Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid

Fabio Scaparro Dept. of Computer,
Control and Management Engineering
Antonio Ruberti

Patrizio Perugini Dept. of Computer,
Control and Management Engineering
Antonio Ruberti

Mattia Castelmare Dept. of Computer,
Control and Management Engineering
Antonio Ruberti

Contents

1 Introduction **3**

2 Problem formulation **3**

3 Constraints **4**
3.1 Rigid Body Dynamics 5
3.2 Contact constraint 5
3.3 ZMP 6
3.4 Friction cone 6
3.5 Momentum Control 7
3.6 PD control on posture 9
3.7 Regularizer on GRFs 9

4 Simulations and results **9**
4.1 PD on posture 9
4.2 Tracking of the CoM 10
4.3 External push 11
4.3.1 Weak push 11
4.3.2 Strong push without ZMP constraint 12
4.3.3 Strong push with ZMP constraint 12

5 Conclusions **14**

1 Introduction

In the context of controlling legged robots that interact with the environment, the choice of control strategy is crucial. Indeed, it is necessary to adopt an algorithm that can generate precise but compliant movements while optimizing interactions with the environment.
For this reason the choice fell on a controller based on the *full dynamic model* of the robot that is suitable for very dynamic motion. This choice can lead to better performance in the presence of external disturbances (such as forces that push the robot so that it loses its balance).
The main task of this work is to develop a hierarchical inverse dynamic controller for balance stabilization of a torque-controller humanoid robot.
Using the full dynamic of the robot allows to control hierarchies of tasks, for example is useful to ensure that a balancing task will precede a task with lower importance (such as keeping the arms still). Some approaches take in account pseudo-inverse techniques but they are limited because they cannot properly handle inequality constraints (such as friction cone constraints).
The method proposed solves the problem of dealing with inequalities by formulating a series of optimization problems that are phrased as cascade of quadratic programs (QPs).

2 Problem formulation

The constraints and tasks considered for the hierarchical inverse dynamics approach are written in the form of affine functions of joint and body accelerations, joint torques and contact forces in order to formulate the control problem as a series of quadratic programs [1]. They constitute the variables that will be optimized by the controller.
At every control cycle the constraints for physical consistency and our control objectives are all expressed as affine equations of the variables $\ddot{\boldsymbol{q}}, \boldsymbol{\lambda}, \boldsymbol{\tau}$. They are divided into different priorities, with highest priority in the hierarchy given to physical consistency (dynamic equation of the robot) while in the lowest one there are expressed balancing and motion tracking tasks.
The control objectives constraints are expressed in the form:

$$\boldsymbol{A}\boldsymbol{y} + \boldsymbol{a} \leq \boldsymbol{0} \tag{1}$$

$$\boldsymbol{B}\boldsymbol{y} + \boldsymbol{b} = \boldsymbol{0} \tag{2}$$

where $\boldsymbol{y} = [\ddot{\boldsymbol{q}}^T, \boldsymbol{\lambda}^T, \boldsymbol{\tau}^T]^T$, $\mathbf{A} \in \mathbb{R}^{m \times (2n+6+6c)}$, $\mathbf{a} \in \mathbb{R}^m$, $\mathbf{B} \in \mathbb{R}^k$ and $m, k \in \mathbb{N}$ the overall task dimensions and $n \in \mathbb{N}$ the number of robot DoFs, $c \in \mathbb{N}$ is the number of constrained end-effectors. The goal of the controller is to find the values of $\ddot{\boldsymbol{q}}, \boldsymbol{\lambda}, \boldsymbol{\tau}$ that satisfies all these objectives as well as possible.
In order to guarantee prioritization we solve a cascade of QPs, in which a QP with constraints imposed by lower priority tasks are optimized over the set of optimal solution of higher priority tasks.

Considering one solution for the QP characterized of priority r all remaining optimal solutions $\mathbf{y}$ are expressed by equations:

$$\boldsymbol{y} = \boldsymbol{y}_r^* + \boldsymbol{Z}_r \boldsymbol{u}_{r+1}. \tag{3}$$

$$\begin{aligned} \boldsymbol{A}_r \boldsymbol{y} + \boldsymbol{a}_r &\leq \boldsymbol{v}_r^* \\ &... \\ \boldsymbol{A}_1 \boldsymbol{y} + \boldsymbol{a}_1 &\leq \boldsymbol{v}_1^* \end{aligned} \tag{4}$$

where $\boldsymbol{Z}_r$ represents a surjective mapping into the nullspace of all the previous equalities $\boldsymbol{B}_r ... \boldsymbol{B}_1$ and $\boldsymbol{u}_r$ is a variable that parameterizes the nullspace. $\boldsymbol{Z}_r$ is computed by the Singular Value Decomposition (SVD) with the QP at priority level r-1.
The consequence of this mapping in the nullspace is a reduction in term of the number variables from one hierarchy level to the next by the number of locked degrees of freedom.
We can now express a QP of the next priority level r+1, imposing the constraints of Eq. 3 and Eq. 4 so to minimize y without violating optimality of higer priority QPs:

$$min_{\boldsymbol{u}_{r+1}, \boldsymbol{v}_{r+1}} ||\boldsymbol{B}_{r+1}(\boldsymbol{y}^* + \boldsymbol{Z}_r \boldsymbol{u}_{r+1}) + \boldsymbol{b}_{r+1}|| + ||\boldsymbol{v}_{r+1}|| \tag{5}$$

$$\begin{aligned} s.t. \quad \boldsymbol{A}_{r+1}(\boldsymbol{y}_r^* + \boldsymbol{Z}_r \boldsymbol{u}_{r+1}) + \boldsymbol{a}_{r+1} &\leq \boldsymbol{v}_{r+1} \\ \boldsymbol{A}_r(\boldsymbol{y}_r^* + \boldsymbol{Z}_r \boldsymbol{u}_{r+1}) + \boldsymbol{a}_r &\leq \boldsymbol{v}_r^* \\ ... \\ \boldsymbol{A}_1(\boldsymbol{y}_r^* + \boldsymbol{Z}_r u_{r+1}) + \boldsymbol{a}_1 &\leq \boldsymbol{v}_1^* \end{aligned} \tag{6}$$

This formulation allows us to solve a stack of hierarchical tasks recursively. Right-multiplying $\boldsymbol{Z}_r$ to the inequality matrix $\boldsymbol{A}_r$ creates a row of zeros for constraints with no degrees of freedom left. This algorithm is guaranteed to find the optimal solution in a least square sense while satisfying priorities [1]. We developed a recursive algorithm to solve a stack of tasks hierarchically according to their priority.

3 Constraints

Now we are going to describe how we modeled the control objectives and their hierarchical position in the QP solver. The choice of the position in which each constraint appears is particularly important, since any solution in lower priority levels will be found in the null space of the previous solution. This framework allows the programmer to have a certain degree of freedom in choosing which task should be more important than others and this choice will be crucial for the solution returned by the QP. For our purposes we used the following order:

Rank	Nr. of constraints	Constraints/task
1	6 eq	Newton - Euler
2	2 x 6 eq	Contact constraints
	2 x 4 ineq	Center of Pressure
	2 x 4 ineq	Friction cone
3	6 eq	Momentum control
4	50 eq	PD control on posture
	2 x 6 eq	Regularizer on GRFs

3.1 Rigid Body Dynamics

The highest priority task is always set to be the Newton-Euler Eq. (7) since we are interested in writing inverse dynamics controller. This means that the solution found by the controller must obey the dynamic equations:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}^T\boldsymbol{\tau} + \boldsymbol{J}_c^T\boldsymbol{\lambda} \tag{7}$$

Where $\boldsymbol{M}(\boldsymbol{q})$ is the $n \times n$ inertia matrix, $\boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ represents the vector of the non contact forces (such as Coriolis and centrifugal terms), $\boldsymbol{\tau}$ is the vector of the commanded joint torques and $\boldsymbol{\lambda}$ are the generalized contact forces, while $\boldsymbol{S} = \begin{bmatrix}\boldsymbol{0}_{n\times 6} & \boldsymbol{I}_{n\times n}\end{bmatrix}$ is the matrix that represents the underactuation. In this case $\boldsymbol{q}$ is the configuration of the robot and it is composed by $\boldsymbol{q}_j \in \mathbb{R}^n$ (joint positions) and $\boldsymbol{x} \in SE(3)$ (floating base) so that $\boldsymbol{q} = \begin{bmatrix}\boldsymbol{x}^T & \boldsymbol{q}_j^T\end{bmatrix}^T$.
We can decompose (7) as follows, highlighting the actuated and the unactuated components:

$$\boldsymbol{M}_u(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}_u(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} + \boldsymbol{J}_{c,u}^T\boldsymbol{\lambda} \tag{8}$$

$$\boldsymbol{M}_l(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}_l(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{J}_{c,l}^T\boldsymbol{\lambda} \tag{9}$$

An advantage of this decomposition is that the torques $\boldsymbol{\tau}$ only occurs in Eq. 8 and are linearly dependent on $\ddot{\boldsymbol{q}}, \boldsymbol{\lambda}$, in fact one can write:

$$\boldsymbol{\tau} = M_u(q)\ddot{q} + N_u(q, \dot{q}) - J_{c,u}^T\lambda \tag{10}$$

From Eq. 10 we can assert that always exists a solution for $\boldsymbol{\tau}$ for any combination of accelerations and contact forces, thus we can remove the torques from our optimization problems, reducing the number of variables involved from $2n+6+12$ to $n+6+12$. However, we still have to ensure that $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\lambda}$ satisfy (9), which becomes our highest priority task, together with satisfying torque limits.

3.2 Contact constraint

In our experiments end - effectors are constrained to remain stationary. Since the feet in contact with the environment do not move, i.e. $\boldsymbol{x}_c = const$, by

differentiating it twice and using the fact that $\dot{\boldsymbol{x}}_c = \boldsymbol{J}_c\dot{\boldsymbol{q}}$, where $\boldsymbol{J}_c$ is the jacobian of the contact point(s), we get:

$$\boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} = -\boldsymbol{K}_d\boldsymbol{v}_{feet} \tag{11}$$

The last term (measured velocity of the feet) is used in order to recover possible errors in velocity increasing robustness. Taking into account the notation previously used, our $\boldsymbol{B}$ matrix would be composed by $\boldsymbol{J}_c$ and the b vector would be the remaining part of the Eq. 11 and this allows us to solve this problem for the variable $\ddot{\boldsymbol{q}}$.

3.3 ZMP

In our experiments we assume that the robot is on flat terrains and that the height of the center the of mass is constant. In this particular case the ZMP coincides with the Center of Pressure of the robot. To ensure stationary contacts, the CoP needs to reside within the interior of the end effector's support polygon. This can be formulated in the form of a linear inequality as follows:

$$\begin{bmatrix} -dx \\ -dy \\ 0 \end{bmatrix} \leq \frac{F_z^l \boldsymbol{CoP}_l + F_z^r \boldsymbol{CoP}_r}{F_z^r + F_z^l} \leq \begin{bmatrix} dx \\ dy \\ 0 \end{bmatrix} \tag{12}$$

where dx and dy denotes the admissible region of the ZMP.
If we look closer to one element of Eq. 12 we start by the definition of the x component of CoP wich is nothing else than $-\frac{\tau_y}{F_z}$. Naturally, this would be valid just for one component of one foot. The next step is to express the CoP in the frame of the feet. The last important aspect to take into account is the fact that in our simulations the robot is always in double support, hence we express in Eq. 12 a weighted average between the two feet. The optimization variables of this constraint are both torques and forces acting on the feet.

3.4 Friction cone

The imposition of the ground reaction forces (GRFs) to fall inside the friction cone is a necessary condition to ensure the feet to not slip on the ground.
The forces that generate the GRFs are the weight force and external forces applied to the feet in the x and y directions. The foot remains in contacts with the ground as long as the contact force $\boldsymbol{F}^c = m\boldsymbol{g} - \boldsymbol{F}^{ext}$ lies inside the *friction cone* directed by $\hat{\boldsymbol{n}}_{\boldsymbol{i}}$ shown in Fig. 2.
In order to deal with linear inequality constraints we modeled cones as a pyramids as shown in Fig 2.
The equations of each foot for the optimization problem are the following:

$$|\boldsymbol{F}_i^c \cdot \hat{\boldsymbol{t}}_i| \leq \mu(\boldsymbol{F}_i^c \cdot \hat{\boldsymbol{n}}_i) \tag{13}$$

$$|\boldsymbol{F}_i^c \cdot \hat{\boldsymbol{b}}_{\boldsymbol{i}}| \leq \mu(\boldsymbol{F}_i^c \cdot \hat{\boldsymbol{n}}_i) \tag{14}$$

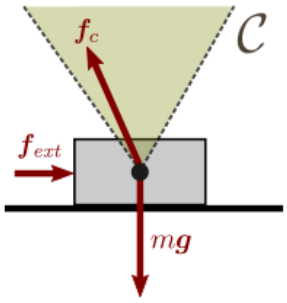

Figure 1: Forces on the foot

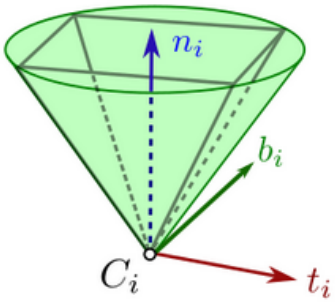

Figure 2: Friction cone approximation

with i = l,r (left and right foot) and the coefficient of static friction μ set to 0.5. In this case ($\hat{\boldsymbol{n}}_{\boldsymbol{i}}$, $\hat{\boldsymbol{b}}_{\boldsymbol{i}}$) are any basis of the tangential contact plane such that ($\boldsymbol{t_i}$, $\hat{\boldsymbol{b}}_{\boldsymbol{i}}$, $\hat{\boldsymbol{n}}_{\boldsymbol{i}}$) is an orthonormal frame as shown in Fig. 2.

3.5 Momentum Control

The regulation of the linear and angular momentum of the robot's center of mass (COM) is a task that has been studied extensively to improve the performance of whole-body balancing controllers. It is most often accomplished through the derivation of the **centroidal dynamics**:

$$\begin{bmatrix} \dot{\boldsymbol{h}}_{ang} \\ \dot{\boldsymbol{h}}_{lin} \end{bmatrix} = \boldsymbol{H}_G \ddot{\boldsymbol{q}} + \dot{\boldsymbol{H}}_G \dot{\boldsymbol{q}} \tag{15}$$

Eq. (15) dictates the relation between joint space acceleration and linear and angular momentum variation expressed at the robot's COM. The matrix $\boldsymbol{H_G}$ is the **Centroidal Momentum Matrix** (CMM) and is the jacobian which maps joint space velocities to momentums. There are multiple ways to compute the

CMM and the residual term $\dot{\boldsymbol{H}}_{\boldsymbol{G}}\dot{\boldsymbol{q}}$, such as the algorithm provided in [2]. In our simulation we developed the solution proposed in [3], which makes use of the joint space dynamics to derive the centroidal dynamics.
This comes in very handy in our implementation since our hierarchical controller already needs the computation of the joint dynamics. The derivation is done as follows: we define the mapping between velocities in world frame and velocities in floating base $\boldsymbol{\Phi_f}$ frame as:

$$\begin{bmatrix}\boldsymbol{\omega}_f\\ \boldsymbol{v}_f\end{bmatrix} = \boldsymbol{\Phi}_f \begin{bmatrix}\boldsymbol{\omega}_0\\ \boldsymbol{v}_0\end{bmatrix} = \begin{bmatrix}{}^f\boldsymbol{R}_0 & \boldsymbol{0}\\ \boldsymbol{0} & {}^f\boldsymbol{R}_0\end{bmatrix}\begin{bmatrix}\boldsymbol{\omega}_0\\ \boldsymbol{v}_0\end{bmatrix} \tag{16}$$

Note that $\boldsymbol{\Phi_f}$ is a jacobian so from (9):

$$\begin{aligned}\boldsymbol{M}_l(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}_l(\boldsymbol{q},\dot{\boldsymbol{q}}) &= -\boldsymbol{g}_l(\boldsymbol{q}) + \boldsymbol{J}_{c,l}^T\boldsymbol{\lambda} = \boldsymbol{\tau}_f\\ \boldsymbol{M}_l(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}_l(\boldsymbol{q},\dot{\boldsymbol{q}}) &= \boldsymbol{\Phi}_f^T\boldsymbol{F}_f\\ \boldsymbol{\Phi}_f^{-T}(\boldsymbol{M}_l(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}_l(\boldsymbol{q},\dot{\boldsymbol{q}})) &= \boldsymbol{\Phi}_f^{-T}(\boldsymbol{\Phi}_f^T\boldsymbol{F}_f) = \boldsymbol{F}_f\end{aligned} \tag{17}$$

Where we have the generalized forces acting on the floating base on the RHS of the equation (including gravity). To transform the forces at the floating base to forces at the COM "frame" we use the spatial transformation matrix ${}^f\boldsymbol{X}_G^T$ defined as:

$$\begin{aligned}{}^f\boldsymbol{X}_G^T &= \begin{bmatrix}{}^G\boldsymbol{R}_f & {}^G\boldsymbol{R}_f\boldsymbol{S}({}^f\boldsymbol{p}_G)^T\\ \boldsymbol{0} & {}^G\boldsymbol{R}_f\end{bmatrix}\\ \boldsymbol{F}_G &= {}^f\boldsymbol{X}_G^T\boldsymbol{F}_f\end{aligned} \tag{18}$$

The orientation of the COM frame is chosen arbitrarily. From the Newton's second law it follows that

$$\begin{aligned}\boldsymbol{F}_G = \dot{\boldsymbol{h}}_G = \boldsymbol{H}_G\ddot{\boldsymbol{q}} + \dot{\boldsymbol{H}}_G\dot{\boldsymbol{q}} &= {}^f\boldsymbol{X}_G^T\boldsymbol{\Phi}_f^{-T}(\boldsymbol{M}_l(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}_l(\boldsymbol{q},\dot{\boldsymbol{q}}))\\ \boldsymbol{H}_G &= {}^f\boldsymbol{X}_G^T\boldsymbol{\Phi}_f^{-T}\boldsymbol{M}_l(\boldsymbol{q})\\ \dot{\boldsymbol{H}}_G\dot{\boldsymbol{q}} &= {}^f\boldsymbol{X}_G^T\boldsymbol{\Phi}_f^{-T}\boldsymbol{c}_l(\boldsymbol{q},\dot{\boldsymbol{q}})\end{aligned} \tag{19}$$

We can then define a simple PD control law to stabilize the position of the COM and the angular momentum of the robot:

$$\boldsymbol{H}_G\ddot{\boldsymbol{q}} + \dot{\boldsymbol{H}}_G\dot{\boldsymbol{q}} = \boldsymbol{K}_P\begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{e}_p\end{pmatrix} + \boldsymbol{K}_D\begin{pmatrix}\boldsymbol{e}_\omega\\ \boldsymbol{e}_v\end{pmatrix} \tag{20}$$

Where $\boldsymbol{e}_p, \boldsymbol{e}_v, \boldsymbol{e}_\omega$ denote the error in COM position, linear velocity and angular velocity, respectively. There is no feedback on the orientation because the frame attached to the center of mass has no predefined orientation (it is a point in $\mathbb{R}^3$). In our simulations we assigned it to be the same as world frame: ${}^0\boldsymbol{R}_G = \boldsymbol{I}_{3\times3} \in SO(3)$.

3.6 PD control on posture

Once the main objective of stability is achieved through the previous constraint, we also need to ensure that the robot has a stable posture. At the beginning of our implementation, without this simple yet effective component, the robot arms (not constrained) were free to move within the null space of previous solutions. By locking more degrees of freedom, we constrain the robot to remain in the original configuration by means of a PD controller:

$$\ddot{\boldsymbol{q}}_j = \boldsymbol{K}_P(\boldsymbol{q}_{j,init} - \boldsymbol{q}_j) - \boldsymbol{K}_D\dot{\boldsymbol{q}}_j \tag{21}$$

Where $\ddot{\boldsymbol{q}}_j$ $\dot{\boldsymbol{q}}_j$ and $\boldsymbol{q}_j$ are respectively accelerations, velocities and positions of all the 50 joints of the humanoid. Although it seems a trivial approach it solved successfully the problem of the unconstrained motion of the arms.

3.7 Regularizer on GRFs

The aim of this constraint is to reduce the intensity of the contact forces acting on the feet of the robot. To accomplish this, we set in the optimization problem the matrix B to be the identity in the correspondence of the 12 x 12 block that multiplies the contact forces while the vector b is set entirely to 0.
This last constraint was very important to stabilize the contact forces and allowed us to achieve our final results.

4 Simulations and results

In this last section we will show a series of behaviours of our robot when we have different constraints acting.
The main objective is to show how any constraint that we have introduced becomes active and what are the effects on the robot. The hierarchical approach is particularly flexible, since the programmer can activate/deactivate any constraint at any time and this allows us to verify the results of the constraint on the robot once it is acting.
The simulations involve tracking of a trajectory for the CoM, to show the effectiveness of the method implemented.
In order to prove the robustness of the algorithm the robot is subject to external disturbances, in the form of an external push acting on the robot which is unknown to the robot. Furthermore, to prove the effectiveness of the ZMP constraint we will increase the external force acting on the robot showing the importance of the ZMP constraint.

4.1 PD on posture

The first result that we show is to prove the behaviour of the robot with and without the controller on the posture. Without this constraint the robot is free to express it self in fancy movements in the null space of previous solutions. The

result is that the arms start moving around, while all other tasks are fulfilled. in Fig. 3 this is evident. On the other hand once the constraint becomes active,

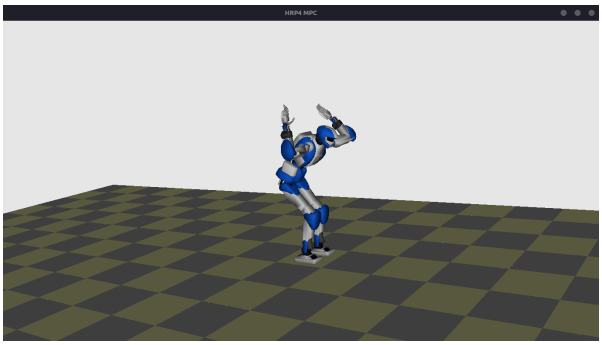

Figure 3: Null space movements

the robot tends to stay in the initial configuration (which is our desired one) and it looks like in in Fig. 4

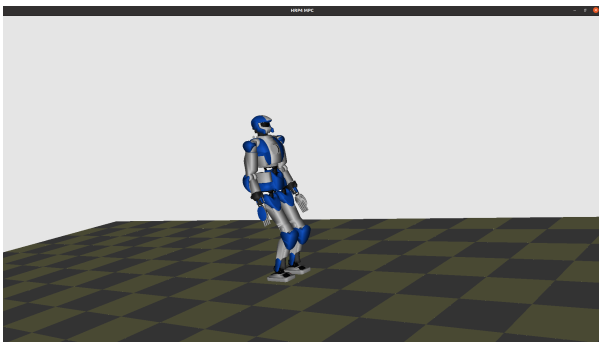

Figure 4: PD control on posture

4.2 Tracking of the CoM

We validated the correctness of our momentum controller via assigning a sinusoidal trajectory for the COM:

$$\begin{aligned} \mathrm{COM}_{d,x} &= \mathrm{COM}_{init,x} + 0.08\sin(\tfrac{2\pi}{1000}t) \\ \mathrm{COM}_{d,y} &= \mathrm{COM}_{init,y} + 0.03\cos(\tfrac{2\pi}{1000}t) \\ \mathrm{COM}_{d,z} &= \mathrm{COM}_{init,z} \end{aligned}$$

Where t denotes the time step of the simulation. This periodic motion has a period of 1000 simulation timesteps, which translates to 10 seconds. The amplitudes have been chosen in a way that allows the robot to follow the trajectory without falling. The results are illustrated in Fig. 5

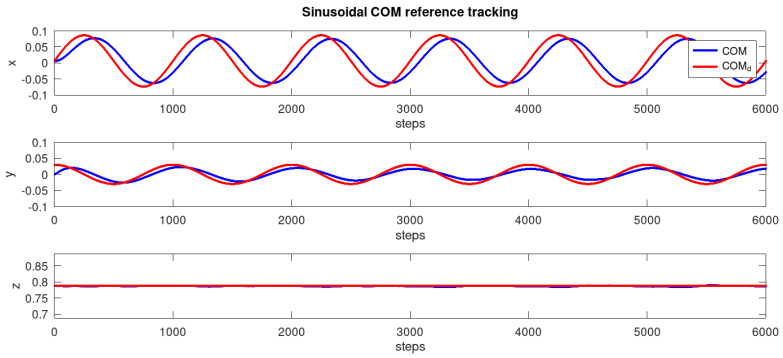

Figure 5: The center of mass of the robot tracks a sinusoidal reference on x [m] and y [m] slightly lagging behind by a constant offset.

4.3 External push

The last part of our simulations is intended to show the robustness of the method under unknown external disturbances.

4.3.1 Weak push

An external force of 50 [N] is applied on the torso of the robot along the y direction. In the simulations the immediately after the push the robot is as in Fig. 7.

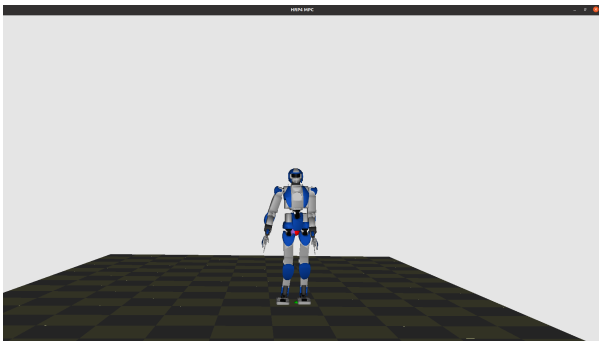

Figure 6: 50 [N] push along y

The robot is able to recover from the external push and the error in position converges. To show this result we report the error of the CoM in both x and y over time. In Fig. 6 is clear the moment in which the push happens since a huge error takes place in the y component. As one can expect, after a transient the error converges to zero and the system is able to recover, proving its robustness to an external push.

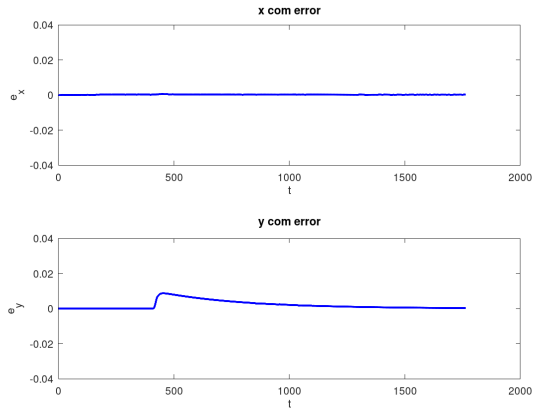

Figure 7: Error on x and y component

4.3.2 Strong push without ZMP constraint

The strong push aims at proving the effectiveness of the ZMP constraint. The robot receives a push of 100 [N] along the y direction and with the ZMP inactive is not able to resist it and falls catastrophically. A glimpse of the situation in Fig. 8.

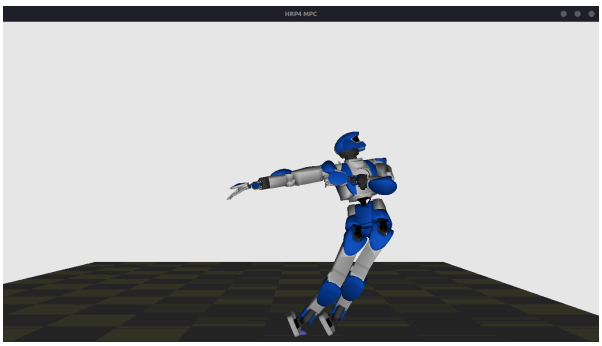

Figure 8: 100 [N] push along y without ZMP

It is evident from this image that the robot is trying to maintain its feet on the ground but the disturbance is to strong and the robot is not able to resist it. The plots in Fig. 9 show how the error diverges once the robot fall, since it is not able to solve the task anymore.

4.3.3 Strong push with ZMP constraint

Finally we activate the ZMP constraint and the results are fascinating. The external push is exactly the same but now the robot has also the inequality

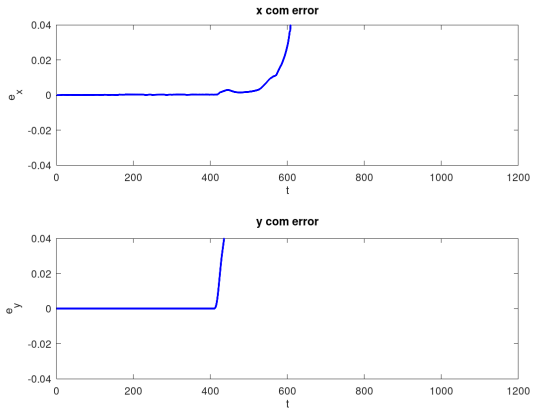

Figure 9: Error on x [m] and y [m] component

constraint on the position of the ZMP. Once the ZMP would fall out of the support polygon, due to the external push, the solution found by the solver is such that the robot relaxes the contact constraint of the feet in order to satisfy the ZMP constraint. One aspect that we would like to underline is the fact that the equality constraint enters the algorithm in the cost function subject to some inequalities. In this case the inequality constraint can not be violated, while the equality constraint can be slightly relaxed. This must be the case since it is unfeasible for the robot to resist such a push maintaining its feet on the ground. For this scenario we report an image of the robot after the push (Fig. 10), once it has recovered, and also the usual graphs of the CoM's error on the x and y coordinates (Fig. 11).

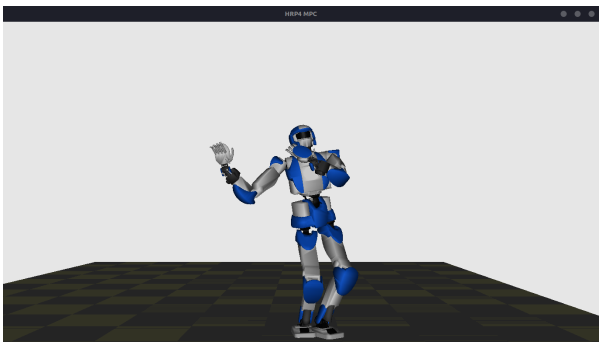

Figure 10: 100 [N] push along y with ZMP

In Fig. 12 we show the ZMP of the robot in the simulation. Is clear how the inequality constraint activates once the y coordinate reaches 0.1 [m](which is the maximum value it can reach) and then the robot is able to recover its stability

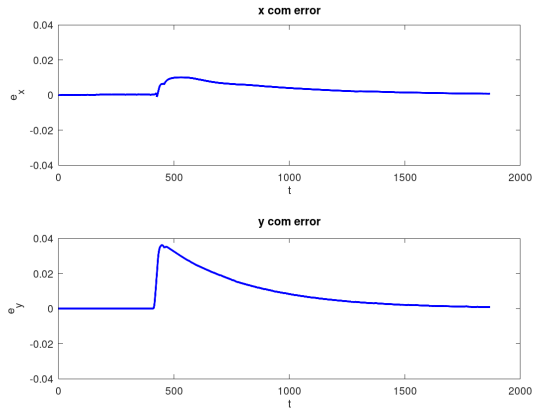

Figure 11: Error on x [m] and y [m] component

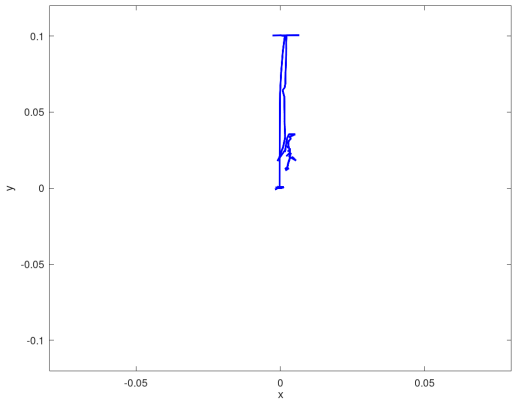

Figure 12: ZMP constraint along x [m] and y [m]

after the push. Since the push happens along the y direction, the change of the ZMP in along the x is minimal, and there is only some noise affecting the plot.

5 Conclusions

This algorithm is very flexible and allowed us to try many different combination of priority level, giving us the possibility to deeply explore different scenarios. Finally we have proved the effectiveness of the method for the balancing task of a humanoid robot, also in the presence of external external pushes.

$(x_{init}, x_{goal}, iterations)\ \mathcal{T} \leftarrow InitializeTree()\ \mathcal{T} \leftarrow InsertNode(x_{init}, \mathcal{T})$

$N \leftarrow 0$ $N \neq iterations$ $x_{rand} \leftarrow RandomState()$ $(x_{near}) \leftarrow NearestState(x_{rand}, \mathcal{T})$
$u \leftarrow SolveInput(x_{near}, x_{rand})$ $x_{new} \leftarrow NewState(x_{near}, u)$ $\mathcal{T} \leftarrow InsertNode(x_{new}, \mathcal{T})$
$GoalCheck(x_{new}, x_{goal})$ Return $\mathcal{T}$ $N \leftarrow N + 1$

References

[1] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti, "Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid," *CoRR*, vol. abs/1410.7284, 2014. [Online]. Available: http://arxiv.org/abs/1410.7284

[2] D. E. Orin and A. Goswami, "Centroidal momentum matrix of a humanoid robot: Structure and properties," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 653–659.

[3] P. Wensing and D. Orin, "Improved computation of the humanoid centroidal dynamics and application for whole-body control," *International Journal of Humanoid Robotics*, vol. 13, p. 1550039, 09 2015.