CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

National Institute of
Advanced Industrial Science and Technology (AIST)
独立行政法人　産業技術総合研究所

Documentation on the JRL's Pattern generator

Olivier Stasse, Ramzi Sellouati

Joint Japanese French Robotics Laboratory (JRL)
CNRS, AIST

Contents

1 User manual **5**
1.1 Introduction 5
1.2 Installation and quick start for the pattern generator 5
1.2.1 Download 5
1.2.2 Compiling and installation 5
1.2.3 Example 6
1.2.4 Needed libraries 6
1.3 Plugin : WalkGenJRL 6
1.3.1 Introduction 6
1.3.2 Functionnalities 7
Foot positionning 7
Walking parameters 11

2 The theory behind **13**
2.1 Inverse Kinematics 13
2.1.1 The legs 13
2.2 Dynamic models of biped robot 15
2.2.1 3D Linear Inverted Pendulum Mode and Zero-moment point . 15
2.2.2 ZMP equations and cart-table model 16
2.3 Walking pattern generation for a given ZMP 17
2.3.1 Pattern generation as an inverse problem 17
2.3.2 ZMP control as a servo problem 17
2.3.3 Pattern generation by preview control 20
2.3.4 Pattern generation for multibody model 22
2.3.5 References 26
2.3.6 The angular momentum problem 26
2.3.7 Arm motion heuristic 26
2.4 Finding the weights for the preview control 27
2.4.1 The general scheme [1] 27
2.4.2 Removing the offset of the ZMP 27
2.4.3 Implementation of the weights computation 28
2.5 Momentum Equation 28
2.5.1 Momentum and joint velocities 28
2.5.2 Constraints of foot contact 28
2.6 Resolved Momentum Control 29
2.6.1 Setting momentum reference 29

2.6.2 Momentum selection and control by pseudo-inverse 30
2.6.3 Calculation of the interia matrices 31
2.6.4 Walking using the Resolved Momentum Control 32
2.7 To change the library 32
2.7.1 Introduction 32
2.7.2 ZMPDiscretization 32
2.7.3 Preview Control 33
2.7.4 Dynamic Multi Body 33
2.7.5 ZMPPreviewControlWithZMPMultiBody 34

3 The Upper body Motion 37
3.1 Walking Mode inside the Pattern Generator V.2 37

Chapter 1

User manual

1.1 Introduction

This chapter presents the pattern generator developped on the work proposed by Kajita-San [2] for the HRP-2 humanoid robot. Some part of this chapter are taken directly from the article. We added some details and underlaying assumptions related to our own implementation, and detailled the usage.

WARNING: Some important safety features such as collision detection and others are **NOT** implemented. Using right away this plugin without any simulation can be ***extremly*** dangerous. You use this software at your own risk.

1.2 Installation and quick start for the pattern generator

The following described how to use the pattern generator assuming that you have OpenHRP-2 with the full HRP-2 model.

1.2.1 Download

Set you current directory into OpenHRP such as:

```
cd ~/src/OpenHRP
```

To get the latest version:

```
svn co svn+ssh://jrlserver/home/svn/PatternGeneratorJRL
```

You should see now a new directory with the name PatternGeneratorJRL.

1.2.2 Compiling and installation

To start the creation of the library type :

```
cd PatternGeneratorJRL/src
```

Check the makefile to make sure that the default g++ compiler is taking a GCC compiler version 3.3. Newer compiler have not yet been tested. You can create the library by typing:

```
make
make install
```

You can then compile the plugin by typing:

```
cd ../plugin
make
make install-OpenHRP
```

1.2.3 Example

There is a python script sample in the plugin directory which should be symbolically linked into the OpenHRP's script directory during the previous installation phase. Its name is TestWalkGenJRL.py. If everything went well the only modification to be done is the hardcoded path for the parameters file at line 16

```
walk = ms.create("WalkGenJRL","walk","/home/username/src/OpenHRP/
PatternGeneratorJRL/src/PreviewControlParameters.ini
/home/username/src/OpenHRP/etc/HRP2JRL/ HRP2JRLmain.wrl")
```

The part **/home/username/** should be changed to the correct OpenHRP directory. They are 6 examples of different walking, just uncomment the one you are interested in.

1.2.4 Needed libraries

The current version relies on the VNL library for matrix computation. It is provided directly compiled with the headers. You also can get it through the VXL project http://vxl.sourceforge.net/

1.3 Plugin : WalkGenJRL

1.3.1 Introduction

This section explains the usage of the WalkGenJRL plugin inside the OpenHRP2 simulator. WalkGenJRL assumes that the sequence player plugin is working as well as the stabilizer. Other than that no other plugin are needed. WalkGenJRL can take support foot position sequences and create the appropriate foot, waist, ZMP trajectories as well as the joint values for every 5 ms of the total motion. Other higher functionnalities are currently under development such as specifying arc and translationnal motion for which the support foot sequences is generated automatically. WalkGenJRL can also be access as a CORBA server from other plugins and from any CORBA client. This latter functionnality is however still at an early stage, and is far from being complete.

In order to be more efficient in the way to use the plugin, it is better to have a little understanding of its structure. The WalkGenJRL plugin maintains two stacks, one is the stack of the support foot position, and the other the ZMP stack. The first one is simply a list of position and orientation of where the robot will put its feet. The second one is calculated from the first, and detail all the foot, CoM, ZMP position which have to be realized every 5 ms. The second one is used for the control loop running at 200

Hz. Some functions act on the first stack, others on the second, and some on both. Fig 1.1 summarizes this structure.

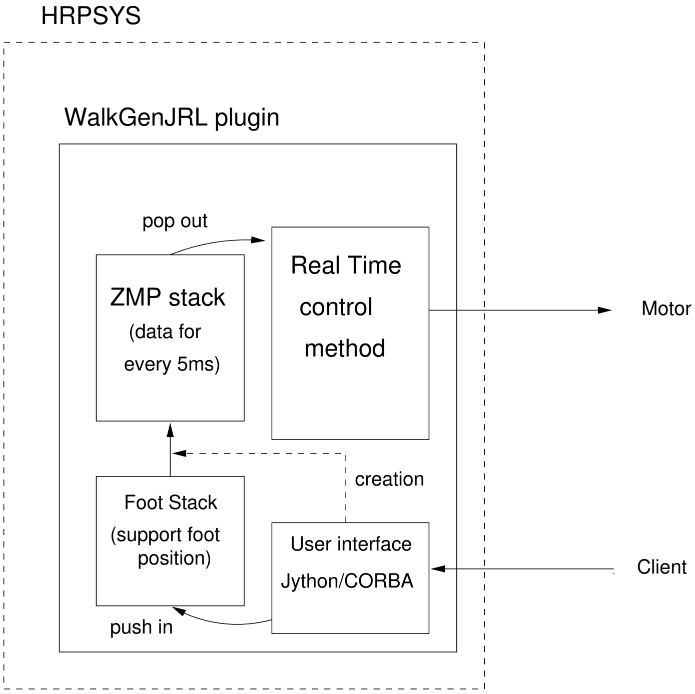

HRPSYS
WalkGenJRL plugin
pop out
ZMP stack
(data for every 5ms)
Real Time control method
Motor
creation
Foot Stack
(support foot position)
User interface
Jython/CORBA
Client
push in

Figure 1.1: Stacks organization and overview of the WalkGenJRL plugin's structure.

1.3.2 Functionnalities

This paragraph describes the functionnalities provided by WalkGenJRL.

Foot positionning

- **stepseq**: Specify the ZMP trajectory by giving relative support foot positionning. The ZMP is assumed to be under the CoM at the beginning of the motion. A relative supporting foot position is given by three parameters (dx_i, dy_i, θ_i). The parameter θ_i gives the angle by which the next foot orientation will be modified, and (dx_i, dy_i) gives the position of the next support foot position according to the direction.x Fig 1.2 gives an example of such a sequence. At first the robot moves its support towards the left foot, switch to the right foot 20 cm ahead, then to the left foot with a rotation of 10 degrees, then to the right foot with also a rotation of 10 degrees. You have to give the last motion for having the robot stop correctly. In this case, you should add the motion for bringing the left foot at the same level than the right foot. Bringing the ZMP between the two foot is done automatically by the plugin.

  The final sequence is then:

```
:stepseq 0 0.095 0 0.2 0.19 0.0 0.2 -0.19 10.0 0.2 0.19 10.0
```

```
0.0 -0.19 0.0
```

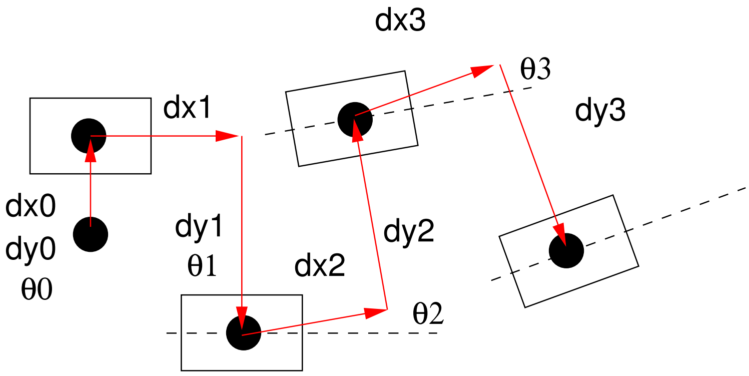


Figure 1.2: Example of foot positionning: (0 0.095 0 , 0.2 0.19 0.0, 0.2 -0.19 10.0, 0.2 0.19 10.0)

The Jython interpreter will exit from this function before the motion is completly finished. BUT the sequence will be performed as soon as the function is exited. Currently you have to wait the end of motion by using a waitInputConfirm call. It is unwise to add a new sequence while the motion is unfinished.

  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside libWalkGenJRL.

- **arc**: The robot walks tangently to a specified arc. This capability is supposed to be compatible with Kajita-San, however you might find some discrepencies. They are four parameters for this function : $(x, y, \theta, Foot)$, where (x, y) (in meters) is the center of the arc relative to the current position of the robot, θ (degrees) the arc's angle, and $Foot$ the starting support foot (-1 : Right, 1: Left). The current strategy is the following:

$$
\begin{aligned}
R &= \sqrt{x^2 + y^2} \\
NbOfSteps &= \frac{\theta R}{StepMax} \\
NbOfStepsInt &= \lfloor NbOfSteps \rfloor \\
\theta_{steps} &= \frac{StepMax}{R} \\
\theta_{last} &= \theta - \theta_{steps} * NbOfStepsInt
\end{aligned}
\tag{1.1}
$$

  The robot perform $NbOfStepsInt$ of θ_{steps} degrees. The last step is of θ_{last} degrees. $StepMax$ is the maximum step length. It is fixed by default at 0.15 cm.

  The function will exit almost immediatly, and the motion will NOT be performed unless you call the **finish** function. Before calling finish, it is strongly advice to use **lastsupport**.

The internal algorithm puts footprint on each side of an arc. Therefore the left foot trajectory and the right trajectory are on two arcs. If we note $\Delta Feet$ the distance between the two feet along the $y-axis$ (in the OpenHRP frame), then the internal arc has a ray of $R-\Delta Feet$. The external arc has a ray of $R+\Delta Feet$. Considering the case depicted in figure 1.3, and assuming that $SF=-1$ if the support is the left foot, and $SF=1$ if the support foot is the right foot, the coordinates of the support foot is :

$$\begin{aligned} x(k)_{sf} &= R+SF\Delta Feetsin(k\theta_{steps})-(R-SF\Delta Feet)sin(k\theta_{steps}) \\ y(k)_{sf} &= -(R+SF\Delta Feetcos(\theta_{steps})-(R-SF\Delta Feet)cos(k\theta_{steps})) \end{aligned}$$

In order to give the proper translation for the relative foot position between support foot $k-1$ and k, we note v_w this translation in the world reference frame (as depicted in figure 1.4). If v_i is the same vector but expressed in the reference frame of support foot k, then

$$v_w = \mathbf{R}_{k\theta_{steps}} v_i$$

One can remark that v_i is the translation (sx, sy) we are looking for. Indeed the support foot k has the proper orientation after $k\theta_{steps}$ rotation. Then by computing $v_w = [x(k)_{sf} - x(k-1)_{sf}\ y(k)_{sf} - y(k-1)_{sf}]^T$, we get

$$v_i = \mathbf{R}^{-1}_{k\theta_{steps}} v_w = \mathbf{R}^{T}_{k\theta_{steps}} v_w$$

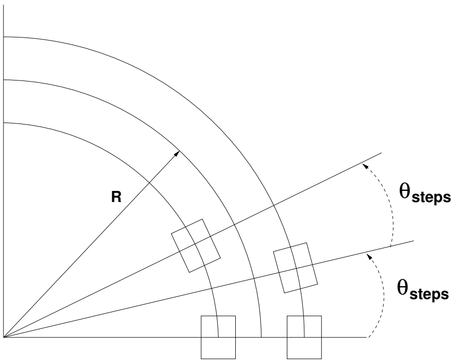


Figure 1.3: Splitting the arc into small arc of θ_{steps} degrees

  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside WalkGenJRL plugin.

- **arccentered**: With this function the robot walks orthogonaly to an arc. They are four parameters for this function : $(x, y, \theta, Foot)$, where (x, y) (in meters) is the center of the arc relative to the current position of the robot, θ (degrees) the

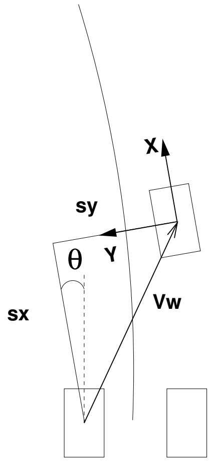


Figure 1.4: Computing v_w into the reference frame of support foot k

arc's angle, and *Foot* the starting support foot (-1 : Right, 1: Left). The current strategy is the following:

$$\begin{aligned} R &= \sqrt{x^2+y^2} \\ NbOfSteps &= \frac{\theta R}{StepMax} \\ NbOfStepsInt &= \lfloor NbOfSteps \rfloor \\ \theta_{steps} &= \frac{StepMax}{R} \\ \theta_{last} &= \theta - \theta_{steps} * NbOfStepsInt \end{aligned} \tag{1.2}$$

The robot perform $NbOfStepsInt$ of θ_{steps} degrees. The last step is of θ_{last} degrees.

The function will exit almost immediatly, and the motion will NOT be performed unless you call the **finish** function. Before calling finish, it is strongly advice to use **lastsupport**.

- **Jython availability:** yes
- **Corba availability:** no
- **Source code location:** Inside WalkGenJRL plugin.

- **lastsupport**: This function generates a finishing sequence in the foot stack in order to make sure that the robot will perform a half-sitting position.

  The function will exit almost immediatly, and the motion will NOT be performed unless you call the **finish** function.

  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside WalkGenJRL plugin.

- **finish**: This function will send the complete foot stack into the ZMP stack to be run by control loop. The function will return only after the completion of the motion.
  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside WalkGenJRL plugin.

Walking parameters

- **omega**: Modify the angle for the foot landing and taking-off. This feature was intended to modify the angle with which the robot land or take-off the support foot. It is currently set by default to 0. It has never been tried yet into the real robot.
  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside ZMPDiscretization.cpp (libWalkGenJRL).
- **stepheight**: Modify the height with which the robot will perform a step. It is currently set by default to 0.12 m. Has been tested on the real robot. The unit is in meters.
  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside ZMPDiscretization.cpp (libWalkGenJRL).
- **singlesupporttime**: Modify the duration of the single support time. The current rule is to have

  $$singlesupporttime + doublesupporttime = 0.8s \tag{1.3}$$

  Has been tested on the real robot. The default value is 0.78 The unit is in seconds.
  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside ZMPDiscretization.cpp (libWalkGenJRL).
- **doublesupporttime**: Modify the duration of the double support time. The current rule is to have

  $$singlesupporttime + doublesupporttime = 0.8s \tag{1.4}$$

  Has been tested on the real robot. The default value is 0.02. The unit is in seconds.
  - **Jython availability:** yes
  - **Corba availability:** no
  - **Source code location:** Inside ZMPDiscretization.cpp (libWalkGenJRL).

Chapter 2

The theory behind

2.1 Inverse Kinematics

2.1.1 The legs

The main purpose of this paragraph is to compute the joint values for a given position and orientation of an hip joint (RLEG_JOINT0 or LLEG_JOINT0) and the corresponding foot. From the specification of the HRP-2 robot, the right leg of the robot can be represented by a sticky drawing depicted in figure 2.1. They are some slight differences compare to the part described in Kajita [3]. Indeed in this case, the leg of the robot can be seen as rotating around the ankle of the foot, until it is tangent to a sphere D around the hip position. Therefore the segment $(P2, P1)$ is orthogonal to the

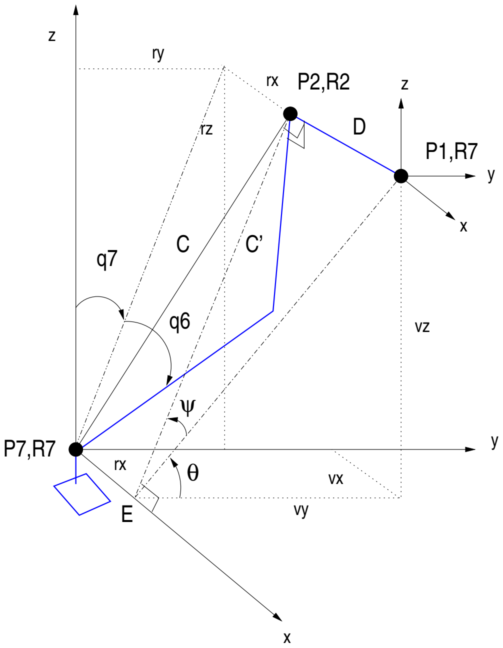


Figure 2.1: Inverse Kinematics for the HRP-2 robot

leg plane. If $P1$ is expressed in the reference frame of $P7$, it is then quite simple to find C', θ and Ψ. Indeed, assuming that $P1 = v_x, v_y, v_z$ and $P2 = r_x, r_y, r_z$ then :

$$\begin{aligned}
r_x &= v_x \\
C' &= \sqrt{v_x^2 + v_z^2 - D^2} \\
\psi &= atan(D, C') \\
\theta &= atan(v_z, v_y) \\
r_y &= cos(\psi + \theta) * C_p \\
r_z &= sin(\psi + \theta) * C_p
\end{aligned} \tag{2.1}$$

where C' is the distance between P_2 and the point E. E is intersecting P_7's x-axis and the plan orthogonal to this axis, which includes P_2 and P_7. ψ is the angle between (E, P_1) and (E, P_2)). θ is the angle between (E, P_1) and the (x, y)-plane.

Once all the values r_x, r_y, r_z are known it is possible to reuse the same arguments than in [2]. Indeed:

$$C = \sqrt{r_x^2 + r_y^2 + r_z^2}$$

It is then possible to compute q_5 as:

$$q_5 = -cos^{-1}(\frac{A^2 + B^2 - C^2}{2AB}) + \pi$$

$$\frac{C}{sin(\pi - q_5)} = \frac{A}{sin\alpha}$$

$$\alpha = asin(\frac{Asin(\pi - q_5)}{C})$$

$$\begin{aligned}
q_5 &= atan2(r_y, r_z) \\
q_6 &= -atan2(r_x, sign(r_z)\sqrt{r_x^2 + r_y^2} - \alpha
\end{aligned}$$

$$\mathbf{R}_7 = \mathbf{R}_1 \mathbf{R}_z(q_2) \mathbf{R}_x(q_3) \mathbf{R}_y(q_4) \mathbf{R}_y(q_5 + q_6) \mathbf{R}_x(q_7)$$

$$\mathbf{R}_z(q_2) \mathbf{R}_x(q_3) \mathbf{R}_y(q_4) = \mathbf{R}_1^T \mathbf{R}_7 \mathbf{R}_x(q_7) \mathbf{R}_y(q_5 + q_6)$$

Then:

$$\begin{bmatrix} c_2c_4 - s_2s_3s_4 & -s_2c_3 & c_2s_4 + s_2s_3c_4 \\ s_2c_4 + c_2s_3s_4 & c_2c_3 & s_2s_4 - c_2s_3c_4 \\ -c_3s_4 & s_3 & c_3c4 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

2.2 Dynamic models of biped robot

2.2.1 3D Linear Inverted Pendulum Mode and Zero-moment point

“In this context a constraint control is applied to an inverted pendulum, such that the mass should move along an arbitrary predefined plane. The subsequent model is called the *Three-Dimensional Linear Inverted Pendulum Model* (3D-LIPM). We take Cartesian coordinates as shown in figure 2.2 and specifiy the x-axis as the ordinal walking direction. The constraint plane is represented with given normal vector $(k_x, k_y, -1)$ and z intersection z_c as:

$$z = k_x x + k_y y + z_c \tag{2.2}$$

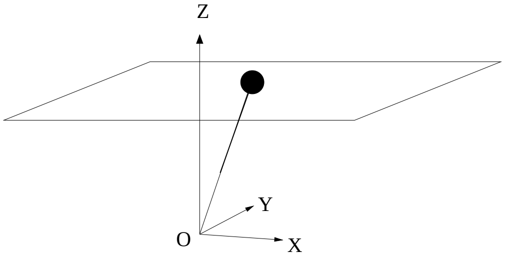


Figure 2.2: Pendulum under constraint

If the constraint plane is horizontal ($k_x = k_y = 0$), the dynamics under the constraint control is given by

$$\ddot{y} = \frac{g}{z_c} y - \frac{1}{m z_c} \tau_x \tag{2.3}$$

$$\ddot{x} = \frac{g}{z_c} x - \frac{1}{m z_c} \tau_y \tag{2.4}$$

where m is the mass of the pendulum, g is the gravity acceleration and τ_x, τ_y are the torques around x-axis and y-axis respectively. This pendulum under constraint is depicted in figure 2.2.

Even in the case of the sloped constraint where k_x,$k_y \neq 0$, we can obtain the same dynamics by applying additional constraint

$$\tau_x x + \tau_y y = 0 \tag{2.5}$$

for the input torques.

Equations 2.3 and 2.4 are linear equations. The only parameter which govens those dynamics is z_c, i.e., the z intersection of the constraint plane and the inclination of the plane never affects the horizontal motion.

For the 3D-LIPM with the horizontal constraint ($k_x = k_y = 0$), we can easily calculate the zero-moment point (ZMP), which is widely used in biped robot research [].

$$p_x = -\frac{\tau_y}{mg} p_y = \frac{\tau_x}{mg} \tag{2.6}$$

where (p_x, p_y) is the location of the ZMP on the floor. By substituing equations 2.6 to the 3D-LIPM (2.3 and 2.4) we obtain:

$$\ddot{y} = \frac{g}{z_c}(y - p_y) \tag{2.7}$$

$$\ddot{x} = \frac{g}{z_c}(x - p_x) \tag{2.8}$$

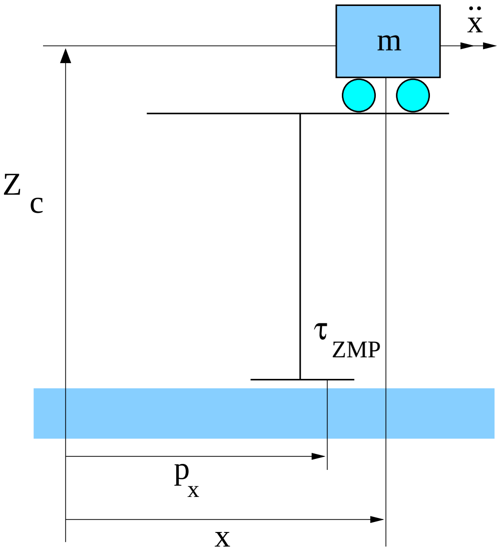


Figure 2.3: The Cart table model

2.2.2 ZMP equations and cart-table model

To control the ZMP, it should appears in the outputs of the system formula. However in 3D-LIPM as described in the last section, it appears to be the input of the system. Therefore equations 2.7 and 2.8 are rewritten to have the ZMP with the following form:

$$p_y = y - \frac{z_c}{g}\ddot{y} \tag{2.9}$$

$$p_x = x - \frac{z_c}{g}\ddot{x} \tag{2.10}$$

In the remainder of this chapter, we will refer to the above equations as the *ZMP equations*.

Figure 2.3 shows a suggestive model directly corresponds to these equations. It depicts a running cart of mass m on a pedestal table whose mass is negligible (we need two sets of a cart on a table for the motion x and y).

As shown in the figure, the foot of the table is too small to let the cart stay on the edge. However if the cart accelerates with a proper rate, the table can keep upright for a while. At this moment, the ZMP exists inside of the table's foot. Since the moment around the ZMP must be zero, we have:

$$\tau_{zmp} = mg(x - p_x) - m\ddot{x}z_c = 0 \tag{2.11}$$

We can verify that this yields to the equation 2.10.

2.3 Walking pattern generation for a given ZMP

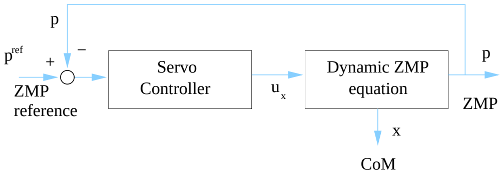


Figure 2.4: Pattern Generation as ZMP tracking control

2.3.1 Pattern generation as an inverse problem

When we represent a robot as the cart-table model and give the cart motion as the trajectory of the center of mass (CoM) of the robot, we can easily calculate the resulted ZMP by using the ZMP equations 2.9 and 2.10. On the other hand, a walking pattern generation is the inverse problem of this. That is, the cart motion should be calculated from the given ZMP trajectory which is determined by the desired footholds and step period.

Takanishi et al. proposed to solve this problem by using Fourier Transformation. By applying the Fast Fourier Transformation (FFT) to the ZMP reference, the ZMP equations can be solved in frequency domain. Then the inverse FFT returns the resulted CoM trajectory into time domain.

Kagami, Nishiwaki et al. proposed a method to solve this problem in the discrete time domain. They showed that the ZMP equation can be discretized as a trinomial expression, and it can be efficiently solved by an algorithm of $O(N)$ for the given reference data of size N.

2.3.2 ZMP control as a servo problem

Let us define a new variable u_x as the time derivate of the horizontal acceleration of CoM.

$$\frac{d}{dt}\ddot{x} = u_x \tag{2.12}$$

Regarding u_x as the input of equation 2.10, it is possible to translate the ZMP equation into a strictly proper dynamical system as:

$$
\begin{aligned}
\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_x \\
p_x &= [1\ 0\ -\frac{z_c}{g}]\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}
\end{aligned}
\tag{2.13}
$$

For equation 2.9, u_y is defined in the same manner, and a similar system is found.

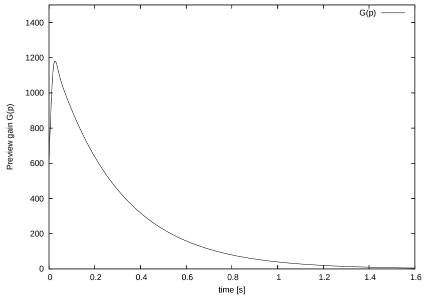


Figure 2.5: Preview controller gain G_p ($T = 5$[ms], $z_c = 0.814$ [m])

By using the dynamics of equation 2.13 we can construct a walking pattern generator as a ZMP tracking control system. The system generates the CoM trajectory such that the resulted ZMP follows the given reference. However, we must consider an interesting feature of this problem as follows. Figure illustrates the ideal trajectories of the ZMP and the CoM of a robot that walks one step forward dynamically. The robot supports its body by hind-leg from 0s to 1.5s, and has a support exchange at 1.5s followed by the foreleg support until 3.0s. Thus the reference ZMP should have a step change at 1.5s and obviously the CoM must start moving *before* this. Assuming the controller in Figure 2.4 , the output must be calculated from the future input !

Although this sounds curious, we do not have to violate the law of causality. Indeed we are familiar with such situation in driving on a winding road, where we steer a car by watching ahead, that is, watching the future reference.

A control that utilizes future information was first proposed by Sheridan in 1966 and was named “Preview control” . In 1969, Hayase and Ichikawa worked on the same concept and solved a linear quadratic (LQ) optimal servo controller with preview action. A digital version of LQ optimal preview controller was developed by Tomizuka and Rosenthal in 1979 and was completed as the controller for MIMO system by Katayama et al. in 1985.

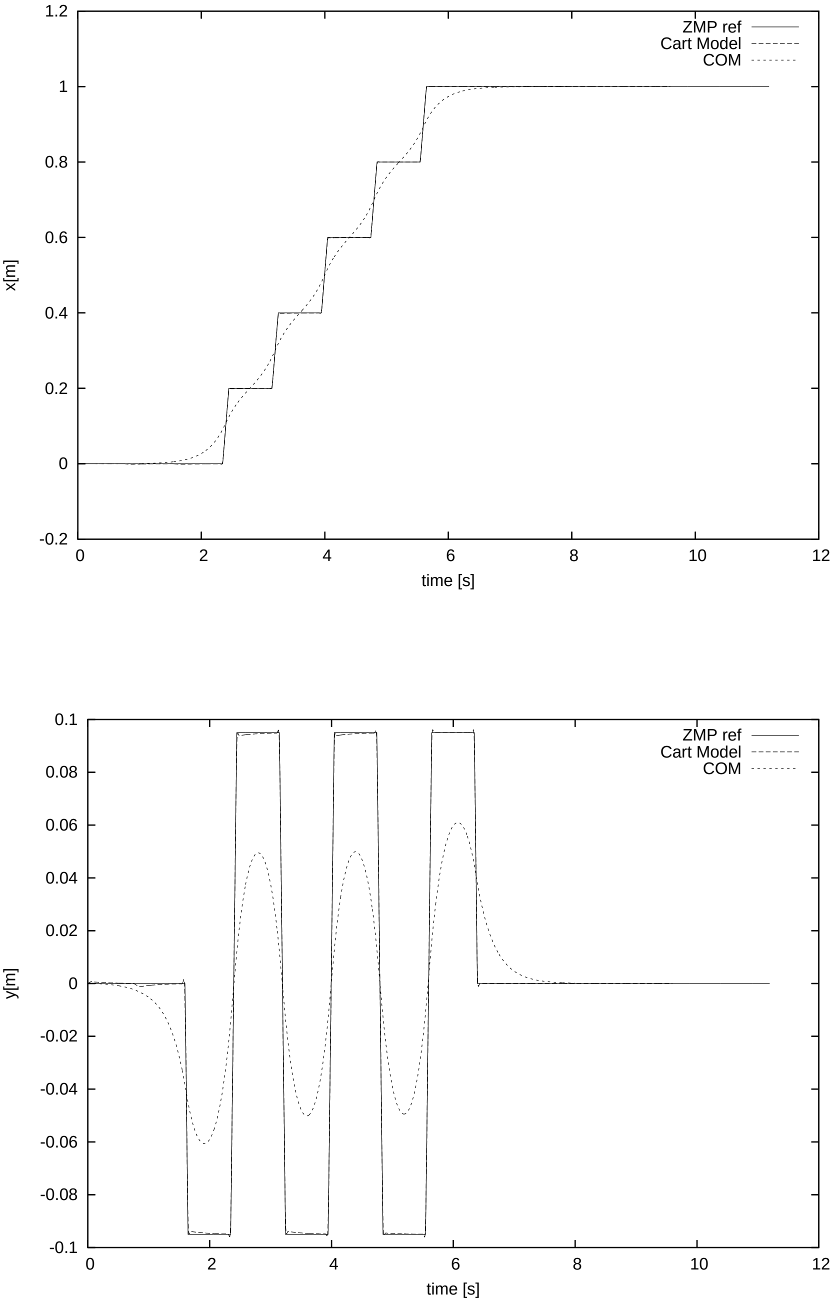


Figure 2.6: Body trajectory obtained by preview control, previewing period $T * NL = 1.6$

2.3.3 Pattern generation by preview control

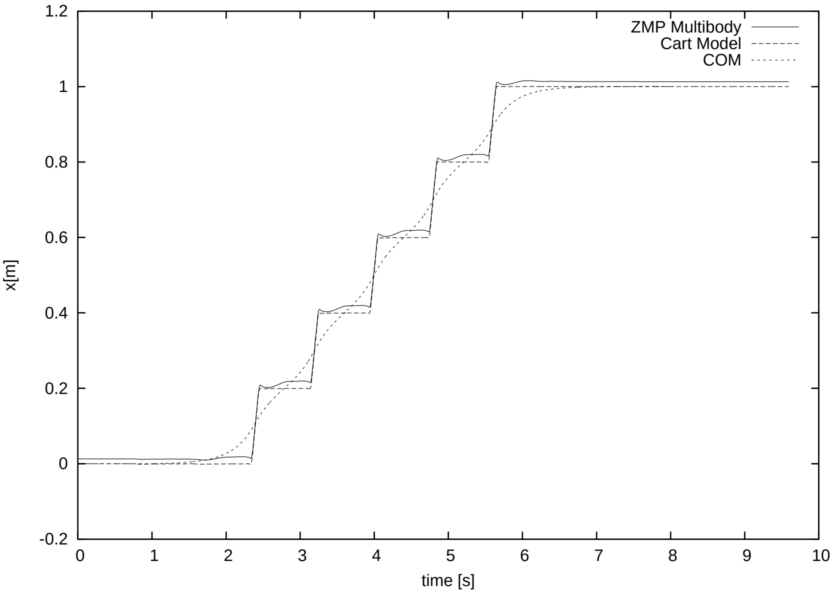
Let us design an optimal preview servo controller following the method proposed by Katayama et al.

First, we discretize the system of equation 2.13 with sampling time of T as:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}u(k), \\ p(k) &= \mathbf{C}\mathbf{x}(k), \end{aligned} \tag{2.14}$$

where

$$\begin{aligned} \mathbf{x}(\mathbf{k}) &\equiv [\mathbf{x}(k)\ \dot{\mathbf{x}}(k)\ \ddot{\mathbf{x}}(kT)]^T, \\ u(k) &\equiv \mathbf{u}_x(kT), \\ p(k) &\equiv \mathbf{p}_x(kT), \\ \mathbf{A} &\equiv \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \\ \mathbf{B} &\equiv \begin{bmatrix} T^3/6 \\ T^2/2 \\ T \end{bmatrix}, \qquad \mathbf{C} \equiv \begin{bmatrix} 1\ 0\ \frac{-z_c}{g} \end{bmatrix} \end{aligned} \tag{2.15}$$



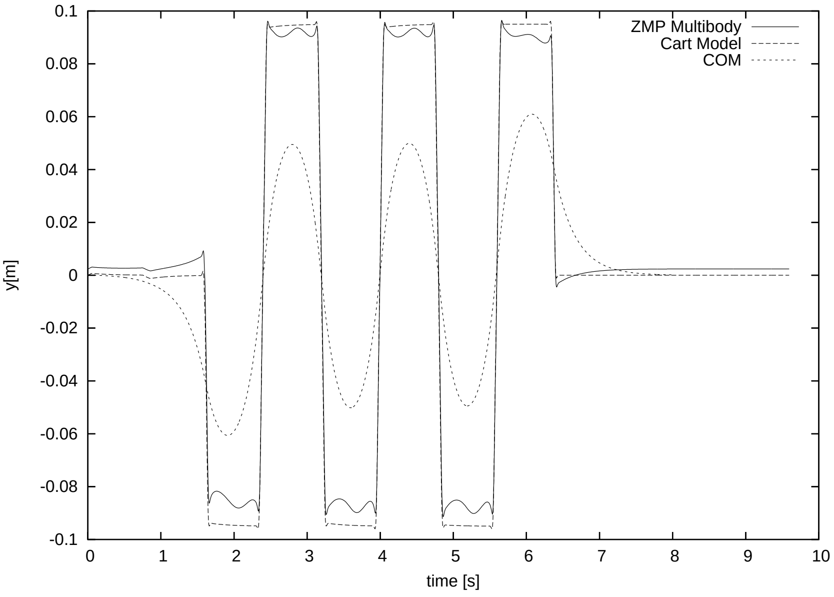


Figure 2.7: Body trajectory obtained by preview control, previewing period $T * NL = 1.6$

With the given reference of ZMP $p^{ref}(k)$, the performance index is specified as:

$$J = \sum_{i=k}^{\infty} \{Q_e e(i)^2 + \Delta \mathbf{x}^T \mathbf{Q}_x \Delta \mathbf{x}(i) + R \Delta u^2(i)\} \tag{2.16}$$

where $e(i) \equiv p(i) - p^{ref}(i)$ is the servo error, $Q_e, R > 0$ and $\mathbf{Q}_x$ is a 3×3 symmetric non-negative definite matrix. $\Delta \mathbf{x}(k) \equiv \mathbf{x}(k)\mathbf{x}(k-1)$ is the incremental state vector and $\Delta u(k) \equiv u(k) - u(k-1)$ is the incremental input.

When the ZMP reference can be previewed for N_L step future at every sampling time, the optimal controller which minimizes the performance index 2.16 is given by:

$$u(k) = -G_1 \sum_{i=0}^{k} e(i) - G_2 \mathbf{x}(k) - \sum_{j=1}^{N_L} G_p(j) p^{ref}(k+j) \tag{2.17}$$

where G_1, G_2 and $G_p(j)$ are the gains calculated from the weights $Q_e, \mathbf{Q}_x, R$ and the system parameter of 2.14.

The preview control is made of three terms, the integral action on the tracking error, the state feedback and the preview action using the future reference.

Figure 2.5 shows the gain for the preview action. We see the controller does not need the information of far future because the magnitude of the preview gain G_p becomes very small in the future farther than 2 seconds.

Figure 2.6 is an example of walking pattern generation with the previewing period of 1.6s. The upper graph is the sagittal motion along the x-axis and the lower graph is the lateral motion along the y-axis. We can see a smooth trajectory of CoM (dashed line) is generated and the resulted ZMP (bold line) follows the reference (thin line) with good accuracy. The generated walking pattern corresponds to the walking of three steps forward. The ZMP reference is designed to stay in the center of support foot during single support phase, and to move from an old support foot to a new support foot during double support phase. To obtain a smooth ZMP trajectory in double support, we filtered the generated ZMP by an Infinite Impulse Response filter.

Figure 8 is the result with the previewing period of 0.8s, which is not sufficient for the ZMP tracking. In this case, the resulted ZMP (bold line) does not follow the reference (thin line) well. We observe undershooting in the sagittal motion and overshooting in the lateral motion. It should be noted that even ZMP tracking performance is poor, the system still remains stable thanks to term of the state feedback.

2.3.4 Pattern generation for multibody model

The walking pattern is calculated by solving an inverse kinematics such that the CoM of the robot follows the output of the preview controller. As the simpler implementation, we can also use the center of the pelvis link since it approximates the motion of the CoM.

If the ZMP error becomes too big relative to the stability margin determined by the foot geometry, the robot can fall. To fix the ZMP error, again we can use the preview control. That is, we first calculate the CoM trajectory from the table-cart model and obtain an expected ZMP error from the multibody model. These information are stored to the buffer memory and loaded to be used after a delay time of $T * N_L$. This way, we can use the *future ZMP error* for the preview control to calculate a proper compensation. Indeed by substraction a Δ_{ZMP} is computed and put inside a preview controller. Because the discrete dynamic system is linear the variation of the CoM can be directly added to the cart-model CoM.

Figure 2.10, shows the improved pattern of ZMP by this method. We can observe now the ZMP computed by the multibody model (bold line) follows well the reference ZMP of the cart-table model (thin line). The maximum ZMP error was 1.2 cm in the x-direction and $0.4cm$ in y-direction. We used the previewing period $T * N_L = 1.6$ (s) for this calculation. In [2] a shorter period was used and it happend to be effective enough because the amount of the compensation is small.

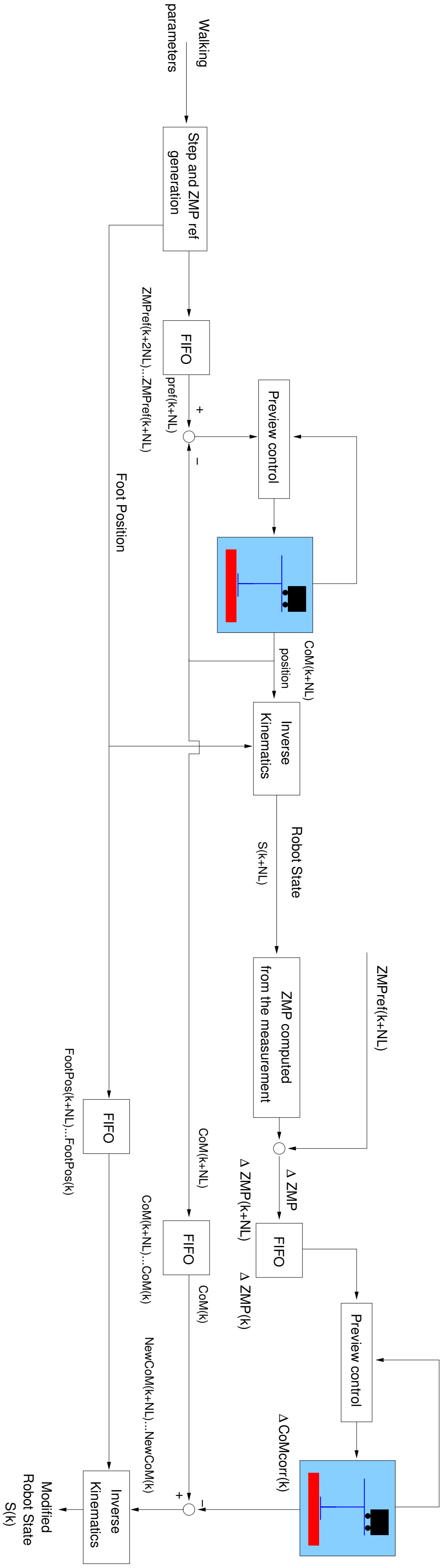
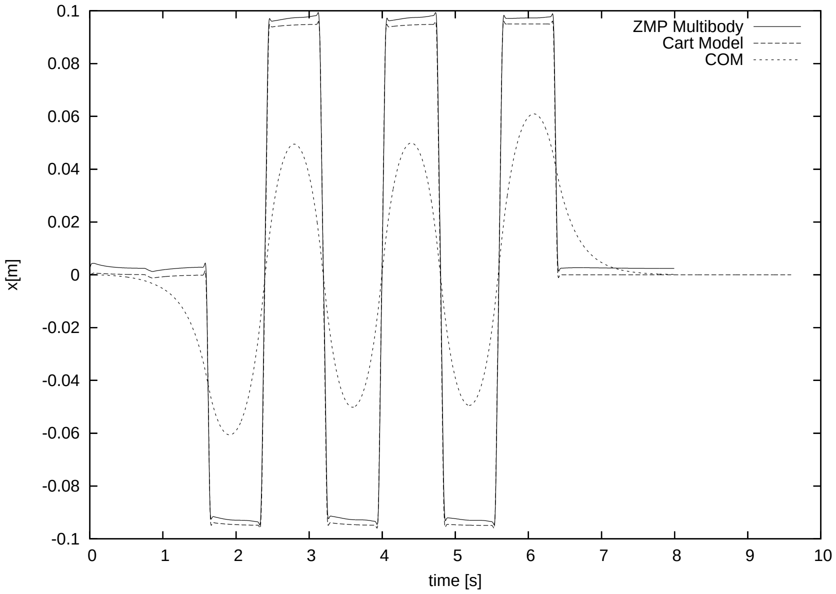

Walking parameters
Step and ZMP ref generation
FIFO
ZMPref(k+2NL)...ZMPref(k+NL)
pref(k+NL)
+
–
Preview control
Foot Position
CoM(k+NL) position
Inverse Kinematics
Robot State
S(k+NL)
ZMP computed from the measurement
ZMPref(k+NL)
Δ ZMP
Δ ZMP(k+NL)
FIFO
Δ ZMP(k)
Preview control
Δ CoMcorr(k)
CoM(k+NL)
FIFO
CoM(k+NL)...CoM(k)
CoM(k)
NewCoM(k+NL)...NewCoM(k)
+
–
FIFO
FootPos(k+NL)...FootPos(k)
Inverse Kinematics
Modified Robot State S(k)

Figure 2.8: ZMP preview control taking into account the ZMP multibody



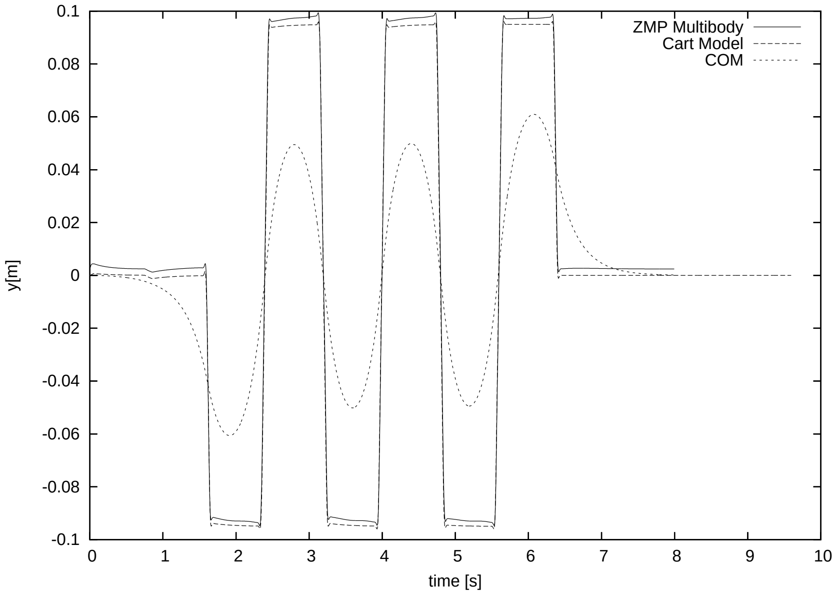


Figure 2.9: Final ZMP after modification of the CoM

2.3.5 References

If you wish to generate new walking criterion, here are some reference curves.

Figure 2.10: Articular value for the right legs

2.3.6 The angular momentum problem

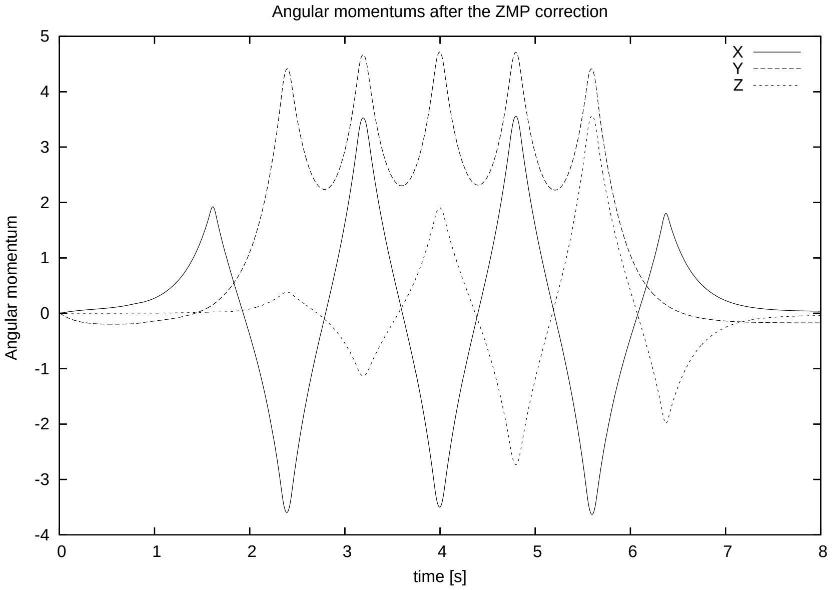


Figure 2.11: Angular Momentum after ZMP correction

The main problem related to this walking pattern is the generation of unwelcome angular momentum around the z-axis. It creates strong rotation for the robot during double-support phase. This is partly due to the fact that in the previous part, the upper body motion has been totally ignored. They are two possibilities to solve the problem: either design a heuristic aiming at compensate for the z-axis momentum, either implement a proper control strategy to move the free upper-body's joints. A heuristic involving only the arms is described, and the remaining part of this document will described the concept of *resolved momentum control.*

2.3.7 Arm motion heuristic

The main idea is to generate a motion with the arms which compensate the Z-axis moment by doing an arm movement. The basic idea is to generate a motion where the hand movement has a linear relationship with the foot of the same side in the waist coordinates.

2.4 Finding the weights for the preview control

In this section we recall some well-known results in control theory in order to compute the gains for the ZMP preview control. Those notes are based on Kajita's book [3] p.147.

2.4.1 The general scheme [1]

The discrete system is defined by three matrix A, b, c such as :

$$\begin{aligned} \mathbf{x}_{k+1} = & \quad \mathbf{A}\mathbf{x}_k + \mathbf{b}u_k \\ p_k = & \quad \mathbf{c}\mathbf{x}_k \end{aligned} \tag{2.18}$$

The optimal critera considered here is :

$$J = \sum_{j=1}^{\infty} \{ Q(p_j^{ref} - p_j)^2 + Ru_j^2 \} \tag{2.19}$$

where Q and R are also given as inputs.
the solution is then

$$\begin{aligned} \mathbf{K} &\equiv (R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}\mathbf{A} \\ f_i &\equiv (R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T(\mathbf{A} - \mathbf{b}\mathbf{K})^{T*(i-1)}\mathbf{c}^T Q \end{aligned} \tag{2.20}$$

where **P** is solution of the following Riccati equation:

$$\mathbf{P} = \mathbf{A}^T\mathbf{P}\mathbf{A} + \mathbf{c}^T Q\mathbf{c} - \mathbf{A}^T\mathbf{P}\mathbf{b}(R + \mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}\mathbf{A} \tag{2.21}$$

The computation of the solution can be done by using a software implementing the method described in [4]. For this we used the publicly available library called NetLib. In order to find **P**, it is necessary to build the Hamiltonian matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{c}^T Q\mathbf{c} & \mathbf{I} \end{bmatrix}, \mathbf{E} = \begin{bmatrix} \mathbf{I} & \mathbf{G} \\ \mathbf{0} & \mathbf{A}^T \end{bmatrix} \tag{2.22}$$

with $\mathbf{G} = \mathbf{b}R^{-}1\mathbf{b}^T$, and I the identity matrix. The function returns the left **V** and right **U** Schur vectors, and **P** is such that :

$$\mathbf{P} = U_{21}U_{11}^{-1} \tag{2.23}$$

2.4.2 Removing the offset of the ZMP

The offset of the ZMP is directly dependant of the CoM's position. Therefore in order to remove this problem, the derivated system is considered:

$$\begin{aligned} \mathbf{x}^*_{k+1} = & \quad \tilde{\mathbf{A}}\mathbf{x}_k + \tilde{\mathbf{b}}\Delta u_k \\ p_k = & \quad \tilde{\mathbf{c}}\mathbf{x}^*_k \end{aligned} \tag{2.24}$$

with

$$\begin{aligned} \Delta u_k &\equiv u_k - u_{k-1} \\ \Delta \mathbf{x}_k &\equiv \mathbf{x}_k - \mathbf{x}_{k-1} \\ \mathbf{x}_k^* &\equiv \begin{bmatrix} p_k \\ \Delta \mathbf{x}_k \end{bmatrix} \end{aligned} \tag{2.25}$$

2.4.3 Implementation of the weights computation

The resolution of the Riccati equation is taken from [5], and is based on a Schur form of the **P** matrix defined as:

$$\mathbf{Laub_Z} = \begin{pmatrix} \mathbf{A} + \mathbf{G}\mathbf{A}^{-T}\mathbf{c}^T Q\mathbf{c} & -\mathbf{G}\mathbf{A}^{-T} \\ -\mathbf{A}^{-T}\mathbf{c}^T Q\mathbf{c} & -\mathbf{A}^{-T} \end{pmatrix} \tag{2.26}$$

2.5 Momentum Equation

The following are notes from Kajita et al [6].

2.5.1 Momentum and joint velocities

We represent a humanoid robot as a mechanism of tree structure whose root is a free-flying base link (pelvis), a rigid body having 6 D.O.F. in 3D space. We define the trame Σ_B embedded in the base link whose translational velocity and angular velocity are $\mathbf{v}_b$ and $\mathbf{w}_b$, respectively. In addition, we define a $n \times 1$ column vector $\dot{\theta}$ which contains velocities of all joints as its elements where n is the total number of joints. The linear momentum **P** (3×1) and the angular momentum (4×1) of the whole mechanism are given by:

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{L} \end{bmatrix} = \begin{bmatrix} \tilde{m}\,\mathbf{E} & -\tilde{m}\,\hat{\mathbf{r}}_{B\to\tilde{c}} & \mathbf{M}_{\dot{\theta}} \\ \mathbf{0} & \tilde{\mathbf{I}} & \mathbf{H}_{\dot{\theta}} \end{bmatrix} \begin{bmatrix} \mathbf{v}_B \\ \mathbf{w}_B \\ \dot{\theta} \end{bmatrix} \tag{2.27}$$

where $\tilde{m}$ is the total mass of the robot, **E** is an identity matrix of size 3×3, $\mathbf{r}_{B\to\tilde{c}}$ is the 3×1 vector from the base link to the total center of mass (CoM) and $\tilde{\mathbf{I}}$ is the 3×3 inertia matrix with respect to the CoM. $\mathbf{M}_{\dot{theta}}$ and $\mathbf{H}_{\dot{\theta}}$ are the $3 \times n$ inertia matrices which indicate how the joint speeds affect to the linear momentum and the angular momentum respectively. îs an operator which translates a vector of 3×1 into a skew symmetric 3×3 matrix which is equivalent to a cross product.

In this paper, we assume all vectors of position, velocity and angular velocity are represented in the Cartesian frame Σ_O fixed on the ground.

2.5.2 Constraints of foot contact

Equation 2.27 gives the total momentum of a flying robot. However, when the robot is in contact with the ground we must take into account of constraints that reduce the total D.O.F. of the system. The foot velocities $(\mathbf{v}_{F_i}, \mathbf{w}_{F_i}$ of the frame Σ_{F_i} $(i = 1, 2)$ are given by

$$\begin{bmatrix} \mathbf{v}_{F_i} \\ \mathbf{w}_{F_i} \end{bmatrix} = \begin{bmatrix} \mathbf{E} & \hat{\mathbf{r}}_{B\to F_i} \\ \mathbf{0} & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{v}_B \\ \mathbf{w}_B \end{bmatrix} + \mathbf{J}_{leg_i}\dot{\theta}_{leg_i} \tag{2.28}$$

where $\mathbf{J}_{leg_i}$ is a Jacobian matrix $(6 \times 6\ )$ calculated from the leg configuration, $\mathbf{r}_{B\to F_i}$ is a position vector (3×1) from the base frame to the foot frame and $\dot{\theta}_{leg_i}$ is the joint speed vector (6×1) of each leg. If $\mathbf{J}_{leg_i}$ $(i = 1,2)$ are non singular, the vectors for the leg are given by:

$$\dot{\theta}_{leg_i} = \mathbf{J}_{leg_i}^{-1} \begin{bmatrix} \mathbf{v}_{F_i} \\ \mathbf{w}_{F_i} \end{bmatrix} - \mathbf{J}_{leg_i}^{-1} \begin{bmatrix} \mathbf{E} & \hat{\mathbf{r}}_{B\to F_i} \\ \mathbf{0} & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{v}_B \\ \mathbf{w}_B \end{bmatrix} \tag{2.29}$$

Let us divied the whole joint speed vector into the leg parts and the free part as:

$$\dot{\theta} = [\dot{\theta}_{leg_1}^T \dot{\theta}_{leg_2}^T \dot{\theta}_{free}^T]^T \tag{2.30}$$

where $\dot{\theta}_{free}$ is the joint speed for waist, arms and head. We also divide the corresponding inertia matrices as:

$$\begin{aligned} \mathbf{M}_{\dot{\theta}} &= [\mathbf{M}_{leg_1}\ \mathbf{M}_{leg_2}\ \mathbf{M}_{free}], \\ \mathbf{H}_{\dot{\theta}} &= [\mathbf{H}_{leg_1}\ \mathbf{H}_{leg_2}\ \mathbf{H}_{free}]. \end{aligned} \tag{2.31}$$

Then we can rewrite the momentum equations 2.27 as:

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{L} \end{bmatrix} = \begin{bmatrix} \tilde{m}\,\mathbf{E} - \tilde{m}\,\hat{\mathbf{r}}_{B\to\tilde{c}} \\ \mathbf{0}\tilde{\mathbf{I}} \end{bmatrix} \begin{bmatrix} \mathbf{v}_B \\ \mathbf{w}_B \end{bmatrix} + \sum_{i=1}^{2} \begin{bmatrix} \mathbf{M}_{leg_i} \\ \mathbf{H}_{leg_i} \end{bmatrix} \dot{\theta}_{leg_i} + \begin{bmatrix} \mathbf{M}_{free} \\ \mathbf{H}_{free} \end{bmatrix} \dot{\theta}_{free} \tag{2.32}$$

By substituing 2.29 into 2.32, we obtain the momentum equation under the constraint as:

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{L} \end{bmatrix} = \begin{bmatrix} \mathbf{M}_B^* & \mathbf{M}_{free} \\ \mathbf{H}_B^* & \mathbf{H}_{free} \end{bmatrix} \begin{bmatrix} \xi_B \\ \dot{\theta}_{free} \end{bmatrix} + \sum_{n=1}^{2} \begin{bmatrix} \mathbf{M}_{F_i}^* \\ \mathbf{H}_{F_i}^* \end{bmatrix} \dot{\xi}_{F_i}, \tag{2.33}$$

where

$$\begin{aligned} \xi_B &\equiv [\mathbf{v}_B^T\ \mathbf{w}_B^T]^T \\ \xi_{F_i} &\equiv [\mathbf{v}_{F_i}^T\ \mathbf{w}_{F_i}^T]^T \\ \begin{bmatrix} \mathbf{M}_B^* \\ \mathbf{H}_B^* \end{bmatrix} &\equiv \begin{bmatrix} \tilde{m}\,\mathbf{E} & -\tilde{m}\,\hat{\mathbf{r}}_{B\to\tilde{c}} \\ \mathbf{0} & -\tilde{\mathbf{I}} \end{bmatrix} - \sum_{n=1}^{2} \begin{bmatrix} \mathbf{M}_{F_i}^* \\ \mathbf{H}_{F_i}^* \end{bmatrix} \begin{bmatrix} \tilde{m}\,\mathbf{E} & -\tilde{m}\,\hat{\mathbf{r}}_{B\to F_i} \\ \mathbf{0} & \mathbf{E} \end{bmatrix}, \\ \begin{bmatrix} \mathbf{M}_{F_i}^* \\ \mathbf{H}_{F_i}^* \end{bmatrix} &\equiv \begin{bmatrix} \mathbf{M}_{leg_i} \\ \mathbf{H}_{leg_i} \end{bmatrix} \mathbf{J}_{leg_i}^{-1}. \end{aligned}$$

The second term in the right hand side of 2.33 indicates the extra momentum generated by specifying the foot speed.

2.6 Resolved Momentum Control

2.6.1 Setting momentum reference

For every mechnical system, no matter how its structure or behavior is complicated, we can determine the position of the CoM $\tilde{\mathbf{c}}$, the linear momentum $\mathbf{P}$ and the angular momentum $\mathbf{L}$ for the total mechanism.

Dividing the total linear momentum $\mathbf{P}$ by the total mass $\tilde{m}$, we obtain the translational speed of the CoM:

$$\frac{d}{dt}\tilde{\mathbf{c}} = \frac{\mathbf{P}}{\tilde{m}} \tag{2.34}$$

Thus, we can control the position of the CoM by manpulating the linear momentum.

As the extension of this, we propose a method of control or pattern generation by manipulating the total (linear and angular) momentum. Let us call this method the *Resolved Momentum Control.* The reference motion of a humanoid robot can be specified by assuming a rigid body whose mass and moment of inertia are equal to the target. However, we cannot associate the orientation of this imaginary object to the robot, since a rigid body can not hold enough information to represent the internal structure of the multi-body system.

2.6.2 Momentum selection and control by pseudo-inverse

In many applications, we do not have to specify all six elements of the momentum. Moreover, in some case we encounters a numerical unstability by specifying the all elements of the reference momentum. Therefore we introduce a selection matrix $\mathbf{S}$ which is $l \times 6 (0 \leq l \leq 6)$ to pick up the elements of the momentum to be controlled. The selection of the momentum is given by

$$\mathbf{S} \equiv \begin{bmatrix} \mathbf{e}_{s_1}^T \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{e}_{s_l}^T \end{bmatrix}, \tag{2.35}$$

where $\mathbf{e}_{s_i}$ is a column vector of 6×1 that has one at s_i-th row and zeros for the rest. s_i specifies the element of the momentum we want to pick up. Transposing the second term of 2.32 from the right side to the left side, then multiplying $\mathbf{S}$ from left, we obtain the following equation:

$$\mathbf{y} = \mathbf{A} \begin{bmatrix} \xi_B \\ \dot{\theta}_{free} \end{bmatrix} \tag{2.36}$$

where

$$\mathbf{y} = \mathbf{S} \left\{ \begin{bmatrix} \mathbf{P}^{ref} \\ \mathbf{L}^{ref} \end{bmatrix} - \Sigma_{n=1}^{2} \begin{bmatrix} \mathbf{M}_{F_i}^* \\ \mathbf{H}_{F_i}^* \end{bmatrix} \xi_{F_i}^{ref} \right\} \tag{2.37}$$

,

$$\mathbf{A} \equiv \mathbf{S} \begin{bmatrix} \mathbf{M}_B^* & \mathbf{M}_{free} \\ \mathbf{H}_B^* & \mathbf{H}_{free} \end{bmatrix} \tag{2.38}$$

Here $\mathbf{P}^{ref}$ is the reference linear momentum, $\mathbf{L}^{ref}$ is the reference angular momentum and $\xi_{F_i}^{ref}$ is the reference velocity for each foot.

Using 2.36, the target speed which realizes the reference momentum and the speed $(\xi_B^{ref}, \dot{\theta}^{ref})$ is calculated as the least square solution by:

$$\begin{bmatrix} \xi_B \\ \dot{\theta}_{free} \end{bmatrix} = \mathbf{A}^{\dagger}\mathbf{y} + (\mathbf{E} - \mathbf{A}^{\dagger}\mathbf{A}) \begin{bmatrix} \xi_B^{ref} \\ \dot{\theta}_{free}^{ref} \end{bmatrix} \tag{2.39}$$

, where $\mathbf{A}^{\dagger}$ is a pseudo-inverse (the least-squares inverse) of $\mathbf{A}$. This equation gives the Resolved Momentum Control.

2.6.3 Calculation of the interia matrices

In this section we describe a method to calculate the inertia matrices $\mathbf{M}_{\dot{\theta}}$ and $\mathbf{H}_{\dot{\theta}}$, which appears in 2.27.

Figure shows a part of the robot links containing the joint $j-1$ and joint j. We assume that the robot can rotate its joints with specified speed inder the proper control law like PD feedback. The joint j's rotation with speed $\dot{\theta}_j$ yields additional linear and angular momentum. of

$$\mathbf{P}_j = \mathbf{w} \times (\tilde{\mathbf{c}}_j - \mathbf{r}_j)\tilde{m}_j \tag{2.40}$$

$$\mathbf{L}_j = \tilde{\mathbf{c}}_j \times \mathbf{P}_j + \tilde{\mathbf{I}}\mathbf{w}_j \tag{2.41}$$

$$\mathbf{w}_j = \mathbf{a}\dot{\theta}_j \tag{2.42}$$

where $\mathbf{r}_j$ and $\mathbf{a}_j$ are the position vector and the rotation axis vector of the joint j, $\tilde{m}_j$, $\tilde{c}_j$ and $\tilde{\mathbf{I}}_j$ mean mass, center of mass and inertia tensor of all link structure driven by the joint j.

Now, the columns of the inertia matrices corrspondig to the joint j are determined by the following equations:

$$\mathbf{m}_j \equiv \mathbf{P}/\dot{\theta}_j, \tag{2.43}$$

$$\mathbf{h}_j \equiv \mathbf{L}/\dot{\theta}_j, \tag{2.44}$$

where $\mathbf{m}_j$ and $\mathbf{h}_j$ are 3×1 vectors. By comparing 2.40 and 2.43, we obtain $\mathbf{m}_j$. Likewise, by comparing 2.42 and 2.44 we obtain $\mathbf{h}_j$ by:

$$\mathbf{m}_j = \mathbf{a}_j \times (\tilde{\mathbf{c}}_j - \mathbf{r}_j)\tilde{m}_j \tag{2.45}$$

$$\mathbf{h}_j = \tilde{\mathbf{c}}_j \times (\tilde{\mathbf{m}}_j) + \tilde{\mathbf{I}}_j a_j \tag{2.46}$$

Using these column vectors, the inertia matrices $\mathbf{M}_{\dot{\theta}}$ and $\mathbf{H}_{\dot{\theta}}$ are constructed like :

$$\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots \mathbf{m}_n] \tag{2.47}$$

$$\mathbf{H}_{\dot{\theta}} =^0 \mathbf{H}_{\dot{\theta}} - \hat{\tilde{\mathbf{c}}}\mathbf{M}_{\dot{\theta}} \tag{2.48}$$

$$^0\mathbf{H}_{\dot{\theta}} = [\mathbf{h}_1, \mathbf{h}_2, \ldots \mathbf{h}_n] \tag{2.49}$$

Equation 2.48 converts the angular momentum from the ground frame into the angular momentum in the robot's CoM frame.

We can calculate $\tilde{m}_j$,$\tilde{\mathbf{c}}_j$ and $\tilde{\mathbf{I}}_j$ from an extremity to the body side by using a recurisve algorithm. Assuming we have already calculated $\tilde{m}_j, \tilde{\mathbf{c}}_j$ and $\tilde{\mathbf{I}}_j$ about joint j, the parameters of adjacent point j-1 can be calculated as:

$$\begin{aligned}
&\tilde{m}_{j-1} = \tilde{m}_j + \tilde{m}_{j-1} \\
&\tilde{\mathbf{c}}_{j-1} = (\tilde{m}_j\tilde{\mathbf{c}}_j + m_{j-1}\mathbf{c}_{j-1})/(\tilde{m}_j + m_{j-1}) \\
&\tilde{\mathbf{I}}_{j-1} = \tilde{\mathbf{I}}_j + \tilde{m}_j D(\tilde{\mathbf{c}}_j - \tilde{\mathbf{c}}_{j-1}) + \mathbf{R}_{j-1}\mathbf{I}_{j-1}\mathbf{R}^T_{j-1} + m_{j-1}D(\mathbf{c}_{j-1} - \tilde{\mathbf{c}}_{j-1}) \\
&\tilde{D}(r) \equiv \hat{\mathbf{r}}\tilde{\mathbf{r}}
\end{aligned} \tag{2.50}$$

where $\mathbf{R}_{j-1}$, m_{j-1} and $\mathbf{I}_{j-1}$ are 3×3 orientation matrix, mass and inertia tensor around the center of the $j-1$th link respectively.

2.6.4 Walking using the Resolved Momentum Control

In this section we evaluate the motion generated by the Resolved Momentum Control using the HRP-2 humanoid robot to realize a walking motion. The target momentum control is given such that the robot's CoM follows given references by

$$P^{ref}_{x,y} = \tilde{m}K_p(\tilde{c}^{ref}_{x,y} - \tilde{c}_{x,y}) + \dot{\tilde{c}}^{ref}_{x,y} \tag{2.51}$$

$$P^{ref}_{z} = \tilde{m}K_p(z^{ref}_{B} - z_B) \tag{2.52}$$

$$\mathbf{L}^{ref} = \mathbf{0}_{3\times 1} \tag{2.53}$$

where K_p is a feedback gain which compensates the error caused by time derivative of Jacobian, $\tilde{\mathbf{c}}^{ref}$ is the target position of CoM, $\dot{\tilde{\mathbf{c}}}^{\mathbf{ref}}$ is the target speed of CoM and z_B is the target height of the pelvis. As specified by [] we made special treatment for the z element of the linear momentum. This is because the constant pelvis height is desirable for most of the case.

The references for the right and left foot $\xi^{ref}_{F_1}$ and $\xi^{ref}_{F_2}$ are computed according to the method described previously.

2.7 To change the library

2.7.1 Introduction

The overall architecture of the library relies on the ZMP preview control scheme as explained previously. The main objects are depicted in Fig. 2.12 and are :

- **ZMPDiscretization**: Transform the stack of foot position in ZMP and position trajectories to be used every 5 ms.
- **ZMPPreviewControlWithMultiBodyZMP**: Realize the control of the ZMP trajectory using Preview Control, and an estimation of the robot's multibody ZMP. The object is made of two main methods **FirstStageOfControl** and **SecondStageOfControl**. The sub-objects used are:
  - **PreviewControl**: Object which implements the preview control of the cart-table model.
  - **InverseKinematics**: Object which implements the simplified inverse kinematics for the legs and the arms.
  - **DynamicMultibody**: Computes the dynamic of the robot to get the multibody's ZMP.

2.7.2 ZMPDiscretization

This object transforms a stack of support foot position into a stack of ZMP, orientation and foot position every 5 ms. The position in X and Y follow third orders polynomial for the foot and the ZMP position.

This object is currently assume to produce a full motion from the beginning where the robot is at the half-sitting position up to the end where the robot is also at the half-sitting position. The method in charge of this is **GetZMPDiscretization**. This function should be changed in order to have a dynamical planning of the foot. The motion is splitted into three parts:

1. The beginning of the motion which adds the time of the preview control window to slowly prepare the robot to move towards the first support foot.
2. For each step there is a decomposition of three phases:
    (a) a half-double phase for the non-support foot take-off,
    (b) a flying phase,
    (c) a landing phase.
3. A final phase where the ZMP is the middle of the two last position of the support foot.

The polynomials used for the foot generation can be changed, but happen to be a problem for stability. Indeed the acceleration is set to be zero at the beginning and at the end of the trajectory with a fifth order polynomial. For the foot this prevent compensation for the momentum around the Z-axis.

TODO: They are unuseful code inside this part which should be removed. The hand behavior should be put here. The code should more versatile for dynamic planning.

2.7.3 Preview Control

This object implements the preview control described previously. It is assumed that the control parameters and the size of the window would be provided through a parameters file. The current format is as follows:

```
ZMP_Height
Sampling_Period
Preview_Control_Time
Kx_0 Kx_1 Kx_2
Ks
F_0 F_1 ... F_SizeOfPreviewWindow
```

where $SizeOfPreviewWindow = Preview\_Control\_Time \times Sampling\_Period$.

TODO: VNL offers the capabilities to solve the Ricatti equations associated with the linear system. The Schur decomposition could be used for that.

2.7.4 Dynamic Multi Body

This object reads a VRML file following the format of OpenHRP2 and creates a tree of bodies. Using the **ForwardVelocity** method it is possible to simulate the dynamic of the robot by specifying the position, orientation and linear velocity of a root body. This root body is set by default to the waist when reading the object at start. It can be modified using **SpecifyTheRootLabel**. To make it work, the joint position and their speed must be also provided.

The computation of ZMP is done by keeping track of the linear and angular momentum through **GetPandL**. You have to perform a simple difference for both of them between two iterations and call **CalculateZMP** which takes them as an input. DynamicMultiBody has been modified to compute the matrices needed for the resolved momentum control. See TestFootPrint2.cpp in src for a practical example.

TODO: Add the angular velocity for the **ForwardVelocity** method. Remove the french naming.

2.7.5 ZMPPreviewControlWithZMPMultiBody

This object implements the queues and the synchronization between the different object for controlling one iteration of the overall scheme. Once the ZMP stack has been build, only one position for each foot and for the ZMP must be provided. The delays between the queues have to be carefully chosen after the explanation provided previously. The corresponding joint values are then computed and are provided as well as the CoM position and orientation. This object should not be modified unless the preview control window is itself modified.

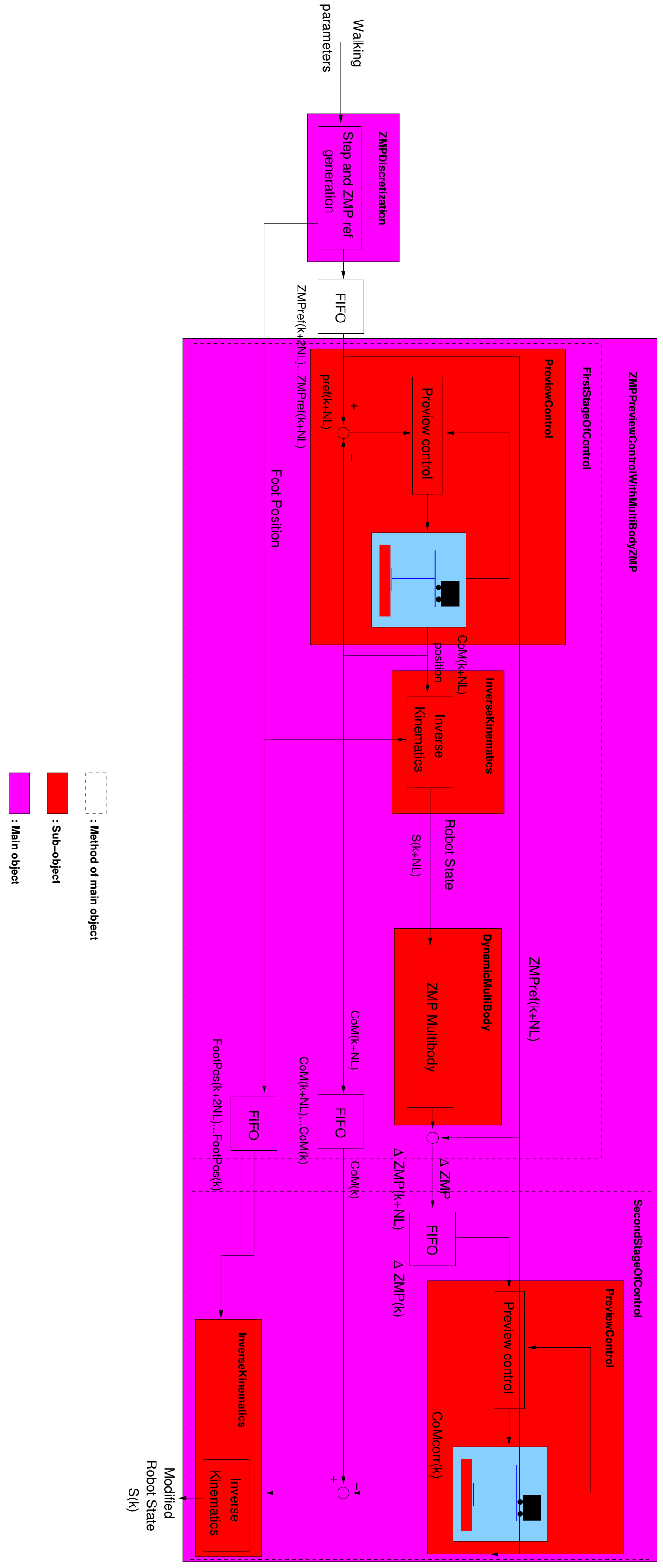

Walking parameters
ZMPDiscretization
Step and ZMP ref generation
FIFO
ZMPref(k+2NL)...ZMPref(k+NL)
ZMPPreviewControlWithMultiBodyZMP
FirstStageOfControl
PreviewControl
Preview control
pref(k+NL)
+
−
Foot Position
CoM(k+NL)
position
InverseKinematics
Inverse Kinematics
Robot State
S(k+NL)
DynamicMultiBody
ZMP Multibody
ZMPref(k+NL)
CoM(k+NL)
FIFO
CoM(k+NL)...CoM(k)
FIFO
FootPos(k+2NL)...FootPos(k)
Δ ZMP
Δ ZMP(k+NL)
FIFO
Δ ZMP(k)
CoM(k)
SecondStageOfControl
PreviewControl
Preview control
CoMcorr(k)
+
+
InverseKinematics
Inverse Kinematics
Modified Robot State S(k)
: Method of main object
: Sub-object
: Main object

Figure 2.12: Overall structure of the library based on the control scheme.

Chapter 3

The Upper body Motion

3.1 Walking Mode inside the Pattern Generator V.2

Walk Mode	Meaning
0	Normal walking
1	Modification of the hip height
2	Stepping over
3	Planification of the upper body motion with way-points
4	Let the other plugin modify the upper body

Table 3.1: Handling of the upper body motion according to the walking mode

Bibliography

[1] Tohru Katayama, Takahira Ohki, Toshio Inoue, and Tomoyuki Kato, "Design of an optimal controller for a discrete-time system subject to previewable demand," *International Journal of Control*, vol. 41, no. 3, pp. 677–699, 1985.

[2] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *International Conference on Robotics And Automation, Taipei Taiwan*, September 2003, pp. 1620–1626.

[3] Shuuji Kajita, *Humanoid Robot*, Omsha, 2005.

[4] William F. Arnold III and Alan Laub, "Generalized eigenproblem algorithms and software for algebraic riccati equations," *Proceedings of the IEEE*, vol. 72, no. 12, December 1984.

[5] Alan J. Laub, "A schur method for solving algebraic riccati equations," *IEEE Transactions on Automatic Control*, vol. AC-24, no. 6, pp. 913–921, December 1979.

[6] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa, "Resolved momentum control: Humanoid motion planning based on the linear and angular momentum," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Las Vegas, Nevada*. IEEE, 2003, pp. 1644–1650.