Push recovery for humanoid robots

Isabella Claire Adriani, Christian D'Amico

July 2020

1 Introduction

In this report two humanoid robot walking control architectures will be analyzed and compared. The first one is a typical three-layered architecture using DCM based methods and a step adapter. The second one is based on a Model Predictive controller that is proven to be stable when using some specific constraints. The second one is the so called IS-MPC scheme for gait generation developed by Scianca et al. [3]. Both are capable of withstanding external disturbances (such as pushes) during operation.

In what follows, a brief theoretical description of both methods will be given. Then they will be compared, both conceptually and practically. The practical comparison will be aided by a few simulations of the HRP4 humanoid robot controlled by the two architectures on simple walking tasks with an external added push.

2 Three layered hierarchical architecture with push recovery

Many different hierarchical architectures have been developed to solve the humanoid locomotion problem. Our sources ([5], [6], [7]) made use of a three-layered control structure. In general, these layers are;

- A *Trajectory optimization for foot-step planning* layer responsible for generating a suitable footstep plan and interpolating robot kinematic quantities around it
- A *Simplified model control* layer that exploits a simplified model of the robot to compute desired center-of-mass trajectories
- A *Whole-body Quadratic Programming (QP) control* layer that generates robot's joint torques to ensure that the desired trajectories are followed.

This comparison will focus on the controller specified by [7], in which the controller is expanded with an additional module: a *Step Adapter* capable of online modifications of the desired trajectories of the feet, timings and DCM whenever

the robot gets affected by external disturbances. In this context, this will be used as a tool to recover from an external push.

The controller in [7] requires implementation of a trajectory optimization layer capable of handling both single and double support phases while ensuring smooth desired DCM trajectories. To achieve this the first layer was structured as follows. Steps were planned to use the optimal step planner in [5]. Trajectory optimization was performed as in [6]. Finally, the simplified model control and the step adapter of [7] are implemented.

In what follows, all the components of the control system will be discussed. As it is not used here, discussion of the implementation of the whole-body QP programming layer will be skipped.

2.1 Trajectory optimization

The goal of trajectory optimization layer is to generate the nominal footprints and the desired feet and DCM trajectories. The problem can be divided into two sub-tasks: the generation of the nominal footprints and the step adapter that adjusts the footsteps quantities by considering the error between the current and the desired ZMP, DCM and impact times. To plan the desired footprint [5] the robot is approximated as a unicycle and the wheels represent the feet. The footsteps are obtained from the sampling of the unicycle trajectories. Whereas the desired feet trajectory can be obtained by a cubic spline interpolation once the footsteps are planned.

2.1.1 Footstep Planner

Humanoid robot walking motion can be seen as the sequence of 4 states: *switch-in* $\rightarrow$ *stance* $\rightarrow$ *switch-out* $\rightarrow$ *swing*. Each of these sequences is terminated by an impact time. The robot is in *single support phase* during leg swing and in *double support phase* in the other three states. Therefore, the footstep planning problem may be understood as the problem of defining the position of the footsteps and their impact times.

A commonly used approach for a solution is to use a unicycle model. In fact, common humanoid walking follows the same kinematic constraints (no movement on the sagittal axis). It's also natural to view the position of the wheels of the unicycle as possible feet positions. In fact, obtaining feet position from the sampled trajectory of the unicycle is just a matter of alternating between setting future steps on the right x_k^r or left foot x_l^l

$$x_k^l = x_k + R_2(\theta_k) \begin{pmatrix} 0 \\ m \end{pmatrix}, \; x_k^r = x_k + R_2(\theta_k) \begin{pmatrix} 0 \\ -m \end{pmatrix} \tag{1a}$$

$$\theta_k^l = \theta_k^r = \theta \tag{1b}$$

Thus, it is possible to proceed as follows. A fictional unicycle is controlled from a given start position up to a final goal. Then, its trajectory is sampled at a fixed rate. Finally, some optimization or search process can be designed to find optimal footstep placement according to some constraints.

The approach of [5] consists in setting it as an optimization problem. This makes it possible to also impose bounds on key quantities:

- The orientation difference $\Delta_\theta = |\theta^l_{ds} - \theta^r_{ds}|$
- The relative feet distance $\Delta_x = ||x^l_{ds} - x^r_{ds}||$
- Step duration $\Delta_t = t^{f+1}_{imp} - t^f_{imp}$

Then the cost function may be chosen so as to balance fast-short steps and slow-long ones

$$\phi = K_t \frac{1}{\Delta_t^2} + K_x ||\Delta_x||^2 \tag{2}$$

where K_t and K_x are two positive numbers. Step length can be increased by lowering K_x; step duration can be decreased by lowering K_t. In the current implementation, a good balance for the robot in the case study has been found to be $K_x = 1$, $K_t = 0.01$.

Thus, the whole optimization problem can be described as:

$$\begin{aligned} \underset{t_{imp}}{\text{minimize}} \quad & \phi(\Delta_t, \Delta_x) && \text{(3a)} \\ \text{s.t:} \quad & t_{min} \leq \Delta_t \leq t_{max} && \text{(3b)} \\ & \Delta_x \leq d_{max} && \text{(3c)} \\ & |\Delta_\theta| \leq \theta_{max} && \text{(3d)} \end{aligned}$$

The only decision variable is the time of the impact. From that, the foot position can be computed by retrieving the unicycle position and applying Eqs. 1. The optimization problem can be easily solved iteratively:

1. Start from an accepted step (either from previous iteration or as the starting step)
2. Iterate over all the remaining unicycle trajectory and remove all unfeasible points.
3. Compute the cost function of the remaining feasible possible steps.
4. Take the step with best possible function as the new accepted one.
5. Go back to 1

An asset of this method is the possibility of linking different sets of plans in an MPC-like fashion. In fact, if at any point during the trajectory the plan has to be changed (for example because of modifications from the step adapter), the remaining plan can be discarded and a new one can be defined from the next footstep position. This operation can be carried out during a double support phase. Hence this planner is an offline one that is capable of quickly generating different plans for quasi-online uses.

2.1.2 The desired DCM trajectory

The DCM follows the time evolution [2][6][7]

$$\xi = r^{zmp} + e^{\omega t}(\xi_0 - r^{zmp}) \tag{4}$$

where ω, in case of constant CoM heigth h_{CoM}, is the inverse of the pendulum time constant, i.e. $\omega = \sqrt{h_{CoM}/g}$, ξ_0 is the initial position of the DCM, r^{zmp} is the position of the ZMP (placed on the stance foot) and t is part of the step domain $t \in [0, t_i^{step}]$ where t_i^{step} is the duration of the i-th step. Assuming that the final DCM and ZMP positions coincide, the desired position of the DCM at the end of each step can be found through the usage of equation (4):

$$\begin{cases} \xi_{N-1}^{eos} = r_N^{zmp} \\ \xi_{i-1}^{eos} = \xi_i^{ios} = r_i^{zmp} + e^{-\omega t_i^{step}}(\xi_i^{eos} - r_i^{zmp}) \end{cases} \tag{5}$$

where ξ_i^{ios} and ξ_i^{eos} are respectively the initial and final positions of the desired DCM of the i-th step.

The following desired DCM trajectory is obtained by substituting (5) into (4):

$$\xi_i(t) = r_i^{zmp} + e^{\omega(t - t_i^{step})}(\xi_i^{eos} - r_i^{zmp}) \tag{6}$$

Differentiating the desired DCM velocity is obtained:

$$\dot{\xi}_i(t) = \omega e^{\omega(t - t_i^{step})}(\xi_i^{eos} - r_i^{zmp}) \tag{7}$$

This technique is limited to single support phases only. The discontinuity of the desired ZMP between two single support phases leads to the discontinuity of the desired joint torques. The desired DCM trajectory must belong at least to a C^1 class to guarantee a continuous ZMP trajectory. This can be obtained by smoothing two consecutive single support phases of the DCM trajectory with a third order polynomial function [6]:

$$\xi^{DS} = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{8}$$

where $a_i, i = 0, 1, 2, 3$ must satisfy the following boundary conditions:

$$\begin{cases} \xi_i^{DS_i} = r_{i-1}^{zmp} + e^{\omega(t_{i-1}^{DS_i} - t_{i-1}^{step})}(\xi_{i-1}^{eos} - r_{i-1}^{zmp}) \\ \dot{\xi}_i^{DS_i} = \omega e^{\omega(t_{i-1}^{DS_i} - t_{i-1}^{step})}(\xi_{i-1}^{eos} - r_{i-1}^{zmp}) \\ \xi_i^{DS_e} = r_i^{zmp} + e^{\omega(t_i^{DS_e} - t_i^{step})}(\xi_i^{eos} - r_i^{zmp}) \\ \dot{\xi}_i^{DS_e} = \omega e^{\omega(t_i^{DS_e} - t_i^{step})}(\xi_i^{eos} - r_i^{zmp}) \end{cases} \tag{9}$$

2.2 Simplified model control

In what follows, [7] makes the assumption of constant centroidal angular moment. This is a typical assumption of human like model that seems to be approximately true in typical human walking. Most importantly, this makes it

so that the VRP projection on the walking surface is exactly coincident with the ZMP. Hence, working with a LIP, the two quantities can be considered equal on the plane of motion. Thus, planning and controlling the ZMP or VRP is completely equivalent.

Building on this, the simplified model control layer described in [7] is a simple control law that computes a desired VRP capable of stabilizing the DCM dynamics around the desired DCM. In fact, requiring

$$r_{des}^{vrp} = \xi_{ref} - \frac{\dot{\xi}_{ref}}{\omega} + K^{\xi}(\xi - \xi_{ref}) \tag{10}$$

the DCM error dynamics is simply

$$\dot{\tilde{\xi}} := \dot{\xi} - \dot{\xi}_{ref} = \omega(I_3 - K^{\xi})\tilde{\xi} \tag{11}$$

Thus, an asymptotically stable close loop error dynamics can be obtained by choosing $K^{\xi} > I_3$.

Such a desired VRP position is then used to shape movement in the whole-body QP control layer. In fact, linear momentum conservation can be exploited to move the VRP position by applying torques at the robot joints. In this implementation, the robot model is a simple LIP. Therefore, the VRP position will simply be an applied control that shapes the movement of the DCM. Thus, in principle, on an LIP, any possible desired DCM trajectory is already stabilized by this control law with no need of a step adapter. However, feasibility can still be studied even in this context by focusing on the position of the VRP with respect to the feet and the support polygon. Hence the usage of a step adapter becomes critical in achieving motion that respects the ZMP feasibility boundaries.

2.3 Step adapter

The step adapter is not too dissimilar from the one in [4]. The main difference is that, in [7], infinite step capturability is not enforced while it's explicitly set as constraint in [4].

The step adapter solves a QP problem to modify the next step according to measurements on the current DCM and VRP positions. The main theoretical idea is trying to find a relation between the ZMP and DCM that would be linear. The problem can be solved in this way by a QP algorithm retaining fast convergence speeds.

The relationship between ZMP and DCM can be obtained by defining an exponential interpolation for the ZMP trajectory:

$$r^{zmp}(t) = Ae^{-\omega t} + B \tag{12}$$

where $A, B \in \mathbb{R}^2$ are chosen to satisfy the boundary conditions of the single support phase $r^{zmp}(0) = r_1^{zmp}$ and $r^{zmp}(T) = r_2^{zmp}$. Thus,

$$A = \frac{(r_2^{zmp} - r_1^{zmp})\sigma}{1-\sigma}, \quad B = \frac{r_1^{zmp} - r_2^{zmp}\sigma}{1-\sigma} \tag{13}$$

where T is the duration of the single support phase and

$$\sigma := e^{\omega T} \tag{14}$$

This, plugged in the DCM dynamics yields the ODE

$$\dot{\xi} - \omega\xi = -\omega r^{zmp}(t) = -\omega A e^{\omega t} - \omega B. \tag{15}$$

Thus, the solution to (15) is:

$$\xi(t) = e^{\int \omega dt}\left[\int \left(-\omega A e^{-\omega t} - \omega B\right) e^{\int -\omega dt} dt + C\right] \tag{16}$$

where $C \in \mathbb{R}^2$ is chosen by the boundary conditions

$$\xi(0) = \xi_0 = \frac{A}{2} + B + C_0 \tag{17}$$

and

$$\xi(T) = \xi_T = \frac{A}{2}e^{-\omega T} + B + C_f e^{\omega T} \tag{18}$$

C_0 and C_f must be equal to satisfy the above boundary conditions. Therefore, by combining (17) and (18), one has:

$$\xi_0 - \frac{A}{2} - B = \left(\xi_T - \frac{A}{2}e^{-\omega T} - B\right) e^{-\omega T} \tag{19}$$

Substituting (13) and (14) in (19) yields:

$$\xi_T - \sigma\xi_0 - \frac{r_1^{zmp} + r_2^{zmp}}{2}(1-\sigma) = 0 \tag{20}$$

By denoting the DCM offset as $\gamma_T = \xi_T - r_T^{zmp}$ where r_T^{zmp} is the ZMP position at the beginning of the next step, the following relationship is obtained:

$$r_T^{zmp} + \gamma_T + \left(\frac{r_1^{zmp} + r_2^{zmp}}{2} - \xi_0\right)\sigma = \frac{r_1^{zmp} + r_2^{zmp}}{2} \tag{21}$$

Hence the desired linear equality constraint has been obtained.

A cost function that is able to maintain desired footstep values close to nominal ones is chosen as:

$$J = \alpha_1 \left\| r_T^{zmp} - r_{T,nom}^{zmp} \right\|^2 + \alpha_2 \left\| \gamma_T - \gamma_{nom} \right\|^2 + \alpha_3 \left| \sigma - e^{\omega T_{nom}} \right|^2 \tag{22}$$

where the nominal quantities are computed on the nominal footsteps of the already planned motion. In many different step adapters used in DCM-based methods it's common to have higher weights on the DCM offset for stability

reasons. In the current implementation it was found useful to also have high gains on the ZMP position term, while having very low ones for the timing term. The gain matrix $H = diag\{\alpha_1\alpha_2, \alpha_3\}$ used in this comparison is[1]

$$H = \begin{pmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 0 & 0.01 \end{pmatrix} \tag{23}$$

Finally, some additional constraints are imposed on the position of the feet and on possible step timings:

$$\begin{pmatrix} I_2 & 0_2 & 0_{2\times 1} \\ -I_2 & 0_2 & 0_{2\times 1} \\ 0_{1\times 2} & 0_{1\times 2} & I_1 \\ 0_{1\times 2} & 0_{1\times 2} & -I_1 \end{pmatrix} \begin{pmatrix} r_t^{zmp} \\ \gamma_T \\ \sigma \end{pmatrix} \leq \begin{pmatrix} r_{T,max}^{zmp} \\ -r_{T,min}^{zmp} \\ \sigma_{max} \\ -\sigma_{min} \end{pmatrix} \tag{24}$$

This QP problem is solved at every iteration of the controller by plugging in the current positions of VRP (in r_1^{zmp}) and DCM (in ξ_0). The solution is then used to modify the smoothing and consequently the simplified model control's output. If the position or timing of the steps has been significantly changed, (i.e. with a difference greater than $0.01m$ for the ZMP or DCMeos position, or $0.01s$ for the timing) the plan has to be recomputed.

3 Intrinsically Stable Model Predictive Control

The second control algorithm that we will discuss is based on a Model Predictive Control-based framework ([3]). It has three main components. An external footstep generation module defines candidate footstep position and timings. Then, the MPC controller generates the ZMP and CoM trajectories while changing footstep position according to an optimal criteria and constraints. This is done by using a simplified model of the robot (a simple LIP) to predict future states. Finally, MPC solutions are fed the robot's kinematic controller together with a swing-foot trajectory generated by an external module. This structure is instrumental in obtaining dynamically stable walking motion.

For a humanoid walking controller to be effectively used in real-world scenarios, low required computation time is essential. Thus, the MPC optimization should be solvable quickly. Therefore, many different solutions are based on constrained quadratic linear MPC problems that may be solved using Quadratic Programming. This solution method can be applied here as the orientation of the footsteps is not a decision variable of the MPC. Hence, the controller remains linear as the external planner handles the nonlinearity.

[1] noting that the first two weights are in fact vectors, while the last one is a scalar

The whole controller, using velocities of the ZMP as input, can be described by:

$$\begin{cases} \min\limits_{\dot{X}_z^k, \dot{Y}_z^k, X_f^k, Y_f^k} & \left\|\dot{X}_z^k\right\|^2 + \left\|\dot{Y}_z^k\right\|^2 \quad +\beta\left(\left\|X_f - \hat{X}_f\right\|^2 + \left\|Y_f - \hat{Y}_f\right\|^2\right) \\ & \text{subject to:} \\ \bullet \text{ ZMP constraints} \\ \bullet \text{ Kinematic constraint} \\ \bullet \text{ Stability constraints} \end{cases} \tag{25}$$

where $\dot{X}_z, \dot{Y}_z$ (controls on the velocity of the ZMP) and X_f, Y_f (position of the feet) are the decision variables, while $\hat{X}_f^2, \hat{Y}_f^2$ are the input sequence of steps obtained by the external planner. An additional parameter β is chosen to specify the relative importance of the velocity and position terms of the cost function. That is often imposed to be $\beta \gg 1$ so as to prefer solutions that are close to the nominal footsteps unless significant external disturbances are present.

Some attention should be given to the time horizon of this MPC controller. In particular, two different time horizons can be identified in the architecture. A *preview horizon* T_p for the generation of nominal steps of the candidate footstep generator; and a *control horizon* $T_c < T_p$ for the MPC controller. The preview horizon proves useful when computing the *anticipative tail* in Sec. 3.1.

The ZMP input is chosen to be piecewise-linear:

$$x_z(t) = x_z^i + (t - t_i)\dot{x}_z^i \quad \text{for } \ t \in [t_i, t_{i+1}] \tag{26}$$

The main feature of this approach is the possibility of using the MPC framework to address constraints of different nature. In fact, alongside typical constraints on ZMP and kinematics, additional constraints on stability are enforced. This results in a system that is proven to be stable in the nominal case (no perturbations). Additionally, the MPC structure is capable of resisting external perturbations by moving the position of the footsteps and changing the ZMP inside the support polygon.

The implementation in the following comparison will not be based on the described framework. In fact, the IS-MPC that we used for the comparison also comprises a step timing adaptation module. It can be shown that, in case of an external perturbation yielding to a loss of feasibility, the so called MPC feasibility region can be modified by appropriately changing the current step duration.

3.1 Stability constraints

Stability conditions are available for the dynamics of the LIP. In fact, Despite its inherent diverging dynamic, for any input ZMP trajectory, there exists an initial DCM that results in bounded motion. This particular value can be expressed as:

$$x_u^k = \eta \int_{t_k}^{\infty} e^{-\eta(\tau - t_k)} x_z(\tau) d\tau \tag{27}$$

The main idea of the MPC stability constraints, is manipulating this condition so that it can be expressed in terms of the decision variables of the MPC problem. Thus, it's possible to search for solutions that are provably stable.

This can be achieved in the following way. Using the piecewise linearity of the input, Eq. 27 can be rewritten as

$$x_u^k = x_z^k + \frac{1 - e^{-\eta\delta}}{\eta} \sum_{1=9}^{\infty} e^{-i\eta\delta} \dot{x}_z^{k+i} \tag{28}$$

which, splitting control variables in the control horizon from the subsequent ones, takes the form

$$\sum_{i=0}^{C-1} e^{-i\eta\delta} \dot{x_z}^{k+1} = -\sum_{i=C}^{\infty} e^{i\eta\delta} \dot{x}_z^{k+1} + \frac{\eta}{1 - e^{-\eta\delta}} (x_u^k - x_z^k) \tag{29}$$

This opens a set of possibilities in handling the part of the infinite sums that is beyond the control horizon. Three possible solutions are presented in [3]:

- A *Truncated tail* that sets all ZMP velocities to 0
- A *Periodic tail* that infinitely replicates the ZMP velocities that were observed within the control horizon
- An *Anticipative tail* that uses the generated footsteps beyond the control horizon to produce ZMP velocities. This is done in two subsequent steps:
  - Generate a feasible motion of the ZMP in admissible regions of space according to the planned footsteps
  - Sample the time derivative of this trajectory

An important result is obtained considering *Anticipative tails*. If the preview horizon is sufficiently large, then MPC controller is recursively feasible when the anticipative tail is used. A recursively feasible MPC scheme always brings the system in feasible states when the system is initialized in a feasible state. Moreover, it can be shown that recursive feasibility implies internal stability for IS-MPC.

In what follows, the anticipative tail will be used in all simulations.

4 Comparison

In what follows the two described algorithms will be compared. At first, a more theoretical comparison will highlight the key differences and point equivalent modules of the two methods. Then simulations will be carried out for a more practical comparison.

4.1 Theoretical differences

Despite giving them different names, even the MPC-based architecture is somewhat based on a three layered hierarchy. In fact, the footstep generator serves a similar purpose to the one of the optimization layer, with the key difference of not interpolating a DCM trajectory. Then we may count as a second layer the MPC step. Finally, the swing foot trajectory generator and the kinematics controller serve an equivalent purpose to [7]'s third layer. In fact, it's just presented in a different light in [3] because it's applied to a position-controlled robots; while [7] controls a robot in torque.

An interesting discussion can be had about how stability is achieved in the two controllers. The MPC-based architectures, ensures stability by applying constraints during the QP solvable MPC step. Then, the MPC structure is able to affect the decision variables to also achieve push recovery features. Hence, no dedicated module is necessary. On the other side, stability of the hierarchical solution comes from two different sources. First, in the simplified model control layer, the VRP is used as an input to stabilize the DCM position around the desired DCM trajectory. In real world applications, this would be a dummy input, that is then moved to actual torques and joints in the whole body QP controller; in the current simulations VRP is used as an actual input. Second, an external module is necessary to ensure that the problem remains solvable whenever external disturbances are applied. Thus, the step adapter modifies future footsteps to ensure that the solutions coming from the simplified model control and the QP controller can be truly effective.

4.2 Simulation context

The two algorithms will be compared in a simulation regarding the movement of HRP4, a lightweight humanoid robot from the Humanoid Robotics Project [1]. All of its characteristics that are relevant to the simulation are reported in Tables 1, 2.

Dynamics			Timing	
Weight [kg]	CoM height [m]	ω [1/s]	T_{min} [s]	T_{max} [s]
39	0.75	3.6	0.2	0.65

Table 1: Dynamical quantities of the robot and timing constraints

Kinematics constraints				
ℓ [m]	$d_{a,x}$ [m]	$d_{a,y}$ [m]	$d_{z,x}$ [m]	$d_{z,y}$ [m]
0.2	0.5	0.05	0.08	0.08

Table 2: Kinematic constraints

In these the constrained quantities are reported in the notation of [3], but analogous ones are implemented in the other controller. In particular, ℓ refers

Step number	1	2	3	4	5
x position [m]	0	0.2481	0.4197	0.5916	0.7638
y position [m]	0.1	-0.1	0.1	-0.1	0.1
Time of impact [ms]	1	53	112	171	230
Step number	6	7	8	9	10
x position [m]	0.9334	1.1032	1.2733	1.4436	1.6141
y position [m]	-0.1	0.1	-0.1	0.1	-0.1
Time of impact [ms]	288	346	404	578	636

Table 3: Nominal position and timing of footsteps

to the nominal distance between feet on the y axis; $d_{a,x}, d_{a,y}$ are the maximum allowed distances on the x and y axis respectively from the nominal quantities; $d_{z,x}, d_{z,y}$ are constraints on the region of validity of the ZMP on the ground (they are equal to the dimension of the robot's foot).

It should be noted how the kinematic constraints on ZMP position in [7] are not present. Some constraints are only available in the Whole Body QP programming layer about contact wrenches. Hence, in the current implementation, it has not been implemented in the LIP. Thus, the simulation has no way of detecting that the desired VRP position is outside the support polygon. A discussion of possible issues regarding this topic is continued in Sec. 4.3.1.

The simulation task will be a simple horizontal walking motion starting from the origin and heading in the direction of the positive x axis. Both algorithms will start from the same nominal planned steps (obtained with the planner in the first layer of the Three-layered hierarchical architecture). Step timing and positions are reported in table 3.

This motion will be disturbed with an external push acting on the CoM on the sixth step. Two simulations with each controller will be carried out: one with horizontal push and one with vertical push. Both pushes will be applied at the sixth step for a duration of 0.10 seconds starting from 0.15 seconds of the previous footstep being completed (so as to push during single support phase). The pushes are set to $94N$ and $78N$ for the horizontal and vertical pushes respectively.

4.3 Results

Focusing on the horizontal push case (upper images), we may notice how IS-MPC (upper left) has a milder response to the push, requiring very little change to keep balance. In fact, the MPC-based controller only lengthens the step by a small amount with minimal variation along the y axis. Instead, the other architecture has both changes on the x and y axis: the next step is significantly narrower. Another difference is notable in timing. As depicted in Fig. 3, the hierarchical controller has a much more aggressive policy of reducing the step duration after the push. On the other side, IS-MPC can recover with only a small time adaptation. Both algorithms only require two more steps to get

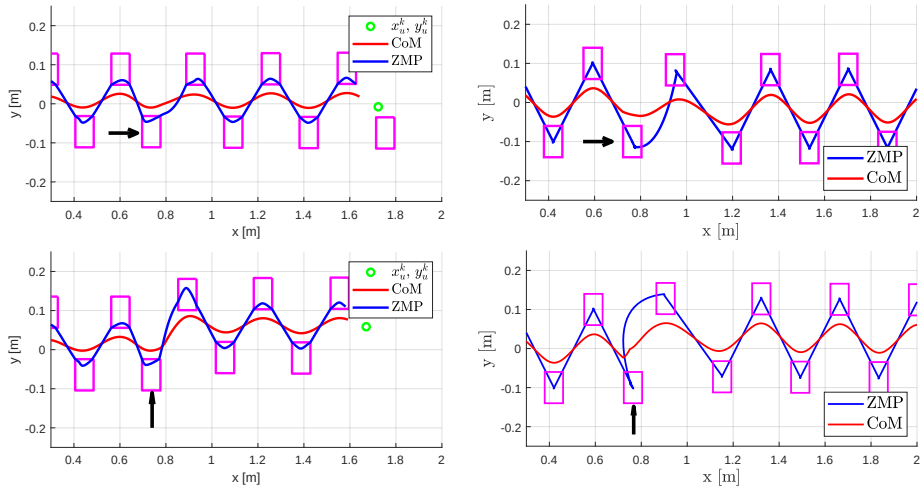

Figure 1: Push recovery along the x direction on top and on y direction on the bottom. Intrinsically stable MPC on the left and three layered hierarchical architecture on the right

back to nominal behavior. The only exception is that the hierarchical controller prefers to keep delaying steps by very small amounts for a few steps after the push.

Insight about the inner workings of the step adapter can be gained by looking Fig. 2. Here, we may notice how the initial solution for the DCMeos coming from the step adapter is very close to the original planned one of an horizontal walk. But, as soon as the push starts affecting the real DCM, the step adapter moves the desired DCMeos to a position that is lower on the y axis to better recover from the push on the x direction.

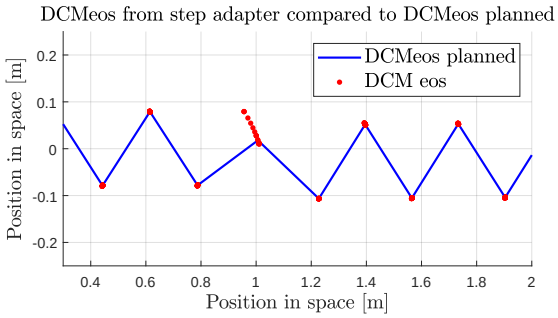

Figure 2: Evolution of the step adapter planned DCMeos after the push

Looking now at the vertical push simulations (bottom row of Fig. 1), the MPC-based controller (bottom left) is the one preferring to move the step sig-

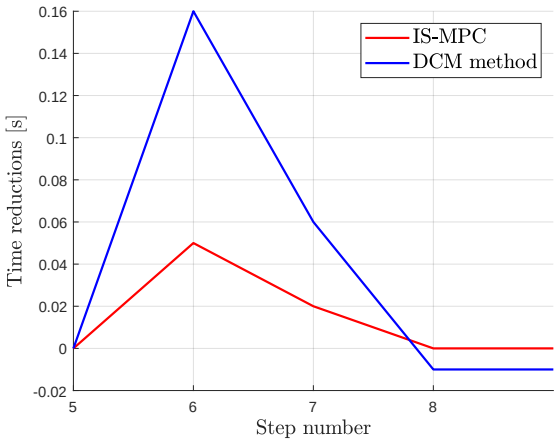

Figure 3: Time difference comparison between the two architectures when an horizontal push is applied to the sixth step

nificantly farther in the y direction. Another interesting difference is that the hierarchical controller prefers lowering the step length along the x axis to better be able to handle the push; the IS-MPC instead retains an x value close to the nominal one.

Comparisons on computational efficiency were not possible with the provided code. The implementation of the IS-MPC controller was an academic one that entailed the possibility of using many more decision variables than is feasible in real life. Hence, the long time to compute solutions[2] (around 5 minutes for 10 steps) cannot really be considered as a drawback of the algorithm. On the other side, performance of the hierarchical controller is more than satisfactory, being able to compute 10 steps in a few seconds. Of course, it should be kept in mind that the third layer (whole body QP control) is not being computed as the model of the robot is a simple LIP here. Moreover, the planner used in the current comparison is a quite simple one. It only plans steps in 2D obstacle-free space. Hence, real world implementation might have a more complex planner that requires more computational power.

It can be noted how the feet placement even before the push are different between the two cases. That is because, during the first few steps of motion, the IS-MPC method prefers to anticipate some steps before settling on steady motion. Modifying this would entail changing the cost function of the MPC and possibly changing its behavior during push recovery. The comparison can still be carried out if looking at the relative differences of subsequent steps between the two methods.

[2]All reported times are for laptop equipped with an intel Core i7-6500U processor with 12GB of DDR3L RAM

4.3.1 VRP boundaries

Particular attention should be pointed at the behavior of the VRP in the solution from [7]. In fact, as stated in Sec.4.2, no module is responsible for the feasibility of the VRP position. There is only a limit to forces expressed on the ground in one of the optimization problems defined in the whole body QP layer. When looking for the desired contact wrench F^* the choice is limited by the constraint

$$C_f F \leq 0 \tag{30}$$

which represents a set of linear relationships that limit the contact wrenches to the current foot on the ground. Our source [7] does not mention feasibility constraints on the VRP position anywhere else, and does not even report plots of the VRP in any simulation.

Thus, the code has no way to detect if the VRP position is feasible or not in the current implementation. In fact, in some simulations, the VRP position falls outside the support polygon of the feet in a way that would make a real robot fall. Looking carefully at plots in 1, it can be noted how in the case of an horizontal push force, the VRP of the three layered controller is just barely in the support polygon; while in the case of an horizontal force, it is clearly outside the support polygon. Hence, in the latter case, the push was not effectively recovered.

In general, the provided IS-MPC implementation is better at handling large pushes. In fact, the DCM-based method with the chosen hyperparamters is able to handle pushes of up to around $95N$ for horizontal pushes and around $70N$ for vertical ones. Both of these are easily recoverable for the IS-MPC method.

Another important point to keep in mind is that in the DCM method the re-planning step has to be computed each time the step adapter changes the next footstep. It is not enough to wait for the completion of the step for a complete re-plan, it has to happen immediately. In fact, even before the next footstep is completed, the DCM interpolation presented in 2.1.2 (from [6]), smooths the DCM trajectory to take into account the next step's DCMeos. Hence, if this is only updated at the time of impact, the smoothing trajectory instantaneously changes. This can result in a discontinuous VRP position when computed via the simplified model control layer. In fact, the newly computed VRP is often so far away as to be outside the support polygon.

Thus, it is a requirement for the planner to be fast enough that the plan can be updated quickly after the push. It should be noted that in this simulation environment, planning is fast enough that this is computationally possible: the algorithms still runs faster than real-time[3]. However, in a real-world application, the higher computation cost could become an issue (especially for a more complex planner that has to take 3D space and obstacles into account).

[3]Note how the whole body QP layer has no been implemented. Computing another more complex QP problem each step would significantly slow down the simulation.

5 Conclusions

Two possible architectures for controlling humanoid robot walking have been described. Both of these have been proposed with the objective of having robust walking motion capable of handling external disturbances (such as pushes on the robot body). A theoretical overview has been given on both methods. At the heart of the theoretical differences are the approaches to stability. IS-MPC is *recursively stable* by applying some special stability constraints to the MPC problem and recovers from pushes by modifying the nominal trajectory using the MPC. The hierarchical approach is stable from the inherent stability of the VRP control law that is used to make the DCM converge to a pre-planned trajectory. The latter method is also equipped with a step adapter that changes the planned footsteps in a way that makes the solution solvable feasible.

Some simulations have been carried out to highlight key practical differences. Here it was possible to note the generally more aggressive behavior of the hierarchical controller. In fact, it is the one that reduces step lengths more substantially and changes feet position on both axii when confronted with a push. Some erratic behavior of the VRP position in the hierarchical controller is also described.

Possible further work on similar issues would entail implementing both methods on a real world robot (or at least a robot simulator) to further highlight different behavior. It would be especially interesting to compare efficient implementation of both methods equipped with the same planner to better understand computational cost issues.

References

[1] Kenji Kaneko et al. "Humanoid robot HRP-4 - Humanoid robotics platform with lightweight and slim body". In: Sept. 2011, pp. 4400–4407. DOI: 10.1109/IROS.2011.6048074.

[2] J. Englsberger et al. "Trajectory generation for continuous leg forces during double support and heel-to-toe shift based on divergent component of motion". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 4022–4029.

[3] N. Scianca et al. "Intrinsically stable MPC for humanoid gait generation". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. 2016, pp. 601–606.

[4] Majid Khadiv et al. "A robust walking controller based on online step location and duration optimization for bipedal locomotion". In: *CoRR* abs/1704.01271 (2017). arXiv: 1704.01271. URL: http://arxiv.org/abs/1704.01271.

[5] Stefano Dafarra et al. "A Control Architecture with Online Predictive Planning for Position and Torque Controlled Walking of Humanoid Robots". In: *CoRR* abs/1807.05395 (2018). arXiv: 1807.05395. URL: http://arxiv.org/abs/1807.05395.

[6] Giulio Romualdi et al. "A Benchmarking of DCM Based Architectures for Position and Velocity Controlled Walking of Humanoid Robots". In: *CoRR* abs/1809.02167 (2018). arXiv: 1809.02167. URL: http://arxiv.org/abs/1809.02167.

[7] Milad Shafiee et al. "Online DCM Trajectory Generation for Push Recovery of Torque-Controlled Humanoid Robots". In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)* (2019), pp. 671–678.