METAPOD Template META-programming applied to dynamics: CoP-CoM trajectories filtering
Maximilien Naveau, Justin Carpentier, Sébastien Barthelemy, Olivier Stasse, Philippe Souères

▶ To cite this version:
Maximilien Naveau, Justin Carpentier, Sébastien Barthelemy, Olivier Stasse, Philippe Souères. METAPOD Template META-programming applied to dynamics: CoP-CoM trajectories filtering. International Conference on Humanoid Robotics, Nov 2014, Madrid, Spain. 10.1109/HUMANOID.2014.7041391. hal-01122475

HAL Id: hal-01122475
https://hal.archives-ouvertes.fr/hal-01122475
Submitted on 4 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
METAPOD - Template META-PrOgramming applied to Dynamics: CoP-CoM trajectories filtering.

M. Naveau¹, J. Carpentier¹, S. Barthelemy², O. Stasse¹, P. Soueres¹

Abstract—In this contribution, Metapod, a novel C++ library computing efficiently dynamic algorithms is presented. It uses template-programming techniques together with code-generation. The achieved performances shows some advantage over the state-of-the art dynamic library RBDL mostly on ATOM processor and for the inertia matrix computation, which are relevant for robotics application. On recent desktop computer, the ratio of the gain is not so obvious and in general the time achieved by both library is not significantly different for inverse dynamics. The advantage of this library is that it is open-source and does not rely on any external symbolic computational software. A main drawback is the increase complexity in debugging the code source due to template programming. Addionally we show how it can help in current control problems for humanoid robots, and more specifically for dynamic filtering of walking gait trajectories.

I. INTRODUCTION

Computing dynamic state estimate for a complex redundant robot such as humanoid robot is a crucial technique for controlling it. For instance, Inverse Dynamics control heavily relies on the computation of the inertia matrix [1], [2]. When considering balance, this matrix is of primary interest. Indeed most of the real-time approaches relies on the Linear Inverted Pendulum Model to generate a Center-Of-Mass reference trajectory which complies with the balance criteria provided by the Center-Of-Pressure and the contacts on the ground. Inverse Dynamics is necessary to counteract the inertial effect involved by the limbs: either by modifying the center-of-mass (CoM) reference [3], either by following the momentum reference trajectories. More advanced approaches are using Model Predictive Control (MPC) [4], [5] and need to compute the whole robot dynamic over a time horizon.

Since 2006, processor speed is limited to 3~4 GHz, because transistors have reached their physical limits mostly due to heat dissipation [6]. The Moore Law is still valid because as the chipssets integrate more cores, the transistor number continues to grow not because of their size, but because of their density. They are overheating when CPU clock is increased, and the focus is now more on concurrent software programming to take advantage of multiple cores. In the case of MPC, it mainly consists in having multiple instances starting with different initial solutions. Boosting the code running concurrently will also increase the performances. It is now necessary to implement code to compute dynamics efficiently while taking into account the underlying computing architecture.

A. Problem formulation

The rigid multibody dynamic model of the robot is assumed to be known and called model. According to [7], the dynamical equations can be written under the canonical form:

\[ H(\text{model}, \dot{q}) \ddot{q} + C(\text{model}, q, \dot{q}) = \tau \]  

(1)

where \( q, \dot{q} \) et \( \ddot{q} \) are respectively the generalized vectors for positions, velocities and accelerations. \( H \) is the generalized inertia matrix, \( C \) is the matrix representing the centrifugal, gravitational and Coriolis forces.

We should then compute as fast as we can:

- \( \tau \) by using the Recursive Newton-Euler Algorithm (RNEA) with the following inputs \( q, \dot{q} \) et \( \ddot{q} \).
- \( H(\text{model}, q) \) by using the Composite-Rigid-Body Algorithm (CRBA).

To represent position, velocities, acceleration, and forces, spatial algebra is nowadays one of the most used representation in robotics. Introduced by Roy Featherstone [7], it allows a correct mathematically algebra notation to represent the relationships between quantities of a rigid multibody system.

B. Applications

1) Walking: In the context of humanoid robotics, and following the method initiated by Nishiwaki et al. [3] for reactive walking, RNEA can be used to filter dynamically a Center-Of-Mass trajectory. Typically, for a humanoid robot, the goal is to compute 160 iterations for 30 DOFs robot in less than 200 \( \mu s \). This corresponds to a 1.6 s window for an optimal controller as the one used classically for the HRP-2 robot, with a 200 Hz sampling.

To reach this objective, a RNEA call should last no more than 1.25 \( \mu s \). The result achieved by our library is 11.32 \( \mu s \) on a i7-2820QM CPU of 2.3 GHz. This imply to use a subsampling mechanism such as the one proposed by Nishiwaki et al. [3].

One may argue that sub-sampling this trajectory can decrease the overall complexity for walking (Nishiwaki uses a 20 \( ms \) time step). However we anticipate a demand on such information with the recent increase in work considering extreme motion involving strong angular moments (such as the Parkour [8]). In such situation linearization will be only correct at high frequency.
2) Whole-body control: Staying in the context of humanoid robotics, and by now considering a control architecture based on the inverse dynamics [9], [1], computing the inertia matrix at a very high speed may be necessary. Following the previous remark on sub-sampling, this is probably not required for motion with a slow dynamic.

C. State of the art

A classical approach is to implement RNEA and CRBA using instances of classes and structures, and to iterate over each element of the kinematic tree. RBDL (Rigid Body Dynamic Library) and KDL (Kinematic Dynamic Library) are using this approach. RBDL is following specifically the directives suggested by Featherstone in his book [7] by implementing the spatial algebra through C++ operators overloading. A different approach is to use symbolic computation. One of the first software implementing this approach is Symoro+ by Khalil et al. [10]. The software is able to produce reduced models and to generate code where unnecessary computation are stripped out. SD/FAST is a commercially available toolbox used by Boston Dynamics and the MIT leg laboratory [11]. HuMAnS is a toolbox written by P.-B. Wieber [12] relying on Maple for manipulating the robot kinematic tree. ROBOTRAN is a JAVA application which performs symbolic manipulation and generates code integrated in MATLAB and the Simulink toolbox [13]. In each case, the robot model is written in a specific language to be simplified and finally generated in C++.

In Chapter 10 of his book, Featherstone indicates that a potential disadvantage of symbolic simplification is that it does not use vector-based arithmetic of CPUs, and that if the code is too large then it does not fit the computer’s instruction cache. In this work the first point is addressed by using Eigen, a widely known linear algebra library carefully written to exploit vector-based floating-point units. The second point is solved by using template programming which allows each instantiation of the template to invoke common code while avoiding indirect calls.

More generally, in this work, we propose to use meta-programming to exploit directly the C++ capabilities to perform symbolic computation at compile time without any additional library or software other than the BOOST libraries. The current solution is completed with a C++ program which is parsing URDF files and generates appropriate structures to decouple model and instance of robots. The current limitation of the approach is the case where model modification is needed. A solution (not yet implemented in the current software solution) is an abstract layer switching from an optimized implementation to a generic implementation (RBDL or KDL). Another more complex solution is to compile the new model and link dynamically to this new library.

Our contribution is to have written, to our knowledge, one of the first meta-programming library aiming at computing inverse dynamic for the robots. The application targeted in this paper is humanoid robot walking. By putting together the power of functional programming with C++ speed, our library is able to achieve performance necessary for real-time control.

II. Template metaProgramming for Inverse Dynamics

This work takes its root from the fact that code generated by symbolic formal computation has in general better performances than generic code for complex robotic models. This comes for the inner structure of the problem itself. For instance, the robot inertia matrix is sparse because the sub-blocks matrices correspond to the kinematic branches. Then when no coupling exists between some branches, the related sub-blocks mainly is equal to zero. Formal computation detects such branches and suppresses the related computation. We are proposing to make the same kind of operations by exploiting C++ template meta-programming.

The result is a template library able to develop an optimized computational tree to evaluate the inverse dynamics of a robot class named Robot and to apply it onto the specific instance named robot, while considering the following complete state \((q, \dot{q}, \ddot{q})\). During execution, the model specificities are used to update robot internal variables according to \((q, \dot{q}, \ddot{q})\).

A. Simple example of functional programming using C++

A famous example of functional programming in C++ is the factorial computation:

```cpp
template <int n> struct factorial {
  enum { value=n*factorial<n-1>::value};
};

template <> struct factorial<0> {
  enum { value = 1 }; 
};
```

At compile time, when the compiler finds `factorial<4>` it directly computes 24 by successive substitutions. The same type of mechanism is used to develop the RNEA algorithm from the model Robot. In order to achieve a decoupling between the model and the robot instances Metapod is using a template based container proposed by boost::fusion which allows an array like syntax with templates \[\text{[\text{template}} \text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{"template <int n> struct factorial {
  enum { value=n*factorial<n-1>::value};
};

template <> struct factorial<0> {
  enum { value = 1 }; 
};
```

A naive description of the approach is available at https://github.com/laas/metapod/wiki/Naive-implementation---First-approach

1 A naive description of the approach is available at https://github.com/laas/metapod/wiki/Naive-implementation---First-approach
parts of the robot are planar. One of the unitary test used in the library available through Github is the planar arm described in [14].

C. Visitor design pattern to compute inverse dynamic

The depth first exploration of the robot structure is performed thanks to the visitor design pattern. This design pattern has been used in all algorithms where an iterative exploration of the tree structure is necessary. Finally, thanks to the design pattern coupled to boost::fusion which make possible to aggregate different types, it is possible to decouple the model from its instance.

D. Benchmarks

We compared the performances of Metapod with RBDL by computing the inertia matrix and the inverse dynamics on the CPUs listed in Table I. First we would like to emphasize that the numbers are only a snapshot and are subject to modifications, since the writing of this paper numerous improvements have been found. Some are currently tested for RBDL and for Metapod lead to the writing of a new library. The reader is invited to look at the link given in the footnote for further discussion. In each case, the library was recompiled using the GNU gcc compiler with the optimization options (-O3 -NDEBUG) trying to generate the code producing the best performances. The model used for benchmarking is the sample humanoid model provided by Tokyo University in OpenHRP. For this specific work we concentrated in two algorithms which are used in controlling a humanoid robot, the Composite Rigid Body Algorithm (CRBA) which computes the inertia matrix, and the Recursive Newton-Euler Algorithm (RNEA) which provides the torques necessary to counterbalance the gravity.

Table II provides a quick view on the performance difference between Metapod and RBDL on various CPUs. The next section will detail some aspects related to the CPUs which will try to explain that the values provided here are to be taken cautiously and are likely to highly vary according to the CPU, the context of its overload, the compiler and the operating system. For this reason, we provide in Table II a ratio according to the CPU between the time provided by Metapod and RBDL. Note that CPU-2 corresponds to the HRP-2 real chipset with its real-time operating system. The code and its data are then prefetched from the memory while the CPU is decoding the current operation. The prediction of the branch is done thanks to the branch predictor. Branch prediction is also able to indirect branch where multiple intermediate variables to avoid costly access memory. Branch predictors can also detect cycles and save in cache the statistics of the branch, the predictor tries to guess what is the next instruction to predict the branches behavior. Depending on the statistics of the branch, the predictor tries to guess what will be the most probable code to be executed by the CPU. The code and its data are then prefetched from the memory while the CPU is decoding the current operation. Branch predictors can also detect cycles and save in cache intermediate variables to avoid costly access memory. Branch predictors is also able to indirect branch where multiple address are possible targets. The interest of the symbolic approach is to avoid as much as possible to make branches in the code. It facilitates the work of the branch predictor as well as the memory pre-fetching mechanism. This may explain why in Table II Metapod provides a 25-30 % increase in efficiency on the CRBA algorithm. Unfortunately chipsets providers, in general, do not give a detailed de-
Table I
CPUs used for the benchmarking. CPU-1 is a laptop, CPU-2 is one used in HRP-2, CPU-3 is a desktop, and CPU-4 is a NetPC. CPU-2 is using only one core due to the real-time operating system recognizing only one core.

<table>
<thead>
<tr>
<th>Nickname</th>
<th>CPU name</th>
<th>Frequency</th>
<th>Core number</th>
<th>Cache size</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU-1</td>
<td>Intel(R) Core(TM) i7-2820QM CPU</td>
<td>2.3 GHz</td>
<td>8</td>
<td>8 Mb</td>
<td>Ubuntu 12.04 LTS</td>
</tr>
<tr>
<td>CPU-2</td>
<td>Intel(R) Core2(TM) Duo E7500</td>
<td>2.8 GHz</td>
<td>1</td>
<td>3 Mb</td>
<td>Ubuntu 10.04 LTS</td>
</tr>
<tr>
<td>CPU-3</td>
<td>Intel(R) Xeon(R) CPU E3-1240 v3</td>
<td>3.4 GHz</td>
<td>8</td>
<td>8 Mb</td>
<td>Ubuntu 12.04 LTS</td>
</tr>
<tr>
<td>CPU-4</td>
<td>Intel(R) Atom(TM) CPU N720</td>
<td>1.6 GHz</td>
<td>1</td>
<td>512 Kb</td>
<td>Ubuntu 12.04 LTS</td>
</tr>
</tbody>
</table>

Table II
Processing time for three basic algorithms used in control of humanoid robot: Inertia matrix computation (CRBA), Inverse Dynamics (RNEA). The computation have been tested on 4 CPUs.

<table>
<thead>
<tr>
<th>Library</th>
<th>Algorithms</th>
<th>CPU-1</th>
<th>CPU-2</th>
<th>CPU-3</th>
<th>CPU-4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>µs</td>
<td>ratio</td>
<td>µs</td>
<td>ratio</td>
</tr>
<tr>
<td>Metapod</td>
<td>CRBA</td>
<td>7.89</td>
<td>0.77</td>
<td>12.97</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>RNEA</td>
<td>9.28</td>
<td>0.97</td>
<td>13.85</td>
<td>0.75</td>
</tr>
<tr>
<td>RBDL</td>
<td>CRBA</td>
<td>10.16</td>
<td></td>
<td>18.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RNEA</td>
<td>9.5</td>
<td></td>
<td>18.66</td>
<td></td>
</tr>
</tbody>
</table>

Table III
Ir: Number of executed instructions, IlmIr: Instruction read misses for cache L1, IlmL: Instruction read misses for last level cache, Dl: Number of cache reads, Dlmr: Data read misses for cache L1, DlLmr: Data read misses for last level cache, Dw: Number of cache write, Dlmw: Data write misses for cache L1, DlLmw: Data write misses for last level cache. Tests realized on CPU-1.

<table>
<thead>
<tr>
<th>Library</th>
<th>Ir</th>
<th>IlmIr</th>
<th>IlmL</th>
<th>Dl</th>
<th>Dlmr</th>
<th>DlLmr</th>
<th>Dw</th>
<th>Dlmw</th>
<th>DlLmw</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBDL</td>
<td>8,295,618,744</td>
<td>39,643</td>
<td>3,095</td>
<td>3,444,245,971</td>
<td>2,936,723</td>
<td>7,322</td>
<td>1,017,919,310</td>
<td>607,848</td>
<td>3,447</td>
</tr>
<tr>
<td>Metapod</td>
<td>4,529,574,180</td>
<td>69,802,287</td>
<td>2,722</td>
<td>2,154,169,130</td>
<td>9,709,886</td>
<td>5,040</td>
<td>1,034,216,699</td>
<td>27,501,390</td>
<td>1,136</td>
</tr>
</tbody>
</table>

Table IV
Bc: Conditional branches executed, Bcm: Conditional branches mispredicted, Bi: Indirect branches executed, Bim: Indirect branches mispredicted. Tests realized on CPU-1.

<table>
<thead>
<tr>
<th>Library</th>
<th>Bc</th>
<th>Bcm</th>
<th>Bi</th>
<th>Bim</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBDL</td>
<td>245,662,040</td>
<td>7,604,170</td>
<td>45,177,857</td>
<td>269,054</td>
</tr>
<tr>
<td>Metapod</td>
<td>98,901,411</td>
<td>977,079</td>
<td>14,105,063</td>
<td>726</td>
</tr>
</tbody>
</table>

III. Dynamic filtering for walking
To illustrate the use of Metapod on a humanoid robot, we consider an application where the robot has to evolve in a factory. AIRBUS/Future of the Aircraft factory is currently evaluating the potential of humanoid robots in such context. The use case considered here is the HRP-2 humanoid robot bringing an electric screw driver to an assembly line. On the other hand, in the context of the Koroirobot project aiming at studying human walking to improve humanoid robot motion, we have created a stair climbing motion primitive. The behavior consisting in bringing a tool while climbing the stairs is then a combination of the two motion primitives. We show the description of their branch predictors. For this reason, users have usually to reverse engineer the behavior of the chipset in order to create branch predictor aware compilers [16], [17]. We used Valgrind framework to analyse cache access and branch prediction. The result is compiled in Table [III] and in Table [IV] From Table [III] Metapod is doing more misprediction than RBDL for the Level 1 cache both for instruction and data, but less for the last level cache. As a result misprediction implies a memory access, with the slower bus frequency it is the most costly. Therefore this result is in favor of Metapod. In addition Table [IV] shows clearly that branch misprediction is far lower for Metapod than for RBDL.

3) Compiler: The importance of the compiling options differs according to the library and the CPUs. It was particularly strong on the Atom chipset with a gain of 20µs for Metapod and RBDL. On RBDL, the impact of compiling option was null on CPU-1 and CPU-3 whereas it is improved by 10% on Metapod. The compiler options used to exploit vector-based arithmetic and loop optimizations are -msse -msse2 -msse3 -march=corei7 -mfpmath=sse -fivopts -ftree-loops -ftree-vectorize -fipa-pta. In addition during the linking phase it is possible to strip out useless code and perform whole-program optimization with the following option: -fwhole-program. Again the static structure of the code generated through Metapod simplifies the work of optimization strategies applied by the compiler (here gcc).
next that, thanks to Metapod, the dynamical consistency can be realized in real time.

A. Dynamic filtering on a sub-sampled walking pattern generator

This work is based upon the real-time walking control system described by Nishiwaki in [3]. One key ingredient is to generate real-time dynamically stable CoM and Center-of-Pressure (CoP) trajectories using the cart-table model [18]. Then to correct the inertial effects induced by the legs movement Nishiwaki uses a subsampled dynamical filter. Indeed, the results tested on HRP-2 have shown that the CoP trajectory followed by the robot are not the desired one, the difference can be as big as 15 mm. To avoid this problem, the solutions consist in designing a controller regulating the linear and angular momentum [19] in addition to a dynamical filter aiming at correcting the CoM trajectory in order to take into account the dynamics of the robot. The first kind of solution is mostly used to reject instantaneous perturbations. However, it uses additional degrees of freedom that could have been used for manipulation tasks for example. For this reason, in our specific case, we have only implemented the dynamical filter as the commercially available stabilizer integrate already a perturbation rejection strategy. In the following the dynamic filter depicted by Nishiwaki and implemented in this paper is described in details:

- The walking pattern generator (WPG) defines a trajectory for the CoM, the feet and the CoP. The desired CoP trajectory is noted CoP∗.
- The dynamic filter starts by computing the joint trajectories using the analytical inverse kinematics. We make the assumption that the CoM and the free flyer are rigidly connected.

\[
(q, \dot{q}, \ddot{q}) = IK(pontual\_model, c, \dot{c}, \ddot{c}, X^f)
\]

with \( c, \dot{c}, \ddot{c} \) et \( X^f \) being respectively the position, the velocity, the acceleration of the CoM and the feet position.

- With the joint trajectory, it is possible to compute the inverse dynamics and to find the CoP matching up to the real robot motion. We call it the multi-body CoP noted CoP^{MB}.

\[
(f, \tau) = ID(model\_complet, q, \dot{q}, \ddot{q})
\]

\[
CoF^x^{MB} = -\frac{x\tau}{mg}
\]

\[
CoF^y^{MB} = \frac{\dot{Z}}{mg}
\]

\[
CoF^z^{MB} = 0
\]

\[
\Delta\text{CoP} = \text{CoP}^* - \text{CoP}^{MB}
\]

- This provides an error between CoP∗ and CoP^{MB}. This error is computed over a time window and is injected in a preview control (PC) in the shape of an LQR described by Kajita and al. [18]. The result of this step is an error for the CoM in position, velocity and acceleration.

\[
\Delta\text{CoM} = PC(\Delta\text{CoP})
\]

- We can then sum this error on the reference CoM (CoM∗) to correct the trajectory.

\[
\text{CoM} = \text{CoM}^* + \Delta\text{CoM}
\]

With CoM = \([c \quad \dot{c} \quad \ddot{c}]^T\).

- Finally, the new CoM trajectory is used to compute the joint trajectory using the Inverse Kinematics.

This dynamic filter is based on the Newton-Raphson algorithm using the CoM trajectory as free variables. This method does not guarantee the convergence. However during tests on HRP-2 with other WPG, we observed that the CoP trajectory is corrected in one iteration. The goal is to implement this method on the WPG of Andrei Herdt [20] and Morisawa’s [21].

B. Application on HRP-2

We propose to use this filter on the WPG proposed by Morisawa and al. in [21]. This algorithm uses an analytical form of the CoM and CoP trajectories assuming that the later is a third order polynomial. The Morisawa’s WPG can either define off line the whole trajectory at once, or change online the foot steps given as reference.

To use the filter with its full potential on the online Morisawa’s WPG we need to compute the trajectory over 1.6 s in advance, i.e. two steps forward. The computation of the trajectory is done every 5 ms including the control, so the remaining time for the WPG is 3 ms. Moreover, because the robot HRP-2 is controlled at 200 Hz, we need to compute 320 RNEA to use the whole preview. Our implementation, as fast as it is, does not allow this kind of computation, hence we used a strategy proposed by Nishiwaki, i.e. we sub sampled the preview window at a period of 0.05 s which leads to 32 RNEA every 0.1 s. We rather compute the whole dynamic filter for the offline WPG of Morisawa.

This study is based on the offline WPG. We computed 1.6 s more than the initial trajectory and we computed the multi-body CoP for each sampling time.

The results obtained are illustrated in the Fig. 1. The graphs here contains each :

- We do not hallucinate.
the trajectories of the reference CoP,
the multi-body CoP computed with RNEA,
and the corrected CoP obtained with the RNEA using
the corrected CoM
The upper graph depicts the evolution on the axis $X$ in
function of the time and the lower graph shows the evolution
of the $Y$ axis in function of the time. This data are extracted
from a straight walking using the offline WPG of Morisawa.
We can here easily see the influence of the filter, i.e. the
corrected multi-body CoP is almost fused with the reference
one. In terms of distance, the maximum error between the
reference CoP and the multi-body CoP is 15.3 mm while the
maximum distance after correction is 1.7 mm. In average the
error before correction is 5.7 mm and after it is 0.7 mm.
In term of computation time, we executed the algorithm
that creates a straight flat walking on the HRP-2 robot. The
results are 15.0 us for the average time used to compute
the RNEA during the whole trajectory and 342.0 us for
the maximum duration of one iteration of RNEA. This
peak corresponds to the first iterations of the algorithm,
corresponding to memory allocation.
We have done a series of experiments using the dynamic
filter including the straight walking and climbing stairs. For
the climbing we had four scenarios: one simple walk, a walk
while swinging the arm and holding a lightweight tool in
one hand, another one holding the tool with 2 hands. In
the last one the tool is hold with the hands lifted over the
head. HRP-2 went through all the pattern has depicted in
the companion video. We also succeeded to make the robot
going down the stairs with the lightweight in two hands,
however we did not manage to make HRP-2 perform the
rest of the scenario while going down. One of the reasons
that explain this behaviour is that the dynamic filter places
the CoM in a way such that the CoP is inside the support
foot. To compensate for the arms being in front, the CoM is
send slightly backward. The leg is then stretched and create
a singularity. This shows the limit of this approach which calls
for a more advanced development such as optimal control,
whole-body predictive control or planning.
In this section we have shown that one iteration of a
Newton-Raphson algorithm can make the CoP converge to-
wards its reference and it could be applied to WPG focusing
on the under-actuated part of the robot.

IV. CONCLUSION
We have presented a C++ library for efficient humanoid
dynamic computation without relying on any particular sym-

dolic computational tool. The achieved performances shows
some advantage over the state-of-the art dynamic library
RBDL mostly on ATOM processor and for the inertia matrix
computation, which are relevant for robotics application. On
recent desktop computer, the ratio of the gain is not so
obvious and in general the time achieved by both library
is not significantly different for inverse dynamics. Some
preliminary results show that we have a computational speed
similar to symbolic computation. This is achieved through a
mix of code generation and C++ meta-programming. Its use
is established in the context of dynamic filtering for real-time
walking gait generation, and more specifically applied to the
humanoid robot HRP-2. The library is available under GPL-
license at the following url: https://github.com/laas/metapod

REFERENCES

humanoids with short cycle pattern generation,” The International
based on online optimization,” in ICHR, 2013, pp. 21–27.
to binary logic switch scaling a gedanken model,” Proceedings of the
2008.

sdfast/index.html
humans toolbox, a homogeneous framework for motion capture,
analysis and simulation,” in Ninth ISB Symposium on 3D analysis
Available: http://www.robottran.de
[15] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weiss-
mann, “Power-management architecture of the intel microarchitecture
code-named sandy bridge,” Micro, IEEE, vol. 32, no. 2, pp. 20–27,
March 2012.
[16] Z. Wang and D. A. Jiménez, “Program interferometry,” in Interna-
tional IEEE International Symposium on Workload Characterization
[17] V. Uzelac and A. Milenkovic, “Experiment flows and microbench-
marks for reverse engineering of branch predictor structures,” in IEEE
International Symposium on Performance Analysis of Systems and
Software (SPASS), 2009, pp. 207–217.
i, and H. Hirukawa, “Biped walking pattern generation by using preview
control of zero-moment point,” in ICRA, September 2003, pp. 1620–
1626.
[19] ——, “Resolved momentum control: Humanoid motion planning
based on the linear and angular momentum,” 2003.
M. Diehl, “Online walking motion generation with automatic footstep
[21] M. Morisawa, K. Harada, S. Kajita, S. Nakaoaka, K. Fujiwara, F. Kane-
hiro, K. Kaneko, and H. Hirukawa, “Experimentation of Humanoid
Walking Allowing Immediate Modification of Foot Place Based on

ACKNOWLEDGMENT
The Metapod library was made possible thanks to the work
of Maxime REIS. Antonio EL-KHOURY took an active part
in the improvement of Metapod. We warmly thank them.