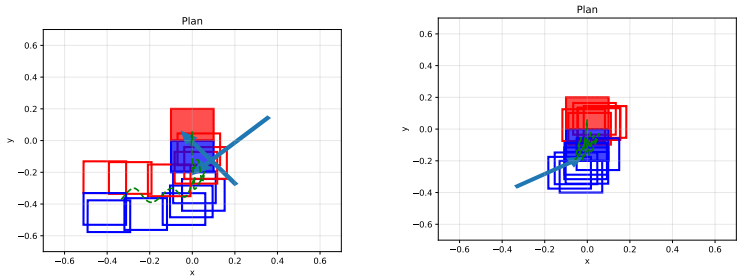
4.2 Simulation 2: Walking on the spot

In this simulation the robot has to walk on the spot, without moving in any direction. This is the first case we show in which our Agent works better than IS-MPC alone. This is likely because, in this case of walking based on the LIP Dynamics in this conditions keeping feet on the same position is impossible to keep balance. Our RL based methods being able to modify the footsteps plan can recover balancing when needed. While hard to notice in these plots, in Figure 12a IS-MPC diverges within a couple of steps, while all three approaches using our Agent manage to maintain equilibrium, with minimal displacement of the footsteps as show in Figures 12b and 12d. Also in this case, it's evident how displacing the whole plan will in general result in worse tracking of the desired trajectory.

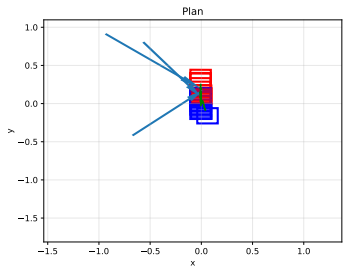
As we can note in Figure 13, in the case of the approach 3 with trainable scaling factor ξ, the interpretation of the ξ predictions are a bit more involved: for walking on the spot task the behaviour of tending to scale the whole plane after a force is still present (more evident after the application of the 2^{nd} and the 3^{rd} forces), but in general when no forces are applied the agent does not show a clear preference for either of the extreme cases: it tends to variy the approach mixing equiprobably the displacement of the whole plan with that of the first footstep only.



(a) Nominal case (b) Approach 1



(c) Approach 2 (d) Approach 3



(e) Approach 3 - trainable ξ

Figure 12: Simulation 2: Walking on the spot - Reference and actual footsteps in the presence of different external forces on flat ground.

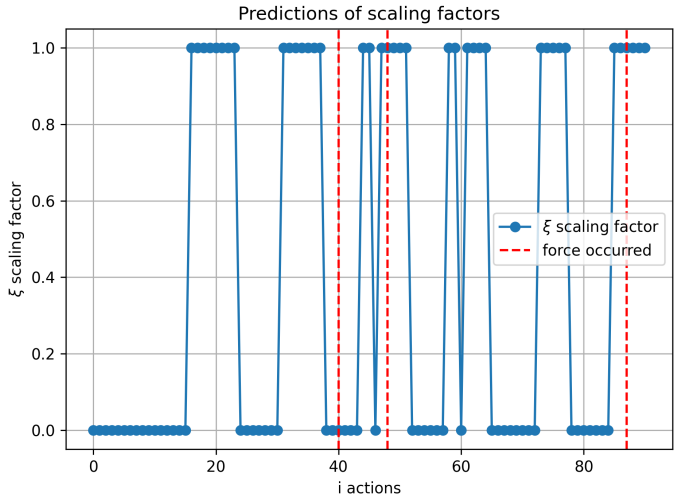


Figure 13: Evolution of the scaling factor ξ predicted in figure 12 (e).

In this case, since the walking on the spot's task is not feasible for the nominal ISMPC controller, it is not possible to impose a desired angular momentum about the contact point L_{des}, so the L based approach is impracticable.

4.3 Simulation 3: Straight line forward - inclined terrain

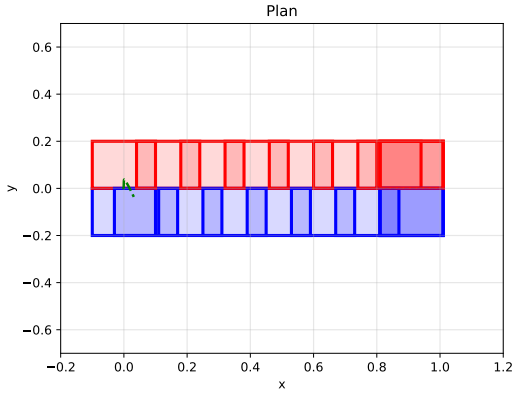
In this simulation the robot follows the same trajectory as Simulation 1, but this time the terrain has of 3° along the x direction and 1° along the y direction. This is simulated by modifying the gravity vector so to have components also along the x and y directions. IS-MPC fails immediately, as the LIP dynamics described in (1.2) assumes the gravity vector to be $g = \begin{bmatrix} 0; 0; -9.81 \end{bmatrix} N$ and even small deviations from this results in quickly diverging modes. With greater computational cost, the MPC prediction horizon can be increased from 150 steps to 250 steps, which results in all approaches reaching the end of the plan even in the presence of more than 10° of slope along both directions.

We can note from Figure 15 that in the case of inclined terrain, the agent trained with learnable scaling factor ξ tends to scale the plane in a very few situations: most steps are performed by displacing the whole plane. In fact, the agent performing in Figure 14e (e) employs a similar approach to that in Figure 14c of displacing the whole plan right following the terrain inclination.

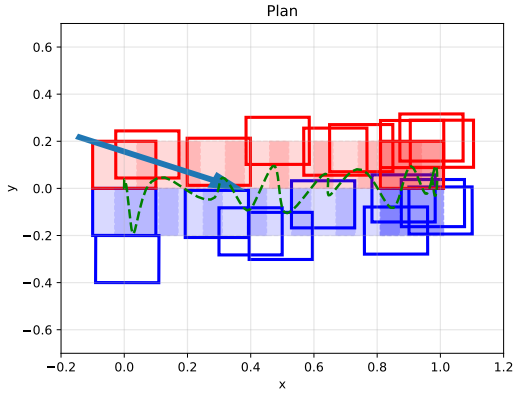
Another interesting result is the difficulty of the Learned L agent to perform well in this scenario. We can explain this phenomenon considering the fact that the agent is trained to solve the path in such a way that its angular momentum about the contact point is, for each step, the same obtained during in nominal conditions over the linear path. But performing the same gait on inclined terrain does not entail good performance. So the agent is not able to learn effectively what to do, since tracking L brings to falling, while by not tracking L the agent can't gain the associated reward. This results in the error e_L being consistently higher in this context, with peaks after forces applications, as can be seen in Figure 15.

4.4 Simulation 4: Validation trajectory 1

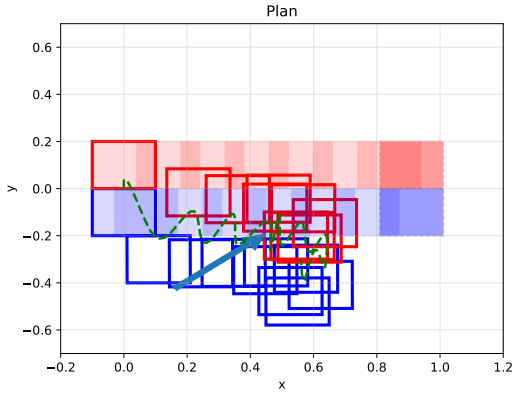
In this simulation the robot follows a validation trajectory which the Agent has never seen before during training. This lets us properly evaluate the performance of the agent and, in particular, its ability to generalize. The trajectory of this simulation is a straight line, followed by a section where the robot has to walk to the left, without rotating on the spot first. The change of direction challenges the agent's ability to avoid feet collision. The plan is also twice as long than all the training plans. To better evaluate the agent's ability to follow the trajectory, no terrain inclination is considered in this case. Results are mixed: the trajectory is feasible for the nominal case, but using approaches 1 and 2 the robot is unable to complete the footstep plan.



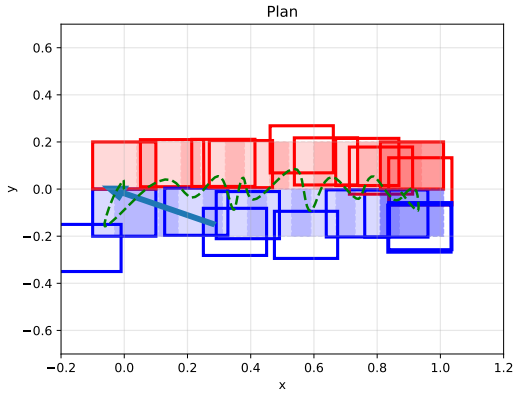
(a) Nominal case



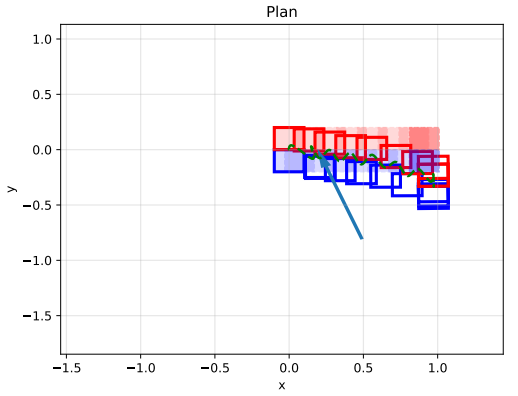
(b) Approach 1



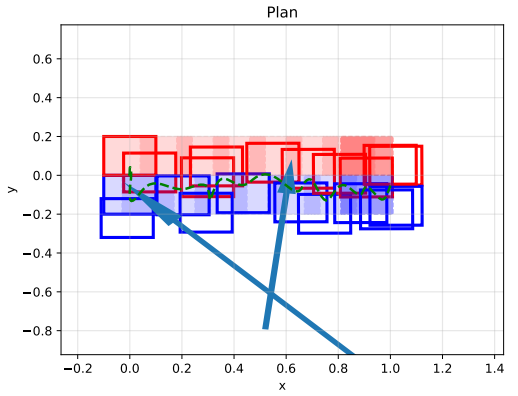
(c) Approach 2



(d) Approach 3



(e) Approach 3 - trainable ξ



(f) Approach 3 - learned L

Figure 14: Simulation 3: Straight line trajectory on a inclined plane - Reference and actual footstep in the presence of different external forces and on an inclined plane.

Across different runs for these two approaches, the most common mode of failure is collision of the feet with one another. Furthermore, tracking of the plan is poor for approach 2. Approach 3 instead consistently ends the footstep plan, however with poor tracking. These results indicates that agents which implement the first two approaches do not generalize well, and may have overfit to the training data. On the

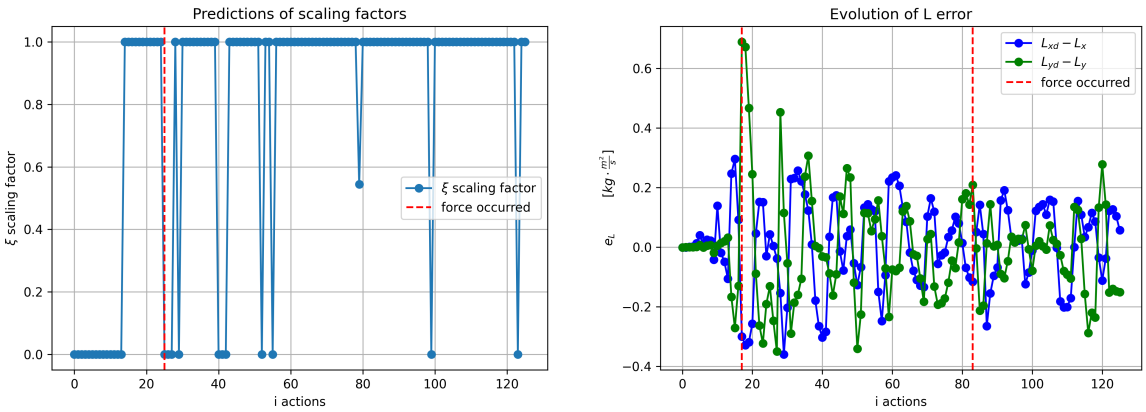


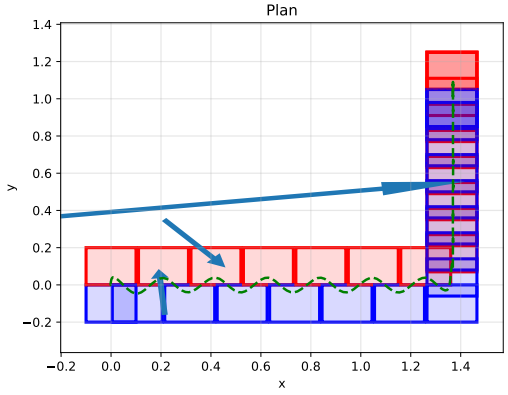
Figure 15: (a) Evolution of the scaling factor ξ predicted in figure 14 (e).
(b) Evolution of error e_L of the agent in figure 14 (f).

other hand, the agent trained according to approach 3 better generalized to unseen trajectories, although both approaches 3 (without and with trainable scaling factor) are not able to manage the feet collision problem on the lateral section of the path. Note that approach 3 with learnable ξ relies a interesting synthesis of performances of approach 1 and 2, while the agent trained according to approach 3 with learned angular momentum improves tracking capabilities, as we can see comparing Figures 16d and 16f.

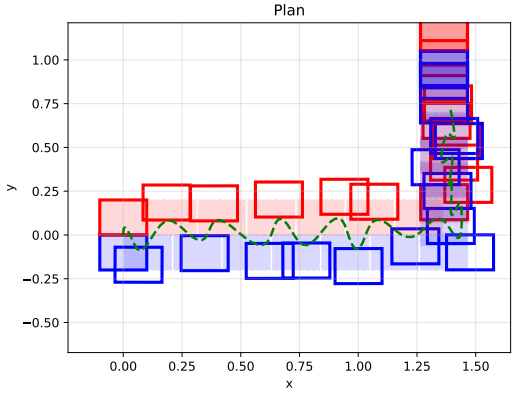
4.5 Simulation 5: Action saturation

In this simulation new agents are trained with the approach in Section 3.3 and adding a saturation to the actions that prevents feet collision, both with and without trainable parameter ξ and learned L. In practice, this is implemented as a bounding box around the current support foot: each time the agent takes an action, if the displaced footstep would end up inside an unsafe region around the support foot, the displacement is prevented. Some simulations following the validation trajectory of Section 4.4 are shown in Figure 17: the first is on flat ground and in the presence of external forces, and the second is on inclined terrain with 3° of inclination along the x direction and 1° along the y direction, but in absence of external forces. The third one is with learned L, with external force and flat ground, to highlight the better reference tracking than in Figure 17a. The fourth, in Figure 17d, is with trainable ξ, without external forces.

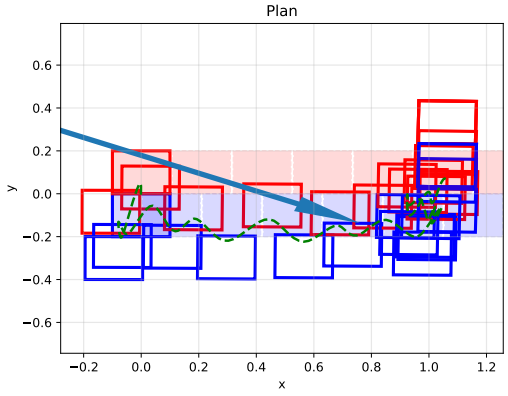
In all four cases, the robot displays the best trajectory tracking of all previous simulations. In Figures 17b and 17d in particular, the robot can complete the footstep plan with terrain inclinations which are unfeasible both for IS-MPC alone and for the agents shown in previous simulations. Figure 17e and 17f shows respectively the evolution of the ξ parameter and the error e_L along their associated episodes.



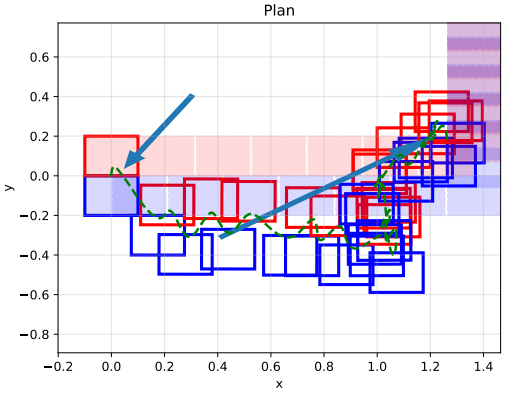
(a) Nominal case



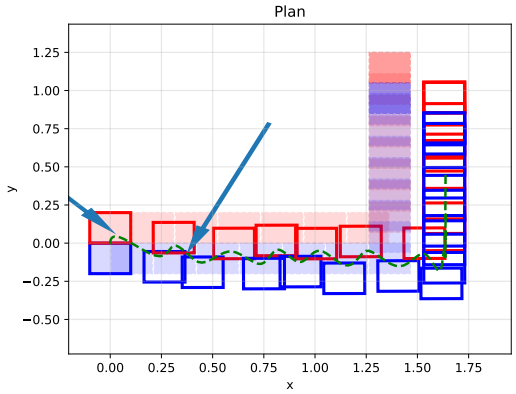
(b) Approach 1



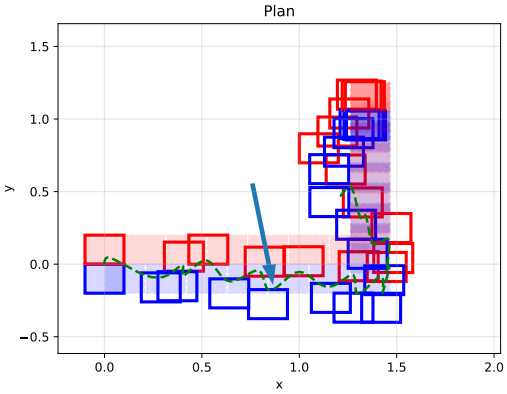
(c) Approach 2



(d) Approach 3

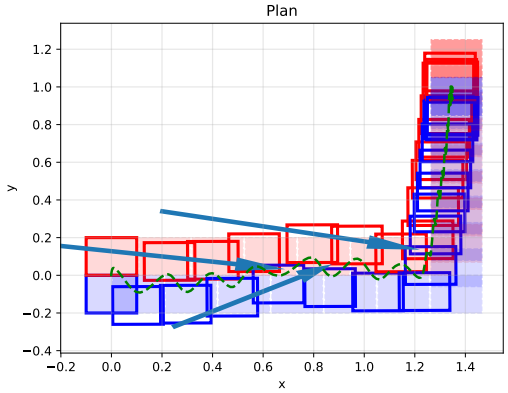


(e) Approach 3 - trainable ξ

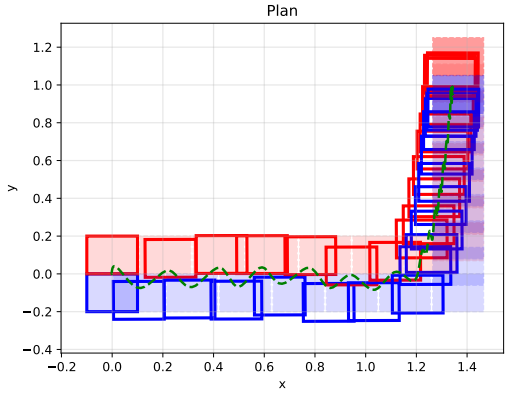


(f) Approach 3 - learned L

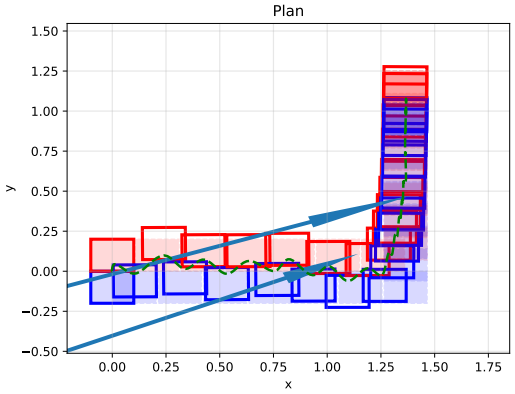
Figure 16: Simulation 4: Validation trajectory - Reference and actual footsteps in the presence of different external forces on flat ground



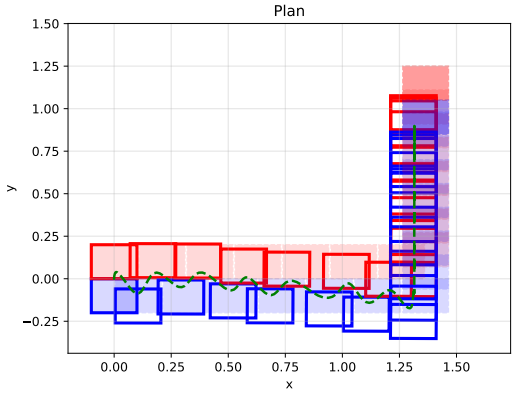
(a) Flat ground, external forces



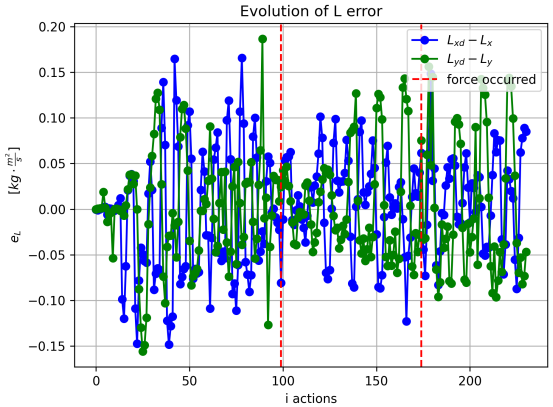
(b) Inclined plane, no external forces



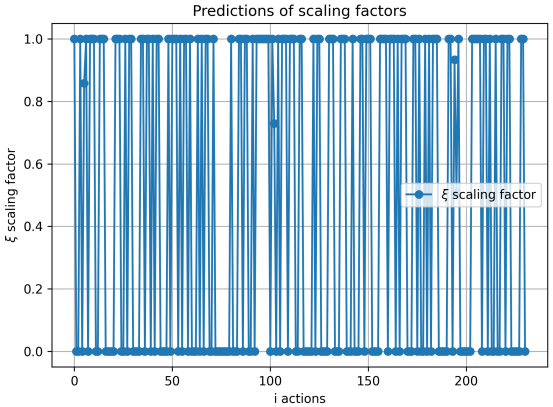
(c) Flat ground, external forces - learned L.



(d) Inclined plane, no external forces - trainable ξ



(e) L error.



(f) Predicted ξ.

Figure 17: Simulation 5: Validation trajectory - the agent is trained with approach 3 and with action saturation to prevent feet collision.

5 When to Use The Agent

The RL-augmented MPC framework works well when there are external disturbances that make the original plan not feasible. However, if no perturbations are present, it is preferable to "turn off" the Agent and let the MPC work in nominal conditions, without changing the desired footsteps. After a lot of training time and with a properly shaped reward function, the agent is able to understand that the optimal action to take in nominal conditions is close to 0, but since the action is sampled from a normal distribution we cannot rely on this. It is thus preferable to not use the Agent at all when in nominal conditions.

5.1 External Forces Detection

The method proposed here uses a residual signal to detect external forces and use the action computed by the Agent only when necessary. The method is inspired by [11]. Starting from the LIP dynamics we know that

$$\ddot{p}_c = \begin{bmatrix} \ddot{x}_c \\ \ddot{y}_x \\ \ddot{z}_c \end{bmatrix} = \begin{bmatrix} \eta^2(x_c - x_z) \\ \eta^2(y_c - y_z) \\ \eta^2(z_c - z_z) - g \end{bmatrix} \tag{19}$$

Where $\eta = \sqrt{\frac{g}{h}}$, with h the assumed constant height of the inverted pendulum and $p_z = \begin{bmatrix} x_z & y_z & z_z \end{bmatrix}^T$ the Zero Moment Point (ZMP) position.

It is possible to derive an observer for the energy of the system that detects any unexpected changes coming from external factors like external forces or floor slopes such that the the gravity vector also has non-null components along the x and y axes. To do this, we find a suitable candidate energy function for the LIP system. We use the kinetic energy as an energy function, as the potential energy should not change since we assume constant height h:

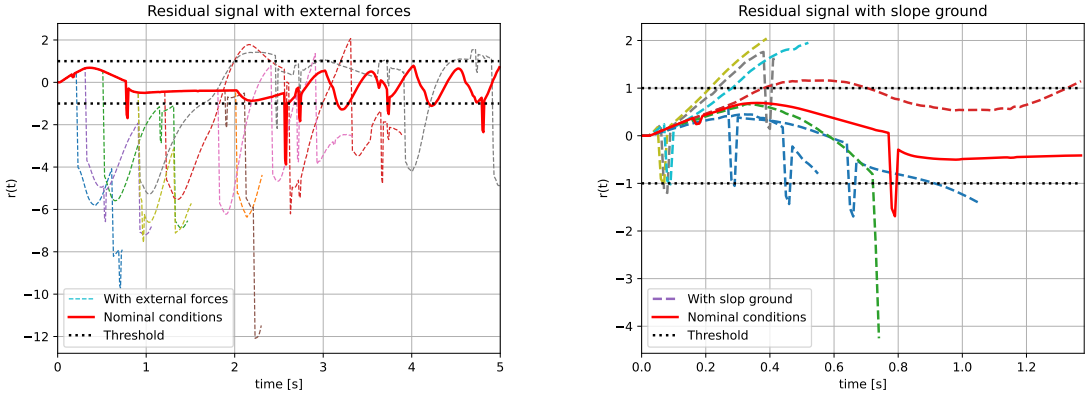
$$E(t) = K(t) + U(t) + E_F(t) = \frac{1}{2\eta^2}\dot{p}^T\dot{p} \tag{20}$$

where the kinetic energy $K(t)$ is scaled by η^2. The effect of $U(t)$ and $E_F(t)$ neglected in (20) should resolve in a change of the CoM velocity and for this create an error between the real kinetic energy of the system and the estimated once and the residual signal will rise.

Computing the time derivative of the kinetic energy of the system using (19) we obtain:

$$\dot{E}(t) = \frac{1}{\eta^2}\dot{p}_c^T\ddot{p}_c = \dot{p}_c^T(p_c - p_z) - \frac{g}{\eta^2}y_c \tag{21}$$

At this point, the residual signal can be defined as:



(a) The behaviour of the residual signal $r(t)$ in presence of external forces.

(b) The behaviour of the residual signal $r(t)$ in presence of floor slope variation.

Figure 18: As can be seen the residual signal is a viable option for monitoring the environment state using a suitable threshold. The spikes presents in the nominal dynamics are prevalently because of the changing contact points in the motion of the robot and can be easily be filtered out.

$$r(t) = G\left(E(t) - \int_0^t \left(\dot{p}_c^T(p_c - p_z) - \frac{g}{\eta^2}y_c + r\right)d\tau - E(0)\right) \tag{22}$$

The dynamics of the residual signal is thus that of a first order low-pass filter with gain G:

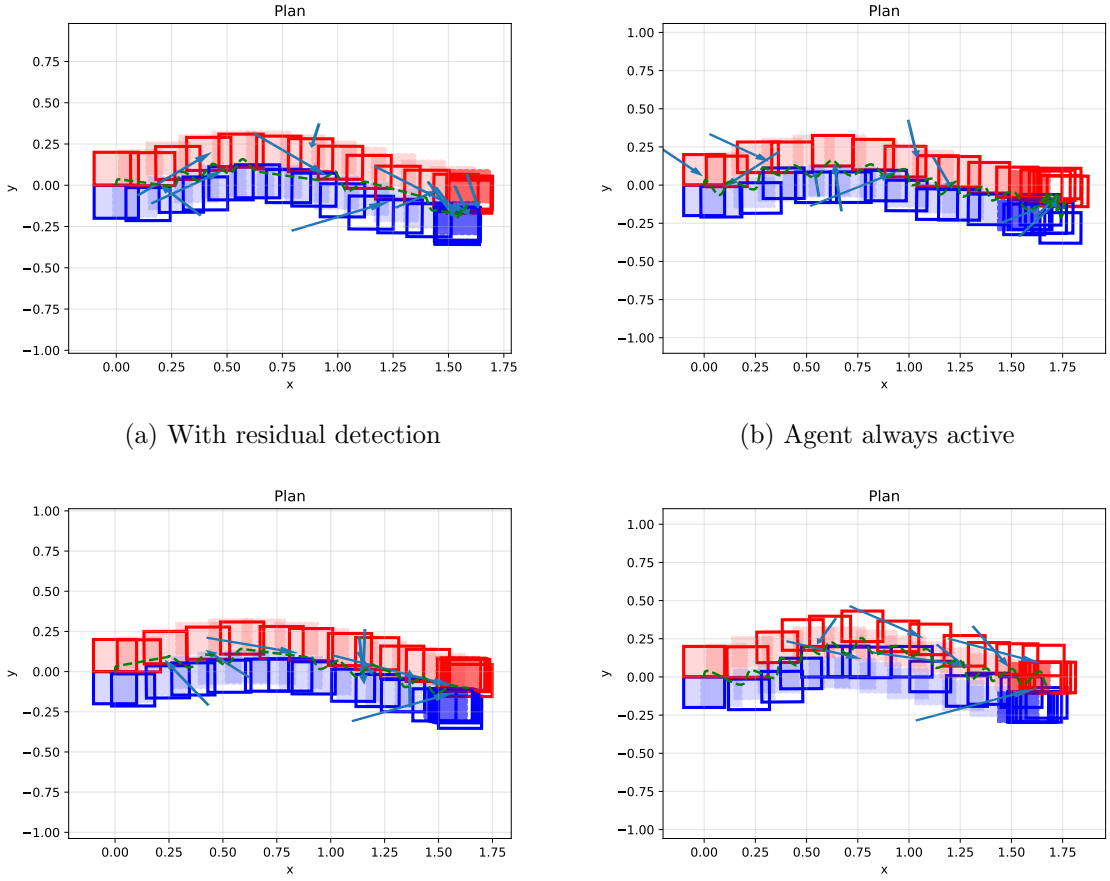
$$\dot{r}(t) = G\left(\dot{E}(t) - \dot{p}_c^T(p_c - p_z) + \frac{g}{\eta^2}y_c - r(t)\right) = G\left(n(t) - r(t)\right) \tag{23}$$

Recall that the term $n(t)$ is the general difference between the ideal kinetic energy computed using the simplified system equation and the real energy of the system, which is indirectly influenced by change of gravity and external forces. Using the Laplace transform to analyse the dynamics we obtain:

$$\mathcal{L}\{\dot{r}(t)\} = sr(s) = G(n(s) - r(s)) \Rightarrow r(s) = \frac{G}{G+s}n(s) \tag{24}$$

The residual accumulates the contribution of all unmodeled terms in (20) that modify the velocity of the CoM, like external forces or floor slopes. By defining a suitable threshold, this residual signal can be used to monitor the environment and detect anomalies in order to "turn-on" the Agent when the model strays too far from nominal conditions.

The observer gain G determines the speed of convergence of the observer: a low value for G entails a slow convergence of the dynamics, and the Agent will not be "turned on" in time to react to disturbances. On the other hand, a too high value for G leads to poor robustness and numerical instability.



(a) With residual detection

(b) Agent always active

(c) With residual detection

(d) Agent always active

Figure 19: Simulations without ground slope but with external forces. The difference of using whatsits the Agent or only when the residual detect a forces is clear: in case (a) and (c), with the residual detection, the desired plan is replicated almost perfectly. In case (b) the agent drive the robot to overshoot the final target and in (d) the agent drift from the desired plan.

For better numerical precision the online numerical integration can be performed using a bilinear method like Tustin integration:

$$\int_{t_{k-1}}^{t_k} f(\tau)d\tau \approx \frac{\Delta t}{2}\Big(f(t_{k-1}) + f(t_k)\Big) \tag{25}$$

Where $\Delta t = t_k - t_{k-1}$. This integration method decreases the numerical errors with respect to a simpler forward Euler integration method but still result in a simple formula that can be easily computed online.

5.2 Hybrid Gaussian-Bernoullian policy

Another approach to the problem consists into delegating to the agent learning's process to decide when it is conveniently to act and when not, independently of the presence of perturbations. This entrusts to the agent two fundamentally different types of decisions: first whether a modification should be applied, and second how much the footstep should be displaced if the agent is effectively acting. These two aspects correspond to discrete and continuous decision spaces, respectively. An hybrid Gaussian-Bernoullian policy can provide a probabilistic framework to model this structure, very similar to the one treated in Section 3.3.2 about the trainable scaling parameter ξ. In both cases the policy has to be extended to one with parametrized action space. The key difference is that, while for ξ predictions a fully continuous Gaussian policy is enough, a discrete component now is needed to represent the purely binary decision variable of whether to act or not.

The Bernoulli component models the probability of applying a correction. Formally, given state observation s, the policy outputs a parameter $p_\theta(s) \in [0, 1]$ defining a Bernoulli distribution:

$$b \sim \text{Bernoulli}(p_\theta(s)) \tag{26}$$

where b determines whether the footstep displacement action is activated. This machinery enforces the original agent's acting interpretation as a "useful correction", since it is encouraged to apply displacements to the nominal plan ideally only if some benefit occurs.

Conditioned on this decision, in evaluation phase (so not during training), the Gaussian component generates the displacement vector $\mathbf{\Delta} \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$ if and only if $p_\theta(s) > 0.5$.

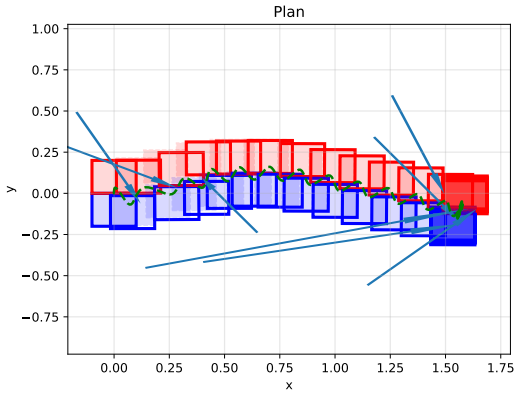


Figure 20: Performance of the hybrid policy. Note the good tracking of the original plan w.r.t. a standard policy one (e.g. see Figure 19)

It is very interesting to note that a trained hybrid Gaussian-Bernoullian policy assumes the role of a Neural Residual Signal: as we can see in Figure 21, the probability of acting, modulated by the parameter $p_\theta(s)$, exhibits spikes when an external forces is applied to the robot, behaving like a force detector and switching on the footstep displacement component of the RL agent in order to recover dynamical balance. On the other hand, if no external forces are applied, the Bernoullian component of the policy decides to not make the Gaussian one act.

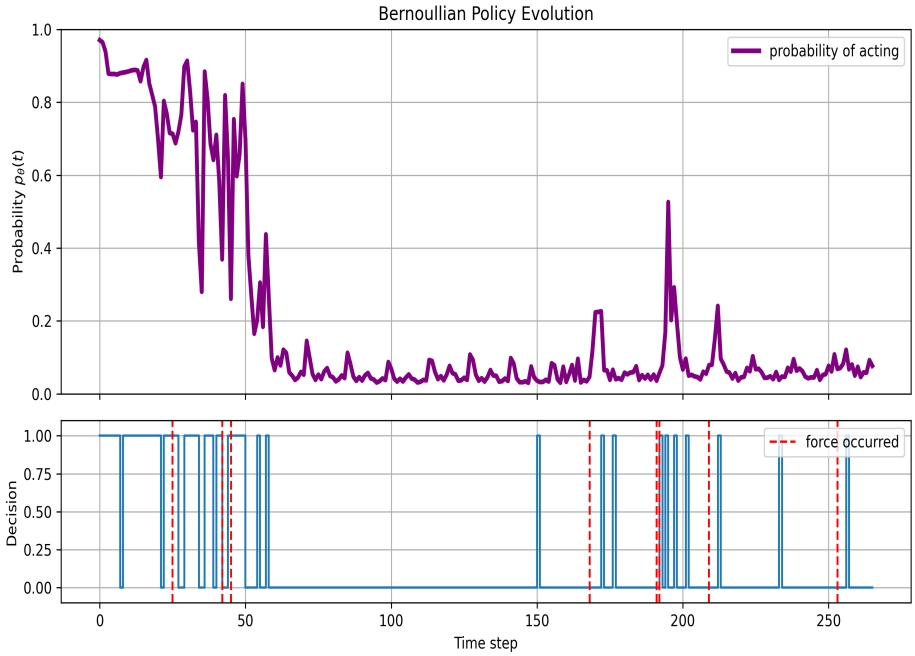


Figure 21: Behavior of the hybrid policy. Note the ability of switching on the agent if an external force is applied.

Finally, we compare the performance of our RL Augmented MPC with the nominal IS-MPC varying the ground slope. For each value (in degree) we performed 3 simulations with both controllers. The considered reference path is the linear one. The continuous red line forms a polygon representing the maximum slope in the four cardinal directions $[+X, -X, +Y, -Y]$ at which the IS-MPC controller is able to perform the task successfully in the all 3 simulations. The red dotted line circumscribes the values within which at least one simulation is successfully completed.

Same holds for the RL Augmented MPC, whose results are the blue ones. It is evident how the system is strengthened with respect to uncertainties in the model regarding the terrain's slope. The tested models are respectively: an hybrid policy with fixed scaling of actions (90%), and a standard policy with the same fixed scaling factor but with the residual's machinery enabled.

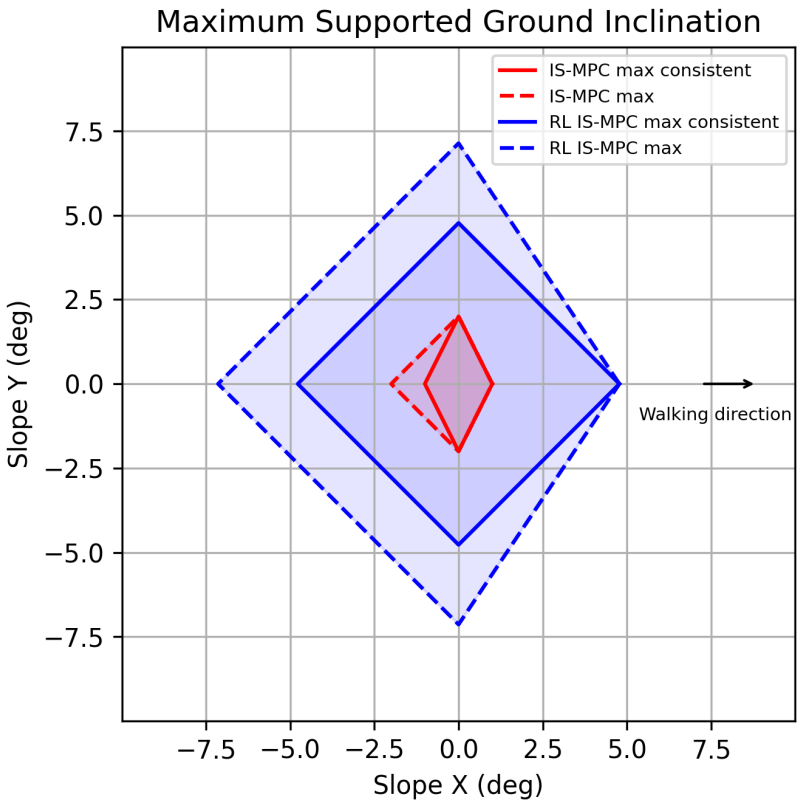


Figure 22: Maximum supported slope comparison using RL-Agents with hybrid policy (blue) vs standard IS-MPC (red)

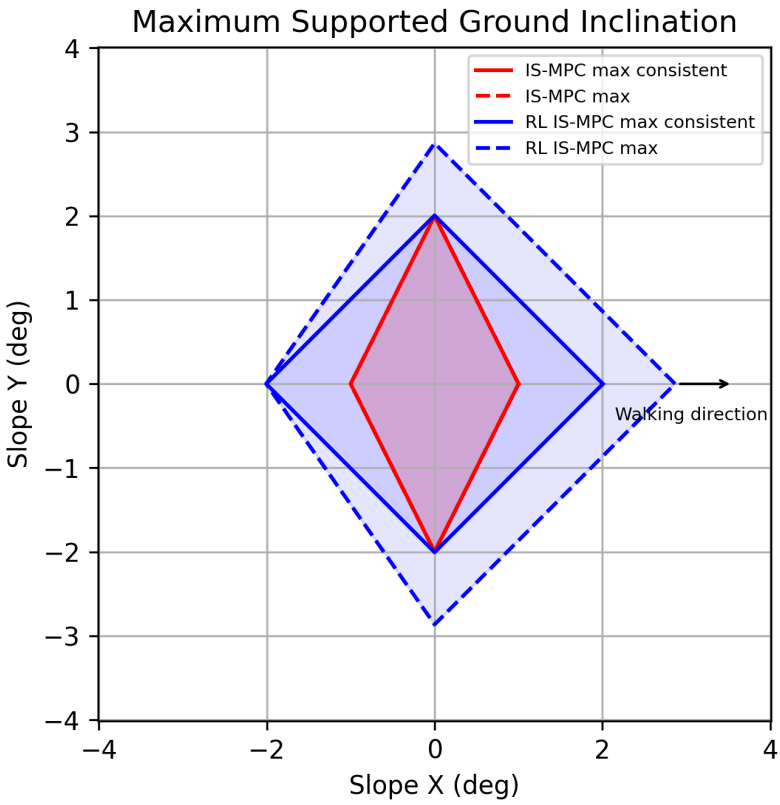


Figure 23: Maximum supported slope comparison using RL-Agents approach 3 with scale 0.9 (blue) vs standard IS-MPC (red)

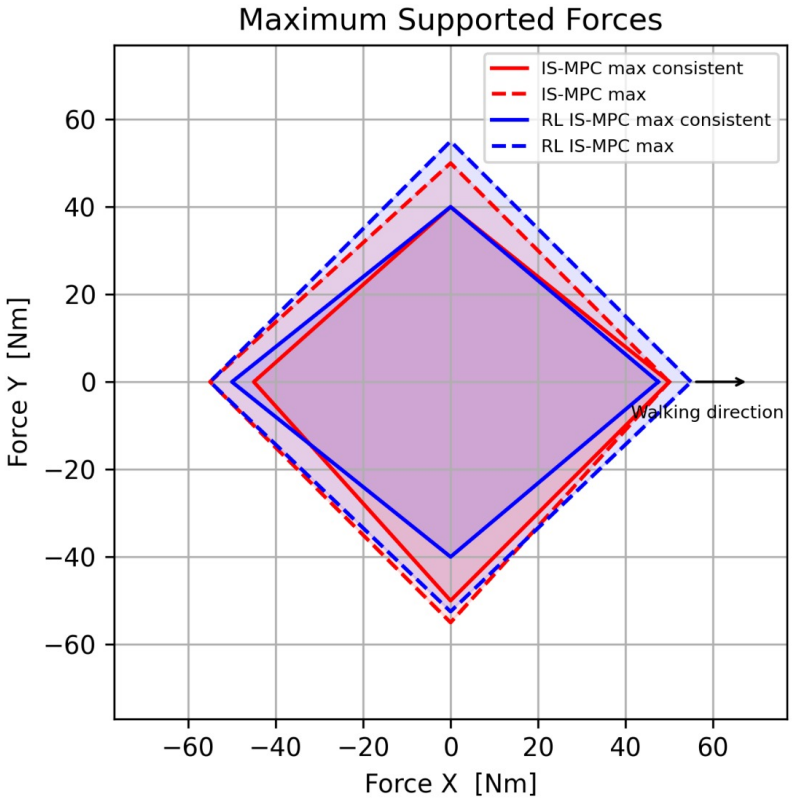


Figure 24: Maximum Forces comparison using RL-Agents approach 3 with scale 0.9 (blue) vs standard IS-MPC (red). The majority of forces without slope variation are absorbed by the inverse dynamics.

6 Conclusions

In this work we explored augmenting a typical MPC-based control scheme for humanoid robot walking with a residual policy based implemented with an Agent trained via Reinforcement Learning, and in particular the PPO algorithm. The MPC formulation is based on the Linear Inverted Pendulum and uses the ZMP velocity as a control input to maintain CoM trajectories bounded, satisfying the stability constraints in [1][2] and maintaining the ZMP within the support polygon defined by the robot's feet.

The plan is initially generated using a unicycle as a template model and is modified by the agent to keep balance despite external forces and terrain inclination. We explored three different approaches to modify the plan: displacing only the next footstep, displacing the whole plan, and fully displacing the next footstep, but gradually scale the displacement to the successive ones.

Displacing the next footstep yields the best tracking of the original plan among the three approaches, but often fails to maintain balance in the presence of external forces. In such a case, multiple footsteps have to be modified to properly maintain balance and, the displaced plan might result in constraints incompatible with the stability constraint of the IS-MPC formulation, which means that the robot is falling. This means that, with this approach, the success of the agent's current actions also depends on its future actions.

Displacing the whole plan avoids this problem. Assuming the reference plan is always feasible, the agent only needs to learn a feasible displacement, and the displaced plan will automatically satisfy the stability constraints: the agent's success in displacing the current footstep does not depend on future ones but in general have worst tracking capability.

A compromise between the two is presented by the last approach. Gradually scaling the displacement does not ensure future footsteps will be feasible but is a reasonable approximation, and it encourages the agent to converge back to the reference plan after displacement. In general, we found this to be the best approach.

The three approaches are implemented by three different agents, trained separately. Differences between the three agents, and in particular different generalization capabilities, can be attribute both to the approaches themselves (which play a major role), and to the different training sessions, which are intrinsically stochastic.

While the structure of our policy is based on that of [3], their approach uses a MPC formulation whose outputs are footsteps, similar to [1]. In their case, both the agent and the MPC work on the same input and output data, which are the desired and actual footsteps respectively. Their agent effectively captures the aspects of the full dynamics of the robot which are not captured in the ALIP model. In our implementation, which is based on that of [2], the MPC controls the ZMP-CoM dynamics, and thus the agent needs not only to learn the nonlinear dynamics of

the footstep, but also the relationship between CoM, ZMP, and the footstep plan. Furthermore, in our implementation the timings of each footstep and each phase of the gait are fixed, while they can vary in [3]. This is useful to keep balance under external forces and inclined terrain. These factors ultimately lead to the poor generalization capabilities of our policies, except when the action space is effectively made smaller by adding saturations to the displacements, which prevents most actions that would result in an unfeasible plan or in feet collision to be taken in the first place.

Finally, we provided two different methods for regulating the agent's intervention. Firstly, an energy-based residual signal has been implemented with the aim of detecting the presence of external disturbances such as contact forces or ground slope: if these perturbations are detected the RL agent is invoked to act, displacing footstep to recover balance. This approach of switching on and off the agent is implemented downstream of the RL training and is based on monitoring the energy of the LIP dynamics: if the residual signal exceeds a certain threshold, some external disturbance are acting on the system and the agent is required to act. If the residual is below the safety threshold instead, the agent is disabled to prevent it from perturbing the already stable performance of IS-MPC.

Secondly, a probabilistic framework is implemented as an Hybrid Gaussian-Bernoullian policy in order to have the agent itself learn whether to act or not during training. The emergent behaviours show that the agent did learn to act (or to predict an increased probability of acting) only in the case of contact with forces, in order to better tracking the reference plan in nominal conditions and correct the footstep placement if the dynamical balance has been compromised by an external impact. This behaviour leads to interpreting of the hybrid policy as a Neural Residual, constituting, together with the energy-based approach, a valid machinery that makes possible to physically apply our RL augmented control scheme to a humanoid robot.

References

[1] Nicola Scianca, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. Mpc for humanoid gait generation: Stability and feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1188, 2020.

[2] Michele Cipriano, Paolo Ferrari, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. Humanoid motion generation in a world of stairs. *Robotics and Autonomous Systems*, 168:104495, 2023.

[3] Seung Hyeon Bang, Carlos Arribalzaga Jové, and Luis Sentis. Rl-augmented mpc framework for agile and robust bipedal footstep locomotion planning and control. In *2024 IEEE-RAS 23rd International Conference on Humanoid Robots (Humanoids)*, pages 607–614. IEEE, 2024.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] DIAG-Robotics-Lab. ismpc. `https://github.com/DIAG-Robotics-Lab/ismpc`, 2023. Original implementation of the DIAG-Robotics-Lab made in python.

[6] DLR-RM. stable-baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2025. Stable-Baselines3 Python library.

[7] Farama-Foundation. Gymnasium. `https://github.com/Farama-Foundation/Gymnasium`, 2025. Gymnasium Python library.

[8] Emanuele Coletta, Giulio Sartori, and Gianluca Sperduto. uni-amr-project. `https://github.com/arch-DEMINION/uni-amr-project`, 2026. Python implementation of this project.

[9] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space, 2024.

[10] Matthew J Powell and Aaron D Ames. Mechanics-based control of underactuated 3d robotic walking: Dynamic gait generation under torque constraints. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 555–560. IEEE, 2016.

[11] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics*, 33(6):1292–1312, 2017.