rUNSWift Walk2014 Report
Robocup Standard Platform League

Bernhard Hengst

bernhardh@cse.unsw.edu.au

September 30, 2014

Abstract

This report describes the development and implementation of the locomotion for the Nao H25 V4 robot as used by team rUNSWift from the University of New South Wales, Australia, for the 2014 RoboCup SPL competition. We refer to the omnidirectional locomotion motion as a *walk*. The main purpose of the report is to document the 2014 walk for the UNSW 2014 SPL code release.

School of Computer Science & Engineering
University of New South Wales
Sydney 2052, Australia

Contents

1 Introduction **1**

2 History of Walk Developments at CSE/UNSW **1**

3 Background **2**

3.1 Walk Basics 2

3.2 Related Work 3

4 Motion Architecture **5**

4.1 ActionCommand 5

4.2 Touch 6

4.3 Generator 6

4.4 Effector 7

5 Walk2014Generator **7**

5.1 Inverted Pendulum Dynamics 8

5.2 Feedback Control 10

5.3 Inverse Kinematics 12

5.4 Stepping Through the Code 15

5.5 Walk2014 Footstep Response to Change in Walk Parameters 20

A Omni-directional Walking Parameterisation **23**

B Kinematic Transforms for Nao Robot **23**

C Kinematic Denavit-Hartenberg convention(D-H) **23**

D Kinematic Chains for Nao **25**

E Inverse Kinematic Matlab Code **30**

1 Introduction

This report describes the rUNSWift 2014 walk for the Nao robots used in the 2014 SPL Robocup competition in Brazil.

The scope of this report is limited to describing the bipedal gait and omni-directional locomotion. It does not include kicking or other canned behaviour such as goalie dives and standing up after a fall.

The basic walk was first developed in 2010 and described in the 2010 rUNSWift report [18] that accompanied the code release at the time. For completeness relevant parts of that report are included here. The 2014 walk is superior to prior bipedal rUNSWift walks in several ways, including speed, reactivity, and integrated resting behaviour to reduce overheating. The forward speed has increased from about 22cm/sec in 2010 to 34cm/sec in 2014, but in competition this was clipped back to a maximum of 30cm/sec.

2 History of Walk Developments at CSE/UNSW

The Standard Platform League originally used the Sony AIBO robotic quadruped. The league took a significant step forward in 2000 when the School of Computer Science and Engineering (CSE), University of New South Wales (UNSW) developed a competitive walk that later became the standard for the competition [10]. The key insight was to describe the trajectory of the paws by a simple geometric figure that was parameterised. This made experimentation with unusual configurations relatively easy. As a result, we were able to devise a gait that was much faster and more stable than any other team. Since then, almost all the other teams in the league have adopted a similar style of locomotion, some starting from our code.

The flexibility of this representation led to another major innovation in 2003. We were the first team to use Machine Learning to tune the robot's gait, resulting in a much faster walk [15]. In succeeding years, several teams developed their own ML approaches to tuning the walk. Starting from the parameterised locomotion representation, the robots are able to measure their speed and adjust the gait parameters according to an optimisation algorithm.

With the introduction of the Nao robots to the league in 2008, bipedal walking became the new challenge. Research in the AI Group, CSE, UNSW includes applications of Machine Learning to bipedal gaits. Yik (a member of the champion 2001 four-legged team) collaborated with Gordon Wyeth of the University of Queensland to evolve a walk for the GuRoo robot [7], which was entered in the humanoid robot league. This method was inspired by the gait learning devised for the Aibos by Kim and Uther [15]. For the humanoid, the same philosophy is applied. Starting from a parameterised gait, an optimisation algorithm searches for a set of parameter values that satisfies the optimisation criteria. In this case, the search was performed by a genetic algorithm in simulation. When a solution was found, it was transferred to the real robot, working successfully. Subsequently, the approach we used was a hybrid of a planner to suggest a plausible sequence of actions and a numerical optimisation algorithm to tune the action parameters. Thus, the qualitative reasoning of the planner provides constraints on the trial-and-error learning, reducing the number of trials

required [29] [22].

In 2009 Tay developed rUNSWift's first bipedal walk [24] for the Nao robot. While Tay's omni-directional open-loop walk was faster then the manufacturer's supplied walk at the time, it could become unstable. Without feedback the robot would fall over.

In 2010 we developed an omni-directional walk for the Nao using closed-loop control in both the coronal and sagittal planes. The walk was used in competition in 2010 a year in which rUNSWift was in the final [18].

Research and development on biped learning to walk and balancing continued over the ensuing years by White [28], Lange [16], Chatfiled [2], Liu [17], and Hengst, et al [12], [11]. Research is ongoing with publications for the 2014 competition walk, and the toe-to-heel walk as demonstrated in the 2014 rUNSWift Challenge, in preparation.

3 Background

3.1 Walk Basics

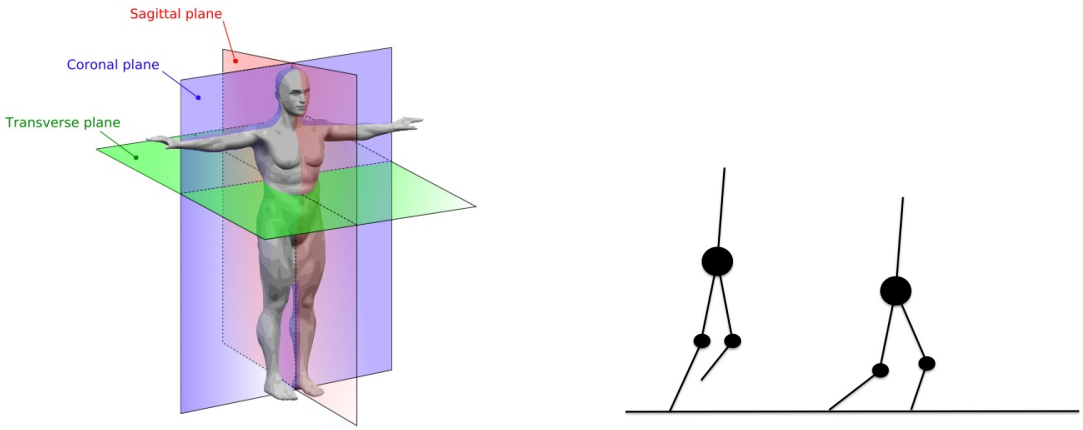


Figure 1: Human Body Planes (left). Single and Double Support Walk Phase (right).

A biped is an open kinematic chain consisting of at least two subchains called *legs* and often a subchain called the *torso*. Additional subchains for a humanoid robot include *arms* and a *head*. One or both legs may be in contact with the ground. In the single support phase, the leg in contact with the ground is called the *stance* leg, the other the *swing* leg [27]. One complete cycle of a bipedal walk can be partitioned into two, usually symmetric, phases. There are two types of partition: a stance phase, followed by a swing phase; or a single support phase followed by a double support phase — see Figure 1 (right) and Figure 2. The three orthogonal human body planes (sagittal, coronal and transverse) are shown in Figure 1 (left).

The center of pressure (CoP) is the point on a body where the total sum of the pressure field acts, causing a force and no moment about that point. The *Zero Moment Point* is defined as that point on the ground at which the net moment of the inertial forces and the gravity forces has no component along the horizontal axes. When the body is dynamically balanced

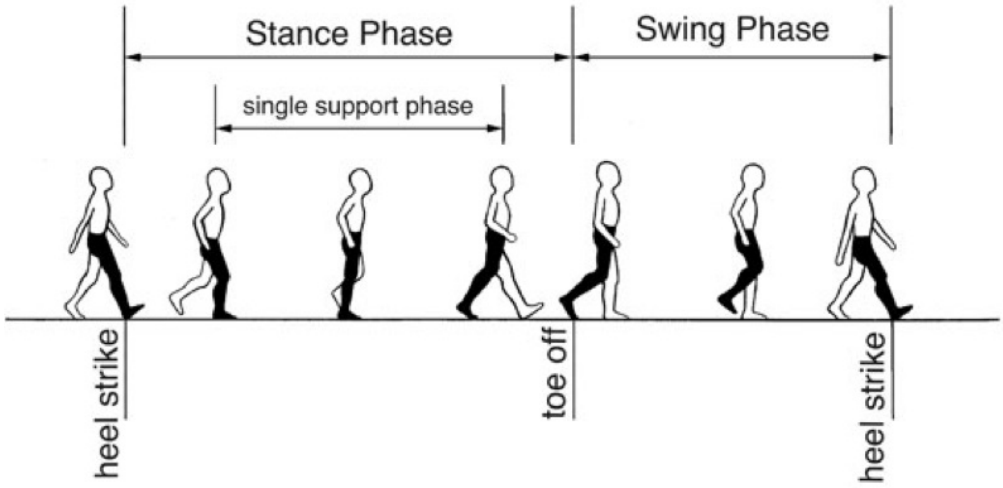


Figure 2: A complete walk cycle showing the stance and swing phase of the right leg in the sagittal plane [9].

the ZMP and the center of pressure (CoP) coincide. For an unbalanced body, the CoP is at the edge of the support polygon and the ZMP does not exist (or is a fictitious value outside the support polygon) [26]. Figure 3 shows ground reaction forces acting on a stance foot and an equation for calculating the CoP (and ZMP) p.

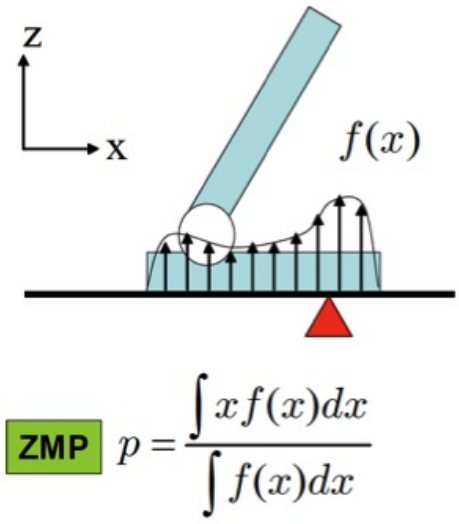


Figure 3: Center of Pressure and Zero Moment Point for a dynamically balanced foot.

3.2 Related Work

There is a considerable and growing body of literature on robotic locomotion including bipedalism. Advanced humanoid robots include Honda's Asimo, Sony's Qrio, Toyota's humanoid and the HRP range at AIST. Many of these robots use the ZMP concepts originally developed by Vukobratovic and Stepanenco in 1972 [26]. Reactive feedback control is inade-

quate to balance a humanoid and successful application of ZMP control relies on anticipating the next step and taking control action even before the stance leg leaves the ground. This type of feed-forward control has been called preview control [14]. Much of the research on bipedal locomotion relies on variations of an inverted pendulum model. The three link system in [6], for example, includes a point mass in each of the legs.

Inexpensive small bipedal robots, such as the one used in the RoboCup humanoid kid size league and the Standard Platform League present their own challenges with the motor frequency response typically low (e.g. 100Hz), and the noise in inertial and foot-sensor measurements high. As the Centre-Of-Mass of these robots is low they fall very quickly.

In 2007 Faber and Behnke presented methods for improving the walking pattern for their kid-size robot Jupp [5]. The underlying open-loop walk was enhanced with two feedback mechanisms. They employed a proportional controller to reduce angular velocity in the sagittal plane and a phase resetting mechanism to switch the walking gait phase using foot-sensors. Our research using reinforcement learning to adjust the ankle tilt to control sagittal balance resulted in a similar control policy. We also switch walking phase based solely on foot-sensor reading, as will be described.

In 2009 several universities had developed their own walks for the Nao. The University of Leipzig's Nao-team HTWK is one of the leading teams in the league. They used an evolutionary algorithms to optimise a closed-loop walk with a reported maximum speed of 32 cm per second in the forward direction. The low vertical actuation of the legs would often cause the robot to fall over [13]. Developments in 2010 improved on the walks omni-directional capabilities, but there are no details available [25].

The Northern Bites team from Bowdoin College implemented an omni-directional ZMP feedback based walk that achieved a stable maximum forward walking speed of 10.5 cm per second. One notable aspect of this walk is the use of efficient iterative inverse kinematics for foot placement [23]. Their implementation uses *Mathematica* to produce symbolic equations to perform the forward kinematic transforms and final desired joint movements. While we implemented closed form inverse-kinematic equations for walking forward and sideways, we largely relied on this technique in 2010 for turning movements because of the complexity of the hip joint of the Nao.

Dortmund University of Technology developed a closed-loop walk based on ZMP control [4]. Their "observer-based" controller included integral tracking error, proportional state feedback and ZMP preview components. This walk was reported to be stable to external disturbances and able to walk on inclined planes tilted at 6 degrees from horizontal.

University of Bremen's team B-Human have generated exemplary motions for the Nao [20]. The inverse kinematic transforms are in closed-form made, possible given the constraints on the Nao's kinematic chains. The walk is closed-loop and balanced by modifying the foot-placement of the next step. The parameter settings of the walk are optimised using a particle swarm algorithm. A smooth transition between different motions is achieved by interpolation.

The walk developed for the Nao by Team rUNSWift in 2014 proved itself to be competitive with walks from other league leaders. The implementation of our walk is described next. An accompanying article is in preparation which describes the research leading to the sagittal

disturbance rejection. This publication is not an essential perquisite for anyone planning to implement the UNSW 2014 walk.

4 Motion Architecture

For completeness, we first reproduce the description of the rUNSWift motion architecture from the 2010 report.

As we are controlling the dynamics of a real robot, it is essential that the motion architecture runs the highest priority thread to achieve near real-time processing movements at 100 Hz. This also means that our normal debugging framework cannot be used within motion, as it might block while writing to the log file.

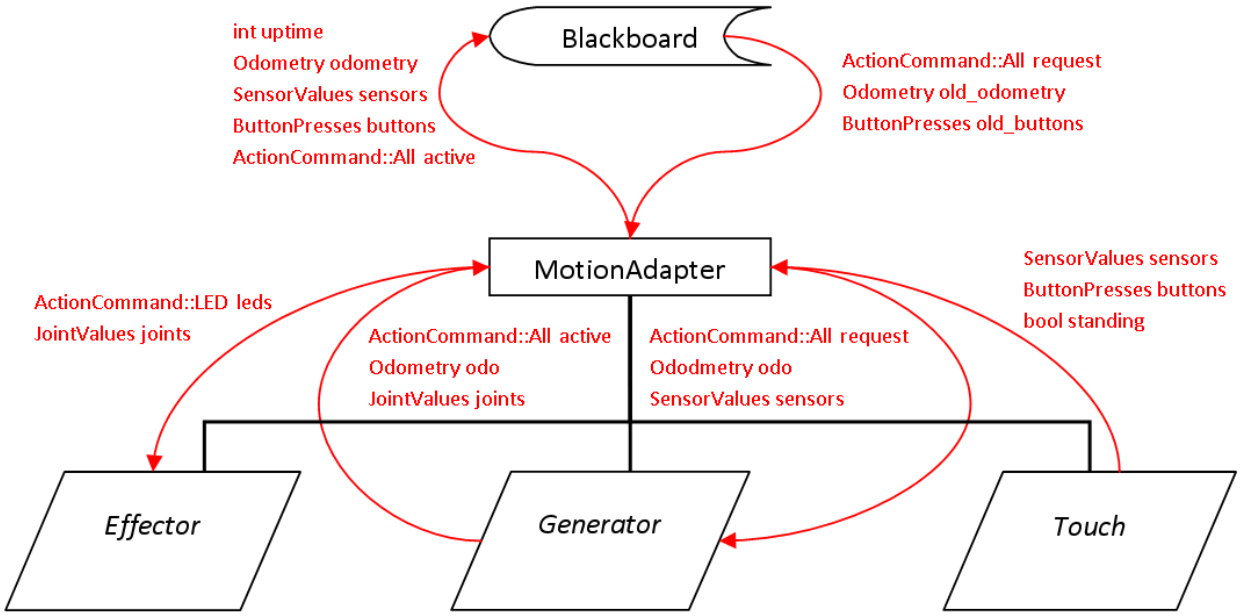


Figure 4: The motion architecture. *Red* indicates data flow; *black* indicates ownership.

Overall control of the Motion thread is provided by the *MotionAdapter* class. The MotionAdapter in turn owns a *Touch* object, a *Generator* object and an *Effector* object. These divide the Motion cycle into three steps — input, processing, and output. This cycle is run every ten milliseconds by the *ThreadWatcher*, which calls the *tick* function of MotionAdapter. Figure 4 provides a summarised outline of the process.

4.1 ActionCommand

The ActionCommand namespace is a collection of data-types that behaviour uses to communicate with Motion. There are 3 main types:

ActionCommand::Body This contains four walk/kick parameters (*forward, left, turn* and *power*. It also contains an *actionType*, which is an enumeration of all the possible body actions. These are also assigned priorities, with higher values indicating priority over lower values. Of note is the STAND action type. This is a standard pose that is used to transfer between all other types. It consists of legs with a 60° knee bend and equal hip and ankle bends of −30°.

ActionCommand::Head This contains two parameters for the head *yaw* and *pitch*, as well as corresponding *yawSpeed* and *pitchSpeed*. Finally, it also contains a *isRelative* flag that determines whether the *yaw* and *pitch* parameters are absolute angles or relative to the current head position.

ActionCommand::LED This contains two 10-bit fields for the left and right ear LEDs (there are 10 individual LEDs in each ear). It also contains RGB values for the left and right eyes (treated as one LED each, even though there are eight separately addressable LEDs), for the chest LED, and for the foot LEDs. The RGB values are limited to boolean values for each of the three colours, yielding 7 possible colours in total (plus an off value).

There is also an ActionCommand::All, which simply is a wrapper around the three main types to ease programming.

4.2 Touch

Implementations of the Touch interface are expected to retrieve sensor data and button press data from the underlying subsystem. This is typically from libagent (using the *AgentTouch*), but could also be from a simulator, or from another Touch instance (*e.g. FilteredTouch*, which filters raw sensor data provided by a child). There is also a special flag, called "standing", that tells Motion that stiffness has been enabled, and hence the current action should be overridden with INITIAL.

In the case of AgentTouch, it waits upon a semaphore shared between it and libagent. When libagent updates a shared memory block with new values, it also signals the semaphore. This will wake up libagent, who then copies the data out of shared memory and passes it to MotionAdapter.

FilteredTouch is merely a decorator around a Touch instance. It simply passes through all data, except for the Sonar readings, which are filtered (see [18]).

There is also a *NullTouch*, which returns dummy values, that is useful for testing the runswift executable off-robot.

4.3 Generator

The Generators are the heart of the Motion system. They take ActionCommands from behaviour and SensorValues from Touch and generate joint movements to be effected. They

also regulate the transition between actions, report to behaviour the currently running action, and maintain odometry for use by localisation.

Most Generators generate joint movements for a walk, kick, or other motion. These are referred to as "body Generators". However, there are some special Generators that are used to control other Generators.

The main one of these is the DistributedGenerator. This has an instance of every body Generator. It implements the action switching policy. When a different action is requested, it sends the currently running generator a *stop* request. When that Generator reports that it is no longer active, DistributedGenerator switches to the new Generator. This process can however be overridden by a series of priorities (declared along with the action types), generally used to implement safety features. For instance, the get-up action has priority and will immediately kill a walk that is running.

DistributedGenerator also owns a HeadGenerator, which processes Head commands from behaviour. DistributedGenerator keeps a list of which body Generators use the head as part of their movement, and which don't. If the current Generator doesn't, HeadGenerator's output will override the current Generator's.

There is also a ClippedGenerator, which is used to wrap around DistributedGenerator. It ensures that joint angles and joint velocities don't exceed the manufacturer limits. If they do, they are clipped to the maximal values.

4.4 Effector

Effector classes are required to implement the JointValues specified by MotionAdapter's Generator, and also the LED commands coming directly from behaviour.

The predominant Effector is *AgentEffector*. It writes the data straight to the memory block shared with libagent, without any changes. This is then processed by libagent during its next DCM cycle. There is also a *NullEffector* which, similar to NullTouch, can be used when developing off-robot.

5 Walk2014Generator

The latest walk is Walk2014. It implements the usual approach of decomposing the walk phase into sagittal and coronal plane dynamics and then synchronously recombines them. The kinematics of the walk in open loop aims to keep the torso of the robot upright and the feet parallel to the assumed flat horizontal ground . The walk is a closed-loop walk with stabilisation feedback in the coronal plane supplied via the foot sensors, and in the sagittal plane by a gyrometer.

The walk is omni-directional in that the robot can be directed to simultaneously move forward, sideways and turn. Naturally the combination of these component vectors must be within the physical capabilities of the machine and need to be controlled to keep the robot balanced. Omni-directional foot placement results in a rich variety of movements, for

example waltzing backwards.

Omni-directional locomotion is achieved by moving the swing foot so that it is replaced on the ground, rotated, and offset in the forward and left directions relative to the body torso hip-joint as shown in Appendix A. The next foot position is specified with three variables, *forward*, *left* and *turn*. Values for these variables represent velocities (in units/second) and are passed as parameters to the walk generator by higher-level skills and behaviours. The actual valued used in the walk generator are calibrated to achieve the passed in velocities as accurately as possible in each individual dimension and converted to meters or radians using the timing of each walk phase. A combination of parameters may reduce the theoretical effectiveness of the resultant motion due to stance leg slip on the ground.

For a given forward step-size the walk movement is constrained to move the point on the robot between the hips at a constant velocity over the ground to minimise the acceleration of the body sub-chains above, including the torso. This is thought to reduce forces during the walk cycle, minimise energy usage and oscillatory disturbances. At the same time the legs are moved to achieve the required omni-directional foot placements. To achieve these body, leg and foot movements we calculate new joint-angles each 1/100 of a second.

5.1 Inverted Pendulum Dynamics

The walk is designed by first dividing the dynamics into orthogonal (coronal and sagittal plane) components and recombining the two motions synchronously to generate bipedal motion. The walk dynamics can be understood by appealing to an inverted pendulum model of the Nao which we analyse next.

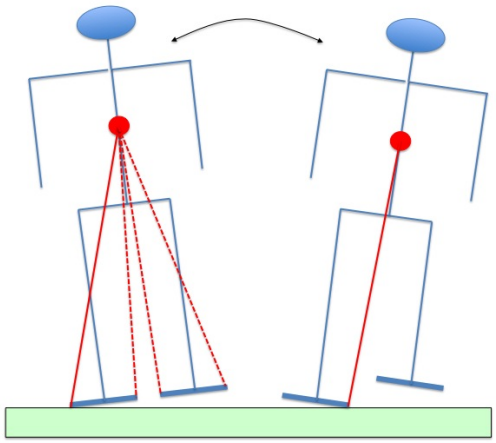


Figure 5: Oscillation in the coronal plane showing the dynamics modelled by inverted pendulums with their pivot points located on the edges of the feet.

Coronal (or Lateral) Dynamics. Figure 5 shows a coronal view of a stylised Nao with the whole mass concentrated at the center-of-gravity (CoG), shown as the red dot in the torso. Each of the flat feet is lifted vertically during the walk, the lift being parameterised by the two variables $footh L$ and $footh R$. The idealised robot will only make ground contact with one of four pivot points in the coronal plane when rocking from side to side. The four points

correspond to the outside and inside edges of the feet. We therefore model the robot as an inverted pendulum with its bob at the CoG and the pivot point at one of the feet edges. As the robot rocks from side to side the pivot point of the pendulum switches depending on which edge touches the ground. In the figure, the depiction on the right shows the robot making contact with the inside left-foot.

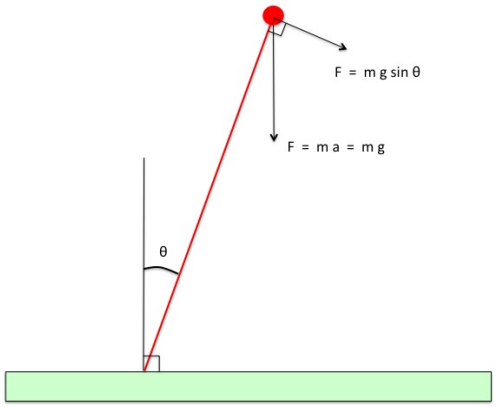


Figure 6: The gravitational force on the bob of an inverted pendulum has a component in the perpendicular direction to the rod proportional to the sin of the angle the rod subtends to the vertical.

The force acting on the bob of an inverted pendulum in its direction of motion is $mg\sin(\theta)$ as shown in Figure 6. We start with an idealised system where total momentum, and hence the magnitude of velocity, is conserved. Each time the pendulum changes pivot we assume the impact results in a loss of energy which we have simply modelled by reducing the velocity of the bob by a small fraction. Energy is added to the system implicitly when the feet push down and lift the robot. We model external disturbances by changing the velocity on impact by a small random quantity with zero-mean. An example time-series plot for the center-of-pressure (the blue squarish wave) and a regular sinusoidal foot lifting action (red wave) for the simple inverted pendulum model of the Nao is shown in Figure 7. The foot-lift plot shown in red is positive when the left foot is lifted and negative when the right foot is lifted. The CoG acceleration alternates from left to right by the alternating foot lift action as the pivot of the pendulum changes from foot to foot.

In the open-loop setting, the timing of the leg lifting action is independent of the state of the rock. External disturbances without feedback or feedforward can cause the robot to loose balance and fall over as shown in Figure 7. Here, towards the end, the sinusoidal action of the feet continues despite the robot being poised on one leg. Our aim is to control the leg-lift motion to stabilise the rock. We describe this in the Section 5.2, but first discuss the dynamics in the sagittal plane.

Sagittal Dynamics. The inverted pendulum model for the sagittal plane is shown in Figure 8. The pivot of the inverted pendulum is again at one of the edges of the feet, this time either at the front or the back. The forces on the robot are determined by the pivot edge and the angle of the rod from the pivot to the CoM (θ in the figure). The stance and swing feet angles to the torso are controlled by the left and right foot $forward$ parameters.

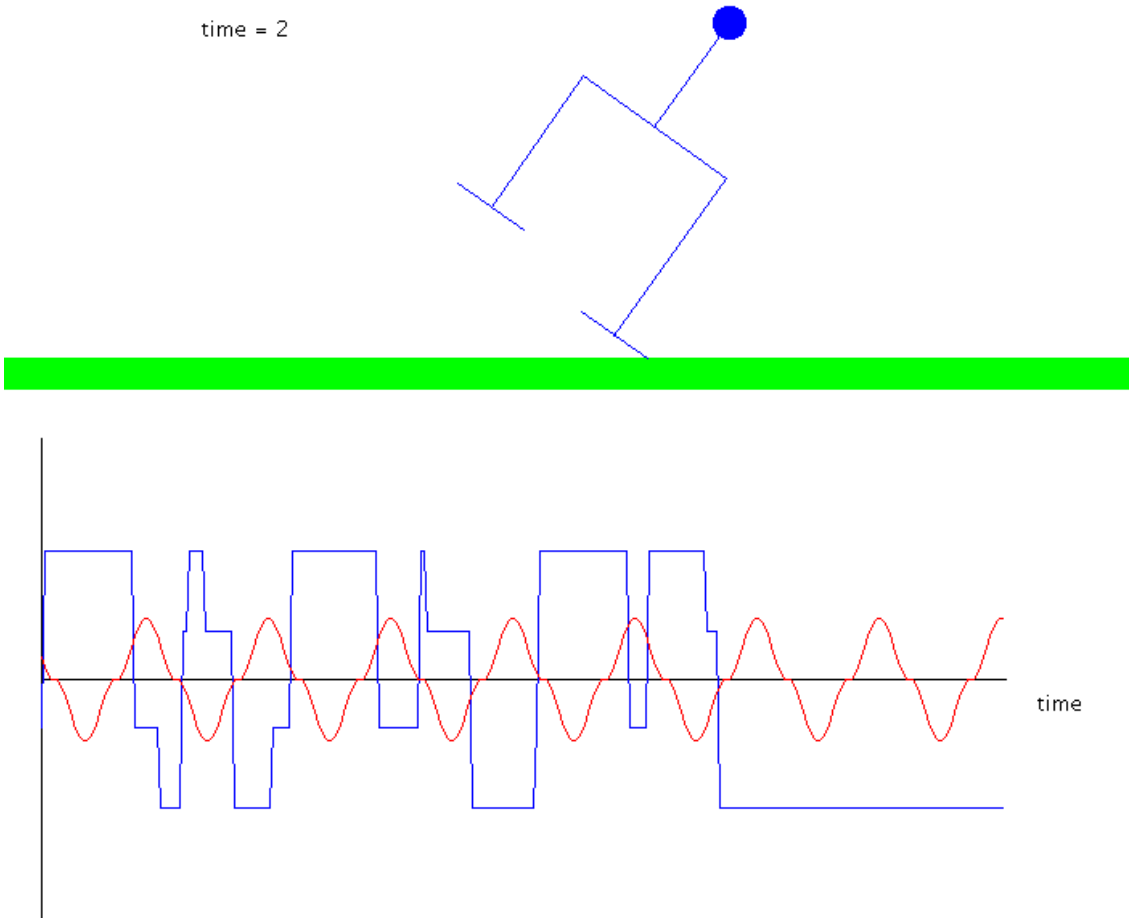


Figure 7: The simple inverted pendulum model of the Nao showing an example CoP (blue) and foot-lift (red) time-series for an open loop walk.

The feet are inclined so as to keep them parallel to the ground plane. For a walk at constant velocity the CoP should stay between the rear and front edges of the stance foot with the torso in a vertical position. If the robot sways forward or backward, (α in the figure) we wish to correct this so that the position and velocity of any sway is zero. A reinforcement learning controller was developed to achieve this in simulation. The optimal policy was approximated by making the ankle tilt proportional to the gyroscope Y reading. The details are further described in a separate report in preparation.

5.2 Feedback Control

Both the coronal and sagittal dynamics of the walk have been stabilised to reduce the incidence of falling. We next describe both the method and results for the stabilisation of the walk in both coronal and sagittal planes with the simulator and for the real Nao robot.

Coronal Rock Stabilisation The coronal rock is stabilised by synchronising the onset of the leg-lift motion with the switch in stance and support foot. We switch the stance and

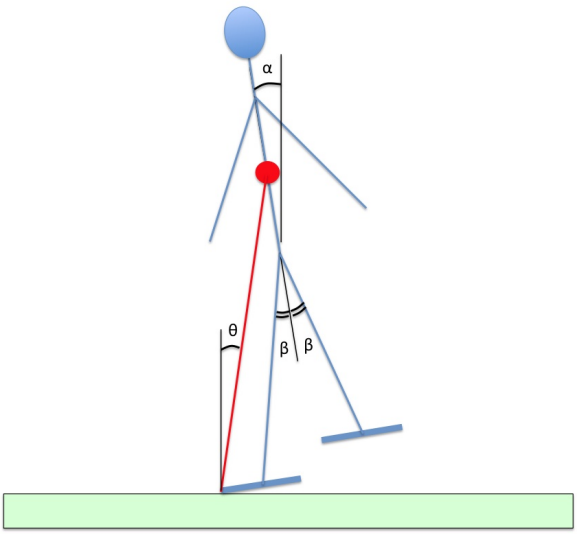


Figure 8: The simple inverted pendulum model of the Nao for the sagittal plane.

support feet by observing the zero-crossing of the measured CoP. The CoP is calculated in the coronal plane with the origin in the middle of the robot between the feet. It is negative when the robot stands on the right foot and positive when it switches to the left foot. The period that the robot spends on the stance foot cannot be determined precisely when there are disturbances such as uneven surfaces, play in motor gears, dirt on the ground, and bumping by other robots. The zero-crossing point of the CoP indicates that the robot is in the process of shifting it weight to the other leg. We use it to reset the starting time for both the left and right swing phases of the walk cycle. The code is reviewed in detail in Section 5.4 item 6.

The CoP and leg-lift time series for the closed-loop coronal rock is show in Figure 9. In comparison with Figure 7 it can be seen that the motion has a more consistent period with the onset of the leg-lift action either delayed or brought forward.

The same controller running on the real Nao produces the time-series for the CoP and leg-lift as shown in Figure 10. It is easy to see the similarity between the results from the simulation and from real robot, even though the inverted pendulum model is very basic. The real Nao was tested on a felt carpet which may explain the ragged edges on the CoP measurement over 8 foot sensors.

Sagittal Stabilisation The real Nao robot is stabilised in the sagittal plane by tilting the ankle joint in direct proportion to the filtered Gyro Y value. The code is explained in Section 5.4 item 7., and the subject of a separate repot in preparation on machine learning sagittal stabilisation using reinforcement learning. While the controller implementation is simple, the theory behind its action is based on a forward controller, akin to preview control, that angles the stance foot in such a way as to decelerate any sway and bring the robot torso to move at a constant velocity as dictated by the *forward* speed setting.

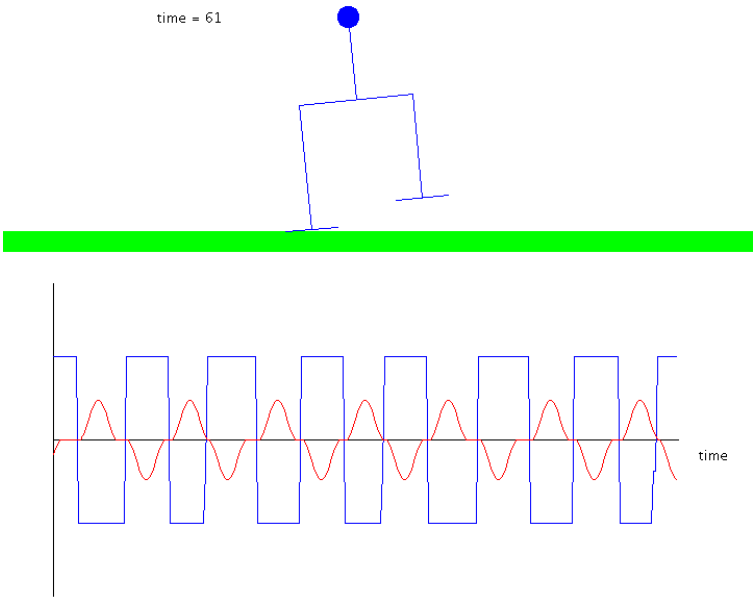


Figure 9: Simulated closed-loop coronal rock using the CoP zero-crossing point. The time series show the CoP and the leg-lift actions over time.

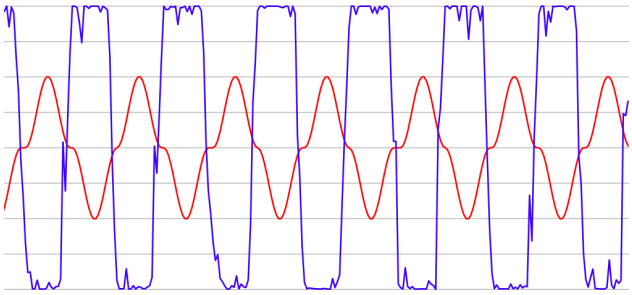


Figure 10: Real Nao closed-loop coronal rock using the CoP zero-crossing point.

5.3 Inverse Kinematics

Determination of appropriate joint angles to position the feet requires inverse kinematic calculations to be performed. Inverse kinematics determines the robot's joint angles given the desired 3D position of the kinematic chain end-points such as the feet. Walk2014 uses two different methods for calculating the joint angles for the walk. A closed-form is used for moving the foot forward, back or sideways. An iterative method for turning is used which involves the more complex hip-yaw joint mounted at 45 degrees to the torso in the coronal plane. We first describe closed-form kinematics, followed by the iterative method.

Close-Form Inverse Kinematics

The stance foot is assumed to be flat on the ground and determines the relative location of all the other joints for a given set of joint angles. To calculate the joint angles required for walking, we use a coordinate frame centered on the hip joint. The following Walk2014

stance variables are sufficient to describe the state of the walk at any point in time:

- the x-position of the ankle joint of each foot in metres in the forward direction relative to the hip. We use the variables *forwardL* and *forwardR* for the left and right foot respectively. Because the CoM of the robot in the x-direction is located more to the back of the foot, the robot falls backwards more easily. An innovation in 2014 was to move the CoM of the robot more towards the middle of the foot to increase the ability to balance the robot sagittally with its flat feet. The shift of the CoM towards the middle is determined by the setting of the variable *comOffset* in the code.
- the lift of the center of each foot above the ground plan in meters directly below the hip-joint (*leghL, leghR*).
- The displacement of the foot in the y-direction when stepping sideways (*leftL, leftR*).

When the *turn* is zero we can use closed-form inverse kinematics to calculate the hip-pitch, knee-pitch, and ankle-pitch of each leg and foot. This is possible because of the unique constraints in Nao's kinematic leg chains. The hip-pitch, knee-pitch and ankle-pitch motors axes of rotation are parallel. Each leg therefore moves in a plane. Figure 11 shows a diagram to visualise the 3D geometry of Nao's legs. We next detail the derivation of all the joint angles. The $*$ symbol in the expressions can be substituted by either L or R for the left or right leg respectively.

We first calculate the vertical distance between ankle and the hip as the hip height above the ground less the distance the foot is lifted, less ankle height above the ground when the foot is on the ground: $legh^* = hiph - footh^* - ankle$.

We extend the leg a little further when sidestepping: $legX0^* = legh^*/\cos(left^*)$.

We include $forward^*$ and *comOffset* resulting in the final leg extension:

$$legX^* = \sqrt{(legX0^*)^2 + (forwardL + comOffset)^2} \tag{1}$$

Given the final leg extension between the hip-joint and ankle-joint $legX^*$ we can calculate the angles $beta1^*$ and $beta2^*$, as shown in Figure 11, using the cosine rule to determine the amount the knee-joint needs to bend.

$$beta1^* = \arccos(thigh^2 + legXL^2 - tibia^2)/(2 * thigh * legXL) \tag{2}$$
$$beta2^* = \arccos(tibia^2 + legXL^2 - thigh^2)/(2 * tibia * legXL) \tag{3}$$

The final hip-pitch is the sum of the angle due to the knee bend plus the angle required to move the leg forward as shown in Figure 11. The ankle-pitch is determined similarly to keep the foot parallel to the ground. The knee-pitch is the sum of the hip and ankle pitch.[1]

This completes the closed-form inverse kinematic calculations for forward and sideways movement of the legs. We now address the case when there is also a turn component that is required to change the direction of the robot when walking.

[1]When the foot displacement is significantly behind the hip so that the thigh slopes backwards, the calculations need to be adjusted slightly in sign as reflected in the code.

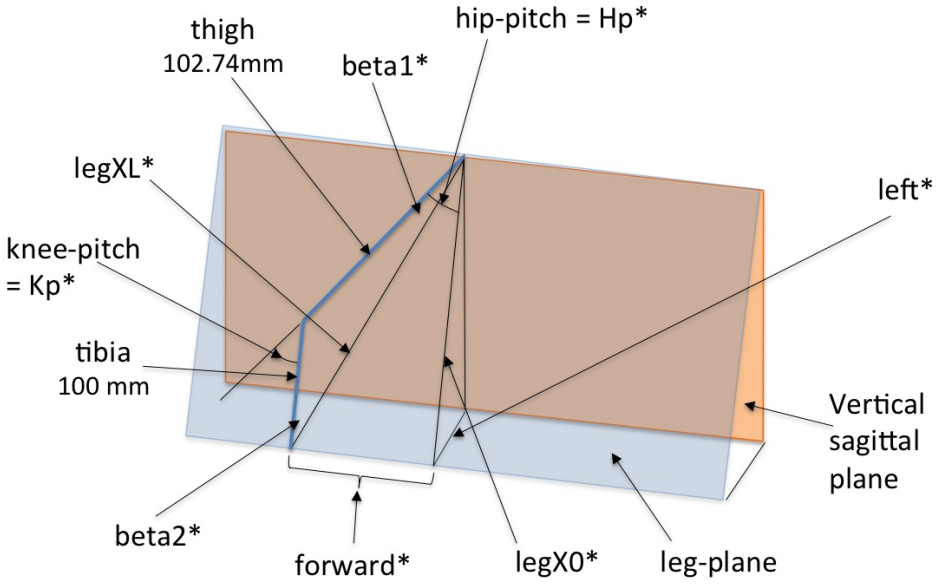


Figure 11: Geometry of leg position and angles. The * is replaced by either L or R to signify left or right leg variable in the code.

Iterative Inverse Kinematics when Turning

The hip joints in the Nao are complex in that the hip-pitch, hip-roll and hip-yaw motor axes coincide. The hip-yaw motors in the Nao are required for turning and are inclined at 45 degrees to the torso in the coronal plane. In addition, the left and right motors are constrained to move together by the same amount of rotation — see [19]. This increases the complexity of the inverse kinematics for foot placement.

Our approach is to determine the new hip-pitch and hip-roll angles that leave the position of the ankle-joint unchanged relative to the hip-joint for a rotation of the hip-yaw joint. In this way we can execute a movement of the foot forward and sideways first and subsequently rotate the foot about its center to complete the omni-directional turn. The knee-pitch is left unchanged as the leg is one unit. The ankle pitch and roll is calculated to ensure that the foot is kept parallel to the ground.

We use a similar iterative inverse kinematic method to that used by Bowdoin College in 2009 based on [1]. The idea is to iteratively guess the joint angles and perform a forward kinematic calculation until we get close enough to the desired position. The "guesses" can be improved by considering the local effect on the change in position of the action of each

joint movement. This is given by a matrix of partial derivatives called the Jacobian. In this way the guesses can be directed and the number of iterations minimised to achieve the desired final position of the chain. In practice we have found that only three iterations are necessary.

In particular we use the forward kinematic transform mapping ankle positions into the hip coordinate frame of reference. The derivation of the mDH parameters for the forward kinematic transform is given in Appendix B, Appendix D and Figure 18. The Matlab code for the iterative inverse kinematics is provided in Appendix E and uses Equation 4 (refer to [1]) to estimate the change in hip-pitch and hip-roll (vector $\Delta\theta$).

$$\Delta\theta = J^T(J * J^T + \lambda^2 I)^{-1} * \overline{e} \tag{4}$$

where J is the Jacobian providing the change in angle with respect to position, λ is a constant (0.4), and $\overline{e}$ is the displacement between the current position and the desired position. To improve processing speed on the Nao Equation 4 was expressed symbolically with Matlab and the expressions copied into the C++ code.

We still need to determine the ankle pitch and roll to keep the foot parallel to the ground. Having determined the location of the ankle joint relative the hip joint, we can use the previous foot-to-hip forward kinematic transform to find the relative position of two points on the foot that are not collinear with the center point and work out the change in ankle-pitch and ankle-roll to ensure the foot is parallel to the ground. To minimise computations we transform two orthogonal unit vectors from the center of the foot along the foot x and y axes. The ankle angle adjustments are then simply $\sin^{-1}\delta z$, where δz is the change in the z coordinate moving along each of the unit vectors.

This completes the inverse kinematics for one of the legs. The other leg is symmetrical, and we reuse the co-ordinate transform by simply inverting the sign of the hip-roll and ankle-roll. We repeat the above iterative inverse kinematic procedure for the other leg to complete the full inverse kinematic calculations for all leg and foot joints for the omni-directional walk.

5.4 Stepping Through the Code

We enumerate the steps in the overall Walk2014 process following the structure of the *Walk2104Generator.cpp* source-code. Variables are described and initialised at the beginning of the source code. The primary method *makeJoints* sets the values and stiffness of each of the body joints every 100th of a second. We only document the walk in this report (not the kicking actions). Item numbers below correspond to comment numbers in the code.

0. The robot stands at different heights depending on whether it is walking or resting. The first time the Walk2014 generator is called, the stance height of the robot is not known and is calculated from the sensed bend of its knee joint. In this case the *walk2014Option* is still at its initialised value of *NONE* and under this condition we set the current hip height *hiph*.

1. The variable t is used as a timer for each phase of the walk. When the phase terminates t is reset to zero. When the robot is not in motion t is kept at zero. When t is zero the walk

is allowed to read in a new set of command values. The values relevant for walking are *forward* (meters/sec), *left* (meters/sec), and *turn* (radians/sec). The *bend* parameter indicates whether there robot should bend its knees or not. With knees straight it can stand and turn down the stiffness of its motors. With knees bent it can stand, reduce camera motion blur, but be ready to spring into action quickly. The *power* parameter is used to control stiffness of the motors or kick strength. These parameter values are passed to the Walk2014 generator from Python skills and behaviour components higher in the task-hierarchy.

1.0 To assist the writing of Python behaviours we introduced a *stand* and *crouch* action command in *actioncommand.py* in addition to *walk* that commanded the robot to stand still with knees straight and bent respectively. This line of code ensures backwards compatibility with previous behaviour code to set the knee bend. It can be ignored (and removed) with Python behaviour written in the future.

1.1 We limit *forward*, *left*, and *turn* values individually to maximum and minimum values in case the behaviour code make demands that are beyond the capability of the robot. This clipping function does not ensure that a combination of values is within the capability of the robot, but this is left to behaviour.

1.2 T is the total time set for each walk phase set at 0.23 seconds. When sidestepping it is increased by up to 0.05 seconds proportionately to take into consideration the longer recovery time of the inverted pendulum.

1.5 Walk2014 is designed to immediately respond to changes in *forward*, *left*, and *turn* values at the start of each phase. Very rapid changes in the larger *forward* values can cause some robots to become unstable and fall over, for example when going from full- speed forward at 30cm/sec to full-speed in reverse at -30cm/sec. The make speed change more gradual we limit the absolute maximum value of the speed change to *FORWARD_CHANGE* - set to 20 cm/sec. This can be increased or removed altogether but with a higher risk of falling. This code has been commented out because ratcheting is now handled in *actioncomand.py* and adjusted depending on the wear of the robot.

1.6 The commanded *forward*, *left*, and *turn* are velocities that the robot should achieve while walking. We first convert these velocities to step sizes in meters and radians by multiplying them by the time period of a walk cycle ($2 * T$). The robot slips while walking. The values are further multiplied by a fraction to scale them to achieve the commanded velocities in practice.

2. Updates the timers by the duration of the time-step $dt = 1/100$th of a second.

3. Determine the *Walk2014Option* or *walkState* for the next walk phase. A *walkState* is a sub-option of a *Walk2014Option*. The non-kicking *Walk2014Option*s are:

STAND: knees straight and stiffness set to zero to conserve energy

STANDUP: process of moving from WALK crouch to STAND height

CROUCH: process of transitioning from STAND to WALK height

WALK process of walking

READY: stand still with knees bent ready to walk

NONE: initial option

*walkState*s can be WALKING, STARTING, STOPPING, and NOT_WALKING.

If a WALK is commanded to stop walking by setting the *forward*, *left*, and *turn* values to zero, we flag a STOPPING *walkState* for the next walk phase.

If the walk is commanded to stand with knees not bent (i.e. *bend*=0), then we STANDUP, and once we reach the *STAND_HIP_HEIGHT* we set the *Walk2014Option* to STAND.

Equally if we command the walk to stop walking with knees bent ,we CROUCH until we are READY.

4. This section of code executes the walk option as determined above.

STAND: The walk parameters are zero. The robot is standing up straight with the stiffness set to the *power* parameter, with default stiffness 0.1. This minimum is required to stop the robot from collapsing given a sight disturbance. STAND is used by behaviour when the robot is not required to walk and is useful for achieving a steady and maximum height view, but importantly to reduce the temperature increase of the motors.

STANDUP: The walk parameters are zero and motor stiffness is set to its maximum value. The robot stands up from its current hip height *hip0* to the *STAND_HIP_HEIGHT*. This motion uses a *parabolicStep* function that is generated with constant acceleration followed by constant deceleration to reach a target position. The motion is timed with a *timer* to last *CROUCH_STAND_PERIOD*. When walking, *comOffset* moves the CoM of the robot closer to the center of the foot in the x-direction. The intention is to allow the robot to better balance with its flat feet. On standing up, the *comOffset* is gradually reduced to zero, so that the robot's weight is over its ankle joint and can be supported with minimum stiffness. Timer t is kept at zero ready to accept new walk commands.

CROUCH: Similar to STANDUP except the robot moves to the *WALK_HIP_HEIGHT*. The *comOffset* is increased again.

WALK: Stiffness is set to 1 (maximum value) while walking.

READY: Similar to STAND, except that the robot is crouched, ready to start walking. Minimum stiffness is 0.4 to ensure the robot does not collapse at in this stance.

5. This section of the code determines the changing walk variables that parameterise the walk throughout the walk phases. It is executed whenever *walk2014Option* = WALK and the phase time t is non-zero.

5.1 The maximum height to which the swing foot is lifted in the vertical direction is a base height of 10mm plus two additional components proportional to the size of the *forward* and *left* walk parameters. The locus of the swing foot in the vertical direction is traced out using motion segments at constant acceleration (i.e. it consists of two parabolicStep functions described above). The parabolic nature of the position is derived by integrating the constant velocity twice.

5.2 Deleted code. It was thought that when walking in an arc the outside foot needs to travel further than the inside foot and an adjustment to *forward* would be necessary for both the swing an stance foot. The ground distance travelled by each foot will vary, but relative to the body frame no adjustment is necessary. This code has therefore been deprecated.

5.3L Walk2014 consists of two swing phases without a double support phase, similar to previous small humanoid gaits [5] [8]. The left and right swing phase code is similar. We will step through the left swing phase code which is activated when *bodyModel.isLeftPhase* is true.

5.3.1L *forwardL* and *forwardR* are the position of the left and right ankle joints respectively, relative to the hip in the x-direction. The stance foot position, *forwardR* , moves backwards relative to the hip from the position it was in, *forwardR0*, at the end of the last walk phase. This motion is at a constant velocity, using the *linearStep* function and based on the total distance to travel give the latest commanded *forward* walk parameter and the time available to complete the phase, *nextFootSwtichT*.

5.3.2L Kicking and in particular the left *jab kick* will be covered in another report.

5.3.3L Walk2014 does not use a sideways lean. This is a placeholder for future use.

5.3.4L When walking sideways we need to determine the position of the feet in the y-direction at each time-step, i.e. *leftR* and *leftL*. With the left food as swing foot, if we wish to move right, the right stance foot pushes the body right, *leftR*, as a parabolic step function of the commanded *left* command. This is handled by the *leftAngle* function. The displacement is only half the commanded *left*, as the other half will be symmetrically accounted for by the left leg - the swing foot. The position of the left foot, *leftL*, is simply $-leftR$.

In case of a commanded reversal in sideways direction, the left movement is adjusted to cancel out any residual *swingAngle* from the previous left movement. For example if we reverse the sideways walk from $left = -0.1$ with the legs apart to $left = 0.1$ the robot will not waist time to bring the legs together first.

I few wish to move right when the left foot is the swing foot, the positions are determined in a similar way, except in this case there is no adjustment possible with a reversal of direction. The only option is to bring the feet together in this phase.

5.3.5L The *turn* command moves the hip-yaw joint of the robot to effect the turn. We use an interruptive inverse kinematic calculation to determine all the joints to rotate the leg and keep the foot parallel to the ground. This is explained in more detail below. In this part of the code, the amount of turn is controlled by the *turn* parameter. The walk recovers any turn left over form the from the previous phase, $turnRL0$, and turns the robot in a parabolic step fashion either with toes pointing together (.33 of the *turn*) or apart (.67 of the *turn*).

5.3.6L The swing foot is set to a height of $varFootHeight$ (see item 5.1). The stance foot height is zero as it is on the ground.

5.3R The code is similar to above except with the swing foot the right foot.

5.4 With the $walkState == STARTING$ we are priming the sideways rocking motion from the READY $Walk2014Option$. For this priming operation we keep the foot positions at zero and reduce the height to which the feet are lifted by a factor of 3.5, just enough to start the sideways rocking motion.

5.5 The arm swing is made proportional to the leg movements in the sagittal plane.

6. This section of code controls the phase change with the interchange of the swing and support foot. The change is triggered under two conditions:

1. When the centre-of-pressure (CoP) changes sign in the y-direction and the current walk phase is at least 3/4 of the way through its designated period, T, and

2. If we exceed three times the designate phase period and no change in the sign of the CoP is detected.

The first condition ensures that we do not react to momentary bounces in the sign of the CoP as it passes through zero. The second condition tries to rescue a robot that is stuck on one leg.

If we detect a change in support foot based on the above criteria we set this in the bodyModel to engage the correct kinematic chain, and if the $Walk2014Option$ is WALK we set several variables that need attention when the support foot changes. We describe these below:

6.1 The *swingAngle* which is the angle that the leg has swung during a sideways motion is stored.

6.2 Decide on the period of the next walk phase and set any walk state. The period, $nextFootSwitchT$ in Walk2014 is always T. The walk state changes from NOT_WALKNG to STARTING, and from STOPPING to NOT_WALKING.

6.3 The phase timer, t is reset to 0.

6.4 Back up previous phase forward and turn values.

7. Sagittal balance is controlled by a filtered Gryo Y value. The ankle pitch is set proportional to this value. The research and explanation behind this approach is the subject of another report. The effectiveness of this method was born out in practice in RoboCup 2014 in Bazil, and is consistent with previous work on bipedal locomotion [5].

In the READY $Walk2014Option$ the feedback is turned off to arrest a "hunting" oscillation evident with some robots in this stance position.

8. This function updates the odometry used by the robot localisation function at 1/100 second intervals. It is noteworthy that the update follows the non-linear parabolic step functions that move the robot during walk phases. The accumulated values are calibrated to match the actual speed of the robot, and not the theoretical values.

9. Determine the joint angles from the stance variables using inverse kinematics as described in Section 5.3.

10. Sets and returns the joint angles and stiffness, completing the *makeJoints* method.

5.5 Walk2014 Footstep Response to Change in Walk Parameters

One of the major advances in Walk2014 over the previous walks is that the change in walk parameter settings takes effect immediately at the start of the next walk phase. There is only a delay of between 0 and 0.23 seconds from the time the command was given by behaviour. Previous rUNSWift walks used a ratcheting mechanism that gradually changed the walk parameters, potentially over several walk cycles, to effect any changes, but see Section 5.4, item [1.5].

Foot positioning is optimised when changing direction, for example when switching from walking left to walking right. This means that the Walk2014 does not necessarily transition through a state where the feet are both together, but may rock with the feet apart when changing direction. We next illustrate the change in walk variables in response to a change in walk commands one at a time. In combination they operate in the same manner independently and concurrently to achieve omni-directional locomotion.

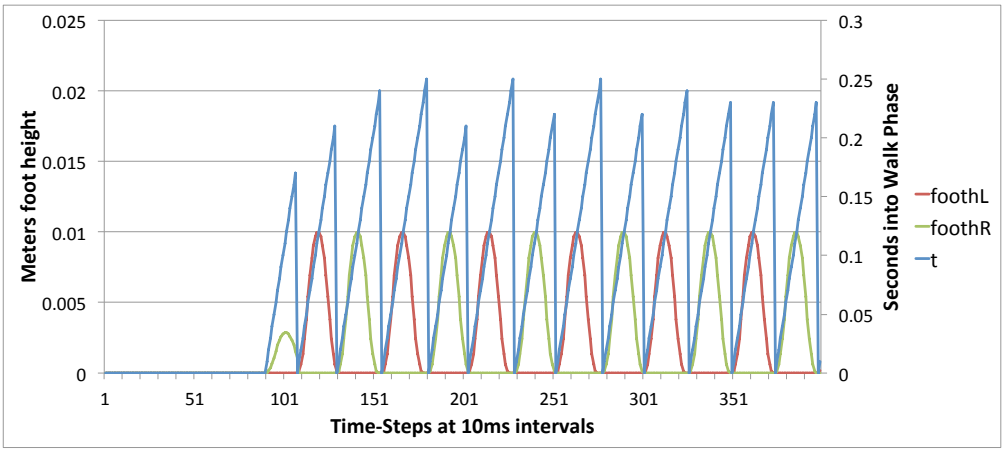


Figure 12: Basic Walking Pattern showing the phase timing did foot lift heights.

Figure 12 shows the initiation of a walking pattern from the READY state. The blue sawtooth graph shows the variable t after the walk is initiated. It resets at the beginning of each phase of the walk when the centre-of-pressure changes sign indicating a change in support foot. Each phase is of a slightly different duration due to noise in the rocking behaviour. The red and green parabolic-like plots show the lift in the left and right foot respectively. The first time a foot is only lifted it is only lifted to about 30% of it walking height with the effect that it initiates the sideways rocking motion from a standing start. This also makes the initial phase shorter than the following ones.

Figure 13 illustrates behaviour commanding the walk to turn counterclockwise and then reverse direction at the same turn speed for several steps. The blue graph shows the value of the *turnRL* variable in radians in relation to the foot-steps in green and red from the previous graph for reference. The turn is not activated until after the sideways rock is initiation. The outward turn of the feet is greater than the inward turn of the feet (see code item 5.3.5 above). When the direction of turn is changed the hip-yaw joint (*turn RL*) is not reset to

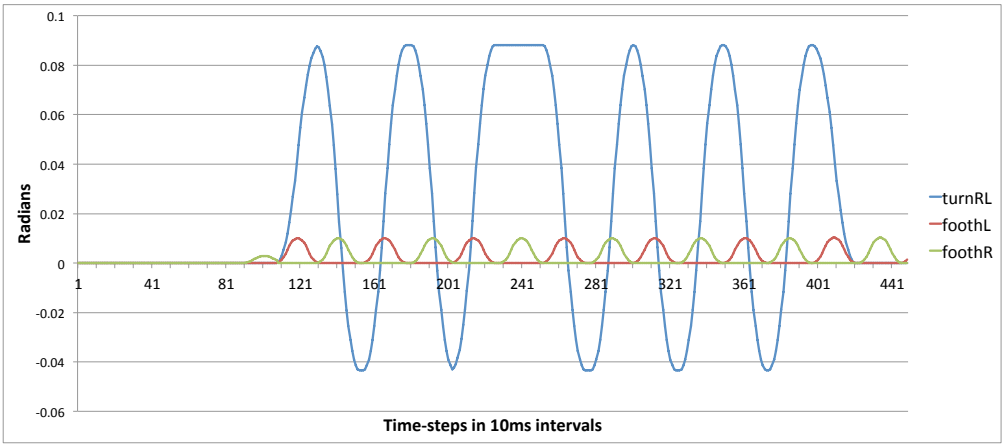


Figure 13: Turning counterclockwise and then clockwise.

zero, but maintains the feet apart while the walk changes support foot. At the beginning the turn motion starts with both feed together and is reset to this position at the conclusion of the turn.

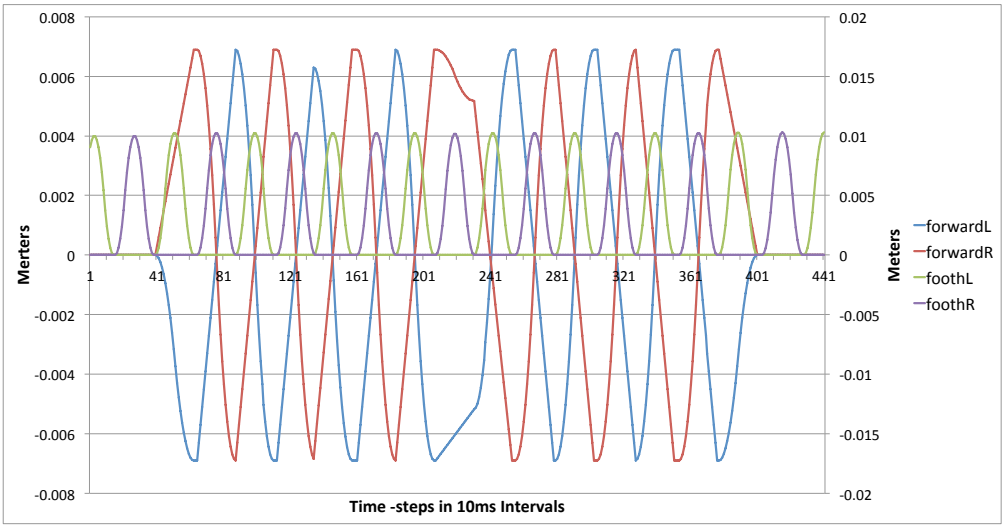


Figure 14: Walk forward and then reverse.

Figure 14 shows a walk that first starts to walk forward and then reverses direction and walk backwards for a few steps. The walk variables $forwardL$ and $forwardR$ are shown in blue and red respectively, with the left and right foot steps shown in green and purple for reference. When reversing direction the walk makes a slight adjustment due to ratcheting the step-size but does not return the feet to a zero position. It is also evident from the graph that the swing foot moves in a parabolic fashion while the support foot moves at a constant velocity with respect to the body of the robot. Starting and stopping the forward/backward motion is smooth and achieved within one phase of the walk.

Figure 15 shows a sideways walk to the left followed by the sideways walk to the right. The red and blue time-series show left and right leg sideways displacements of the foot. In this case the timing of the reversal in direction is such that the optimum way to reverse direction

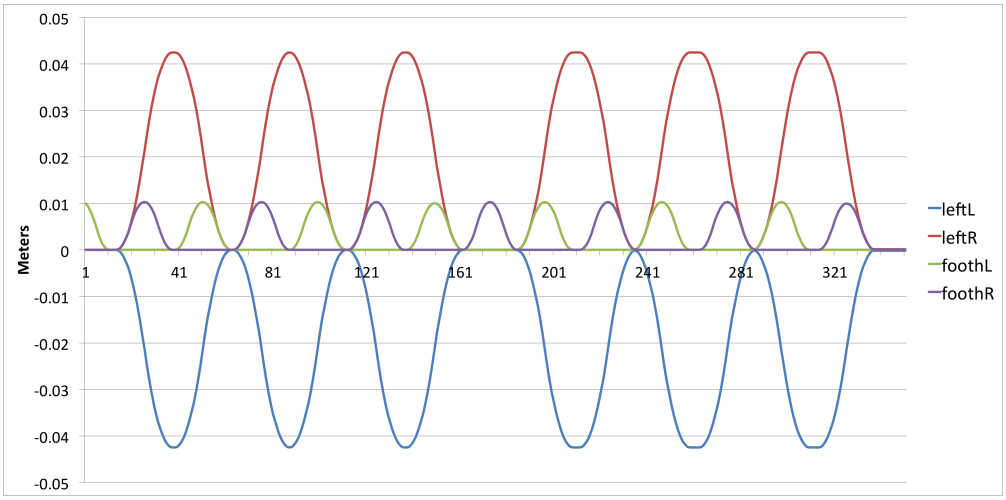


Figure 15: Walk left and then right.

is bring both feet together for one phase. If on the other foot at the time of reversal, the walk would pause with legs apart before resuming the walk in the the other direction.

A Omni-directional Walking Parameterisation

The foot position is parameterised in relation to the hip-joint by *forward*, *left*, and *turn*. Figure Figure 16 shows how these parameters position a foot.

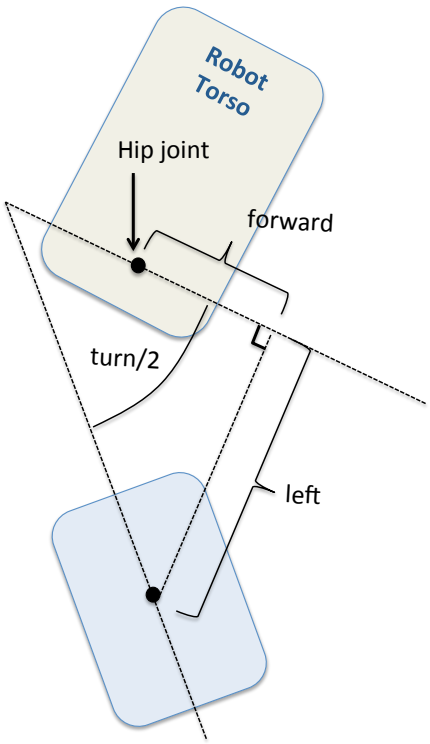


Figure 16: Omni-directional foot placement

B Kinematic Transforms for Nao Robot

C Kinematic Denavit-Hartenberg convention(D-H)

[2] The Denavit-Hartenberg convention is a set of mathematical rules describing the relative position of coordinate systems for links in a kinematic chain. The proximal DH convention, also called the modified DH convention is described below. It is this convention that we have used. There is sometimes confusion because the major part of the robotics literature uses the so-called distal convention (which is also called standard convention).

For the modified DH convention the axes are chosen according to the following rules:

[2]This appendix is a reproduction of a section of the notes from [21] with minor editing changes.

1. The robot consists of $n+1$ links $G_0, G_1, ..., G_n$.
2. Links are connected through joints $j_1, j_2, ..., j_n$, with joint j_i connection link G_{i-1} and G_i.
3. For each pair of successive joints j_i, j_{i+1}, we can find a common perpendicular l_i.
4. The z_i-axis is chosen to lie along the joint axis of joint i, so $z_i = j_i$ (and the origin also lies on j_i).
5. The x_i-axis is perpendicular to both j_i and j_{i+1} (or equivalently to z_i and z_{i+1}) and points towards the next joint.
6. The y_i-axis is chosen such that x_i, y_i, z_i form a right hand system.
7. The origin O_i of frame i is chosen as intersection between l_i and j_i resp. z_i.

There are some special cases that are not explained sufficiently through above rules:

- If two joint axes are parallel, the common perpendicular is not defined uniquely — it is obviously possible to translate the common perpendicular along the direction of the joint axes. This also means that the origin of the coordinate system can be chosen arbitrarily, at least in theory. In practice, origins are chosen such that as many DH parameter as possible are $= 0$.
- If two joint axes intersect, the direction of the common perpendicular is not unique. According to Craig [3], it is recommended to choose the direction such that it points in the same direction as the following x-axis.
- The first coordinate system x_0, y_0, z_0 can be chosen freely. If possible, it is however chosen such that it coincides with the system z_1, x_1, y_1 if $\theta_1 = 0$ resp. $d_1 = 0$ (for rotational resp. translational joints).
- For the final system x_n, y_n, z_n, the origin of the system (along the joint axis) as well as the direction of x_n can be chosen freely. But again, this is usually done such that as many parameters of the DH convention as possible are 0.

The meaning of the parameters will now be illustrated by explaining how the corresponding transformations reflect the transfer between successive coordinate frames:

1. a_i is the distance between z_i and z_{i+1} w.r.t. x_i, or the distance between joint axes j_i and j_{i+1} w.r.t. their common perpendicular.
2. α_i is the angle between z_i and z_{i+1} w.r.t. clockwise rotation around x_i, so the angle between joint axes j_i and j_{i+1} w.r.t. to their common perpendicular.
3. d_i is the distance between xi_1 and x_i w.r.t. z_i, so the distance between l_{i-1} and l_i along their common joint axis j_i.

4. θ_i is the angle between x_{i-1} and x_i w.r.t. clockwise rotation around z_i, so the angle between l_i and l_{i-1}.

The parameters are denoted by $a_0, ..., a_{n-1}, \alpha_0, ...\alpha_{n-1}, d_1...d_n, \theta_1, ..., \theta_n$. This is why the parameters are usually written down as follows:

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	a_0	α_0	d_1	θ_1
2	a_1	α_1	d_2	θ_2
...	...	...	...	...
...	...	...	...	...

Table 1: Table of modified D-H parameters

The transformation from system i_{i-1} to system i can now be decomposed into separate transformations as follows:

1. Translate the system along z_i-axis with offset d_i.
2. Rotate the system around the z_i-axis with angle $theta_i$.
3. Translate the system along x_{i-1}-axis with offset a_{i-1}.
4. Rotate the system around the x_{i-1} with angle α_{i-1}.

Overall, we retrieve the following transformation matrix:

$$ {}^{i-1}_{i}T = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & \alpha_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1} d_i \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5} $$

where the transformation is computed as matrix product of:

$$ {}^{i-1}_{i}T = R_X(\alpha_{i-1}) D_X(\alpha_{i-1}) R_Z(\theta_i) D_Z(d_i) $$

Here R_X, R_Z are rotations w.r.t. the corresponding z- resp. x-axes, and D_X as well as D_Z are translations along those axes.

D Kinematic Chains for Nao

The diagrams show a schematic for all the Nao's joints and the mDH parameters in the previous section for three kinematic chains, one from the left-ankle to hip, one from the bottom-camera to left- foot and the inverse from the left-foot to the bottom-camera. While the latter two coordinate frame transform matrices are the inverse of each other, we derived each separately to save calculating the inverse.

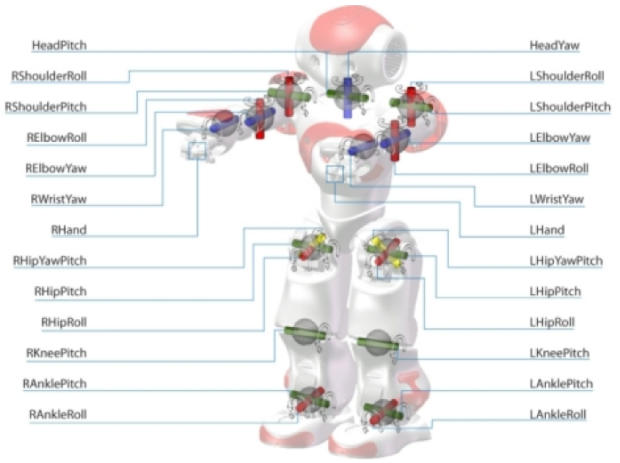
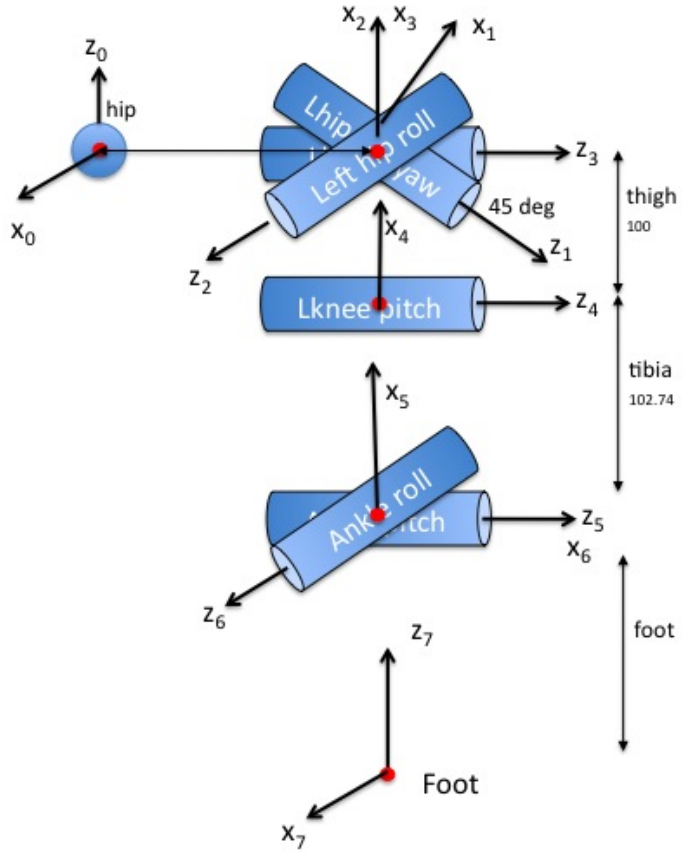


Figure 17: Nao's joints

Supporting definitions for forward kinematic calculations for chains between foot to bottom camera:

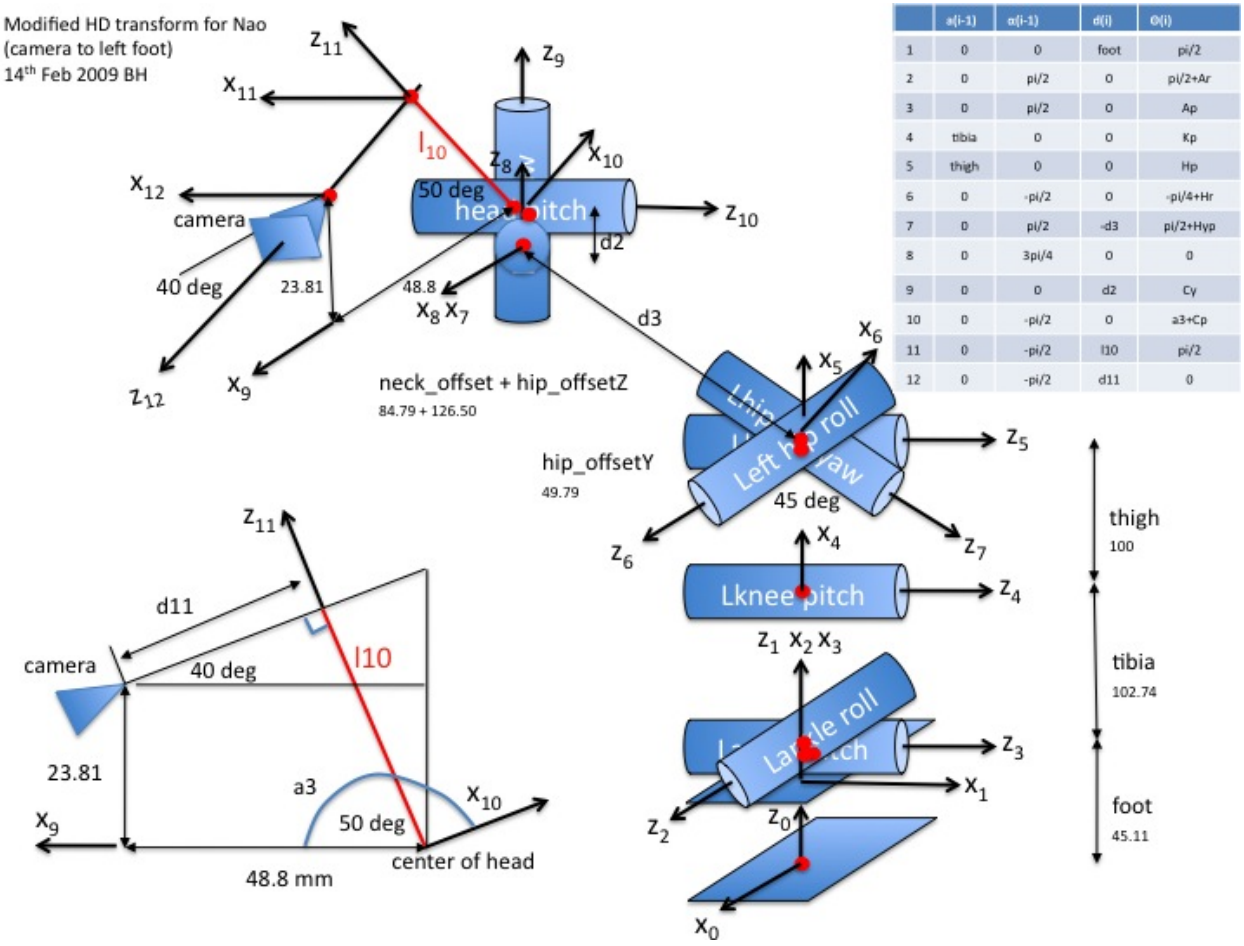
$$
\begin{aligned}
foot &= 45.11 && (6)\\
tibia &= 102.74 && (7)\\
thigh &= 100.00 && (8)\\
hip &= 49.79 && (9)\\
hip\_offsetZ &= 84.79 && (10)\\
neck\_offsetZ &= 126.50 && (11)\\
trunk\_length &= hip\_offsetZ + neck\_offsetZ && (12)\\
camera\_out &= 48.80 && (13)\\
camera\_up &= 23.81 && (14)\\
d1 &= sqrt(camera\_out^2 + camera\_up^2) && (15)\\
d2 &= trunk\_length - hip && (16)\\
d3 &= hip * sqrt(2) && (17)\\
a1 &= atan(camera\_up/camera\_out) + deg2rad(40) && (18)\\
a2 &= atan(camera\_up/camera\_out) + pi/2 && (19)\\
l10 &= d1 * sin(a1) && (20)\\
d11 &= d1 * cos(a1) && (21)\\
a3 &= deg2rad(40) - pi && (22)\\
& && (23)
\end{aligned}
$$



mDH transform parameters for Nao (Left ankle to Left hip) 10th Feb 2009 BH

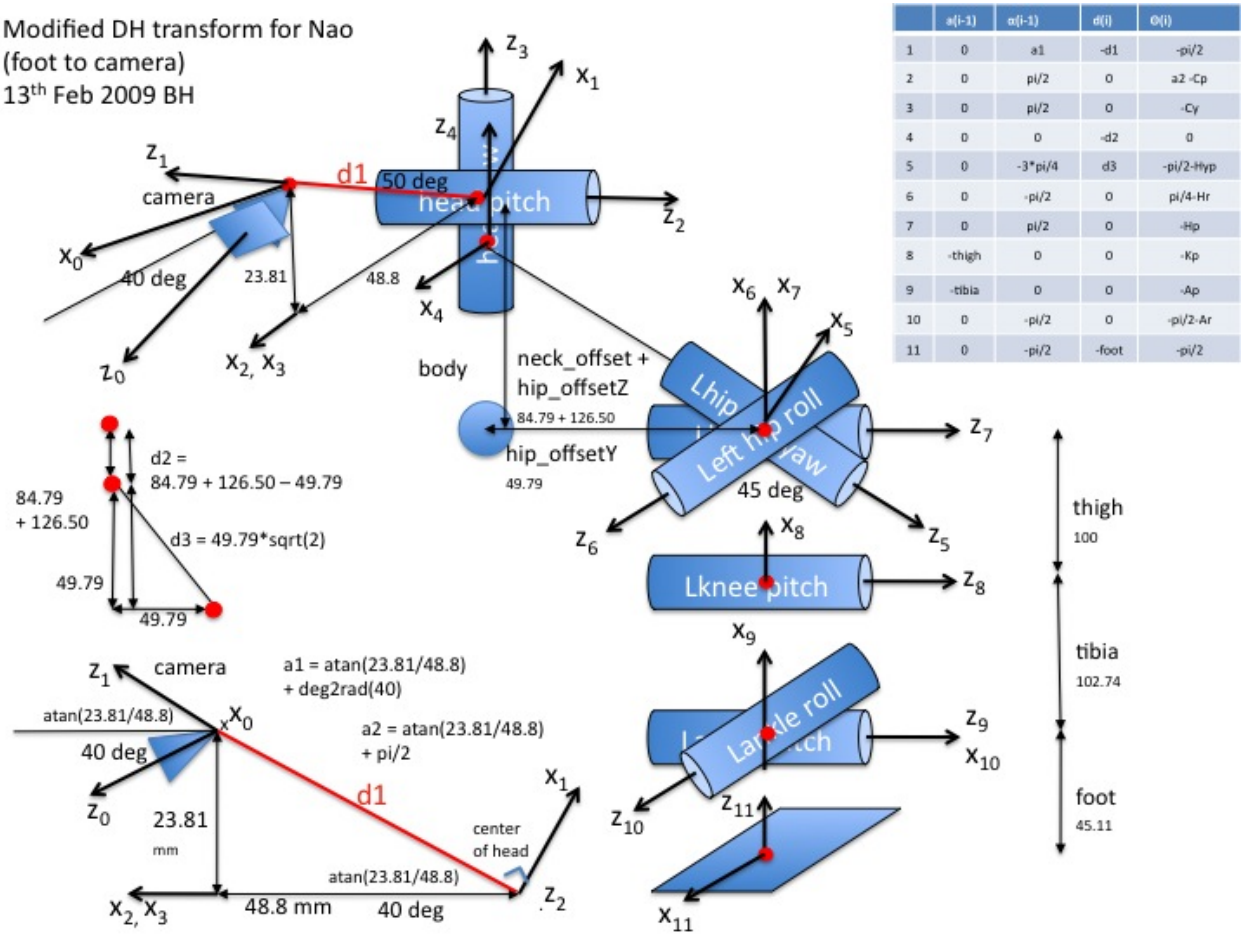
Joints 0	a(i-1)	α(i-1)	d(i)	Θ(i)
1	0	-3*pi/4	0	-pi/2 +Hyp
2	0	-pi/2	0	πpi/4 +Hr
3	0	pi/2	0	0 +Hp
4	-thigh	0	0	0 +Kp
5	-tibia	0	0	0+Ap
6	0	-pi/2	0	-pi/2 +Ar
7	0	-pi/2	0	-pi/2

Figure 18: mDH parameters transforming the coordinate frame of the left-ankle to the left-hip



	a(i-1)	α(i-1)	d(i)	Θ(i)
1	0	0	foot	pi/2
2	0	pi/2	0	pi/2+Ar
3	0	pi/2	0	Ap
4	tibia	0	0	Kp
5	thigh	0	0	Hp
6	0	-pi/2	0	-pi/4+Hr
7	0	pi/2	-d3	pi/2+Hyp
8	0	3pi/4	0	0
9	0	0	d2	Cy
10	0	-pi/2	0	a3+Cp
11	0	-pi/2	l10	pi/2
12	0	-pi/2	d11	0

Figure 19: mDH parameters transforming the coordinate frame of the bottom-camera to the left-foot



	a(i-1)	α(i-1)	d(i)	Θ(i)
1	0	a1	-d1	-pi/2
2	0	pi/2	0	a2 -Cp
3	0	pi/2	0	-Cy
4	0	0	-d2	0
5	0	-3*pi/4	d3	-pi/2-Hyp
6	0	-pi/2	0	pi/4-Hr
7	0	pi/2	0	-Hp
8	-thigh	0	0	-Kp
9	-tibia	0	0	-Ap
10	0	-pi/2	0	-pi/2-Ar
11	0	-pi/2	-foot	-pi/2

Figure 20: mDH parameters transforming the coordinate frame of the left-foot to the bottom-camera

E Inverse Kinematic Matlab Code

```
% Objective is to keep ankle position after rotating hip yaw-pitch
% Inverse Kinematics mDH ankle to body with an iterative method

clear all;
clc;

% Dimensions of the Nao in mm (from Aldebaran documentation)
foot_height  =  45.11;
tibia        = 102.74;
thigh        = 100.00;
hip_offsetY  =  -49.79;
hip_offsetZ  =  84.79;
neck_offsetZ = 126.50;
trunk_length = hip_offsetZ + neck_offsetZ;

syms Hr Hp Hyp Kp x y z

% Forward Kinematic transform from body to ankle
% mDH        a             alpha   d             theta
DHparams = [ 0             pi/4    0             Hyp;
             0             pi/4    0             pi/2+Hp;
             0             pi/2    0             pi+Hr;
             thigh         -pi/2   0             -pi/2+Kp;
             0             pi/2    -tibia        0          ];

M = FKchain(1,5,DHparams); % symbolic forward chain
F = M*[0; 0; 0; 1];        % symbolic ankle position in body coords
V = [Hp Hr];
Js = jacobian(F,V);
Hyp = 0.0; Hp =  .1; Hr =  .2; Kp =  0.3; %example starting values

% Evaluate body coords for Hyp, Hp, Hr. Kp at start (target t)
t = subs(F);

% give turn angle
Hyp = deg2rad(45);

for i = 1:3;

    % Evaluate latest s
    s = subs(F);

    % Evaluate J at s
    J = subs(Js);

    % desired change in position for x,y,z
    e = t - s;

    % change in angles required to move towards target
    lambda = .4; % 0.4; % follow Northern Bites
    Jt = transpose(J);
    dA = Jt/(J*Jt+lambda^2*eye(4,4))*e;
    %
    % apply dA to test solution to see if it is the same as t
    Hp = Hp + dA(1)
    Hr = Hr + dA(2)

end
```

References

[1] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, IEEE Journal of Robotics and Automation, 2004.

[2] Car Chatfield. runswift 2012 inverse kinematics, the build system, and the python bridge. Technical report, Computer Science and Engineering, University of New South Wales, http://cgi.cse.unsw.edu.au/eport2012.pdf, 2012.

[3] J. J. Craig. *Introduction to Robotics*. Addison-Wesley, Reading, MA, 1989.

[4] Stefan Czarnetzki, Sören Kerner, and Oliver Urbann. Applying dynamic walking control for biped robot. In *RoboCup 2009: Robot Soccer World Cup XIII, Lecture Notes in Computer Science*, volume Volume 5949/201, pages 69–80. Springer Berlin Heidelberg, 2010.

[5] Felix Faber and Sven Behnke. Stochastic optimization of bipedal walking using gyro feedback and phase resetting. In *2007 7th IEEE-RAS International Conference on Humanoid Robots, November 29th - December 1st, Pittsburgh, PA, USA*, pages 203–209, 2007.

[6] Shuai Feng and Zengqi Sun. Biped robot walking using three-mass linear inverted pendulum model. In *ICIRA '08: Proceedings of the First International Conference on Intelligent Robotics and Applications*, pages 371–380, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] Tak Fai Yik Gordon Wyeth, Damian Kee. Evolving a locus based gait for a humanoid robot. In *International Conference on Robotics and Intelligent Systems*, 2003.

[8] C. Graf and T. Röfer. A closed-loop 3d-lipm gait for the robocup standard platform league humanoid. In C. Zhou, E. Pagello, S. Behnke, E. Menegatti, T. Röfer, and P. Stone, editors, *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the 2010 IEEE-RAS International Conference on Humanoid Robots*, 2010.

[9] Ian J. Harrington. Symptoms in the opposite or uninjured leg. In *Workplace Safety and Insurance Appeals Tribunal*, 505 University Ave., 7th Floor, 505 University Ave., 7th Floor, Toronto, ON M5G 2P2, 2005. Ontario Government, Ontario Workplace Tribunals Library.

[10] B. Hengst, D. Ibbotson, S.B. Pham, and C. P., Sammut. Omnidirectional locomotion for quadruped robots. In S. Tadokoro A. Birk, S. Coradeschi, editor, *RoboCup International Symposium RoboCup 2001: Robot Soccer World Cup V, Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence*, volume LNAI 2377, page 368. Springer 2002, 2001.

[11] Bernhard Hengst. Reinforcement learning of bipedal lateral behaviour and stability control with ankle-roll activation. In *16th International Conference on Climbing and*

Walking Robots, Sydney Australia (CLAWAR2013). World Scientific Publishing Company, 2013.

[12] Bernhard Hengst, Manuel Lange, and Brock White. Learning ankle-tilt and foot-placement control for flat-footed bipedal balancing and walking. *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

[13] Karl-Udo Jahn, Daniel Borkmann, Thomas Reinhardt, Rico Tilgner, Nils Rexin, and Stefan Seering. Nao-team HTWK team research report 2009. http://naoteam.imn.htwk-leipzig.de/documents/techReportHTWK.pdf, 2009.

[14] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, and Kensuke Harada Kazuhito Yokoi. Biped walking pattern generation by using preview control of zero-moment point. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1620–1626, 2003.

[15] Min Sub Kim and William Uther. Automatic gait optimisation for quadruped robots. In *In Australasian Conference on Robotics and Automation*, 2003.

[16] Manuel Lange. Developing a bipedal walk using a developing a bipedal walk using a cycloid ii. Technical report, Computer Science and Engineering, Univeristy of New South Wales, 2011.

[17] Roger Liu. Bipedal walk and goalie behaviour in robocup spl. Technical report, Computer Science and Engineering, University of New south Wales, 2013.

[18] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, David Claridge, Hung Nguyen, Jayen Ashar, Stuart Robinson, and Yanjin Zhu. rUNSWift team report 2010 robocup standard platform league. Technical report, School of Computer Science and Engineering, University of New South Wales, http://www.cse.unsw.edu.au/about-us/help-resources/for-students/student-projects/robocup/, 2010.

[19] Aldebaran Robotics. Developer documentation for the nao (red). http://robocup.aldebaran-robotics.com/docs/site_en/reddoc/index.html, July 2010.

[20] Thomas Röfer, Tim Laue, Judith Müller, Oliver Bösche, Armin Burchardt, Erik Damrose, Katharina Gillmann, Colin Graf, Thijs Je ry de Haas, Alexander Härtl, Andrik Rieskamp, André Schreck, Ingo Sieverdingbeck, and Jan-Hendrik Worch. B-human team report and code release 2009. http://www.b-human.de/index.php?s=publications, 2009.

[21] Oliver Ruepp. Recapitulation of dh convention. http://www6.in.tum.de/ ruepp/robotik0910/dh.pdf.

[22] Claude Sammut and Tak Fai Yik. Multistrategy learning for robot behaviours. In *Advances in Machine Learning I*, volume Volume 262/2010 of *Studies in Computational Intelligence*, pages 457–476. Springer Berlin / Heidelberg, 2010.

[23] Johannes Strom, George Slavov, and Eric Chown. Omnidirectional walking using zmp and preview control for the nao humanoid robot. In *RoboCup*, pages 378–389, 2009.

[24] Aaron James Soon Beng Tay. Walking nao omnidirectional bipedal locomotion. In *UNSW SPL team report.*, 2009.

[25] Rico Tilgner, Thomas Reinhardt, Daniel Borkmann, Tobias Kalbitz, Stefan Seering, Robert Fritzsche, Christoph Vitz, Sandra Unger, Samuel Eckermann, Hannah Müller, Manuel Bellersen, Martin Engel, and Michael Wünsch. Team research report 2011. Technical report, Leibzig University, 2011.

[26] Miomir Vukobratovic and Branislav Borovac. Zero-moment point - thirty five years of its life. *I. J. Humanoid Robotics*, 1(1):157–173, 2004.

[27] Eric R. Westervelt, Jessy W. Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*, volume 1. CRC Press, Boca Raton, 2007.

[28] Bock White. Humanoid omni-directional locomotion. Technical report, School of Computer Science and Engineering Engineering, University of New South Wales, http://cgi.cse.unsw.edu.au/ robocup/2011site/reports/White-Locomotion-Thesis.pdf, 2011.

[29] Tak Fai Yik. Locomotion of bipedal robots: Planning and learning to walk. In *PhD Thesis*. University of New South Wales, 2007.