Dynamically Balanced Omnidirectional Humanoid Robot Locomotion

An Honors Paper for the Department of Computer Science

By Johannes Heide Strom

Bowdoin College, 2009

Contents

List of Figures

Abstract

Fast-paced dynamic environments like robot soccer require highly responsive and dynamic locomotion. This paper presents a comprehensive description of an omnidirectional dynamically balanced humanoid walking engine for use in the RoboCup Standard Platform League on the NAO robot. Using a simplified inverted pendulum model for Zero Moment Point (ZMP) estimation, a preview controller generates dynamically balanced center of mass trajectories. Preview control requires a limited degree of future path planning. To this end, we propose a system of global and egocentric coordinate frames to define step placement. These coordinate frames allow translation of the Center of Mass (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. For each component of the walking system, this paper explains the concepts necessary for reimplementation. This report is also a useful reference to help in understanding the code which was released under the GNU Lesser General Public License (LGPL) as a result of this project.

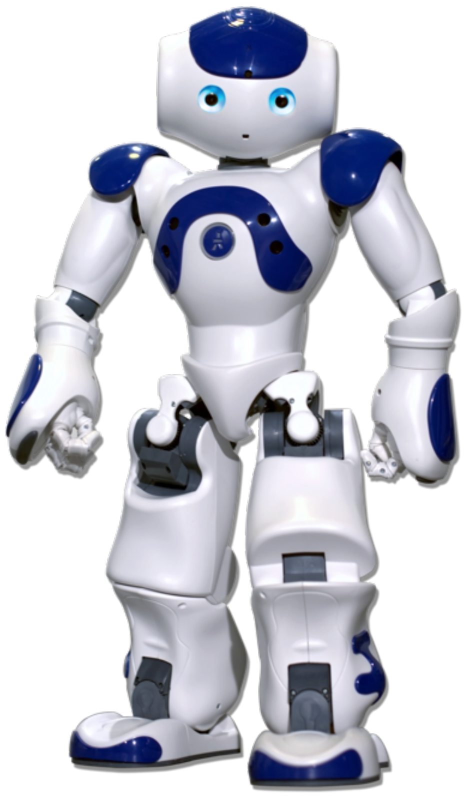

Figure 1: The NAO robot from Aldebaran-Robotics [12].

1 Introduction

Robust biped locomotion is crucial to effective soccer play for humanoid robots. Successful soccer players must be able to move to the ball quickly, change direction smoothly, and withstand physical interference from opponents. For humans, even when aided by millions of years of evolution, this task takes several years to perfect.

In modern robotics, the notion of balanced humanoid robot locomotion is only four decades old. The first humanoid walkers were implemented on small proof-of-concept robots [14]. It wasn't until the late 1990's when high-profile robots like Asimo and Qrio were developed in the R&D laboratories of multi-national corporations of Honda and Sony [6,7]. Although these modern robots garnered considerable coverage in the media, little was ever published about the inner functioning of their walk engines.

Very recently, widespread investigation of humanoid walking is made possible by the emerging presence of much less expensive entertainment robots, such as the Aldebaran NAO, which cost less than $5,000. The NAO robot is a kid-size humanoid robot produced by Aldebaran Robotics in Paris, France. It stands 57 centimeters tall, and weighs 4.4 Kilograms. It has 21 actuated joints, two 30 Hz VGA video cameras and a 500 MHz Geode processor (see Figure 1). Because the release of this robot into research laboratories is so recent, there is still relatively little published in the literature about the practical details of implementing a walking engine on a real Nao robot.

Our main motivation for developing a humanoid walking engine is to use it as part of the Northern Bites robot soccer team in the RoboCup Federation's Standard Platform League [10]. RoboCup is an international collaboration of researchers created to drive forward the state-of-the-art in artificial intelligence and robotics by providing a practical proving ground for theoreies in areas such as robot navigation, locomotion and cooperation, among others. RoboCup fosters several leagues which compete annually. In the Standard Platform League, all teams use standard hardware produced by Aldebaran Robotics. Since we are interested in having a successful robot soccer team, fast and balanced locomotion is an absolute necessity.

At the highest level, humanoid walking can be simply conceptualized as specifying the desired direction of motion – left, right, forward, or any such combination. Alternatively, the highest level could also be specified as the locations for a sequence of desired steps (e.g. when crossing a river using stepping stones). At the lowest level, the problem is to find the correct sequence of joint angles which will result in a walking motion that remains upright.

Not surprisingly, trying to solve this problem solely at either level is impossible. The sequence of joint angles necessary to maintain motion depends heavily on the stance of the robot, and there is no trivial modification to these angles which will compensate for external forces. In reality, several layers of abstractions need to work together to connect the highest and the lowest level, and therein lies the bulk of the problem:

How to translate high level motion objectives into a sequence of joint waypoints resulting in a dynamically balanced walk?

The rest of this paper will describe various concepts and abstractions which connect these two layers. Although there are many paradigms for generating humanoid gaits, not all of them are particularly suited to playing soccer. Of particular importance is the design of a clean system which is conceptually understandable at all levels, which is easily maintainable, can run in realtime on the robot and which most importantly can maintain balance while walking quickly in a dynamic environment.

1.1 Related Work

Although many researchers from around the world ([5], [4], [2]) have successfully demonstrated walking algorithms on humanoid robots, a robust solution which works independent of hardware configuration does not yet exist. In particular, both Kajita and Czarnetzki's work gloss over some crucial details of the implementation, and only provide results based on simulated experiment. Even though Czarnetzki's [4] method was demonstrated on simulated NAO robots, the lack of a detailed description of all parts of the approach make re-implementation non-trivial. We are thus left to craft our own solution which will work on real robots, filling in the unpublished practical details of implementation.

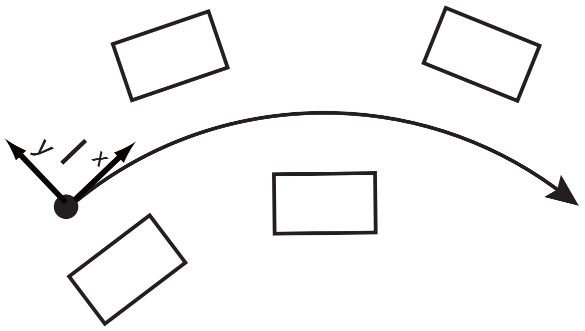

Figure 2: A sample omnidirectional footstep pattern generated from a constant motion vector (x, y, θ) that, in this case, has both a forward and rotational component.

Both Czarnetzki and Kajita [4], [5] provide the core of a control theory approach to walking using the Zero Moment Point (ZMP) balance criterion and a preview controller. By calculating the location of the ZMP, it is possible to tell whether a robot is balanced. The ZMP concept can thus be used to guide the design of a balanced walk. The ZMP is discussed in more detail in section 3.1. The preview controller determines the motion of the Center of Mass (CoM) which keeps the robot balanced according to the desired location of the ZMP.

Behnke [2] introduces an omnidirectional gait for use with the Humanoid League in RoboCup, where movement of the robot's limbs is defined by manually tuned trajectories, without regard for the balance of the robot. An omnidirectional gait allows motion in an arbitrary direction, parameterized by (x, y, θ), where x is forward velocity, y is lateral velocity, and theta is angular velocity about the vertical axis (see Figure 2).

Tuning a manual configuration like the one presented by Benke requires tinkering with a complex system of interdependent constants, without any theoretical guarantee of maintaining the robot's balance. Furthermore, maintaining per-robot tunings of a gait is made more difficult by this approach. Instead, a system using the balanced-based ZMP method can be built using conceptually coherent layers, while simultaneously expressing the constraint that the robot must maintain its balance. This approach also increases the ease with which sensor readings of the robot's balance can be used as feedback to keep the robot from falling. However, a major drawback of the ZMP-based approaches introduced by Czarnetzki and Kajita is that the necessary layers which bridge the ZMP-based balance control with the actual motion of the robot's limbs are not discussed in sufficient detail for re-implementation.

This paper builds on the previously published control theory and ZMP based approaches by presenting in detail an implementation of an omnidirectional walk engine on the NAO robot used by the Northern Bites team in the Standard Platform League of RoboCup.

1.2 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. Furthermore, the preset trajectories are unable to react to external disturbances.

The walk presented here is omnidirectional because it has the capability to place footsteps to execute movement in an arbitrary direction (see Figure 2). Given a pattern of steps, a preview controller can use a simplified ZMP balance criterion (discussed in sections 3 and 4) to generate a motion of the center of mass (CoM) 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.

Computationally complex, non-simplified ZMP-based approaches to walking that model 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 ZMP model and a preview controller. The preview controller generates valid CoM trajectories by examining the balance criteria necessary to execute future foot steps. A certain degree of previewing is required for walking, since it is impossible to change walking direction 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 desired direction of motion changes. This allows a quick response to changes in the environment without compromising the robustness of the walk.

The advantage of using the preview control paradigm is that it is much easier to explicitly and correctly introduce sensor-based stabilization into the control loop. To incorporate sensor feedback into the walk requires the addition of an observer to the preview controller [4]. The observer determines the correct response to differences between the expected balance and pose of the robot and the balance measured using the robot's accelerometers. Including sensor feedback closes the control loop, resulting in a "closed loop" 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 3.

1.3 Notation

For clarification, we describe here the notation that we will use consistently throughout this paper.

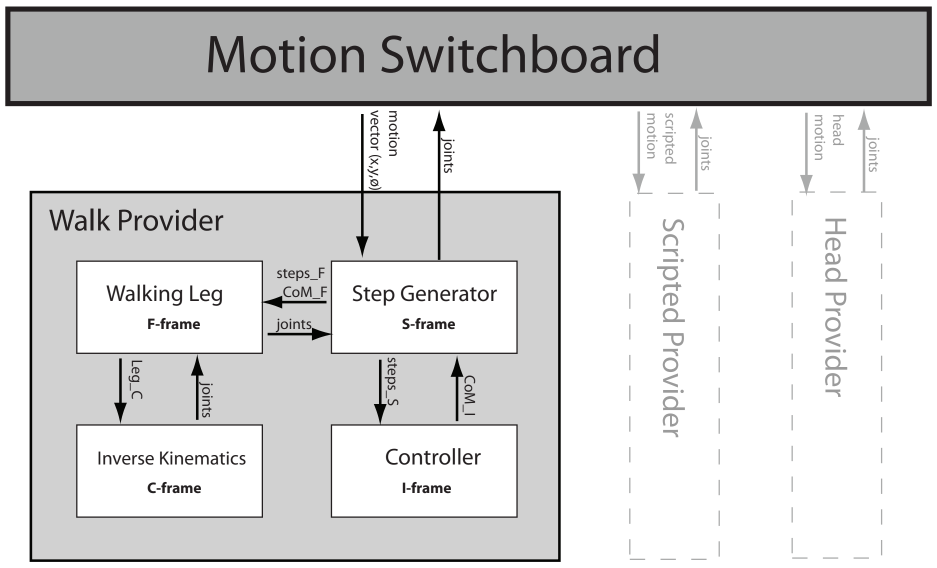

Figure 3: An overview of the motion architecture. A switchboard manages many modules seeking to provide motion functionality. The walk provider provides the robot with walking capability, while the scripted provider enables execution of scripted motions. The four main components of the walk provider correspond to the four coordinate frames discussed in section 2.

- Matrices are indicated by a bold capital letter: $\mathbf{B}, \mathbf{M}, \mathbf{W}, \mathbf{A}$, etc.
- Column vectors are indicated by a bold lower-case letter: $\mathbf{x}, \mathbf{v}$, etc.
- Row vectors are indicated by a transposed bold lower-case letter: $\mathbf{c}^T$.
- Scalar quantities (numbers or functions that return numbers) are indicated by a non-bold character: $g, z_h, p_x, G_d(i)$, etc.

1.4 Glossary

The following are brief definitions of some terms which are central to the method described in this paper.

- **Cart and Table Model:** A simplified model of the robot which assumes the entire mass of the robot is concentrated at its center of mass. See section 3.2.
- **Center of Mass (CoM):** The location of the center of mass of the robot. When considering the full dynamics, this is the weighted average of all the locations of the masses of each link in the robots kinematic configuration.
- **Closed Loop:** A control system is said to be closed when the output of the system is fed back as an input. In this case, the walk controller is considered to be "closed loop" when the feedback from the robot's sensors is used as input to the controller.

- **Degrees of Freedom (DoF):** The NAO has 21 degrees of freedom, which means that there are 21 distinct actuated axes of rotation which can be used to control the robot's 21 joints.

- **Double Support:** The portion of the gait cycle when both feet are in contact with the ground during which the support foot is switched.

- **Forward Kinematics:** The process of determining the location in 3-space of the end of one of the robot's limbs given the angles at each joint.

- **Gait Cycle:** One gait cycle is the combination of one single support and one double support phase.

- **Hip Offset (H_O):** The horizontal distance between the center of the robot and the center of the robot's leg. For the NAO, this is 5 centimeters.

- **Homogeneous Coordinates:** A linear algebra convention for expressing coordinates to enable translating and rotating the coordinate using matrix algebra. In 3-space, the point (x, y, z) is represented as the vector $[x, y, z, 1]^T$, or more generally as $[ax, ay, az, a]^T$, where a is an arbitrary constant.

- **Inverse Kinematics:** The process of determining the joint angles required to move one of the robot's limbs to a specific location in 3-space.

- **Joint Chain:** One of five sequences of joints in the robot which correspond to the limbs of the robot (i.e. head, arms, and legs).

- **Omnidirectional Walk:** An omnidirectional walk is a walk which has the ability to walk in an arbitrary direction of motion.

- **Preview Controller:** A controller which, by examining future desired ZMP values, determines the motion of the center of mass such that the desired sequence of Zero Moment Points is achieved, thus maintaining the balance of the robot.

- **Single Support:** The portion of the gait cycle when only one foot is in contact with the ground (the support foot). The other foot is the swinging foot, swinging from its last position as the support foot to the location where it will be the support foot again during the next gait cycle.

- **Sagittal :** In human anatomy, this is the plane perpendicular to the lateral plane, when viewed from above. It is analogous to the x direction in the robot's local frame of reference.

- **Support Polygon:** The convex hull surrounding the foot or feet which have contact with the ground plane.

- **Zero Moment Point (ZMP):** A point which is used to express the balance of the robot. The point always lies on the ground plane, and must remain inside the support polygon of the foot for the robot to avoid rotating about the edges of the foot. The ZMP is discussed in section 3.1.

2 Omnidirectional Step Placement

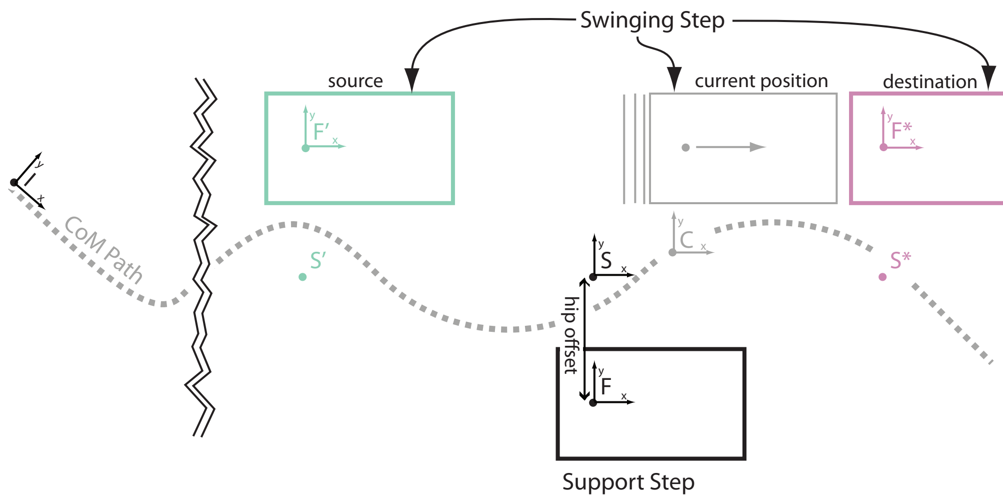

Figure 4: The various coordinate frames are shown in single support mode while the right leg is supporting and the left leg is swinging from its source to its destination. The support foot is always anchored at F. The previous F coordinate frame is F', and the next one will be F* once the swinging leg 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.

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. On the one hand, the preview controller requires input be expressed in a consistent, nonmoving coordinate frame. On the other, walking is expressed most generally (and elegantly) with respect to a coordinate frame that moves with the robot. To handle both of these requirements, we introduce four 2D 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 4 and the following sections for details). (Note: The coordinate frames discussed in this section are projections onto the ground plane. For example, the location of the swinging leg, even while it is lifted from the ground, is projected into the plane for simplicity.)

These four coordinate frames allow step planning, step execution and leg control to each be expressed in its 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. Furthermore, the coordinate frames provide an abstraction which will make it much simpler to introduce walk vectors which also contain a turning component. Since each coordinate frame corresponds directly to one of the components of the implementation (see Figure 3), the corresponding part of the architecture is listed in brackets in the section headings for the relevant coordinate frame below.

To translate between each coordinate frame, we maintain some transformation

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

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 footstep – 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 as is done in [2]. This removes the need for many matrix multiplications, 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 worthwhile to avoid the complexity needed to design the system another way.

2.1 Steps in the S frame [StepGenerator]

During the walking process, old 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 5. The S frame is always offset by H_O towards the CoM from the F frame (See Table 1). Every time the robot enters a new gait cycle, the S frame is shifted to the inside of the current supporting foot. Its location is equivalent to the ground projection of the robot's CoM when the robot is standing still. Defining steps in this manner allows step planning without needing to consider any history of steps. Since the S frame moves to the inside of the next support step after each step, it is easy to generalize the combination of multiple successive steps.

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 that will keep the robot balanced. Since the controller operates in the I coordinate frame, we maintain a transformation matrix from the current S frame

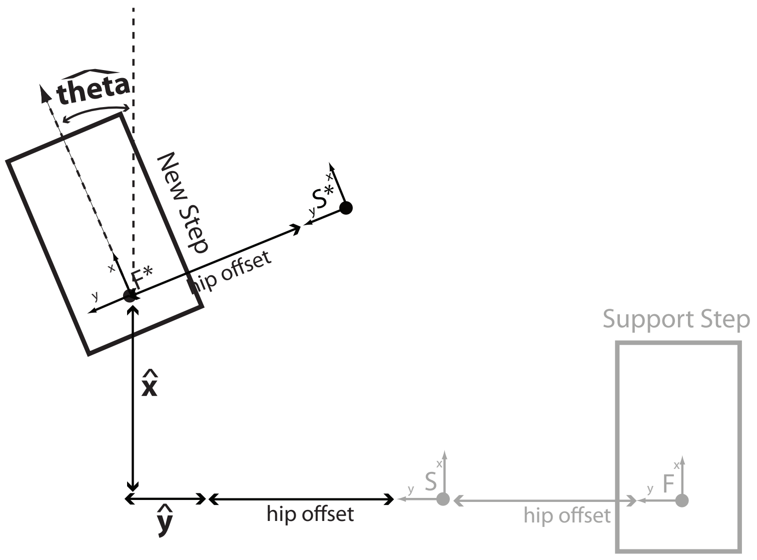

Figure 5: An additional step is being added using a motion vector $(\hat{x}, \hat{y}, \hat{\theta})$.

to the I frame that gets updated each time a new future step is created. [1] A more detailed discussion of the controller is in section 4.2.

2.3 Leg Trajectories in the F frame [WalkingLeg]

Movement trajectories for each leg can be expressed elegantly in the F frame. Since the F coordinate frame is anchored to the support foot (and temporarily to the world), the motion of the swinging leg is a simple interpolation between start position and desired end position. 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 4.) 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 [12]). 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. This enables us to translate the target destination for the CoM, provided in the I frame, to the F frame, which is tied to the support foot, and is thus much more useful to the robot who can only act in a local coordinate frame.

[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 continuous walking in a single direction, which is not possible on a soccer field. For other applications, such overflow could be detected, and the system could be reset appropriately.

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 from the controller - the current rotation is stored as the robot rotates the support foot relative to the CoM. Target destinations for the legs can be translated from the C frame into joints using inverse kinematics and the body height z_h (see sections 3.2 and4.5 for details).

2.5 Odometry

Accurate odometry information is crucial for good self-localization during soccer. Since we keep careful track of the progress of the robot relative to a fixed world frame, it is relatively easy to extend the system we use for placing footsteps to generate cumulative updates in position whenever queried by the localization system. Each odometry update provides a translation vector in lateral, saggital, and rotational planes from the C' frame at the last time the walking system was polled for an update to the current C frame. To accomplish this we keep a translation matrix to relate the C' frame to the I frame. When we get a new request, we can use the current I to F and I to C transforms together to find the amount the robot has moved and rotated. The details are left as an exercise during re-implementation.

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 ZMP, which is widely used in biped walking [4, 5, 14]. 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. This simplification is called the Cart Table Model, and is discussed in section 3.2.

3.1 The Zero Moment Point (ZMP)

As previously mentioned, we use the Zero Moment Point (ZMP) as a measure of balance for the robot. The method is described in detail in [14], but a brief explanation is provided here for completeness.

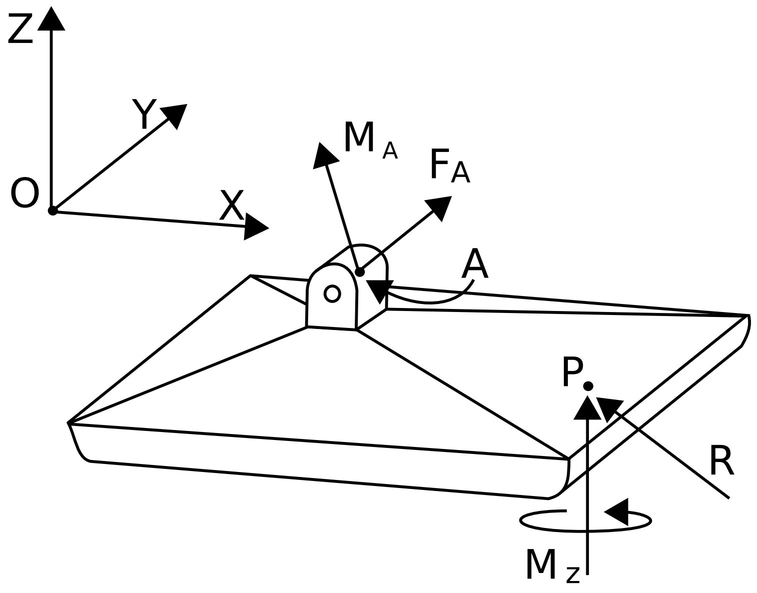

Figure 6: The forces acting on a robot's foot, from [14]. The ZMP is at point P, where the sum of the moments around the $x-$ and $y-$axis must add to zero. M_z is the component of the momentum about the vertical axis at point P, which can be non-zero. R is the sum of reactionary forces on the foot, which can be considered to act together at point P. M_a and F_a are the moments and forces acting on the ankle, point A, respectively.

The Zero Moment Point is the point P on the foot (support polygon) where we consider the sum of the ground reaction forces, R, (normal forces and friction forces) to counteract the force of gravity, F_A, and the x,y components of the moment M_A at point A, the center of the foot. (See Figure 6.) This point (P) is useful to quantify, because we are very interested in producing motions of the robot where the robot does not tip over (i.e., there are no net moments about the x or y axes).

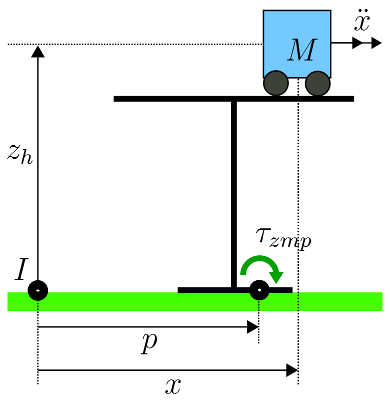

Figure 7: The cart table model for ZMP, from [5], where z_h is the constant height of the CoM, M is the total mass of the cart, p is the ZMP, and x is the position of M relative to the initial coordinate frame I. τ_{zmp} is the torque about the ZMP which must be zero when p is inside the table support.

It is important to note that the ZMP can only exist within the support polygon; otherwise, if the point is outside the polygon, it is impossible to balance the forces, and the moments are no longer zero (thus there is no more ZMP).

Based on the robot's state (position, velocity and acceleration), it is possible to calculate the location of the ZMP, and determine if it lies within the support polygon. If the calculated ZMP is outside the support polygon, then the robot is no longer dynamically balanced, and will fall.

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, following [5]. This can be conceptualized as a cart on a table (see Figure 7.) When the cart is near the edge of the table, the table will naturally tip. To avoid tipping the table, the cart can accelerate away from the center to balance the torques around the table's base. 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 (the cart):

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

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 $9.81\frac{m}{s^2}$, the magnitude of gravity.

This simplification has obvious drawbacks, since it does not account for the complete dynamics of the robot (such as massive legs). Empirically, we have found that the simplified model works fine when there are no external disturbances by tuning some parameters discussed in section 5.

3.3 Alternate Models

Other researchers have proposed extensions or replacements for the simple cart table model. Park augments the current position and acceleration based representation of the robot by also modeling the angular momentum of the CoM along two axes [11]. This method could provide stability improvements over a simple model provided here, but does incur costs associated with higher complexity, since the preview controller must manage a 5 dimensional state. A final alternative to modeling the robot would be to model the dynamics of the robot completely, without making the simplifying assumption that all the mass of the robot is concentrated at the CoM of the robot. An approach similar to this is presented in [1]. While this approach has a higher fidelity robot model, it is too computationally expensive to run on a Nao and does not lend itself to easily "closing the loop" with sensor feedback.

4 Controlling the Dynamics of the Robot

In this section, we will discuss in detail all the components of the system which actually control the motion 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 equation 1, which is not symbolically obtainable because there are an infinite number of ways that a ZMP p can be generated from variable x and $\ddot{x}$.

To calculate the (approximate) inverse numerically, we can use elements from control theory to control the robot in such a fashion that the robot maintains its balance. By specifying that the robot should maintain the ZMP in the center of the supporting foot, we can take this as the input ZMP reference (p_{ref}) for the control model. Using a preview controller, we are able to examine the desired ZMP reference values for the upcoming steps, and the preview controller will iteratively calculate the CoM location at each moment in time which will achieve the desired ZMP. On the NAO, we are able to send updated joint targets to the actuators on a 20 ms schedule, so the controller is built to provide CoM updates also at this frequency. When the preview controller is augmented with an observer, it can theoretically incorporate sensor feedback to calculate a CoM trajectory which maintains its balance even in the face of disturbances. Once the CoM path is known, we can translate the CoM path into joint angles for the robot using a process called inverse kinematics. A sample ZMP reference curve with accompanying CoM trajectory is shown in Figure 8.

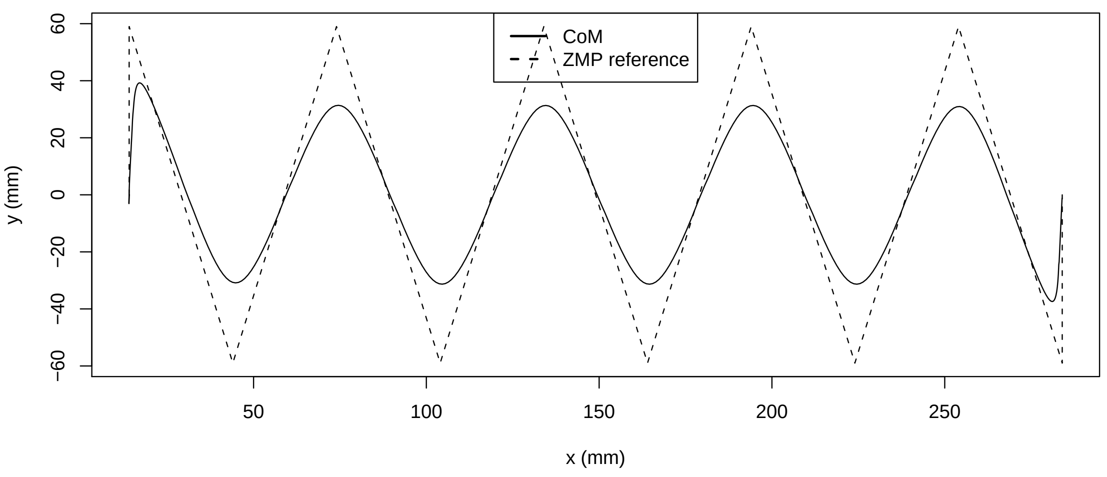

Figure 8: The two dimensional CoM movement calculated from a reference ZMP curve is shown in both x and y directions.

4.1 Computing the ZMP Reference from a Sequence of 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 shown in Figure 9. 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. One important feature of ZMP generation is deciding how far (if at all) the ZMP reference should move forward over the length of the foot during the execution of a step. In Figure 8, there is no ZMP movement along the foot, while in Figure 9 the ZMP is shown to move almost the whole length of the foot. In practice, for an omnidirectional gait, leaving the ZMP centered at the middle of the foot throughout the entire step can help reduce oscillations while turning. When walking straight however, moving the ZMP during the step can help to produce a smoother gait at high speeds, but this smoothness can also be adjusted by tuning the controller's parameters.

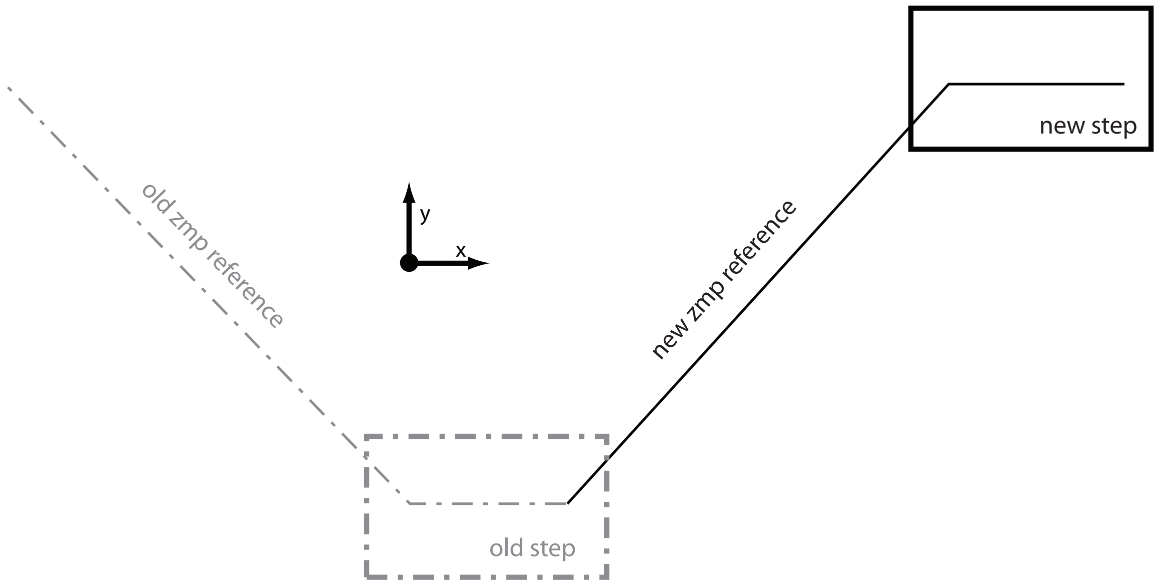

Figure 9: New ZMP reference values are needed when a new step is created: The solid line indicates the 2 dimensional path of the reference ZMP corresponding to the added step.

4.2 Preview Control

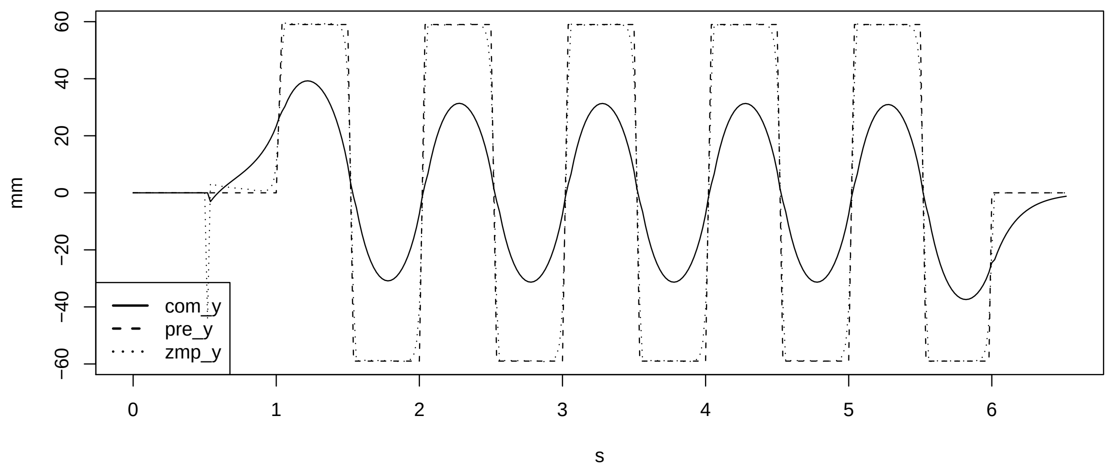

Figure 10: Lateral CoM movement given a reference ZMP plotted during a sequence of five forward steps.

As mentioned several times above, the preview controller is the central part of the walking engine which finds a numeric solution to the inverse of the ZMP equation [5]. The problem is very similar to a servo tracking problem (e.g. a PID controller) where a controller is built to match the output of the system to the desired input (or target) by manipulating the system's control. In a servo, this is analogous to manipulating the current given to a motor to achieve the desired joint angle. Another analogy is the cruise control system in a car – the control is in this case the amount of gas sent to the engine, which should be adjusted so that the speed of the car (output) matches the desired speed specified by the driver (input). In each of these systems, the control system is represented by a one dimensional state – joint angle, or vehicle speed is the only variable which is important.

In the walking robot case, our state has not one but three dimensions - position, speed, and acceleration. By changing the acceleration of the system (the control), we hope to manipulate the robot's stance so that the desired ZMP is produced. Note that the ZMP can be computed from position and acceleration at any moment using equation 1. This controller has also one other very important difference over a simple PID controller – in order to maintain its balance correctly, the robot must already begin responding *in advance* to future desired inputs. The way the controller achieves this is by previewing the future ZMP reference values. In other words, the robot must be aware where it is planning to step so it can correctly anticipate the way it will need to shift its weight to maintain its balance. The number of future ZMP reference values which are previewed by the controller is denoted by N. Since there must always be enough ZMP values to preview, steps which are planned to be started fewer than

N steps in the future are no longer mutable if the desired walk direction changes. This means the response to a change in direction can be delayed. In choosing the appropriate N, one has to weigh the tradeoff between responsiveness and the degree to which the controller will accurately track the reference. We choose $N = 70$, which corresponds to 1.4 seconds because this level of previewing showed very good tracking of the reference ZMP in our tests.

During each time step, the controller models the state transitions using the inverted pendulum/ cart-table model, irrespective of the actual state transitions of the real robot. This controller which can preview future demand is called a preview controller and was first introduced in 1985 by Katayama [8].

In this case, the preview controller acts on a list of future ZMP reference values $p_{ref}(k)$, which defines the location of the desired ZMP at a time $k\Delta t$ seconds in the future, where Δt is the duration of a motion time step. The determination of the future reference values is discussed in section 4.1. The state of the robot is modeled by the controller in one dimension using its position, velocity and ZMP as $\mathbf{x} \equiv [x, \dot{x}, p]^T$. Note that it is better to store the ZMP in the state vector instead of the acceleration, since both are equivalent (see equation 1), and storing p explicitly simplifies finding the output of the system [4]. The controller works to converge p, from the state vector, with the future p_{ref} values, which effectively constitutes an inverse to the ZMP equation (1). 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. One controller working in the lateral direction during a sequence of 10 steps is shown in Figure 10. Since the I coordinate frame will not always remain aligned with the C or F coordinate frames, the two controllers will not always remain split exactly between lateral and sagittal motion when the robot is turning.

Each motion frame, the controller updates its internal model of the robot's state using the discretized preview control state update 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:

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

Where N is the number of previewed time steps, and $\mathbf{k}^T$ is the proportional state feedback row vector. $\mathbf{A}$ is the state update matrix, $G_d(j)$ is the preview gain function, and $\mathbf{b}$ is a constant vector. Details of the preview controller, including the process for finding the optimal values for these matrices is discussed in more detail in [4,5,8] and in appendix B. The process for numerically obtaining them relies on an offline numerical computing environment such as Matlab or Octave.

4.3 Preview Control with Observer

Since the regular preview controller maintains its own simplified model of the state transitions of the robot based on the cart-table paradigm, it is unable to correct inaccuracies in the model (such as when a robot's mass is not concentrated entirely at the CoM), or to compensate for external pushes or uneven flooring. To this end, the preview controller can be augmented with an observer which watches the actual state transitions of the robot, and provides feedback to the controller [4]. The NAO robots are equipped with an inertial sensor unit with accelerometers and gyroscopes, as well as pressure sensors in the feet, and joint encoders which can be theoretically used to measure the true ZMP of the robot. Using a measured ZMP, p_{sensor} the controller is able to more accurately design the CoM path to compensate for sensed reality. The process of determining the sensor ZMP is discussed in section 4.4.

Similar to the plain preview controller discussed in 4.2, the state for the observer is given as $\mathbf{x} \equiv [x, \dot{x}, p]^T$. The state update is different however since it must include sensor feedback information:

$$\mathbf{x}(k+1) = \mathbf{A}_0\mathbf{x}(k) + \mathbf{L}[p_{sensor}(k) - \mathbf{c}^T\mathbf{x}(k)] + \mathbf{b}u(k) \tag{4}$$

where $\mathbf{L}$ is the observer matrix, which determines how to correct the assumed state updates provided by the $\mathbf{A}_0$ matrix with the difference between the modeled ZMP, $\mathbf{c}^T\mathbf{x}(k)$, and the sensed ZMP $p_{sensor}(k)$. ($\mathbf{c}^T$ is a constant output row-vector.) The non-sensor based control $u(k)$ is determined by a combination of factors including integrated feedback error, state feedback, and integrating preview action:

$$u(k) = -G_I \sum_{i=0}^{k} [\mathbf{c}^T\mathbf{x}(i) - p_{ref}(i)] - \mathbf{G}^T{}_x\mathbf{x}(k) - \sum_{j=1}^{N} G_d(j)p_{ref}(k+j) \tag{5}$$

Finding the optimal values for the constants G_I, $\mathbf{G_x}$ and $G_d(j)$ is discussed in appendix B. $G_d(j)$ is simply a weighting function which determines how much future values of the ZMP affect the magnitude of the control (see Figure 11).

4.4 Sensor-based ZMP Readings

Having sensors correctly inform the observer about the *real* ZMP is quite tricky. The overall idea is to use the many sensors which are on the NAO, such as accelerometers, foot sensors, and joint encoders to provide an instantaneous estimate of the ZMP (p_{sensor}). However, in practice this is not so easy.

Given good accelerometer values, it is trivial to calculate the actual ZMP of the robot using equation 1. However, in practice, the accelerometers are very noisy. This necessitates smoothing out the accelerometer values using a Kalman Filter [15]. When the robot is stepping quickly, the ZMP must switch in only 20 or 40 ms from one foot to the next. However, in our trials we were not able to tune the Kalman

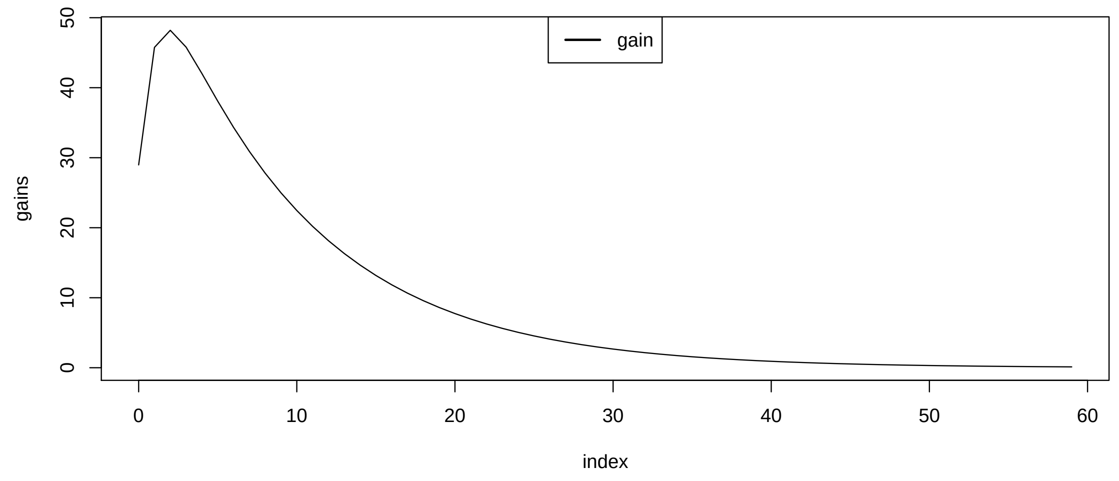

Figure 11: Optimal values for $G_d(i)$, where the graph shows the appropriate weighting of a previewed ZMP reference value at i motion frames in the future.

filter to include oscillations of this frequency without also including sensor noise.

Even when the step frequency of the robot is reduced, and the accelerometer Kalman filter can apply stronger smoothing, the delayed response of the filter causes the robot to swing much more initially than required, and then catch itself as it notices it has swung too far. This type of sensor feedback was not helpful to improving the overall stability of the robot. In 10 trials walking on the edge of the carpet, the robot fell equally often with the observer on as it did with the sensor feedback turned off.

4.5 Kinematics

The final component of controlling the robot is translating leg trajectories from the 3-dimensional version of the C frame into joint angles which can be sent to the actuators (see Figure 12). This process is called inverse kinematics, but is often used implicitly in the literature with little or no explanation.

4.5.1 Forward Kinematics

To understand inverse kinematics, it helps to first understand forward kinematics. Forward kinematics is conceptually and computationally much easier than inverse kinematics. Given a sequence of joint angles, it is always possible to give the resultant location of the end effector. The process could easily be done by hand in two dimensions, with pen, paper and a protractor. Given each angle and the length of each link, one could easily find the location of the end of the arm, for example. Inverse kinematics is the opposite of this process and can not be solved symbolically.

Though there are many possible ways to calculate forward kinematics, we will

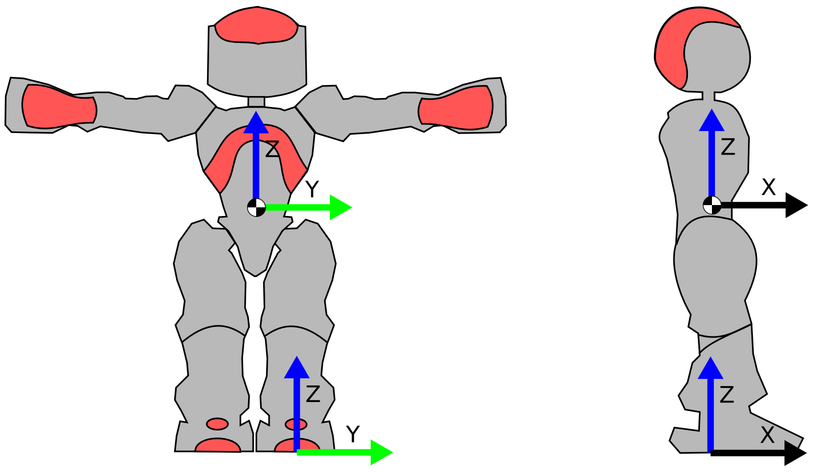

Figure 12: For inverse kinematics, we consider a 3-dimensional version of the C and F frames (one for each foot).

use the modified Denavit Hartenberg convention (referred to as mDH from now on), which is described online in many places (e.g. Chapter 1 of Springer Handbook of Robotics [13].) The basic idea behind mDH is to describe the method in which joints are connected to each other in a standard way. If all the links in the system are described in such a way, we can then write a general method to translate the joints for a given chain into a point in 3-space if we are also given the mDH parameters for that chain.

Generally, the mDH convention provides a standard way of converting between the local reference frames of each actuator to the next one in the chain. In addition, a small number of pre- and post-transforms define the offsets between the origin coordinate frame and the first actuator, and between the last actuator and the coordinate frame of the end effector, respectively.

In effect, each step in this conversion involves iteratively applying a rotation matrix to the origin coordinate frame, where the iterating matrix depends on the actuator's angle of rotation.

Using a symbolic mathematics package such as Mathematica, we can precalculate the translation between the joint space and 3-space for each chain to save computation at run time. (See appendix C.)

4.5.2 Inverse Kinematics

Inverse kinematics is the opposite of forward kinematics. The goal is to find a set of angles which will move the end of the target chain to the correct location in 3-space. It is important to note that most points in 3-space cannot be achieved with a unique set of joints, and even more points are not reachable at all. For example, it is impossible to reach a point with your ankle that is further from your pelvis than the length of your leg. Similarly, there are many possible arm configurations which will ring a doorbell.

The method we use to solve inverse kinematics is basically an iterative solution to a minimization problem in the error space of forward kinematics. Consider a target, T, in 3-space for the end effector: $T = (x, y, z)$. Consider an initial set of joints j_0, then we can apply forward kinematics (FK) to find the current location in 3-space, t_0, given those joint angles: $t_0 = FK(j_0)$. We can also measure the current error as the distance from the current end effector to the desired target $e(t_i) = (T - t_i)^2$. The inital error is given as $e(t_0)$. Now, we'd like to find a set of joints $j_2, \cdots, j_n$ such that each joint combination j_i has a lower error $e(t_i)$, until $e(t_n) \approx 0$. In this situtation, the end effector for the joint angles j_n will have arrived at the specified location.

In practice, we solve this problem using Jacobians to find the "direction" of steepest descent in order to minimize the error as quickly as possible. We can perform this process quickly enough so that it can be used without trouble at run-time on the robot. An outline of the Jacobian approach is presented here [3]. For balanced walking, we also impose a second condition that the foot remain parallel to the ground.

Using inverse kinematics (sometimes referred to as IK), we can control all chains of the robot in 3-space, rather than the joint space. This extra layer of abstraction helps to connect the high and low level tasks required for walking.

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 similar to the 11.3 cm/s top speed of the Aldebaran walk engine achieved in [9]. In addition, we are able to move omnidirectionally. Furthermore, the closed-source Aldebaran walk does not have any potential way to add sensor feedback. (It is not known exactly which method the Aldebaran walk uses for gait synthesis.) From this comparison, it can be seen that there is a definite advantage to developing our own walking system, since we are able to decide specifically which direction we would like to walk. However, walking at 10.5 cm/s with either walk engine can be very unstable, particularly since the robot can build up oscillations which can cause the robot to fall after several steps of walking.

Theoretically, introducing sensor feedback can compensate for such oscillations, but as noted in section 4.4, we had significant difficulties implementing sensor feedback. This was due largely to the fact that we were unable to estimate a good sensor ZMP from the accelerometers with low time delay. Though the introduction of the observer was visibly able to compensate for some build up of oscillation, the extra oscillations introduced due to sensor lag did not make the closed loop system a significant improvement. Furthermore, the addition of sensor feedbacks adds an even greater amount of variability to the walk, which makes it more difficult to tune gait parameters effectively.

Even when the sensor feedback was turned off[2], we found that the CoM control resulting from the preview controller with observer was smoother and more stable than the CoM control which was generated from the simple preview controller. This is most likely due to the slight differences in the way the optimal controller is derived in each case.

With or without sensor feedback, we were able to walk consistently while remaining balanced with a speed of 5 cm/s. The speed reduction reduced the amount of oscillation buildup when switching directions, but it did not eliminate it. In practice, a strategy to reduce such oscillations sufficiently requires stopping before large changes in velocity (i.e. changing directions). Though this is not optimal, without working sensor feedback a better solution is unlikely to be found.

Since we did not satisfactorily complete sensor feedback to effectively compensate for external disturbances, we rely on the accelerometers to distinguish an upright posture, and sense when the robot has fallen, allowing a standup move to be executed before restarting the walking.

Additionally, there are differences in the motors from robot to robot, and we must make slight adjustments to two parameters individually per robot to keep the robot from falling over. The two additional parameters we have introduced to correct for the perfect model of the robot are an induced offset in the lateral swing offset, and an artificial compensation to one of the hip joints to compensate for weak motors.

[2] Using the preview controller with observer, it is possible to effectively turn off sensor feedback by feeding the expected, perfect sensor data into the observer.

The first adjustment we make was inspired by the walk Aldebaran ships with the robots [12]. 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 an offset to each individual hip-roll (hip lateral swing) joint during the swinging phase (using a cycloid function). Though this offset is configurable for both right and left hips, we have found a constant maximum correction of about 4 degrees works well across multiple robots.

The second offset we introduced helps balance the robot by moving the reference ZMP laterally away from the inside of the foot, inducing a greater hip swing. Since the robot is unable to measure the actual ZMP of the robot, we are able to tune this parameter to get the correct swing on a per robot basis. Since each robot has different motors, we have found manipulating this offset an effective means of compensating for this variability. The offset is distinct for each hip so this provides considerable help in offsetting asymmetries in robots that might have asymmetrically weakened knees or hips.

An additional important result of this project is that the code implementation of our system, written in C++ using Boost's UBLAS linear algebra package, is publicly accessible under the LGPLv3 license using `git` at `http://github.com/northern-bites/nao-man.git`.

5.1 US OPEN 2009

As part of preparation for the worldwide RoboCup competitions which will be held in July 2009 in Graz, Austria, Bowdoin College hosted the US OPEN for the Standard Platform League in May 2009. This event provided a crucial chance to test our walking engine in real soccer games, and also allowed our team to discuss walking approaches with other teams.

Of the 4 teams in attendance, two of the others used the closed source engine from Aldebaran, while only one other team besides us had designed their own walk engine. Although our walk was nearly as fast in straight line speed as the fastest team, our ability to respond to the movements of the ball was worse. This was in part due to our walking engine. Also, our lack of ability to have equally good gaits on all our robots severely hampered our ability to have thee working robots on the field at each time. These observations led us to discover some key weaknesses in our system:

1. The starting and stopping of our engine was slower than other teams. While this is in part due to the necessity of previewing 1.4 seconds in the future, it was also due to a design flaw in our step storing system which requires firm commitment to at least 2 future steps regardless of step frequency.

2. Our inability to tune all our robots correctly is most likely due to the fact that we never vary the stiffnesses of any of the joints in our legs during the gait cycle. This means with certain types of gaits, our robots quickly lost balance after small disturbances.

Of all the teams in attendance, no team was able to find a better solution to reducing the oscillations resulting from changing directions. The common strategy, as mentioned above, is to come to a complete stop before switching direction.

6 Future Work

Future work is particularly necessary in advance of the RoboCup world championships which are coming this summer. We have pinpointed several concrete improvements which will allow us to fix the weaknesses we observed at the US OPEN 2009.

Perhaps the most important improvement will be to introduce stiffness modulation into the gait cycle. This will allow the hips and the supporting leg to maintain high stiffness, while the knee and ankle of the swinging leg will have medium and low stiffness respectively. Reducing the stiffness in the ankles should reduce the amount of backlash that the robot experiences as the swinging leg touches down, and should allow the development of faster gaits. The extra smoothness which comes with the lower stiffness may also improve our ability to rely on the sensor ZMP, and may allow the observer to help create a noticeably stabler walk. Reducing the rigidity of the ankles also has the potential to help enormously with compensating for the differences between robots' actuators by lowering the impact of imperfections in the gait.

The architectural flaw which reduces our ability to respond quickly to the environment must also be fixed. In order to stop quickly, it is crucial to be able to change, update, or delete the current future planned steps, and replaced them with stopping steps instead. The flaw stems from two places which limit our ability to make such changes. The first is that during a gait cycle we are unable to move the destination of the swinging leg, even if the destination has not yet been taken into account for the generation of preview values. This means we must wait at least 2 additional steps before we are able to stop. To fix this, we will need make the WalkingLeg component of the architecture able to gracefully respond to moving targets for the destination of the swinging leg during the gait cycle. The second flaw stems inherently from the nature of the preview controller, since we are unable to move the locations of future desired steps once the controller has begun to consider the future reference ZMP values which were generated from the step. It is further complicated by the fact that we generate ZMP reference for a whole gait cycle at one time. This makes it difficult to respond quickly to a changing environment, since this generated ZMP can no longer be changed, even if only part of it is needed for use as preview-able ZMP reference values.

Though one possible fix for this problem is to reduce the preview period to less than 1.4 seconds, this will probably not be enough. Although we have not experimented on the real robot with lowering the preview period, offline tests show that it can only be lowered slightly before the controller starts to loose the ability to achieve the desired ZMP. Likely, we will need to build into the controller the ability to move steps even after the controller has begun to consider the associated ZMP reference values. Although this may break the theoretical optimality of the controller, it may be necessary to achieve fast enough response times to the movement of the ball. The fix for this would involve recomputing the next N preview frames every time step, so that as the steps are moved towards a stopped configuration, the controller can begin responding immediately. The advantage of the current, flawed model is that it is computationally faster, because it only computes the ZMP reference values once, and keeps them in the I frame. By continually recalculating the ZMP reference values, we would increase the flexibility of the walking engine while simultaneously decreasing the computational efficiency of the engine.

In comparing our walk engine with the Aldebaran walk engine, one thing which they may be doing differently (and in fact better), is that they may be considering the full kinematics of the robot in their inverse kinematics step. We fix the belly button of the robot to follow the CoM path of the robot produced by the preview controller, regardless of the movement of the swinging leg. This means the mass of the legs is not considered when we follow the output of the preview controller and we consequently are not correctly following the CoM path. When the swinging leg is moving quickly, or is extended far from the rest of the body, this can have especially harmful effects on the balance of the robot. To fix this shortcoming, it would be possible to build a more complex inverse kinematics system which finds the appropriate angles for the leg joints to send both legs, and the combined CoM of the body to their appropriate destinations.

Finally, as a possible alternative to the observer, it may be desirable to move the locations of future or current footsteps to respond to large sustained disturbances. Using the modifications proposed in the previous paragraph, this type of compensation would be much easier.

7 Conclusion

There is an increasing emphasis on humanoid robots since humanoid robots have several advantages over wheeled robots. Bipeds have higher mobility on rough or humanized terrain (such as stairs), and are more socially compatible with humans. In RoboCup, this increased interest reflects the desire to play soccer on even terms with humans. As part of this effort to achieve RoboCup's mission to beat the human FIFA world cup champions in 2050, researchers from around the world will be drawn to the humanoid robotics as a platform for research. Our intent with this project is to provided a primer on humanoid walking, complete with an open-source code implementation and strong documentation, both at the conceptual level (which is provided in this report) as well as at the implementation level (which is provided in the code itself). This desire was particularly fueled by our initial frustration as we attempted this project with limited background in control theory and humanoid robots. We hope that this report will enable future researchers in our position to quickly familiarize themselves with one method of humanoid gait generation, and thus ultimately speed up the rate of research in the field.

Although we faced many challenges, some of which were not possible to solve within the time frame of the project, we believe that our system is nonetheless well designed and resource efficient. With the many layers of abstraction we put in place, the extension of our code becomes much easier, and thus enables future research to be conducted more effectively. Since we were outperformed at the US OPEN by some of the teams who had exceptional tunings of the default Aldebaran walk, we will need to leverage the advantages that we gain by being able to modify the architecture and inner workings of the walk engines. In addition, the open-source nature of this project will allow other interested researchers in our league to build a robust system for comparing different methods of gait synthesis.

Acknowledgements

This work was conducted in close collaboration with George Slavov. Without his background in applied math, the work you read about in this paper would have been impossible. Also, Professor Eric Chown's continued support for student-run research, despite his being on sabbatical, has been a great help as well.

References

[1] Yariv Bachar. Developing controllers for biped humanoid locomotion. Master's thesis, University of Edinburough, 2004.

[2] Sven Behnke. Online trajectory generation for omnidirectional biped walking. *2006 IEEE International Conference on Robotics and Automation*, 2006.

[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 Elsevier*, 2009.

[5] Shuuji Kajita et al. Biped walking pattern generation using preview control of zero-moment point. In *International Conference on Robotics and Automation; Proceedings of the 2003 IEEE*, 2003.

[6] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. Development of honda humanoid robot. In *International Conference on Robotics and Automation*, 1998.

[7] Tatsuzo Ishida. Development of a small biped entertainment robot qrio. In *Proceedings of the 2004 International Symposium on Micro-Nanomechatronics and Human Science*, 2004.

[8] 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.

[9] Jason Kulk and James Welsh. A low power walk for the NAO robot. Technical report, University of New South Wales, 2008.

[10] Standard Platform League of the RoboCup Federation. `www.tzi.de/spl`.

[11] Joghoon Park and Youngil Youm. General ZMP preview control for bipedal walking. In *IEEE International Conference on Robotics and Automation*, 2007.

[12] Aldebaran Robotics. `www.aldebaran-robotics.com`.

[13] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Spring, 2008. `http://books.google.com/books?id=Xpgi5gSuBxsC`.

[14] Miomir Vukobratović and Branislav Borovac. Zero moment point – thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.

[15] Greg Welch and Gary Bishop. An introduction to the kalman filter.

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.

B Finding the optimal preview-controller (with observer)

This appendix shows the numerical process of deriving the optimal solutions for the components of the preview controller with observer presented in section 4.3.

Modeling discretized state transitions with matrices

Using the model presented in section 3.2, we can model the continuous state transitions of the robot using linear algebra:

$$\frac{d}{dt}\mathbf{x} = \mathbf{A}'\mathbf{x} + \mathbf{b}^{T'}v \tag{B-1}$$

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{z_h} & 0 & \frac{g}{z_h} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ p \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} v \tag{B-2}$$

Where $\mathbf{x}$ is the state of the system, and v is the control input. In this instance of the preview control, the control vector v is actually a single number. g is the *magnitude* of gravity ($9.8\frac{m}{s^2}$.) For the Nao robot, we pick $z_h = 0.31$ meters.

Since we need to model this system on a digital computer, we need to discretize this model into timesteps Δt. The frequency of updates to the joint actuators is 50 ms, so $\Delta t = 0.02$ seconds. The discretized state update is given by

$$\mathbf{x}(k+1) = e^{\mathbf{A}'\Delta t} + \left(\int_0^{\Delta t} e^{\mathbf{A}'v} dv\right) \mathbf{b}'v(k) \tag{B-3}$$

This gives the discrete-time time-invariant model of the system

$$\mathbf{x}(k+1) = \mathbf{A}_0\mathbf{x}(k) + \mathbf{b}u(k) \tag{B-4}$$

$$\mathbf{y}(k) = \mathbf{c}^T\mathbf{x}(k) \tag{B-5}$$

Where $\mathbf{A}_0 = e^{\mathbf{A}'\Delta t}$, and $\mathbf{b} = \left(\int_0^{\Delta t} e^{\mathbf{A}'v} dv\right)\mathbf{b}'$. This yields

$$\mathbf{x}(k+1) = \begin{bmatrix} 1 & \Delta t & 0 \\ \frac{g}{z_h}\Delta t & 1 & \frac{g}{z_h}\Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ p \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \Delta t \end{bmatrix} u(k) \tag{B-6}$$

$$\mathbf{y}(k) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \mathbf{x}(k) \tag{B-7}$$

Up to this point, we have simply developed a model for the state transitions of the robot without considering sensor feedback. To incorporate sensor feedback, we introduce the observer matrix $\mathbf{L}$, whose derivation described in the next section. The complete state update (including sensor feedback) is given by

$$\mathbf{x}(k+1) = \mathbf{A}_0\mathbf{x}(k) + \mathbf{L}[p_{sensor}(k) - \mathbf{c}^T\mathbf{x}(k)] + \mathbf{b}u(k) \tag{B-8}$$

Finding optimal control

Now that we have developed a matrix model of the system, the next step is to find the optimal way to control the system so that the output converges to the desired target quickly. This is covered in [4], but is reproduced with some annotations and clarifications here.

First we need to find an optimal $\mathbf{L}$ such that the matrix $\mathbf{A}_0 - \mathbf{b} * \mathbf{G}_x$ is assymptotically stable. This is done using the Linear Quadratic Regulator(LQR) method in Matlab: `L = dlqr((A0-b*Gx)', c', eye(3) , R)';`

To find the optimal controller, we define a performance index J that we will minimize to find the optimal controller. J is defined as

$$J = \sum_{j=1}^{\infty} \left\{ Q_e \left[p(j) - p_{ref}(j)\right]^2 + \Delta\mathbf{x}^t\mathbf{Q}x\mathbf{x} + R\mathbf{v}^t(j)\mathbf{v}(j) \right\} \tag{B-9}$$

where the first term serves to minimize the tracking error (the difference between the current ZMP and the reference (target) ZMP), the second term to minimize changes in the state vector and the third term serves to minimize the magnitude of the control being applied to the system. $\mathbf{Q}_x$ is a square diagonal matrix with Q_x on the diagonals. The constants Q_e, Q_x and R are arbitrary. In our work we used $Q_e = 0.3, Q_x = 0.25, R = 1.0 \times 10^{-6}$ [4].

Minimizing the performance index is discussed in [8], and yields the control law which allows us to calculate $u(k)$:

$$u(k) = -G_I \sum_{i=0}^{k} [\mathbf{c}^T\mathbf{x}(i) - p_{ref}(i)] - \mathbf{G}^T{}_x\mathbf{x}(k) - \sum_{j=1}^{N} G_d(j)p_{ref}(k+j) \tag{B-10}$$

where the optimal $G_I, \mathbf{G}^T{}_x, G_d(i)$ are determined by solving the following discrete Riccati equation:

$$\mathbf{P} = \mathbf{A}^t\mathbf{P}\mathbf{A} - \mathbf{A}^t\mathbf{P}\mathbf{B}\left[R + \mathbf{B}^t\mathbf{P}\mathbf{B}\right]^{-1}\mathbf{B}^t\mathbf{P}\mathbf{A} + \mathbf{Q} \tag{B-11}$$

where

$$\mathbf{Q} = \begin{bmatrix} Q_e & 0 \\ 0 & \mathbf{Q}_x \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{c}^T\mathbf{b} \\ \mathbf{b} \end{bmatrix}, \mathbf{A} = \begin{bmatrix} \mathbf{D} & \mathbf{F} \end{bmatrix}, \mathbf{D} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} \mathbf{c}^T\mathbf{A}_0 \\ \mathbf{A}_0 \end{bmatrix} \tag{B-12}$$

Equation B-11 can be solved using the Matlab/octave routine and `P=dare(A, B, Q, R);`

In addition, the following asymptotically stable matrix is computed to find the preview function:

$$\mathbf{A}_c = A - \mathbf{B}\left[R + \mathbf{B}^t\mathbf{P}\mathbf{B}\right]^{-1}\mathbf{B}^t\mathbf{P}\mathbf{A} \tag{B-13}$$

Together, these matrices allow solving for the optimal controller constants:

$$G_I = \left[R + \mathbf{B}^t\mathbf{P}\mathbf{B}\right]^{-1}\mathbf{B}^t\mathbf{P}\mathbf{D} \tag{B-14}$$

$$\mathbf{G}_x = \left[R + \mathbf{B}^t\mathbf{P}\mathbf{B}\right]^{-1}\mathbf{B}^t\mathbf{P}\mathbf{F} \tag{B-15}$$

$$G_d(j) = \left[R + \mathbf{B}^t\mathbf{P}\mathbf{B}\right]^{-1}\mathbf{B}^t\left[\mathbf{A}_c^t\right]^{j-1}\mathbf{P}\mathbf{D} \tag{B-16}$$

C Forward Kinematics

Forward kinematics using the mDH convention as discussed in section 4.5.1 involves coordinate transformations using homogeneous coordinate transformations. There are four types of 3D homogenous coordinate transformation matrices: translation by (x, y, z), or rotation about one of three axis by θ:

$$Rot_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Rot_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C-17}$$

$$Rot_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Trans(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C-18}$$

Each link in a robots chain is assigned mDH parameters $\vec{m} = (\alpha, L, \theta, D)$, where the translation update for that link in the chain is given by the matrix expression

$$LinkUpdate(\alpha, L, \theta, D) = Trans(L, 0, 0)Rot_x(\alpha)Rot_z(\theta)Trans(0, 0, L) \tag{C-19}$$

D Code

The C++, R, Octave and Mathematica code created as part of this project is licensed under the LGPL as part of the Northern Bites RoboCup code repository and is available using the `git` versioning control system from `www.github.com/northern-bites`. Additional documentation and instructions for compiling and using the source are available on the team's wiki at `robocup.bowdoin.edu/trac`. At the time of writing, no release candidate has been designated since the Northern Bites code base is under constant development.