Modular Architecture for Humanoid Walking Pattern Prototyping and Experiments

Olivier Stasse[1], Björn Verrelst[3], Pierre-Brice Wieber[4], Bram Vanderborght[3],
Paul Evrard[1], Abderrahmane Kheddar[1] and Kazuhito Yokoi[2]

[1] *JRL, CNRS-ST2I,* [2] *JRL, AIST-IS,*
AIST-Central 2 Umezono 1-1-1, Tsukuba, Ibaraki 305-8568
{*olivier.stasse,paul.evrard,abderrahmane.kheddar,kazuhito.yokoi*}*@aist.go.jp*
[3] *Vrije Universiteit Brussel, Belgium*
{*bjorn.verrelst, bram.vanderborght*}*@vub.ac.be*
[4] *INRIA - Rhone-Alpes, Grenoble, France*
pierre-brice.wieber@inrialpes.fr

Abstract

In this paper we describe the use of design patterns as a basis for creating a Humanoid Walking Pattern Generator Software having a modular architecture. This architecture made possible the rapid porting of several novel walking algorithms on a full size humanoid robot HRP-2. The body of work currently available allows extracting a general software architecture usable with inter-exchange between simulations and real experiments. The proposed architecture with the associated design patterns are described together with several applications: a pattern generator for a HRP-2 with passive toe-joints, a pattern for dynamically stepping over large obstacles, and a new quadratic problem (QP) formulation for the generation of the reference ZMP. Thanks to the versatility and the modularity of the proposed framework, the QP method has been implemented and experienced within four days only.

keywords: design pattern, walking pattern generator, humanoid robot

1 Introduction

The field of research on humanoid robots is becoming a competitive research and development area asking for reliable and very fast software development time, and integrated on complex and sophisticated hardware. Interestingly the most advanced teams regarding impressive experimental scenarios in this area are two companies: HONDA [24] and SONY [20]. On the other hand the need for researchers to create new algorithms is usually at odd with the time needed for developing full scale reliable software.

The Japanese government acknowledged the need for components widely available to the research community and fostered the creation of common simulation tools such as OpenHRP [16], and more recently through RTM [2] for the control part. Regarding the use of middleware and robotic applications, there are already solutions with a proven set of real life experiments such as CLARAty for NASA's rovers [8]. To populate those high-level robotic application tools such as there is a need to design modular architectures specific to robotic applications. They are necessary especially to rapidly design and test algorithms. This work proposes such a tool particularly for humanoid robotics. This paper is organized as follows: first recent walking pattern generation algorithms for humanoid robots are presented. This is followed by a common global architecture and its related implementation presented altogether with applied software engineering techniques. The efficiency of the proposed architecture is exemplified by three innovative works regarding walking pattern generators and validated with real experiments.

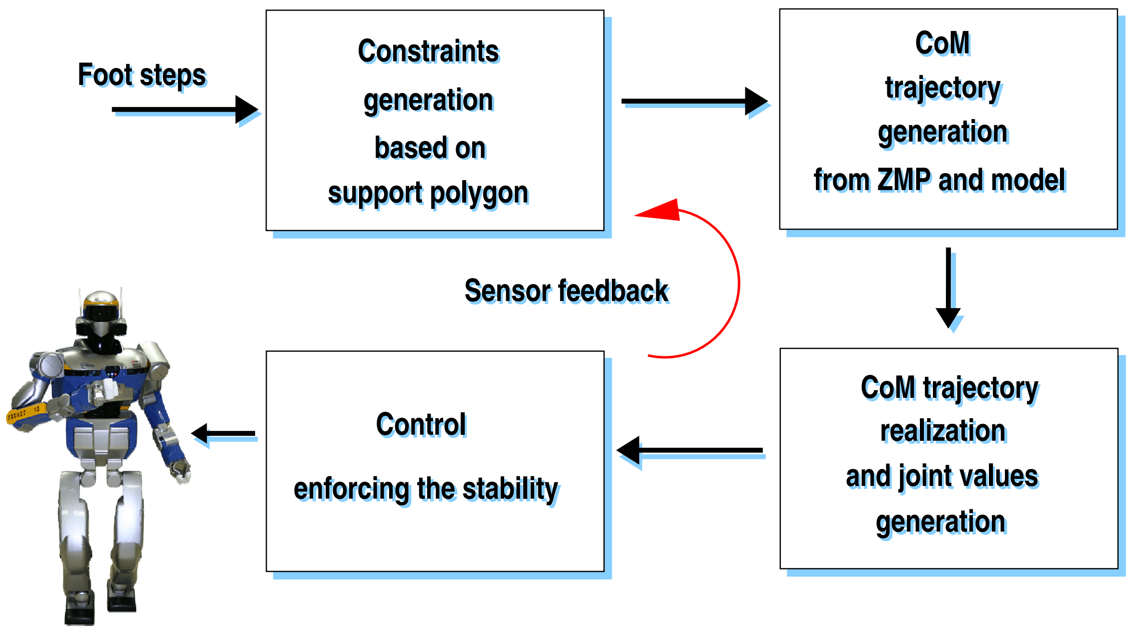
Foot steps
Constraints generation based on support polygon
CoM trajectory generation from ZMP and model
Sensor feedback
CoM trajectory realization and joint values generation
Control enforcing the stability

Figure 1: General structure of a Humanoid Walking Pattern Generator.

2 Humanoid Walking Pattern Generator (HWPG)

2.1 Introduction

In this section, the fundamental principles of a Walking Pattern Generator for humanoids which have been experimented on real robots are presented. They are summarized by the scheme depicted in figure 1. From a set of foot steps a set of constraints is generated. Those constraints are used to compute a trajectory for the humanoid Center of Mass (CoM) using a model of the robot that integrates explicitly a stability criterion. This trajectory is used to compute a set of desired joint values, which are passed

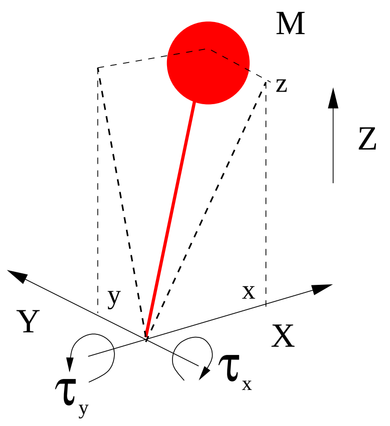

Figure 2: 3D Inverted Pendulum Model.

to a controller which guarantees that the stability criterion remains valid against external disturbance.

2.2 Simplified model

Recent popular real-time pattern generators for humanoid robots rely mostly on a point mass model called the 3D Inverted Pendulum [13] depicted in figure 2. The CoM motion equations are thus:

$$\begin{aligned} M(-z\ddot{y} + y\ddot{z}) &= \tau_y - Mgy \\ M(-z\ddot{x} + x\ddot{z}) &= \tau_x - Mgx \end{aligned} \tag{1}$$

where (x, y, z) is the position of the CoM, M is the total mass of the robot, g the gravitational acceleration field, and τ_x and τ_y are the torques applied to the foot point. Kajita [15] proposed to add a constraint resulting on a linearization of the model and named it the 3D Linear Inverted Pendulum Model (LIPM). The constraint forces the motion of the CoM along the z axis to be along a plane specified by a normal vector $[k_x, k_y, -1]$:

$$z = k_x x + k_y y + z_c \tag{2}$$

where z_c is a constant height offset of the CoM. The 3D pendulum equations become:

$$\begin{aligned} \ddot{y} &= \frac{g}{z_c}y - \frac{k_x}{z_c}(x\ddot{y} - \ddot{x}y) - \frac{1}{Mz_c}\tau_y \\ \ddot{x} &= \frac{g}{z_c}x + \frac{k_y}{z_c}(x\ddot{y} - \ddot{x}y) - \frac{1}{Mz_c}\tau_x \end{aligned} \tag{3}$$

By setting k_x and k_y to zero, i.e. having the plane horizontal, the system simplifies to:

$$\begin{aligned} \ddot{y} &= \frac{g}{z_c}y - \frac{1}{Mz_c}\tau_y \\ \ddot{x} &= \frac{g}{z_c}x - \frac{1}{Mz_c}\tau_x \end{aligned} \tag{4}$$

This constraint is equivalent to set $\dot{\mathcal{L}}_x = \dot{\mathcal{L}}_y = 0$, where $\mathcal{L}_x$ and $\mathcal{L}_y$ are the moment around the CoM for respectively the x and y axis. It avoids any tilting motion of the body. In fact, the main advantage of this constraint is to decouple the link between forward and sideways motions of the robot. From this simplified motion, it is possible to take explicitly into account a stability criterion.

2.3 Stability criterion

A popular stability criterion for walking is the Zero Momentum Point (ZMP) which is also equivalent to the Center of Pressure (CoP) when the ground is a horizontal planar surface. This criterion considers the vertical resultant of the force applied to the feet, assuming it is horizontal. It states that if the ZMP is inside the contact support polygon then the robot is dynamically stable. The relationship between the ZMP and the CoM's motion is given by [11]:

$$p_x = \frac{Mgx + p_z\dot{\mathcal{P}}_x - \dot{\mathcal{L}}_y}{Mg + \dot{\mathcal{P}}_z} \tag{5}$$

$$p_y = \frac{Mgy + p_z\dot{\mathcal{P}}_y + \dot{\mathcal{L}}_x}{Mg + \dot{\mathcal{P}}_z} \tag{6}$$

with $\mathcal{P}$ and $\mathcal{L}$ being respectively the linear and angular momentum. When considering the previous constraint 4, equations 5 and 6 simplify to:

$$\begin{aligned} p_x &= x - \frac{z_c}{g}\ddot{x} \\ p_y &= y - \frac{z_c}{g}\ddot{y} \end{aligned} \tag{7}$$

2.4 CoM trajectory generation

Assuming that the ZMP trajectory can be formulated by a polynom; there exists an analytical formulation of the constrained dynamical system. Nagasaka solved an optimization function to generate whole body motion for QRIO [20] by specifying the starting and end conditions for each phase of the walking, i.e. simple and double support phases. Using a similar approach, Harada in [10] suggests to connect polynomial trajectories of ZMP by using an analytical solution of the ZMP equations.

A different approach consists in taking into account the future by integrating a preview window [11]. Assuming the future steps are known, it is possible to deduce a set $\{p_0, ..., p_{N_L-1}\}$ of future ZMP positions. The main issue is to solve the inverse problem which finds a CoM trajectory from the desired ZMP trajectory. For a given axis, e.g. x, this can be achieved by solving the following optimization problem:

$$\min_{\dddot{x}(k),\cdots,\dddot{x}(k+N_L)} \sum_{i=k}^{k+N_L-1} \frac{1}{2}Q(p_x(i+1) - p_x^{ref}(i+1))^2 + \frac{1}{2}R\dddot{x}^2(i) \tag{8}$$

where the first term represents the difference between the actual ZMP and the desired one, and the second term is the jerk of the CoM. Q and R are weights influencing the importance of each part in the cost function. Using an iterative scheme, and assuming that both CoM and ZMP trajectories are discretized by piecewise cubic polynomials over time intervals of constant length T, the following recursive relationship is obtained:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\dddot{x}(k), \\ p(k) &= \mathbf{C}\mathbf{x}(k), \end{aligned} \tag{9}$$

where

$$\mathbf{x}(k) \equiv [x(k)\ \dot{x}(k)\ \ddot{x}(k)]^T,$$

$$\mathbf{A} \equiv \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{B} \equiv \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix}, \mathbf{C} \equiv \begin{bmatrix} 1 & 0 & -\frac{z_c}{g} \end{bmatrix} \tag{10}$$

Finally the optimal controller which minimizes the performance index, formulated by equation (8), is given by:

$$\dddot{x}(k) = -G_1 \sum_{i=0}^{k} e(i) - G_2 \mathbf{x}(k) - \sum_{j=1}^{N_L} G_p(j) p^{ref}(k+j) \tag{11}$$

where $e(i) = p_x(i) - p_x^{ref}(i)$, and G_1, G_2 and $G_p(j)$ are the gains calculated from the weights Q and R and the parameters used in the system described by equation (9).

It is interesting to note the unicity of the weights $G_p(j)$ for a given control period T, preview window N_L and constant CoM height z_c. An alternative explanation to the initial solution [11] is given in section 6.3.

This latter extension is important since it enables the robot to take into account the following step. However if there is a sudden change due to modifications of the environment, it is necessary to recompute the trajectory. This new trajectory should handle the current state of the robot, and the next step. Nishiwaki [22] proposed an extension of Kajita's [11] control scheme to handle this case.

2.5 Realizing the CoM trajectory

In section 2.4 we explained that the CoM trajectory provided by the algorithm is based upon a simplified model of the robot. From this trajectory we can adopt several strategies to compute the corresponding joint values. It is possible to use the real model of the robot and other motions or even a stack of tasks. The following describes two strategies which have been used experimentally.

2.5.1 Waist and CoM rigidly bound together

The first strategy, initially suggested by Kajita [11], is to assume that the waist is rigidly fixed to the CoM. In the case of robots with 6 DOF legs, such as HRP-2, it is possible to solve the problem by inverse kinematics. A second preview-loop is computed to compensate for the discrepancy between the ZMP reference trajectory and the ZMP trajectory computed from the previous solution considering the multibody model. This second preview-loop acts as a dynamic filter, and offers a modification of the CoM trajectory.

2.5.2 Constraining the robot to be a 3D LIPM

The second strategy consists in constraining the robot to act like the 3D Linear Inverted Pendulum. This can be done by a stack of tasks using the Jacobian of the CoM or by using the Resolved Momentum Control designed by Kajita [12]. The latter one has been used successfully by Neo [21] to achieve real-time motion in the context of teleoperation.

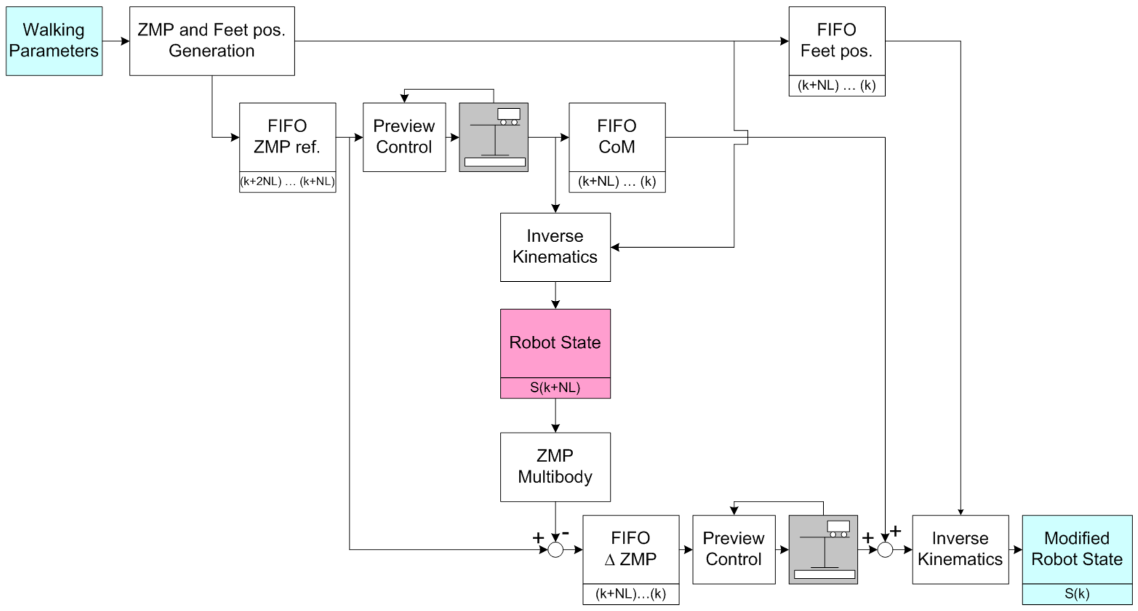

Figure 3: ZMP Preview Control scheme

2.6 Overall structure

Figure 3 illustrates the previous considerations by describing the algorithm presented in [11]. It is also the basis for the three extensions presented in this paper. The inputs are the walking parameters, i.e. step height, double support and single support time, and the foot steps. The feet and ZMP reference trajectories stem from those informations. They are put inside FIFOs, which is crucial for real-time applications as explained in the next section. The ZMP reference trajectory is used to generated the first set of CoM trajectories. These trajectories together with the position of the feet allow, via inverse kinematics to compute a robot attitude, which is itself used to compute the ZMP multibody.

3 Design Patterns and Components

3.1 Definitions

Design Pattern is originally an architectural concept introduced by Alexander [1]. It gained popularity in computer science with the book “Design Patterns: Elements of Reusable Object-Oriented Software” of Gamma *et al.* [9], although earlier works had already tried to apply patterns to programming [3][4]. Thus, a design pattern is commonly understood as

an architectural solution to some frequently encountered situations of software design [18].

The main reasons for this success are the design pattern catalogs which describe frequent situations in software design [9]. Those catalogs have succeeded in establishing a common vocabulary between software developers. Moreover [18], design patterns are a collective repository of “Best Software Design

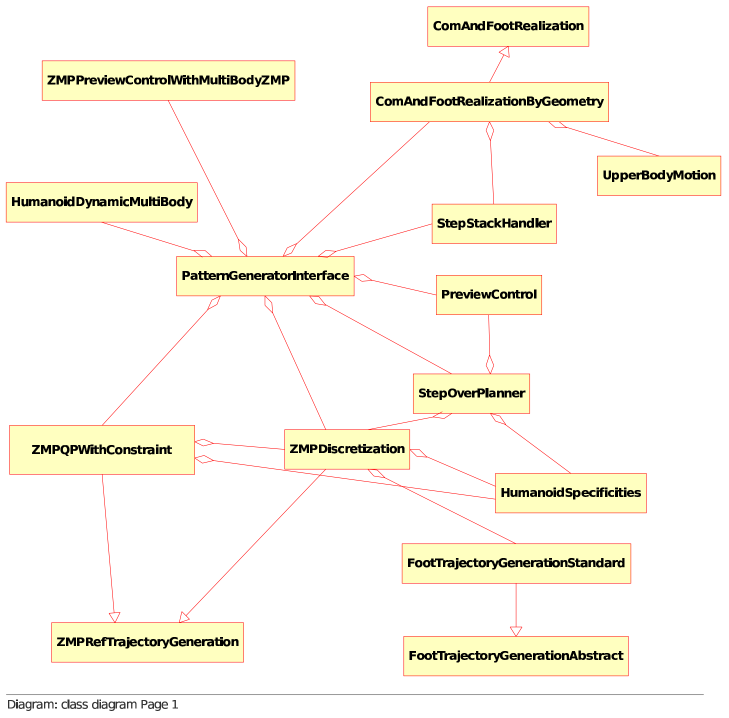
ComAndFootRealization
ZMPPreviewControlWithMultiBodyZMP
ComAndFootRealizationByGeometry
UpperBodyMotion
HumanoidDynamicMultiBody
StepStackHandler
PatternGeneratorInterface
PreviewControl
StepOverPlanner
ZMPQPWithConstraint
ZMPDiscretization
HumanoidSpecificities
FootTrajectoryGenerationStandard
ZMPRefTrajectoryGeneration
FootTrajectoryGenerationAbstract
Diagram: class diagram Page 1

Figure 4: Simplified Class Diagram of the implemented Humanoid Walking Pattern Generator.

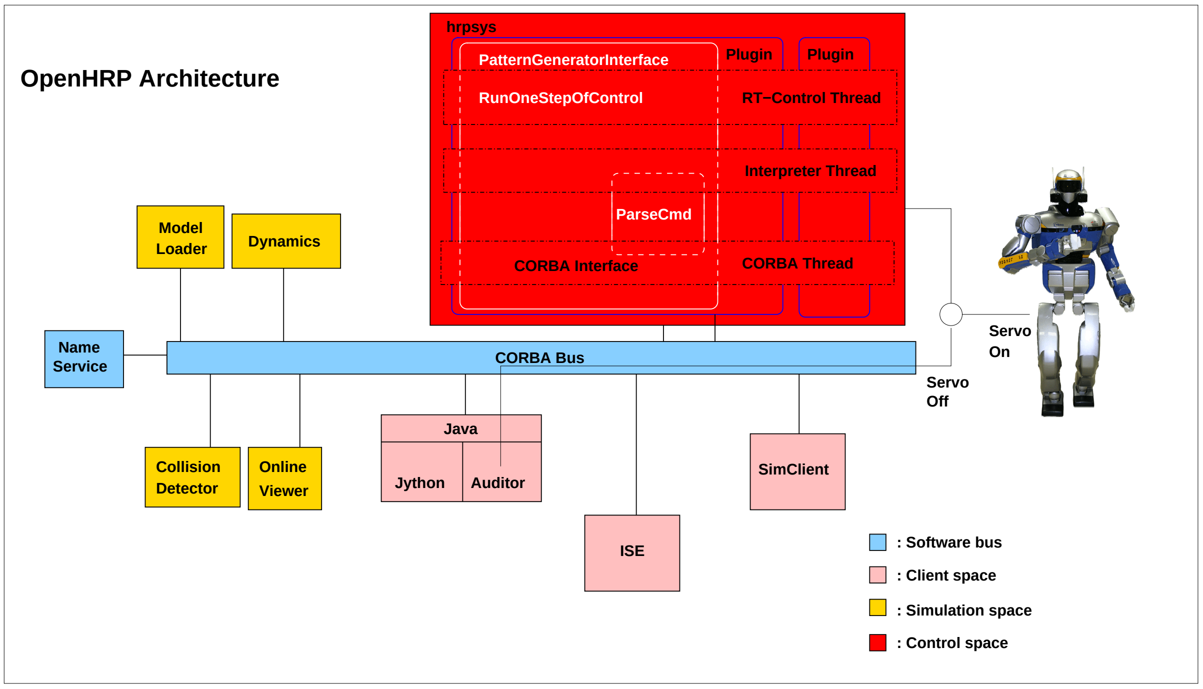

Figure 5: Our pattern generator as a plugin inside OpenHRP.

Practices" because they evolved from the collective wisdom of many designers. Because a design pattern ***describes*** the solution to a problem, it is often considered to go only half-way towards the solution [18], and should be turned into an off-the-shelf component. We believe that both aspects are important. Indeed re-implementation of design pattern and algorithms are often mainly motivated by learning. In these situations design patterns allow speeding up the process by putting the learner in the good tracks. On the other hand componentization allows a user to master the architecture so as to quickly test new ideas which are coherent with the framework. Both are presented in this paper.

3.2 Design Pattern and Components in robotics

There are recent examples of robotic design patterns for instance regarding general architecture models for global automation systems such as [6]. There are also well-known software architecture used for very successful robot missions such as: CLARAty for NASA's rovers [8], OpenHRP for humanoid robots [17], GenoM [7] for rover-like and museum-guide robots. All have in common the use of middleware structure on top of a real-time operating system abstracted by distributed and parallel multi-tasks. Several other's systems exist such as MARIE, YARP, MIRO, all of which are quite generic (see [5] for comparative results). Because such technology is available it is becoming necessary to introduce good software practice to organize the libraries used with such frameworks. This point will be further explained in the next section by describing how we designed a modular Pattern Generator Library using design patterns and the OpenHRP architecture. One of the numerous advantages of this software environment is the possibility to use exactly the same executable for simulation and for real experiments. It provides a clear separation between the real-time and the non real-time parts of a controller, and allows the use of CORBA to solve middleware access.

4 HWPG Modular architecture

The overall organization of the Pattern Generator is depicted in figure 4. Briefly, a client has access to the library through a **PatternGeneratorInterface** object. Its role is to hide all the underlying complexity of the algorithm. As an input, it takes a VRML model of the robot and some specific files that provide semantic elements specific to humanoids. The output is a ZMP position, and the robot's joint values for each control iteration. The following explains the different software aspects which make this implementation interesting and describes the associated design patterns.

4.1 Algorithm Interchangeability

A very important aspect for a research tool is the possibility to interchange some key parts which have the same functionality but use different approaches. This is commonly known as the **Strategy** design pattern in [9] and its intent is to:

"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the

algorithm vary independently from clients that use it." In our specific case, there are four abstract classes which are used for this: **ComAndFootRealization** which represents the strategy to generate a robot pose for a given CoM and Foot position. An actual implementation of this class is **ComAndFootRealizationByGeometry**, which solves the problem by assuming that the CoM is fixed regarding the waist. **ZMPRefTrajectoryGeneration** handles the strategies to build a reference trajectory of ZMP. Class **ZMPDiscretization** implements one of these strategies proposed by Kajita in [11]. This heuristic is very meaningful as it fixed the ZMP reference at the most stable position, i.e. the center of the convex hull formed by the supporting foot. A different strategy is to solve a Quadratic Program with constraints in [29] and is implemented by **ZMPQPWithConstraint**. Foot trajectories strategies derived from the abstract class **FootTrajectoryGenerationAbstract**. The classical 3rd order polynomials are implemented in the class **FootTrajectoryGenerationStandard**. **HumanoidDynamicRobot** represents the strategy to compute the dynamics of the robot. It was also obtained by applying the **Bridge** design pattern, the goal of which is to decouple the abstraction from its implementation so the two can vary independently.

4.2 Code stability and simplicity of use

In order to simplify the use of the library, we used the design pattern known as **Facade** [9]. The main advantage is to offer a simple view to most of the clients, therefore requiring only those who wish to use new algorithms to look beyond the facade. The class implementing this design pattern is **PatternGeneratorInterface** (figure 4). In order to avoid to change the interface of this class, the design pattern **Decorator** has also been used. Further detail is given in the following.

4.3 Incrementing functionalities without changing the interface

The problem is to give the user the possibility to change parameters of algorithms which were not initially inside the system. This is achieved by the **Decorator** design pattern. This pattern consists in making it possible to add functionalities to classes without modifying their interface. The general method **ParseCmd** of class **PatternGeneratorInterface** which can parse commands. Each subclass of **SimplePlugin** can register dynamically commands which can be handled. Therefore the interaction with the interface is extremely simplified for the user. An example is given in listing 1. At line 5 the lift-off and landing angle **omega** is specified by **:omega 0.0**. The other parameters are self-explaining except for **:armparameters 0.5** which is a gain related to the arm motions. It is used when upper-body motion is not specified: the position of the hand is directly proportional to the distance of the opposite foot in the waist reference frame. This minimizes the moment around the vertical axis. A key point is that conversions and number of parameters handling is done internally, as a result of the use of **istringstream**. The interface of the class **PatternGeneratorInterface** thus does not have to be modified. This way the number of steps that can be specified is not apriori limited as shown in line 11.

4.4 Example: a stand-alone program

Listing 1 is a minimal program which is used in our group to debug and test new algorithms. Again the use of **istringstream** is essential to have a constant constructor interface. The arguments (line 20) are a file for the controller, the directory of the robot's VRML model, and the entry file. The last two files first specify the semantic of the robot's joints, i.e. which joint is the head, which are the arms, the feet... and second which joints are actuated. From line 26 to 38 the initial pose of the robot is specified. Line 44 calls the function explained previously. Finally by calling **RunOneStepOfTheControlLoop** (line 45) the user triggers a step of the walking pattern generator. The returned values in **CurrentConfiguration** are a vector of values for the actuated joints plus a free-flyer. In the context of HRP-2 the free-flyer is the waist position in the world coordinates. In **CurrentVelocity** the speed of all the actuated joints plus the linear and angular speed of the free-flyer are returned. Finally the value of the ZMP in the waist reference frame is returned in **ZMPTarget**. **ZMPTarget** and **CurrentConfiguration** are necessary for HRP-2's stabilizer.

Macros, such as **MAL_VECTOR_DIM**, represent a Matrix Abstraction Layer which is used to change the matrix library without any code modification. So far it has been used internally with 3 different libraries: boost's ublas[1], VNL[2] and internally developed specific libraries. As all the algorithms use this layer, it has helped to reduce code's dependancy to specific mathematical libraries.

5 Real-time consideration

Real-time performance is crucial as the information provided by the HWPG is directly used inside the control loop of OpenHRP through a plug-in. It is therefore very important to insure fast computation, and avoid any overhead. The latter would essentially be due to the computation of the ZMP reference trajectory. This computation should thus not be correlated with the generation of the joint values.

5.1 Predefined foot-steps

For realizing predefined foot-steps, the ZMP reference trajectory is computed in the non real-time part. More specifically, it is computed when the user passes **:stepseq** through the method **ParseCmd**. It creates a stack of ZMP reference positions which is then fed to the control loop. Thus, according to the Preview Control Scheme depicted in figure 3, the steps to be computed in the control-loop are:

1. the first preview control,
2. the realization of the related CoM,
3. the ZMP for the multibody model,
4. the second preview control, and

[1]http://www.boost.org
[2]http://www.doc.ic.ac.uk/ ajd/Scene/download.html

Listing 1: A stand-alone program

```
#include <sstream>
#include <PatternGeneratorInterface.h>
using namespace::PatternGeneratorJRL;
void StraightWalking(PatternGeneratorInterface &aPGI){
  const char lBuffer[5][256] =  {":omega 0.0", ":stepheight 0.07",
        ":singlesupporttime 0.78",":doublesupporttime 0.02",
        ":armparameters 0.5"};
  for(int i=0;i<5;i++) {
      std::istringstream strm(lBuffer[i]);
      aPGI.ParseCmd(strm); }
  istringstream strm2(":stepseq 0.0 -0.095 0.0 \
                       0.2 0.19 0.0 0.2 -0.19 0.0 0.2 0.19 0.0  \
                       0.2 -0.19 0.0 0.2 0.19 0.0 0.0 -0.19 0.0");
  aPGI.ParseCmd(strm2);
}
int main(int argc, char *argv[]){
  string Global(argv[1]), VRMLPath(argv[2]),
    VRMLFileName(argv[3]), SpecificitiesFileName(argv[4]),
    LinkJointRank (argv[5]);
  Global += " " + VRMLPath + " " + VRMLFileName + " " +
         SpecificitiesFileName + " " + LinkJointRank;
  std::istringstream strm(Global);
  PatternGeneratorInterface * aPGI;
  aPGI = new PatternGeneratorInterface(strm);
  // Normal position - New half sitting
  double dInitPos[40] = {
    0.0, 0.0, -26.0, 50.0, -24.0, 0.0, 0.0, 0.0, -26.0, 50.0, -24.0, 0.0,  // legs
    0.0, 0.0, 0.0, 0.0, // chest and head
    15.0, -10.0, 0.0, -30.0, 0.0, 0.0, 10.0, // right arm
    15.0,  10.0, 0.0, -30.0, 0.0, 0.0, 10.0, // left arm
    -10.0, 10.0, -10.0, 10.0, -10.0  // right hand
    -10.0, 10.0, -10.0, 10.0, -10.0  // left hand};
  // This is a vector corresponding to the DOFs actuated of the robot.
  MAL_VECTOR_DIM(InitialPosition,double,40);
  MAL_VECTOR_DIM(CurrentPosition,double,40);
  for(unsigned int i=0;i<MAL_VECTOR_SIZE(InitialPosition);i++)
    InitialPosition(i) = dInitPos[i]*M_PI/180.0;
  aPGI->SetCurrentJointValues(InitialPosition);
  // This is a vector corresponding to ALL the DOFS of the robot:
  // free flyer + actuated DOFS.
  MAL_VECTOR_DIM(CurrentConfiguration,double,46);
  MAL_VECTOR_DIM(CurrentVelocity,double,46);
  MAL_VECTOR_DIM(ZMPTarget,double,3);
  StraightWalking(*aPGI);
  while(aPGI->RunOneStepOfTheControlLoop(CurrentConfiguration,
                                         CurrentVelocity,
                                            ZMPTarget)){}
}
```

5. finally the realization of the second CoM.

As the realization of the CoM is computed twice in this context, it has to be extremely fast. On a Pentium D Xeon running at 3.2GHz, those steps take 0.4ms. They take about 0.8ms on HRP-2's Pentium M 1.8GHz. There is therefore no performance issue when footsteps are generated off-line.

5.2 On-line foot steps

The case of the on-line foot step generation is a bit more complex to handle. Indeed the nature of the problem is both continuous (the trajectory of the feet, CoM and ZMP) and discrete (the next step as a whole). First it is necessary to make sure that there will always be an available ZMP reference trajectory in the control loop. For this, the object **ZMPPreviewControlLoopWithMultiBodyZMP** checks the number of elements in the stack. If no element is available a new step is pulled up from **StepStackHandler**. The latter inspects its stack to check whether a step is available. If planning or reaction to sensor feedback do not provide any, a default strategy is needed. Following the strategy of the commercial pattern generator provided with HRP-2, the **StepStackHandler** class generates one step to have the robot stay at its current location.

There are then two ways to create the set of ZMP reference trajectories: Generating all the ZMP positions for the complete step motion, or generating them on-line. As each algorithm takes the future into account, if the pattern-generator allows modification of the step in its future window, it has to recompute the overall trajectory [19][23] whenever such a modification occurs. On the contrary, if we assume that no modification is allowed in the future window but only to add a new foot step, thus the computation for one position in the ZMP reference trajectory can be realized at each control loop iteration as in the algorithm proposed by Kajita [11]. We implemented the solution where a full ZMP reference trajectory is computed for one foot. For a preview window of 1.6s it takes 0.6ms on a Pentium D Xeon running at 3.2GHz. This mode has only been tested in simulation, but it should fit in the 5ms window of HRP-2's control loop. In comparison, Neo's Resolved Momentum Control method [21] takes 1.2ms on the real robot, where the ZMP reference trajectory is computed on-line.

5.3 Integration within OpenHRP

OpenHRP is a software framework designed by Kanehiro [16] and used on HRP-2. It provides a set of components to perform dynamical simulation, collision detection, display the state of the robot, and control over HRP-2. The overall architecture is depicted in figure 5.

The low-level control part is organized as a sequence of plugins which can take the current state of the robot, the command to the robot generated by the previous plugins and create a new one. A plugin is divided into three real-time parts which are the starting phase, the control phase, and the cleanup phase. A plugin can also receive requests to change its internal state in two ways: an interpreter with minimal capabilities, and acting as a CORBA server.

The user controls the robot through a Java application called the auditor. It is then possible to script the behavior of the components accessible through CORBA using **Jython**. It is also possible to send orders through the **hrpsys**'s interpreter, which is also a CORBA server. As Jython is a Java implementation of python, any direct access to the plugins implies the creation of a Java CORBA client for the user. Although this architecture provides a very flexible software environment, the impact of any interface changes is quite important. This is a problem we managed to avoid with the **Facade** design pattern.

Moreover this kind of architecture is complex and needs some time to be mastered by visitors, students and non-specialists in related computer software problems. We therefore set as a goal that they are able to work on specific software parts with a simplified environment, thus re-mapping the subsequent modified software to the real architecture with minimal impact. We do think we achieved this thanks to OpenHRP and the modular architecture proposed in this paper. To illustrate this assertion, three research developments published in referred conferences and based on this software are now presented.

6 Case studies

In this section, three case studies regarding the use of this library for research purposes are presented. The main target of the applied software strategy is easiness and flexibility of use for researchers. Therefore two criteria were taken into account: the ease of use for non-computer specialists, and the time for implementing novel algorithms both in realistic simulation and on a real robot. The two principal researchers in the first two case-studies are originally from the mechanical field, and had little experience on software architecture such as OpenHRP. However in little less than one year, they were able to produce new concepts regarding Walking Pattern Generation. Those concepts were validated on realistic simulation for the first study-case, and in addition on the real HRP-2 for the second one. The third study-case is an extreme situation where the implementation of an algorithm and carrying out of a real experience have been done within four days.

6.1 Simulation: toe-joint

6.1.1 Goal

In this work [26], we have simulated a modified HRP-2 humanoid robot equipped with the passive toe-joints depicted in figure 7 (left). By considering an underactuated phase during walking and an appropriate ZMP trajectory it was possible to increase by 1.5 times the step-length of HRP-2, and have smoother CoM trajectories.

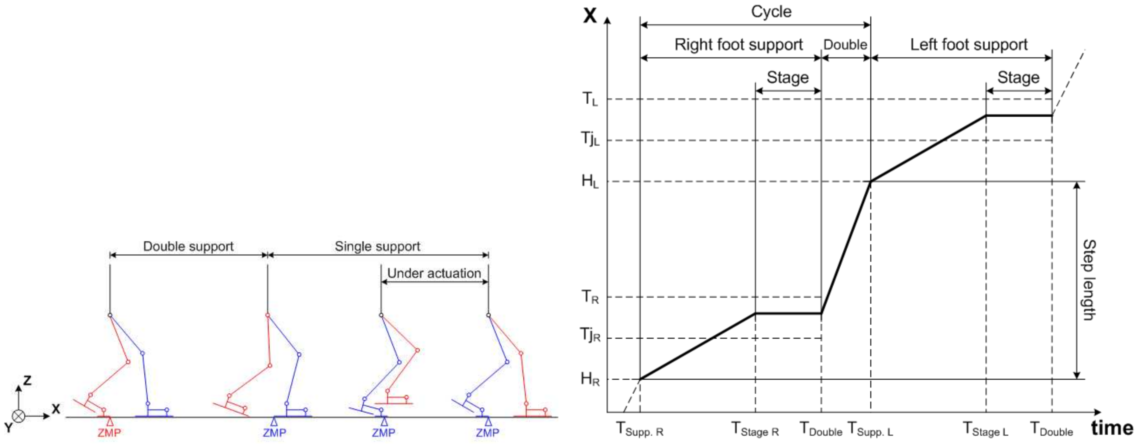

Figure 6: (left) Walking Gait With Toed Feet (right) ZMP trajectory for toed feet gait.

6.1.2 ZMP trajectory during single support phase with a toe joint

During the single support phase, the ZMP moves linearly from the heel (H) to the toes (T) (figure 6 (right)). At the end of this phase (from T_{Stage} to T_{Double}), the ZMP is kept in a constant position between the toe joint (T_j) and the tip of the toe (T), which makes the whole body rotate around this joint. During the double support phase, starting with the swing foot touch down (T_{Double}), the ZMP is transferred from the toes of the support foot to the heel of the swing one.

6.1.3 Underactuated phase

During the flat feet walking gait, the feet are kept parallel to the ground. However, with toed feet, a rotation around T_j occurs during the single support phase. In order to know that T_j's angle is, a model of the spring related to the ground reaction force is used. The toe joint is orthogonal to the sagittal plan (X, Z). Its angular position depends on the dynamical effects in this plan. During single support phase, the forces acting on the CoM are gravitational and forward accelerations. A simple model was used in order to compute the toe joint angle as depicted in figure 7 (right). At the end of this phase, the ZMP is located under the toe, which can be considered as stable on the ground (Slipping phenomena are not considered at this stage). This leads to the following equations:

$$\mathbf{f} = -M(\mathbf{g} - \ddot{\mathbf{x}}) \tag{12}$$

$$\tau_{toe} = -\mathbf{p}_{toe} \times \mathbf{f} \tag{13}$$

$$\tau_{toe} = -(K_p\alpha + K_d\dot{\alpha})\mathbf{Y} \tag{14}$$

with $\mathbf{f}$ the ground reaction force, m the mass of the robot, $\ddot{\mathbf{x}}$ the CoM acceleration along the X axis, τ_{toe} the toe joint's torque, $\mathbf{p}_{toe}$ the position vector of ZMP with respect to the toe joint, α the toe joint angle, $\mathbf{Y}$ the position of the toe in the foot reference frame, and finally K_p and K_d toe joint spring and damper constants. Given the CoM acceleration along the X axis, the toe joint's torque is computed

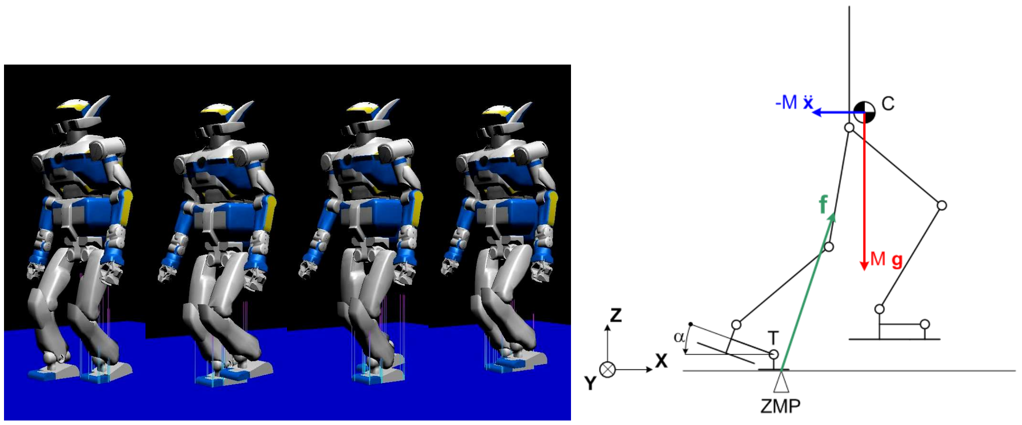

Figure 7: (left) Dynamical simulation of the HRP-2 model with Toe Joint (K_p = 10Nm/rad K_d = 5Nms/rad) (right) Force and toe joint angle.

using equations (12) and (13). Then, the angular position is derived from the spring dumper model at each simulation step from equation (14).

6.1.4 Implementation

Based upon the above description, an implementation has been done by modifying **ZMPDiscretization**, the object which generates ZMP reference trajectory. We simply added a linear slope during the first phase, until the middle of the toe. The second modification was to change the realization of the CoM trajectory and the feet position. To shorten computation time the evaluation of the multibody ZMP was done on the flat-feeted HRP-2 model. The dynamic simulation using the dynamic model based on equation 12 to compute the toe angle only comes in during the last phase. For the evaluation of α, the new position of the feet is derived and fed to the inverse kinematic computation of the usual HRP-2 model.

The merit of this work was to show that it is possible to increase the step-length, and to decrease the variation of the knee joints using passive joints. Interestingly Kajita [14] uses a similar technique to make his prototype hop. The main difference in his approach is to additionally consider the energy restitution during the spring release phase. The simulation reported by Kajita shows the robot able to run at 3km/h.

6.2 Application: Stepping over obstacles

6.2.1 Approach

Verrelst in [28] proposed a method to dynamically step over obstacles as large as 15cm. The input is an evaluation of the obstacle position and size provided by vision for instance. First the algorithm acts as a dynamical planner. It uses the preview control to have a first CoM trajectory, and checks whether there is a feasible trajectory to avoid the obstacle. A trajectory is said to be feasible if it fulfills the following criteria:

- the knee angle is beyond a minimum angle to avoid over-stretching,
- there is no collision between the legs and the obstacle and,
- there is a kinematic solution for a chosen height of the waist.

If no such trajectory exists for a given height, then the CoM's height is lowered and the preview control is rerun. For a given height once a configuration similar to that of figure 8 (left) is possible, a foot trajectory is designed.

6.2.2 Foot Trajectory

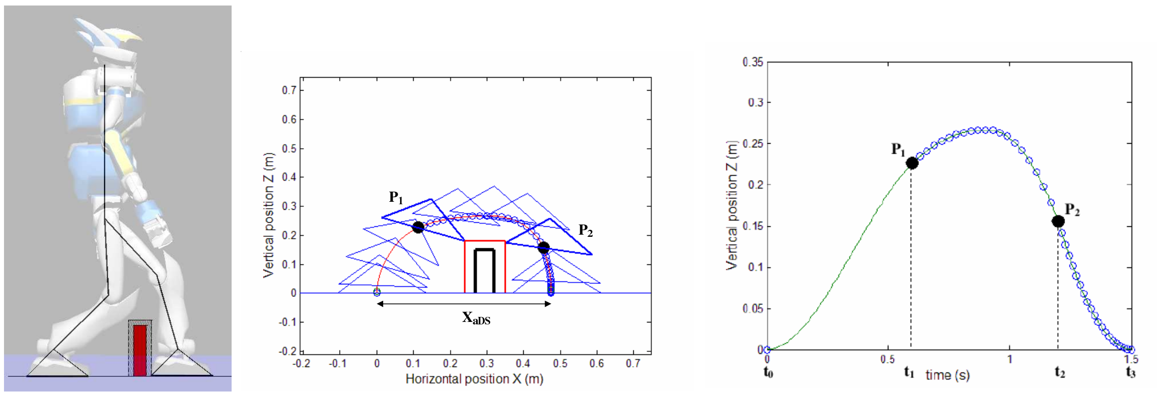

Figure 8: (left) Double support phase feasibility (middle) Spline Trajectory in the XZ plane (right) Spline Trajectory to time for vertical direction Z

The foot trajectory for this application is different from more classical polynomial trajectories because the latter ones tend to oscillate in the presence of control points. Instead, Clamped Cubic Splines are used to specify intermediate velocities. This is needed to specify control points $\mathbf{P}_1$ and $\mathbf{P}_2$, depicted in figure 8(right), to avoid the obstacle and to minimize impacts upon touch-down. Indeed, for such moments Clamped Cubic spline functions ensure zero velocity, they do not specify what the acceleration should be. In real experimentation, the desired trajectory cannot be perfectly tracked due to the compliance in the foot. Clamped Cubic splines allow smoothing of the touch-down condition.

For normal walking the step length is 0.23m, while stepping over requires a step length of 0.48m. The size of the step being almost doubled, its duration is increased. The step sequence lasts 0.78s in single support and 0.02s in double support for the normal steps. Stepping over lasts 1.5s and the preceding and the succeeding steps 0.04s.

6.2.3 Upper body motion

To clear more space during the double support over the obstacle and consequently allow for larger obstacles to be stepped over, the waist of the robot is rotated. The HRP-2 robot includes two extra

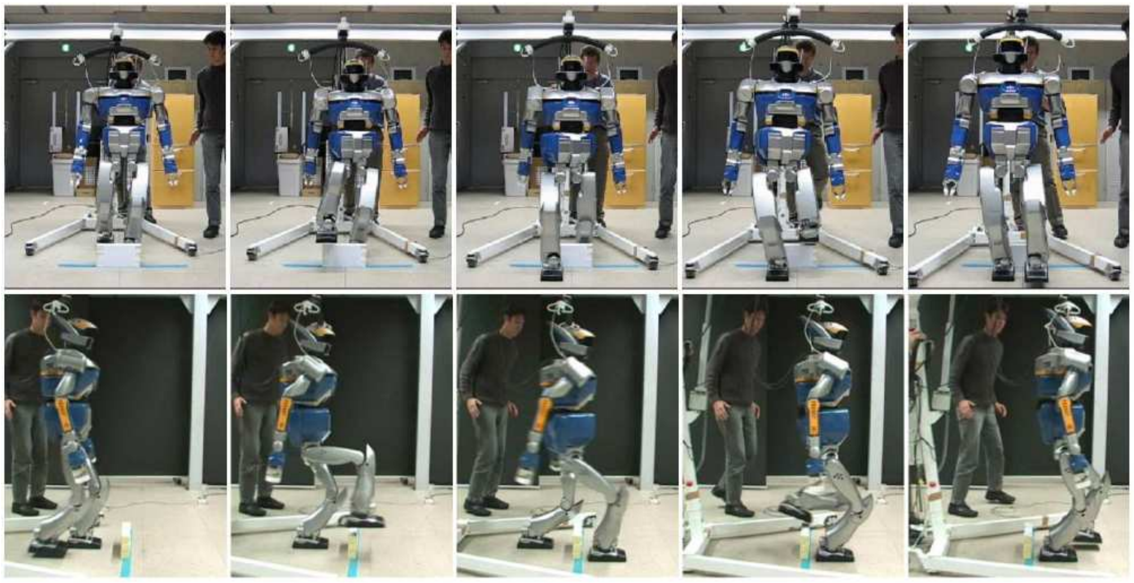

Figure 9: Stepping over experiments with HRP-2.

degrees of freedom (yaw and pitch) between the waist and upper-body so the upper-body and head remain oriented towards the walking direction.

6.2.4 Implementation

The dynamic planner is implemented by the class **StepOverPlanner** which uses objects from the pattern generator to plan the trajectory. Collision detection is done by a custom made object which is unfortunately specific to HRP-2, due to performance concerns. Extending the algorithm can be done by using a collision checker such as Vclip[3]. New information was added to the foot data structure used by **StepStackHandler**, namely individual double support time and single support time. In addition, inside **ComAndFootRealizationByGeometry**, we have added a **UpperBodyMotion** class which implements the specific strategy for the upper body while stepping over. The coordination is done through the type of footsteps. The ability to change the waist was also included in the part generating the CoM's trajectory. In order to parameterize the upper body's motion, and to indicate the obstacle's position, new features were added to the method **ParseCmd** through the plugin mechanisms.

Again through the **Decorator** design pattern, we are able to control the different strategies without any impact on the interface as depicted in listing 2. The first line of 2 specifies that the stepping over strategy is used. Time distribution of the different control points is defined by the time distribution parameters on line 2. The obstacle parameters to specify are the position of the obstacle, its orientation in the plane, and its dimensions. A sequence of steps is given, but it is overwritten by the stepping over strategy. The result of this experiment is depicted in figure 9.

[3] http://www.merl.com/projects/vclip

Listing 2: The python instruction to specify an obstacle

```
walk.sendMsg(":walkmode 2")
walk.sendMsg(":TimeDistributeParameters 2.0 3.7 1.5 3.0")
walk.sendMsg(":obstacleparameters 1.0 0.0 0.0 0.0 0.12 1.0 0.05")
walk.sendMsg(":stepseq 0.0 -0.095 0.0 0.2 0.19 0.0 0.2 -0.19 0.0 0.0 0.19 0.0 ")
```

6.3 Application: Quadratic programs and ZMP

6.3.1 Quadratic reformulation of the preview control

Wieber in [29] suggested to reformulate the optimization problem formulated by Kajita in [11] using the N_L versions of equation (9):

$$\begin{bmatrix} p_x(k+1) \\ \vdots \\ p_x(k+N_L) \end{bmatrix} = \begin{bmatrix} 1 & T & T^2/2 - z_c/g \\ \vdots & \vdots & \vdots \\ 1 & N_L T & N_L^2 T^2 - z_c/g \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} T^3/6 - Tz_c/g & 0 & 0 \\ \vdots & \ddots & \vdots \\ (1+3N_L+3N_L^2)T^3/6 - Tz_c/g & \dots & T^3/6 - Tz_c/g \end{bmatrix} \begin{bmatrix} \dddot{x}(k) \\ \vdots \\ \dddot{x}(k+N_L-1) \end{bmatrix} \tag{15}$$

The second big matrix is a $N \times N$ lower triangular Toeplitz matrix (i.e. with constant diagonals). This relation can be considered in a more compact presentation:

$$\mathbf{Z}_{k+1} = \mathbf{P}_x \mathbf{x}(k) + \mathbf{P}_u \dddot{\mathbf{X}}(k) \tag{16}$$

This allows to rewrite the control scheme linked with the performance index given by equation (8) as:

$$\min_{\dddot{\mathbf{X}}(k)} \frac{1}{2} Q(\mathbf{Z}(k+1) - \mathbf{Z}^{ref}(k+1))^2 + \frac{1}{2} R \dddot{\mathbf{X}}^2(k) \tag{17}$$

This can be solved analytically by:

$$\dddot{X}_k = -(\mathbf{P}_u^T \mathbf{P}_u + \frac{R}{Q} I_{N\times N})^{-1} \mathbf{P}_u^T (\mathbf{P}_x \mathbf{x}(k) - \mathbf{Z}_k^{ref}) \tag{18}$$

where $I_{N\times N}$ is an identity matrix. The reader is kindly invited to read [29] for further details regarding stability and applications in stabilizing a humanoid against strong perturbations.

6.3.2 Explicitly taking into account the limits of the position of the ZMP

One important limitation of the initial algorithm of [11] is the reference trajectory of the ZMP given by a heuristic. One practical reason, when using HRP-2, to have this heuristic is due to the stabilizer commercially available. Indeed its role is to make sure that the ZMP is at the center of the convex hull. However, in general, there are various ZMP trajectories for the same footsteps. Wieber proposed in [29] to further simplify the performance index by solving the following quadratic program:

$$\begin{gathered} \min_{\dddot{\mathbf{X}}(k)} \frac{1}{2} \dddot{\mathbf{X}}^2(k) \\ \mathbf{Z}^{\min}(k) \leq \mathbf{Z}(k) \leq \mathbf{Z}^{\max}(k) \end{gathered} \tag{19}$$

Listing 3: The python instruction to parameterize the new problem

```
walk.sendMsg(":SetAlgoForZmpTrajectory PBW")
walk.sendMsg(":setpbwconstraint XY 0.07 0.05")
walk.sendMsg(":setpbwconstraint T 0.01 ")
walk.sendMsg(":setpbwconstraint N 150")
```

Practically the generated trajectory tends to be very close to the boundary of the constraint. For this reason, we considered the following criteria:

$$\min_{\dddot{\mathbf{X}}(k)} \frac{1}{2}(\dddot{\mathbf{X}}^2(k) + \alpha\dot{\mathbf{X}}^2(k) + \beta(\mathbf{X}(k) - \mathbf{Z}_{ref}(k))^2) \tag{20}$$

The parameters α and β are respectively the weight of the speed and of the heuristic's reference trajectory.

6.3.3 Implementation and experiments

The implementation which corresponds to the Quadratic Program formulated by equation (19) has been carried out during a four days stay of Pierre-Brice Wieber at JRL-Japan with the first author. The impact on the architecture is minimal as it is purely related to the ZMP and COM reference trajectory generation. Therefore it was simply a matter of deriving a class from **ZMPRefTrajectoryGeneration**, called **ZMPQPWithConstraint**. The heritance of the plugin mechanism enables specifying the parameters directly to the object. It allows going directly through three software abstraction layers without any change. The python code depicted in listing 3 illustrates how we could easily set those parameters from within the control window of the robot. In this example N is the size of the preview window, and T is the period for each linear constraint, XY is the total margin we set for the foot in the X and Y direction. In this example 0.07 means that a safety margin of 3.5 cm in front and at the back of the foot is taken into account, where as 0.05 means that a 2.5 safety margin is chosen at the left and the right of the foot.

The current main limitation of this approach is the time spent to solve the quadratic problem given by equation (19). For this we used the **ql** solver function implementing a variant of Powell algorithm written by Schittkowski [25]. On our HRP-2 with a 1.8 GHz Pentium M, it takes almost 5 minutes to compute the CoM and ZMP trajectories for 6 steps.

7 Conclusion

Based on the current state-of-the-art walking pattern generator algorithms we devised a modular architecture using Design Patterns. Through our implementation within OpenHRP, we demonstrated that they make a real difference in state-of-the-art software frameworks for complex robots. Indeed they allow the creation of framework-independent libraries and make this modular architecture possible. Moreover we have illustrated the benefits of this library by presenting three scientific studies relying on it; each of them have been presented in refereed conferences. The last two case-studies have been tested on two HRP-2s: number 10 available at JRL-Japan located in Tsukuba (Japan), and number 14 which is at JRL-France located in Toulouse (France). The library was also used in a different context to improve Self Localization and Map building [27].

8 Acknowledgments

The authors would like to thank the Japan Society for the Promotion of Science (JSPS) for funding, Baptiste Lagarde for proof reading this paper and Ramzi Sellaouti for his work on the passive toe-joint.

REFERENCES

[1] C. Alexander. *A Pattern Language: Towns, Buildings, Construction.*, New York: Oxford University Press., 1977.

[2] N. Ando, T. Suehiro, K. Kitagaki, and T. Kotoku. Rt(robot technology)-component and its standardization - towards component based networked robot systems development. In *International Joint Conference SICE-ICASE*, pp.2633-2638, (2006).

[3] K. Beck and W. Cunningham. Using pattern languages for object-oriented programs. In *Technical Report No. CR-87-43,*, (1987).

[4] K. Beck and W. Cunningham. A laboratory for teaching object-oriented thinking. In *Proc. of the OOPSLA-89: Conference on Object-Oriented Programming: Systems*, pp. 1-6, Languages and Applications, New Orleans, LA, (1989).

[5] G. Broten, S. Monckton, J. Geisbrecht, and J. Collier. Software systems for robotics an applied research perspective. *Advanced Robotic Systems*, 3(1):11-16, (2006).

[6] D. Brugali and G. Menga. Architectural models for global automation systems. *IEEE Transactions on Robotics and Automation*, 18(4):487-493, (2002).

[7] S. Fleury, M. Herrb, and R. Chatila. G^{en}om: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robotics and Systems, IROS*, volume 2, pp. 842-848, (1997).

[8] L. Flückiger and H. Utz. Lessons from applying modern software methods and technologies to robotics. In *International Workshop on Software Development and Integration in Robotics, ICRA 2007*, (2007).

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computer Series, (1994).

[10] K. Kaneko K. Harada, S. Kajita and H. Hirukawa. An analytical method for real-time gait planning for humanoid robots. *International Journal of Humanoid Robotics*, 3(1):1-19, (2006).

[11] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *International Conference on Robotics And Automation, ICRA*, pp. 1620-1626, (2003).

[12] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Resolved momentum control: Humanoid motion planning based on the linear and angular momentum. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 1644-1650. IEEE, 2003.

[13] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa. The 3d linear inverted pendulum mode : A simple modeling of a biped walking pattern generation. In *International Conference on Intelligent Robots and Systems, IROS*, pp. 239-246, (2001).

[14] S. Kajita, K. Kaneko, M. Morisawa, S. Nakaoka, and H. Hirukawa. Zmp-based biped running enhanced by toe springs. In *IEEE International Conference on Robotics and Automation, ICRA*, pp. 3963-3969, (2007).

[15] S. Kajita, T. Yamaura, and A. Kobayashi. Dynamic walking control of a biped robot along a potential energy conserving orbit. *IEEE Transactions on Robotics and Automation*, 8(4):431-438, (1992).

[16] F. Kanehiro, K. Fujiwara, S. Kajita, K. Yokoi, K. Kaneko, H. Hirukawa, Y. Nakamura, and K. Yamane. Open architecture humanoid robotics platform. In *International Conference on Robotics and Automation*, pp. 24-30, (2002).

[17] F. Kanehiro and H. Hirukawa. Openhrp: A software platform for humanoid robotics. In *Humanoids 2006 Workshop on Humanoid Technologies*, (2006).

[18] B. Meyer and K. Arnout. Componentization: The visitor example. *IEEE Computer*, 39(7):23-30, (2006).

[19] M.Morisawa, K.Harada, S.Kajita, K.Kaneko, F.Kanehiro, and H.Hirukawa K.Fujiwara, S.Nakaoka. A biped pattern generation allowing immediate modification of foot placement in real-time. In *IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 581-586, (2006).

[20] K. Nagasaka, Y. Kuroki, S. Suzuki, Y. Itoh, and J. Yamaguchi. Integrated motion control for walking, jumping and running on a small bipedal entertainment robot. In *IEEE International Conference on Robotics and Automation, ICRA*, volume 4, pp. 3189-3194, (2004).

[21] E. S. Neo, K. Yokoi, S. Kajita, F. Kanehiro, and K. Tanie. A switching command-based whole-body operation method for humanoid robots. *IEEE/ASME Transactions on Mechatronics*, 10(5):546-559, (2005).

[22] K. Nishiwaki and S. Kagami. Short cycle pattern generation for online walking control system of humanoids. In *International Symposium on Experimental Robotics (ISER)*, pp. 156, (2006).

[23] K. Nishiwaki, W. Yoon, and S. Kagami. Motion control system that realizes physical interaction between robot's hands and environment during walk. In *IEEE-RAS International Conference on Humanoid Robots*, pp. 542-547, (2006).

[24] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, and N. Higaki K. Fujimura. The intelligent asimo: system overview and integration. In *IEEE/RSJ International Conference on Intelligent Robots and System, 2002.*, volume 3, pp. 2478-2483, (2002).

[25] K. Schittkowski. *QL: A Fortran Code for Convex Quadratic Programming - User's Guide, Version 2.11.* Report, Department of Computer Science, University of Bayreuth, (2005).

[26] R. Sellaouti, O. Stasse, S. Kajita, K. Yokoi, and A. Kheddar. Faster and smoother walking of humanoid hrp-2 with passive toe joints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 4909-4914, (2006).

[27] O. Stasse, A. Davison, R. Sellaouti, and K. Yokoi. Real-time 3D slam for humanoid robot considering pattern generator information. In *International Conference on Intelligent Robots and Systems, IROS*, pp. 348-355, (2006).

[28] B. Verrelst, O. Stasse, K. Yokoi, and B. Vanderborght. Dynamically stepping over obstacles by the humanoid robot hrp-2. In *IEEE RAS/RSJ Conference on Humanoids Robots*, pp. 117-123, (2006).

[29] P. B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *International Conference on Humanoid Robots*, pp. 137-142, (2006).

Olivier Stasse received the MSc degree in operations research (1996) and the PhD degree in intelligent systems (2000), both from University of Paris 6. He is assistant professor of Paris 13 His research interests include humanoids robots as well as distributed and real-time computing applied to vision problems for complex robotic systems. From 2000 and 2003, he was at the Laboratoire de Transport et Traitement de l'Information (L2TI), and then joined the SONY robot-soccer team of the Laboratoire de Robotique de Versailles (LRV). Since 2003, he has been a member of the Joint French-Japanese Robotics Laboratory (JRL) in a secondment position as a CNRS researcher. He is a member of the IEEE Robotics and Automation Society.

Björn Verrelst was born in Antwerp, Belgium, in 1972. He received the degree in study of Mechanical Engineering at the Vrije Universiteit Brussel in 1996 and a PhD in Applied Sciences in 2005. Currently he is Post-doc researcher at the Vrije Universiteit Brussel. The focus of his research is the use of pneumatic artificial muscles in the walking biped Lucy for dynamically balanced walking and compliant actuation for robotic applications in general. During the period 2005/2006 he conducted a one year JSPS post-doc research at the National Institute for Advanced Industrial Science and Technology in Tsukuba, Japan

Pierre-Brice Wieber graduated from the Ecole Polytechnique, Paris, France, in 1996 and received his Ph.D. degree in Robotics from the Ecole des Mines de Paris, France, in 2000. Since 2001, he has been with the INRIA Rhône-Alpes in the BIPOP team. His research interests include the modeling and control of artificial walking.

Bram Vanderborght was born in Wilrijk, Belgium, in 1980. He received the degree in study of Mechanical Engineering at the Vrije Universiteit Brussel in 2003. Since 2003 he was researcher at the Vrije Universiteit Brussel, supported by the Fund for Scientific Research Flanders (FWO). In May 2007 he received his PhD in Applied Sciences. The focus of his research is the use of adaptable compliance of pneumatic artificial muscles in the dynamically balanced biped Lucy. In May 2006 he performed research on the humanoids robot HRP-2 at the Joint Japanese/French Robotics Laboratory (JRL) in AIST, Tsukuba (Japan) in the ongoing research "Gait Planning for Humanoids Robots: Negotiating Obstacles". From October 2007 he will work as post-doc researcher at the Italian Institute of Technology in Genua, Italy.

Paul EVRARD is a PhD student at University of Evry. He received a Master degree in computer science from the Ecole Nationale Superieure d'Informatique pour l'Industrie et l'Entreprise and joined the Joint Japanese-French Robotics Laboratory (JRL) as a research assistant (2006). His interests include software engineering, robot control and haptic.

Abderrahmane KHEDDAR is currently a professor at the university of Evry, France. He received a DEA (Master of Science by research) and the Ph.D. degree in robotics and computer science, both from the University Paris 6, France. He was several months a visiting researcher at the formal Mechanical Engineering Laboratory (Bio-Robotics Division) in Japan, working with Professor Kazuo Tanie. Since 2003, in the frame of his CNRS secondment he took a Director of Research position and is the Codirector of the AIST/CNRS Joint Japanese-French Robotics Laboratory (JRL) in Tsukuba, Japan. His research interests include chronologically teleoperation and telerobotics, haptic interaction with virtual and machined avatars, humanoids, and electro-active polymers. He is a founding member of the IEEE-RAS chapter on Haptics and Associate Editor of IEEE Transactions on Haptics since Jan. 2008.

Kazuhito Yokoi received his B. E. degree in Mechanical Engineering from Nagoya Institute of Technology in 1984, and the M. E. and Ph. D. degrees Mechanical Engineering Science from the Tokyo Institute of Technology 1986 and 1994, respectively. In 1986, he joined the Mechanical Engineering Laboratory, Ministry of International Trade and Industry. He is currently scientific group leader of Autonomous Behavior Control Research Group and co-director of IS/AIST-CNRS Joint Robotics Laboratory, Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), at Tsukuba, Japan. He is also an adjunctive professor of Cooperative Graduate School at University of Tsukuba. From November 1994 to October 1995, he was a Visiting Scholar at Robotics Laboratory, Computer Science Department, Stanford University. His research interests include humanoids, human-centered robotics, and intelligent robot systems. Dr. Yokoi is a member of the IEEE Robotics and Automation Society.