Karlsruhe Institute of Technology

Dynamically Stable Walking For Humanoid Bipedal Robots Based On Walking Patterns

Bachelors's Thesis
of

Patrick Niklaus

At the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
High Perfomance Humanoid Technologies Lab (H^2T)

You can specify the advisors and/or referees with
\advisor, \advisortwo, \referee, \refereetwo.
If none needed: `\newcommand{\noadvisors}{}`

Duration: 15. September 2014 – Please redefine the `timeend` macro:
`\renewcommand{\timeend}{...}`

Contents

1 Introduction

motivation, and a bit of overview of humanoid walking. I recommend to leave it for later, start with the sections that you feel its easier to write (usually, the ones that have more content).

- motivation:
  - navigating in human environments
- walking in humans:
  - CoM movement, gait phases, differences to what we do here
- static vs. dynamic walking
- overview of models used for dynamic walking

2 Models for humanoid walking

2.1 The linear inverted pendulum model

A simple model for describing the dynamics of a bipedal robot during single support phase is the 3D inverted pendulum. We reduce the body of the robot to a point-mass at the center of mass and replace the support leg by a mass-less telescopic leg which is fixed at a point on the supporting foot. Initially this will yield non-linear equations that will be hard to control. Howevery by constraining the movement of the inverted pendulum to a fixed plane, we can derive a linear dynamic system. This model called the 3D *linear* inverted pendulum model (short *3D-LIPM*).

Use different name for CoM, p will be rather used for the ZMP, maybe c?

picture of 3D-LIPM

2.2 The inverted pendulum

To describe the dynamics of the inverted pendulum we are mainly interested in the effect a given actuator torque has on the movement of the pendulum.

For simplicity we assume that the base of the pendulum is fixed at the origin of the current cartesian coordinate system. Thus we can describe the position inverted pendulum by a vector $p = (x, y, z)$. We are going to introduce an appropriate (generalized) coordinate system $q = (\theta_R, \theta_P, r)$ to get an easy description of our actuator torques: Let m be the mass of the pendulum and r the length of the telescopic leg. θ_P and θ_R describe the corresponding roll and pitch angles of the pose of the pendulum.

add image with angles here

Now we need to find a mapping between forces in the cartesian coordinate system and the generalized forces (the actuator torques). Let $\Phi : \mathbb{R}^3 \longrightarrow \mathbb{R}^3, (\theta_R, \theta_P, r) \mapsto (x, y, z)$ be a function that maps the generalized coordinates to the cartesian coordinates. Then the jacobian $J_\Phi = \frac{\partial p}{\partial q}$ maps the *generalized velocites* to *cartesian velocites*. Furthermore we know that the transpose J_Φ^T maps *cartesian forces* $F = m(\ddot{x}, \ddot{y}, \ddot{z})$ to *generalized forces* (τ_r, τ_p, f).

We write x, y and z in terms of our generalized coordinates to compute the corresponding jacobian J_Φ. From the fact that the θ_P is the angle between the projection of p onto the xz-plane and p and θ_R the angle between p and the projection onto the yz plane we can derive the following equations :

reference paper

$$\begin{aligned} x &= r \cdot \sin\theta_P && =: r \cdot s_P \\ y &= -r \cdot \sin\theta_R && =: -r \cdot s_R \\ z &= \sqrt{r^2 - x^2 - y^2} = r \cdot \sqrt{1 - s_P^2 - s_R^2} \end{aligned} \tag{2.1}$$

From which we can compute the jacobian by partial derivation:

$$J = \frac{\partial p}{\partial q} = \begin{pmatrix} 0 & r \cdot c_P & s_P \\ -r \cdot c_R & 0 & s_P \\ \frac{2 \cdot r \cdot s_P c_P}{\sqrt{1 - s_P^2 - s_R^2}} & \frac{2 \cdot r \cdot s_R c_R}{\sqrt{1 - s_P^2 - s_R^2}} & \sqrt{1 - s_P^2 - s_R^2} \end{pmatrix} \tag{2.2}$$

Using the equation of motion as given by

$$\begin{aligned} F &= (J^T)^{-1}\Gamma + f_g \\ m \cdot \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} &= (J^T)^{-1} \begin{pmatrix} \tau_R \\ \tau_P \\ f \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -m \cdot g \end{pmatrix} \end{aligned} \tag{2.3}$$

and equations 2.2 and 2.1 we can derive the following equations:

$$m(-z\ddot{y}+y\ddot{z}) = \frac{\sqrt{1-s_P^2-s_R^2}}{c_R} \cdot \tau_R + mgy \tag{2.4}$$

$$m(z\ddot{x}-x\ddot{z}) = \frac{\sqrt{1-s_P^2-s_R^2}}{c_P} \cdot \tau_P + mgx \tag{2.5}$$

Observe that the terms of the left-hand side are not linear. To remove that non-linearity we are going to use the *linear* inverted pendulum model.

2.3 Linear Inverted Pendulum Model

In a man-made environment it is fair to assume that the ground a robot will walk on can be approximate by a slightly sloped plane. In most cases it can even assumed that there is no slope at all.

The basic assumption in the next section will be that the CoM will have a *constant displacement* with regard to our ground plane. Thus we can constrain the movement of the CoM to a plane that is parallel to the ground plane. Note that this assumption is, depending on the walking speed, only approximately true for human walking as shown by Orendurff et. al. For slow to fast walking (0.7 m/s and 1.6 m/s respectively) the average displacement in z-direction was found to be between $2.7cm$ and 4.81 cm. While the walking patterns generated based on the LIP-model will guarantee dynamic stability, they might not look natural with regard to human walking.

cite Orendurff

We are going to constrain the z coordinate of our inverted pendulum to a plane with normal vector $(k_x, k_y, -1)$ and z-displacement z_c:

$$z = k_x \cdot x + k_y \cdot y + z_c \tag{2.6}$$

Subsequently the second derivative of z can be described by:

$$\ddot{z} = k_x \cdot \ddot{x} + k_y \cdot \ddot{y} \tag{2.7}$$

Substituing 2.6 and 2.7 into the equations 2.4 and 2.5 yields the following equations:

$$\ddot{y} = \frac{g}{z_c} y - \frac{k_x}{z_c}(x\ddot{y}-\ddot{x}y) - mz_c \cdot \tau_R \cdot \frac{\sqrt{1-s_P^2-s_R^2}}{c_R} \tag{2.8}$$

$$\ddot{x} = \frac{g}{z_c} x + \frac{k_y}{z_c}(x\ddot{y}-\ddot{x}y) + mz_c \cdot \tau_P \cdot \frac{\sqrt{1-s_P^2-s_R^2}}{c_P} \tag{2.9}$$

The term $x\ddot{y}-\ddot{x}y$ that is part of both equations is still causing the equations to be non-linear. To make this equations linear we will assume that our ground plane has no slope, thus $k_x = k_y = 0$ and the non-linear terms will vanish.

Another problem is that the actuator torques τ_R and τ_P both have non-linear factors $\frac{\sqrt{1-s_P^2-s_R^2}}{c_R}$ and $\frac{\sqrt{1-s_P^2-s_R^2}}{c_P}$ respectively. This can be solved by substituting with the following *virtual inputs*:

$$\tau_P \cdot \frac{\sqrt{1-s_P^2-s_R^2}}{c_P} = u_P \tag{2.10}$$

$$\tau_R \cdot \frac{\sqrt{1-s_P^2-s_R^2}}{c_R} = u_R \tag{2.11}$$

Which yields our final description of the dynamics:

$$\ddot{y} = \frac{g}{z_c}y - \frac{u_R}{mz_c} \tag{2.12}$$

$$\ddot{x} = \frac{g}{z_c}x + \frac{u_R}{mz_c} \tag{2.13}$$

include pattern generation just based on 3D-LIPM, I don't understand how they derived the controller

2.4 The Zero Moment Point

A very popular approach to humanoid walking are schemes based on the Zero Moment Point. One reason for that might be that it is very simple to describe constrains for dynamic stability using this reference point. As long as the following condition is met we will have full ground contact of our support foot and thus can realize dynamically stable walking: *The ZMP is strictly inside the support polygone of the support foot.*

For flat ground contact of our support foot with the floor the ZMP corresponds with the position of the center of pressure (CoP). Indeed, some author (notably Pratt) prefer to use the term CoP instead of ZMP.

The CoP of an object in contact with the ground can be computed as the sum of all contact points $p_1, \ldots, p_n$ weighted by the forces in z-direction $f_{1z}, \ldots, f_{nz}$ that is applied:

$$p := \frac{\sum_{i=1}^N p_i f_{iz}}{\sum_{i=1}^N f_{iz}} \tag{2.14}$$

An important fact (and the origin of its name) is that there are no torques around the x and y axis at the ZMP:

$$\tau = \sum_{i=1}^{N} (p_i - p) \times f_i \tag{2.15}$$

Splitting that up into each component using the definition of the cross product yields:

$$\tau_x = \sum_{i=1}^{N} (p_{iy} - p_y) f_{iz} - \overbrace{(p_{iz} - p_z)}^{=0} f_{iy} \tag{2.16}$$

$$\tau_y = \sum_{i=1}^{N} \overbrace{(p_{iz} - p_z)}^{=0} f_{ix} - (p_{ix} - p_x) f_{iz} \tag{2.17}$$

$$\tau_z = \sum_{i=1}^{N} (p_{ix} - p_x) f_{iy} - (p_{iy} - p_y) f_{ix} \tag{2.18}$$

Since we have flat ground contact, all contact points have the same z-coordinate as the ZMP, thus we can simplify τ_x and τ_y to:

$$\tau_x = \sum_{i=1}^{N} (p_{iy} - p_y) f_{iz} = \sum_{i=1}^{N} (p_{iy} f_{iz}) - (\sum_{i=0}^{N} f_{iz}) \cdot p_y \tag{2.19}$$

$$\tau_y = \sum_{i=1}^{N} -(p_{ix} - p_x) f_{iz} = \sum_{i=1}^{N} -(p_{ix} f_{iz}) + (\sum_{i=0}^{N} f_{iz}) \cdot p_x \tag{2.20}$$

Furthermore we can use the corresponding components p_x and p_y from the definition of the ZMP 2.14 and substitude in the equations 2.19 and 2.20.

This will yield: $\tau_x = \tau_y = 0$.

Please note that τ_z will in general not be zero, nonetheless in case of straight walking it is often assumed to be zero as well.

2.4.1 The table-cart model

The table-cart model is equivalent to the 3D-LIPM model discussed before, but somewhat more intuitive for computing the resulting ZMP from an CoM motion. The model consists of an (infinitely) large mass-less table of height z_c, while the foot of the table has the shape of the support polygone. Given a frictionless cart with mass m that moves on the table we can compute the resulting ZMP in the support foot. Please note that the 3D-dimensional model is equivalent to having two independent tables with two carts each in the xz and yz-plane respectively. First of all, lets compute the torque τ_x and τ_y around the x-axis and y-axis at the ZMP on the support foot.

$$\tau_y = \overbrace{-mg(c_x - p_x)}^{\text{torque due to gravity}} + \overbrace{m\ddot{x} \cdot z_c}^{\text{torque due to acceleration of cart}} \tag{2.21}$$

$$\tau_x = -mg(c_y - p_y) + m\ddot{y} \cdot z_c \tag{2.22}$$

Please note the similarity to the equations 2.12 and 2.13 when assuming the base of the pendulum is located at p. If we now use the property of the ZMP that the torque around the x and y-axis is zero, we can solve for the ZMP position p:

$$p_x = c_x - \frac{z_c}{g}\ddot{c}_x \tag{2.23}$$

$$p_y = c_y - \frac{z_c}{g}\ddot{c}_y \tag{2.24}$$

Maybe describe the full-body methode to compute the ZMP

short introduction how that works and some problems with the approach

2.5 Simulating rigid body dynamics

3 Pattern generator

To generate a walking pattern for a bipedal robot two basic approaches are common:

1. Generate (or modify) foot trajectories that realize a prescribed trajectory of the CoM
2. Generate a CoM trajectory for prescribed foot trajectories

The first approach is generally used for implementing pattern generators soley based on the 3D-LIPM model.

The second approach is the more versatile one, since it is easy to incorporate constrains of our environment (e.g. only limited foot holds) in the input of the pattern generator. However care must be taken while chosing adequate step width and step length parameters for the foot trajectory, so that they can actually be realized by the robot. citation needed

The pattern generator proposed by Kajita et al. based on Preview Control realizes the second approach. We will discuss the theoretical background of this pattern generator here in more detail, since all pattern that we used where generated that way. add citation

3.1 Computing the CoM from a reference ZMP

As we saw in the section 2.4.1 it is easy to compute the resulting ZMP given the CoM and its acceleration. However for generating the walking pattern, we want to compute the CoM trajectory from a given ZMP. If you rearange the equations 2.23 and 2.24 you see that we have to solve a second order differential equations:

$$c_x = \frac{z_c}{g} \cdot \ddot{c_x} + p_x \tag{3.1}$$

$$c_y = \frac{z_c}{g} \cdot \ddot{c_y} + p_y \tag{3.2}$$

There are several ways to solve this differential equations, for example by transforming them to the frequency-domain. This however would mean, the ZMP trajectory needs to be transformed to the frequency domain as well, e.g. using Fast Fourier Transformation. This has two main problems:

1. It has a significant computational overhead. (For FFT the additional runtime would be in $O(n \log n)$)
2. We need to know the whole ZMP trajectory in advance.

Instead Kajita et. al. chose to define a dynamic system in the time domain that describes the CoM movement.

3.1.1 Pattern generation as dynamic system

For simplicity we will only focus on the dynamic description of one dimension, as the other one is analogous. To transform the equations to a strictly proper dynamical system, we need to determine the state vector of our system. For the table-cart model it suffices to know the position, velocity and acceleration of the cart. Thus the state-vector is defined as $x = (c_x, \dot{c_x}, \ddot{c_x})$. We can define the evolution of the state vector as follows:

maybe do a formal introduction into dynamic system and the state space approach

$$\frac{d}{dt}\begin{pmatrix} c_x \\ \dot{c_x} \\ \ddot{c_x} \end{pmatrix} = \overbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}}^{=:A_0} \cdot \begin{pmatrix} c_x \\ \dot{c_x} \\ \ddot{c_x} \end{pmatrix} + \overbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}}^{=:B_0} u \tag{3.3}$$

As you can see the jerk of the CoM was introduced as an input $u_x = \frac{d}{dt}\ddot{c_x}$ into the dynamic system.
We use equation 2.23 to calculate the actual output of the dynamic system the resulting zmp, that will be controlled:

$$p_x = \begin{pmatrix} 1 & 0 & \frac{-z_c}{g} \end{pmatrix} \cdot \begin{pmatrix} c_x \\ \dot{c_x} \\ \ddot{c_x} \end{pmatrix} \tag{3.4}$$

Using this formulation of the dynamic system we need to derive the evolution of our state vector using the state-transition matrix. Since our input ZMP trajectory will consist of discrete samples at equal time intervals T we define the discrete state as $x[k] := x(k \cdot T)$. Please note that this system is a linear time-invariant system (LTI), and both matrices A_0 and B_0 are constant. We can therefore use the standart approach to solve this system using the equation:

$$x(t) = e^{A_0\cdot(t-\tau)}x(\tau) + \int_\tau^t e^{A_0\cdot(t-\lambda)}B_0u(\lambda)d\lambda \tag{3.5}$$

In our discrete case that becomes:

$$\begin{aligned} x[k+1] &= e^{A_0\cdot((k+1)T-kT)}x[k] + \int_{kT}^{(k+1)T} e^{A_0\cdot((k+1)T-\lambda)}B_0u(\lambda)d\lambda && (3.6) \\ &= e^{A_0\cdot T}x[k] + \left(\int_{kT}^{(k+1)T} e^{A_0\cdot((k+1)T-\lambda)}d\lambda\right)\cdot B_0u[k] && (3.7) \\ &= e^{A_0\cdot T}x[k] + \left(\int_T^0 e^{A_0\cdot\lambda}d\lambda\right)\cdot B_0u[k] && (3.8) \end{aligned}$$

Keep in mind that $u(\lambda) = u[k], \lambda \in (kT, (k+1)T)$ so we can move it outside of the integral. Let us first compute a general solution for the matrix exponential $e^{A_0\cdot t}$. It is easy to see that A_0 is nilpotent and $A_0^3 = 0$, thus the computation simpilfies to the following:

$$e^{A_0t} := \sum_{i=0}^{\infty} \frac{(A_0\cdot t)^i}{i!} = I + A_0\cdot t + A_0^2\cdot\frac{t^2}{2} + 0 = \begin{pmatrix} 1 & t & \frac{t^2}{2} \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

Using that computing the integral in 3.6 is quite easily:

$$\int_T^0 e^{A_0\cdot\lambda}d\lambda = -\int_0^T \begin{pmatrix} 1 & t & \frac{t^2}{2} \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix} dt = -\left.\begin{pmatrix} t & \frac{t^2}{2} & \frac{t^3}{6} \\ 0 & t & \frac{t^2}{2} \\ 0 & 0 & t \end{pmatrix}\right|_0^T = \begin{pmatrix} T & \frac{t^2}{2} & \frac{T^3}{6} \\ 0 & T & \frac{T^2}{2} \\ 0 & 0 & T \end{pmatrix} \tag{3.10}$$

Substituting the results in 3.6 yields:

$$x[k+1] = \overbrace{\begin{pmatrix} T & \frac{t^2}{2} & \frac{T^3}{6} \\ 0 & T & \frac{T^2}{2} \\ 0 & 0 & T \end{pmatrix}}^{=:A} x[k] + \overbrace{\begin{pmatrix} \frac{T^3}{6} \\ \frac{T^2}{2} \\ T \end{pmatrix}}^{=:B} \cdot u_x[k] \tag{3.11}$$

3.1.2 Controlling the dynamic system

To control this dynamic system we need to determine an adequate control input u_x to realize the reference ZMP trajectory. A performence index J_x for a given control input u_x is needed to formalize what a "good" control input would be. A naive performence index could be:

$$J_x[k] := (p_x^{ref}[k] - p_x[k])^2 \tag{3.12}$$

To minimize it, we need to find u_x for which $p_x = p_x^{ref}$. By substituting $p_x[k]$ with 3.4 and $x[k]$ with 3.11 this yields:

$$u_x[k] = \frac{p_x^{ref}[k+1] - C \cdot A \cdot x[k]}{C \cdot B} = \frac{p_x^{ref}[k+1] - (1, T, \frac{1}{2}T^2 - \frac{z_c}{g}) \cdot x[k]}{\frac{1}{6}T^3 - \frac{z_c}{g}T} = \frac{p_x^{ref}[k+1] - p_x[k] - T\dot{c}_x[k] - \frac{1}{2}T^2\ddot{c}_x[k]}{\frac{1}{6}T^3 - \frac{z_c}{g}T} \tag{3.13}$$

To analyse the behaviour of this control law for u_x we simulate the rapid change of reference ZMP when changing the support foot.

insert plot

As you can see the reference ZMP is perfectly tracked. However, the CoM does not behave as expected. To achive the required ZMP position the CoM will be *accelerated indefinietly* in the opposite direction. Clearly this is not desired and will lead to falling on a real robot. A more sophisticated performence index is needed. To eventually reach a stable state at which the CoM comes to rest, the performence index should include a state feedback. Also note the large jerk that is applied to the system when the reference ZMP position changes rappidly. In a real mechanical system large jerks will lead to oszillations, which will disturb the system. Thus the performence index should also try to limit the applied jerk.

Another problem becomes apparent when you think about the nature of a controller: The controller starts to act *after* we have a deviation from our reference ZMP trajectory. Trying make this lag as small as possible can lead to very high velocities, which might not be realizable by motors of a robot. However we have at least limited knowledge of the future reference trajectory. This knowledge can be leveraged by using Preview Control, which considers the next N timesteps for computing the performence index.

Kajita et. al. use a performence index proposed by Katayama et. al. to solve all of the problems above:

add citation katayama

$$J_x[k] = \sum_{i=k}^{\infty} Q_e e[i]^2 + \Delta x[i]^T Q_x \Delta x[i] + R \Delta u_x[i]^2 \tag{3.14}$$

Q_e is the error gain, Q_x a symmetric non-negative definite matrix (typically just a diagonal matrix) to weight the components of $\Delta x[i]$ differently and $R > 0$. Conviently Katayama also derived an optimal controller for this performence index, which is given by:

$$u[k] = -G_i \sum_{i}^{k} = 0e[k] - G_x x[k] - \sum_{j=1}^{N} G_p p_x^{ref}[k+j] \tag{3.15}$$

The gains G_i, G_x, G_p, can be derived from the parameters of the performence index. Since the calculation is quite elaborate we refer to the cited article by Katayama p. 680 for more details.

3.2 Implementation

To generate walking patterns based on the ZMP preview control methode, the approach from Kajita was implemented in a shared library. A front-end was developed to easily change parameters, visualize and subsequently export the trajectory to the MMM format. The implementation was build on a previous implementation, which was refactored, extended and tuned with respect to results from the dynamics simulation.

block diagramm of architechture

The pattern generator makes extensive usage of Simoxs VirtualRobot, for providing a model of the robot and the associated task of compting the forward- and inverse kinematics.
Generating a walking pattern consists of multiple steps:

1. Generate foot trajectories: `FootstepPlaner`
2. Generate reference ZMP trajectory: `ReferenceZMPPlaner`
3. Compute resulting ZMP and CoM trajectories: `ZMPPlaner`
4. Compute inverse kinematics: `WalkingIK`
5. Exporting or visualizing the trajectory: `TrajectoryExporter`

Each step is contained in dedicated modules that can be easily replaced, if needed. We will outline the implementation of each module seperately.

3.2.1 Generating foot trajectories

To generate the foot trajectories several parameters are needed:

- Step length
- Step width (the distance between booth TCP on the feet)
- Duration of the single support phase
- Duration of the dual support phase

Implementation Dynamic simulation

4 Controllers to stabilize a trajectory

Theory Implementation Evaluation

5 Push recovery

Theory Implementation Evaluation

6 Results

Make sure that somewhere, either here or in the evaluation sections, you show how you plotted the desired vs. real zmp, even the phantom robot if you want, and the graphics that you generated.

7 Conclusions

things to improve summary of work done and results

Bibliography