BIPED WALKING CONTROL DESIGN BASED ON ZERO MOMENT POINT DYNAMICS

BY

YINAI FAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Professor Seth Hutchinson

ABSTRACT

Biped walking control for humanoid robots has been a challenging yet promising research topic in the past decades. The complicated nonlinear dynamics and numerous degrees of freedom are the main obstacles for biped walking control. This thesis studies the complete control scheme of biped walking control based on Zero Moment Point dynamics. Starting with basic definitions, a new trajectory planning method using boundedness solutions of Linear Inverted Pendulum Model is studied and implemented as a replanning algorithm. Stabilizers for walking trajectory tracking are investigated and evaluated, with an emphasis on the Whole Body Cooperation method, which is then implemented along with the replanning algorithm in simulations.

ACKNOWLEDGMENTS

I would like to thank my parents and friends, for their love and support in my research and interest. I won't have this much courage and optimism without their support. I would also like to thank my advisor, Professor Seth Hutchinson, for his enlightening and patient guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Conventional robot manipulators have become common in industries, medical surgeries, and our daily lives, because of their simple structures and easy control algorithms. In the past few decades, researchers have started to explore humanoid robots, which have a stronger ability to perform more versatile and demanding tasks and adapt to human environments. Among the topics of humanoid robots, biped walking is one of the most attractive topics. Biped robots can perform tasks in human environments such as carrying heavy loads in uneven and winding terrain, and entering high-risk areas. The biggest advantage compared with wheeled robots is flexibility. They can walk like a human to avoid obstacles and step on steep and uneven terrain by flexibly planning the motions of two legs, while it's not physically possible for wheeled robots. However, everything has its other side. The challenge of biped walking is its complex, nonlinear dynamics. Numerous degrees of freedom and lacking fixed base link leads to an extremely complex system; therefore how to keep this complex system stable while walking like a human is the most challenging part. Starting with very early work like [1] stating basic theories of biped walking, researchers have developed lots of outstanding control methods for biped walking ([2] to [3] for older works and [4],[5],[6] for the most recent developments).

1.2 Background

1.2.1 Zero Moment Point

The Zero Moment Point (ZMP) is the point on the ground where the total moment generated by the ground reaction forces is zero. The details of derivation and usage for ZMP will be introduced in 2.1.1. There are two kinds of stabilities for biped walking [7]: static stability and dynamic stability. Static stability is trivial since it requires the projection of the center of mass to be restricted in the convex hull enclosed by contact points at any time, which is not how humans walk. So researchers have been focusing on achieving dynamic stability for biped robots. In [7], the condition for dynamic stability of a linear inverted pendulum model (LIPM) is discussed. A new quantity "Extrapolated center of mass position" (XcoM) is used to measure the stability of the model. Different stability situations are listed based on the relation between XcoM, center of mass (CoM), center of pressure (CoP) and base of support (BoS). Most of the research have been focus on zero moment point tracking based biped walking control. The concept of ZMP was first introduced in [1], in which ZMP is defined and used as a stability criterion for biped robot. It's proved that the ZMP should always be restricted in the support polygon during walking to maintain stability.

The robot's dynamics is usually complicated, due to numerous degrees of freedom and nonlinear properties, so instead of measuring all the parameters of a biped robot and calculate the exact dynamics, a simple linear inverted pendulum model (LIPM) is used to generate walking pattern. Then a stabilizer is used to compensate the tracking error. The LIPM will be discussed in detail in 2.2.1. In [8], the linear inverted pendulum mode is proposed. By assuming the CoM of robot moves in a particular straight line, which is controlled by a constraint controller (PD control is used in the simulation of [8]), the model is completely linear in all state space and easy to plan and control. In the same work, a walking pattern is generated by defining the constraint line and using conservation of orbital energy. In addition, ankle torque is used as an input to compensate the disturbance during walking. This work was extended to 3D situation in [9], where the motion of CoM is constrained in an arbitrarily defined plane by introducing two virtual input to compensate the nonlinearity.

Based on LIPM, there are several ways to generate a desired CoM trajectory such that it has the desired ZMP trajectory. One of the most outstanding methods is [2]. A preview controller is used to obtain CoM trajectory from reference ZMP trajectory. The biped robot is simplified as a cart-table model, which is the inverse of LIPM. The cart-table model uses the third derivative of CoM as input and actual ZMP as output. A preview controller [10] which consists of state feedback term, ZMP error term and preview reference ZMP term is applied to the system to track the reference ZMP, therefore the corresponding CoM can be obtained as the state of the system. Also when the actual biped robot is tracking the reference ZMP, another preview controller uses future ZMP error to compensate tracking error. By taking advantage of future information, the robot is able to act before each step. However, this kind of pattern generation is done off-line and is not able to deal with modification of foot placement. A real-time gait generation based on ZMP is proposed in [11]. Generally, given a reference ZMP trajectory and two boundaries of CoM (initial and final conditions), CoM can be solved uniquely, which is used in off-line pattern generation. In order to generate trajectory in real time, the generated trajectory has to be connected smoothly with the robot's actual current initial condition. The proposed method generates a new ZMP reference to match initial conditions of CoM to ensure smooth connection. Based on [11], [12] improves the on-line gait generation method by adjusting the period of single support phase and using preview control to compensate the ZMP fluctuation. While [11] cannot handle immediate modification of foot placement (new ZMP sometimes won't stay in support polygon), this new method can ensure the stability of immediate modification of foot placement. In [13], vertical CoM motion is captured from human data and used for improving preplanned CoM trajectory.

One advantage of ZMP is that dynamic biped walking control can be decomposed into two parts, a walking pattern generation and a stabilization around it [3]. The main contribution of [3] is the CoM/ZMP tracking controller, also known as stabilizer. A clear hierarchical control structure is given in this work, which consists of three layers: a CoM/ZMP control layer ensures ZMP tracking based on LIPM; a posture force control layer calculates reference joint angles; finally a servo control layer controls the joints to track the given reference joint angles.

1.2.2 Capture Point

In recent years, a new perspective of solving stability problem for biped walking was studied, which is the application of capture point. The capture point is defined as the position that the robot have to step such that the robot will come to a complete stop [14]. To simplify the biped model, a linear inverted pendulum plus flywheel model is used. The solution of capture point for a given state is calculated using conservation of orbital energy of LIP, also the capture region is calculated if the flywheel torque is considered. If the set of the intersection of support polygon and capture region is not empty, the robot will be able to stabilize. To apply capture point in biped walking, N-step capture point was calculated in [15] for three simplified models: point foot LIP, finite-size foot LIP and finite-size foot LIP with a reaction mass. In [15], a new possibility is provided to solve biped walking problem, which is to plan walking pattern based on N-step capture point.

In [16], the LIPM is decoupled into a stable subsystem and an unstable subsystem by changing the coordinates, where the unstable state coincides with the definition of instantaneous capture point. A constraint is found to link the desired ZMP trajectory to CoM initial conditions, in order to have a bounded CoM trajectory. Based on this result, a new way to generate CoM trajectory from desired ZMP trajectory is proposed in [5]. One can adjust the parameterized ZMP trajectory such that the actual CoM initial conditions will satisfy the constraint calculated from ZMP. The CoM trajectory generated is anticipatory since the constraint takes account the future values of desired ZMP trajectory. The walking pattern generation in this thesis will mainly focus on this method.

1.3 Thesis Goals

This thesis aims to study the current state of art of biped walking control and then propose a new control method of biped walking by improving some of the existing methods. Since the feedback control scheme includes different layers, and there are different problems need to be solved in each layer, the specific goals for each layer or stage are listed:

1. The pattern generation method needs to be studied and proposed. More specifically, a CoM/ZMP trajectory that satisfies the ZMP stability criterion need to be generated. This thesis solves this problem by using the boundedness condition of LIPM proposed in [5]. A new way for CoM/ZMP replanning based on boundedness condition is proposed.

2. The controller for CoM/ZMP tracking needs to be studied and compared. There are numerous of existing controller for CoM/ZMP tracking in the past two decades. This thesis evaluates several of the most representative methods and chooses the whole body cooperation controller for deeper investigation.

3. To show the efficiency of proposed CoM/ZMP replanning algorithm and the tracking controller, we have to implement them in simulation. So the implementation and simulation details are designed in this thesis.

1.4 Thesis Outline

Based on the thesis goals, the structure of following chapters are:

- Chapter 2
  Definition of ZMP and LIPM are introduced. The boundedness condition for CoM/ZMP generation is derived.

- Chapter 3
  Several mostly used stabilizer controller designs and their derivations are studied, with an emphasis on the whole body cooperation controller.

- Chapter 4
  Different stabilizers are implemented in simulations in Gazebo. Also,

walking control simulations are done using generated trajectory of Chapter 2 and stabilizer in Chapter 3.

- Chapter 5
  Different stabilizers are evaluated, based on the simulation results and difficulties in implementation.

CHAPTER 2

WALKING PATTERN GENERATION

Walking pattern generation is a procedure to generate the reference CoM and ZMP trajectories. To generate the trajectories, we have to start with the gait design, for example, the step length and time of each step, double and single support time, etc. From gait information, we can generate a ZMP to fit the gait. Then from this desired ZMP trajectory, we can use the method in [5] to get a CoM trajectory which will result in our desired ZMP trajectory.

2.1 ZMP generation

In this section, we will show that the ZMP has to always stay in the interior of the support polygon. With this constraint, one can easily design the ZMP trajectory once the support polygon profile is given. Here, ZMP trajectory is designed using splines of polynomials that stays inside the support polygon at any time instant.

2.1.1 Definition of Zero Moment Point

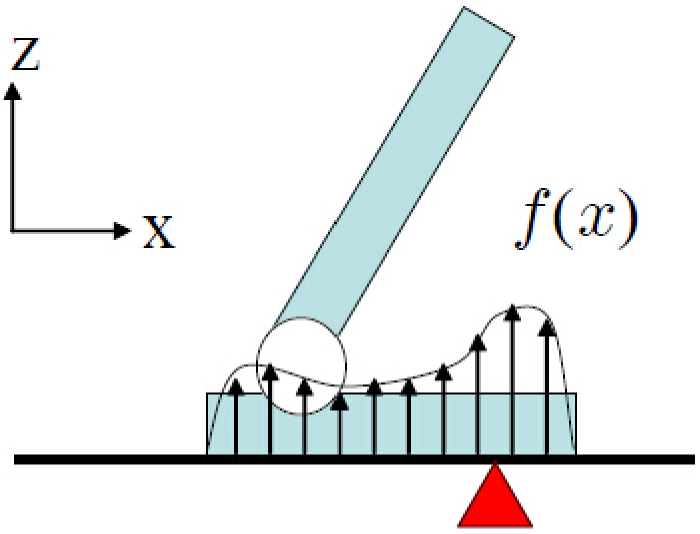


Figure 2.1: ZMP and distributed ground reaction force [17]

The Zero Moment Point (ZMP) is defined as following in [1]: The distributed floor reaction force can be replaced by a single force R acts on Zero-moment Point. In another word, it specifies the point on the ground such that the contact force between the foot and the ground does not produce any moment around any axis passing this point on the ground plane. Therefore, for a finite size foot in 3D locomotion, the ZMP coincide with the center of pressure (CoP) point, and can be found by measuring the distributed ground reaction force f_x and f_y:

$$x_{zmp} = \frac{\int x f_x(x)dx}{\int f_x(x)dx} \qquad (2.1)$$
$$y_{zmp} = \frac{\int y f_y(y)dy}{\int f_y(y)dy}$$

For a biped robot, the support polygon is the region formed by enclosing all the contact points between the robot and the ground, mathematically the smallest convex hull including all contact points as shown in Figure 2.2.

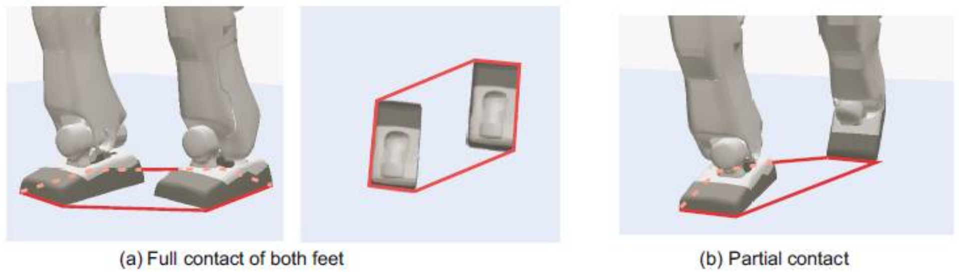


Figure 2.2: Support polygon [18]

One useful property of ZMP is that it always stays in the support polygon of the robot. To prove this, let's assume the ZMP is outside the support polygon, for example, it's ahead of the toes. Since the ground cannot provide suction reaction force, the reaction force at any contact points is pointing up, so the overall torque generated by the reaction force around the ZMP will be a non-zero value (can be positive or negative depending on how we define the positive rotation direction), which contradicts with the definition of ZMP.

Instead of calculating ZMP by measuring the reaction forces at the foot, another way to calculated ZMP is to solve the dynamics equations of the robot.

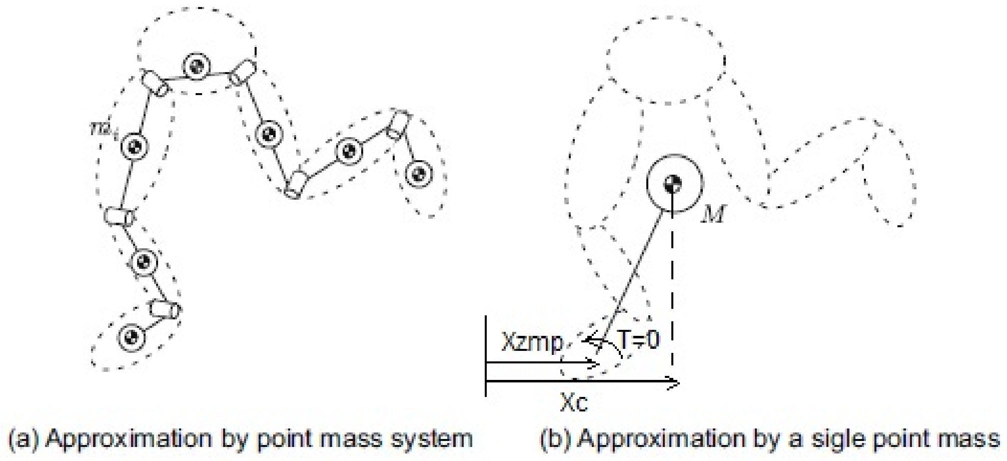

m_i
M
Xzmp
T=0
Xc
(a) Approximation by point mass system
(b) Approximation by a sigle point mass

Figure 2.3: CoM of a biped robot [18]

As shown in Figure 2.3, for a complex biped robot, we can find the overall equivalent center of mass. Then the robot can be modeled as an inverted pendulum with CoM at the top, the ZMP at the base, and with a variable length. In x direction, from the fact that the total moment around the CoM is the sum of torques produced by the ground reaction forces, we can calculate the total moment as following (the same method can be used for y direction).

$$T = mg(x_c - x_{zmp}) + m\ddot{z}_c(x_c - x_{zmp}) - m\ddot{x}_c z_c \tag{2.2}$$

where m is the mass, g is the gravitational acceleration, x_c and z_c are the horizontal and vertical coordinates of CoM, x_{zmp} is the ZMP in x direction, T is the torque generated by ground reaction force about the ZMP. The term $-m\ddot{x}_c z_c$ is the torque generated by the horizontal reaction force, since this force also produces horizontal acceleration $\ddot{x}_c$. The term $mg(x_c - x_{zmp}) + m\ddot{z}_c(x_c - x_{zmp})$ is the torque generated by the vertical reaction force, since this force supports the gravity and produces vertical acceleration $\ddot{z}_c$.

By definition, for ZMP, $T = 0$, so we have:

$$x_{zmp} = x_c - \frac{z_c}{g + \ddot{z}_c}\ddot{x}_c \tag{2.3}$$

The LIPM is an inverted pendulum with a constant CoM height h_0, which means the vertical acceleration $\ddot{z}_c$ is 0. So (2.3) becomes:

$$x_{zmp} = x_c - \frac{h_0}{g}\ddot{x}_c \tag{2.4}$$

ZMP in y direction can be calculated using the same method.

2.1.2 ZMP Trajectory Generation

In the previous section, it is proved that the ZMP always stays in the support polygon of the robot. This fact will be used as the stability criterion of biped walking. It should be noted that the stability here is the ability of walking without tipping over or falling. It's different from the one in control theory.

When the robot tends to tip over, the actual ZMP is approaching the edge of support polygon and when it is tipping over, the ZMP will stay at the edge of support polygon, since the only set of contact points between foot and ground is the edge. So in order to make sure the robot will not tip over, the ZMP should be strictly inside the support polygon at all time.

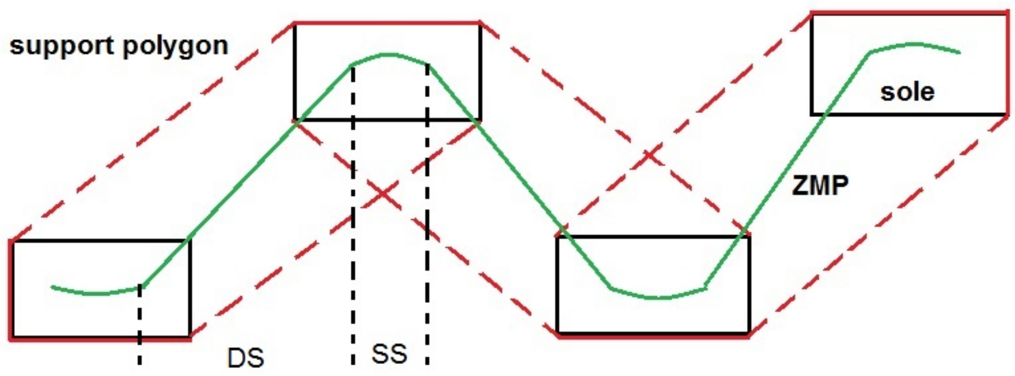


Figure 2.4: ZMP generation from desired gaits

As shown in Figure 2.4, once the gaits are specified, that is, the step length and step period for each step, the foot size are specified, we are able to locate the soles of each step, therefore draw the path of the support polygon, which is shown in red lines in Figure 2.4. Then a ZMP trajectory can be designed such that it always stays inside the support polygon, as shown in green line. In double support phase, the ZMP move from heel to the toe (or closed to toe in the real case), then in single support phase, it moves from the toe of the foot to the heel of the other foot, while maintains in the polygon enclosed by the two feet.

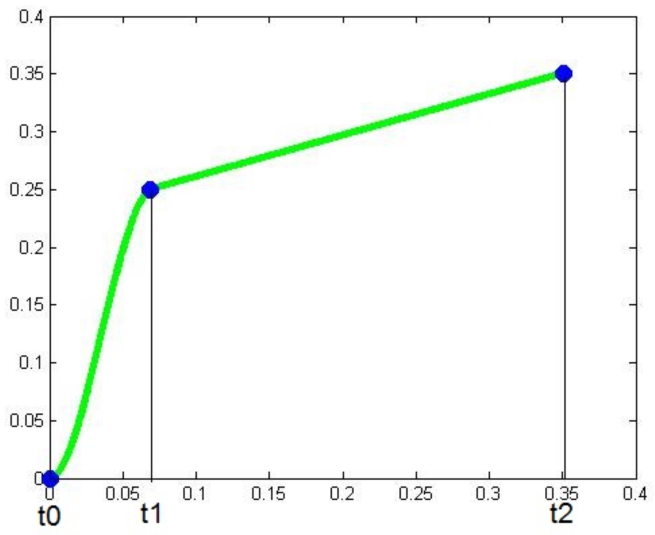


Figure 2.5: Cubic/Linear ZMP

In order to have a smooth mathematical expression for the trajectory, we use spline to fit the ZMP trajectory as in [16], [5], [11], [12]. As shown in

Figure 2.5. Starting with double support, every step consists of 1 double support phase and 1 single support phase. The values of ZMP at time t_0,t_1 and t_2 are known by predefined gait specifications. For double support, we use a cubic spline which corresponds the shift of ZMP between feet. For single support, slightly different from [5], we use a linear spline to express the shift of ZMP from heel to toe in one foot.

For an arbitrary step:

$$\begin{aligned} x_{cubic}(t) &= a_0 + a_1(t-t_0) + a_2(t-t_0)^2 + a_3(t-t_0)^3, \quad t \in [t_0, t_1) \quad (2.5) \\ x_{linear}(t) &= b_0 + b_1(t-t_1), \quad t \in [t_1, t_2) \end{aligned}$$

where $[t_0, t_1)$ is the double support phase and $[t_1, t_2)$ is the single support phase. We need to solve the 4 unknown parameters a_0,a_1,a_2,a_3 of the cubic equation and b_0,b_1 of the linear equation by solving the following boundary conditions,

$$\begin{aligned} x_{cubic}(t_0) &= x_{zmp}(t_0) \qquad (2.6) \\ \dot{x}_{cubic}(t_0) &= \dot{x}_{zmp}(t_0) \\ x_{cubic}(t_1) &= x_{zmp}(t_1) \\ x_{linear}(t_1) &= x_{zmp}(t_1) \\ x_{linear}(t_2) &= x_{zmp}(t_2) \\ \dot{x}_{cubic}(t_1) &= \dot{x}_{linear}(t_1) \end{aligned}$$

where $x_{zmp}(t_0)$, $x_{zmp}(t_1)$, $x_{zmp}(t_2)$ are given waypoints.

By solving these equations, the parameters can be found. Since the expression is long and tedious, we won't show here but will directly implement in the code in next section.

2.2 CoM Generation

Once the ZMP trajectory is designed, the CoM trajectory of the robot is still not fixed. If we use LIPM as the simplification of the robot, from (2.4), the solution of CoM is not unique given a fixed ZMP trajectory because the boundary conditions for x_c and $\ddot{x}_c$ are free to choose. In this section, the LIPM is introduced. Then a new method based on boundedness solution of

CoM as in [16] and [5] is studied and used for CoM generation.

2.2.1 Linear Inverted Pendulum Model

To calculate the CoM that will give a desired ZMP, we have to go through the dynamics of the biped robot. Due to the complexity of a real biped robot (too many links and joints), a linear inverted pendulum model is often used as a good approximation of the biped robot. Then when implementing the walking pattern generated using LIPM, a closed-loop control scheme has to be used as well to stabilize the real biped robot during locomotion.

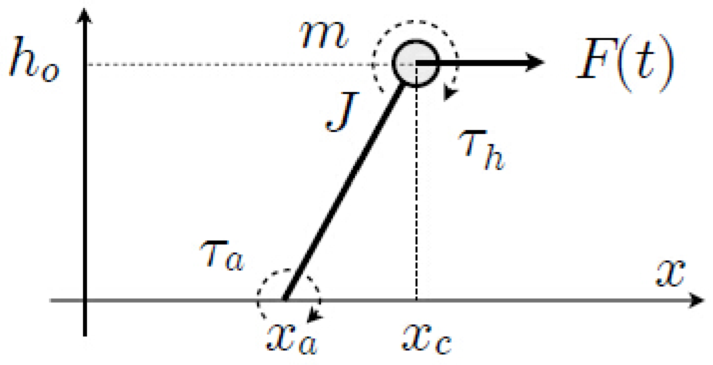


Figure 2.6: LIPM diagram [16]

The 3D LIPM for biped walking was first proposed and studied by Kajita in [9]. In 2D case, as shown in Figure 2.6, the biped robot is modeled as a linear inverted pendulum (LIP) which has a mass m concentrated at the top. A point foot is at the bottom end of the pendulum, which contacts with the ground at x_a. x_c is the position of the mass. h_0 is its constant height. τ_a is the ankle torque. τ_h is the hip torque and $F(t)$ is a time varying disturbance force.

The LIPM dynamics [2],[16] in sagittal plane is

$$\ddot{x}_c(t) = \frac{g}{h_0}(x_c - x_a) + \frac{1}{mh_0}(\tau_a - \tau_h) + \frac{F(t)}{m} \tag{2.7}$$

Since real biped robot has a finite-sized foot and we assume there's no external disturbance and no hip torque, will only have x_a and τ_a as input.

Introducing the center of pressure(CoP) position

$$x_{cop} = x_a - \frac{\tau_a}{mg} \tag{2.8}$$

which coincide with ZMP position $x_{zmp} = x_{cop}$, substitute back to (2.7) and defining $\omega_0 = \sqrt{g/h_0}$. we get the following model:

$$\ddot{x}_c(t) = \omega_0^2 x_c - \omega_0^2 x_{zmp} \tag{2.9}$$

The state space representation is

$$\begin{pmatrix} \dot{x}_c \\ \ddot{x}_c \end{pmatrix} = A_c \begin{pmatrix} x_c \\ \dot{x}_c \end{pmatrix} + B_c x_{zmp} \tag{2.10}$$

where

$$A_c = \begin{pmatrix} 0 & 1 \\ \omega_0^2 & 0 \end{pmatrix}, \quad B_c = \begin{pmatrix} 0 \\ -\omega_0^2 \end{pmatrix} \tag{2.11}$$

2.2.2 Bounded Solutions For The LIPM

According to the LIPM dynamics (2.9), if we use x_{zmp} as input and x_c as state, then the solution is determined by the specific trajectory of x_{zmp} and boundary conditions of x_c. A bounded CoM solution not guaranteed if we assign arbitrary boundary conditions to x_c for a given x_{zmp} trajectory. In this part, we'll discuss boundedness issues of x_c trajectory.

We can decouple the system by changing the coordinate

$$\begin{pmatrix} x_u \\ x_s \end{pmatrix} = \begin{pmatrix} 1 & 1/\omega_0 \\ 1 & -1/\omega_0 \end{pmatrix} \begin{pmatrix} x_c \\ \dot{x}_c \end{pmatrix} \tag{2.12}$$

where x_u is the unstable state and x_s is the stable state. The unstable state coincides with the definition of capture point, which makes sense since the capture point is an unstable property. Similarly, we can express x_c and $\dot{x}_c$ in terms of the new states

$$\begin{aligned} x_c &= \frac{1}{2}(x_u + x_s) \\ \dot{x}_c &= \frac{\omega_0}{2}(x_u - x_s) \end{aligned} \tag{2.13}$$

The decoupled system becomes

$$\begin{aligned} \dot{x}_u &= \omega_0 x_u - \omega_0 x_{zmp} \\ \dot{x}_s &= -\omega_0 x_s + \omega_0 x_{zmp} \end{aligned} \tag{2.14}$$

Generally, the unstable eigenvalue ω_0 will lead to divergent behavior. For a given ZMP trajectory as input, we can find an initial value for x_u analytically such that the resulting x_u trajectory is bounded. In another point of view, if we have a known initial condition for x_u, we can find a ZMP trajectory input which stabilizes x_u. Either way needs us to find a constraint that relates the initial condition and the ZMP.

For a linear first order system given the input x_{zmp}, the general solution is

$$x_u(t, x_{zmp}) = e^{\omega_0(t-t_0)} x_u(t_0) - \omega_0 \int_{t_0}^{t} e^{\omega_0(t-\tau)} x_{zmp}(\tau) d\tau \tag{2.15}$$

If we choose an initial condition as derived in [16]

$$x_u(t_0) = x_u^\star(t_0, x_{zmp}) = \omega_0 \int_{t_0}^{\infty} e^{-\omega_0(\tau-t_0)} x_{zmp}(\tau) d\tau \tag{2.16}$$

we obtain the particular solution and choose $t_0 = 0$

$$x_u^\star(t, x_{zmp}) = \omega_0 \int_{0}^{\infty} e^{-\omega_0 \tau} x_{zmp}(\tau + t) d\tau \tag{2.17}$$

The notation $x_u^\star(t, x_{zmp})$ is used to denote the particular solution solved under input x_{zmp} and initial condition (2.16). This particular solution is bounded, given a desired ZMP trajectory. A special property of this particular solution is that it need the future value of ZMP trajectory, which makes the CoM generated noncausal and anticipatory, that is, the robot will able to move its CoM before the step by knowing the future information of ZMP.

The particular initial condition in (2.16) gives a constraint of x_c and $\dot{x_c}$ if we apply (2.12)

$$x_u^\star(t_0, x_{zmp}) = x_u(t_0) = x_c(t_0) + \frac{1}{\omega_0}\dot{x}_c(t_0) \tag{2.18}$$

which is called the boundedness condition in [16] and [5].

Now let's consider the stable subsystem, which has a eigenvalue $-\omega_0$. The general solution becomes

$$x_s(t, x_{zmp}) = e^{-\omega_0(t-t_0)}x_s(t_0) + \omega_0 \int_{t_0}^{t} e^{-\omega_0(t-\tau)}x_{zmp}(\tau)d\tau \tag{2.19}$$

For a particular initial condition

$$x_s^\star(t_0, x_{zmp}) = x_s^\star(0, x_{zmp}) \tag{2.20}$$

we have a particular solution under this initial condition

$$x_s^\star(t, x_{zmp}) = e^{-\omega_0(t)}x_s^\star(0, x_{zmp}) + \omega_0 \int_{0}^{t} e^{-\omega_0(t-\tau)}x_{zmp}(\tau)d\tau \tag{2.21}$$

From (2.13), we have the full bounded trajectory of CoM and velocity

$$\begin{aligned} x_c^\star(t) &= \frac{1}{2}(x_u^\star(t, x_{zmp}) + x_s^\star(t, x_{zmp})) \\ \dot{x}_c^\star(t) &= \frac{\omega_0}{2}(x_u^\star(t, x_{zmp}) - x_s^\star(t, x_{zmp})) \end{aligned} \tag{2.22}$$

It should be noted that the notation $x_u^\star(t, x_{zmp})$ denote the x_u trajectory derived from the boundedness condition, while $x_s^\star(t, x_{zmp})$ is the x_s trajectory derived from its particular initial condition $x_s^\star(0, x_{zmp})$. Actually, for a given desired ZMP trajectory, $x_u^\star(0, x_{zmp})$ is fixed, knowing any one initial condition of x_s, x_c and $\dot{x}_c$ will provide sufficient information to solve for the bounded CoM trajectory.

When the initial state of stable subsystem deviate from $x_s^\star(0, x_{zmp})$, which means the initial condition $x_c(0)$ and $\dot{x}_c(0)$ deviate from $x_c^\star(0)$ and $\dot{x}_c^\star(0)$, we can prove the convergence

$$
\begin{aligned}
x_c &= \frac{1}{2}(x_u^\star + x_s) \qquad (2.23)\\
&= \frac{1}{2}(x_u^\star + e^{-\omega_0 t}x_s(0) + \omega_0 \int_0^t e^{-\omega_0(t-\tau)}x_{zmp}(\tau)d\tau)\\
&= \frac{1}{2}(x_u^\star + e^{-\omega_0 t}(x_s(0) - x_s^\star(0, x_{zmp})) + e^{-\omega_0 t}x_s^\star(0, x_{zmp})\\
&\quad +\omega_0 \int_0^t e^{-\omega_0(t-\tau)}x_{zmp}(\tau)d\tau)\\
&= \frac{1}{2}(x_u^\star + e^{-\omega_0 t}(x_s(0) - x_s^\star(0, x_{zmp})) + x_s^\star(t))\\
&= x_c^\star + \frac{1}{2}e^{-\omega_0 t}(x_s(0) - x_s^\star(0, x_{zmp}))
\end{aligned}
$$

The term $\frac{1}{2}e^{-\omega_0 t}$ converges to 0 so x_c will converge to $x_c^\star$.

2.2.3 CoM Generation Based on Parameterized ZMP

From the conclusion of the previous section, if the boundedness constraint (2.18) is satisfied, then a bounded solution of the unstable state is guaranteed. This lead us to two possible ways to generate CoM trajectory. If the desired ZMP trajectory is given, a pair of initial CoM condition can be chosen to satisfy the constraint, which is not possible in the implementation of a real robot, since the initial CoM condition is arbitrary. The other choice is to re-shape or re-design the ZMP trajectory, such that the constraint is satisfied, for a pair arbitrarily given CoM initial condition.

A method of CoM/ZMP design is proposed in [5], where the step length of each step is unknown and need to be found such that the ZMP will satisfy the constraint condition specified by actual CoM Initial condition. Also a constant/cubic/constant ZMP is used as the reference ZMP. In this method, the step time of each step has to be known. The unknown step lengths can be solved efficiently since they enter the constraint condition linearly. It should be noted that in this section, we only discuss the motion in x direction since the derivations in y direction follow the same procedures.

2.2.3.1 Case I: Unknown step period

For a mobile robot, the moving speed is usually specified. Therefore, we'll set both the step lengths and the step time unknown, but relate them using the specified walking speed,

$$d = vT \tag{2.24}$$

where d is the step length, T is the step period of one step and v is the walking speed.

For a single step, or for a series of steps with same step length and same step period, only one degree of freedom is left to solve if the walking speed is specified, based on (2.24). Solving for one of d and t will directly give the value for the other one, so here we'll use T as the only unknown parameter that needs to be solved such that the ZMP will satisfy the boundedness constraint condition.

If we have n steps, then for the i-th step, we have the following ZMP boundary conditions from (2.6), which gives smooth a connection between i-th and previous step, and between cubic and linear spline:

$$\begin{aligned} x_{cubic}^{(i)}(t_i) &= x_{linear}^{(i-1)}(t_i) \\ \dot{x}_{cubic}^{(i)}(t_i) &= \dot{x}_{linear}^{(i-1)}(t_i) \\ x_{cubic}^{(i)}(t_i + T_d) &= x_{cubic}^{(i)}(t_i) + d - d_s \\ x_{linear}^{(i)}(t_i + T_d) &= x_{cubic}^{(i)}(t_i) + d - d_s \\ x_{linear}^{(i)}(t_i + T) &= x_{cubic}^{(i)}(t_i) + d \\ \dot{x}_{cubic}^{(i)}(t_i + T_d) &= \dot{x}_{linear}^{(i)}(t_i + T_d) \end{aligned} \tag{2.25}$$

where t_i is the starting time of i-th step, d_s is the single support length(distance), T_d is the double support period, as illustrated in Figure 2.7. Both d_s and T_d should be specified by gait specifications.

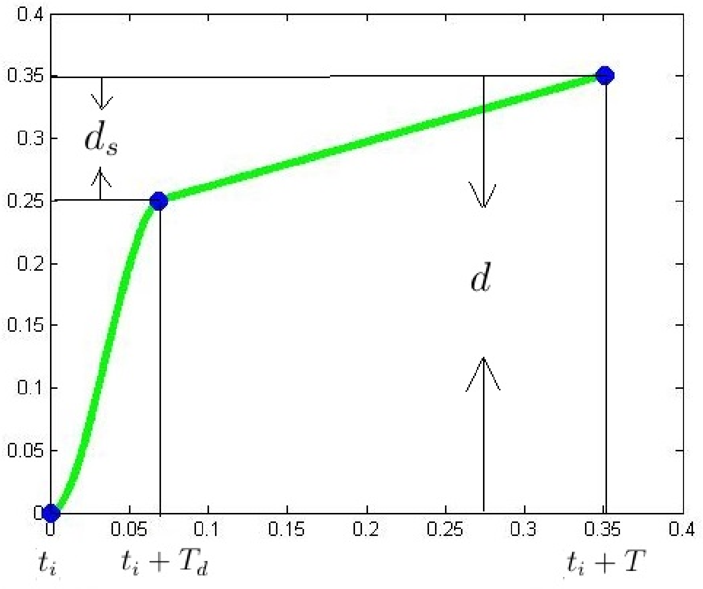


Figure 2.7: Cubic/Linear ZMP

Substituting the boundary condition for i-th step (2.25) into the cubic/-linear spline in (2.5), we have the cubic/linear spline for the i-th step:

$$\begin{aligned} x^{(i)}_{cubic}(t,T) &= a^{(i)}_0 + a^{(i)}_1(t-(i-1)T) + a^{(i)}_2(t-(i-1)T)^2 + a^{(i)}_3(t-(i-1)T)^3 \\ & \quad t \in [(i-1)T, (i-1)T + T_d) \\ x^{(i)}_{linear}(t,T) &= b^{(i)}_0 + b^{(i)}_1(t-(i-1)T + T_d) \\ & \quad t \in [(i-1)T + T_d, nT) \end{aligned} \tag{2.26}$$

which can be written as:

$$\begin{aligned} x^{(i)}_{cubic}(t,T) &= \overline{a}^{(i)}_0(T) + \overline{a}^{(i)}_1(T)t + \overline{a}^{(i)}_2(T)t^2 + \overline{a}^{(i)}_3(T)t^3 \\ & \quad t \in [(i-1)T, (i-1)T + T_d) \\ x^{(i)}_{linear}(t,T) &= \overline{b}^{(i)}_0(T) + \overline{b}^{(i)}_1(T)t \\ & \quad t \in [(i-1)T + T_d, nT) \end{aligned} \tag{2.27}$$

where $\overline{a}^{(i)}_k(T)$ and $\overline{b}^{(i)}_k(T)$ are new coefficients that consist of polynomials of the unknown parameter T, and the splines are now a function of both time t and the unknown parameter T.

For n steps, the full ZMP trajectory will be,

$$\begin{aligned} x_{zmp}(t,T) &= \sum_{i=1}^{n}[x_{cubic}^{(i)}(t,T)(H(t-(i-1)T) \\ &\quad -H(t-(i-1)T-T_d)) \\ &\quad +x_{linear}^{(i)}(t,T)(H(t-(i-1)T-T_d)-H(t-iT))] \\ &\quad +x_{linear}^{(n)}(nT,T)H(t-nT) \end{aligned} \tag{2.28}$$

where x_{zmp} is now a function of both time t and the unknown T, and the function $H(t)$ is a unit step function. The long summation term sums up the cubic and linear parts for the i-th step. The cubic part of i-th step is calculated by multiplying its cubic expression derived from (2.27) with a step function between time $t-(i-1)T$ and $t-(i-1)T-T_d$. The linear term part of i-th step is calculated similarly. The final term $x^{(n)}{}_{linear}(nT,T)H(t-nT)$ makes sure the ZMP stays at its final value for the rest of the time.

Then substitute (2.28) into (2.16), the specific initial condition becomes:

$$\begin{aligned} x_u^{\star}(0,x_{zmp}(t,T)) = \ & \omega_0 \textstyle\sum_{i=1}^{n}(\int_{(i-1)T}^{(i-1)T+T_d} e^{-\omega_0\tau}x_{cubic}^{(i)}d\tau \\ & + \textstyle\int_{(i-1)T+T_d}^{iT} e^{-\omega_0\tau}x_{linear}^{(i)}d\tau) \\ & + \textstyle\int_{nT}^{\infty} e^{-\omega_0\tau}x_{linear}^{(n)}(nT)d\tau \end{aligned} \tag{2.29}$$

where the cubic and linear spline terms x_{cubic} and x_{linear} enter the integral linearly.

As we can see in (2.27), $x_{cubic}^{(i)}$ and $x_{linear}^{(i)}$ can be broken into power terms of t with T appearing in the coefficients of these power terms, $\overline{a}_k^{(i)}(T)$ and $\overline{b}_k^{(i)}(T)$. Therefore, if we substitute (2.27) back to (2.29), the integral terms can be broken into the following basic modulus:

$$\begin{aligned} \omega_0\int e^{-\omega_0\tau}c_1^{(i)}(T)d\tau &= -c_1^{(i)}(T)e^{-\omega_0 t} \\ \omega_0\int e^{-\omega_0\tau}c_2^{(i)}(T)td\tau &= \frac{-c_2^{(i)}(T)e^{-\omega_0 t}(\omega_0 t+1)}{\omega_0} \\ \omega_0\int e^{-\omega_0\tau}c_3^{(i)}(T)t^2d\tau &= \frac{-c_3^{(i)}(T)e^{-\omega_0 t}(\omega_0^2t^2+2\omega_0 t+2)}{\omega_0^2} \\ \omega_0\int e^{-\omega_0\tau}c_4^{(i)}(T)t^3d\tau &= \frac{-c_4^{(i)}(T)e^{-\omega_0 t}(\omega_0^3t^3+3\omega_0^2t^2+6\omega_0 t+6)}{\omega_0^3} \end{aligned} \tag{2.30}$$

where $c_1^{(i)}$ to $c_4^{(i)}$ are coefficients of powers of t for the i-th step and can be any of the $\overline{a}_k^{(i)}(T)$ and $\overline{b}_k^{(i)}(T)$ in (2.27). Combining all equations from (2.24) to (2.30), we'll have the final closed form of the constraint, in terms of T,

$$x_u^\star(0) = x_u^\star(0, x_{zmp}) = x_c(0) + \frac{1}{\omega_0}\dot{x}_c(0) \tag{2.31}$$

where $x_{zmp} = x_{zmp}(t, T)$.

The required initial condition of x_u is a nonlinear function of unknown variable T, since it includes high order polynomials of T. It can be solved using numerical method in Matlab, given arbitrary initial conditions of CoM,$x_c(0)$ and $\dot{x}_c(0)$.

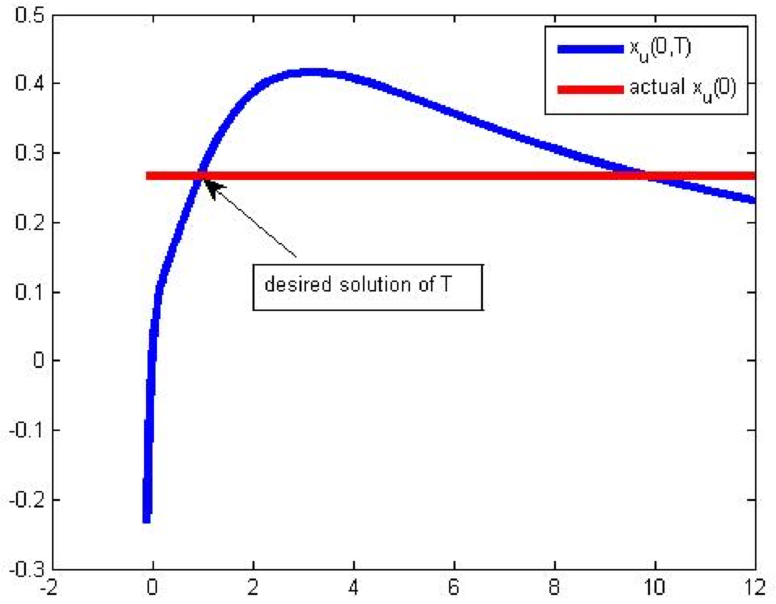


Figure 2.8: Solution of T:4 steps planned, $v = 0.5m/s$, $T_d = 0.25T$, $d_s = 0.1m$, $x_c(0) = 0$,$\dot{x}_c(0) = 1$

Figure 2.8 shows a typical nonlinear function of $x_u^\star(0, x_{zmp}(t, T))$ in terms of T. For a given set of initial conditions for CoM, in another word, for a given value of $x_u^\star(0, x_{zmp}(t, T))$, generally there are two solutions in $[0, \infty]$. We'll only take the one that is feasible for real robot, which is usually less than 2 second.

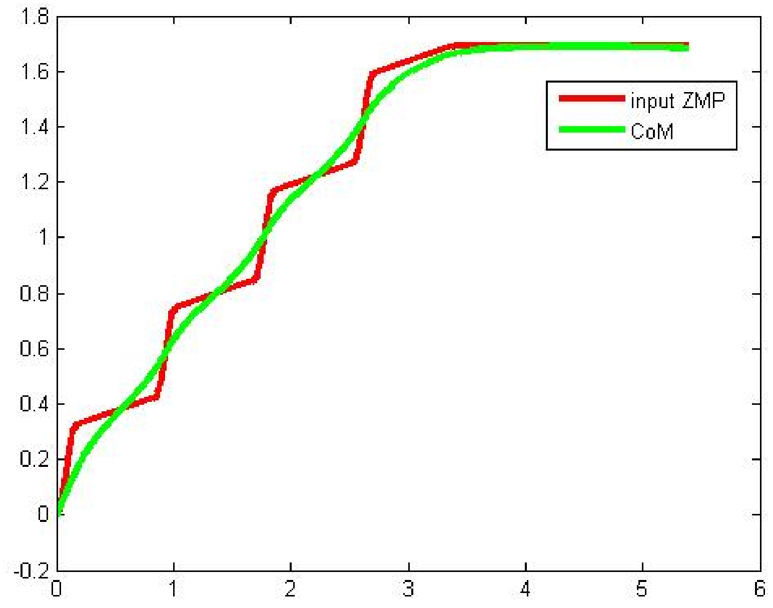


Figure 2.9: CoM generation:4 steps planned, $v = 0.5m/s$, $T_d = 0.2T$, $d_s = 0.1m$, $x_c(0) = 0$,$\dot{x}_c(0) = 1$

As shown in Figure 2.9, CoM trajectory of 4 steps is generated by solving the variable T, to achieve a walking speed of $0.5m/s$. After 4 steps the CoM converges to the final value of ZMP, which shows the convergence of the solution.

This algorithm can be used in a situation where changing of walking speed is needed. When the desired walking speed is changed, based on the actual initial condition of CoM, it's able to adjust the future reference ZMP to satisfy the actual CoM initial condition, while maintaining the required walking speed.

The challenge of this method is that, since we have to solve a nonlinear function, it's not guaranteed that we can find a solution for T, given an arbitrary initial condition of CoM. One might need to add more design parameters for ZMP to solve this problem, which will lead to a underdetermined case. The next case is where we can find a linear relation between the unknown parameter of ZMP and the initial condition of x_u, so that there's always a solution for arbitrary values of CoM initial conditions.

2.2.3.2 Case II: Unknown step length

To make the planning process easier to implement, we will assume the only unknown parameter is the step length. Because the step length enters the cubic/linear spline ZMP equation linearly, it will take much less computational time to solve. Here, we start the step with single support, followed by double support. So the ZMP trajectory for one step will be a linear/cubic spline as shown in Figure 2.10:

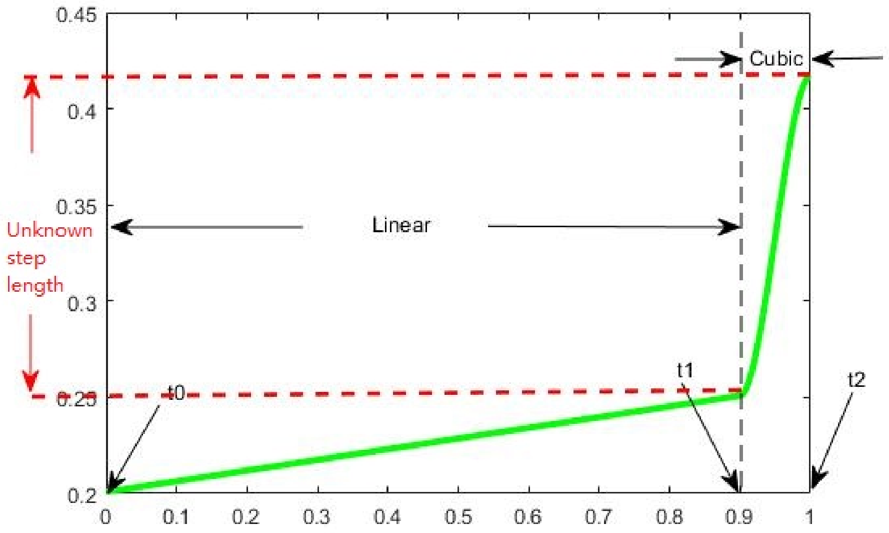


Figure 2.10: Linear/Cubic ZMP

In this case, we assume the distance traveled in single support is fixed and smaller than the foot length, since ZMP has to be in some safe margin to avoid tipping over. The only unknown becomes the step length during double support phase, Δx and Δy, for x and y direction. The algorithm for Δy is the same as Δx, so here only the procedures to solve for Δx is showed.

With the unknown step length Δx enters the linear/cubic spline of ZMP equation, the coefficient in (2.27) for the i-th step becomes:

$$\begin{aligned} \overline{a}_k^{(i)} &= \overline{a}_k^{(i)}(\Delta x), k \in 0, 1 \\ \overline{b}_k^{(i)} &= \overline{b}_k^{(i)}(\Delta x), k \in 0, 1, 2, 3 \end{aligned} \tag{2.32}$$

Unlike (2.27), now $\overline{a}_k^{(i)}$ and $\overline{b}_k^{(i)}$ are linear functions of Δx, since Δx only enters the a_k and b_k linearly in (2.5) but won't enter the polynomials of time.

Then follow the same procedure from (2.28) to (2.31) we have:

$$x_u^\star(0) = x_u^\star(0, x_{zmp}(t, \Delta x)) = x_c(0) + \frac{1}{\omega_0}\dot{x}_c(0) \tag{2.33}$$

where $x_u^\star(0, x_{zmp}(t, \Delta x))$ is also a linear function of Δx. The solution for Δx exists and is unique.

Up to this point, we have a known ZMP trajectory that guarantees convergence of CoM. Now we can recover CoM trajectory using this ZMP as input.

First, using the given initial condition for x_u and the ZMP trajectory designed by solving unknown parameter, we can recover x_u from (2.17). The initial condition for x_s can be calculated using relation (2.13). Then x_s can be recovered using solution (2.21) with the designed ZMP trajectory as input. Now we have both x_u and x_s trajectories ready, x_c can be recovered by applying the change of coordinate (2.13) again.

CHAPTER 3

STABILIZER DESIGN

From walking pattern generation part, we have generated a CoM trajectory which matches the desired ZMP trajectory. However this CoM trajectory is only valid for LIP model, that is, we won't have the expected desired ZMP trajectory if we control the robot to follow this CoM trajectory, due to the deviation between a real robot and LIP model. At this point, it's necessary to design a series of feedback controllers that compensate the error due to model deviation and achieve the desired CoM and ZMP trajectories on the real robot, or modify the generated CoM and ZMP such that the real robot can still keep its balance when walking. We call this series of controllers stabilizers.

The advantage of this 2 layer control structure is that, it decouples the complicated task into two separate tasks, and it's easier to achieve each of the tasks. For path planning, we only need to focus on trajectory generation based on LIP model while in the stabilization part, we only need to be focus on the tracking. In other words, it will have the same procedures as controlling a traditional manipulator. This hierarchical control method is used in most of the biped walking control works, among which, [19], [3], [4] and [20] are the most representative works, though they use different planning methods and stabilizers.

However, the drawback of this control structure is that, if there's error or uncertainties in each part of the procedures, the errors will accumulate, which may cause instability. Especially in implementation, if the robot turns out to be falling, it's hard to tell which part contribute to the failure.

There are a large variety of different types of stabilizer, based on different types of pattern generation methods. Here, based on the functionality, one can categorize most of the existing stabilizers into two level:

- High level stabilizer

  The higher level controller takes care of the stability of walking, namely, it will feedback the actual CoM and ZMP of the robot, then compensate the error between the desired and actual CoM, ZMP by modifying the reference CoM or ZMP that the robot is actually tracking. Some of the controllers modify the desired CoM given by pattern generation then the robot will follow this modified CoM, which implicitly modifies the actual ZMP such that it will track the desired CoM [21], [19], [20]. Another slightly different approach is that the controller directly modifies the reference ZMP for the robot to track [22], [3], [4].

- Low level stabilizer
  From the higher level controller of stabilizer, either a CoM or a ZMP will be sent to robot to track. The lower level controller will focus on this tracking task by directly controlling each individual joint. The joints could be position controlled via PID controller [23], [24], [25], [21], [19], or they could be directly torque controlled [20], [3], [6].

Up to this point, we can draw the complete scheme of biped walking control, based on the "planning + stabilizing" method:

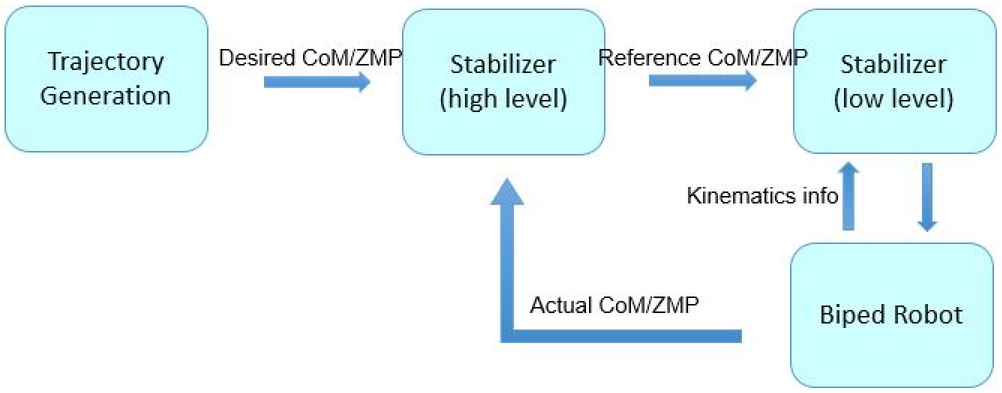


Figure 3.1: Control flow of biped walking

It should be noted that the planning process is included in "trajectory generation" part. This diagram shows the general case of the hierarchical control: plan a trajectory (for n steps) then use stabilizer to stabilize and track it. One can also add replanning procedure in the planning part, which will replan the trajectory for each step and then use stabilizer to stabilize and track it for that step. The replanning procedure is used in implementation in Section 4.3.

In this chapter, we will concentrate on the low level controller of stabilizer, assuming we already have a stabilized CoM/ZMP trajectory, and we want to track this trajectory. For ZMP tracking, early works have proposed some intuitive methods to directly control the ZMP of the robot by changing the torso angle and ankle angle [26]. In 2010 Kajita proposed a novel way to track the ZMP by calculating the distribution of ZMP for each foot and designed a torque controller for ankles to track the calculated ZMP [3]. For CoM tracking, the most famous method is the Whole Body Cooperation (WBC) control and Resolved Momentum control, which need the cooperation of each individual position controlled or velocity controlled joints of the robot. The concept of WBC was proposed by Sujihara in 2002 [23], which utilize the CoM Jacobian to map the joint velocities and CoM velocity. Later WBC method was extended to embedded limb motions by Choi in 2007 [19]. The Resolved Momentum control was first proposed by Kajita in 2003 [24], which instead of mapping CoM velocity, maps the overall linear and angular momentum with joint velocities. Later this method is extended to include embedded motion in [25]. These two methods are similar in some aspect since they all map the joint velocities to the target velocities via some Jacobian matrices. More recent works have been using torque controlled robot and tracked the CoM acceleration by generating appropriate torques in each joint, calculating inverse dynamics [6].

3.1 Ankle Position Controller

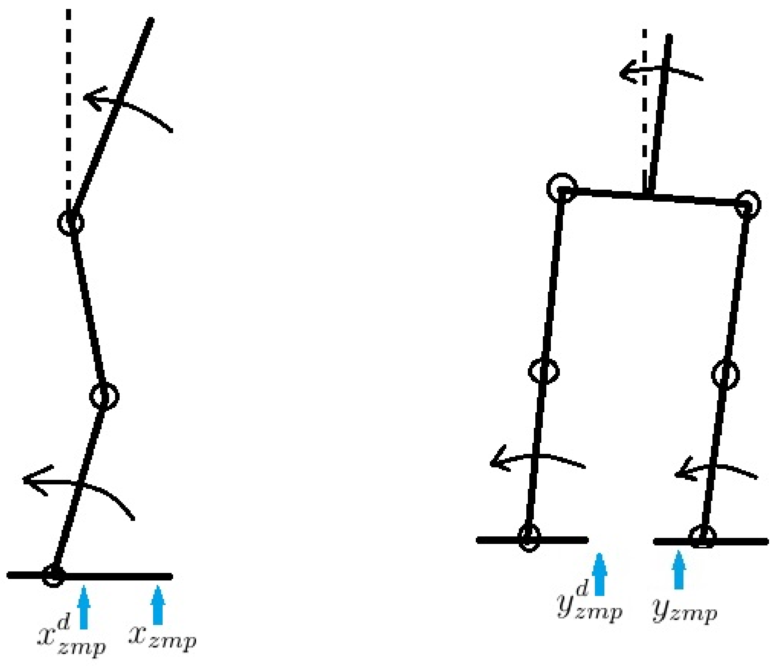


Figure 3.2: Ankle position control

Earlier works have focused on achieving ZMP tracking by modification of desired joint trajectories derived by inverse kinematics. The early version of Honda humanoid robot uses 2 position controllers for balance control [27]. One of the controllers modifies the ankle angle to control the actual ground reaction force indirectly, such that the actual ZMP tracks the desired ZMP. The other controller is used to adjust the posture of the robot by changing the hip angle, such that a torque is applied on the torso which restores the posture of the body. Another similar work gives a more straightforward control method to achieve ZMP tracking and balancing in [26]. Desired ankle angle is modified base on the ZMP error. While this controller can intuitively track the desired ZMP, the body posture is changed and tend to deviate from the desired posture. Therefore a body posture controller is proposed to maintain the torso orientation by changing the hip angle.

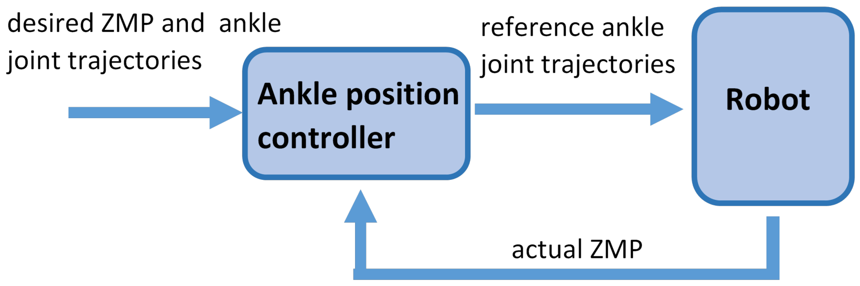

desired ZMP and ankle joint trajectories
Ankle position controller
reference ankle joint trajectories
Robot
actual ZMP

Figure 3.3: Ankle position controller scheme

According to these previous works, we can summarize the idea of ZMP tracking using ankle position controller as following (also see Figure 3.3). LIPM is used to generate the desired ZMP and corresponding desired CoM trajectory. By assuming the CoM of the robot is concentrated at torso, the torso trajectory is the same as CoM trajectory. We can solve for the joint trajectories using inverse kinematics given the trajectory of torso. We call these joint trajectories desired joint trajectories. While the joints are tracking the desired joints trajectories but there's error between actual ZMP and desired ZMP due to model difference and disturbance, the reference ankle joint angle that is actually being tracked will be modified base on ZMP error: If the actual ZMP is ahead of the desired ZMP, the ankle will rotate the body backward a little to bring the ZMP back towards the desired value, if the actual ZMP is behind the desired value, the ankle will rotate the body forward a little. The amount of rotation is a function of ZMP error. Here we make the controller in a PD controller form:

$$\begin{aligned}
q^r_{5,i} &= q^d_{5,i} + (K_{px}\delta x_{zmp} + K_{dx}\delta \dot{x}_{zmp}) \\
q^r_{6,i} &= q^d_{6,i} + (K_{py}\delta y_{zmp} + K_{dy}\delta \dot{y}_{zmp}) \\
\delta x_{zmp} &= x_{zmp} - x^d_{zmp} \\
\delta y_{zmp} &= y_{zmp} - y^d_{zmp} \\
\delta \dot{x}_{zmp} &= \dot{x}_{zmp} - \dot{x}^d_{zmp} \\
\delta \dot{y}_{zmp} &= \dot{y}_{zmp} - \dot{y}^d_{zmp}
\end{aligned} \tag{3.1}$$

where the superscript d is the desired trajectory from inverse kinematics assuming the CoM is concentrated at the origin of the torso, as explained in the previous paragraph. The superscript r is the actual reference trajectory that the ankle joint is tracking. So if there's no model difference or any disturbance, there will be no error therefore the reference joint trajectory is the same as desired joint trajectory. δx_{zmp}, δy_{zmp}, $\delta \dot{x}_{zmp}$ and $\delta \dot{y}_{zmp}$ are ZMP errors and derivatives of ZMP errors. The subscript 5 and 6 corresponding to the two ankle joints as shown by $J5$ and $J6$ in Figure 3.4. And $i = R, L$ indicates the left or right ankle. K_{px}, K_{py}, K_{dx} and K_{dy} are controller gains.

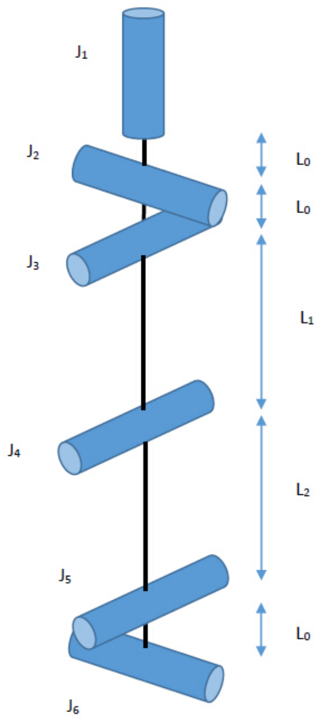


Figure 3.4: Joints in one leg: J1 to J3 are hip joints, J4 is knee joint, J5 and J6 are ankle joints.

From the simulation of ZMP tracking by using ankle position controller (see Figure 4.1), we conclude that the performance is bad. Based on (2.4), if we consider the robot as an LIPM, then the ZMP can be decided only using CoM. So if we track the CoM trajectory, the ZMP should be automatically tracked. The CoM tracking controller is:

$$\begin{aligned}
q^r_{5,i} &= q^d_{5,i} + (K_{px}\delta x_c + K_{dx}\delta\dot{x}_c) \\
q^r_{6,i} &= q^d_{6,i} + (K_{py}\delta y_c + K_{dy}\delta\dot{y}_c) \\
\delta x_c &= x_c - x^d_c \\
\delta y_c &= y_c - y^d_c \\
\delta\dot{x}_c &= \dot{x}_c - \dot{x}^d_c \\
\delta\dot{y}_c &= \dot{y}_c - \dot{y}^d_c
\end{aligned} \tag{3.2}$$

which is similar to the previous controller for ZMP tracking. However, the error terms used here are δx_c, δy_c, $\delta\dot{x}_c$ and $\delta\dot{y}_c$, which are the errors of CoM and their derivatives. Assuming the changing of CoM height is negligible (true for static stance case and some slow motion), we can simply change the CoM in horizontal plane by rotating the body forward and backward.

In addition, the noises of position sensors for links and joints are much smaller than force-torque sensors used for ZMP calculation, so the calculated actual CoM trajectory will be much smoother than the calculated actual ZMP trajectory, which allows us to implement derivative control.

3.2 Whole Body Cooperation Control

While the high level stabilizer is modifying the reference CoM from the planned CoM to stabilize the biped walking, whole body cooperation (WBC) control is a lower level controller which make sure the biped robot tracks the modified reference CoM. WBC Control is controlling each individual joint's angle such that the robot CoM follows the reference trajectory. Instead of using the nonlinear inverse kinematics, this is achieved by mapping joint velocities and CoM velocity, using CoM Jacobian. This method is first proposed in [23], where quadratic programming is used to find the discrete joint angles under given CoM and CoM Jacobian constraints. Since the walking motion includes swing foot motion tracking, later works developed similar WBC method allowing tracking embedded motion like swing foot motion at the same time, as proposed in [28], [21] and [19]. Some recent works also used WBC control based capture point dynamics and centroidal moment pivot dynamics, [4] and [29].

3.2.1 CoM Jacobian

For a manipulator with a fixed base and n joints, the CoM of a manipulator can be expressed in terms of the CoM of each link:

$$P_c(q) = \frac{\sum_1^n m_i P_i(q)}{\sum_1^n m_i} \tag{3.3}$$

where $P_c = [x_c \; y_c \; z_c]^T$ is the position vector of CoM in world frame, P_i is the CoM position of link i in world frame, $q = [q_1 \; q_2 \; ... \; q_n]^T$ are the joint positions, and m_i is the mass of link i.

Since the position of each link $P_i(q)$ has a nonlinear relation between joint positions due to forward kinematics, the CoM position $P_c(q)$ also has a nonlinear relation with joint positions. Therefore, it's hard to calculate the inverse P_c^{-1} to get the joint position trajectories corresponding to a desired CoM trajectory, especially for biped robot since the number of joints is much larger than 3. However, the velocities of joints and CoM has a linear relation and are related by CoM Jacobian. One can control the joint velocities to achieve a desired CoM velocity trajectory.

For a serial-link manipulator, the velocity of the end-effector can be calculated using joint velocities and the Jacobian, where the Jacobian gives the relation between joint velocities and end effector velocity:

$$\begin{pmatrix} \dot{P} \\ \omega \end{pmatrix} = J(q)\dot{q} \tag{3.4}$$

where J is the $6 \times N$ Jacobian matrix which is a function of the joint positions q, P is the position vector and ω is the angular velocity vector of the end-effector. More specifically, we have:

$$\begin{pmatrix} \dot{P} \\ \omega \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{3.5}$$

and

$$J(q) = \begin{pmatrix} J_v(q) \\ J_\omega(q) \end{pmatrix} \tag{3.6}$$

in which $\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z$ are linear and angular velocities of the end effector. J_v and J_ω are $3 \times N$ Jacobians for linear velocities and angular velocities. We will use this concept to relate joint velocities of our humanoid robot to body velocities of specific links, as shown later in this section and Section 3.2.2.

The Jacobian for linear velocities is:

$$J_v(q) = \frac{\partial P(q)}{\partial q} \tag{3.7}$$

which is the derivative of end effector respect to joint positions.

The Jacobian for angular velocities is a little more complicated (assuming revolute joints):

$$J_\omega(q) = \begin{pmatrix} k_0(q) & k_1(q) & ... & k_{n-1}(q) \end{pmatrix} \tag{3.8}$$

where k_i is axis of rotation for the i-th joint expressed w.r.t the base link frame.

Similarly, for a biped robot with n revolute joints, the CoM can be treated as a special end effector with only linear velocity. We can find the Jacobian to relate joint velocities and CoM velocity in world frame:

$$\dot{P}_c = J_c(q)\dot{q} \tag{3.9}$$

where P_c is the position vector of CoM in world frame, J_c is called CoM Jacobian and $q = (q_1 \; q_2 \; ... \; q_n)^T$ are joint angles.

In biped robot, we usually take the floating torso as the base (link 0). The CoM position can be written in terms of the CoM position of each link in the base link frame:

$$\begin{aligned}
{}^{0}P_c &= \frac{\sum_0^n m_i {}^0P_i}{\sum_0^n m_i} \\
&= \frac{\sum_0^n m_i {}^0P_i}{M} \\
{}^{0}\dot{P}_c &= \frac{\sum_0^n m_i {}^0\dot{P}_i}{M} \\
&= \frac{\sum_1^n m_i {}^0\dot{P}_i}{M} \qquad since\ {}^0\dot{P}_0 = 0 \\
&= \frac{\sum_1^n m_i {}^0J_i\dot{q}}{M} \\
&= \frac{\sum_1^n m_i {}^0J_i}{M}\dot{q} = {}^0J_c\dot{q}
\end{aligned} \tag{3.10}$$

where M is the total mass of robot. The subscript c stands for CoM, and the notation ${}^0P_c$ represents the position of CoM expressed w.r.t 0 frame (torso frame). Compare with (3.9), the CoM Jacobian in base frame is:

$${}^0J_c = \frac{1}{M}\sum_1^n m_i {}^0J_i \tag{3.11}$$

where ${}^0J_i$ is the Jacobian of CoM for link i in base frame.

Generally, if a rigid body is moving in both frame A and frame B, and a local frame a is attached to the rigid body with origin at point a, then the velocity of frame a in frame A can be calculated using the relation between frame a and frame B, as well as frame B and frame A (also used in [23], [28], [21], [19]):

$$\begin{pmatrix} {}^A\dot{P}_a \\ {}^A\omega_a \end{pmatrix} = \begin{pmatrix} {}^A\dot{P}_B + {}^A\omega_B \times ({}^AR_B {}^BP_a) + {}^AR_B {}^B\dot{P}_a \\ {}^A\omega_B + {}^AR_B {}^B\omega_a \end{pmatrix} \tag{3.12}$$

in which, ${}^AR_B$ is the orientation of frame B w.r.t coordinate frame A. ${}^B\dot{P}_a$ is linear velocity of origin of frame a, w.r.t frame B expressed relative to coordinate frame B, ${}^B\omega_a$ is angular velocity of frame a, w.r.t frame B expressed relative to coordinate frame B. ${}^BP_a$ is the position of origin of frame a, w.r.t frame B, expressed relative to coordinate B. Other variables share the same notation rules.

It should be noted that even though the CoM is a point and its velocity in world frame can be calculated from the first row of (3.12), the second row is still needed for base frame velocities calculation for later use.

We can compute the velocity of CoM in world frame using the first row of formula (3.12):

$$\dot{P}_c = \dot{P}_0 + \omega_0 \times (R_0{}^0P_c) + R_0{}^0\dot{P}_c \tag{3.13}$$

where R_0 is the orientation of base link in w.r.t world frame. $\dot{P}_0$ and ω_0 are base link velocities w.r.t world frame, expressed in world frame. They can be calculated by the following procedures.

Using both rows of the formula (3.12), foot velocities w.r.t world frame, expressed in world frame can be expressed as:

$$\begin{pmatrix} \dot{P}_f \\ \omega_f \end{pmatrix} = \begin{pmatrix} \dot{P}_0 + \omega_0 \times (R_0{}^0P_f) + R_0{}^0\dot{P}_f \\ \omega_0 + R_0{}^0\omega_f \end{pmatrix} \tag{3.14}$$

where the subscript f denote the foot frame.

If it's a stance foot, then both linear and angular velocities w.r.t world frame are zero:

$$\begin{aligned} \dot{P}_f &= 0 \\ \omega_f &= 0 \end{aligned} \tag{3.15}$$

Substituting the zero velocities of foot (3.15) into (3.14) and rearrange, we have the torso velocities w.r.t world frame, expressed in world frame:

$$\begin{pmatrix} \dot{P}_0 \\ \omega_0 \end{pmatrix} = \begin{pmatrix} -\omega_0 \times (R_0{}^0P_f) - R_0{}^0\dot{P}_f \\ -R_0{}^0\omega_f \end{pmatrix} \tag{3.16}$$

Substituting torso velocities $\dot{P}_0$ and ω_0 from (3.16) into (3.13), we have:

$$
\begin{aligned}
\dot{P}_c &= \dot{P}_0 + \omega_0 \times (R_0{}^0P_c) + R_0{}^0\dot{P}_c && (3.17)\\
&= -\omega_0 \times (R_0{}^0P_f) - R_0{}^0\dot{P}_f - (R_0{}^0\omega_f) \times (R_0{}^0P_c) + R_0{}^0\dot{P}_c \\
&\quad \textit{substituting } \omega_0 && (3.18)\\
&= (R_0{}^0\omega_f) \times (R_0{}^0P_f) - R_0{}^0\dot{P}_f - (R_0{}^0\omega_f) \times (R_0{}^0P_c) + R_0{}^0\dot{P}_c \\
&= R_0({}^0\omega_f \times {}^0P_f) - R_0{}^0\dot{P}_f - R_0({}^0\omega_f \times {}^0P_c) + R_0{}^0\dot{P}_c \\
&= R_0({}^0\omega_f \times ({}^0P_f - {}^0P_c) + {}^0\dot{P}_c - {}^0\dot{P}_f) \\
&= R_0(({}^0P_c - {}^0P_f) \times {}^0\omega_f + {}^0\dot{P}_c - {}^0\dot{P}_f) \\
&= R_0(([{}^0P_c - {}^0P_f]_\times)^0J_{\omega_f} + {}^0J_c - {}^0J_{P_f})\dot{q} = J_c\dot{q}
\end{aligned}
$$

where ${}^0J_{\omega_f}$ and ${}^0J_{P_f}$ are Jacobians of the linear and angular velocity of stance foot link in base frame, and $[]_\times$ is the outer product matrix of a vector. So the CoM Jacobian in world frame is:

$$J_c = R_0([{}^0P_c - {}^0P_f]_\times{}^0J_{\omega_f} + {}^0J_c - {}^0J_{P_f}) \tag{3.19}$$

3.2.2 Embedded motion

For a biped walking robot, there are lots of constraints for the motion and posture. For example, to minimize the impact of landing, the swing leg's velocity in world frame should be zero when it contact with the ground. Sometimes arm motions are specified to compensate the momentum. These constraints are in the form of embedded motion.

For a specified link i, since the stance foot frame is fixed in world frame, each extreme link of limb, link i, can be treated as an end effector of a manipulator whose base link is the stance foot. The torso is one of the links in this manipulator. The embedded motion in world frame of this link can be expressed as:

$$\begin{pmatrix} \dot{P}_i \\ \omega_i \end{pmatrix} = J_i(q)\dot{q} \tag{3.20}$$

For example, the motion of swing leg is:

$$\begin{pmatrix} \dot{P}_{sw} \\ \omega_{sw} \end{pmatrix} = J_{sw}(q)\dot{q} \tag{3.21}$$

where subscript sw denotes the swing foot link, and J_{sw} is the Jacobian of swing foot frame w.r.t world frame. If each leg has 6 joints, then J_{sw} is a 6×12 matrix. And it can be calculated using ${}^0J_{sw}$, ${}^0J_f$ and the information of base link.

In the torso base frame, we have the following relation of stance foot and swing foot:

$$\begin{aligned} \begin{pmatrix} {}^0\dot{P}_{sw} \\ {}^0\omega_{sw} \end{pmatrix} &= {}^0J_{sw}\dot{q} \\ \begin{pmatrix} {}^0\dot{P}_f \\ {}^0\omega_f \end{pmatrix} &= {}^0J_f\dot{q} \end{aligned} \tag{3.22}$$

In the stance foot frame:

$$\begin{pmatrix} {}^f\dot{P}_{sw} \\ {}^f\omega_{sw} \end{pmatrix} = {}^fJ_{sw}\dot{q} \tag{3.23}$$

Applying formula (3.12) by assigning stance foot frame to frame A, base (torso) frame to frame B and swing foot frame to frame a, we can write the velocity of swing foot in stance foot frame by using base (torso) frame as an intermediate frame:

$$\begin{aligned} \begin{pmatrix} {}^f\dot{P}_{sw} \\ {}^f\omega_{sw} \end{pmatrix} &= \begin{pmatrix} {}^f\dot{P}_0 + {}^f\omega_0 \times ({}^fR_0{}^0P_{sw}) + {}^fR_0{}^0\dot{P}_{sw} \\ {}^f\omega_0 + {}^fR_0{}^0\omega_{sw} \end{pmatrix} \\ &= \begin{pmatrix} {}^f\dot{P}_0 - ({}^fR_0{}^0P_{sw}) \times {}^f\omega_0 \\ {}^f\omega_0 \end{pmatrix} \\ &\quad + \begin{pmatrix} {}^fR_0{}^0\dot{P}_{sw} \\ {}^fR_0{}^0\omega_{sw} \end{pmatrix} \end{aligned} \tag{3.24}$$

Let

$$\begin{aligned} T_1 &= \begin{pmatrix} I_3 & -[{}^fR_0{}^0P_{sw}]_\times \\ 0_3 & I_3 \end{pmatrix} \\ T_2 &= \begin{pmatrix} {}^fR_0 & 0_3 \\ 0_3 & {}^fR_0 \end{pmatrix} \end{aligned} \tag{3.25}$$

then (3.24) becomes:

$$\begin{pmatrix} {}^f\dot{P}_{sw} \\ {}^f\omega_{sw} \end{pmatrix} = T_1 \begin{pmatrix} {}^f\dot{P}_0 \\ {}^f\omega_0 \end{pmatrix} + T_2 \begin{pmatrix} {}^0\dot{P}_{sw} \\ {}^0\omega_{sw} \end{pmatrix} \tag{3.26}$$

Also, we can replace the velocities of torso frame w.r.t stance foot frame expressed in stance frame (${}^f\dot{P}_0$ and ${}^f\omega_0$) with the velocities of stance frame w.r.t torso frame expressed in torso frame (${}^0\dot{P}_f$ and ${}^0\omega_f$):

$$\begin{pmatrix} {}^f\dot{P}_0 \\ {}^f\omega_0 \end{pmatrix} = -\begin{pmatrix} {}^fR_0 & 0_3 \\ 0_3 & {}^fR_0 \end{pmatrix} \begin{pmatrix} {}^0\dot{P}_f \\ {}^0\omega_f \end{pmatrix} = -T_2 \begin{pmatrix} {}^0\dot{P}_f \\ {}^0\omega_f \end{pmatrix} \tag{3.27}$$

Substituting (3.27) into (3.26), we have:

$$\begin{aligned} \begin{pmatrix} {}^f\dot{P}_{sw} \\ {}^f\omega_{sw} \end{pmatrix} &= -T_1T_2 \begin{pmatrix} {}^0\dot{P}_f \\ {}^0\omega_f \end{pmatrix} + T_2 \begin{pmatrix} {}^0\dot{P}_{sw} \\ {}^0\omega_{sw} \end{pmatrix} \\ &= -T_1T_2{}^0J_f\dot{q} + T_2{}^0J_{sw}\dot{q} \\ &= (-T_1T_2{}^0J_f + T_2{}^0J_{sw})\dot{q} = {}^fJ_{sw}\dot{q} \end{aligned} \tag{3.28}$$

Now we have the Jacobian ${}^fJ_{sw}$. We need to transform it to world frame. Applying formula (3.12) again by assigning world frame to frame A, stance foot frame to frame B and swing foot frame to frame a, and noting that the velocities of stance frame w.r.t world frame are zero, we have:

$$\begin{aligned} \begin{pmatrix} \dot{P}_{sw} \\ \omega_{sw} \end{pmatrix} &= \begin{pmatrix} R_f & 0_3 \\ 0_3 & R_f \end{pmatrix} \begin{pmatrix} {}^f\dot{P}_{sw} \\ {}^f\omega_{sw} \end{pmatrix} \\ &= \begin{pmatrix} R_f & 0_3 \\ 0_3 & R_f \end{pmatrix} {}^fJ_{sw}\dot{q} = J_{sw}\dot{q} \end{aligned} \tag{3.29}$$

3.2.3 WBC Controller

Combining (3.17) and (3.21), we have the overall Jacobian J that transforms the joints velocities to CoM velocity and embedded motion of swing foot:

$$\begin{pmatrix} \dot{P}_c \\ \dot{P}_{sw} \\ \omega_{sw} \end{pmatrix} = J(q)\dot{q} \tag{3.30}$$

where J has to be full rank, in order to map the whole target space velocities to joints space velocities.

$$J = \begin{pmatrix} J_c \\ J_{sw} \end{pmatrix} \tag{3.31}$$

is a $9 \times n$ matrix, and $n \geq 12$ since it at least includes the lower body.

Therefore, the corresponding joint velocities needed to achieve the desired CoM velocity and embedded motion is:

$$\dot{q} = J^{\dagger}(q) \begin{pmatrix} \dot{P}_c \\ \dot{P}_{sw} \\ \omega_{sw} \end{pmatrix} \tag{3.32}$$

where $J^{\dagger}(q)$ is the pseudo inverse of $J(q)$, which can be calculated using following formula with the assumption that J is full rank and the fact that all entries of J are real:

$$J^{\dagger} = J^T (J J^T)^{-1} \tag{3.33}$$

In addition, $J^{\dagger}(q)$ can be calculated in real time using the equations derived in the Section 3.2.1, Section 3.2.2 and the joint angles from sensors.

The CoM position trajectory, the position and orientation trajectories of embedded motion are given as references. We need to control the joint velocities such that the actual CoM and embedded motion will follow the references. Here we can use a proportional controller, where the target space velocities are inputs:

$$\begin{aligned} \begin{pmatrix} \dot{P}_{c,ref} \\ \dot{P}_{sw,ref} \end{pmatrix} &= K_{po} \left(\begin{pmatrix} P_{c,ref} \\ P_{sw,ref} \end{pmatrix} - \begin{pmatrix} P_c \\ P_{sw} \end{pmatrix} \right) \\ \omega_{sw,ref} &= -R_{sw,ref}^{-1} K_{or} u \lambda \end{aligned} \tag{3.34}$$

where the subscript ref denote the reference trajectory, P is position and R is rotation matrix. So $P_{c,ref}$, $P_{sw,ref}$ and $R_{sw,ref}$ are reference trajectories, expressed w.r.t world frame.

In (3.34), the first equation is a simple proportional controller for CoM position and embedded motion position tracking, K_{po} is the controller gain. For embedded motion orientation tracking, the controller (second equation) will be a little more complicated since we cannot simply subtract the reference rotation matrix by the actual orientation matrix to calculate the error.

The subtraction of rotation matrix is meaningless. Here we use axis-angle representation for the orientation. If we consider the rotation from reference orientation to actual orientation, then u is the axis of rotation expressed in reference orientation frame and λ is the angle of rotation. So $-u\lambda$ is a measurement of the orientation difference between the actual orientation and reference orientation. It should be noted that u and λ are both changing during motion. When $-u\lambda$ is zero, it means that the reference is perfectly tracked. K_{or} is the controller gain to calculate control input (angular velocity). So $-K_{or}u\lambda$ is the controller input, expressed in reference frame. Finally, multiplying this by $R_{sw,ref}^{-1}$ we get the control input in world frame.

Now we have world frame velocities as control input to track the CoM and embedded motion. By applying (3.32), we can calculate the joint velocities that are needed for tracking. However, in most cases, instead of joints velocities, only the positions of joints can be controlled using PID controller. In this case, we can implement the above controller numerically:

$$\begin{aligned} q_{ref}^{k+1} &= q^k + \delta q_{ref}^k \\ \delta q_{ref}^k &= J^\dagger(q^k)\begin{pmatrix} K_{po}\left(\begin{pmatrix} P_{c,ref} \\ P_{sw,ref}\end{pmatrix} - \begin{pmatrix} P_c \\ P_{sw}\end{pmatrix}\right) \\ -R_{sw,ref}^{-1}K_{or}u\lambda \end{pmatrix}\delta t \end{aligned} \tag{3.35}$$

where δt is the sample time. The reference joint angle for each individual joint PID controller to track during the (k+1)-th sample period is q_{ref}^{k+1}. It's calculated by summing up the actual joint angle at k-th sample and the increment of joint angle calculated using WBC controller. The increment of joint angles δq_{ref}^k is calculated using controller (3.34) and the Jacobian $J(q^k)$, where q^k comes from the sensors. This equation holds for small sample time δt since if we multiply each side by $1/\delta t$, we get exactly Eq .3.32.

It should be noted that the reference CoM trajectory $P_{c,ref}$ can be generated using the boundedness condition method derived in Section 2.2.3. The reference swing foot position and orientation trajectories are planned to form an support polygon such that the ZMP will be inside it all the time. The details of swing foot trajectory generation will be discussed in Section 4.3.1.2.

CHAPTER 4

SIMULATIONS

Simulations are done using the humanoid robot Reem-C [30] developed by Pal Robotics, under the physics engine Gazebo [31] in ROS [32]. The simulations in this chapter use the lower body of Reem-C, whose detailed parameters can be found in Appendix A.0.1. The controller is written in Matlab, and the Robotics System Toolbox [33] is used for exchanging information between Matlab and Gazebo. All simulation videos can be found at [34].

The stabilizers discussed in Chapter 3 are implemented. First, the ankle position controller in 3.1 is implemented to track a desired ZMP trajectory and then a CoM trajectory. The poor performance of this tracking controller brings us to the use of WBC controller.

The WBC controller is tested by tracking a CoM trajectory of squat motion. The result shows that the performance is good enough for us to implement it in walking simulation.

The last part of the simulation is walking simulation, which combines replanning algorithm based on the boundedness condition in Section 2.2.3, and the WBC controller as the lower level stabilizer. The details of implementation will be discussed.

4.1 Implementation of Ankle Position Controller

In this section, the controller derived in Section 3.1 is implemented in simulation. First, the ankle position controller for ZMP tracking is tested on Reem-C to track a sinusoidal ZMP motion, which gives an unsatisfactory result, due to the noise in ZMP calculation and affect of CoM acceleration. Therefore, the controller is improved by tracking the CoM directly, which gives a better simulation result.

4.1.1 ZMP tracking

In order to test the efficiency of the ankle position controller, a simulation for stance is constructed, as following. A sinusoidal motion for CoM in y direction is generated, the corresponding ZMP trajectory is calculated using LIPM and is used as the desired ZMP.

$$y_c^d = A \sin\left(\frac{2\pi}{T} t\right) \tag{4.1}$$

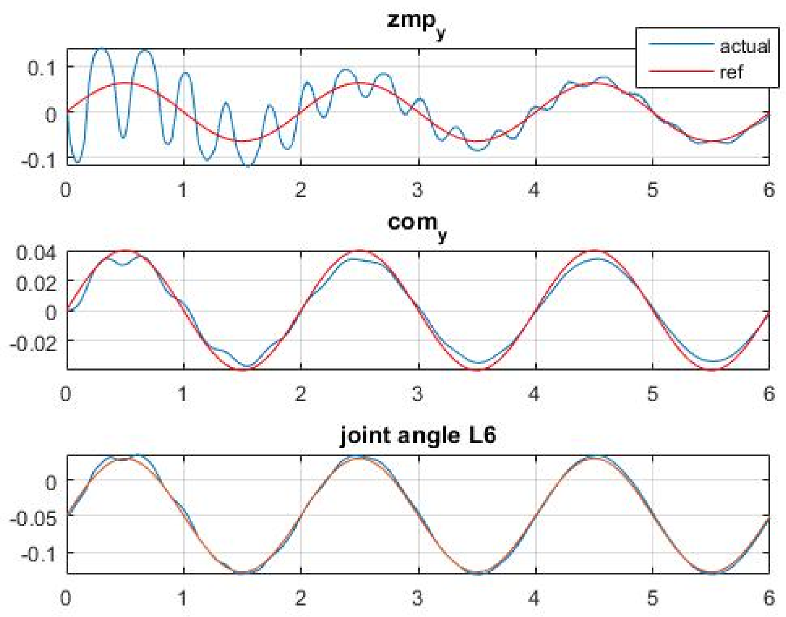
$$y_{zmp}^d = y_c^d - \frac{h_0}{g} \ddot{y}_c^d \tag{4.2}$$

where $A = 0.04m$ and $T = 2s$, the corresponding ZMP trajectory will also be sinusoidal and the magnitude is small enough to keep inside the support polygon. Here $h_0 = 0.59m$ and the robot posture is adjusted such that the actual CoM height matches h_0 (the corresponding torso height is 0.75, which is used for inverse kinematics). The desired trajectories for joints are generated by assuming the torso follows desired y_c^d and $x_c^d = 0$.

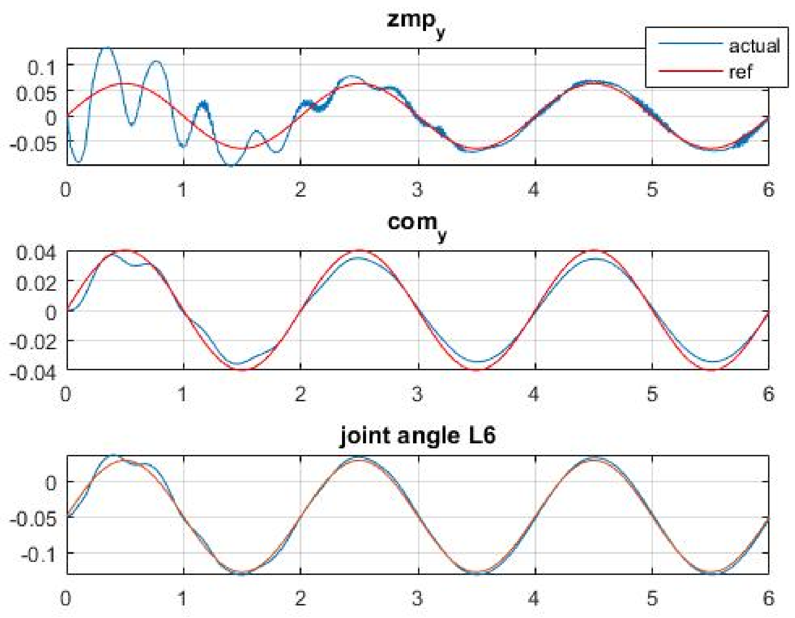
At this point, since most of the mass is on the torso and the torso will rotate if the ankle rotate, which increase the total angular momentum, a simple compensation for hip angles is used:

$$q_{2,i}^r = q_{2,i}^d - (K_{py} \delta y_{zmp} + K_{dy} \delta \dot{y}_{zmp}) \tag{4.3}$$

where the subscript 2 corresponds the hip joints, and $(K_{py} \delta y_{zmp} + K_{dy} \delta \dot{y}_{zmp})$ term is actually the modification made to the ankle joints, but with negative sign. The reason is that, if the ankle is rotated by q, then the torso is rotated by q, in order to bring the torso back to its desired posture, the hip should rotate by $-q$.



(a) Openloop



(b) P control with $k_{py} = 0.005$

Figure 4.1: Simulation for sinusoidal stance of ZMP tracking in y direction

In Figure 4.1, it's shown that the ankle angle is tracked pretty well, and the reference angle is slightly modified by the controller compared with openloop.

The actual CoM trajectory follows the desired trajectory, although it's not because of the controller. For ZMP trajectory, the simulation shows that the ankle position controller makes the ZMP converge to the desired ZMP quicker than openloop. But still the convergence is slow since it need almost 2 seconds to converge to an acceptable range, which won't work well for walking, since we need immediate convergence of ZMP for walking, otherwise the large and persistent oscillation of ZMP error will cause the actual ZMP reach the boundary of support polygon and the robot tips over.

The maximum control loop frequency the simulation environment can achieve is around 150Hz, which is low. Theoretically, we can have any control rate, if the iteration of physics engine is slow enough. However, observations show that the maximum rate that can be achieved is around 150Hz. Even if we use a very slow iteration in physics engine (for example, 10 per second), the control rate will stay at 150Hz. The control loop frequency is highly related to Gazebo speed and the transmission delay between Matlab and ROS. In order to have 150Hz control frequency and an acceptable time consumption, the physics engine in Gazebo is slowed down by 20 times. However, the frequency of control loop only affects the smoothness of tracking.

The large oscillation at the first few seconds is due to the discontinuity of the desired ZMP. The actual ZMP and it's derivative start at 0, however the desired ZMP's initial derivative is not zero (since it's a sinusoidal function), so a possible solution to have better tracking is to have a desired ZMP trajectory that match the actual ZMP initial condition.

The stiffness of foot will also enlarge the oscillations. The joints of this Reem-C robot is rigid and not compliant due to the fact that it's a position controlled robot. So if there's impact between foot and ground, it won't be absorbed. Also during the tracking of ZMP, there's always tracking error, which sometimes causes the soles to detach the ground, or to be in bad contact with the ground. This will cause bouncing of the feet.

In order to reduce overshoot, PD controller is also implemented, as shown in Figure 4.2. Since the force sensor readings for ZMP calculations are noisy even with an average filter added, it's a challenging to implement, which makes the behavior of PD controller worse than the P controller.

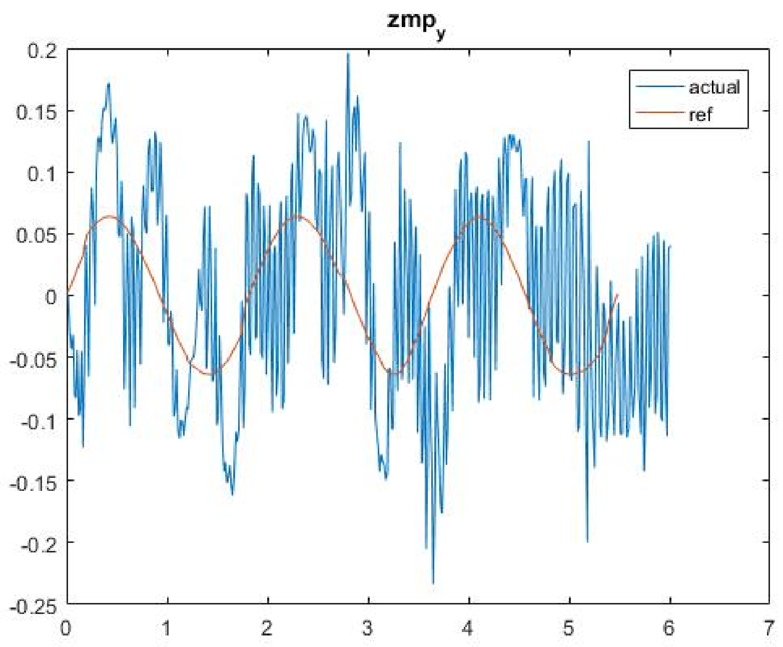


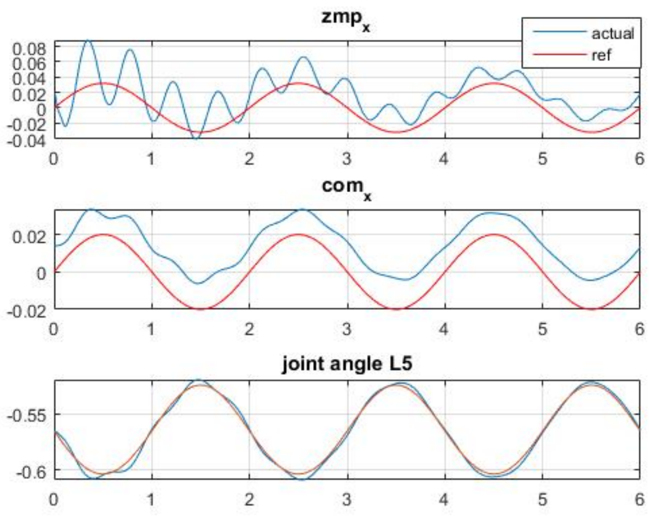
Figure 4.2: PD control for ZMP tracking with $k_{py} = 0.1$, $k_{dy} = 0.002$

The tracking of ZMP is not guaranteed since the ZMP is a function of both CoM and the acceleration of CoM. According to (2.3), for an LIPM, the ZMP can be calculated using CoM and CoM acceleration. For y direction, it will follow the same dynamics. Controlling the ankle angles is actually controlling the CoM position, since CoM is directly related to ankle angles for a LIPM. However, based on (2.3), CoM acceleration also has to be controlled in order to track the ZMP. Therefore, if we simply control the ankle angle depending on the ZMP error, the tracking of ZMP is not guaranteed. For example, if the actual ZMP is ahead of reference ZMP, the controller will try to rotate ankle such that the CoM will rotate backward. In this case, even though x_c is decreased, the rotation will cause a negative acceleration $\ddot{x}_c$, which will be subtracted according to (2.3), and counteract on the ZMP tracking.

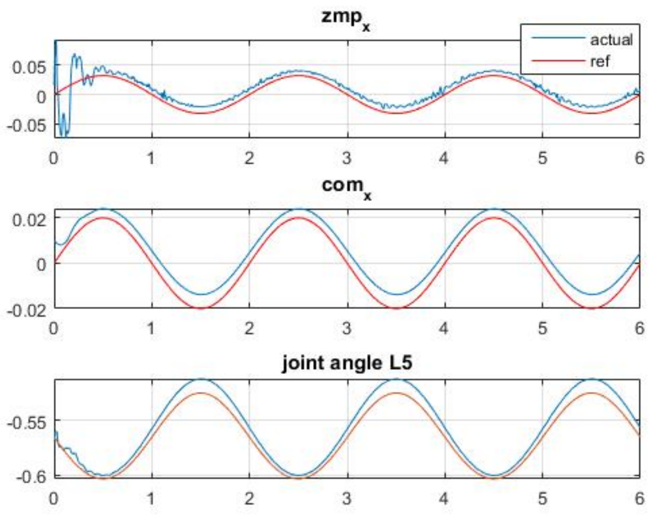
So instead of tracking ZMP, it's expected that it will have a better performance if we directly track the CoM trajectory, since the ZMP is a function of CoM and it's second derivative. However in this case, we have to make sure the reference CoM trajectory that we are tracking will result in the desired ZMP trajectory. Simulations are done in the next section.

4.1.2 CoM tracking

First, a CoM tracking in x direction is simulated using Reem-C lower body. Since the corresponding two ankle joints share the same axis, we can control both joints using the same ankle position controller and the soles will always be in contact with the ground (which means the soles are always in the same plane since the controller will modify the angles by the same amount).



(a) Openloop



(b) PD control with $kpx = -2, kdx = -0.2$

Figure 4.3: Simulation for sinusoidal stance of CoM tracking in x direction

The simulation results show that the CoM is tracked with a small steady state error, and the actual ZMP is also kept closed to the desired ZMP. To eliminate the steady state error, an integral term can be added.

In the openloop simulation, it's shown that the actual CoM is always ahead of the desired CoM. This is due to the bending of legs in Figure. 4.4. Since in the CoM generation, we assume the robot is an LIPM and all the mass is concentrated in the torso, so the desired CoM trajectory we planned is actually the torso trajectory. However, in order to simulate the posture of walking (or any movements that need to lower the torso), the legs are bent forward in reality, which causes the actual CoM to be ahead of the desired CoM. To avoid this, one can reduce the bending of the knee while walking. However, this will reduce the step length of walking due to kinematics constraint. Also having a straight knee while walking is bad for landing of swing foot, since a straight knee cannot absorb any impact during landing.

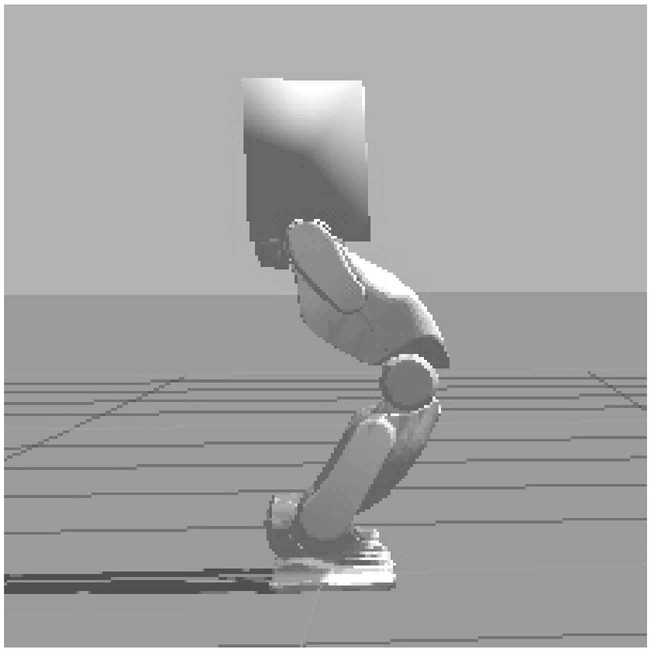


Figure 4.4: Side view of Reem-C lower body in stance

Then the CoM tracking in y direction is simulated:

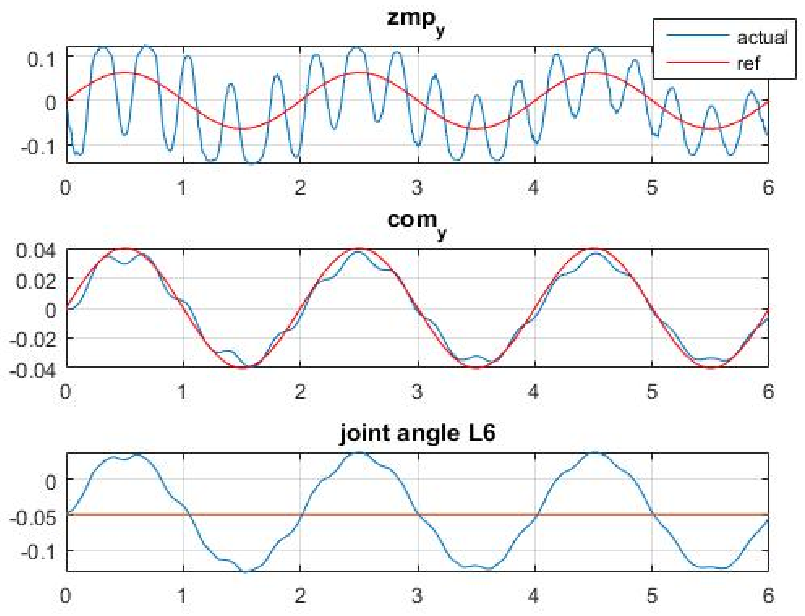


Figure 4.5: Simulation of CoM tracking in y direction:PD control

The results show that the same controller that's working good for x direction is not performing good in y direction. The CoM is roughly tracked but there are too many oscillations.

We are using the same controller for both ankles in y direction. However, the two ankle joints are not sharing the same axis, so if we simply modify the joint angles by the same amount, the two soles won't stays in the same plane anymore, the robot will stand on the rims of soles (Figure. 4.6), which changes the support polygon. The actual ZMP will fall on the boundaries of the soles and this causes the oscillations.

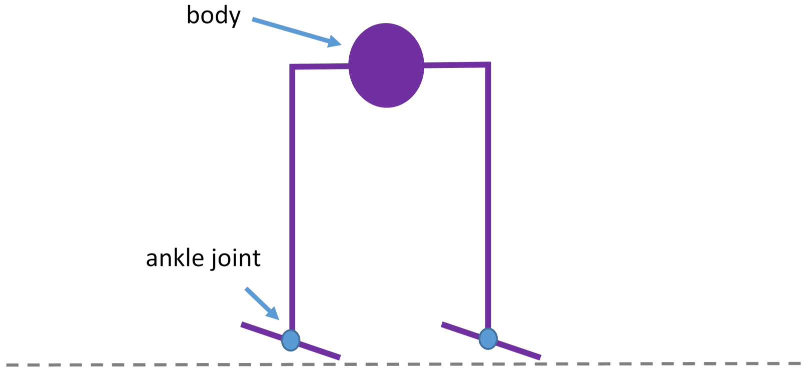


Figure 4.6: Frontal plane: Ankle joints rotated by the same amount

To sum up, this kind of intuitive ankle position controller is not good enough for walking, since ZMP has to be tracked with little error during walking to ensure it to stays inside the support polygon. Next, the WBC controller will be tested for CoM tracking.

4.2 Implementation of WBC Controller

Before implementing the WBC controller to the closed-loop walking simulation, we have to test the controller to see the performance of CoM tracking. A sinusoidal squat motion is tracked in the following simulation. The reference CoM trajectory is such that x and y are kept at constant but z is a sinusoidal function of time:

$$P_{c,ref} = \begin{pmatrix} 0.02 \\ 0 \\ h_0 + A\sin\left(\frac{2\pi}{T}t\right) \end{pmatrix} \tag{4.4}$$

Here $h_0 = 0.53m$ and the period $T = 2s$.

The trajectory of feet also needs to be specified. Both feet are standing still. If we can set the left foot to be stance foot while the right foot to be swing foot, with a constant position and orientation, the reference position and orientation of swing foot are:

$$\begin{aligned} P_{sw,ref} &= \begin{pmatrix} x_{sw,ref} \\ y_{sw,ref} \\ z_{sw,ref} \end{pmatrix} = \begin{pmatrix} x_{sw}(0) \\ y_{sw}(0) \\ z_{sw}(0) \end{pmatrix} \\ R_{sw,ref} &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \end{aligned} \tag{4.5}$$

where $P_{sw,ref}$ and $R_{sw,ref}$ are swing foot reference position and orientation, as used in (3.34). This reference trajectory simply means we want the swing foot to stay at its initial position. And the orientation is constant (here it's not an identity matrix, it depends on how the foot frame is defined in the specific robot model).

Another very important embedded motion is the torso motion. Since the torso has most of the mass and inertia, we want the orientation of torso to be vertical all the time to minimize the total angular momentum. The reference of torso orientation will be:

$$R_{0,ref} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

which is an identity matrix.

Now we have all the world frame references, which are $P_{c,ref}$, $P_{sw,ref}$, $R_{sw,ref}$ and $R_{0,ref}$. To track these world frame references, we need to control at least 12 DoF.

Other than (3.31), now the overall Jacobian including torso motion is:

$$J(q) = \begin{pmatrix} J_c(q) \\ J_{sw}(q) \\ J_{0,\omega}(q) \end{pmatrix} \tag{4.7}$$

where J_c is 3×12 CoM Jacobian, J_{sw} is 6×12 Jacobian and $J_{0,\omega}$ is a 3×12 Jacobian only for orientation of torso.

Based on (3.34) and (3.35) with torso motion added, we have the following controller for numerical implementation:

$$\begin{aligned} q_{ref}^{k+1} &= q^k + \delta q_{ref}^{k}{}^{k} \\ \delta q_{ref}^k &= J^\dagger(q^k) \begin{pmatrix} K_{po}\left(\begin{pmatrix} P_{c,ref} \\ P_{sw,ref} \end{pmatrix} - \begin{pmatrix} P_c \\ P_{sw} \end{pmatrix}\right) \\ -R_{sw,ref}{}^{-1} K_{sw,or} u_{sw} \lambda_{sw} \\ -R_{0,ref}{}^{-1} K_{0,or} u_0 \lambda_0 \end{pmatrix} \delta t \end{aligned} \tag{4.8}$$

in which, the Jacobian J is now for motion of CoM, swing foot and torso. u_{sw}, λ_{sw}, u_0 and λ_0 are axes and angles of rotation as defined in Section 3.2.3, for swing foot and torso orientations. K_{po}, $K_{sw,or}$ and $K_{0,or}$ are controller gain matrices.

By implementing numerically following (4.8) and choosing appropriate gain matrices K_{po}, $K_{sw,or}$ and $K_{0,or}$ as shown in (A.1), we have the following tracking result for the squat motion:

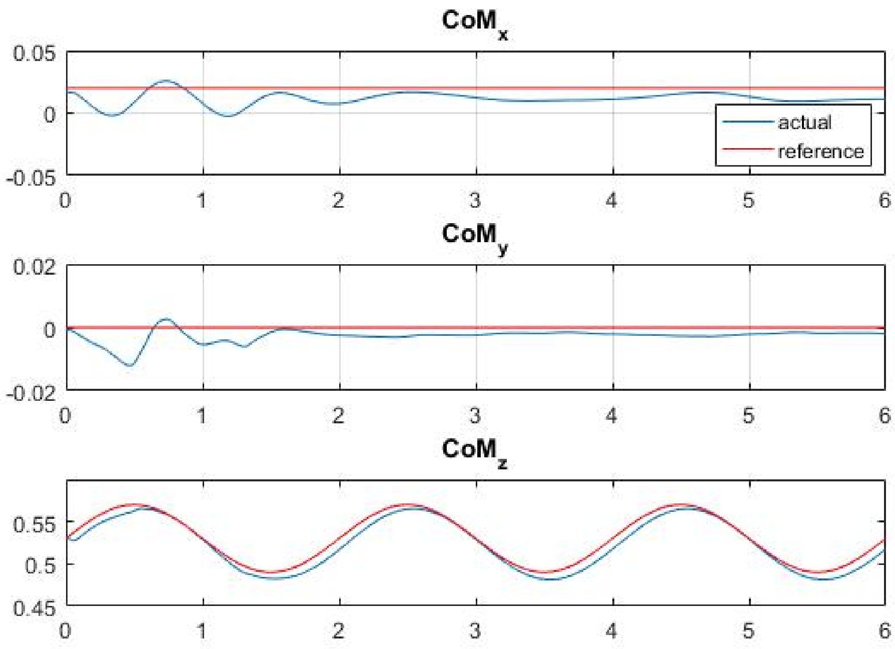


Figure 4.7: Simulation for sinusoidal CoM tracking in z direction

The tracking performance is good in z direction, with a slight delay. In x and y direction, there's a little oscillation at the beginning. Simulations for tracking sinusoidal reference in x and y direction are also made, the results are similar with z direction tracking, which validated the efficiency of WBC controller.

4.3 Walking Simulation

In this section, we will close the loop and integrate all the parts: ZMP/CoM replanning, CoM/ZMP stabilization and tracking. The control scheme used in the walking control is shown in Figure 4.8:

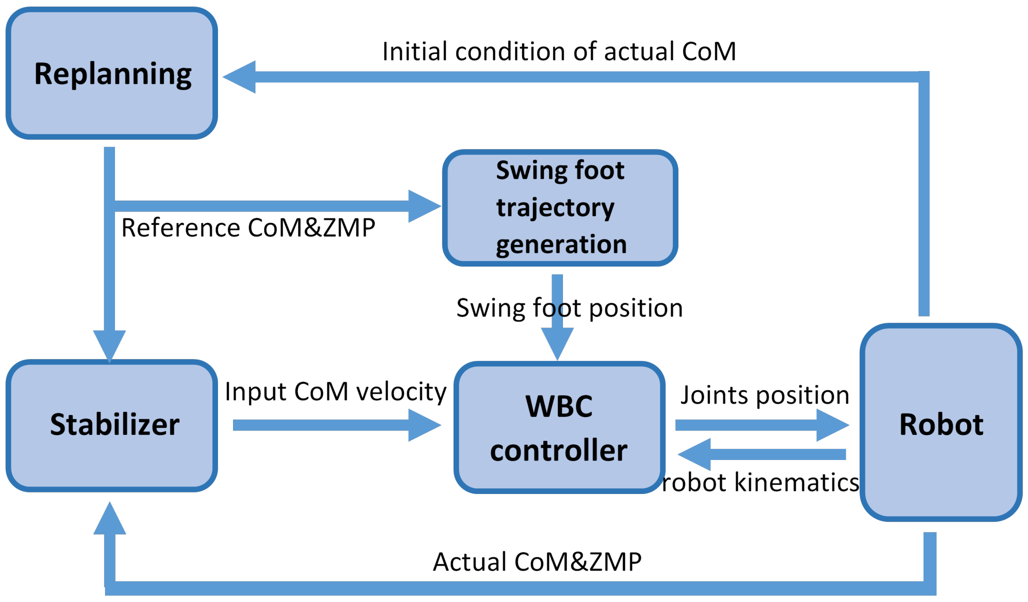


Figure 4.8: Control Diagram

The walking pattern is replanned at the beginning of each step, based on the actual initial condition of CoM. The swing foot trajectory is generated using replanned ZMP to ensure the ZMP will stay in the support polygon for the next step. These two procedures will be discussed in Section 4.3.1. Then the replanned CoM and ZMP are sent to the high level stabilizer, the stabilizer read the actual CoM and ZMP, and send a CoM velocity that can stabilize both CoM and ZMP. This CoM velocity along with the swing foot trajectory then is sent to WBC controller, which has been derived in Section 4.2. WBC controller will directly send joint position command to the robot such that the robot will track the CoM velocity and swing foot trajectory.

4.3.1 Implementation

The detailed procedures of the scheme Figure 4.8 are discussed as following.

4.3.1.1 CoM and ZMP Replanning

Input:
- Actual initial Condition of $P_c(t_0)$
- desired gait parameter: step period T, double support ratio r (which is the ratio of double support duration to step period), single support length d_{ss}

Output:
- step length Δx,Δy
- CoM and ZMP trajectory for one step, $P_{c,ref}$ and $x_{zmp,ref}$, $y_{zmp,ref}$

The robot is tracking the planned CoM and ZMP when walking. Therefore, it's sure that there will be tracking error. The tracking error will accumulate and result in falling. So the replanning takes place at the beginning of each step. Instead of using the planned CoM from the previous step, it uses the actual CoM initial condition for planning. This will make sure the replanned CoM initial condition is consistent with actual initial condition.

1. The procedures in Section 2.2.3 Case II is followed. The ZMP is parameterized as a linear/cubic spline, starting with linear splines for single support phase, followed by cubic spline for double support phase. Here we assign the double support time to total step period ratio and the step period. So the only unknown parameter is the step length Δx and Δy in x and y direction.

   Parameters assigned in this step:

   Table 4.1: Gait Parameters

Parameter	Notation
step period	T
double support ratio	r
single support length	d_{ss}

2. Plug the actual initial condition of ZMP and the unknowns Δx and Δy into the ZMP spline function, we will have a ZMP spline function with coefficients in (2.32), which are linear functions w.r.t Δx and Δy. Then using CoM initial condition calculated from sensors along with the ZMP spline function (with Δx and Δy in it), we can use the boundedness condition (2.27) to solve for Δx and Δy in x and y direction such that the CoM is convergent given the parameterized ZMP as input.

   Parameters assigned in this step:

   Table 4.2: Replanning Parameters

Parameter	Notation
ZMP initial condition	$x_{zmp}(t_0)$,$y_{zmp}(t_0)$
CoM initial position	$x_c(t_0)$,$y_c(t_0)$
CoM initial velocity	$\dot{x}_c(t_0)$,$\dot{y}_c(t_0)$

   The boundedness condition in 2.27 becomes:

   $$x_u^\star(t_0) = x_u^\star(t_0, T, r, d_{ss}, x_{zmp}(t_0), \Delta x) = x_c(t_0) + \frac{1}{\omega_0}\dot{x}_c(t_0) \tag{4.9}$$

   where $x_u^\star(t_0, T, r, d_{ss}, x_{zmp}(0), \Delta x)$ is an known linear function of Δx, which can be calculated by going through (2.25) to (2.30), with Δx as unknown parameter instead of T. Plug in all the known parameters, Δx can be solved. Solving of Δy follows the same procedures.

3. Plug the Δx and Δy into the parameterized ZMP, we will have the reference ZMP trajectory for the step. Reference CoM trajectory can be solved by plugging the ZMP trajectory into dynamics equation of LIPM.

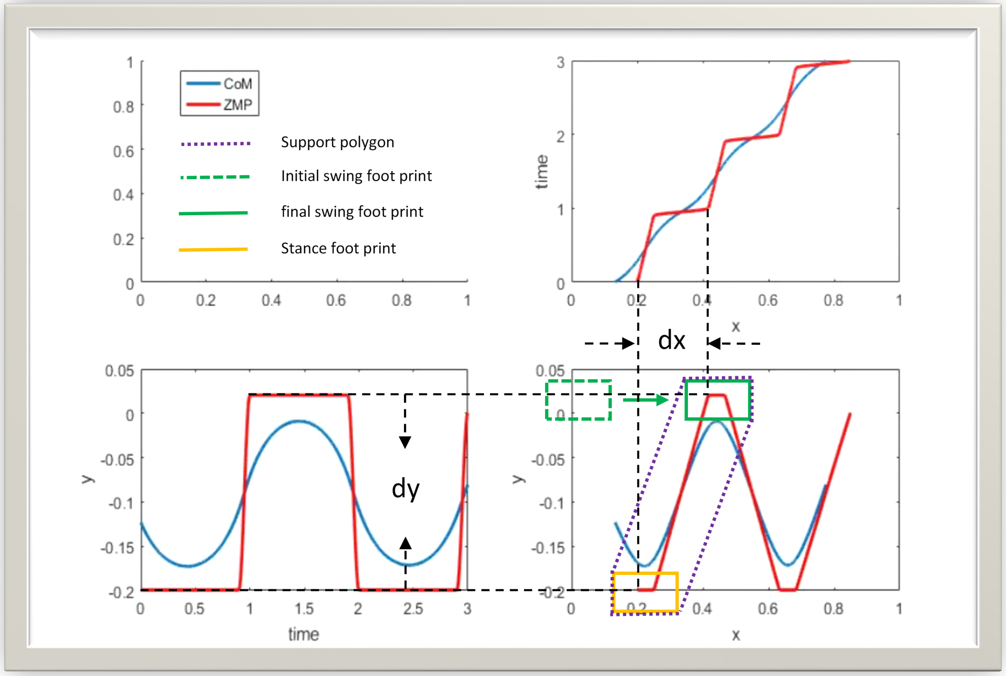


Figure 4.9: Replanning for 3 steps: the ZMP and CoM trajectories are planned separately for x (top right) and y (bottom left) direction using the boundedness condition and parameterized ZMP. Combining the ZMP and CoM in both x and y direction, we can draw the trajectory of ZMP and CoM in the ground plane (bottom right). The footprint profile (green and yellow lines) are designed such that they will form a support polygon (purple line) that can always contain the ZMP inside.

Figure 4.9 shows the relation of replanned ZMP and the foot prints. Here, based on the CoM initial condition, the step lengths in x and y direction Δx and Δy are solved, then 3 steps are generated though only the first step will be executed. The replanned ZMP provides information for swing foot trajectory generation, which is the next step.

4.3.1.2 Swing Foot Trajectory Generation

Input:
- Replanned ZMP trajectory $x_{zmp,ref}$, $y_{zmp,ref}$
- initial condition of swing foot $P_{sw}(t_0)$

Output:
- swing foot trajectory for one step $P_{sw,ref}$

The swing foot trajectory is generated using a 5th order spline. The reason for choosing 5th order is that we have 6 boundary conditions to meet: initial and final position, velocity and acceleration. Including the acceleration as boundary condition is important since it's directly related to the force applied on the foot (including the ground reaction force). The initial conditions are the actual initial condition of the swing foot (ideally the initial velocity and acceleration are zero since swing foot is not moving at the beginning), we need to assign final condition for the foot in order to satisfy the support polygon criteria and reduce the impact.

In the x,y plane, the final position at the end of swing phase is assigned to be the final position of ZMP that replanned from previous part. This will make sure the ZMP will stay inside the new support polygon during stance phase, see Figure 4.9. The final velocity of swing foot are set to zero since it will stand still once contact with the ground. The final acceleration is also assigned to be zero, which physically means the contact force is zero at the instant of contact. This will minimize the effect of impact.

Initial condition and final condition for 5th order swing foot trajectory spline:

Table 4.3: Swing foot Trajectory Spline Parameters in x,y Direction

time	t_0	time	$t_0 + T(1-r)$
x direction			
position	$x_{sw}(t_0)$	position	$x_{zmp}(t_0 + T)$
velocity	0	velocity	0
acceleration	0	acceleration	0
y direction			
position	$y_{sw}(t_0)$	position	$y_{zmp}(t_0 + T)$
velocity	0	velocity	0
acceleration	0	acceleration	0

Here x_{zmp} and y_{zmp} are the replanned ZMP trajectory.

The foot has to detach from the ground in order to move. So for the trajectory in z direction, the swing foot have to lift up and come back to the ground during swing phase and stays still at stance phase, a similar 5th order spline can be used for swing phase:

Table 4.4: Swing foot Trajectory Spline Parameters in z Direction

time	t_0	time	$t_0 + T(1-r)/2$	time	$t_0 + T(1-r)$
position	$z_{sw}(t_0)$	position	h_{sw}	position	0
velocity	0	velocity	0	velocity	0
acceleration	0	acceleration	0	acceleration	0

The swing foot starts at it's initial z position, lifted to a predefined height h_{sw} at the middle of swing phase, then lowers to 0 height at the end of swing phase. So h_{sw} is the highest point the foot can reach, so at this point the velocity should be 0. h_{sw} is chosen such that the swing motion satisfies the kinematics constraints of legs and the posture looks natural. If h_{sw} is too small, the swing foot is likely to contact the ground and slide during swing phase. If h_{sw} is too large, the CoM of swing leg is lifted up too much so the torso has to be lowered to keep the overall CoM at constant height, which causes an unexpected posture. Here we choose $h_{sw} = 5cm$, which is a trade off of these two effects. In stance phase, the foot's z position is kept at 0.

Since we add an intermediate point in the middle of swing phase for z direction, we need 2 pieces of splines for swing phase, the boundary conditions are given in Table 4.4.

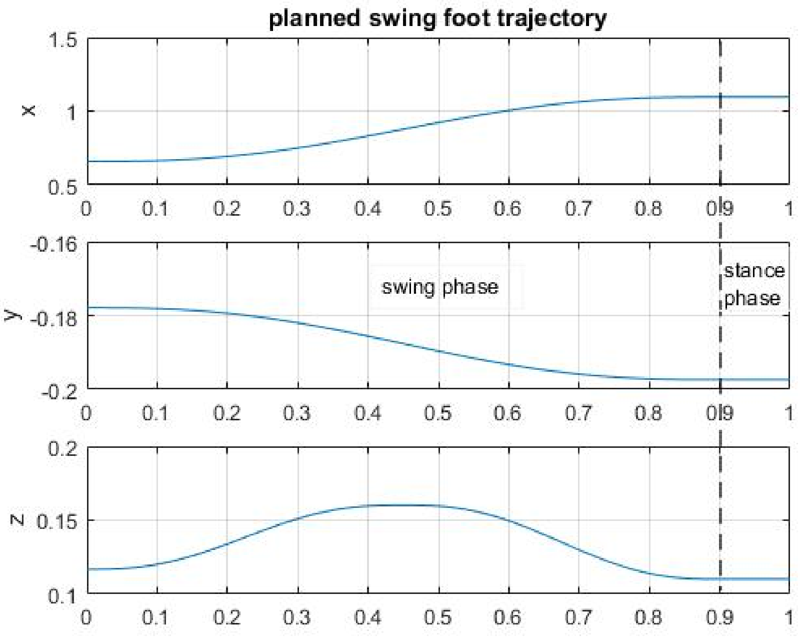


Figure 4.10: Splined swing foot trajectory: $T = 1s$, $r = 0.1$

Figure 4.10 shows an example of generated swing foot trajectory for one step. In x direction, the swing foot moves forward by around $0.4m$ in swing phase. It should be noted that in y direction, the swing foot also moves $2cm$ in swing phase in order to form a support polygon that can contain the planned ZMP when landing. In other words, the distances that the swing foot moves in x and y direction during swing phase depend on the planned ZMP position at the time of landing.

Since landing position of the swing foot depends on the planned ZMP position, if the planned ZMP is drifted to the same side for each step, the robot will not walk in a straight line but tend to lean to this side more and more. This is actually happening in simulation since the ZMP and CoM are replanned at the beginning of each step based on the actual ZMP and CoM initial condition. That is to say, the robot will forget the trajectory and tracking error it has walked with and plan the next step based on the current initial condition. For example, let's consider two successive steps. If we start the first step with the left foot, then for this step, the stance foot is right foot and assume it's position is at origin. At the end of the first step, there's a tracking error for ZMP, $\delta y_{zmp}(T)$, where T is step period. The second step will switch the stance foot so left foot will be the stance foot and the right foot will be swing foot. At the beginning of the second step, the CoM and ZMP error will be reset to zero, CoM and ZMP trajectories are replanned using the CoM and ZMP final condition of the first step. Since the final condition of ZMP has an error of $\delta y_{zmp}(T)$ at the end of first step, the replanned trajectory for the second step will have an offset of $\delta y_{zmp}(T)$. Therefore the ZMP position at the time when right foot land on ground is shifted by $\delta y_{zmp}(T)$, which means that the y position of right foot has to move from origin to $\delta y_{zmp}(T)$ during the swing phase of second step.

The functionalities of replanning part and swing foot trajectory generation part is summarized as follow: The replanning algorithm can make sure that the tracking errors of CoM and ZMP will be kept small for each step since it prevent the errors from accumulating by resetting them. The swing foot trajectory generator can force the ZMP to stay inside the support polygon all the time. So these two modules together will guarantee the stable walking of robot. However, global trajectory tracking can not be guaranteed, since the replanning algorithm forgets the previous trajectories and starts over a new planning and tracking task for the next step, so there will be drifting of actual global ZMP and CoM trajectory. This will be also discussed in Section 4.3.2.2 and a possible solution is discussed.

4.3.1.3 Stabilizer

Input:
- Actual CoM's x,y position x_c, y_c and ZMP position x_{zmp}, y_{zmp}
- reference CoM's x,y position $x_{c,ref}$, $y_{c,ref}$ and ZMP $x_{zmp,ref}$, $y_{zmp,ref}$

Output:
- CoM velocity that need to tracked $\dot{x}_c$, $\dot{y}_c$

The stabilizer in [19] is implemented:

$$\begin{aligned} \dot{x}_c &= \dot{x}_{c,ref} - k_x e_{xc} + k_{zx} e_{zx} \\ \dot{y}_c &= \dot{y}_{c,ref} - k_y e_{yc} + k_{zy} e_{zy} \end{aligned} \tag{4.10}$$

where $\dot{x}_c$, $\dot{y}_c$ are CoM velocities that sent to WBC controller to track. $\dot{x}_{c,ref}$, $\dot{y}_{c,ref}$ are replanned CoM velocity, or desired CoM velocity. k are the controller gains. e_{xc}, e_{yc}, e_{zx} and e_{zy} are CoM errors and ZMP errors respect to replanned CoM and ZMP:

$$\begin{aligned} e_{xc} &= x_{c,ref} - x_c \\ e_{yc} &= y_{c,ref} - y_c \\ e_{zx} &= x_{zmp,ref} - x_{zmp} \\ e_{zy} &= y_{zmp,ref} - y_{zmp} \end{aligned} \tag{4.11}$$

The velocities $\dot{x}_c$, $\dot{y}_c$ in (4.10) will be the reference CoM velocities for the robot to track.

4.3.1.4 Whole Body Cooperation Control

Input:
- Input CoM velocities from stabilizer $\dot{x}_c$,$\dot{y}_c$
- Generated swing foot position $P_{sw,ref}$
- Actual joint positions q and link positions, orientations

Output:
- reference joint positions q_{ref}

This controller will cooperate each individual joint such that the robot will track both the reference CoM and swing foot motion, as well as the torso orientation.

For CoM tracking, x_c and y_c are controlled directly by the stabilizer in previous procedure to stabilize CoM and ZMP. However, the other target coordinates are tracked using the controller (4.8). These target coordinates include CoM position in z direction, z_c, position and orientation of swing foot P_{sw}, R_{sw}, and torso orientation R_0. So the overall controller modified from (4.8) is:

$$\begin{aligned} q_{ref}^{k+1} &= q^k + \delta {q_{ref}^k}^k \qquad (4.12) \\ \delta q_{ref}^k &= J^{\dagger}(q^k) \begin{pmatrix} \dot{x}_c \\ \dot{y}_c \\ k_{zc}(z_{c,ref} - z_c) \\ K_{sw,po}(P_{sw,ref} - P_{sw}) \\ -{R_{sw,ref}}^{-1} K_{sw,or} u_{sw} \lambda_{sw} \\ -{R_{0,ref}}^{-1} K_{0,or} u_0 \lambda_0 \end{pmatrix} \delta t \end{aligned}$$

in which, the only difference from controller (4.8) is that the control inputs of first two states, are replaced by the control inputs given by the stabilizer $\dot{x}_c$ and $\dot{y}_c$. The gain and gain matrix k_{zc} and $K_{sw,po}$ are decomposed from the bigger diagonal gain matrix K_{po} from (4.8).

1. Plug the input CoM velocities calculated by the stabilizer for x and y direction into (4.12). Read the actual swing foot position P_{sw} and CoM position in z direction x_z, plug into (4.12).

2. Read the actual swing foot and torso orientation R_{sw} and R_0. Calculate the rotation axis and angle for swing foot and torso, as defined in Section 3.2.3, u_{sw}, u_0, λ_{sw} and λ_0. Then plug these values into (4.12).

3. Plug all the reference positions and orientations into (4.12). The reference for CoM position in z direction is chosen to be constant during each step, the value is the initial value of CoM height of each step. The swing foot reference position is given by the swing foot trajectory generation part. And the reference rotation matrix of swing foot and

torso are the same as (4.5) and (4.6) since we still want the torso and swing foot to keep horizontal at all time.

4. Calculate the CoM Jacobian and embedded motion Jacobians (in this case, swing foot Jacobian and torso Jacobian), from link and joint kinematics. We will have J_c, J_{sw} and J_0 as in (4.13). Calculate the pseudo inverse (in this case it's simply inverse since it's 12×12 matrix) of J using (3.33).

$$J(q^k) = \begin{pmatrix} J_c(q^k) \\ J_{sw}(q^k) \\ J_0(q^k) \end{pmatrix} \tag{4.13}$$

5. Up to this point, we have plugged all the values of variables in controller (4.12). We can calculate the joint angle increments δq^k_{ref}. Adding up the current readings of joint angles q_k and the increments δq^k_{ref}, finally we can get the reference joint angles for next step, q^{k+1}_{ref}. The reference joint angles are then tracked by PID controller of each joint.

4.3.1.5 Implementation Notes

In addition to the previous main steps of implementation, some minor implementation details should be noted.

WBC control is based on the assumption that the stance foot is fixed on the ground. So if the stance foot detached from the ground due to tracking error or disturbances, the WBC calculation won't be valid anymore. Therefore, an ankle joint position compensator is added to the stance ankle joints. This compensator will compensate the ankle joints' angles such that the orientation of the stance foot still keeps horizontal, therefore the stance foot will stay in good contact with the ground at all time. This is actually a limitation of WBC controller, and will be discussed in Section 5.2.1.

The compensator for stance leg's ankle joints has the following form:

$$\begin{aligned} q'_{5,ref} &= q_{5,ref} + q_{5,comp} \\ q'_{6,ref} &= q_{6,ref} + q_{6,comp} \end{aligned} \tag{4.14}$$

where the subscript 5 and 6 correspond to the two ankle joints as shown by $J5$

and $J6$ in Figure 3.4. $q_{5,ref}$ and $q_{6,ref}$ are the original reference joint angles calculated by the WBC controller. $q'_{5,ref}$ and $q'_{6,ref}$ are the compensated reference joint angles that the joints are actually tracking. $q_{5,comp}$ and $q_{6,comp}$ are the amounts of compensation needed to bring the deviated orientation of stance foot to the horizontal orientation. They can be simply calculated using the euler angles needed to transform from the actual orientation to the horizontal orientation.

The PID gains of ankle joints of swing foot are set to 0 during landing to reduce impact. This is due to the fact that the robot is position controlled and lack of compliance, which will be discussed in Section 5.2.2.

4.3.2 Simulation results and discussion

In this section, simulation results of walking are discussed. In order to evaluate the performance of each module in the whole control loop, several different walking simulations are made. This section will focus on the performance of stabilizer, so walking simulation with and without stabilizer are made to be compared. More evaluations will be discussed in the next chapter.

4.3.2.1 Walking Without Stabilizer

First, to show the effect of stabilizer in (4.10), simulation without the stabilizer is made. The gait paratemters are chosen to be:

Table 4.5: Simulation Gait Parameters

Parameter	Notation	value
step period	T	1s
double support ratio	r	0.1
single support length	d_{ss}	0.05m

The step period is chosen to be 1 second, which is very close to slow walking of human. During double support, the two soles are not necessarily in the same plane due to tracking errors. To reduce this effect, the double support ratio is chosen to be 0.1, which is a little smaller than human's double support ratio. Since the foot length is 0.15m, we choose the single support length to be 0.05m to ensure the ZMP stays in the stance sole during swing phase.

According to the replanning procedures, the initial desired ZMP position need to be assigned. Here we assign the desired ZMP position with the actual stance foot position, so that during the following step, the desired ZMP will stay inside the support polygon (which is the area enclosed by stance foot). Also, in simulation, we only assign the desired CoM initial position with actual CoM initial position, but not the velocity. The desired initial CoM velocity will be left as a variable and will be chosen such that the solved step length is physically possible. As a result, the replanned CoM trajectory will match the actual CoM initial position but not the velocity, however, it can be still tracked by the WBC controller. In the simulation, we chose the following initial conditions:

$$\begin{aligned}
x_{zmp,ref}(0) &= x_{st}(0) \qquad (4.15)\\
y_{zmp,ref}(0) &= y_{st}(0)\\
x_{c,ref}(0) &= x_c(0)\\
y_{c,ref}(0) &= y_c(0)\\
\dot{x}_{c,ref} &= (x_{zmp,ref}(0) - x_{c,ref}(0) \pm 0.0175)w_0\\
\dot{y}_{c,ref} &= (y_{zmp,ref}(0) - y_{c,ref}(0) \pm 0.0043)w_0\\
z_{c,ref}(t) &= z_c(0)
\end{aligned}$$

where the initial condition for ZMP and CoM positions are actual values calculated from sensors at the beginning of the step. The CoM initial velocities can be chosen arbitrarily since we only need the reference CoM positions to match with actual CoM positions of the robot. However, the value of CoM initial velocities will affect the solution of step lengths Δx and Δy linearly. By choosing the initial velocities in (4.15), we are able to have a solution that's feasible for the robot.

By choosing the above initial desired initial condition $\dot{x}_{c,ref}$ and $\dot{y}_{c,ref}$, the solved step length Δx and Δy are both approximately $0.2m$. The sign $\pm$ simply denote the different cases when different foot (left or right) is considered to be stance foot. Also, the desired z position of CoM is constant during the step, whose value is just the initial value of z_c.

Controller (4.8) is used, since there's no stabilizer. The gain matrices are listed in (A.1). The gain matrices are diagonal, so each coordinate is actually decoupled and controlled individually. This makes it easier to tune the gains.

Here the gain values are chosen such that the tracking is fast enough and the overshoot is acceptable.

The PID gains for each joint's position control can be found in Table A.1 in Appendix A.0.2, which is given designed by Pal Robotics.

The robot successfully walked for 6 steps then fell down. The following result shows the tracking of CoM trajectory.

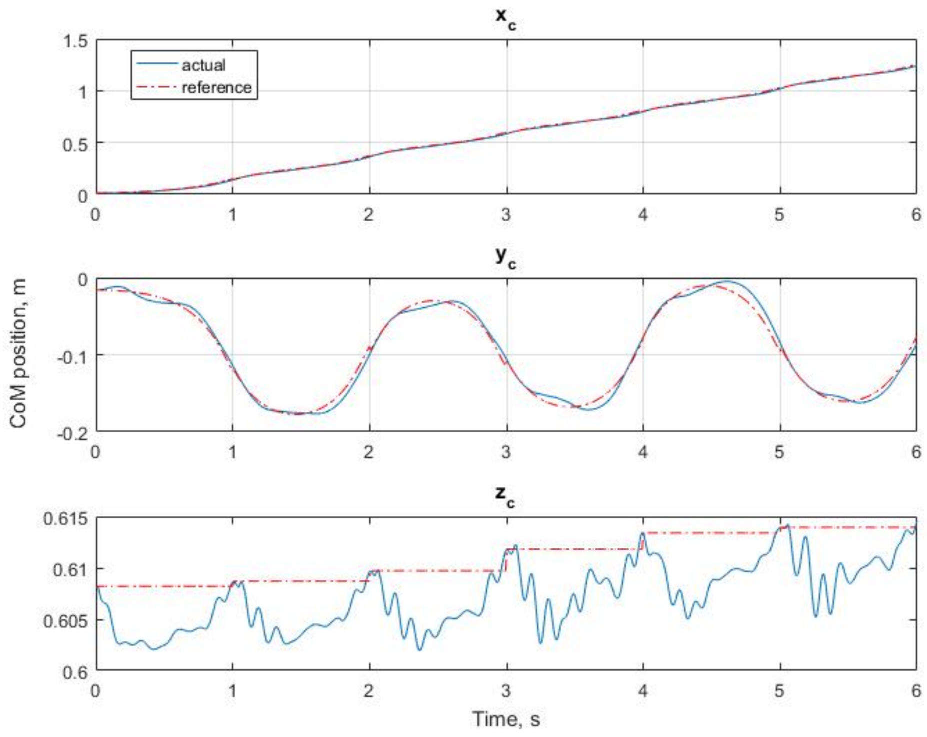


Figure 4.11: Walking: CoM tracking without stabilizer

As we can see from Figure 4.11, generally, the CoM in x and y direction is tracked very well. The tracking in z direction is worse compared with x and y direction because the WBC controller gain for z direction is smaller. The reason is that the tracking in x and y direction has higher priority since it's related to ZMP tracking, therefore related stability of walking.

Also, it's shown that the reference CoM trajectories for x and y direction are not continuous, due to replanning. A closer look for the discontinuity of reference CoM in y direction is shown in Figure 5.3 in Section 5.1. At the beginning of each step (here the step period is $T = 1s$), the CoM is replanned such that the replanned CoM start from the actual CoM initial position, but the initial velocities are chosen according to (4.15). The reference for z direction is not continuous since it used the actual initial CoM height as reference. This also caused the problem that the actual CoM height for each step is increasing. There's always a positive error at the end of each step for CoM height, and this final value will become the reference for next step, so the CoM height is increasing slowly.

While the CoM is tracked, the ZMP is not guaranteed to be tracked, because the reference ZMP is replanned using LIPM dynamics, which is obviously different from the actual robot dynamics. The ZMP error will accumulate and eventually the actual ZMP moves out of support polygon. This explained why the robot only walked for steps. To track the ZMP at the same time, the stabilizer should be used.

4.3.2.2 Walking With Stabilizer

Using the same gait parameters, we add the stabilizer in (4.10) to the loop. This stabilizer is based on LIPM. By choosing proper gains, the stability is proved in [19], for any given LIPM. Here, stabilizer gains are chosen based on the parameters of LIPM of Reem-C Robot, using the tuning method in [19]:

Table 4.6: Stabilizer Gains

k_x	150	k_{zx}	5
k_y	100	k_{zy}	10

As described in the Section 4.3.1.4, the reference for x_c and y_c are given by the stabilizer independently, while the other target space coordinates are still controlled by WBC controller, with the gain matrices in (4.12):

$$\begin{aligned} K_{zc} &= 75 \\ K_{sw,po} &= diag(140, 140, 140) \end{aligned} \tag{4.16}$$

which are the decomposed diagonal matrices of previous gain matrix K_{po}, without the first two element for x_c and y_c.

The way to assign initial values for desired CoM and ZMP is the same as the previous section, 4.3.2.1.

Since the actual ZMP position is calculated using force sensor data, which is noisy, the ZMP position is filtered using a second order low pass filter with cutoff frequency around $5hz$:

$$\frac{0.0201s^2 + 0.0402s + 0.0201}{s^2 - 1.5610s + 0.6414} \tag{4.17}$$

With the help of stabilizer, the robot walked for 30 steps without falling in the simulation.

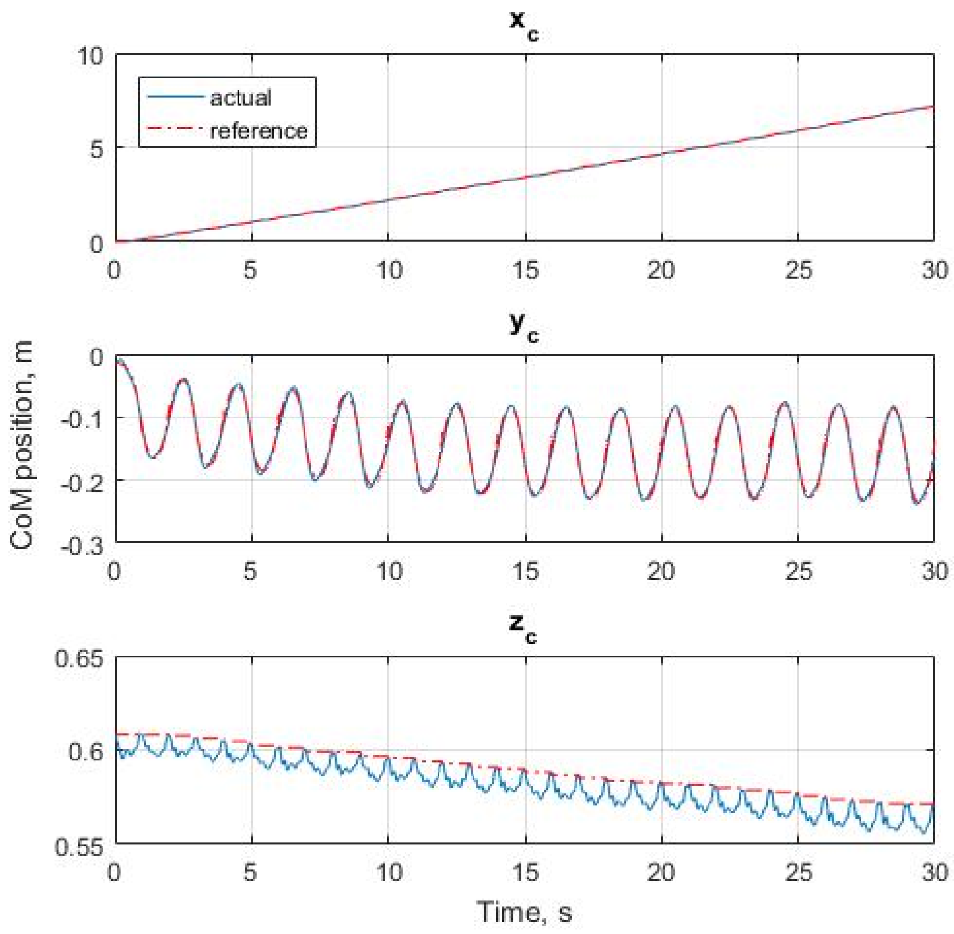


Figure 4.12: Walking: CoM tracking with stabilizer

Figure 4.12 shows similar behavior as the simulation without stabilizer in Figure 4.11. But this time the robot can walk 30 steps without falling. It should be noted that the CoM in y direction is drifting slowly. This is caused by the error of tracking and replanning algorithm. That is, if there's tracking error of CoM at the end of the step, the replanning algorithm will replan the CoM of next step based on this actual CoM final value. So if there's a negative tracking error at the end of each step. The initial condition of each step is actually drifting to the right, and the replanned CoM trajectory of each step is also drifted to the right. To improve this problem, some calibration of the robot's world frame position can be added in the future work. For example, if the actual CoM trajectory is detected to be drifted to the right, some calibration can be added to the replanned CoM trajectory of next step such that the CoM will return to the center by the end of next step.

Similarly with Figure 4.11, the CoM position in z direction is also drifting for the same reason with the drifting in Figure 4.11. In this case, the tracking error is negative at the end of each step. To avoid the drifting of CoM height, a constant reference CoM in z direction is used, instead of taking the initial CoM z position of each step as reference for that step. The result is shown in Figure 4.13. The robot walked for 16 steps and falls, which is worse than the 30 steps simulated using variable reference height in Figure 4.12. The possible reason for the falling is that, if the reference of CoM height is constant, the error of CoM height will be accumulated and won't be reset to zero at beginning of each step. So when there's large disturbance, it's very likely that the CoM error caused by this disturbance will be accumulated and divergent. Also, since the reference for CoM height is constant, the value of this reference has to be carefully chosen. In Figure 4.13, the reference is chosen to be the CoM height of it's initial pose. Better choice of reference CoM height may be existed, though this work will not focus on choosing reference CoM height.

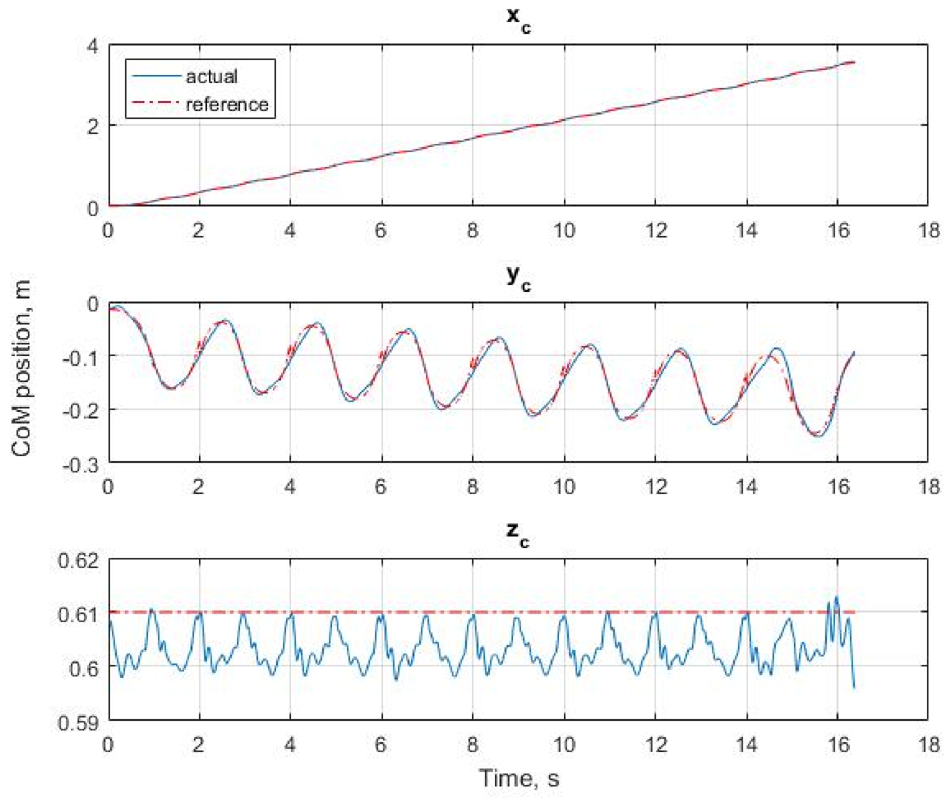


Figure 4.13: Walking: CoM tracking with stabilizer (constant $z_{c,ref}$)

Back to the simulation of Figure 4.12, with the stabilizer, the critical difference is that the ZMP is also tracked as shown in Figure 4.14.

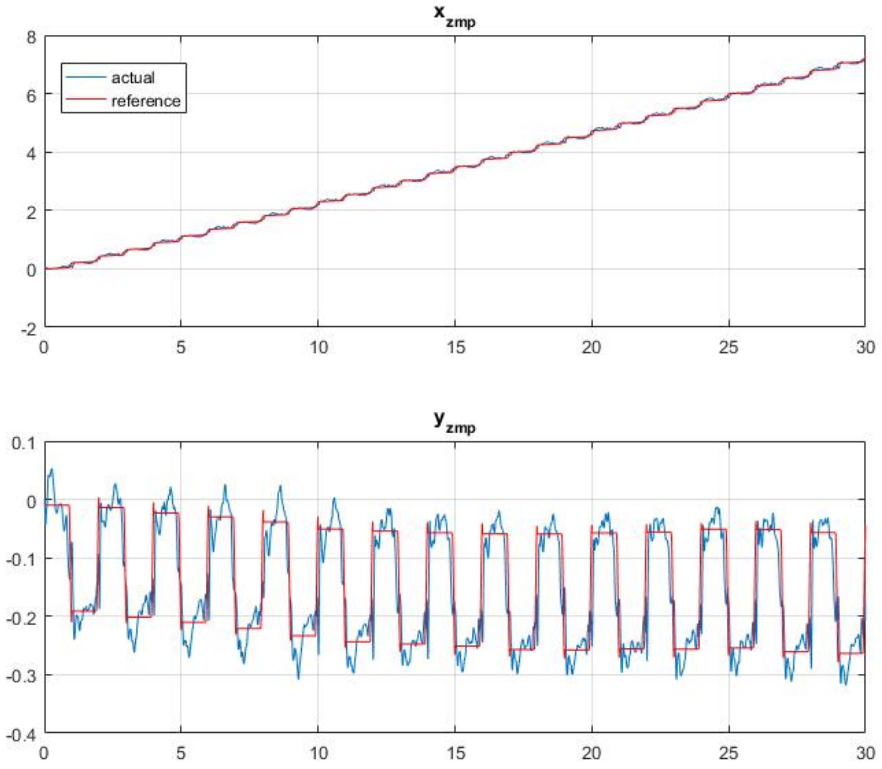


Figure 4.14: Walking: ZMP tracking with stabilizer

Figure 4.14 shows the tracking of ZMP trajectory. We see that although there's still noises and oscillations in actual ZMP, the ZMP error is small all the time, which ensures that it will always stay inside the support polygon (since the support polygon is determined by swing foot position which is designed to contain the desired ZMP).

CHAPTER 5

EVALUATION OF STABILIZERS

Two lower level stabilizers are discussed in detail in Chapter 3. The first one, ankle position controller is an intuitive controller modified from the one developed 15 years ago in [27] to track CoM or ZMP. The simulation results in Figure 4.1, Figure 4.3 and Figure 4.5 in Section 4.1 shows that it cannot track the desired ZMP and CoM trajectory very well, therefore it's not implemented in walking simulation in this work.

The second lower level stabilizer, WBC controller is the most broadly used controller among position controlled robots. And from the simulation of this thesis, the performance of this controller is validated.

In this Chapter, the advantage of the hierarchical control in Figure 4.8 implemented in Section 4.3 will be discussed. Then the challenge and difficulties of this method will be discussed.

5.1 Evaluation of Replanning Algorithm

In order to show the performance of replanning algorithm, a simulation without the replanning procedure is made for comparison. In this simulation, the CoM and ZMP trajectories are generated for n-steps, using actual initial condition, without replanning at the beginning of each step. In other words, it only planned once. Then the trajectories are followed using higher level stabilizer in Section 4.10 and WBC controller.

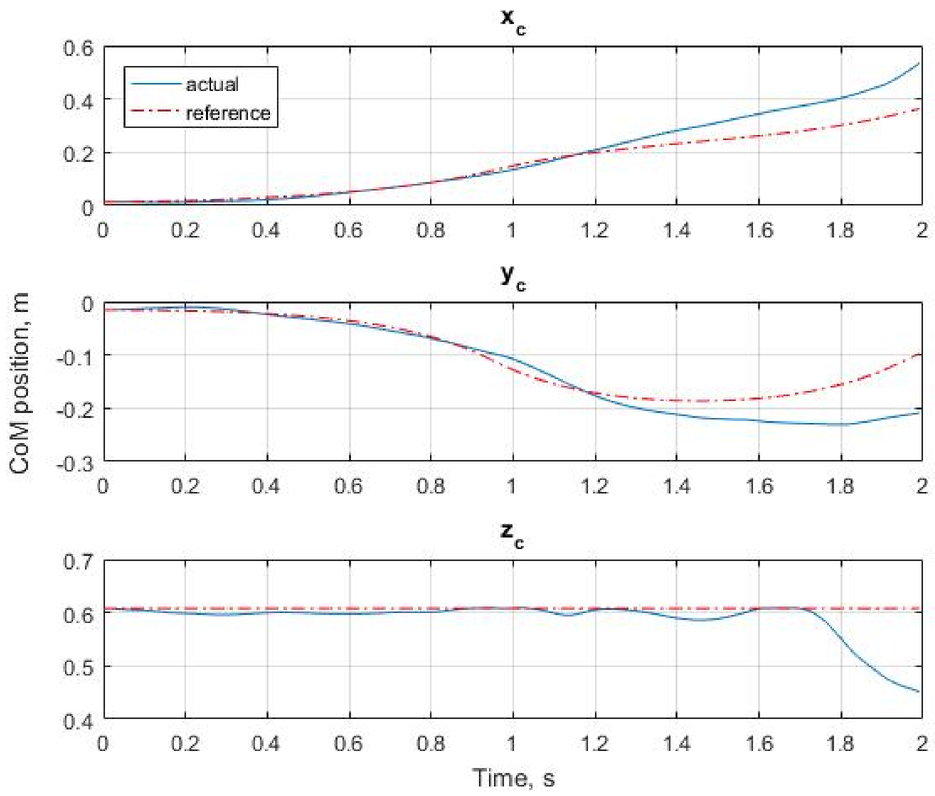


Figure 5.1: Walking Without Replanning: CoM

The reference trajectories in Figure 5.1 and Figure 5.2 are for two steps with step period 1 second. The robot steps its right foot for the first step from rest, then followed by the second step with the left foot as swing foot. So in y direction, the ZMP stays constant during the first step to stay in the sole of stance (left) foot then shifts to right foot at the end of the first step, then stays constant to stay in the sole of stance (right) foot. The corresponding CoM slowly approaches right foot at the beginning of the first step then approaches faster in the double support phase since the ZMP is shifting fast in double support. At the beginning of the second step, the acceleration of CoM if positive, according (2.4), CoM has to move to the left in order to keep ZMP constant.

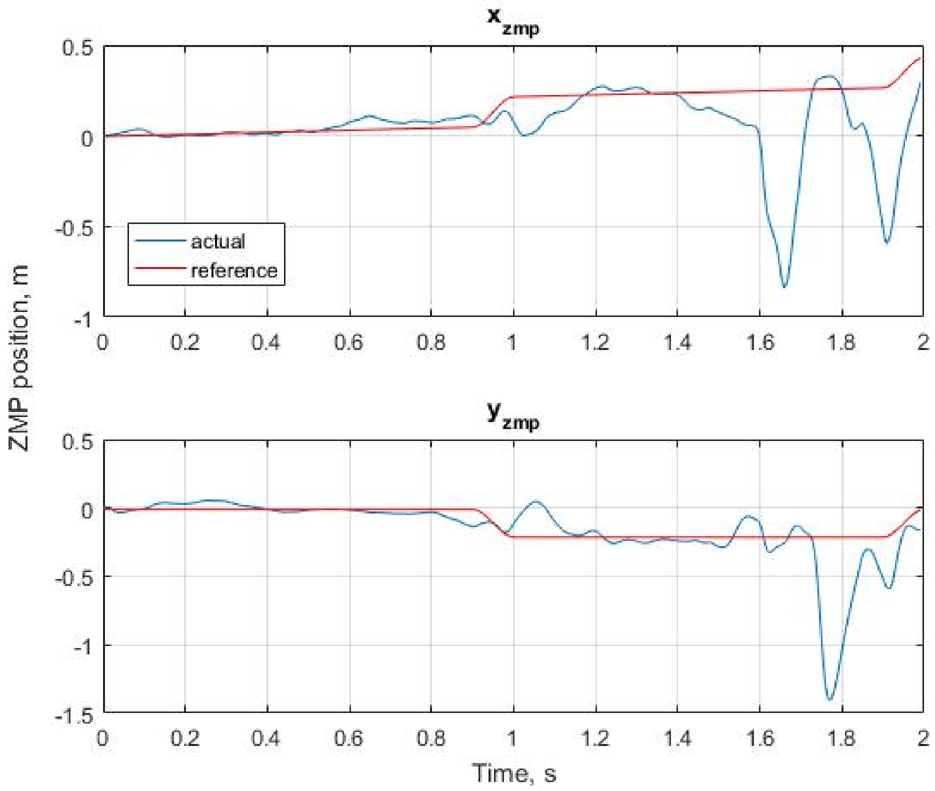


Figure 5.2: Walking Without Replanning: ZMP

As shown in Figure 5.1 and Figure 5.2, the robot tips over during the second step. If we take a close look at the CoM at $t = 1s$, where the second step start, we see that the actual CoM has large accumulated error respect to the reference, since the reference CoM is not replanned to match the initial CoM condition of the second step. This large CoM error directly results in a large error in ZMP tracking, which causes large overshoot and oscillations. At some point, the oscillated ZMP moves out of the support polygon, the robot tips over.

If we take a close look at the CoM tracking with replanning in Figure 4.12, for example, for y_c at $t = 16s$, where the replanning process takes place, we see that the replanning (reference) CoM is forced to match the actual CoM initial condition, so the tracking error is 0 at this moment. Starting with 0 error, the WBC controller is able to keep the error small for the rest of the time.

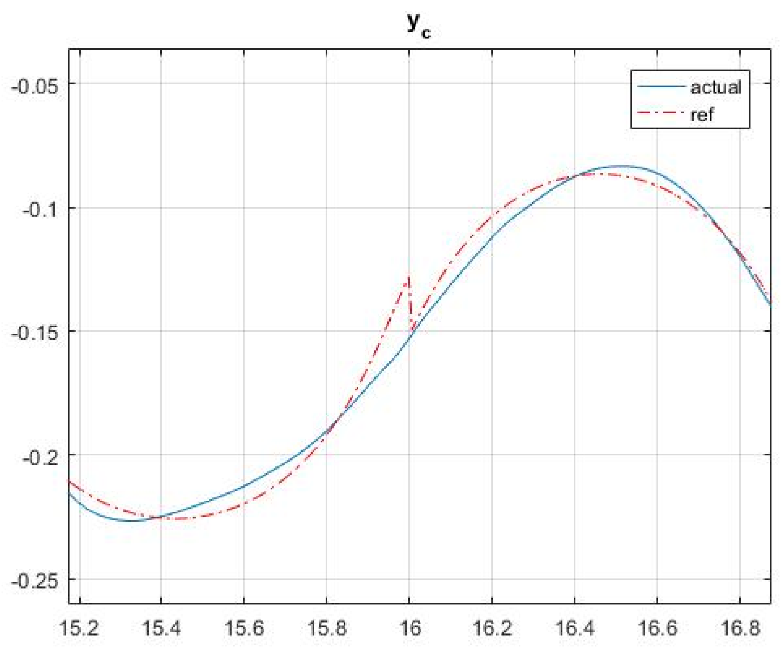


Figure 5.3: Close Look of Replanning for CoM

Up to this point, it has been shown that the replanning process is important for keeping the tracking error bounded and make the reference trajectory adaptive.

5.2 Evaluation of WBC controller

The main power of WBC controller is that it can cooperate all the joints to track both desired CoM and embedded limb motions. It's already shown in the squat motion simulation (Figure 4.7) and walking simulation Figure 4.12 that the CoM is tracked. Here, we show that the embedded motion, that is, the swing foot trajectory is also tracked. This is important since the landing area of the swing foot will determine the support polygon for double support phase and single support phase for next step.

For the 30-step walking simulation in Figure 4.12, the swing foot position is tracked:

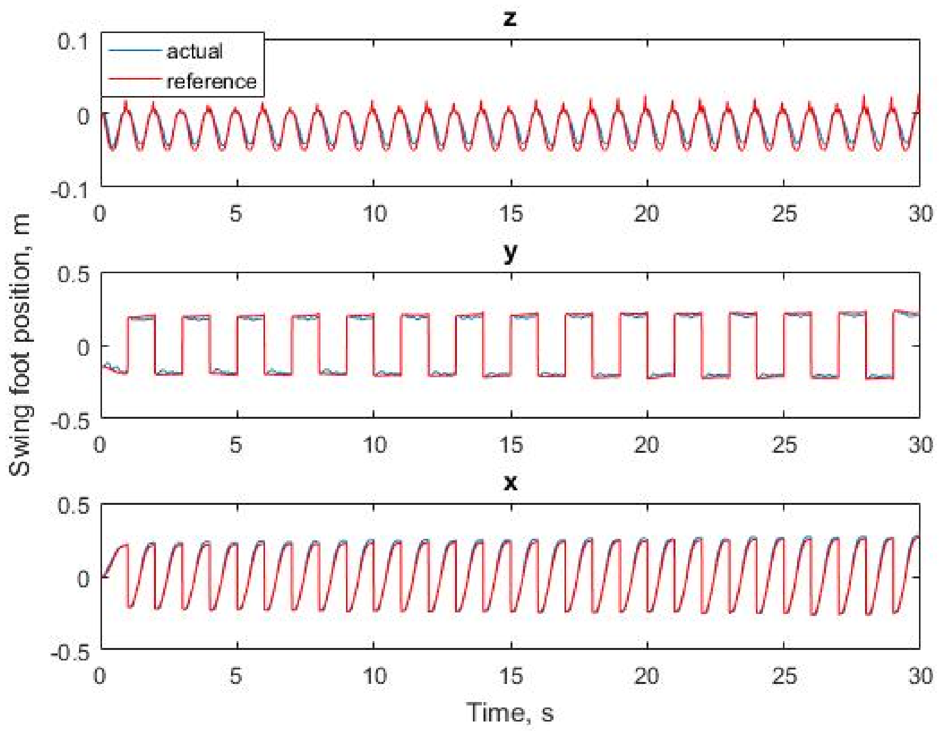


Figure 5.4: Swing foot Position Tracking in Stance Foot frame

Figure 5.4 shows the swing foot trajectory in the stance foot frame. The reference trajectory discussed in Section 4.3.1.2 is calculated in the world frame. When implementing this reference trajectory, it's transformed into stance foot frame, so that the calculations for tracking can be done in stance foot frame, just like tracking the end-effector of manipulator. This transformation happens at the beginning of every control loop, so even if the stance foot is not perfectly pointed forward due to tracking error, the reference swing foot position for this sample period is transformed into stance foot frame using the actual position and orientation of the stance foot, so the world frame tracking can be still achieved.

The discontinuity at the beginning of each step is due to switch of stance foot. Since it's in stance foot frame and the stance foot is switching for each step, the trajectory actually has a periodic behavior.

While all the tracking results show the efficiency of the WBC controller, there's still some challenging issues with it, which caused most of the failed simulations. In the next section, challenges and limitations of WBC controller will be discussed.

5.2.1 Limitation of WBC Controller: Fixed Stance Foot Assumption

The derivation of WBC method is based on a very important assumption, which is the fixed stance foot assumption, which assumes that the stance foot is fixed on the ground. In this case, the robot becomes simply a manipulator, and all the Jacobians calculated based on this assumption will holds, as long as the stance foot stays fixed. However, there will be some cases during the walking that break this assumption.

1. Large CoM error

   Since the WBC is a proportional controller that reads CoM error and send joint velocities as command. If the CoM error is large, the instantaneous joint velocities will be large. Among these joints, ankle joint plays an important role. If the instantaneous ankle joint velocity is too large, the foot will detach from the ground.

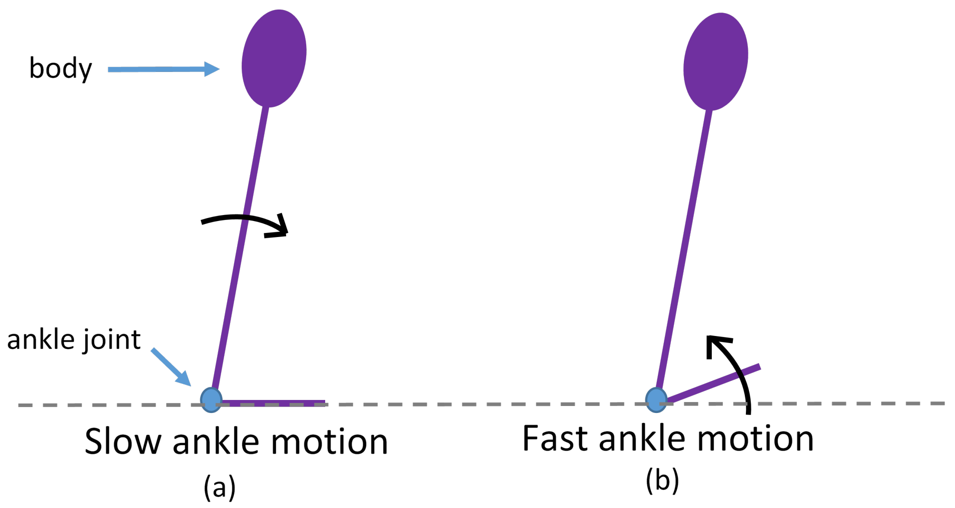


Figure 5.5: Ankle Joint Motion in Sagittal Plane

   In the sagittal plane, as shown in Figure 5.5 (a), if the foot rotates counter clockwise respect to the body in slow motion, the stance foot will stay on the ground but the body will rotate clockwise. This is the case when the WBC calculations can be valid. However, in Figure 5.5

(b), if the stance foot rotates too fast, it will detach from the ground before the body rotate clockwise. In this case, the fixed foot assumption is broken and all the WBC calculations are invalid.

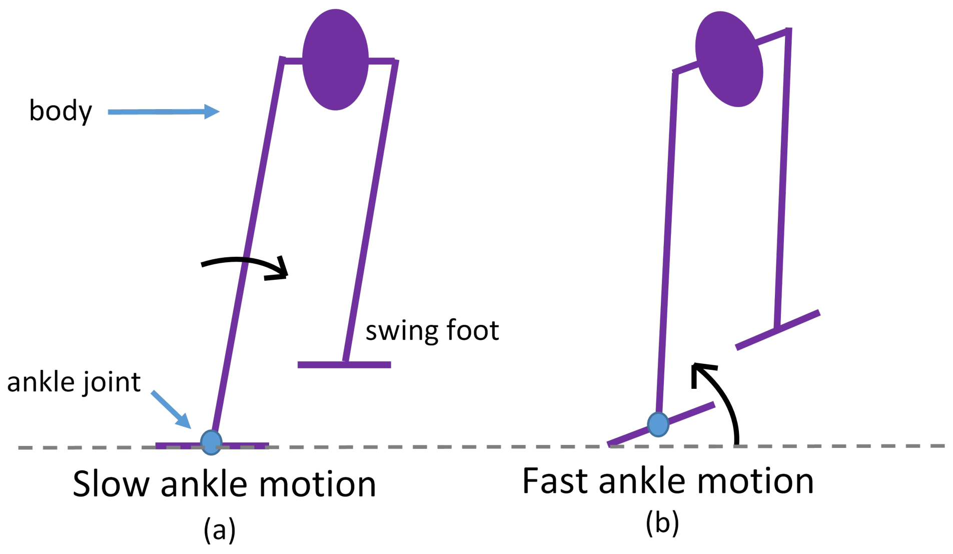


Figure 5.6: Ankle Joint Motion in Frontal Plane

Similar problem happens in the lateral motion as shown in Figure 5.6 (a). In single support phase, if the foot rotates counter clockwise respect to the body in slow motion, the stance foot will stay on ground but the body will rotate clockwise. In Figure 5.6 (b), if the stance foot rotates too fast, it will detach from the ground before the body rotate clockwise.

2. Collision of Swing Foot

In WBC method, the stance foot is considered as the base link, the swing foot is considered as the end-effector. During the double support phase, the swing foot contacts the ground. Even though the swing foot position and orientation are designed such that the swing foot will stay in the same plane as the stance foot, there's always tracking error, which causes the stance foot to detach from the ground (since it's not a real base link), as shown in Figure 5.7. The situation is worse in the transition from single support to double support, when the swing foot collides with the ground.

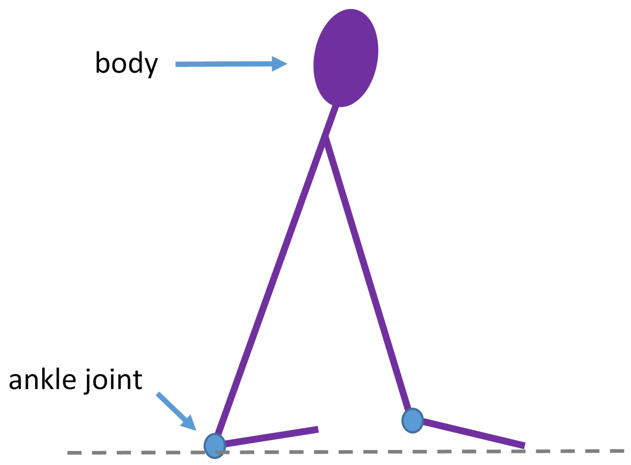


Figure 5.7: Double Support in Sagittal Plane

In order to avoid the effect of this limitation, and keep the fixed foot assumption valid all the time, an ankle joints position compensator is added in the simulations. The compensator simply read the orientation of stance foot, if the stance foot is detached from the ground, it will compensate the ankle joints positions such that the foot will resume the fixed pose respect to ground. This compensator is already implemented in the simulations. The details of the compensator can be found in Section 4.3.1.5. The trade off is that the tracking of CoM will be affected.

5.2.2 Limitation of WBC Controller: Stiff Joint Motion

Theoretically, the WBC method can directly control the joint velocities to achieve CoM tracking and embedded motion tracking, while in implementation it's actually controlling the joint positions' increments. And the position tracking is done by PID loop of each individual joint. In order to track the joint positions command from WBC controller almost simultaneously, the PID gains are set to be very large. The disadvantage of a high gain controller is that the joints become very stiff and lack of compliance.

During the landing of swing foot, human use the hip, knee and ankle joints to absorb the impact, such that the swing foot won't bounce. These can not be achieved by stiff joints of robots, so there's always large disturbances and discontinuities in swing foot position tracking.

To minimize the effect of impact, in the simulations of this paper, PID gains of swing foot's ankle joints are reduced to 0 during landing. So the ankle joints become completely compliant during landing, and reduces the impact to some degree.

CHAPTER 6

CONCULSION

The thesis has presented the development of biped robot walking control based on ZMP dynamics, with the emphasis on stabilizer and replanning algorithm design. This chapter is dedicated to conclude the study presented in this thesis. A brief summary of the conclusion and contribution of this study is presented in Section 6.1. Possible future developments will be discussed in Section 6.2.

6.1 Summary

In this thesis, the ZMP based biped walking control is studied. A new way of trajectory generation based on LIPM is discussed and developed. Several stabilizers for walking control are investigated and the WBC method is studied and derived in detail.

The walking pattern generation is based on the walking stability criteria of ZMP and the simplified model, LIPM. Because of the unstable nature of ZMP dynamics of LIPM, the boundedness issue of the solution is studied in Section 2.2.2. A boundedness constraint for the initial condition of CoM and the input ZMP trajectory is derived. This boundedness constraint provides a new way to generate convergent CoM trajectory, which allows us to parameterize the ZMP trajectory and solve for the unknown parameters using this boundedness constraint, as shown in Section 2.2.3. Based on this logic, a replanning algorithm is proposed and implemented in walking simulation of Reem-C, in Section 4.3.1.1. The replanning algorithm uses the actual CoM initial condition and solve for the step length that can result in a convergent CoM trajectory. The simulation results in Section 4.3.2.2 show that the replanning algorithm plays a critical role, which prevents the CoM tracking error to accumulate.

Two different stabilizers are studied and implemented in simulation. The first stabilizer, ankle position controller in Section 3.1, is an intuitive controller modified from the controller in [27]. The ankle joints angles are directly controlled to track CoM and ZMP. The simulations in Section 4.1 show that this controller cannot guarantee the tracking of CoM and ZMP since the tracking is slow and the steady state error is large. Also, since the ankle joints are controlled individually and they don't cooperate, which causes the soles not sharing the same plane. The robot will stand on the boundary of foot, therefore start to tip over.

Most of the work in this thesis is focus on the lower level stabilizer, WBC controller, which cooperate all the joints to track the CoM as well as embedded motion. The core concept, CoM Jacobian is derived as a mapping from joint velocities to world frame CoM velocities, in Section 3.2.1. A similar Jacobian, embedded motion Jacobian is also derived to map from the joint velocities to world frame embedded motion velocities, in Section 3.2.2. Based on these mappings, in Section 3.2.3, the WBC controller is derived in a proportional controller form, to track the overall CoM of the robot. Simulations in Section 4.2 and 4.3.2 show the good performance of WBC controller in walking. The CoM are tracked very fast with small errors. However, the limitations of WBC controller is obvious, due to the assumption that the stance foot is fixed on the ground. This assumption can be easily violated if the contact force needed to accelerate the robot exceed the contact force constraint or there's unexpected contact of other links, as discussed in Section 5.2.1. The stiffness of joints also limits the performance of WBC controller, by giving too much impact during the landing of swing foot.

To implement the complete control loop in simulation, the detailed procedures for implementation are listed in Section 4.3.1. It provides a comprehensive step-by-step instruction of implementing the integrated control algorithm for walking.

6.2 Future Works

The main contribution of this thesis is that the combination of a new planning method based on boundedness solution of LIPM and the WBC control, with their successful application in a realistic humanoid robot in the simulations. The preliminary simulation results show a promising future of this control method. Therefore, future work will focus on more application of the proposed control method.

To overcome the limitation of WBC controller, the contact force constraint needs to be studied and added to the control and planning algorithm.

So far, only walking in straight line is simulated. Algorithms for turning, and stopping should be studied in future work.

Also, the terrain is flat in all the simulations. A stabilizer that can handle uneven terrain needs to be studied.

APPENDIX A

SIMULATION PARAMETERS

A.0.1 Reem-C Lower Body Parameters (URDF file)

```
<?xml version="1.0"?>
<!--

  Copyright (c) 2011-2012, PAL Robotics, S.L.
  All rights reserved.

  This work is licensed under the Creative Commons
      Attribution-NonCommercial-NoDerivs 3.0 Unported License.
  To view a copy of this license, visit http://
      creativecommons.org/licenses/by-nc-nd/3.0/ or send a
      letter to
  Creative Commons, 444 Castro Street, Suite 900, Mountain
      View, California, 94041, USA.
-->
<robot name="reemc" xmlns:xacro="http://ros.org/wiki/xacro">

  <!--File includes-->
  <xacro:include filename="$(find reemc_description)/urdf/
      deg_to_rad.xacro" />
  <xacro:include filename="$(find reemc_description)/urdf/leg
      /leg.transmission.xacro" />
  <xacro:include filename="$(find reemc_description)/urdf/
      sensors/ftsensor.gazebo.xacro"/>
  <xacro:include filename="$(find reemc_description)/urdf/
      sensors/laser.urdf.xacro"/>

  <!--Constant parameters-->
  <xacro:property name="leg_friction"      value="1.0" />
  <xacro:property name="leg_damping"       value="1.0" />
  <xacro:property name="leg_reduction"     value="1.0" />
  <xacro:property name="leg_eps"           value="0.0" />
```

```
<xacro:property name="leg_friction_ankle"           value=
    "1.0" />
<xacro:property name="leg_damping_ankle"           value="
    1.0" />

<xacro:property name="leg_1_joint_max_vel"        value="
    2.35" />
<xacro:property name="leg_2_joint_max_vel"        value="
    2.27" />
<xacro:property name="leg_3_joint_max_vel"        value="
    3.64" />
<xacro:property name="leg_4_joint_max_vel"        value="
    3.225" />
<xacro:property name="leg_5_joint_max_vel"        value="
    2.44" />
<xacro:property name="leg_6_joint_max_vel"        value="
    2.27" />

<xacro:property name="leg_1_joint_effort"        value="
    42.7" />
<xacro:property name="leg_2_joint_effort"        value="
    64.0" />
<xacro:property name="leg_3_joint_effort"        value="
    55.7" />
<xacro:property name="leg_4_joint_effort"        value="
    138.3" />
<xacro:property name="leg_5_joint_effort"        value="
    80.9" /> <!-- defult: 80.9-->
<xacro:property name="leg_6_joint_effort"        value="64"
     /> <!-- defult: 64.0-->


<xacro:macro name="reemc_leg" params="name parent side
    reflect">

  <link name="leg_${side}_1_link">
      <inertial>
        <mass value="1.02901"/>
        <origin xyz="-0.02571 0 0.02557" rpy="0 0 0"/>
        <inertia ixx="0.00132187534" ixy="0.00000001514"
            ixz="0.00040022268" iyy="0.0018528617" iyz="
            0.00000050713" izz="0.00084231334"/>
      </inertial>
      <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://reemc_description/meshes/
            leg/leg_1.dae" scale="1 1 1"/>
      </geometry>
      <material name="LightGrey" />
    </visual>
    </link>

  <joint name="leg_${side}_1_joint" type="revolute">
      <origin  xyz="0  ${-reflect*0.075}  -0.14353" rpy="0 0
           0"/>
      <axis xyz="0 0 1"/>
      <parent link="${parent}"/>
      <child link="leg_${side}_1_link"/>
      <dynamics friction="${leg_friction}" damping="${
          leg_damping}"/>
      <limit lower="${-37.5*deg_to_rad + reflect*7.5*
          deg_to_rad}" upper="${37.5*deg_to_rad + reflect
          *7.5*deg_to_rad}" effort="${leg_1_joint_effort}"
          velocity="${leg_1_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${-37.5*
                             deg_to_rad + reflect*7.5*
                             deg_to_rad + leg_eps}"
                         soft_upper_limit="${ 37.5*
                             deg_to_rad + reflect*7.5*
                             deg_to_rad - leg_eps}" />
    </joint>

    <link name="leg_${side}_2_link">
      <inertial>
        <mass value="0.69621"/>
        <origin xyz="0.00057    ${-reflect*0.00881}
            -0.01125" rpy="0.0  0.0  0.0"/>
        <inertia ixx="0.0008416476" ixy="-0.00000268743"
            ixz="0.00000667199" iyy="0.00039794844" iyz="
            0.00002668971" izz="0.00084196694"/>
      </inertial>
      <visual>
```

```
        <origin rpy="${90*deg_to_rad} 0 0" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.10" radius="0.04"/>
        </geometry>
        <material name="LightGrey" />
      </visual>
    </link>

    <joint name="leg_${side}_2_joint" type="revolute">
      <origin xyz="0 0 0.0" rpy="0 ${ 90.0 * deg_to_rad} 0"
          />
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_1_link"/>
      <child link="leg_${side}_2_link"/>
      <dynamics friction="${leg_friction}" damping="${
          leg_damping}"/>
      <limit lower="${-22.5*deg_to_rad - reflect*7.5*
          deg_to_rad}" upper="${22.5*deg_to_rad - reflect
          *7.5*deg_to_rad}" effort="${leg_2_joint_effort}"
          velocity="${leg_2_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${-22.5*
                             deg_to_rad - reflect*7.5*
                             deg_to_rad + leg_eps}"
                         soft_upper_limit="${ 22.5*
                             deg_to_rad - reflect*7.5*
                             deg_to_rad - leg_eps}" />
  </joint>

  <link name="leg_${side}_3_link">
    <inertial>
      <mass value="3.97852"/>
      <origin xyz="0.14784 -0.01646 ${-reflect*0.01793}"
          rpy="0.0 0 0.0"/>
      <inertia ixx="0.00507885174" ixy="-0.00034076421" ixz
          ="-0.00225952731" iyy="0.02850097402" iyz="
          0.00003047874" izz="0.02632392239"/>
    </inertial>
   <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
```

```
          <mesh filename="package://reemc_description/meshes/
              leg/leg_3.dae" scale="1 1 ${-1*reflect}"/>
        </geometry>
        <material name="LightGrey" />
      </visual>
      <collision>
        <origin rpy="0 ${90* deg_to_rad} 0" xyz="0.15 0 ${-
            reflect*0.015}"/>
        <geometry>
          <box size="0.15 0.10 0.20"/>
        </geometry>
      </collision>
    </link>

    <joint name="leg_${side}_3_joint" type="revolute">
      <origin xyz="0 0 0.0" rpy="${ -90.0 * deg_to_rad} 0 0"
          />
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_2_link"/>
      <child link="leg_${side}_3_link"/>
      <dynamics friction="${leg_friction}" damping="${
          leg_damping}"/>
      <limit lower="${ -100.0 * deg_to_rad}" upper="${45 *
          deg_to_rad}" effort="${leg_3_joint_effort}" velocity
          ="${leg_3_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${-100.0 *
                             deg_to_rad + leg_eps}"
                         soft_upper_limit="${  45.0 *
                             deg_to_rad - leg_eps}" />
    </joint>

    <link name="leg_${side}_4_link">
      <inertial>
        <mass value="2.86055"/>
        <origin xyz="0.14618 -0.0022 ${-reflect*0.02189}" rpy
            ="0.0 0.0 0.0"/>
        <inertia ixx="0.00361006492" ixy="0.00012218208" ixz=
            "0.00160208242" iyy="0.02470649045" iyz="
            0.00008129755" izz="0.02272322206"/>
      </inertial>
      <visual>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <geometry>
          <mesh filename="package://reemc_description/meshes/leg/leg_4.dae" scale="1 1 ${-1*reflect}"/>
        </geometry>
        <material name="LightGrey" />
      </visual>
      <collision>
        <origin rpy="0 ${-reflect* 90* deg_to_rad} 0" xyz="0.15 0 ${-reflect*0.015}"/>
        <geometry>
          <box size="0.15 0.10 0.20"/>
        </geometry>
      </collision>
    </link>

    <joint name="leg_${side}_4_joint" type="revolute">
      <origin xyz="0.300 0 0.0" rpy="0 0 0" />
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_3_link"/>
      <child link="leg_${side}_4_link"/>
      <dynamics damping="0.1" friction="0"/>
      <dynamics friction="${leg_friction}" damping="${leg_damping}"/>
      <limit lower="${0 * deg_to_rad}" upper="${ 150.0 * deg_to_rad}" effort="${leg_4_joint_effort}" velocity="${leg_4_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${  0.0 * deg_to_rad + leg_eps}"
                         soft_upper_limit="${150.0 * deg_to_rad - leg_eps}" />
    </joint>

    <link name="leg_${side}_5_link">
      <inertial>
        <mass value="0.68213"/>
        <origin xyz="-0.00028 0.01119  ${-reflect*0.00859}" rpy="0.0 0.0 0.0"/>
```

```
        <inertia ixx="0.00082035023" ixy="0.00000048534" ixz=
            "0.00000226301" iyy="0.00082074105" iyz="
            -0.00002892247" izz="0.00037614067"/>
      </inertial>
      <visual>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.104" radius="0.04"/>
        </geometry>
        <material name="LightGrey" />
      </visual>

    </link>

    <joint name="leg_${side}_5_joint" type="revolute">
      <origin xyz="0.30 0 0.0" rpy="0 0 0" />
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_4_link"/>
      <child link="leg_${side}_5_link"/>
      <dynamics friction="${leg_friction_ankle}" damping="${
         leg_damping_ankle}"/>
      <limit lower="${-100 * deg_to_rad}" upper="${ 45.0 *
         deg_to_rad}" effort="${leg_5_joint_effort}" velocity
         ="${leg_5_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${-100.0 *
                            deg_to_rad + leg_eps}"
                         soft_upper_limit="${  45.0 *
                            deg_to_rad - leg_eps}" />
    </joint>

    <link name="leg_${side}_6_link">
      <inertial>
        <mass value="2.00843"/>
        <origin xyz="0.06111 ${-reflect*0.00336} -0.00497"
            rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.00366502559" ixy="0.00020479295" ixz=
            "0.00112439297" iyy="0.00532618026" iyz="
            0.00012133755" izz="0.00353091463"/>
      </inertial>
      <visual>
```

```
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <geometry>
          <mesh filename="package://reemc_description/meshes/leg/leg_6.dae" scale="1 ${-1*reflect} 1"/>
        </geometry>
        <material name="LightGrey" />
      </visual>
      <collision>
        <origin rpy="0 0 0" xyz="0.1 ${-reflect*0.01} 0.025 " />
        <geometry>
          <box size="0.02 0.14 0.210"/>
        </geometry>
      </collision>
    </link>

    <joint name="leg_${side}_6_joint" type="revolute">
      <origin xyz="0 0 0" rpy="${90.0 * deg_to_rad} 0 0" />
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_5_link"/>
      <child link="leg_${side}_6_link"/>
      <dynamics friction="${leg_friction_ankle}" damping="${leg_damping_ankle}"/>
      <limit lower="${-22.5*deg_to_rad + reflect*7.5*deg_to_rad}" upper="${ 22.5*deg_to_rad + reflect*7.5*deg_to_rad}" effort="${leg_6_joint_effort}" velocity="${leg_6_joint_max_vel}" />

      <safety_controller k_position="100"
                         k_velocity="100"
                         soft_lower_limit="${-22.5*deg_to_rad + reflect*7.5*deg_to_rad + leg_eps}"
                         soft_upper_limit="${ 22.5*deg_to_rad + reflect*7.5*deg_to_rad - leg_eps}" />
    </joint>

    <link name="${side}_sole_link">
      <inertial>
        <mass value="0.001"/>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz=
            "0.0" izz="0.0"/>
      </inertial>
    </link>

    <joint name="${side}_sole_joint" type="fixed">
      <!-- origin x taken from link_${side}_6_link/visual/
          origin/x above -->
      <origin xyz="0.117 0.0 0.0" rpy="0.0 ${-90.0 *
          deg_to_rad} 0.0"/>
      <axis xyz="0 0 1"/>
      <parent link="leg_${side}_6_link"/>
      <child link="${side}_sole_link"/>
      <dynamics friction="${leg_friction}" damping="${
          leg_damping}"/>
      <limit lower="0.0" upper="0.0" effort="60" velocity="0"
          />
    </joint>


   <gazebo reference="leg_${side}_1_link">
     <mu1>0.9</mu1>
     <mu2>0.9</mu2>
   </gazebo>
   <gazebo reference="leg_${side}_2_link">
     <mu1>0.9</mu1>
     <mu2>0.9</mu2>
   </gazebo>
   <gazebo reference="leg_${side}_3_link">
     <mu1>0.9</mu1>
     <mu2>0.9</mu2>
   </gazebo>
   <gazebo reference="leg_${side}_4_link">
     <mu1>0.9</mu1>
     <mu2>0.9</mu2>
   </gazebo>
   <gazebo reference="leg_${side}_5_link">
     <mu1>0.9</mu1>
     <mu2>0.9</mu2>
   </gazebo>

   <!-- contact model for foot surface -->
   <gazebo reference="leg_${side}_6_link">
```

```
      <kp>1000000.0</kp>
      <kd>100000000000000.0</kd>
      <mu1>5.0</mu1>
      <mu2>5.0</mu2>
      <fdir1>0 0 1</fdir1>
      <maxVel>1.0</maxVel>
      <minDepth>0.00</minDepth>
      <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>

    <!--force torque sensor -->
    <!--xacro:reemc_force_torque_sensor name="leg_${side}
        _6_link"  update_rate="100.0"/-->

    <gazebo reference="leg_${side}_1_joint">
       <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>
    <gazebo reference="leg_${side}_2_joint">
       <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>
    <gazebo reference="leg_${side}_3_joint">
       <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>
    <gazebo reference="leg_${side}_4_joint">
       <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>
    <gazebo reference="leg_${side}_5_joint">
       <implicitSpringDamper>1</implicitSpringDamper>
    </gazebo>
    <gazebo reference="leg_${side}_6_joint">
      <implicitSpringDamper>1</implicitSpringDamper>
      <provideFeedback>1</provideFeedback>
    </gazebo>

     <xacro:reemc_leg_simple_transmission name="${name}" side=
         "${side}" number="1" reduction="${leg_reduction}" />
     <xacro:reemc_leg_simple_transmission name="${name}" side=
         "${side}" number="2" reduction="${leg_reduction}" />
     <xacro:reemc_leg_simple_transmission name="${name}" side=
         "${side}" number="3" reduction="${leg_reduction}" />
     <xacro:reemc_leg_simple_transmission name="${name}" side=
         "${side}" number="4" reduction="${leg_reduction}" />
    <xacro:reemc_leg_simple_transmission name="${name}" side=
        "${side}" number="5" reduction="${leg_reduction}" />
    <xacro:reemc_leg_simple_transmission name="${name}" side=
        "${side}" number="6" reduction="${leg_reduction}" />

   <xacro:reemc_laser name="${side}_laser" parent="${side}
       _sole_link" ros_topic="${side}_scan" update_rate="10.0"
        min_angle="${-90*deg_to_rad}" max_angle="${90*
       deg_to_rad}" nrays="360" >
     <origin xyz="${0.170 -0.085} ${0.071 -((1+reflect)*0.01
         + 0.061)} ${0.073}" rpy="0.0 0.0 ${-30.0*reflect*
         deg_to_rad}" />
    </xacro:reemc_laser>

  </xacro:macro>
</robot>
```

A.0.2 Controller Gains

For all joints, the PID gains for joint position tracking are the same (given by Pal Robotics):

Table A.1: Joint Psosition Controller PID Gains

P gain	I gian	D gian
3000	1	10

The WBC controller gain matrices are tuned to be:

$$\begin{aligned} K_{po} &= diag(150, 100, 75, 140, 140, 140) \\ K_{sw,or} &= diag(90, 90, 75) \\ K_{0,or} &= diag(55, 55, 55) \end{aligned} \quad (A.1)$$

where $diag()$ is a diagonal matrix.

REFERENCES

[1] M. Vukobratović and J. Stepanenko, "On the stability of anthropomorphic systems," *Mathematical biosciences*, vol. 15, no. 1-2, pp. 1–37, 1972.

[2] 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 *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2. IEEE, 2003, pp. 1620–1626.

[3] S. Kajita, M. Morisawa, K. Miura, S. Nakaoka, K. Harada, K. Kaneko, F. Kanehiro, and K. Yokoi, "Biped walking stabilization based on linear inverted pendulum tracking," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 4489–4496.

[4] J. Englsberger, C. Ott, M. A. Roa, A. Albu-Schäffer, and G. Hirzinger, "Bipedal walking control based on capture point dynamics," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on.* IEEE, 2011, pp. 4420–4427.

[5] L. Lanari and S. Hutchinson, "Planning desired center of mass and zero moment point trajectories for bipedal locomotion," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on.* IEEE, 2015, pp. 637–642.

[6] A. Herzog, L. Righetti, F. Grimminger, P. Pastor, and S. Schaal, "Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on.* IEEE, 2014, pp. 981–988.

[7] A. Hof, M. Gazendam, and W. Sinke, "The condition for dynamic stability," *Journal of biomechanics*, vol. 38, no. 1, pp. 1–8, 2005.

[8] S. Kajita and K. Tani, "Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on.* IEEE, 1991, pp. 1405–1411.

[9] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, "The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2001, pp. 239–246.

[10] D. Rosenthal, "On the optimal digital state vector feedback controller with integral and preview actions1," *Journal of Dynamic Systems, Measurement, and Control*, vol. 101, p. 173, 1979.

[11] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa, "An analytical method for real-time gait planning for humanoid robots," *International Journal of Humanoid Robotics*, vol. 3, no. 01, pp. 1–19, 2006.

[12] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara, S. Nakaoka, and H. Hirukawa, "A biped pattern generation allowing immediate modification of foot placement in real-time," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, 2006, pp. 581–586.

[13] K. Harada, K. Miura, M. Morisawa, K. Kaneko, S. Nakaoka, F. Kanehiro, T. Tsuji, and S. Kajita, "Toward human-like walking pattern generator," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 1071–1077.

[14] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, 2006, pp. 200–207.

[15] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models," *The International Journal of Robotics Research*, vol. 31, no. 9, pp. 1094–1113, 2012.

[16] L. Lanari, S. Hutchinson, and L. Marchionni, "Boundedness issues in planning of locomotion trajectories for biped robots," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE, 2014, pp. 951–958.

[17] S. Kajita, "Overview of zmp-based biped walking," Keynote Presentation, 2008.

[18] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to humanoid robotics*. Springer, 2014, vol. 101.

[19] Y. Choi, D. Kim, Y. Oh, and B.-J. You, "Posture/walking control for humanoid robot based on kinematic resolution of com jacobian with embedded motion," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1285–1293, 2007.

[20] R. Tedrake, S. Kuindersma, R. Deits, and K. Miura, “A closed-form solution for real-time zmp gait generation and feedback stabilization,” in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on.* IEEE, 2015, pp. 936–940.

[21] Y. Choi, D. Kim, and B.-J. You, “On the walking control for humanoid robot based on the kinematic resolution of com jacobian with embedded motion,” in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE, 2006, pp. 2655–2660.

[22] S. Kajita, M. Morisawa, K. Harada, K. Kaneko, F. Kanehiro, K. Fujiwara, and H. Hirukawa, “Biped walking pattern generator allowing auxiliary zmp control,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 2993–2999.

[23] T. Sugihara and Y. Nakamura, “Whole-body cooperative balancing of humanoid robot using cog jacobian,” in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2575–2580.

[24] 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 *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2003, pp. 1644–1650.

[25] K.-h. Ahn and Y. Oh, “Walking control of a humanoid robot via explicit and stable com manipulation with the angular momentum resolution,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 2478–2483.

[26] Q. Huang, K. Kaneko, K. Yokoi, S. Kajita, T. Kotoku, N. Koyachi, H. Arai, N. Imamura, K. Komoriya, and K. Tanie, “Balance control of a piped robot combining off-line pattern with real-time modification,” in *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*, vol. 4. IEEE, 2000, pp. 3346–3352.

[27] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, “The development of honda humanoid robot,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 2. IEEE, 1998, pp. 1321–1326.

[28] D. Kim, Y. Choi, and C. Kim, “Motion-embedded cog jacobian for a real-time humanoid motion generation.” in *ICINCO*, 2005, pp. 55–61.

[29] R. Beranek, H. Fung, and M. Ahmadi, “A walking stability controller with disturbance rejection based on cmp criterion and ground reaction force feedback,” in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on.* IEEE, 2011, pp. 2261–2266.

[30] “Remm-c humanoid robot by pal robotics,” http://pal-robotics.com/en/products/reem-c, 2016.

[31] “Gazebo simulator,” http://gazebosim.org/, 2016.

[32] “Robot operating system,” http://www.ros.org, 2016.

[33] “Robotics system toolbox, mathworks,” 2016, https://www.mathworks.com/products/robotics.html.

[34] “Simulation play list,” https://www.youtube.com/watch?v=PcKEdUVqZtQ, 2016.