Omnidirectional Walking using ZMP and Preview Control for the NAO Humanoid Robot

Johannes Strom, George Slavov and Eric Chown

Bowdoin College

Abstract. Fast-paced dynamic environments like robot soccer require highly responsive and dynamic locomotion. We present an implementation of an omnidirectional ZMP-based walk engine for the Nao robot. Using a simple inverted pendulum model, a preview controller generates dynamically balanced center of mass trajectories. To enable path planning, we introduce a system of global and egocentric coordinate frames to define step placement. These coordinate frames allow translation of the CoM trajectory, given by the preview controller, into leg actions. Walk direction can be changed quickly to suit a dynamic environment by adjusting the future step pattern.

1 Introduction

Robust locomotion is crucial to effective soccer play. Successful soccer players must be able to move to the ball quickly, change direction smoothly, and withstand physical interference from opponents. While concepts like omnidirectional walking, Zero Moment Point (ZMP) and constructs like preview control have been explored extensively in the biped walking literature [1, 5, 4], these discussions often gloss over the realities of implementation. Particularly, these results are often based on simulated experiments, or do not provide the detailed workings of the walk engine. In addition, a system for omnidirectional walking using ZMP and preview control has yet to be presented. This article presents a successful implementation of an omnidirectional walk engine on the Nao robot used in the Standard Platform league.

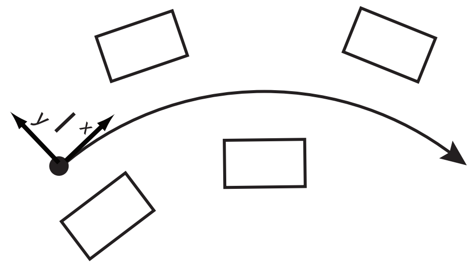

Fig. 1. A sample omnidirectional footstep pattern generated from a constant motion vector (x, y, θ) that in this case, has both a forward and rotational component.

1.1 Overview

Omnidirectional walking is crucial in soccer, since a soccer player is constantly changing her objective in a quickly changing environment. Other methods, such as the capability to walk in preset directions which ships with the Nao robot, are not adequate for playing soccer because they do not allow fine-grained control over the direction of motion. The preset trajectories can walk straight, to the side, or turn but cannot combine the three.

Our walk is omnidirectional because we have the capability to place footsteps in any position and orientation: given a desired direction of motion, each successive step is placed along this direction (Figure 1). Given a pattern of steps, a preview controller can use the ZMP balance criterion (discussed in sections 3 and 4) to generate a motion of the center of mass which maintains dynamic balance during the execution of the footsteps [5]. Finally, using the locations of the footsteps and the center of mass trajectory, the motions of the legs can be calculated using inverse kinematics.

ZMP-based approaches to walking that consider the full dynamics of the robot traditionally rely on pre-calculated trajectories and are thus ill-suited for dynamic environments such as robot soccer. Alternatively, dynamically balanced walking patterns can be created at runtime using a simplified model and a preview controller. The preview controller generates valid Center of Mass (CoM) trajectories by examining future foot steps, so motion velocity cannot be changed instantaneously. A certain degree of previewing is absolutely necessary for walking, since it is impossible to change walking vectors instantaneously without falling over. The duration of the preview controller's look ahead determines explicitly which future steps can be safely replaced or updated when the motion vector changes. This allows a quick response to changes in the environment without compromising the robustness of the walk.

What follows is a discussion of each of the components of the walk engine, starting with an overview of step placement, followed by a description of the implementation of a preview controller using the ZMP balance metric, finishing with a discussion of our inverse kinematics system. A schematic overview of the system is shown in Figure 2.

2 Omnidirectional Step Placement

The implementation of an omnidirectional walking system is non-trivial. Translating a series of steps to joint angles requires many layers of abstraction in order to build a well designed system. The central parts of this abstraction are the four homogeneous coordinate frames which we define to allow each part of the system to be expressed in the simplest possible terms (See Table 1, Figure 3 and the following sections for details). The coordinate frames allow step planning, step execution and leg control to be expressed in their natural frame of reference. This ensures that the system stays manageable because each component only acts on a limited amount of information anchored to its appropriate coordinate frame.

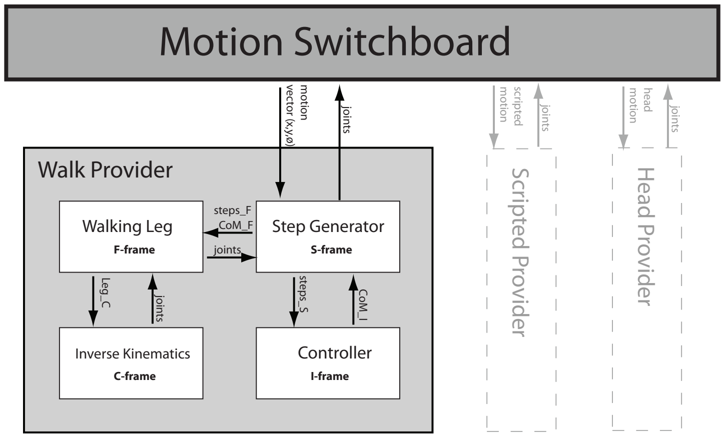

Fig. 2. An overview of the motion architecture. A switchboard manages many modules seeking to provide motion functionality. The walk provider provides the robot with walking capability, while the scripted provider enables execution of scripted motions. The four main components of the walk provider correspond with the four coordinate frames discussed in section 2.

Since each coordinate frame corresponds directly to one of the components of the implementation (see Figure 2), we list the corresponding part of the architecture in brackets in the section headings for the relevant coordinate frame below.

To translate between each coordinate frame, we maintain some transformation matrices which can be applied to move motion trajectories from one coordinate frame to the next (See Appendix A for details). Although matrix multiplication can incur a heavy computational load, they reduce human error and improve maintainability by reducing the complexity of the system. In addition, the matrices are small (3x3), and many are updated only once every walking step – only one matrix must be updated each time step. One alternative to our approach would be to specify the entire walking motion of each leg as a locus relative to the body's CoM. This removes the need for many matrix translations, but the process of integrating the controller is no longer well defined. Additionally, under such a model, omnidirectional walking is very complex. The small overhead potentially incurred by the coordinate transformations is worth avoiding the complexity needed to design the system another way.

2.1 Steps in the S frame [StepGenerator]

During the walking process, irrelevant steps are discarded and new steps must be planned in the future (as required by the preview controller). Each successive step is generated from the currently desired walk vector in the S frame as shown in Figure 4. The S frame is always offset by H_O towards the CoM from the F frame (See Table 1). Defining steps in this manner allows step planning without needing to consider any history of steps. After

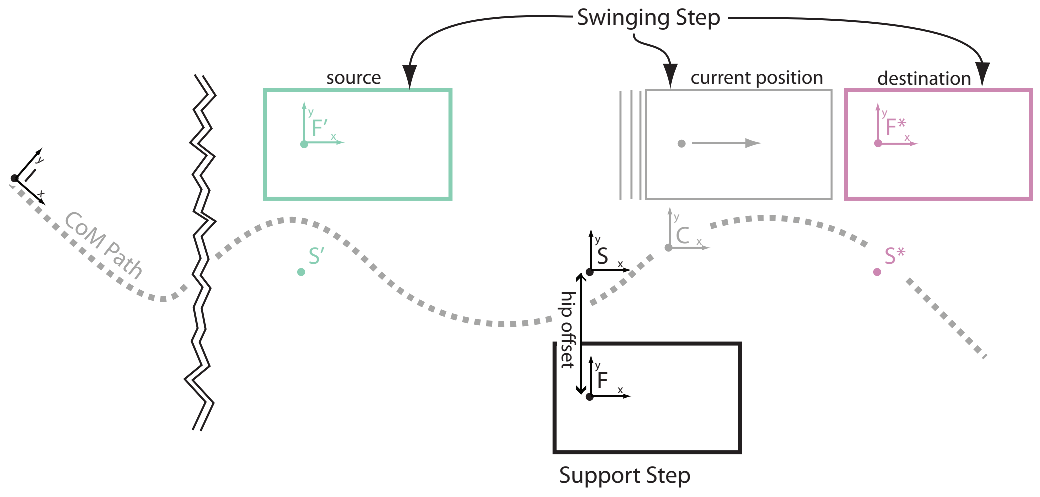

Fig. 3. The various coordinate frames are shown in single support mode while the right leg is supporting and the left leg is swinging from its source to its destination. The support foot is always anchored at F. The previous F coordinate frame is F', and the next one will be F* once the swinging leg arrives at its destination and becomes the next supporting foot. The I coordinate frame is located at the initial starting position of the robot, and does not depend on the footsteps shown above.

each step, the S frame moves to the inside of the next support step, so it is easy to chain multiple steps together.

2.2 CoM trajectories in the I frame [Controller]

Using the steps defined in the S frame, the preview controller can calculate the optimal CoM posture which will keep the robot balanced. Since the controller operates in the I coordinate frame, we maintain a transformation matrix from the current S frame to the I

Table 1. The four coordinate frames necessary for specifying step placement and leg movements of the robot. H_O is the 50 mm horizontal offset between the CoM and the hip joint. Some of the frames move with the robot and must be updated at various intervals.

Coordinate Frame	**Anchor**	**Updated**
C (Center of Mass)	CoM	Every motion time step
F (Foot)	Support Foot	When switching from single to double support
S (Step)	F $\pm H_O$	When switching from single to double support
I (Initial)	World	Never

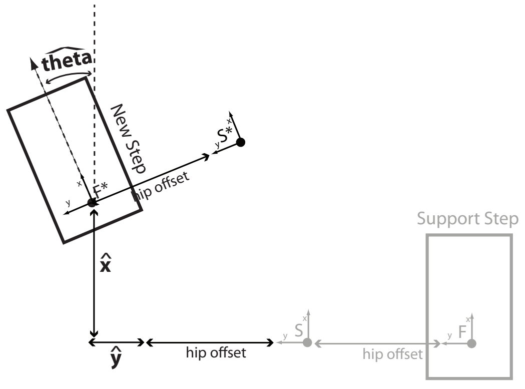

Fig. 4. An additional step being added using a motion vector $(\hat{x}, \hat{y}, \hat{\theta})$.

frame that gets updated each time a new future step is created. [1] A more detailed discussion of the controller is in section 4.1.

2.3 Leg trajectories in the F frame [WalkingLeg]

Trajectories for each leg can be expressed simply in the F frame. Since the F coordinate frame is anchored to the support foot, it provides a consistent frame of reference to define the motion of both legs regardless of how many steps have already been taken. In the F frame the support foot's position always remains at the origin by definition. For the other (swinging) leg, the motion is interpolated between the swinging source and the swinging destination (See Figure 3.) We use a cycloid function to generate a smooth stepping motion which has zero velocity when the foot begins to lift and when it arrives at the destination (inspired by the walk Aldebaran Robotics ships with the robots [7]). In order to eventually specify the motion of the legs in the C frame, we maintain a second transformation matrix from I to F, which is updated at the beginning of each new walking cycle.

2.4 Leg trajectories in the C frame [Inverse Kinematics]

Once the leg trajectories are known in the F coordinate frame, they are translated into the C coordinate frame with another transformation matrix. Since the CoM is always moving, this matrix is recalculated each time step from the I to F transformation matrix and the current position and rotation of the CoM in the F frame. The position is easily obtainable

[1] The controller runs in a static coordinate frame because if the coordinate frame of the controller were to move with the robot (like the F frame, for example), each of the previewed ZMP values would need to get translated as well, which is expensive. Instead the cost is only that of updating a matrix once per step. The only danger is overflowing the float type, but this will only happen after 500m of walking in a single direction, which is not possible on a soccer field.

from the controller - the current rotation is stored as the robot rotates the support foot relative to the CoM. Targets for the legs can be translated from the C frame into joints using inverse kinematics and the body height z_h (see sections 3 and 4.3).

2.5 Turning

Planning the turning motion on the Nao robot is more complex than on a standard humanoid because each hip does not have 3 linearly independent actuators [7]. Each hip has a pitch (Y-axis) and roll (X-axis) actuator, but both hips share in common a transverse yaw-pitch (ZY-axis) actuator. Without this extra joint, turning is impossible, since the legs are unable to rotate around the Z axis with respect to the body.

To achieve turning, we manually control the hip-yaw actuator to rotate the swinging leg open with respect to the support foot during one step, and then close it again on the next step. By alternating these types of steps, we can induce a good turning motion during a sequence of steps (see Figure 4). Using inverse kinematics, we are able to compensate for the Y-axis hip rotation introduced by employing the yaw-pitch actuator (see Section 4.3).

3 Modeling the Dynamics of the Robot

Given a series of steps, our goal is to specify a trajectory for the CoM which will allow the robot to remain upright and balanced. A good criterion for determining whether a robot will fall is the Zero Moment Point, which is widely used in biped walking [10, 5, 4]. To simplify the calculation of the ZMP, we follow [5] to simplify the dynamics of the robot by modeling it as an inverted pendulum with the entire mass of the robot concentrated at the CoM.

3.1 The Zero Moment Point

The Zero Moment Point (ZMP), is the point on the support polygon of the robot where the moments acting on the robot are balanced by an opposing moment from the ground [10]. When this point exists (i.e. it is inside the support polygon), then the robot will not rotate about the edges of the foot and will remain upright. Given a pattern of steps, we are thus interested in defining the motion of the robot such that the ZMP always remains near the center of the robot's supporting foot during single support. In double support, when both feet are in contact with the ground, our aim is to quickly pass the ZMP to the other foot.

3.2 Cart Table/Inverted Pendulum Model

Calculating the ZMP of the robot using its full dynamics is computationally intensive and not suitable for online computation. Instead, we simplify the model of the robot as an inverted pendulum [5].

This model allows the ZMP in one dimension to be calculated easily from the position and acceleration of the center of mass of the robot:

$$p = x - \frac{z_h}{g}\ddot{x} \tag{1}$$

Where x is the position of the CoM, $\ddot{x}$ is its acceleration, z_h is the constant height of the CoM from the ground, and g is 9.81, the magnitude of gravity.

This simplification has obvious drawbacks, since it does not account for the complete dynamics of the robot. However, these simplifications can be dealt with in the controller by incorporating sensor feedback, as in [4] – see Section 5.

4 Controlling the Dynamics of the Robot

In the previous section, we discussed how to model the robot in order to use the ZMP as a balance metric. Though this model allows us to discover if a certain movement will maintain the robot's dynamic balance, it will not allow us to calculate motions which are inherently balanced. In fact, what we need is an *inverse* to the ZMP equation above, which is not symbolically obtainable [5]. To calculate the inverse numerically, we can use a preview controller which is able to generate motions which result in a specified ZMP trajectory.

4.1 Preview Control

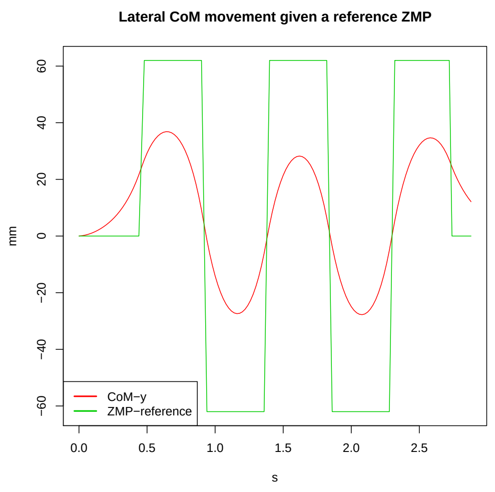

Fig. 5. Lateral CoM movement given a reference ZMP plotted during a sequence of five steps.

Kajita proposes to solve the inverse by casting the problem as a servo control problem using preview control [5]. A preview controller acts on a ZMP reference function $p_{ref}(k)$, which defines the location of the desired ZMP at time $k\Delta t$, where Δt is the duration of a motion time step. $p_{ref}(k)$ is determined by ensuring the ZMP remains over the support foot during a series of steps (see section 4.2). The state of the robot is modeled in one dimension using its position, velocity and ZMP as $[x, \dot{x}, p]^t$. The controller works to converge p, from the state vector, with p_{ref} given by the reference function. Given proper preview values, two controllers can work in parallel to generate the states needed to follow the 2-dimensional reference ZMP necessary for walking. This is effectively an inverse to the ZMP equation, (1). One controller in the lateral direction is shown in Figure 5.

The preview control state update is given by

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{b}u(k) \tag{2}$$

Where the optimal system control is given as:

$$u(k) = \sum_{j=1}^{N} G_d(j) p_{ref}(k+j) \tag{3}$$

Where N is the number of previewed time steps. $\mathbf{A}$ is the state matrix, $G_d(j)$ is the preview gain function, and $\mathbf{b}$ is a constant vector. The optimal values for these are defined in [5, 4, 6, 9]. The process for numerically obtaining them relies on an off-line numerical computing environment such as Matlab or Scilab. We believe the following additional references would be useful in this pursuit [9, 2].

4.2 Computing ZMP reference from Steps

A crucial part of the preview controller is previewing the reference ZMP in the future. In order to generate the reference ZMP, we turn each desired step into a sequence of reference ZMP values. As mentioned in section 3.1, when the robot is in single support, we desire the ZMP to rest in the center of the foot - when the robot is in double support, we want to quickly pass the ZMP to the next foot, before beginning to lift the new swinging leg. Though [4] uses a Bézier curve to have a smooth reference ZMP passing between support feet during double support, we have found that a simple linear interpolation from one foot to the next is sufficient, since the controller naturally smoothes the state transitions. The preview values are initially expressed in the S frame since they are generated from steps, but are then translated into the I frame for use in the controller. To facilitate this, we maintain another transformation matrix which is updated each time a future step is generated.

4.3 Kinematics

The final component of controlling the robot is translating leg trajectories from the C frame into joint angles which can be sent to the actuators. This process is called inverse kinematics, but is often used implicitly in the literature with little or no explanation. The method we use is iterative. That is, an initial solution is improved by perturbing the joint angles until the error between the goal (x, y, z) of the end-effector and its current position, which is calculated through forward kinematics, is minimized. The initial solution is the current joint angles of the robot. This is convenient because the net change between subsequent goals of the end-effector during walking is very small.

Let J be the joint space and $\mathbb{R}^3$ be the 3D space. Then we can define forward kinematics as a function $f : J \rightarrow \mathbb{R}^3$ which takes a set of joint angles $\boldsymbol{\theta} = (a_1, a_2, \ldots, a_6)$ to the position of the end-effector in space (x, y, z). This function would normally be defined using a set of linear transformations defined by the modified Denavit Hartenberg convention [8]. We used *Mathematica* to symbolically perform the matrix algebra for forward kinematics for a generic set of joint angles. Evaluating the resultant expression at a $\boldsymbol{\theta}$ is significantly faster than performing the matrix algebra that would otherwise be necessary.

In order to minimize the number of iterations in the algorithm, we follow the path of largest decrease in the error. First we define a vector $\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{s}$ where $\boldsymbol{t}$ is the target and $\boldsymbol{s}$ is the current position of the end-effector. $\boldsymbol{e}$ is then the desired direction in $\mathbb{R}^3$. What we need is the desired direction in J. A Jacobian matrix, denoted by M, helps us accomplish this change of variables. Each partial derivative in M is evaluated at the current iteration's $\boldsymbol{\theta}$. x, y and z are the components of the output of f and can be viewed as functions of $\boldsymbol{\theta}$ as well. These partial derivatives were computed symbolically using *Mathematica.* Only their evaluation is performed online. We then have an expression ready for the calculation of $\Delta\boldsymbol{\theta}$ which is the desired amount of perturbation in each joint value.

$$\Delta\boldsymbol{\theta} = (M^t M + \lambda^2 I)^{-1} J^t \boldsymbol{e} \tag{4}$$

λ is a dampening factor which increases the number of iterations required for the algorithm to converge but increases numeric stability. $\Delta\boldsymbol{\theta}$ is clipped because it only provides a direction, not a magnitude. Thus, the final part of the algorithm is deciding how long it should follow this direction in this iteration. Our implementation of this algorithm uses $\lambda = 0.4$ and $max\Delta\boldsymbol{\theta} = 0.5$ radians. These were chosen through trial and error be selecting for quick convergence. The latter is used as a maximum for each component of $\Delta\boldsymbol{\theta}$. We can perform tens of thousands of calls to inverse kinematics in a second, so it does not present a significant efficiency bottleneck. For balanced walking, we also impose a second condition that the foot remain parallel to the ground. We accomplish this by splitting the leg chain into two end-effectors: the ankle and the heel, each with its separate goal. The algorithm achieves high levels of both accuracy and precision. A more detailed presentation of this approach is presented here [3].

As discussed in Section 2.5 dealing with the peculiar kinematics of the Nao robot is necessary to achieve turning motion. Since the addition of the yaw-pitch actuator as a variable to the inverse kinematics algorithm greatly increases the number of possible joint combinations for any given (x, y, z) end effector target, we hold the yaw-pitch joint constant during the iterations. This allows explicitly setting that actuator as required by the turning algorithm, but also results in more regular movements of the legs, since the algorithm keeps the thigh and shin generally aligned to the ground.

4.4 From Theory to Practice

A crucial part of our implementation was bridging the gap from theory to practice. Imperfections such as asymmetry in the robot's joints can cause us to fall. To compensate for this we introduced some adjustments in addition to the core parameters of the walking engine. The most important adjustment we made was inspired by the walk Aldebaran ships with the robots. The actuators struggle to give enough power to the hip joints in order to lift the swinging leg from the ground. To compensate for this, we gradually add a sinusoidal offset to each individual hip-roll (hip lateral swing) joint during the swinging phase. Since the offset is distinct for each hip, this provides considerable help in offsetting asymmetries in the robots, which may have slightly stronger left or right legs.

In addition, extreme values of this adjustment can be used to lift the feet from the ground. By lifting from the hips, the robot does not build up downward momentum with its feet as it steps down, and thus experiences a much smoother gait.

Another offset we introduced helped balance the robot by moving the reference ZMP laterally away from the inside of the foot, inducing a greater hip swing. This helps to compensate for the simplicity of the inverted pendulum model.

The final breakthrough we had was to dramatically reduce the duration of the steps, as well as to reduce the portion of the step spent in double support. By doing this, the robot was able to balance better and move faster because the magnitude of the hip swing was reduced.

5 Results

Using our system of coordinate frames coupled with preview control, we are currently able to achieve maximum forward walking speeds of 10.5 cm/s, which is comparable to the maximum walk speed of the Aldebaran walk engine. However, at such speeds, the robot is not very stable. In practice, we prefer a gait which has a maximum speed of 7 cm/s, with a step frequency of 1Hz, which is much less prone to falling, even during large changes in the motion vector.

We have also been able to extend the preview controller with an observer as described in [4]. However, estimating the sensor ZMP from the accelerometers while minimizing the lag time is non-trivial. In practice, using an observer informed by lagging, noisy sensor values adds instability to the walk even while visibly controlling larger disturbances. This trade-off makes the closed loop controller perform mostly on par with the open loop one. Further refinement of sensor based state estimation is being actively researched.

Videos of our implementation can be found on our team's blog[2].

The code implementation of our system, written in C++ using Boost, is publicly accessible under the LGPL using using `git`[3]. However, as of this writing, no stable release candidate has yet been designated.

6 Conclusion

Since humanoid robots are best suited to coexist with humans, there is an increasing emphasis on humanoid robots. In RoboCup, this reflects the desire to compete on even terms with humans. A critical part of that competition will rely on developing motion engines which are at least as quick and agile as humans are. Among the necessary advances are developing motion systems capable of executing omnidirectional motion in real time. This paper provides an implementation of omnidirectional walking which will serve to help those who are arriving in this field for the first time. Furthermore, it attempts to fill in some of the gaps which have been left open by other papers in the field (particularly in the implementation, and testing on real robots). The elegant nature of the preview control comes with some draw backs due to its computational simplicity, however, they can theoretically be overcome using the observer.

References

1. Sven Behnke. Online trajectory generation for omnidirectional biped walking. *2006 IEEE International Conference on Robotics and Automation*, 2006.
2. P. Benner and V. Sima. Solving algebraic riccati equations with slicot. In *Proceedings of the 11th Mediterranean Conference on Control & Automation MED'03, June 2003*, 2003. http://www-user.tu-chemnitz.de/ benner/pub/med03.pdf.
3. Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. http://math.ucsd.edu/ sbuss/ResearchWeb/ikmethods/iksurvey.pdf, April 2004.
4. Stefan Czarnetzki, Sören Kerner, and Oliver Urbann. Observer-based dynamic walking control for biped robots. *To appear in Elsevier*, 2008.
5. Shuuji Kajita, Fumio Kanehiro, Kenjio Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, , and Hirohisa Hirukawa. Biped wlaking pattern generation using preview control of zero-moment point. In *International Conference on Robotics and Automation; Proceedings of the 2003 IEEE*, 2003.
6. Tohru Katajama, Takahira Ohki, Toshio Inoue, and Tomoyuki Kato. Design of an optimal controller for a discrete-time system subject to previewable demand. *International Journal of Control*, 41(3):677–699, 1985.
7. Aldebaran Robotics. www.aldebaran-robotics.com.
8. Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics.* Spring, 2008. http://books.google.com/books?id=Xpgi5gSuBxsC.

[2] http://robocup.bowdoin.edu/blog
[3] http://github.com/northern-bites/nao-man.git

9. George Slavov. The math of dynamically balanced humanoid locomotion, 2009. Senior Thesis, Bowdoin College.
10. Miomir Vukobratović and Branislav Borovac. Zero moment point – thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.

A Translating between coordinate frames

The four key coordinate frames used to generate walking motions are discussed in section 2. The matrices to do the transformations are given below

$$
\begin{gathered}
\mathbf{T_{if}}(n) = \mathbf{F_n} \times \mathbf{F_{n-1}} \times \cdots \times \mathbf{F_2} \times \mathbf{F_1} \\
\mathbf{F_i} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \pm H_O \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & -s_x \\ 0 & 0 & -s_y \\ 0 & 0 & 1 \end{bmatrix} \\
\begin{bmatrix} x_f(n\Delta S + t) \\ x_f(n\Delta S + t) \\ 1 \end{bmatrix} = \mathbf{T_{if}}(n) \begin{bmatrix} x_i(n\Delta S + t) \\ y_i(n\Delta S + t) \\ 1 \end{bmatrix} \\
\mathbf{T_{fc}}(n\Delta S + t) = \begin{bmatrix} \cos(\phi(n\Delta S + t)) & -\sin(\phi(n\Delta S + t)) & 0 \\ \sin(\phi(n\Delta S + t)) & \cos(\phi(n\Delta S + t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & -x_f(n\Delta S + t) \\ 0 & 0 & -y_f(n\Delta S + t) \\ 0 & 0 & 1 \end{bmatrix} \\
\begin{bmatrix} destx_c \\ desty_c \\ 1 \end{bmatrix} = \mathbf{T_{fc}} \begin{bmatrix} destx_f \\ desty_f \\ 1 \end{bmatrix}
\end{gathered}
\tag{A-1}
$$

Where $\mathbf{T_{if}}(n)$ is the transformation matrix between coordinate frames I and F after n steps. H_O is the horizontal offset between the CoM and the hip joint, and $\mathbf{F_i}$ is the matrix to translate from the F(i-1) coordinate frame to the next F(i) coordinate frame given the ith step (s_x, s_y, θ). $(x_i(n\Delta S + t), y_i(n\Delta S + t))$ is the position of the CoM in the I coordinate frame at time t after the nth step was started (ΔS is the duration of a step). $\phi(n\Delta S + t)$ is the rotation of the center of mass at time t between the C frame and the F frame. $\mathbf{T_{fc}}$ is the transformation matrix between the F and C coordinate frames, and $destx_c, desty_c$ is the destination of a heel in the c coordinate frame.