SAPIENZA
UNIVERSITÀ DI ROMA

DIPARTIMENTO DI INGEGNERIA, AUTOMATICA E GESTIONALE

Underactuated Robots

Modular Passive Tracking for stack of tasks applying on the Whole-body Controller of Humanoid robot

Students:
Alessandro Angelo Anzellini
Elisa Martinelli
Daniele Teni
An Nguyen

Academic Year 2024/2025

Contents

1 Introduction

1.1 The Objective

Robust and flexible control of robotic systems is a cornerstone of modern robotics research. The ability to achieve multiple, simultaneous objectives—such as end-effector positioning, obstacle avoidance, and dynamic stability—requires advanced control strategies capable of handling task prioritization and conflict resolution. This need is particularly evident in humanoid robotics and complex multi-contact systems, where interactions with the environment are highly dynamic and often unpredictable.

The paper [1] introduces a novel approach to address these challenges. MPTC is designed to independently regulate multiple sub-task objectives within a single cohesive control framework. Unlike traditional control methods that rely on strict task hierarchies, MPTC employs a passivity-based strategy to blend tasks smoothly, ensuring system stability even in overdetermined and conflicting scenarios.

The primary objective of this research is to demonstrate how MPTC can improve the execution of multi-objective tasks in robotic control by leveraging the Stack-of-Tasks (SoT) architecture. This architecture allows for modular control, where different sub-tasks such as balance, manipulation, and locomotion are independently optimized and prioritized. MPTC's strength lies in its ability to ensure passivity at each task level, thus preventing instability that could arise from modeling errors or external perturbations.

Additionally, the research explores how MPTC handles underactuated systems and dynamic constraints through Quadratic Programming (QP) optimization. This allows for real-time adjustments to task execution without compromising stability or safety, which is critical in real-world applications involving humanoid robots and multi-limbed manipulators.

1.2 Experiments

The model was simulated using a Kawada HRP-4 Robot with DART running the MPTC at 100Hz.

3 different setups were tested: Normal Walk, Walking affected by an external push (applied on different points), and a Normal Walk on a non-flat environment.

Every test has shown, under different aspects, the robustness and capability to reject disturbances in real-world settings of MPTC compared to a regular inverse dynamics method.

2 Task Space Robot Dynamic

2.1 Definition of the Model

The first step is to define a model and describe the complete dynamics of the robot. In the joint space, the dynamics is:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + \tau_g(q) = \tau, \tag{1}$$

where $q \in \mathbb{R}^n$ represents the generalized coordinates, $M(q)$ is the inertia matrix, $C(q,\dot{q})$ is the Coriolis and centrifugal matrix, $\tau_g(q)$ is the gravitational torque, and τ represents the generalized forces applied.

It is also defined τ as follows:

$$\tau = S(\tau_j + \tau_{int}) + L_{all}^{\top} w_{all}, \tag{2}$$

where S is a selection matrix, having a form of $S = [0_{n_{act}\times n-n_{act}}, I_{n_{act}\times n_{act}}]^T$, $\tau_j \in \mathbb{R}^{n_{act}}$ represents n_{act} dimension actuated joint torque vector, τ_{int} is the vector contains any disturbance torque (e.g friction), w_{all} is the vector of all the external wrenches applied to the links of the robot while $L_{all} = [L_1^T ... L_{n_L}^T]^T$ is the vector containing all the Jacobians of the corresponding links, used to map back w_{all} to τ.

2.2 Definition of the Task Space of the Model

Task space velocities and accelerations are defined through the task Jacobian J_k as follows:

$$\dot{x}_k = J_k\dot{q}, \quad \ddot{x}_k = J_k\ddot{q} + \dot{J}_k\dot{q}. \tag{3}$$

Solving the model equation for the generalized accelerations $\ddot{q}$:

$$\ddot{q} = M^{-1}(\tau - C\dot{q} - \tau_g) \tag{4}$$

The task space acceleration can now be derived as:

$$\ddot{x} = \dot{J}_k\dot{q} + J_kM^{-1}(\tau - \tau_j - C\dot{q}) = (\dot{J}_k - J_kM^{-1}C)\dot{q} + J_kM^{-1}(\tau - \tau_g) \tag{5}$$

For the tracking control error law, denote the velocity error:

$$\dot{\tilde{x}}_k = \dot{x}_{k,ref} - J_k\dot{q} \tag{6}$$

While the acceleration error is:

$$\ddot{\tilde{x}}_k = \ddot{x}_{k,ref} - (J_kM^{-1}(\tau - \tau_g) - (J_kM^{-1}C - \dot{J}_k)\dot{q} \tag{7}$$

Where we can directly define Q_k as:

$$Q_k = (J_k M^{-1} C - \dot{J}_k) \tag{8}$$

and from now on directly using it inside the equation. The task space inertia matrix for the given task k is defined as:

$$M_k = \left(J_k M^{-1} J_k^T \right)^{-1} \tag{9}$$

This expression projects the joint-space inertia into the task space, and the result is a symmetric positive-definite matrix of dimension $k \times k$.

The Coriolis and centrifugal effects in the task space can be represented by the matrix C_k, which is given by:

$$C_k = M_k Q_k T_k^T \tag{10}$$

In this formulation, Q_k captures the Coriolis and centrifugal terms in joint space projected along the task directions, and T_k denotes the weighted pseudoinverse of the Jacobian, defined as:

$$T_k = M_k J_k M^{-1} \tag{11}$$

This matrix T_k ensures that the mapping from joint space to task space remains dynamically consistent, taking into account the non-uniform inertia of the system.

3 Modular Passive Tracking Controller

3.1 Formulation for one task

The Modular Passive Tracking Controller (MPTC) is generalized by stacking multiple subtask controllers, each defined independently. These are merged via a unified control law, forming an overall structure capable of dealing with multiple objectives simultaneously.

The formulation begins by selecting a Lyapunov function composed of energy-related terms. The goal is to derive the desired task force $f_{k,\text{des}}$ in such a way that the task dynamics exhibit passivity with respect to the task velocity error $\dot{\tilde{x}}_k$ as the system output and the task force error $\tilde{f}_k$ as the system input.

The chosen Lyapunov function is expressed as:

$$V_k = \underbrace{\frac{1}{2}\dot{\tilde{x}}_k^\top M_k \dot{\tilde{x}}_k}_{E_{\text{kin},k}} + \underbrace{\frac{1}{2}\tilde{x}_k^\top K_k \tilde{x}_k}_{E_{\text{pot},k}} \tag{12}$$

In this expression, K_k is a symmetric and positive definite stiffness matrix. The function V_k is positive definite with respect to both the task position error $\tilde{x}_k$ and the task velocity error $\dot{\tilde{x}}_k$.

To evaluate its behavior, the time derivative of V_k is computed, using Eq.12 and incorporating the acceleration error from Eq.7 :

$$\ddot{x}_k = \ddot{x}_{k,\text{ref}} - \underbrace{\left(J_k M^{-1}\left(\tau - \tau_g\right) - Q_k \dot{q}\right)}_{\ddot{\tilde{x}}_k} \tag{13}$$

$$\begin{aligned}\dot{V}_k &= \dot{\tilde{x}}_k^\top \left(M_k \ddot{\tilde{x}}_k + \frac{\dot{M}_k}{2}\dot{\tilde{x}}_k + K_k \tilde{x}_k\right) \\ &= \dot{\tilde{x}}_k^\top \left(M_k J_k M^{-1}(\tau_g - \tau) + M_k Q_k \dot{q} + M_k \ddot{x}_{k,\text{ref}} + C_k \dot{\tilde{x}}_k + K_k \tilde{x}_k\right)\end{aligned} \tag{14}$$

This derivation uses the equality:

$$\frac{\dot{M}_k}{2} = \frac{C_k^\top + C_k}{2} = \underbrace{\frac{C_k^\top - C_k}{2}}_{\text{skew-symmetric}} + C_k \tag{15}$$

The desired task force $f_{k,\text{des}} = T_k \tau_{k,des}$ is designed to cancel the terms within the bracketed expression, ensuring that $\dot{V}_k < 0$. This results in the desired task force [1]:

$$f_{k,\text{des}} = T_k \tau_g + M_k Q_k \dot{q} + M_k \ddot{x}_{k,\text{ref}} + (C_k + D_k)\dot{\tilde{x}}_k + K_k \tilde{x}_k \tag{16}$$

In real-world implementations, the actual task force f_k is typically used instead of the ideal $f_{k,\text{des}}$. The error between the two is given by:

$$\tilde{f}_k = f_{k,\text{des}} - f_k \quad \Rightarrow \quad f_k = f_{k,\text{des}} - \tilde{f}_k \tag{17}$$

Substituting this into the derivative of the Lyapunov function gives:

$$\dot{V}_k = -\dot{\tilde{x}}_k^\top D_k \dot{\tilde{x}}_k + \dot{\tilde{x}}_k^\top \tilde{f}_k = \dot{V}_{k,\text{des}} + \dot{\tilde{V}}_k < \dot{\tilde{x}}_k^\top \tilde{f}_k \tag{18}$$

This indicates that:

> The system at the task level maintains passivity with respect to the input $\tilde{f}_k$, the output $\dot{\tilde{x}}_k$, and the storage function V_k from equation (12). Although the desired rate $\dot{V}_{k,\text{des}}$ is strictly dissipative, as ensured by the positive definite matrix D_k, the actual rate $\dot{V}_k$ might not be zero. This deviation can result from factors such as unknown disturbances, limited actuation, underactuation, task inconsistencies, or prioritization mechanisms.

Considering these influences, the task dynamics at the acceleration level takes the form:

$$M_k \ddot{\tilde{x}}_k + (C_k + D_k)\dot{\tilde{x}}_k + K_k \tilde{x}_k = \tilde{f}_k \tag{19}$$

In this formulation, D_k and $K_k \succ 0$ are positive definite tuning parameters, while M_k and C_k represent internal dynamic properties of the system. The dynamics of each individual task exhibits a passive behavior with respect to the input task force error and the output task velocity error. This property holds regardless of actuation constraints or modelling inaccuracies and is a key feature that improves robustness in practical implementations.

3.2 Develop for a stack of tasks

To extend the modular formulation to multiple concurrent objectives, individual task controllers are aggregated into a unified structure. Each task retains its independent formulation, but contributes to a common control objective through task stacking. To handle multiple tasks simultaneously, all the desired task forces $f_{k,\text{des}}$ for $k \in \{1, \ldots, n_T\}$ are stacked into a single vector:

$$f_{\text{des}} = \begin{bmatrix} f_{1,\text{des}} \\ \vdots \\ f_{n_T,\text{des}} \end{bmatrix} \tag{20}$$

Similarly, the actual task forces are stacked as follows:

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_{n_T} \end{bmatrix} = T\tau, \quad T = \begin{bmatrix} T_1 \\ \vdots \\ T_{n_T} \end{bmatrix} \tag{21}$$

where T is the mapping from the generalized forces τ to the stack of actual task forces f, formed by vertically concatenating the task Jacobian mappings.

A new term, referred to as the *commanded task force* f_{cmd}, is introduced. It is computed via an optimization-based controller that minimizes the difference between the desired and commanded task forces. The desired task force expression must therefore be updated according to the solution of the optimization problem, resulting in:

$$f_{\text{cmd}} = \begin{bmatrix} f_{1,\text{cmd}} \\ \vdots \\ f_{n_T,\text{cmd}} \end{bmatrix} = T\tau_{\text{cmd}} = TUu_{\text{cmd}} \tag{22}$$

Here, u_{cmd} denotes the optimization variables, while U is the actuation mapping matrix that converts the optimization variables into generalized forces contributing to the system dynamics and will be explained later in sec 3.3.

The corresponding stack of task force command errors is defined as:

$$\tilde{f}_{\text{cmd}} = f_{\text{des}} - f_{\text{cmd}} = f_{\text{des}} - T\tau_{\text{cmd}} = f_{\text{des}} - TUu_{\text{cmd}} \tag{23}$$

In overdetermined or conflicting task setups, direct enforcement of all desired task forces is infeasible. To address this, a quadratic cost function is formulated that penalizes the deviation between the commanded and desired task forces. This allows soft prioritization and leads to a stable and feasible solution under constraints. Based on the error vector $\tilde{f}_{\text{cmd}}$, the following quadratic cost function is defined [1]:

$$G = \frac{1}{2}\tilde{f}_{\text{cmd}}^{\top} W \tilde{f}_{\text{cmd}} \tag{24}$$

which can be expanded as:

$$G = \frac{1}{2}u_{\text{cmd}}^{\top} T_u^{\top} W T_u u_{\text{cmd}} - f_{\text{des}}^{\top} W T_u u_{\text{cmd}} + \frac{1}{2} f_{\text{des}}^{\top} W f_{\text{des}} \tag{25}$$

with:

$$T_u = TU \tag{26}$$

The weighting matrix $W \in \mathbb{R}^{31\times 31}$ used in the cost function is defined as [1]:

$$W = \Lambda^{-1}\Psi \tag{27}$$

Here, Λ is a block-diagonal matrix composed of the task space inertia matrices $\{M_1, M_2, \ldots, M_{n_T}\}$ along its diagonal. Each matrix M_k corresponds to the inertia associated with task k.

The matrix Ψ is a diagonal matrix that collects scalar weighting factors for each task. It takes the form:

$$\Psi = \begin{bmatrix} \psi_1 I & & \\ & \ddots & \\ & & \psi_{n_T} I \end{bmatrix} \tag{28}$$

where ψ_k represents the relative importance of task k and I is the identity matrix of appropriate dimension. So, W is a positive definite weighting matrix used to scale the contribution of each task to the overall cost.

3.3 Applying to the under-actuated case of humanoid robots

Recalling the structure of the overall dynamic in the generalized coordinate (joint coordinate) as in Eq.1, in the case of humanoid robot, the external wrenches that are needed to maintain the robot's pose both in the standing and walking coming from the reaction wrenches that the ground applied on the robot's feet. Denotes U as the *actuation mapping matrix* [1] that maps the actuated joint torque and the ground reaction wrenches to the generalize torque in the dynamic equation, its components include the selection matrix and the feet's Jacobian matrix:

$$U = \begin{bmatrix} S & L_{EE}^T \end{bmatrix} = \begin{bmatrix} S & \Gamma_l . J_{lfoot}^T & \Gamma_r . J_{rfoot}^T \end{bmatrix}, \; u_{cmd} = \begin{bmatrix} \tau_{j,cmd}^T & w_{lfoot}^T & w_{rfoot}^T \end{bmatrix}$$

And thus, the variables that need to be optimized are composed in the vector u_{cmd}. Additional variables Γ_l, Γ_r are introduced to capture the contact status of the robot's feet during locomotion. $\Gamma_l = 1$ means that the left foot is in contact with the ground, $\Gamma_l = 0$ means that it is in the swing phase. The same concept is applied to the right foot. The values of these variables are taken from the given footstep planner.
Moreover, the optimization variable must be subject to suitable constraints to maintain balanced walking. In the case of the actuated joint's torque, their values must be bounded by the maximum torque that the joint's actuators can provide. In the case of contact wrenches, they are constrained in unilateral conditions and the friction cone structure.

4 Implementation and simulation

4.1 Implementation on HRP4 robot

As in [2], we implement the proposed whole-body controller on the Kawada HRP-4, a $151cm$ tall humanoid robot, with a weight of $40kg$ and 24 actuated joints.

Figure 1: HRP-4 humanoid robot

In particular, the code of gait and trajectory generation is taken from [3], using the Dynamic Animation and Robotics Toolkit (DART), which provides a physics-based environment for simulating robot locomotion [4]. Our main work was to modify the provided humanoid whole-body controller with the passivity-based one in [1].
The implementation is quite straightforward, and it follows what is already shown in [1], while the optimization part is solved using CasADi [5] with the linear solver osqp. For the implementation, we also relied on PINOCCHIO [6] in order to compute the Coriolis matrix.
In order to coordinate multiple sub-tasks and ensure desirable closed-loop behavior, the Modular Passive Tracking Controller (MPTC) framework introduces an optimization-based structure that unifies task-level controllers. These individual controllers are blended through a weighted aggregation of desired task forces, and the resulting global behavior is analyzed using Lyapunov theory. To maintain passivity even in the presence of conflicting or redundant tasks, specific formulations of the overall cost function and associated weight matrices are employed. These include task-space inertia considerations and scalar task weights, enabling consistent and robust control across a wide range of system configurations. The desired closed-loop dynamic gains, denoted by D_k and K_k, are designed as symmetric positive definite matrices, i.e., $D_k, K_k \succ 0$, and are elements of $\mathbb{R}^{31\times 31}$. The dimension 31 corresponds to the total size of the overall task space vector.

4.2 Simulation setup and result

The proposed approach employs a QP solver running at 100Hz to compute joint torques that enable the execution of a Center of Mass (CoM) trajectory generated by the

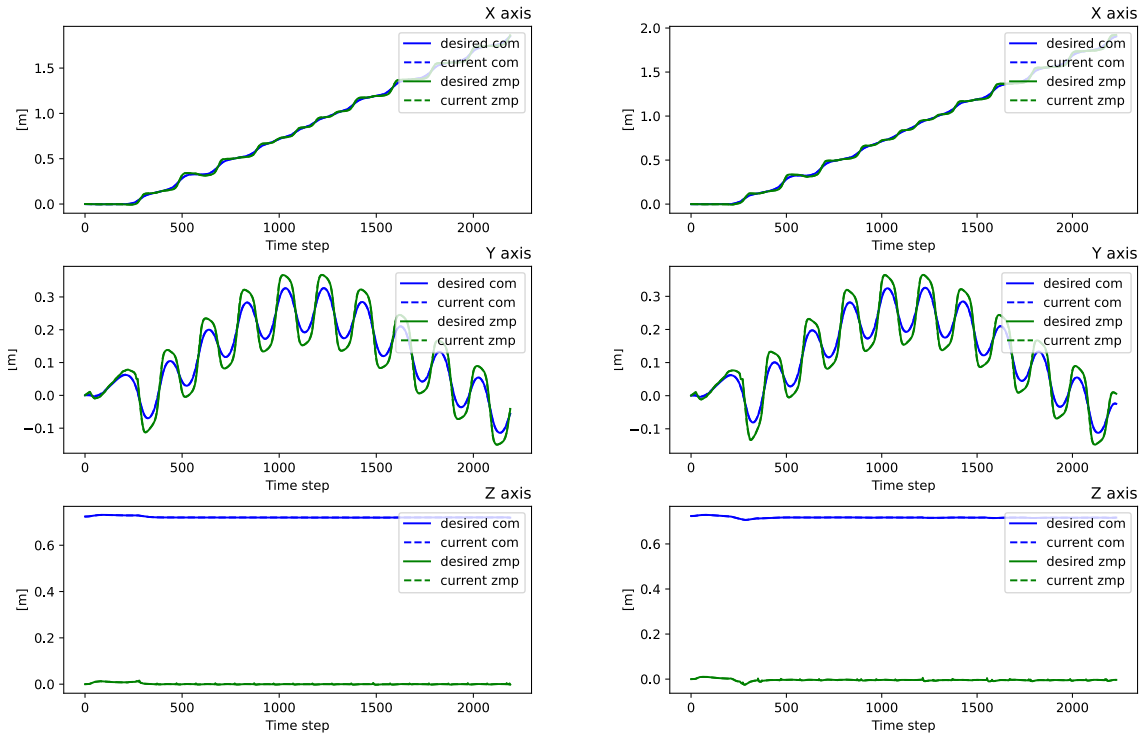
ISMPC [2], while ensuring consistency with the planned foot trajectories. To this end, we define six hierarchical tasks:

- Two 6-dimensional tasks to control the position and orientation of each foot
- A 3-dimensional task to track the desired CoM position
- Two 3-dimensional tasks to regulate the desired orientation of the torso and the base
- A 10-th dimensional task for deal with the nine redundant joint

Then the procedure follows as shown in 3

4.2.1 Normal walk

In this experiment setup, we consider the robot walking on a flat ground, the trajectory planner is generated with high-level velocity inputs for the virtual unicycle, and the robot manages to deal with all the possible feasible trajectories that we give by modifying the velocity inputs



(a) Proposed MPTC

(b) Inverse dynamics

Figure 2: COM and ZMP trajectory for normal walk

In particular, as shown in figure 5, for normal walking, the CoM position responses from the two whole-body controllers do not differ a lot, and also the computational time is similar.

4.2.2 Walking with external push

4.2.2.1 Push applied on the foot In the scenario from Fig.5a, the robot is subjected to an external horizontal force of $-4N$ along the x axes, applied to one of its feet during the swing phase, lasting for $0.30s$. With our control implementation, the robot is able to maintain stable locomotion despite the disturbance. In contrast, when using the baseline inverse dynamics controller, the system is unable to tolerate forces greater than approximately $2.8N$

4.2.2.2 Push applied on the torso or base The robot is subject to an external push acting on his torso or its base, with different trials to prove that the robot can sustain forces up to $10N$ acting on x and y axis even at the same time (meaning $F = [10, 10, 0]'$). It is important to note that the following results are obtained **without any fine-tuning**, in fact, as shown in Figure 3, all tasks have the same parameters, except for the two tasks of the foot position and orientation.

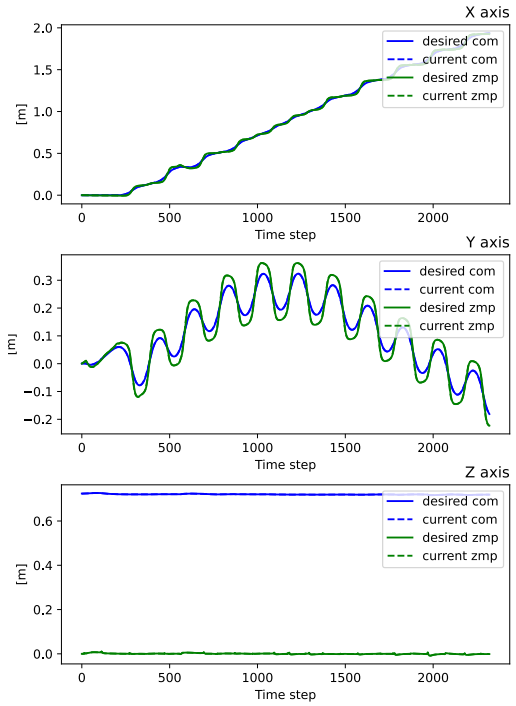
```
# weights and gains
tasks = ['lfoot', 'rfoot', 'com', 'torso', 'base', 'joints','knee_l','knee_r']
weights   = {'lfoot':  1., 'rfoot':  1., 'com':  1., 'torso': 1., 'base': 1., 'joints': 1.,'knee_l':0,'knee_r':0}
pos_gains = {'lfoot': 10., 'rfoot': 10., 'com':  1., 'torso': 1., 'base': 1., 'joints': 1.,'knee_l':2,'knee_r':1 }
vel_gains = {'lfoot': 10., 'rfoot': 10., 'com': 2., 'torso': 2., 'base': 2., 'joints': 2.,'knee_l':2,'knee_r':2}
```

Figure 3: Weight and gains used for built matrix D_K,K_K and Ψ

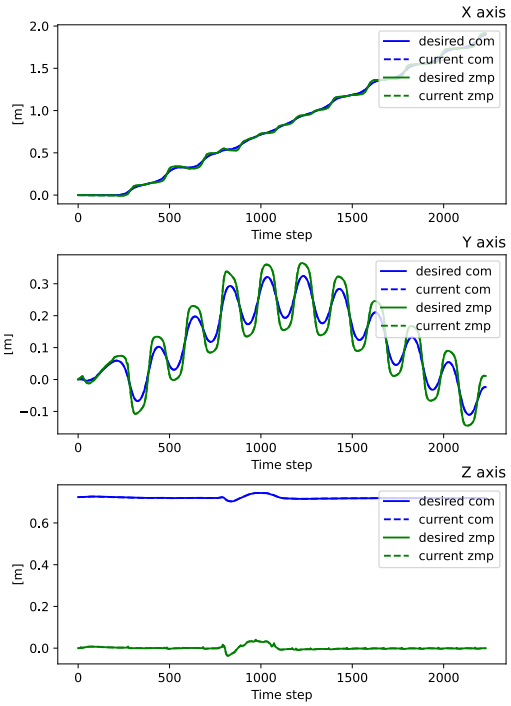
It is also remarkable to notice that for a feasible walk, the robot could assume two different poses depending on the joint knee value, as shown in Fig.4. In particular, by

Figure 4: Feasible walking with different joint angles of the knees

experiment we observe that if the robot is pushed on the torso with a force over $10N$ along x-axis (in the body frame), in order to keep stability, the knee joints assume the second configuration, and it does not change during walking. To solve this issue and correct the robot pose, in order to achieve normal walking, we add two one-dimensional tasks for the knee joints (left and right) that become active only when their values

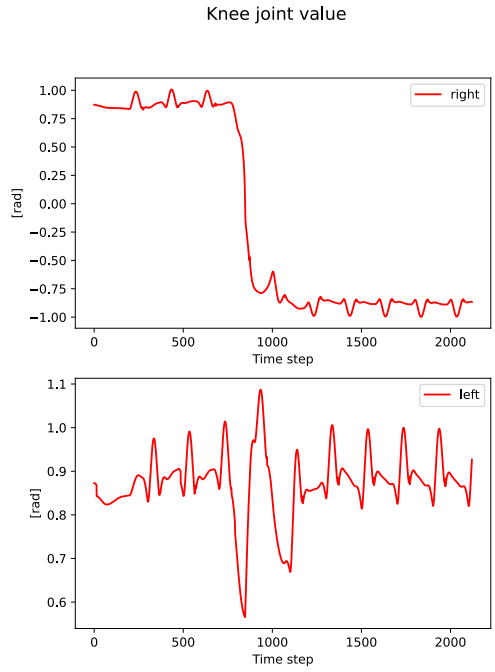


(a) Push on right foot after 5.2 seconds

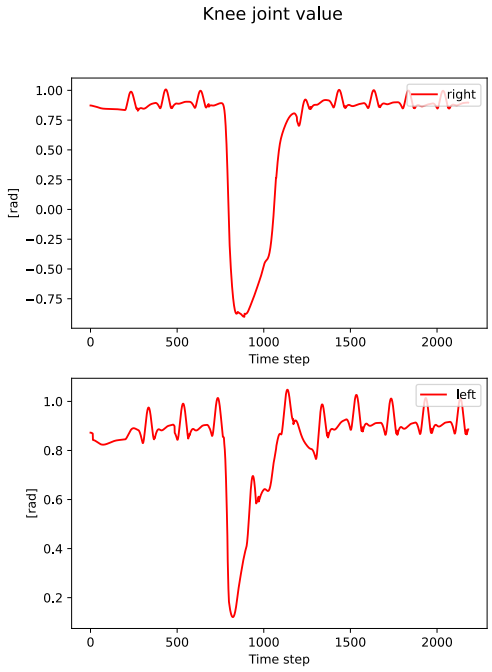


(b) Push on torso after 7.2 seconds for 0.3s of [10,10,0]N

Figure 5: Push perturbations on foot and torso



(a) without additional task



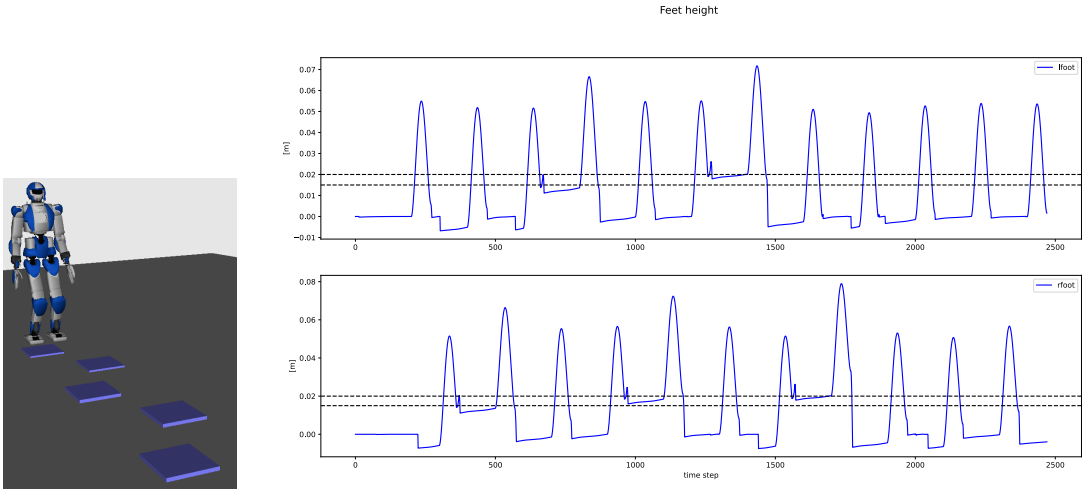
(b) with additional task

Figure 6: Value assumed by the knee's joints

start to change the robot's configuration Fig.6b. We repeated this experiment with the baseline controller, but even with fine-tuning, the robot still did not manage to walk with an external force up to $5N$.

4.2.3 Non flat environment

As a final experiment, the robot was tested on uneven terrain, as in Fig.7a, where obstacles with a maximum height of $2cm$ were placed along its path. The robot is capable of executing the trajectory, with the feet height recorded in the Fig.7b. However, due to the high disturbances, the gain of the desired task dynamics required fine-tuning to guarantee the feasible locomotion.



(a) Uneven terrain

(b) Feet height during locomotion

Figure 7: Walking on uneven terrain experiment

4.3 Comparison

As shown in the previous experiments, the proposed approach proves to be more robust than the baseline inverse dynamics controller, managing to withstand nearly twice the external force. It is also worth noting that, except for the final experiment on uneven terrain, all other scenarios required low effort on fine-tuning of the control gains. However, it should be noted that the failures during simulation (especially in the uneven terrain setup) are primarily due to the gait generation algorithm used in the MPC. This suggests that employing a different gait generation strategy could further improve the robot's robustness.

5 Conclusions

In this project, we have studied the whole-body controller for humanoid robots using the MPTC framework. We have also successfully set up the simulation experiments to validate the robustness of this framework under the disturbances influenced by external pushes and uneven terrain. In addition, by leveraging the advantage of the modular stack of task setup, a task of correcting the configurations of the knees has been added to maintain a proper walking pose of the robot after its reaction against the external force. Some concerns during the simulation process are listed below:

- The values of contact status variables Γ_l, Γ_r are fixed in advance by the footstep planner. Thus, during the experiments on uneven terrain, the higher the obstacles, the worse the accuracy of contact status captured by these variables. This greatly affects the structure of the matrix U in the cost function.
- During the gain tuning of MPTC, the simulator sometimes collapses and reports failures in the gait generation MPC. This is still an open question for us.

References

[1] Johannes Englsberger, Alexander Dietrich, George-Adrian Mesesan, Gianluca Garofalo, Christian Ott, and Alin Olimpiu Albu-Schäffer. Mptc - modular passive tracking controller for stack of tasks based control frameworks. In *16th Robotics: Science and Systems, RSS 2020*, 2020.

[2] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo. Mpc for humanoid gait generation: Stability and feasibility. In *16th Robotics: Science and Systems, RSS*, 2020.

[3] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo. Ispc. In *ismpc*, 2020.

[4] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

[5] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[6] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *International Symposium on System Integration (SII)*, 2019.

[7] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.