Metapod : Template META-PrOgramming applied to the Dynamics

ZMP-COM trajectory dynamic filter

Summary

Introduction

Template Meta Programming for Inverse Dynamics

Dynamic Filter for CoM-ZMP walking based

Conclusion

Introduction

- **General issue :**
  - How to control a humanoid robot? (complex and redundant)

- **Basic Principles :**
  - Inverse Dynamic (ID) based control. Need the computation of the inertia matrix.
  - Linear Inverted Pendulum model Walking. Need the inverse dynamics to counteract the momentum due to the inertia of the bodies by :
    - ⇨ modifying the Center Of Masse (CoM) reference trajectory
    - ⇨ tracking the reference momentum

Introduction

- **Main constraint :**
  - Computation time
    - problem dimension i.e. 30DoF for HRP2
    - algorithm massively used (control period = 5ms * size of the preview window)
    - processor limited : 3GHz ; Moore Law ⟺ Multiple Core ➡ Clever algorithms.

- **Our challenge :**
  - Designing a C++ library capable of computing rapidly and efficiently the whole dynamic of a robot based on template programming

Template Meta Programming for Inverse Dynamics

Template Meta Programming for Inverse Dynamics

- (Featherstone 2008) Equations of the dynamics :

$$\boldsymbol{H}(model, \boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}$$

  - $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$: configuration vector
  - **H** : inertia matrix
  - **C** : matrix of the Coriolis effects
  - $\boldsymbol{\tau}$: forces applied on the robot

- The challenges, compute as fast as possible :
  - $\boxed{\boldsymbol{\tau}}$ using Recursive Newton Euler Algorithm (RNEA) with $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ as input
  - $\boxed{\boldsymbol{H}(model, \boldsymbol{q})}$ using Composite-Rigid-Body Algorithm (CRBA)

Template Meta Programming for Inverse Dynamics

- **Applications :**
  - Whole-body control (ID based control)
    - for high performance motion computing the inertia matrix at high speed can be crucial.
  - Walking
    - filter dynamically the CoM (Nishiwaki 2009).

⇒ for 30DoF : time(160*(RNEA)) < 200μs

⇔ 1.6s = preview window & control sampled at 200Hz

⇒ time(RNEA) = 1.25μs

Our results : 11.32μs on a i7-820QM processor running at 2.3 GHz.

Template Meta Programming for Inverse Dynamics

- **State of the art :**
  - Classical approach (Featherstone 2008): implement RNEA and CRBA using instance/structure and iterate along the kinematic tree.
    - RBDL (Rigid Body Dynamic Library)
    - KDL (Kinetic Dynamic Library),using C++ function overload
  - Symbolic computation : reduced model in simple language, C++ code generation (avoid useless computation => more efficient)
    - SYMORO+ (Khalil et al)
    - HuMAnS (Pierre-Brice Wieber)
    - METAPOD
    - ROBOTRAN

Template Meta Programming for Inverse Dynamics

- **Use of generated codes by formal computation :**
  - e.g. Inertia Matrix is sparse :
    - independent kinematic chain ⇨ null block matrix ⇨ no generation of code
  - Prefactoring, e.g. the factorial example
    - « factorial<4> » = « 24 »

- METAPOD uses the C++ template mechanism to do this kind of operation

Template Meta Programming for Inverse Dynamics

- **METAPOD : Template library**
  - Creation of a tree model in a template class "**Robot**"
  - Apply it on an instance **robot**, considering ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$)
  - Instruction in C++ :
    - *rnea< Robot, true >::run(robot, q, dq, ddq);*

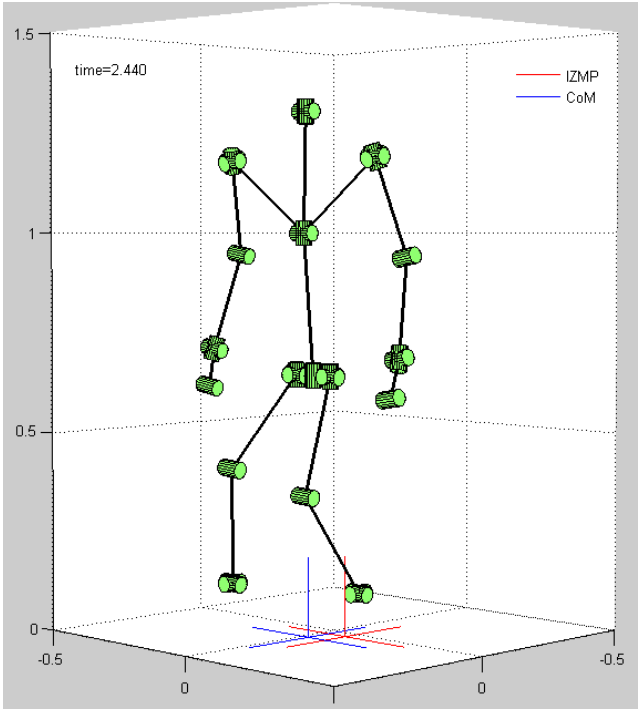
- ⇨ Algorithm adapted to **Robot** while the compilation
- ⇨ At running time update the inner variables with the instance **robot** and its state ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$)

Template Meta Programming for Inverse Dynamics

- Featherstone Algebra in meta-programming
  - Static polymorphism ⟺ simple implementation (no virtual methods needed)
  - Typed operator ⟺ Minimize the computation error
  - Increase of computation speed ⟺ Implementation of the all optimised algorithms of Featherstone
    - Taking in account the particular structures of a robot

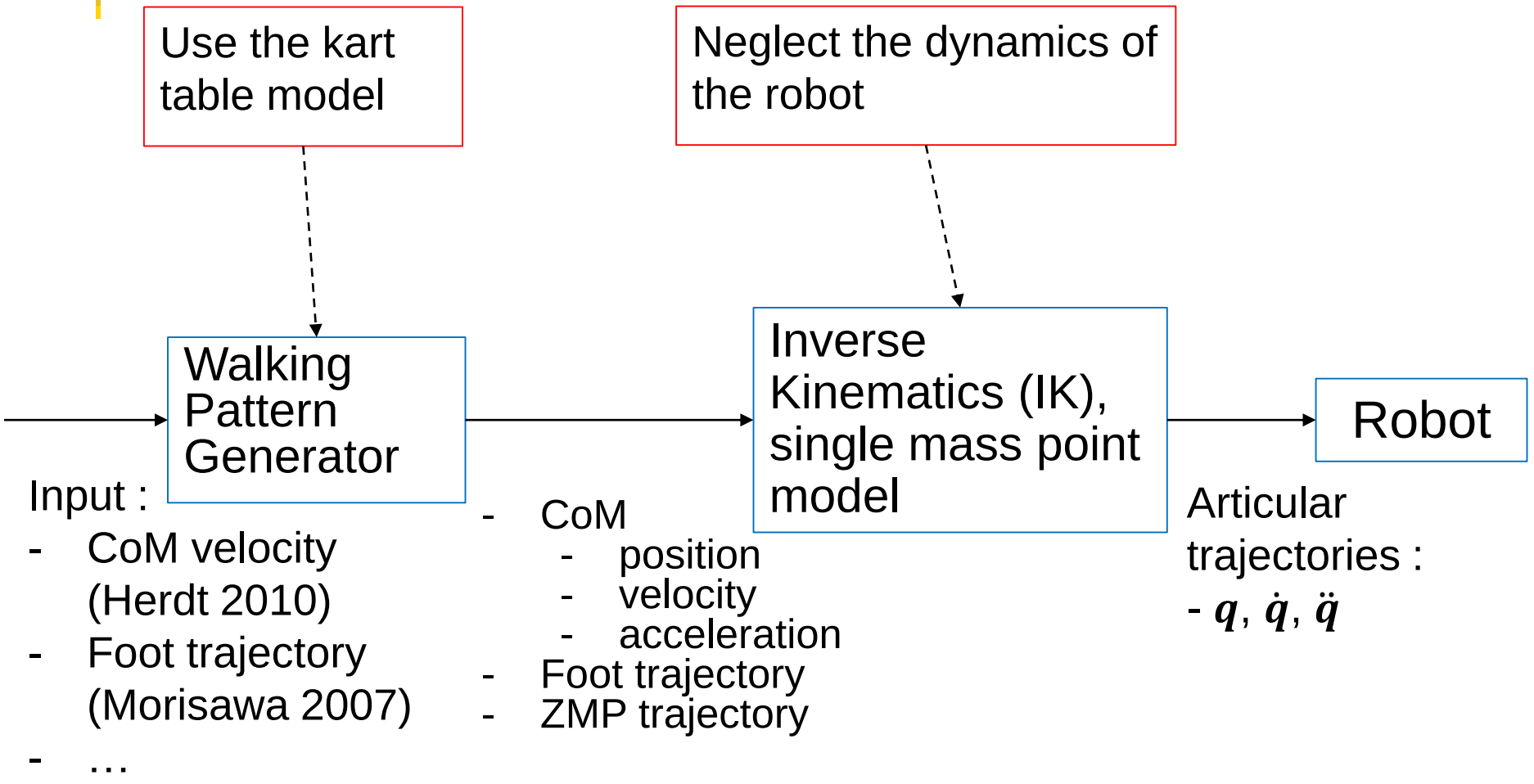
Template Meta Programming for Inverse Dynamics

- **Visitor Design Pattern for ID computation**
  - Process the kinematic tree while compile time
  - Combining with boost::fusion allow the decoupling model ⟺ instance



Dynamic Filter for CoM-ZMP walking based

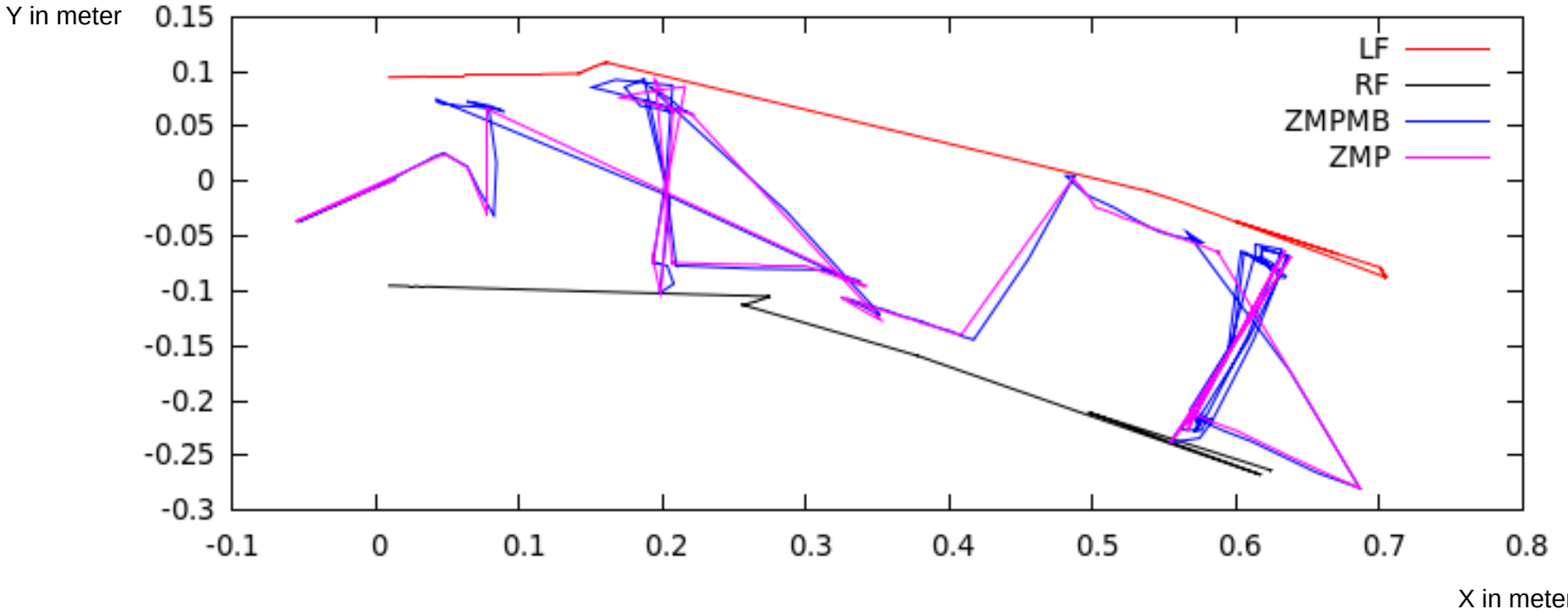
Dynamic Filter for CoM-ZMP walking based



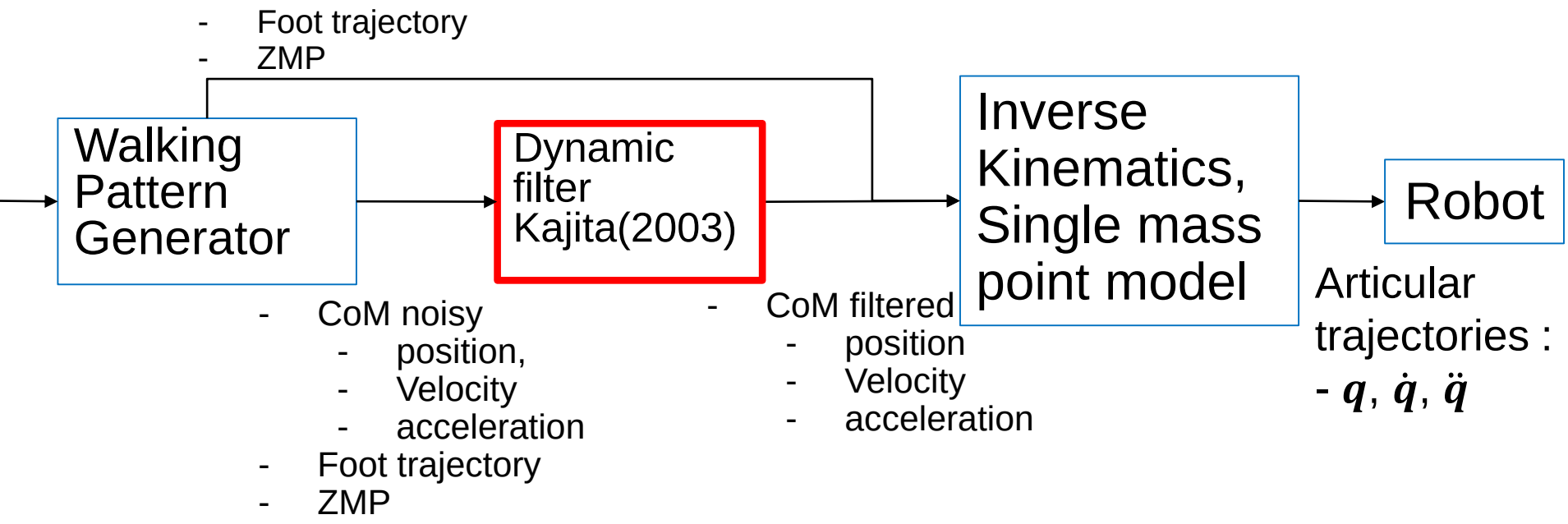
Dynamic Filter for CoM-ZMP walking based

ZMP is the reference trajectory
ZMP multibody is the ZMP trajectory computed from RNEA

Top view of the robot trajectory



Dynamic Filter for CoM-ZMP walking based



Dynamic Filter for CoM-ZMP walking based

- **The algorithm :**
  - The WPG define trajectories : CoM*, Feet*, ZMP*
  - $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = IK(single\_masse\_model, \boldsymbol{c}, \dot{\boldsymbol{c}}, \ddot{\boldsymbol{c}}, \boldsymbol{F})$
  - $(\boldsymbol{f}, \boldsymbol{\tau}) = ID(complete\_model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$
    - $ZMP_x^{MB} = -\frac{\tau_y}{f_z}$
    - $ZMP_y^{MB} = \frac{\tau_x}{f_z}$
    - $ZMP_z^{MB} = 0$
  - Evaluate the error on the ZMP over a preview window
    - $\Delta ZMP_k = ZMP_k^* - ZMP_k^{MB}$
  - Compute the equivalent error on the CoM
    - $\boldsymbol{\Delta CoM} = KajitaPreviewControl(\boldsymbol{\Delta ZMP})$
  - Correct the CoM :
    - $\mathbf{CoM} = \mathbf{CoM}^* + \mathbf{\Delta CoM}$

Dynamic Filter for CoM-ZMP walking based

- **Application on HRP-2**
  - HRP-2 control period : 0,005s
  - WPG (Herdt 2010) period : $WPG_T = 0.1s$
  - Preview window : $N_T = 1.6s$
  - Preview window sampling period : $RNEA_T = 0.05s$
  - $\Leftrightarrow$ WPG + Dynamic Filter fits in $F_T = 3ms$
- ***First results :***
  - $F_T = 2.4ms$
  - $\Rightarrow \frac{FT}{\frac{N_T \quad WPG_T}{WPG_T RNEA_T}} = \frac{2.4}{\left(\frac{1.6}{0.05}\right)} = 75\mu s$

Dynamic Filter for CoM-ZMP walking based

- **Comparison with other libraries :**
  - Computation of CRBA on HRP-2 :

  - Metapod : 7,25 µs
  - RBDL : 15 µs
  - Humans : 7 µs

    - Test realized on a i7-2820QM with 2.3 GHz
    - On Ubuntu 12.04.04 LTS, with g++ v4.6.3 and eigen v3.0.5.

Conclusion

- **Metapod :**
  - C++ library only (no symbolic software needed) and parse URDF robot description.
  - Compute efficiently the robots dynamic
  - Performance similar to library using symbolic methods
  - Its use can lead to create dynamic filter for the walking of humanoid robots

Thanks for your attention

- https://github.com/laas/metapod

- **Thanks to :**
  - Developers : Maxime REIS, Sébastien BARTHELEMY et Olivier STASSE
  - Additionnal hands : Martin FELIS et Antonio EL-KHOURY

- **The work on the dynamic filter has been done as part of :**
  - the European Project KOROIBOT FP7-ICT-2013-10/611909
  - And the projet OSEO ROMEO-2