A Heel to Toe Gait for Efficient Bipedal Walking

and the Striker Behaviour for rUNSWift's RoboCup SPL 2014

Luke Tsekouras, z3330983

October 28, 2014

THE UNIVERSITY OF NEW SOUTH WALES
The School of Computer Science and Engineering
Supervisor: Dr. Bernhard Hengst
Assessor: Dr. Maurice Pagnucco

Abstract

A heel to toe walk is designed for the Aldebaran NAO Humanoid Robot in order to test whether bipedal locomotion can be made more efficient by emulating the gait of a human being. A solution to overcoming inverse kinematics singularities is presented, and used to successfully describe a walk with full knee extension, heel strike and toe off motion. A closed-form ZMP trajectory function is derived from a simplified dynamics model, and used to stabilise the robot. The efficiency of the walk is compared with that of a typical bipedal walk, and is deemed to be less efficient due to the active use of more motors and higher degree of instability. We conclude that although a heel to toe walk is very efficient for a human, it is not necessarily well suited for the efficient locomotion of a servo-based biped.

Contents

Outline

This report is divided into two chapters, each of which can be read in isolation of the other. The first chapter contains the primary content of this thesis, and describes an investigation into creating a heel to toe gait for efficient bipedal walking. The second chapter contains a very rough description of the work done this year on the rUNSWift team's Striker behaviour.

Chapter 1

A Heel to Toe Gait for Efficient Bipedal Walking

1.1 Introduction

Bipedal locomotion in robotics is a topic of great interest to modern researchers. A large portion of this interest is believed to be motivated by the somewhat innate human desire to emulate nature. Although this might seem frivolous at first, there is most certainly much to be learned from the result of several billions of years of evolution.

Humans, as experienced bipedal creatures, are able to traverse a vast variety of complex terrains with little effort and impressive speed. We can hike up steep rocky surfaces, jump over gaps, and duck under obstacles. We're also very agile - we can quickly change direction to either avoid obstacles or navigate a complex path at speed, and can go from a flying sprint to a standstill in very little time, without destabilising. On top of all this, throughout all these scenarios we are able to move extremely efficiently, especially when compared to other quadrupedal mammals. Research has shown that one of the factors that gave hominids the upper hand in the evolutionary war was the fact that we could travel long distances with very little energy, essentially chasing our prey until they collapsed from exhaustion [1].

It is the lure of these benefits that has given bipedal locomotion such a great focus in research at the moment. For example, Boston Dynamics are doing some very impressive work in bipedal mobility with their Atlas humanoid robot, which has demonstrated stable walking across rocky and uneven terrains [2]. Honda have been working on their biped ASIMO for over a decade now, and recently demonstrated their latest model running and hopping [3]. Cornell University have done some excellent work in bipedal efficiency, creating robots which have only a few motors and rely largely on mechanical phenomena to move, instead of having a large motor at each joint [4].

In this report, we focus on the efficiency benefits that bipedal locomotion can offer. The motivator behind this is that a large proportion of bipeds make certain assumptions in order to simplify their motion, causing them to operate in an inefficient way. Typically the torso is kept at a constant height, the feet are kept parallel to the ground, and the knees are bent in order to avoid inverse kinematics problems. These assumptions were made in Atlas and ASIMO mentioned above, and were also made in the walk used by rUNSWift team for RoboCup 2014 and the years before, chiefly developed by Bernhard Hengst [5].

Efficient locomotion is important in robotics for several reasons. As it helps humanity to travel long distances, it would allow robots to also travel longer distances. Since battery technology still delivers relatively poor energy capacities, efficient locomotion would allow robots to operate for longer by making better use of their energy. Moving inefficiently also causes motors to heat up more quickly, which results in reduced motor performance and increases the risk of motor damage. This was a huge problem for rUNSWift and other RoboCup teams in previous years, where after long bouts in the competition the robots would overheat their motors, causing failure and damage.

As a result, in this thesis we set out to engineer a gait more efficient than a typical bipedal walk. Throughout this study we have selected the rUNSWift walk used in RoboCup to represent a "typical bipedal walk". Since the human gait is so efficient, work was focused on emulating the aspects of the gait which gave it its efficiency. The experiment was carried out on the Aldebaran NAO Humanoid Robot, the same robots used in the RoboCup robotic soccer competition. In this way we could easily make a comparison between the rUNSWift walk and the walk developed for this thesis, and make comments on the relative efficiency of each.

One of the most prominent differences between a typical robotic walk and a human walk is that a human keeps the knee joint of the support leg straight while under load. It was made a key goal in this project to keep this knee joint straight throughout the walk cycle, since a bent knee requires energy to remain bent under a load, whereas a straight knee does not. During RoboCup it was often the knee joints which overheated most quickly, which can be explained by the walk's bent knees. This was a key motivator in embarking upon this thesis.

In order to keep a straight knee throughout the gait, the normal assumptions of keeping the torso at a constant height above the ground and keeping feet parallel to the ground have to be broken. Unsurprisingly the human gait does not operate under these constraints either. The human gait sees the torso bobbing up and down throughout the walk, and sees the heel of the foot strike the ground and the toe of the foot leaving the ground last. It was therefore decided that in order to achieve a straight support knee during the gait, the human solution should be emulated and a heel to toe gait created.

Another key goal of the project was to ensure that joint movements were smooth, with minimal acceleration throughout. The more a motor is forced to accelerate, the more energy it consumes and the less efficient the movement becomes. The naive approach to animating the limbs and straightening the knee results in a very large velocity at the knee joint as an extension is made. This is because near the limits of a leg's extension, a small difference in the position of the ankle results in a large difference in the angle of the knee. This is known as a kinematics singularity, where there is a change in the expected number of instantaneous degrees of freedom [6]. It was a key challenge to overcome this problem in the design of the heel to toe walk.

This kinematics singularity problem was overcome largely through creative inverse kinematics formulations. Each leg in the NAO has three joints which operate in the coronal plane (hip, knee, ankle), requiring 3 variables to be fixed in order to fully define the values of those three joints. Typically this was the (x, y) position of the heel or toe, and the angle θ between the foot and the ground. But by fixing different variables in different parts of the walk and freeing others, it was possible to directly control problem joints and avoid the kinematics singularity problem entirely.

It has been shown that the stride length and frequency of a gait have strong implications for its efficiency in terms of energy consumed over time [7]. Although it was not a focus of this thesis to find the optimal stride length for the NAO robot, it was a goal to have the robot walk with a stride length and frequency that appeared to mimic that of a human, proportionally. This resulted

in a stride length longer than that of the rUNSWift walk, which by analogy with the human gait was hoped to yield a more efficient robotic gait.

In the final implementation of the heel to toe gait, stride length was freed up as a parameter to the walk, which meant it could work for any possible stride length. However in tuning and testing the walk both in the simulator and in the physical robot, high values for stride length were chosen in order to align with this goal.

One final key goal of the project was for the gait to be implemented and demonstrated on a physical robot, and to have its efficiency compared to that of the existing rUNSWift walk. This meant that methods for stabilising the robot needed to be developed, since the real world is much more noisy and unforgiving than a simulator.

Stabilisation was achieved in multiple ways. The double support phase of the gait was extended for as long as possible in order to grant the robot extra stability as a result of having multiple points of contact with the ground for a longer time. The centre of pressure or "zero moment point" (ZMP) of the support foot was ensured to be in the centre of the foot by planning a trajectory for the centre of mass in both the forwards and sideways directions. This resulted in a hip-sway which kept the robot balanced in the sideways direction, and a rocking motion forwards and backwards which helped prevent the robot from tripping. In order to account for unexpected disturbances in the forwards and backwards direction, the gyroscope was used to derive an adjustment angle which was added into the ankle and hip joints. This was a technique derived from the rUNSWift walk. In order to account for unexpected disturbances in the sideways directions, the foot Force Sensitive Resistors were used to detect when a swinging foot prematurely hit the ground, stopping the swing phase immediately to avoid kicking the toe into the ground forcefully. This technique was also derived from rUNSWift walk.

In order to test whether the heel to toe gait resulted in more efficient locomotion than the rUNSWift walk, both walks were run on a cold robot until a motor began to overheat. Temperatures of each motor in the robot were recorded over time, and the rate of increase of these temperatures gave an indicator of the amount of energy being consumed by each motor. In this way a comparison was made between the walks as to which was consuming energy faster in each joint. Additionally a supplementary experiment was performed in order to invesigate specifically whether standing with straight knee joints under load was more efficient than with bent knee joints.

The gait was developed in a simple simulator before being ported over to a physical robot. Within the simulator the gait displayed all of the desired properties outlined by the goals above. Each knee joint was kept straight whenever there was a load upon it, which was made possible by adopting a heel to toe gait. Despite the walk having three discrete phases per step, joints were moved smoothly throughout the gait. Large stride lengths were possible, but of course there were limits as to how long a stride could be.

The walk was successfully ported onto the NAO robot, and could sustain a walk without falling at a speed of approximately 100 mm/s. Speeds of up to 120 mm/s were possible with falls occurring only every so often, but speeds higher than this were not achieved in the time allotted for the project. The instabilities occurred due to inaccuracies in motor function, uneven surfaces, and a poor estimation of state from the NAO's hardware. These issues were made more difficult to deal with by the long stride length of the gait, and the fact that during the double support phase both feet were angled with respect to the ground for a long period of time.

Unfortunately it was determined that the existing rUNSWift walk was more efficient than the heel

to toe walk, as the average rate of increase in temperature of all joints was smaller in the rUNSWift walk than in the heel to toe walk. It is thought that the primary reason for this was that more motors were actively used in the heel to toe walk than the rUNSWift walk, in particular the joints in the hips and ankles were used to sway the robot left and right to achieve ZMP targets. On top of this, the techniques used for stabilisation broke a lot of the initial goals of the gait, resulting in inefficiencies.

It became apparent during this project that although a heel to toe gait ideal for a human, robots are at best crude approximations of the human morphology, and as such they will have optimal gaits which may look completely different to that of a human being. If we wish to emulate the human gait in order to reap its benefits, the first step is to construct a robot which is structured more similarly to a human. Despite not achieving the goal of efficiency, the project still proved to be an interesting and worthwhile experiment. This was validated by this project's performance in the RoboCup 2014 SPL Open Challenge, where it placed 2$^{\text{nd}}$ according to a league-wide vote. This project's work in inverse kinematics, used to architect a smooth yet dynamic heel to gait, as well as its closed-form solution to ZMP targetting, are hoped to be useful in other research, particularly in projects where a more human like robot is used.

1.2 Background

There has been a great deal of research into efficient bipedal locomotion, but it is interesting to note that some of the most successful pieces of research in the area of efficiency come from a subset of bipedal walkers called "passive-dynamic walkers". These bipeds are able to achieve locomotion without little to no active energy input or control systems. Once started on a gentle slope, they will quickly settle into a gait which has a striking resemblance with that of a human.

In 1990, McGeer analysed the dynamics of such walkers, and demonstrated that a passive walking effect was achievable for use in robotics [8]. His research was inspired by observing a bipedal toy, a type of passive-dynamic walker, which was able to traverse down a slope by swaying left and right, allowing each foot to swing forward on an axle, taking a step forwards each time it swayed. McGeer was one of the first to apply this concept to robotics, discussing the possibility of creating extremely efficient bipedal machines which required very little active energy to operate. Deciding that the left/right swaying motion was too chaotic in practice, a machine which operated only along a two dimensional axis was conceived, using small motors to lift each swing leg in turn. It worked similarly to how a person would use crutches. Although the machine was able to successfully move, it still required that it be placed on a gentle slope, and did not operate as efficiently as was hoped.

McGeer later went on to modify his design to include passive knee joints, which removed the need for active motor input [9]. The gait which resulted was remarkably human like, with a heel strike and toe-off motion, a swing phase which bent the knee to lift one foot in front of the other, and straight knees in the support feet at all times. McGeer suggests that this uncanny similarity to the human gait is a testament to the quality of nature's designs.

McGeer's work has inspired many others to pursue the possibility of creating efficient bipeds through passive-dynamic walking. In particular the work by Collins et. al. at Cornell University has proved to be very successful [4]. They have developed a wide array of passive-dynamic walkers, both powered and unpowered. For example, the Cornell walker is a powered walker designed to travel across flat surfaces in an energy efficient manner (Figure 1.1, leftmost image). It is closer to the morphology of a human than McGeer's work, using arms which were kinematically attached to the legs to keep the robot dynamically balanced on each foot. The robot was actuated at the ankles using a simple mechanism which extended one foot as the other came into contact with the ground. The robot's actuators consumed roughly the same amount of energy as what the equivalent passive dynamic walker gained in gravitational potential energy, and was reported to be comparable in efficiency to that of a walking human being. The Cornell walker was also determined to be 10 times more efficient than ASIMO, chosen to be representative of typical servo-based bipedal robots.

Figure 1.1: *An assortment of passive-dynamic walkers developed at Cornell. The first robot is powered at the ankles, the second is powered at the hips, and the last is entirely passive.*

It would appear that passive dynamic walkers have successfully taken advantage of the efficiency benefits in a human-like gait, indicating that emulating this gait is a fruitful route in the pursuit of

efficiency. What these passive dynamic walkers gain in efficiency however, they lose in mobility. They have an extremely restricted range of movement when compared to typical servo-based robots, which make up the vast majority of bipeds in research. Where the passive-dynamic walkers have much of their gait constrained through mechanics, servo-based robots allow researchers to control the angle of every joint in their body with a high degree of accuracy.

Figure 1.2: *The Aldebaran NAO Humanoid Robot*

The Aldebaran NAO, used in the RoboCup SPL soccer league, falls into this category of servo-based robots. RoboCup is a competition designed to motivate research in robotics, and the NAO, although not originally designed as a research tool, has become a key component of many researcher's work around the world. In particular the SPL league has motivated work in bipedal locomotion, including our own team's work on the rUNSWift walk, the widely used B-Human walk, and of course this thesis.

The rUNSWift walk was designed in order to move as quickly as possible, display agility in changing direction, and give great control of omnidirectional movement [5]. The walk is architected with a single phase per step, which is always a swing phase, meaning that the robot is technically in a single support phase for the entire duration of the walk. The walk is stabilised in the left/right directions by developing a natural rhythm, controlled by switching feet whenever the zero moment point moves to the other side of the robot. Stabilisation in the forward/backward directions is achieved by adjusting the ankle pitch in each foot proportionally with the gyroscope's measurement of angular velocity. The torso is kept at a constant height, the knees are kept bend to avoid a kinematics singularity, the feet are kept parallel to the ground, and the robot's forward motion is linear. The rUNSWift walk is characterised throughout this report as representative of a typical bipedal walk for servo-based robots, and is later used for comparison with the walk developed as a part of this project. The success of the stabilisation techniques used in this walk triggered the inclusion of similar/identical techniques in the construction of this project.

The B-Human walk had similar aims to that of the rUNSWift walk, but had both similarities and differences in their approach. Their walk too maintained a constant torso height, bent knees and flat feet for similar reasons to ours. Additionally however, keeping the torso at a constant height caused the robot to match their dynamics model more closely - the 3D linear inverted pendulum [10]. This model was made popular by Kajita et. al. when they published a paper on the simplified dynamics model used for their bipedal research [11]. The motion of the centre of mass (CoM) of the robot was constrained to an arbitrary plane above the ground, and the dynamics equations formulated allowed one to plan a trajectory for the CoM which placed the zero moment point of the system at a desired position. The B-Human walk uses an iterative approach to determining the trajectory of the CoM for each step. This is different from the approach taken in this thesis, where we derive a closed-form trajectory calculation. Similarly to the rUNSWift walk, the B-Human walk has no double support phase, and also uses the timing of the foot switch to control the CoM of the robot.

Controlling the zero moment point of a biped, abbreviated to "ZMP", is a very common way of ensuring dynamic stability in a walking robot. The ZMP is the point along the support foot which

experiences a turning moment of 0 - the sum of the torque applied at this point from all forces in the system add to nothing. This is mathematically equivalent to the centre of pressure of the support foot, where the highest density of reaction forces with the ground is found.

Kajita et. al. use these ZMP calculations in order to maintain stability in their bipeds at AIST. One simple model they use to calculate the ZMP of a robot is coined the cart-table model [12]. This sees an actuated cart atop a massless table with a single leg. The leg represents the foot of the robot, and the moving cart represents the CoM of the robot. Interestingly, the cart-table model is the dual of the linear inverted pendulum model. As the cart accelerates in one direction, the table below it receives a reaction force in the the opposite direction. So even if the cart is outside the bounds of the table's foot, it can accelerate such that the table receives a counteracting force that keeps it upright. The required acceleration to keep the table upright can be calculated by ensuring that the ZMP lies within the leg of the table[1]. This model was used to construct a servo controller which dynamically generated a trajectory for the CoM, ensuring that the ZMP appeared within the bounds of the support foot at all times. The controller was interesting in that, unlike most servos, it used future information about where the ZMP would appear in order to more quickly and accurately achieve its ZMP targets.

The assumptions made by these simple models are deemed acceptable since most robotic walks maintain a constant torso height, helping to keep the CoM physically constrained the plane it is assumed to be constrained to. This project however uses the cart-table model as a basis for its dynamics while neglecting to maintain a constant torso height, and we later see whether this was enough to break the simplified model's validity.

Goddard et. al. investigated an alternative to ZMP for creating dynamic balance which involved controlling the reaction forces in the foot as the heel was lifted just off the ground, and as the toe was lifted off the ground shortly afterwards [13]. The proposed system claimed to account for terrain of unknown shape, and a simulation was used to demonstrate that the mechanism had merit. Despite the almost orthogonal aims of the project, there were similarities in how the gait was decomposed. A research paper in biomechanics by Czerniecki was referenced [14], which identified three phases of the human gait: foot contact to foot flat, foot flat to heel off, and heel off to toe off. These phases were determined to have distinct kinematic and kinetic properties, which is why the distinction was made between the phases. Goddard therefore structured his solution around these three phases, since each had different properties to consider in the gait's dynamics. It is interesting to note that these three phases were also used in this thesis, for reasons also to do with their distinct kinematic and kinetic properties.

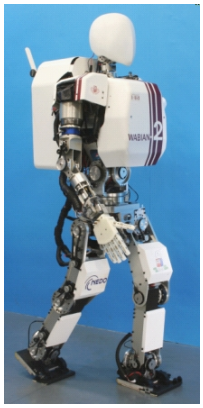


Figure 1.3: *The WABIAN-2R by Ogura et. al., performing a heel to toe walk*

Ogura et. al. aimed to create a human-like heel to toe gait which included a heel-contact and toe-off motion, similar to the aim of this thesis. They aimed to have the robot walk with a straight knee, strike the ground with its heel, and leave the ground with its toe. However, efficiency was not a goal of their project, and instead the emulation of human movement was key. The robot was modified to include more human-like effectors, most notably the heel and toe mechanisms

[1]For a diagram of the cart-table model, see Section 1.3.7.1, where the derivation of the equations of motion are presented

in the foot of the robot (see Figure 1.3). The heel to toe gait was defined using cubic splines to describe trajectories, and target positions for the ZMP to balance the robot. The ZMP was not calculated using a simplified model however, and instead the Newton-Euler method for calculating the dynamics of a rigid body was used to more accurately determine the ZMP. The researchers encountered a kinematics singularity in straightening the knee, whereby a degree of freedom was suddenly lost and inverse kinematics equations broke down. Their solution to this was to control the knee angle directly, making it a parameter to inverse kinematics as opposed to a result. In order to make up for this now permanently lost degree of freedom in the knee, extra degrees of freedom in the hips of the robot were used instead. Additionally, the height of the torso was not predetermined, but was allowed to be controlled by other constraints on the walk. It is interesting to note the similarities in our approaches to the inverse kinematics problems presented by a heel to toe walk. We both control the knee angle to avoid the singularity, and both allow the torso height to be a free variable, however the projects differ in that the hip joints in the NAO were not used to overcome a lost degree of freedom. Other variables within the leg were freed instead, depending on the phase of the walk.

Endo et. al. also attempted to create a human-like gait, but tried to apply more naturally inspired techniques to achieve the desired motion, in the way of neural oscillators [15]. Things like inverse kinematics and ZMP targets were described as unnatural and mathematically inconvenient, pointing towards kinematic singularities as an example. Instead empirically defined oscillators were constructed to describe the motions of a human gait and to have the gait respond to changing conditions. The project was largely successful in that the robot could walk over varying kinds of surfaces, recovering from instabilities successfully. However, the biped was required to be stabilised by a large arm attached to a pivot on the floor, as the project only concerned itself with dynamics in the Sagittal plane. One of the secondary aims of the project was to optimise the movement's efficiency by way of tweaking a single parameter, described as the neural activity. Although the maximally efficient gait along this parameter was determined, it was concluded that the gait was not as efficient as other methods of locomotion, pointing to passive-dynamic walkers as an example. The report hypothesises that humans gain a large portion of their efficiency from their ability to walk passively when they can, allowing momentum and gravity to do work for them.

Channon et. al. attempted to achieve a similar feat in optimising a walking gait for the sake of efficiency, but constructed a much more detailed and sophisticated model in order to do this [16]. The motion of a walk was generalised as a hip trajectory and a foot trajectory, defined as 3rd order polynomials, and the constants over these functions were optimised for distance travelled per unit of energy expended. Energy expenditure was calculated by considering each servo motor's heat losses, work done in reaching target joint angles, and work done in maintaining the correct angle on impact. The key factor affecting the energy use of the motor was determined to be the torque required to achieve the desired motion. The study concluded that stride length was the most important factor in creating an energy efficient walk, with longer stride lengths being more efficient in general than the shorter ones, except when moving slowly. The degree to which the knee was straight was not considered by the author to be an important factor in the gait's efficiency, but this may have been a result of how the gaits were parametrised, making the knee singularity a large issue and therefore an inefficient option. What this paper does highlight however is that in order to improve servo efficiency, the torque required on each motor needs to be minimised. Based on dynamics equations derived in this report, in general the torque required at each motor is proportional to the acceleration of the target angle at that motor's joint, which is important to this project.

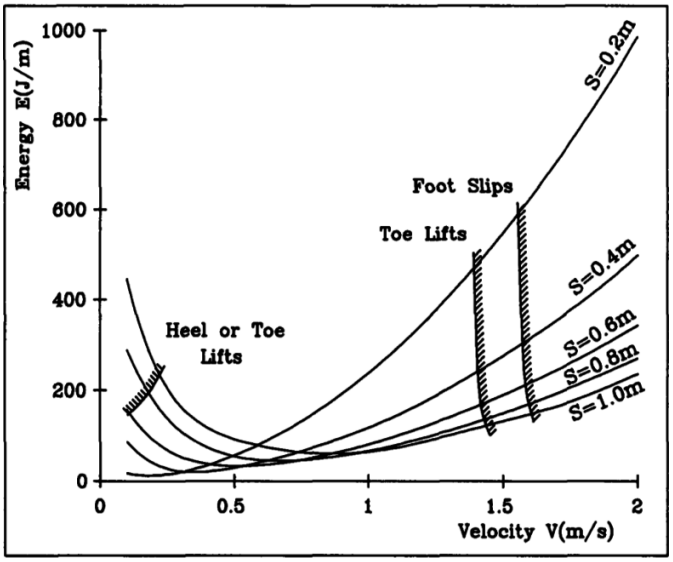


Figure 1.4: *A graph from Channon's report displaying the effect of stride length on the efficiency of the walk for various velocities.*

Efficient bipedal motion appears to be most effectively achieved using some kind of passive-dynamic walker, by a very significant degree. However, passive-dynamic walkers are by no means agile robots. They can only walk forwards, and always at the same speed, since their kinematics are defined through physical attachments. The only accessible alternatives to passive-dynamic walkers are servo-based robots, which are very easy to control, but extremely inefficient. Although there is the potential for a new kind of motor to be produced which allows the advantages of both classes of biped to be realised simultaneously, at present these motors are not accessible. The best we can do at this point is to take servo-based robots and try to find ways to use them more efficiently.

This project attempts to do just that. Since the human gait is known to be a very energy efficient gait, and since emulations of it have been shown to give excellent efficiency results, we hypothesise that emulating the human gait in a servo-based robot may allow the robot to move more efficiently.

1.3 Methodology

A simulator was first built in order to make early development of the walk engine more swift. Once the walk was at a minimally functional stage, it was translated into a form which could be executed on the NAO robot. From this stage forward work alternated between the code bases, as some new features were most easily developed in the physical world, whereas as others were best simulated.

The heel to toe gait is decomposed here into three phases: Rock, Lift and Swing. Each of these phases has well defined start and end positions, referred to here as "anchors", and the motions in between were interpolated in such a way that the transition of the joint angles remained smooth while still achieving their goals. Several different interpolation equations and techniques were created in order to achieve the appropriate mixture of smoothness and control.

The walk engine is a time-based animation as opposed to a step-based animation. The "state" of the walk engine, which can be thought of as the input data to the mathematics of the walk, is well defined, such that it is very easy to see exactly what information is used to generate a particular frame of the walk animation. It also allows any phase of the walk to be interrupted and a new phase to be begun without issue - every phase of the walk has a clear way to finish at its anchor given any reasonable starting point.

The joint angles were determined separately in each leg by fixing three variables, (x, y, θ), which in turn fully defined the three joint angles in the leg through inverse kinematics. However, throughout the walk, depending on the phase, a different set of three variables were chosen to control the position of the leg in order to more easily define and interpolate between gait features. In order to use these different sets of variables, transformations into the regular set of variables were developed.

In order to stabilise the robot, a zero moment point targeting strategy was developed. By simplifying the dynamics of the robot to be a rolling mass upon a weightless table, equations of motion were derived such that we could plan for the centre of the mass of the robot to be at a particular point at a particular time, following a trajectory which ensured that the ZMP lay within the support foot the entire way. This was applied to both the forward/backward direction and the left/right direction of the robot, generating a hip sway and a small hesitation over the support foot to maintain balance.

In order to take advantage of sensor feedback to stabilize the robot, some empirically proven methods were incorporated into the walk. The gyroscope measurements on angular velocity in the forward/backward direction were incorporated linearly into an angle adjustment, which was in turn incorporated into the ankle and hip joints to help counteract angular rotations in the torso. The four foot sensors were summed together to give an estimate on overall pressure, and if this number exceeded a threshold it was deemed that the foot was in contact with the ground. When this occurs, the foot is stopped from moving further forwards, the stride length is adjusted as such, and the next phase of the walk continues from the current position of the robot.

1.3.1 Convention

In order to make explanations more clear, a quick summary on conventions used in measurements and models is included here.

Almost always, a 2D cross-section of the robot was made when performing calculations or constructing a model of the robot. There are three planes that are commonly referred to when describing

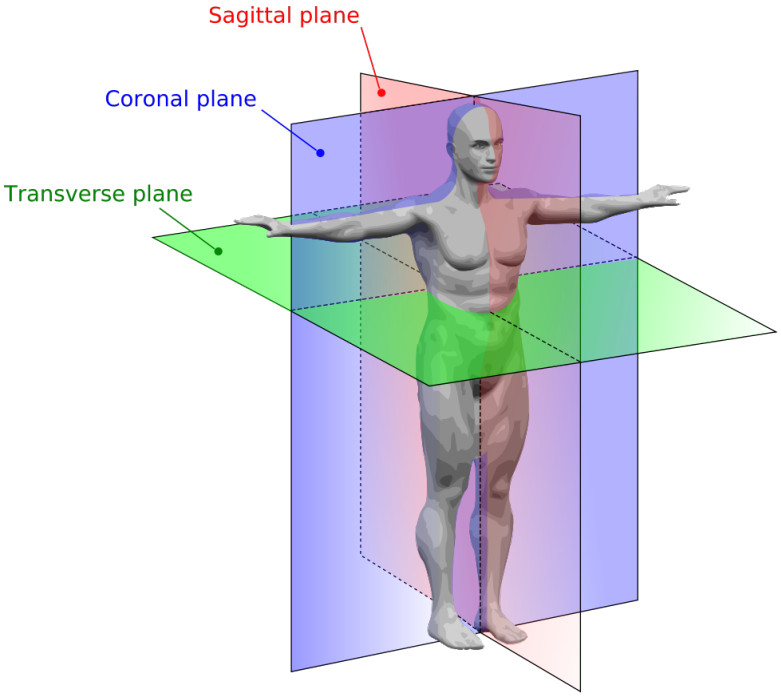


Figure 1.5: *A diagram showing the three planes of a human body - the Sagittal plane, Coronal plane, and Transverse plane.*

cross-sections of a humanoid, and these terms will be used throughout the report. These are the Sagittal plane, the Coronal plane, and the Transverse plane. See Figure 1.5 for a visualisation.

When expressing co-ordinates in a 3D setting, there is often disagreement on which direction the axes should be oriented. Throughout this report we follow the convention adopted by Aldebaran in creating the NAO robots, having the x-axis face forwards, the y-axis face to the left, and the z-axis face upwards. Figure 1.6 displays this visually. The origin is taken as the halfway point between the centre of the two hip joints.

The angles of the hip, knee and ankle joints of a particular leg are assigned the letters α, β and γ respectively. These angles describe how far a joint has bent away from its natural position, which is the upright standing position. A positive angle corresponds to a clockwise rotation if we see the z-axis facing upwards and the x-axis facing to the right. The angle of a foot to the ground is assigned the letter θ. A positive angle corresponds to a foot's toe being higher than its heel, and a negative angle corresponds to the opposite. The ankle, toe and heel have co-ordinates defined along the Sagittal plane, using the hip joint as the origin. These are given the symbols (x_a, z_a), (x_t, z_t) and (x_h, z_h) respectively. See Figure 1.7 for a visual explanation of the above.

1.3.2 The Simulator

The walk was first developed in a simulator before being ported onto the NAO robots. The simulator was written in JavaScript/HTML5, as it was the technology stack most familiar to the author. As a result the simulator is accessible over the Internet, conveniently available at http://www.cse.unsw.edu.au/~lukegt/heel-to-toe/ for your viewing pleasure.

The key reason behind using a simulator at first instead of developing straight on the robot was to speed up the early phases of the walk's development. Trying to write a motion engine from scratch on a robot is difficult because when things initially do not work correctly, there is a very

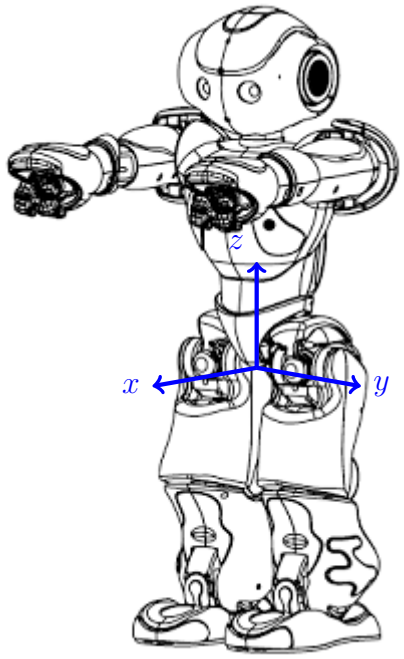


Figure 1.6: *The orientation of the axes relative to the NAO robot.*

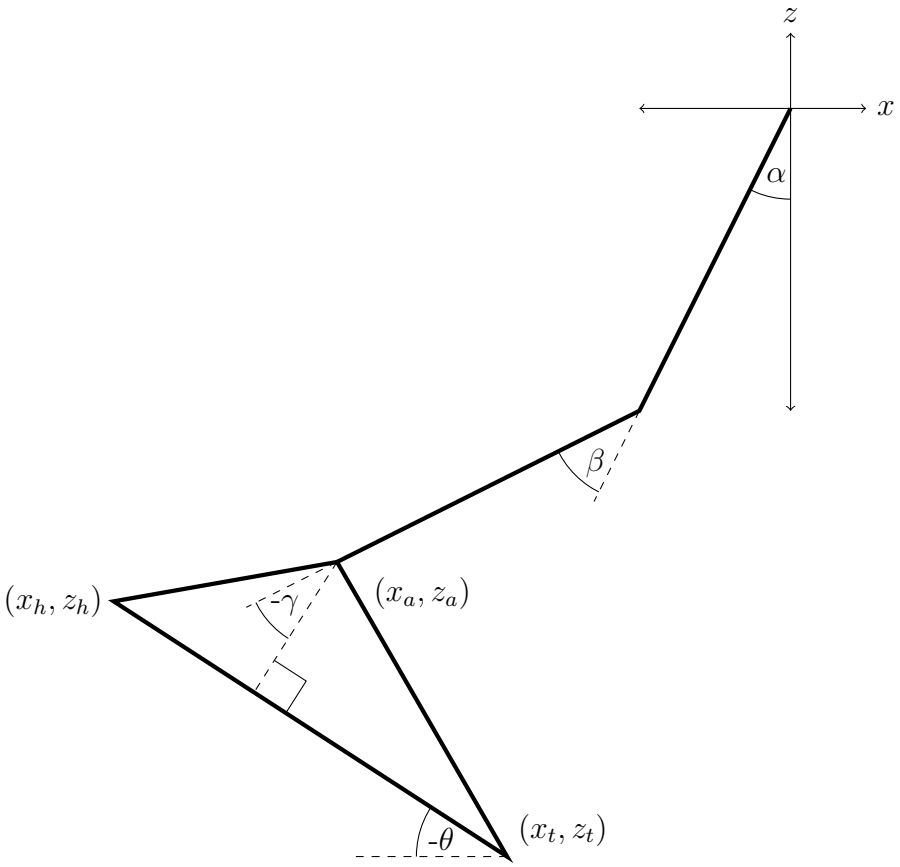


Figure 1.7: *A figure showing where the joint angles α, β, γ and the angle to the ground θ are found, as well as their sign. We also see the points (x_a, z_a), (x_t, z_t) and (x_h, z_h) corresponding to the position of the ankle, toe and heel respectively.*

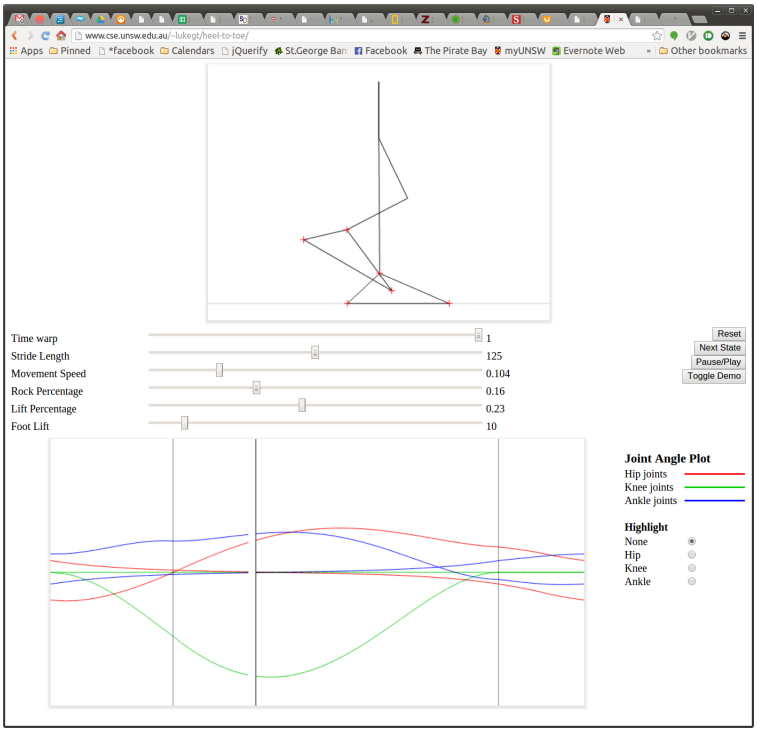


Figure 1.8: *The 2D simulator in action. You can see the walking animation at the top of the screen, the parameter sliders just below that, and the joint plot at the bottom of the screen.*

large amount of factors which may add up to this failure. In a simulator there are far fewer factors, and these factors are far more transparent and easily debugged. This does not only cover code bugs, but also mathematical errors which may arise. There was more than one occasion where equations were slightly off, causing very minor inaccuracies in the placement of feet, and these small inaccuracies would only have been detectable in the simulator.

The simulator is a fairly crude approximation of the actual NAO Humanoid Robot, but was more than enough to satisfy its purpose. It is a two-dimensional stick model of the torso and legs, with accurate measurements of the lengths of each limb and extremity taken from the NAO official documentation [17]. It is a 2D cross-section of the Nao along the Sagittal plane (forward/back, up/down), and so only includes 3 angles for each leg - the angle at the hip, the knee, and the ankle along this plane.

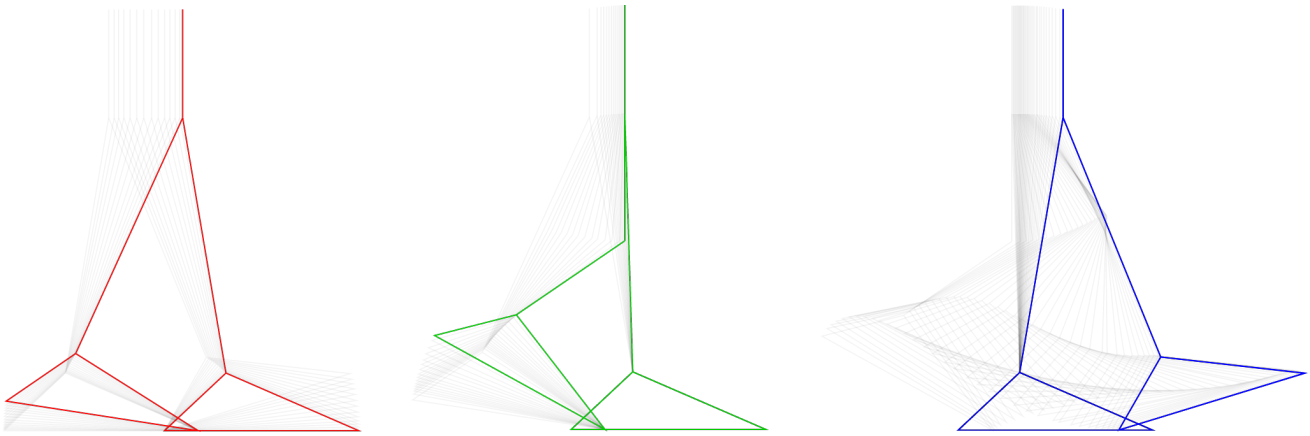
There is no physics engine, but instead the model is positioned with the torso upright and with the lowest extremity of the model just touching the ground. The model does not technically move forwards as it animates, but simply slides its feet along the ground.

A screen shot of the simulator can be seen in Figure 1.8. A visualisation of the walking robot is visible on screen, along with sliders controlling all the various parameters that the walk can take in (to be discussed later). The simulator has an additional slider which allows time to be sped up or slowed down, so that small nuances in the walk may be closely observed.

There is also a small plot of each of the joint angles in each leg. This plot made it very easy to determine visually whether the joint angles were being animated smoothly or not, since it was often deceiving to judge this purely from the walking animation.

1.3.3 Gait Phases

The heel to toe gait is divided into three phases - the Rock phase, the Lift phase, and the Swing phase. Each phase uses a different variation of the inverse kinematics equations to calculate the poses on the boundaries of each phase, as well as how to interpolate between these boundary poses. The specific details on how these are calculated will be explained in later sections, but a general overview is given here to contextualise upcoming content. See Figure 1.9 for a visual depiction of the three phases.



(a) *'rock' phase* **(b)** *'lift' phase* **(c)** *'swing' phase*

Figure 1.9: *The three phases of the walk. The three images shown depict a phase each, with the final position of each phase clearly visible and the motion throughout the phase faded behind.*

1.3.3.1 The Rock Phase

is where the weight of the robot shifts from one foot to another. The front foot, which has just had its heel struck into the ground, rotates towards the ground until it is flat against it. The rear foot, which begins flat against the ground, lifts its heel up while keeping its toe on the ground. While this happens, the centre of mass maintains a forward trajectory such that the ZMP switches from the rear foot to the front foot. During this process, both knees are kept straight, since this is a double support phase.

1.3.3.2 The Lift Phase

was introduced in order to lengthen the amount of time where both feet are in contact with the ground. However, the ZMP is positioned in the front foot, and the rear foot is just there to dampen any possible instabilities. The knee of the rear foot is taken from a straight position to a bent position, and as a result the heel of the rear foot will be brought further upwards, pointing the toe further towards the ground. The front foot remains planted flat on the ground, and the centre of mass moves forwards in order to keep the ZMP on this foot. The rear foot's knee joint exists the lift phase with an angular velocity, which makes the transition into the swing phase smooth. The exact angular velocity used is a manually tuned parameter.

1.3.3.3 The Swing Phase

is where the rear foot is lifted into the air, swung forwards, and then planted into the ground ahead of the other foot. The support foot is kept flat against the ground, and its knee is kept straight throughout the whole phase. The swing foot's knee joint transitions from a bend into being straightened, ending with its heel planted into the ground. Since both feet at the end of the phase need to be at the same vertical distance from the torso, the angle of the heel when planted is adjusted to facilitate this. It is the only phase in the walk where one foot is in contact with the ground, also referred to as a single support phase. The centre of mass moves forward in order to keep the ZMP on the support foot.

1.3.4 Gait State

The "state" of the gait is a data structure which contains all the necessary data (apart from the current time) to generate the current pose of the robot. The reason for explaining this data structure is so that the reader can understand what parametrises the walk, and because its values will be referred to in later sections as they are required.

- `timestamp`: The time in milliseconds since some arbitrary moment in the past (e.g. the epoch). It refers to the time at which information in the current state was last updated.
- `initial`: A special flag which tells us whether the walk is in a special "initial" state. If this flag is set, the engine transitions out of the current state as soon as it is able, in order to begin the walk.
- `phase`: Which phase of the walk we are currently in. This can be one of `"rock"`, `"lift"` or `"swing"`.
- `side`: Since the walk is symmetric in both the left and right feet, the three phases only describe half a complete walk cycle (a single stride). This variable describes which foot is the support foot during the current phase. For the rock phase, it refers to the front foot, which becomes the support foot by the end of the phase.
- `parameters`: a collection of properties which describe a number of parameters to the walk. Some are set by the user, and some are calculated from other parameters but stored here for convenience.
  - `stride_length`: The distance in mm between the ankle joints of the robot during the walk when both feet are flat on the ground.
  - `movement_speed`: The average rate at which the centre of mass moves forward over time, measured in ms^{-1}
  - `lift_percentage`: The percentage of time of a single stride spent in the lift phase. This affects the style of the gait, and in particular allows control over how long two feet are kept on the ground for stabilisation.
  - `rock_percentage`: The percentage of time of a single stride spent in the rock phase. This affects the style of the gait, and in particular allows control over how much time is spent transferring weight from one foot to another.
  - `foot_lift`: How much the swing foot is additionally elevated above the ground during its motion.

  - `swing_percentage` (calculated): The percentage of time of a half walk cycle spent in the swing phase. This is calculated as $1 - rock\_percentage - lift\_percentage$.
  - `stride_duration` (calculated): The duration of a single stride. This is calculated as $stride\_length/movement\_speed$.
  - `lift_duration`, `swing_duration`, `rock_duration` (calculated): The amount of time spent in the lift, swing and rock phases respectively. These are calculated by multiplying their respective percentage parameters with `stride_duration`.
  - `nod_gyro_ratio`: The manually determined constant which is multiplied with the GyroY measurement to give an angle adjustment, used for stability.

- `beginning`: A complex structure of properties which contain information about the state of the walk at the beginning of the current phase. Some information is redundant, but is stored here anyway so that calculations need not be repeated each frame.
  - `timestamp`: The time at which the current phase began.
  - `position`: A collection of information to describe where the key points of each foot were located along the Sagittal plane, as well as the angle of each foot relative to the ground. The key points stored are the ankle, the toe and the heel of each foot. These values are used in interpolation calculations for the current phase.
  - `leg_angles`: All of the joint angles in each leg of the robot. This includes the hip, knee and ankle joint angles along the Sagittal plane. These values are used in phases where particular joint angles are controlled and interpolated directly, as opposed to the foot's position and angle.
  - `joint_velocities`: An approximation of the velocities of each of the joints in each leg, measured in radians per millisecond. These are updated every frame that the walk engine is run, which occurs approximately 100 times per second. When the state is updated, each velocity is approximated as the difference between the new `leg_angles` and the existing `leg_angles` is divided by the difference in `timestamp`. These values are used to ensure that sudden changes in joint velocities are avoided.
  - `sway`: A 2D point along the Transverse plane describing how far in the X and Y directions that the centre of mass has been shifted for stability purposes.

- `leg_angles`, `join_velocities`: The same as the properties within `beginning`, but updated every frame as opposed to only on phase transition.
- `filtered_sensors`: Smoothed and calibrated versions of sensor data, taken by a combination of rolling averages. Specifically only the gyroscope around the Y axis was filtered for the walk, but in the implementation all other sensors had the ability to be filtered in the same way.

1.3.5 Inverse Kinematics

Kinematics is the process of taking a mathematical description of a robot's pose and calculating the position and orientation of each limb given that description. Generally the robot is described as a series of limbs and joints, with lengths and angles all defined. Inverse kinematics is, as the name suggests, the process of inverting kinematics - given the position and orientation of a particular limb, come up with a pose for the robot which achieved that position.

Some inverse kinematics equations can give multiple solutions to a particular request, but it is often desirable to structure your equations in such a way that only one answer is possible. In order

to achieve this in the NAO, we focus only on the legs of the robot, and only consider joint angles which lie along the Sagittal plane. Since each leg has 3 joints, 3 variables need to be fixed in order to give a unique inverse kinematics solution. In general, the three variables used were (x_h, z_h, θ) or (x_t, z_t, θ), representing the position of the heel or toe of the robot in the Sagittal plane, and the angle of the foot relative to the ground (assuming the torso is vertical).

However, even though these three variables allowed any required foot position to be easily expressed statically, they posed problems for when they changed with time. In particular, when the knee joint was almost straight, a small change in these variables can cause a very large change in the knee joint's angle. In fact, if the ankle is moving directly away from the hip in a linear fashion, the knee joint quickly approaches an infinite rate of change as the ankle approaches its extremity. This is called a kinematics singularity, and is a difficult problem to tackle. On the physical robot a singularity such as this would cause motors to attempt to move very quickly, which causes undesirable accelerations and jerkiness.

It was identified that the underlying issue behind the singularity is in the variables selected as inputs to the inverse kinematics equations, referred to as the "control variables" from here forward. By selecting a different set of variables and changing these over time instead, the singularity in the area of interest can be avoided entirely. All that must be done is to find a transform from the new set of variables into the original set of three.

Ideally we would want to pick a set of variables which generates no singularities. We can achieve this by selecting each joint angle itself as a variable, but this is somewhat inconvenient and essentially means circumventing inverse kinematics altogether. It might not be possible to find other variables which have no singularities, but it is definitely possible to find variables which remove a specific singularity, even though they generate others. This allows us to pick and choose variable selections throughout the motion as necessary, avoiding known singularities in each phase of the motion.

The knee singularity mentioned above caused problems in the `lift` phase, where the knee joint went from straight to bent, and also in the `swing` phase, where the knee went from bent to straight.

To solve the `lift` phase singularity, the knee joint itself (β) became a control variable, in addition to the position of the toe (x_t, z_t). It is clear that if the knee joint β becomes directly controlled during interpolation, it will not unexpectedly move at a high velocity.

To solve the `swing` phase singularity, despite what was mentioned above, all angles were controlled directly, meaning the control variables were (α, β, γ). This was acceptable in this situation because the swing foot's exact position was of little consequence, making inverse kinematics unnecessary during the swinging motion. However, care needed to be taken to ensure that the swing foot left and returned to the ground only at the intended moments. To achieve this, a high degree polynomial was used to interpolate each of the joint angles individually, ensuring all angles passed through a safe waypoint along the way, urging the limb away from the ground during its motion. This will be explained more in section 1.3.6.

A slightly different problem with a similar solution was also present within the walk's motion. One of the key goals of the walk was that the support foot's knee joint was to be completely straight throughout. As the robot moves forward, the support foot must move backwards relative to the torso, and as it does this, the support foot's knee must be completely straight. The x-coordinate of the foot is easily placed once we calculate the target position for the centre of mass, to be discussed later. But as the x-coordinate changes over time, the z-coordinate needs to change such that the knee remains straight. The solution was to define the motion using the variables (x_h, θ, β),

where both θ and β were set to 0, then find a way to transform these co-ordinates into the regular (x_h, z_h, θ).

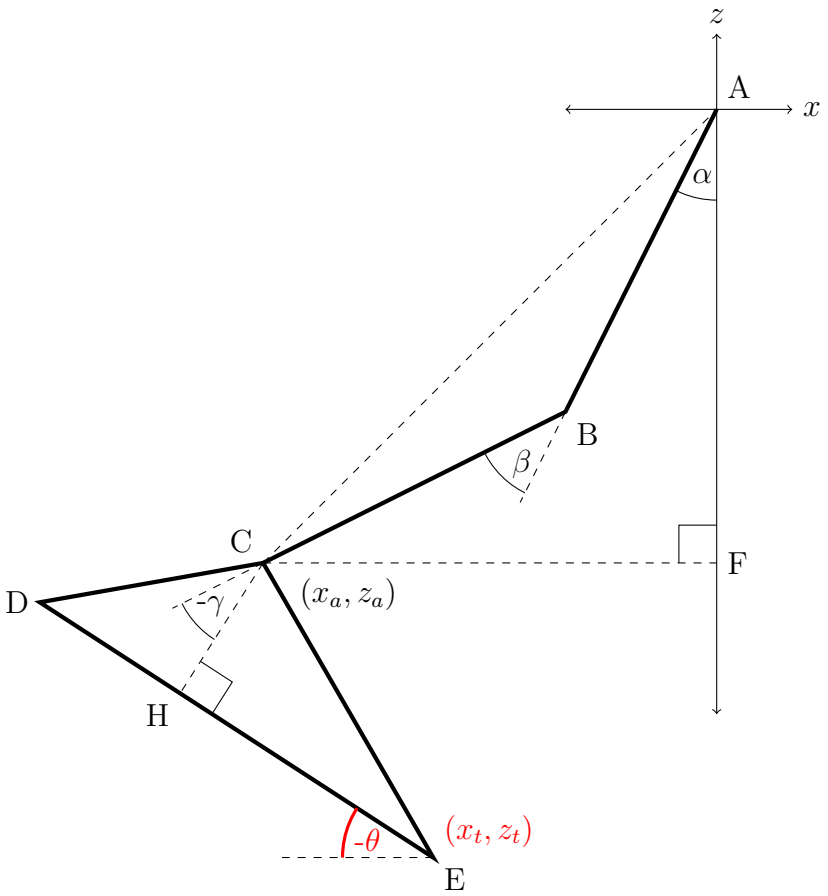
This process of transforming a convenient set of variables back down to either (x_h, z_h, θ) or (x_t, z_t, θ) was used extensively to enforce constraints on the walk's motion and to provide smooth kinematics during phases. The various formulations will be described in detail in the following sections, and be given context as to where in the walk engine they are used.

There was a very small piece of inverse kinematics formulated for the robot's motions along the y-axis. Essentially, we first ignore any y-axis movement, and calculate the joint angles in both legs under the assumption that the robot is upright along the Coronal plane. Afterwards, we adjust the angles in the hip and ankles which rotate around the x-axis to shift the robot back or forth along the y-axis. This is done in such a way that the torso remains upright and the legs remain parallel, so as not to ruin any joint angle selections from previous calculations.

In the diagrams that follow, variables marked in red indicate the control variables. All thick lines are of known length, and correspond to limbs on the NAO Humanoid Robot.

1.3.5.1 Toe Position and Foot Angle

This is one of the core inverse kinematics equations that other control variable combinations are transformed into. It is used primarily during the `rock` phase for the rear foot, and during the `lift` phase for the rear foot also. Our goal here is to take the control variables (x_t, z_t, θ) and calculate the joint angles (α, β, γ) from them.



We first translate the toe coordinates (x_t, z_t) into the ankle coordinates (x_a, y_a). This is possible since we know the shape and orientation of the foot. To achieve this we find the vector $\vec{EC}$ by rotating the vector formed by the hypotenuse of the right angled triangle $\triangle CEH$ by θ, then add this vector to the toe coordinates.

$$\vec{EC} = \begin{pmatrix} cos\ (-\theta) & -sin\ (-\theta) \\ sin\ (-\theta) & cos\ (-\theta) \end{pmatrix} \begin{pmatrix} -EH \\ CH \end{pmatrix} \qquad \text{(rotate } \triangle CEH\text{)} \qquad (1.1)$$

$$\begin{pmatrix} x_a \\ z_a \end{pmatrix} = \begin{pmatrix} x_t \\ z_t \end{pmatrix} + \vec{EC} \qquad \text{(simple vector math)} \qquad (1.2)$$

From here we can find β by applying the cosine rule in $\triangle ABC$. However we need to use Pythagoras' theorem to find the length AC beforehand.

$$AC^2 = x_a^2 + z_a^2 \qquad \text{(Pythagoras' theorem in } \triangle ACF\text{)} \qquad (1.3)$$

$$\angle ABC = cos^{-1}\left(\frac{AB^2 + BC^2 - AC^2}{2(AB)(BC)}\right) \qquad \text{(Cosine rule in } \triangle ABC\text{)} \qquad (1.4)$$

$$\beta = \pi - \angle ABC \qquad \text{(supplementary angles on the line } AG\text{)} \qquad (1.5)$$

We can see that α is simply the difference between $\angle CAF$ and $\angle BAC$, and these can be found by the inverse tangent function and the sine rule respectively.

$$\angle CAF = tan^{-1}\left(\frac{x_a}{z_a}\right) \qquad \text{(simple trigonometry in } \triangle ACF\text{)} \qquad (1.6)$$

$$\angle BAC = sin^{-1}\left(\frac{BCsin(\angle ABC)}{AC}\right) \qquad \text{(sine rule in } \triangle ACF\text{)} \qquad (1.7)$$

$$\alpha = \angle CAF - \angle BAC \qquad \text{(adjacent angles)} \qquad (1.8)$$

Now, observe the line $ABCHE$. Imagine we stand at at A, facing the positive x-axis. As we walk along $ABCHE$, taking into account that clockwise motions are negative, we will rotate by $-\frac{\pi}{2}$ to face downwards, then $-\alpha$, $-\beta$, $-\gamma$, $\frac{\pi}{2}$, and then $-\theta$. At the end of the walk, we will be facing the same direction as at the start and will not have performed a full revolution, so we can conclude that the sum of all these angles must be 0. Writing this as an equation, we get an expression for γ in terms of variables which have already been determined.

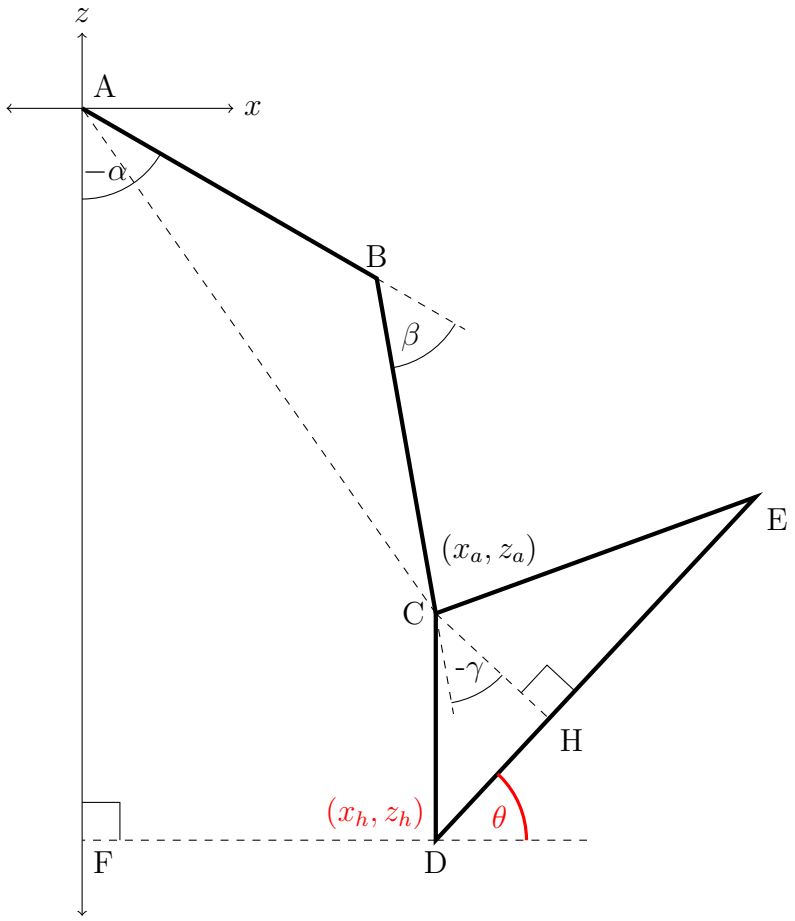
$$0 = -\frac{\pi}{2} - \alpha - \beta - \gamma + \frac{\pi}{2} - \theta \qquad \text{(Angle sum along } ABCHE\text{)} \qquad (1.9)$$

$$\gamma = \theta - (\alpha + \beta) \qquad (1.10)$$

1.3.5.2 Heel Position and Foot Angle

This is one of the core inverse kinematics equations that other control variable combinations are transformed into. It is used in every phase of the walk for the majority of calculations, although

the previous equations could have been used equally as well for most purposes. Our goal here is to take the control variables (x_h, z_h, θ) and calculate the joint angles (α, β, γ) from them .



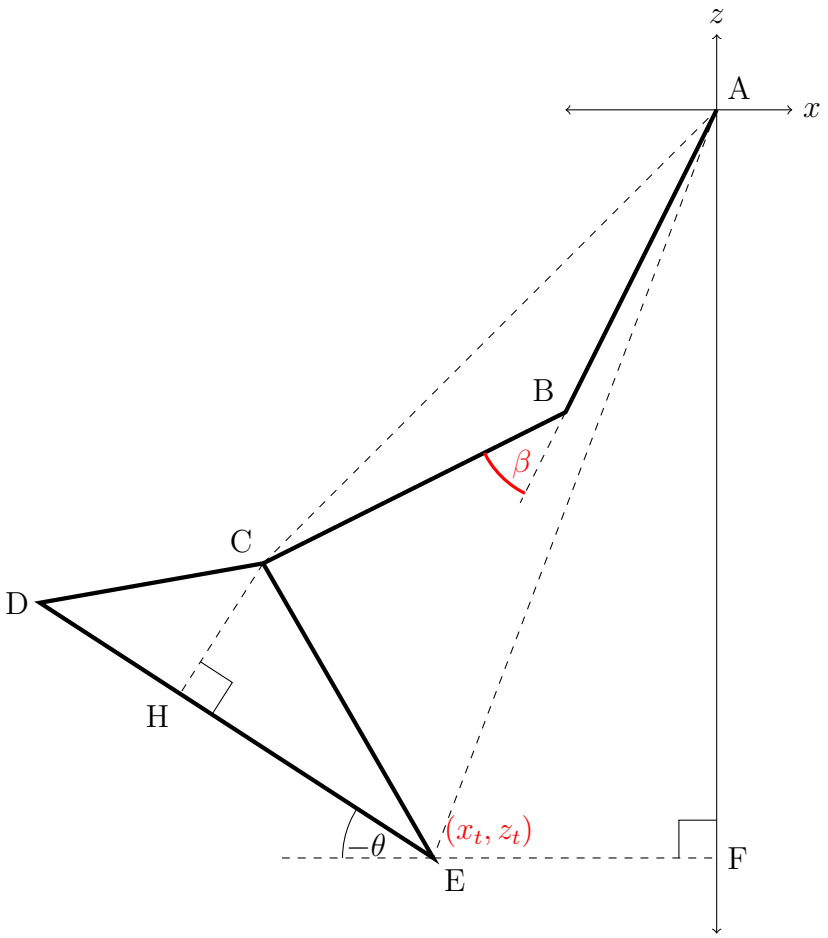
Conveniently, the problem is very similar to the above. Once we translate from (x_h, z_h) to (x_a, z_a), everything from there forward is exactly the same. We can achieve this translation in a similar way - by taking the vector $\vec{DC}$ and adding it to (x_h, z_h).

$$\vec{DC} = \begin{pmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{pmatrix} \begin{pmatrix} DH \\ CH \end{pmatrix} \qquad \text{(rotate } \triangle CDH\text{)} \qquad (1.11)$$

$$\begin{pmatrix} x_a \\ z_a \end{pmatrix} = \begin{pmatrix} x_h \\ z_h \end{pmatrix} + \vec{DC} \qquad \text{(simple vector math)} \qquad (1.12)$$

1.3.5.3 Toe Position and Knee Angle

This selection of control variables helped to overcome kinematics singularity issues in the `lift` phase, by allowing us to control the knee joint angle directly. Our aim is to take (x_t, z_t, β) and determine the angle θ, giving us (x_t, z_t, θ) to plug into inverse kinematics.



Our approach here is to simply determine the angles surrounding θ on the straight line EF. $\angle CEH$ is calculated using the inverse tangent function.

$$\angle CEH = tan^{-1}\left(\frac{CH}{EH}\right) \qquad \text{(Simple trigonometry in } \triangle CEH) \qquad (1.13)$$

$\angle AEC$ can be found using the cosine rule in $\triangle AEC$, but first requires the side AC to be determined by using cosine rule in $\triangle ABC$, and requires side AE to be determined by Pythagoras' theorem in $\triangle AEF$.

$$\angle ABC = \pi - \beta \qquad \text{(Supplementary angles)} \qquad (1.14)$$

$$AC^2 = AB^2 + BC^2 - 2(AB)(BC)cos(\angle ABC) \qquad \text{(Cosine rule in } \triangle ABC) \qquad (1.15)$$

$$AE = \sqrt{(x_t^2 + z_t^2)} \qquad \text{(Pythagoras on } \triangle AEF) \qquad (1.16)$$

$$\angle AEC = cos^{-1}\left(\frac{AE^2 + CE^2 - AC^2}{2(AE)(CE)}\right) \qquad \text{(Cosine rule in } \triangle ACE) \qquad (1.17)$$

$\angle AEF$ can be found through the inverse tangent function.

$$\angle AEF = tan^{-1}\left(\frac{z_t}{x_t}\right) \qquad \text{(Simple trigonometry in } \triangle AEF) \qquad (1.18)$$

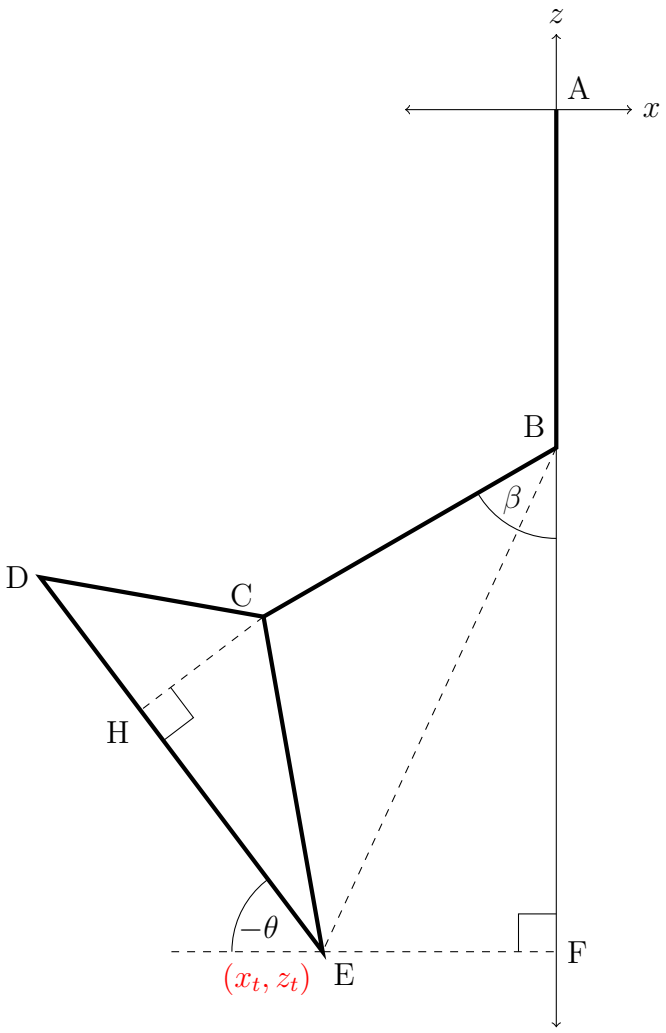
And finally, we can find θ by using the angle sum on the straight line EF.

$$-\theta = \pi - (\angle CEH + \angle AEC + \angle AEF) \qquad \text{(Supplementary angles)} \qquad (1.19)$$

1.3.5.4 Toe Position and Vertical Thigh

This selection of control variables was useful in determining the final pose of the `lift` phase. It was decided through observation of the human gait that an appropriate place for the `lift` phase to end and the `swing` phase to begin would be the place where the knee joint just passes below the hip joint. This is the position obtained when $\alpha = 0$. Our goal then is to take $(x_t, z_t, \alpha = 0)$ and get β, giving us (x_t, z_t, β). This in turn can be transformed into (x_t, z_t, θ).

Although it is possible to calculate θ without finding β, it is more useful to find β, since the `lift` phase interpolates over (x_t, z_t, β), and this transformation is used to find the final pose in terms of those control variables.



Our approach here is to find the angles $\angle EBF$ and $\angle CBE$ which add to get β. $\angle EBF$ can be found through the inverse tangent function in $\triangle BEF$, after first finding the length of BF.

$$BF = -z_t - AB \tag{1.20}$$

$$\angle EBF = tan^{-1}\left(\frac{EF}{BF}\right) \tag{1.21}$$

To find $\angle CBE$, we can use the cosine rule in $\triangle BCE$, but first we must find the length BE using Pythagoras' Theorem in $\triangle BEF$.

$$BE = \sqrt{BF^2 + x_t^2} \tag{1.22}$$

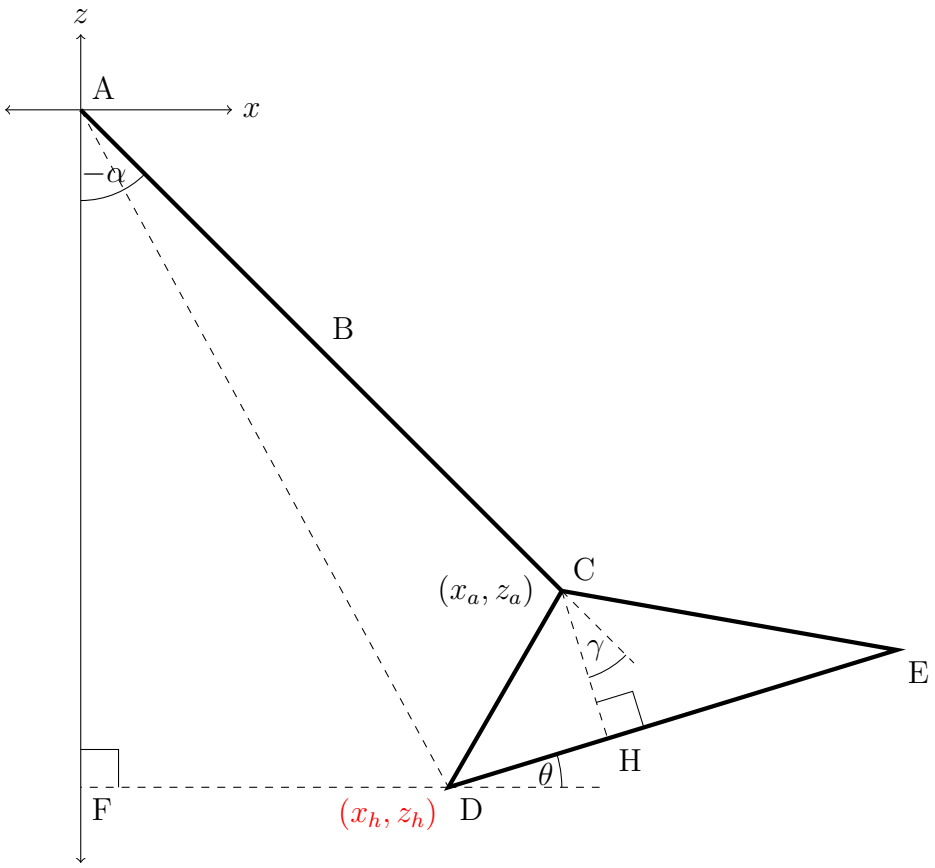
$$\angle CBE = cos^{-1}\left(\frac{BE^2 + BC^2 - CE^2}{2(BE)(BC)}\right) \tag{1.23}$$

And finally, we can find β by adding $\angle EBF$ and $\angle CBE$.

$$\beta = \angle EBF + \angle CBE \tag{1.24}$$

1.3.5.5 Heel Position and Straightened Knee

This selection of control variables was useful in both the `swing` phase and `rock` phase for controlling the front foot. In the `swing` phase it was used to determine the angle at which the heel should strike the ground, and in the `rock` phase it was used to interpolate throughout the phase's motion, allowing the foot's position relative to the torso to be set and the front foot's angle to the ground to be calculated. Our goal is to take $(x_h, z_h, \beta = 0)$ and get θ, giving us (x_h, z_h, θ) for use in inverse kinematics.



Our approach here is to simply determine the angles surrounding θ on the straight line FD. $\angle ADF$ can be found through the inverse tangent function.

$$\angle ADF = tan^{-1}\left(\frac{-z_h}{x_h}\right) \qquad \text{(Simple trigonometry in } \triangle ADF) \qquad (1.25)$$

$\angle ADC$ can be found by applying the cosine rule, but AD must first be calculated using Pythagoras' theorem.

$$AD = \sqrt{AF^2 + FD^2} \qquad \text{(Pythagoras on } \triangle ADF) \qquad (1.26)$$

$$\angle ADC = cos^{-1}\left(\frac{AD^2 + CD^2 - AC^2}{2(AD)(CD)}\right) \qquad \text{(Cosine rule in } \triangle ACD) \qquad (1.27)$$

$\angle CDH$ can be found through the inverse tangent function.

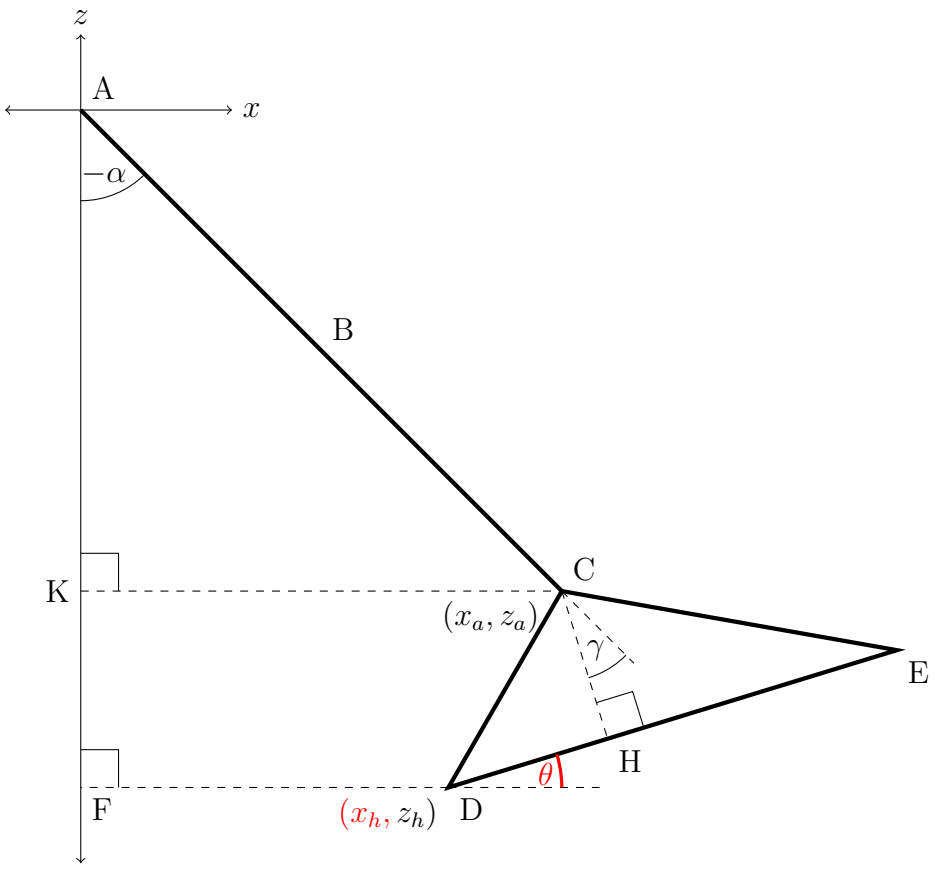
$$\angle CDH = tan^{-1}\left(\frac{CH}{DH}\right) \qquad \text{(simple trigonometry in } \triangle CDH) \qquad (1.28)$$

And finally, we can find θ by using the angle sum on the straight line FD.

$$\theta = \pi - (\angle ADF + \angle ADC + \angle CDH) \qquad \text{(supplementary angles)} \qquad (1.29)$$

1.3.5.6 Heel x, Foot Angle and Straightened Knee

This selection of control variables was useful in all phases for controlling the motion of the support foot. It allowed the x-coordinate of the support foot to change over time while allowing the knee to remain straight throughout the motion. Our goal is to form equations which take $(x_h, \theta, \beta = 0)$ and get z_h, giving us (x_h, z_h, θ) for use in inverse kinematics.



Since we know the orientation and dimensions of the foot, we can find the vector $\vec{DC}$. Through vector addition we can then determine the value of x_a from x_h.

$$\vec{DC} = \begin{pmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{pmatrix} \begin{pmatrix} DH \\ CH \end{pmatrix} \qquad \text{(rotate } \triangle CDH\text{)} \qquad (1.30)$$

$$\begin{pmatrix} x_a \\ z_a \end{pmatrix} = \begin{pmatrix} x_h \\ z_h \end{pmatrix} + \vec{DC} \qquad \text{(simple vector math)} \qquad (1.31)$$

$|x_a|$ is equal to the length CK, and we know the length AC, so using Pythagoras' theorem in $\triangle ACK$ we can determine the length AK, which is equivalent to $-z_a$, assuming that the ankle is positioned below the hip.

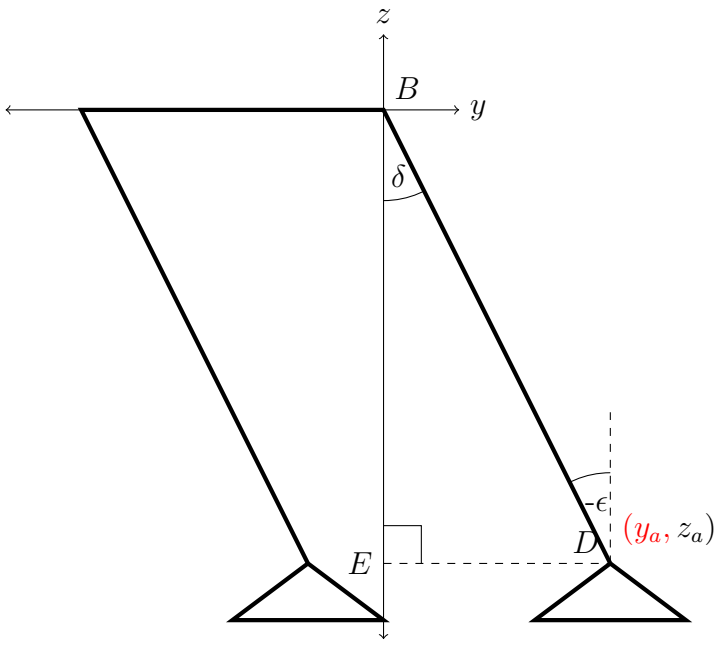
$$z_a = -\sqrt{AC^2 - x_a^2} \qquad (1.32)$$

Now by a rearrangement of the above vector mathematics, we can determine the value of z_h.

$$\begin{pmatrix} x_h \\ z_h \end{pmatrix} = \begin{pmatrix} x_a \\ z_a \end{pmatrix} - \vec{DC} \qquad \text{(simple vector math)} \tag{1.33}$$

1.3.5.7 Ankle y position

For this inverse kinematics equation, we instead take a cross-section along the Coronal plane, and determine the two angles ϵ and δ for the ankle and hip joints respectively which shift the foot to the desired position y_a along the y-axis. Clearly these angles must be the same for each leg if we are to keep the legs parallel. Note that the origin has been moved to the hip joint of the support foot, which means that the knee joint must be exactly straight, and the length BD is known.



Firstly we find δ by using the inverse sin function in $\triangle BDE$.

$$\delta = sin^{-1}\left(\frac{y_a}{BD}\right) \tag{1.34}$$

And we can see that δ is equal to ϵ as they are alternate angles on parallel lines, so we also get

$$\epsilon = -sin^{-1}\left(\frac{y_a}{BD}\right) \tag{1.35}$$

1.3.6 Interpolation Functions

In this section we describe the different interpolation functions which described the way in which motion during each phase was animated over time. When determining the pose of the robot mid-way through a phase, the engine knows what the pose of the robot was at the beginning of the phase, and what it intends the pose to be at the end of the phase. As mentioned in the Inverse Kinematics section, these initial and final poses can be defined by a wide variety of variable tuples, and during interpolation we can select the representation that gives the smoothest motion or correct control over positions. Once a representation is decided on, all that is left to do is to interpolate each of the three values in the tuple across the duration of the phase. The same interpolation function can be used for all three variables, or different functions can be used for each, depending on what is required in the motion.

An interpolation function here is defined as a function which takes in a variable $x \in [0, 1]$, representing our progress through the current phase's duration. In addition to this it takes in y_i and y_f, representing the initial and final values of the variable to be interpolated. The function returns y, which represents the value of the interpolated variable at x. Additionally the function may take a number of configuration parameters which change the shape of the function. If we graph an interpolation function, it must begin at the point $(0, y_i)$ and end at $(1, y_f)$, remaining continuous throughout.

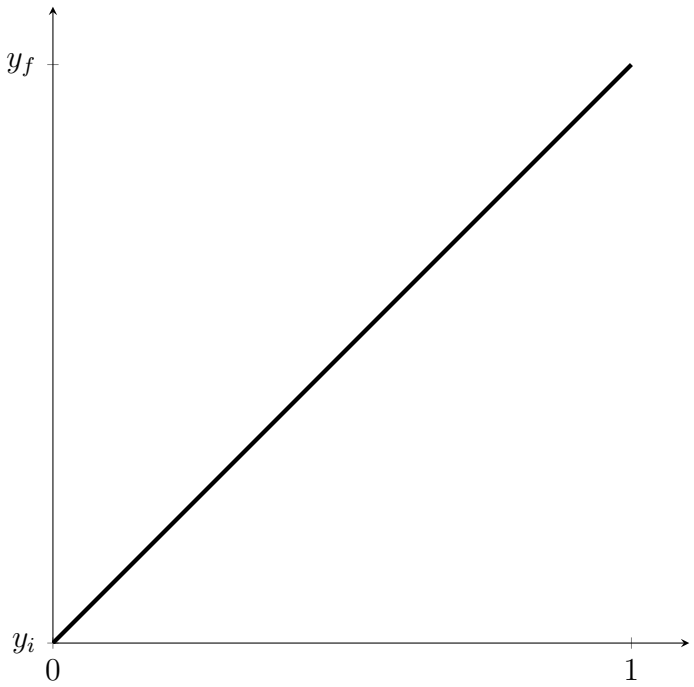
The functions used in this project were all polynomial based, and ranged from a simple linear function to complex functions allowing control over the rate of change or value at specific points in time. Each of the functions used is described below, along with their motivation, derivation, and usage in the walk.[2]

1.3.6.1 Linear Function

The linear function is a simple straight line from $(0, y_i)$ to $(1, y_f)$. This function was used extensively during initial stages of the walk's development, however as more complex interpolation functions were developed, its usage waned. In the final version of the walk, the linear function was only used to interpolate variables which were expected to have initial and final values which were equal.[3]

[2]Technically speaking, the equations used to derive the position of the centre of mass for stability purposes are also interpolation functions. However, these functions are omitted from this section since they are included in Section 1.3.7.

[3]We will see later in the report that phases could be interrupted before they were able to reach their final position, meaning that values that would have otherwise been the same needed to change over time.



Although trivial, we will derive this equation in a manner similar to latter interpolation functions. To derive the equation of this function, we note that there are two constraints on it, being that it must pass through the point $(0, y_i)$ and $(1, y_f)$. Taking a polynomial of degree 1 should therefore be able to satisfy these contraints:

$$y = ax + b \tag{1.36}$$

The two constraints on this function can be expressed as linear equations:

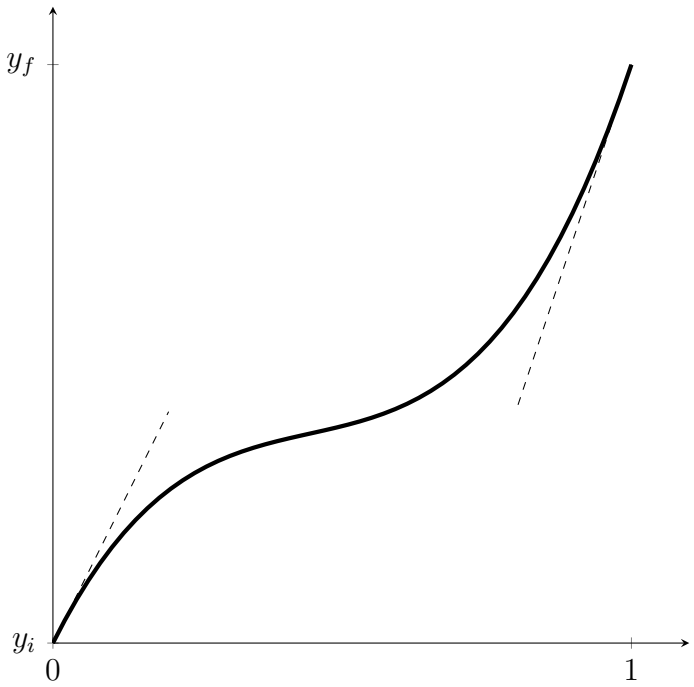
$$y_i = 0 + b \tag{1.37}$$
$$y_f = a + b \tag{1.38}$$

Solving this set of linear equations, we get

$$y = (y_f - y_i)x + y_i \tag{1.39}$$

1.3.6.2 Velocity Control Function

This function allowed the initial and final velocity of the interpolation to be controlled, while still beginning and ending in the correct places. It was used in the `lift` phase to control the knee joint. The knee joint needed to start with a velocity of 0, then end its movement with a velocity in preparation for the swing phase. Although it was only used for this one purpose, it was constructed in a parametrised way in case it became useful in other areas.



We have four constraints on the curve, two positional constraints and two constraints on the gradient. This requires a degree 3 polynomial, which has its general form and the derivative of this form below.

$$y = ax^3 + bx^2 + cx + d \tag{1.40}$$
$$y' = 3ax^2 + 2bx + c + 0 \tag{1.41}$$

If the desired initial velocity and final velocity are denoted as v_i and v_f respectively, then the constraints can be expressed as linear equations:

$$y_i = 0 + 0 + 0 + d \tag{1.42}$$
$$y_f = a + b + c + d \tag{1.43}$$
$$v_i = 0 + 0 + c + 0 \tag{1.44}$$
$$v_f = 3a + 2b + c + 0 \tag{1.45}$$

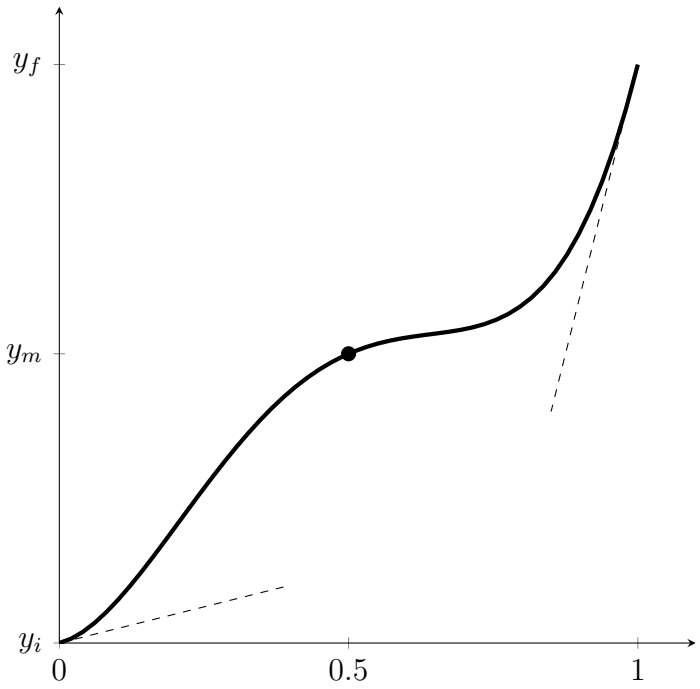
Solving these linear equations we get the following as our final interpolation function:

$$y = (-2y_f + 2y_i + v_i + v_f)x^3 + (3y_f - 3y_i - 2v_i - v_f)x^2 + v_i x + y_i \tag{1.46}$$

Note that the units of the velocities v_i and v_f will be skewed due to the x-axis being normalised between 0 and 1. To take a velocity in ordinary units and assign it its intended value to one of these variables, you must multiply by the duration of the interpolation.

1.3.6.3 Guided Velocity Control Function

This function is similar to the above Velocity Control Function, but in addition to allowing the control of the initial and final velocity it allows the control of the value of y when $x = 0.5$ (halfway through the interpolation). This was the function used to interpolate each joint in the `swing` phase. The swing foot had to pass through an intermediate pose during its motion in order to avoid hitting the ground, and this function allowed the introduction of that constraint. The intermediate pose was found by averaging the initial and final poses of the `swing` phase over the variables (x_h, z_h, θ), subtracting the parameter `foot_lift` from z_h, then running the result through inverse kinematics to get the pose's joint angles. Although forcing the interpolation function through a midway point did not guarantee that the swing foot would remain above the ground until intended, it did achieve this result for sensible combinations of parameters tested in the simulator.



We have five constraints on the curve, three positional constraints and two constraints on the gradient. This requires a degree 4 polynomial, which has its general form and the derivative of this form below.

$$y = ax^4 + bx^3 + cx^2 + dx + e \tag{1.47}$$

$$y' = 4ax^3 + 3bx^2 + 2cx + d + 0 \tag{1.48}$$

If the desired initial velocity and final velocity are denoted as v_i and v_f respectively, and the desired halfway position is denoted as y_m, then the constraints can be expressed as linear equations:

$$y_i = 0 + 0 + 0 + 0 + e \tag{1.49}$$
$$y_m = (0.5)^4 a + (0.5)^3 b + (0.5)^2 c + 0.5d + e \tag{1.50}$$
$$y_f = a + b + c + d + e \tag{1.51}$$
$$v_i = 0 + 0 + 0 + d + 0 \tag{1.52}$$
$$v_f = 4a + 3b + 2c + d + 0 \tag{1.53}$$

Solving these linear equations we get the following as our final interpolation function:

$$\begin{aligned} y = {} & (-8y_f + 16y_m - 8y_i - 2v_i + 2v_f)x^4 + (14y_f - 32y_m + 18y_i + 5v_i - 3v_f)x^3 \\ & + (-5y_f + 16y_m - 11y_i - 4v_i + v_f)x^2 + v_i x + y_i \end{aligned} \tag{1.54}$$

1.3.7 Stabilisation

Stabilisation is key to any walk, since a walk is not very useful if a robot is not upright. The unfortunate thing about interacting with the real world is that a physical environment is far less predictable than a simulated one. A gait which may balance perfectly within a largely accurate simulation can become an unsteady mess in practice, largely due to the inaccuracy of assumptions and the natural noise experienced in any physical environment.

The simulator used to develop this walk can hardly be described as "largely accurate", since some reasonably significant assumptions were made to simplify it. For starters, an entire dimension (the y-axis) was ignored, and this clearly needs to be addressed if the robot is to maintain its balance along this axis - if a foot is lifted off the ground, unless measures are taken to prevent it, the robot will begin to fall in the foot's direction. In addition to this, it was assumed that the torso would remain vertical at all times, implicitly assuming that each foot would maintain the intended angle with the ground at all times. Similar to above, unless precautions are taken, as soon as a foot is lifted from the ground the robot will begin to fall, breaking the upright torso assumption.

In order to tackle this problem, a simplified model of the dynamics of the robot was generated, and a trajectory for the centre of mass of the robot was planned for each step in order to ensure that the centre of pressure or zero moment point was kept in the centre of the support foot. The model used is referred to as the cart-table model, seeing a rolling cart moving atop a massless table, with a single leg and a flat base. This model was used by AIST in Japan in developing their HRP series bipeds [12].

Assumptions about the environment aside, noise is the other factor which makes stabilisation a difficult problem. Noise occurs not only in the sensors that the robot uses to perceive its environment, but also in the actuators that allow it to interact with its environment. If we send precisely identical instructions to the actuators of a robot at exactly the same time, and begin the robot in exactly the same position, it is likely that drastically different results will occur each time. This is especially true in the case of a bipedal walk, where the subtleties of the dynamics of the robot can have chaotic effects as time continues forward and errors compound.

In order to combat this, a robot must have a way to perceive its surroundings, gain a greater understanding of the current situation it is in, and react to it accordingly. This is achieved by

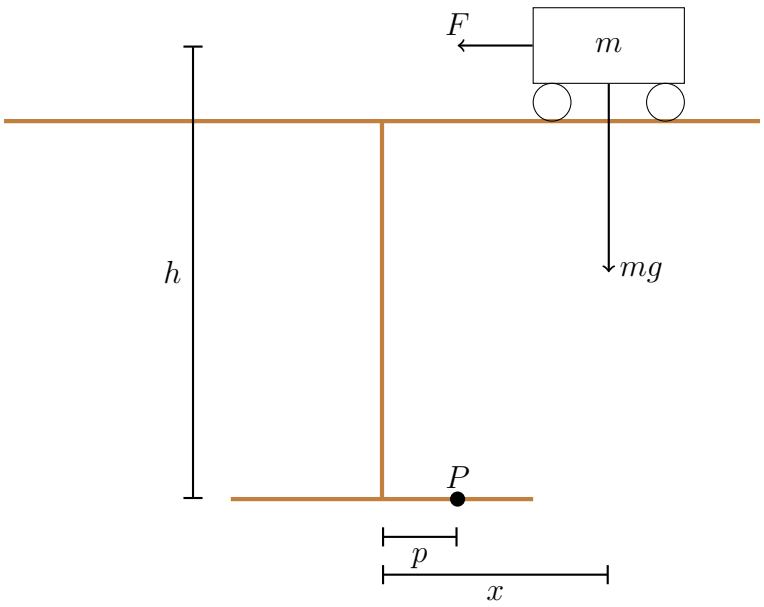
taking in sensor data throughout the walk, making adjustments to the walking gait to account for observations. In this project, two simple techniques were employed. The foot pressure sensors were used to detect when the swing foot unexpectedly made contact with the ground, causing the walk to enter the next phase if this occurred. Also, the gyroscope measurement around the y-axis was filtered and combined linearly with some of the joint angles in order to combat unexpected motions back and forth. These two techniques were inspired by Bernhard Hengst's work in the existing rUNSWift walk.

1.3.7.1 ZMP Trajectory

During a walking motion, it is a non-trivial task to calculate a gait which will cause the robot to remain balanced throughout its motion. Intuition will first say that it is as simple placing the centre of mass over the centre of the support foot, since this will create a balanced position. Although this is true in a static sense, it is not always true in a dynamic sense - if a robot has its centre of mass placed above its support foot, but has a significant velocity in a sideways direction, it is likely that the robot will fall down. This means that something more sophisticated must be used in order to express stability in a dynamic system.

The zero moment point is a construct commonly used to do just this. Intuitively it can be thought of as the centre of pressure of the support foot, where the majority of weight is placed across the foot's surface. Mathematically it can be defined as the point along the support foot which has a moment of 0.

A moment, also referred to as torque, is the tendency of a force to rotate an object about an axis, and is measured by multiplying the strength of a force by its perpendicular distance from an axis of rotation. Both the weight of the robot and the acceleration of its centre of mass contribute a moment at the foot, particularly accelerations along the Transverse plane. If the robot is in a dynamically stable state, then there exists a point on the support foot which we can select as the axis of rotation where the moment from these two forces cancel out, giving 0. If this point lies outside the foot, then we have lost stability.



Above we see the cart-table model, with the cart representing the centre of mass of the robot, and the foot of the table representing the supporting foot. This diagram can be interpreted as either a cross-section through the Sagittal plane or the Coronal plane, and is used in both contexts.

The cart weighs m kg and is accelerating to the right, causing a force F to be applied to the table in the opposite direction. If the acceleration of the cart is given by $\ddot{x}$, then the force F is given by

$$F = m\ddot{x} \tag{1.55}$$

Taking the moments M around the point P, with clockwise rotations being positive, we get

$$M = mg(x - p) - m\ddot{x}(h) \tag{1.56}$$

In order to find the location of the zero moment point, we set M to 0.

$$0 = mg(x - p) - m\ddot{x}(h) \tag{1.57}$$

$$p = x - \ddot{x}\frac{h}{g} \tag{1.58}$$

We wish to see what conditions must be met in order to have the zero moment point positioned in the centre of the foot, so we set p to 0, giving the following equation.

$$\ddot{x}\frac{h}{g} = x \tag{1.59}$$

$$\ddot{x} = x\frac{g}{h} \tag{1.60}$$

Here we see that the acceleration $\ddot{x}$ of the mass must be linearly proportional to its position x away from the centre in order to place the zero moment point in the middle of the foot. Our goal now is to craft an equation which will allow us to plan a trajectory for the mass between any two values of x while maintaining the correct acceleration to position the ZMP at the origin.

In order to do this, we need a function which when double differentiated gives us a function linearly proportional to the original. The hyperbolic functions *sinh* and *cosh* suit this criteria perfectly. We construct two functions which return x given a time t, having a constant $k = \sqrt{\frac{g}{h}}$, and two arbitrary constants a and b to parametrise the shape of these functions.

To calculate k, we need an approximation of h, the height of the centre of mass of the robot above the ground. This was measured experimentally by having the robot walk for a long period of time, recording the current centre of mass each frame, which was calculated by the existing kinematics system. The median of these values was taken as the value for h. g was approximated as $9.8ms^{-2}$. Fixing h as a constant is not strictly accurate, since as the robot walks the centre of mass moves up and down, however this motion is negligible, and this assumption greatly simplifies the mathematics.

Interestingly, a given trajectory is only solvable by one of *sinh* or *cosh* but not both, which is why we will work with both equations from here onwards. We now show that these functions hold the condition in (1.60) by differentiating twice, then substituting the original function for x back in, along with the value we assigned to k.

$$x = a\ cosh(kt - b) \qquad x = a\ sinh(kt - b) \tag{1.61}$$

$$\dot{x} = ka\ sinh(kt - b) \qquad \dot{x} = ka\ cosh(kt - b) \tag{1.62}$$

$$\ddot{x} = k^2 a\ cosh(kt - b) \qquad \ddot{x} = k^2 a\ sinh(kt - b) \tag{1.63}$$

$$\ddot{x} = \frac{g}{h} a\ cosh(kt - b) \qquad \ddot{x} = \frac{g}{h} a\ sinh(kt - b) \tag{1.64}$$

$$\ddot{x} = \frac{g}{h} x \qquad \ddot{x} = \frac{g}{h} x \tag{1.65}$$

So if we plan a trajectory between two values of x using one of the two above functions to describe the motion, then we will always satisfy the zero moment point condition, making the robot dynamically stable.

Let's say that the centre of mass is at position x_s at time $t = 0$, and that at a desired time t_f we wish to be at position x_f. This information can be summarised by these equations for each of *sinh* and *cosh*.

$$x_s = a\ cosh(-b) \qquad x_s = a\ sinh(-b) \tag{1.66}$$

$$x_f = a\ cosh(kt_f - b) \qquad x_f = a\ sinh(kt_f - b) \tag{1.67}$$

By solving a pair of these equations simultaneously, we can obtain the required values for a and b which describe the function that gives the desired trajectory. We attempt to solve these equations by substitution of a.

$$a = \frac{x_s}{cosh(-b)} \qquad a = \frac{x_s}{sinh(-b)} \tag{1.68}$$

$$x_f = \frac{x_s}{cosh(-b)} cosh(kt_f - b) \qquad x_f = \frac{x_s}{sinh(-b)} cosh(kt_f - b) \tag{1.69}$$

$$x_f\ cosh(-b) = x_s\ cosh(kt_f - b) \qquad x_f\ sinh(-b) = x_s\ sinh(kt_f - b) \tag{1.70}$$

We now substitute *cosh* and *sinh* for their exponential representations, and make b the subject of the equation.

$$x_f\left(\frac{e^{-b}+e^{b}}{2}\right) = x_s\left(\frac{e^{kt_f-b}+e^{b-kt_f}}{2}\right) \qquad x_f\left(\frac{e^{-b}-e^{b}}{2}\right) = x_s\left(\frac{e^{kt_f-b}-e^{b-kt_f}}{2}\right) \tag{1.71}$$

$$x_f e^{-b} + x_f e^{b} = x_s e^{kt_f-b} + x_s e^{b-kt_f} \qquad x_f e^{-b} - x_f e^{b} = x_s e^{kt_f-b} - x_s e^{b-kt_f} \tag{1.72}$$

$$x_f + x_f e^{2b} = x_s e^{kt_f} + x_s e^{2b-kt_f} \qquad x_f - x_f e^{2b} = x_s e^{kt_f} - x_s e^{2b-kt_f} \tag{1.73}$$

$$x_f e^{2b} - x_s e^{2b} e^{-kt_f} = x_s e^{kt_f} - x_f \qquad x_s e^{2b} e^{-kt_f} - x_f e^{2b} = x_s e^{kt_f} - x_f \tag{1.74}$$

$$e^{2b}(x_f - x_s e^{-kt_f}) = x_s e^{kt_f} - x_f \qquad e^{2b}(x_s e^{-kt_f} - x_f) = x_s e^{kt_f} - x_f \tag{1.75}$$

$$e^{2b} = \frac{x_s e^{kt_f} - x_f}{x_f - x_s e^{-kt_f}} \qquad e^{2b} = \frac{x_s e^{kt_f} - x_f}{x_s e^{-kt_f} - x_f} \tag{1.76}$$

$$b = \frac{1}{2} ln\left(-\frac{x_f - x_s e^{kt_f}}{x_f - x_s e^{-kt_f}}\right) \qquad b = \frac{1}{2} ln\left(\frac{x_f - x_s e^{kt_f}}{x_f - x_s e^{-kt_f}}\right) \tag{1.77}$$

Once we have a solution for b, we simply substitute it back into (1.68) to get a. Note that the solutions for b are almost identical except for the negative sign within the natural logarithm. Since $ln(x)$ is only defined for $x > 0$, we can determine what function to use based on the following conditions.

$$\frac{x_f - x_s e^{kt_f}}{x_f - x_s e^{-kt_f}} > 0 \qquad x = a\ sinh(kt - b) \tag{1.78}$$

$$\frac{x_f - x_s e^{kt_f}}{x_f - x_s e^{-kt_f}} < 0 \qquad x = a\ cosh(kt - b) \tag{1.79}$$

$$x_f - x_s e^{kt_f} = 0 \qquad \text{No solution} \tag{1.80}$$

Now we have successfully derived a closed-form trajectory function which will ensure that the ZMP will remain in the centre of the support foot.

As alluded to previously, this ZMP trajectory function is a kind of interpolation function, and is key to controlling the motion of the robot in both the forward/backward directions and the left/right directions. Whenever a foot was in contact with the ground, the foot's position (x, y) along the Transverse plane was determined by this function, so that the robot remained dynamically balanced throughout its motion. As a result this function was the most commonly used interpolation function throughout the walk.

Note that this function assumes that the support foot is at the origin, meaning that the frame of reference shifts whenever the support foot changes. Although the gait's state dictates that there is only one support foot per phase, for the purposes of these equations the `rock` phase only transitioned support foot halfway through its duration.

Unlike other interpolation functions, this function was not used from the beginning of a phase to its end, since the final position mid-way through a phase is difficult to determine. All that is known is that by the middle of the next `rock` phase, we need the torso to be positioned halfway between the ankles when looking along the Transverse plane. So this interpolation function spanned over entire steps as opposed to individual phases, implicitly determining the correct final poses of each phase as it interpolates.

1.3.7.2 Early Swing Contact Detection

Although calculating a ZMP trajectory greatly improved stability, this was still an open-loop technique that took in no feedback from the environment, meaning that it was still greatly affected by noise in the environment.

The problem that caused the greatest instability was during the `swing` phase, when the robot would lean slightly to the left or right, causing the swing foot to prematurely make contact with the ground. The robot continued to move its swing foot forwards, digging the foot into the ground, causing the robot to experience unexpected forces which ultimately caused the robot to fall.

The solution to this problem was to make use of the Force Sensitive Resistors found in each foot of the robot. There are four sensors in each foot, arranged in a rectangle, and all report a value between 0 N and 25 N. These values were summed for each foot to get a measure of the force applied on the foot as a whole. Noise experienced by all sensors meant that when the foot was in the air, the summed values were not necessarily 0. To combat this, an experimentally determined constant was selected as the threshold. If the sum of the FSRs in a foot exceeded this threshold, it was concluded that the foot must be touching the ground. This threshold worked out to be 0.2, although many values at a similar magnitude appeared to work well also.

In order to avoid detecting an early contact at the very beginning of the phase, when the foot is just leaving the ground, the FSRs were only compared against the pressure threshold after completing half of the `swing` phase.

When an early contact with the ground was detected, the `swing` phase was immediately concluded, and the rock phase began. The `stride_length` was adjusted to equal the current distance between the feet of the robot, in order to prevent the feet moving relative to each other and causing instability. This was measured as the difference in heel x-coordinates. In the event that the swing foot landed before or very close to the support foot, the `stride_length` was assigned a distance of at least 10 mm. The `movement_speed` of the robot was scaled down by the same percentage that the `stride_length` was reduced by, in order to maintain a consistent `stride_duration` and avoid jerkiness. After these adjustments, the other parameters were calculated as per usual, and the walk was allowed to continue.

So that the walk returned to its normal `stride_length` after recovering from early contact, the parameters were reset to their original values upon entering the swing phase.

Although the primary reason for implementing this mechanism was to detect early ground contact, it also detected late ground contact. The robot did not transition from the `swing` phase to the `rock` phase until the pressure threshold was exceeded. If the robot reached the end of the `swing` phase without detecting the ground, it stopped moving its foot forward. Once the foot (hopefully) hit the ground, the robot entered the `rock` phase and continued normally.

1.3.7.3 Rotational Adjustments

Just as movement across the y-axis was affected by noise, movement across the x-axis was also affected, causing the robot to sway back and forwards, often causing it to destabilise. In order to combat this, information from the robot's Inertial Unit was taken into account, and the robot's pose was adjusted in order to counteract instabilities before they had a chance to compound.

Specifically, the gyroscope's y-axis reading was taken into account. This reports the angular velocity of the robot's torso around the y-axis in radians per second, and is updated at 100 Hz.

The gyroscope also had to be calibrated, since the values reported by the gyroscope were sometimes offset by a constant, depending on the robot used. This was done by taking a long-running average of the gyroscope's values, making the assumption that if the robot does not perform any complete revolutions, accurate readings should average to exactly 0. Every time a new value was read from the gyroscope, the calibrated zero point was updated as follows.

$$new\_gyro\_zero = old\_gyro\_zero * 0.99 + gyro\_value * 0.01 \tag{1.81}$$

The value of the gyroscope reading itself was also averaged over time, in order to remove sharp changes in the gyroscope's values, which could cause undesirable feedback loops. This was calculated in a similar way to above, but each new reading was allowed to change the running average more quickly.

$$new\_filtered = old\_filtered * 0.8 + gyro\_value * 0.2 \tag{1.82}$$

This filtered reading was multiplied by an experimentally determined constant, `nod_gyro_ratio`, to give an adjustment angle. The decided value of this constant was 0.04, although many values of similar magnitude appeared to work just as well. After the pose of the robot was determined by the engine, this adjustment angle was added to particular joint angles in the robot to give an adjusted pose. Specifically these were the hip and ankle joints operating in the Sagittal plane.

The adjustment in the ankle served to push the robot onto its toes if a forward rotation was present, and onto its heels if a backward rotation was present. The adjustment in the hip served to move the torso (and therefore the centre of mass) in the opposite direction to the detected rotation.

1.3.8 Gait Anchors and Interpolation

Now that the inner workings of the walk have been described, a high level summary of how these pieces fit together is included here in order to bring clarity to an otherwise scattered collection of information.

As mentioned previously, the walk is defined at a high level by the poses that occupy the boundaries of each phase, called anchors, and by the functions that are used to interpolate between the anchors. Additionally, there is a selection of control variables to be made in respect to both defining the anchors and controlling the interpolation, described in detail in the Inverse Kinematics section (1.3.5).

The key difficulty in constructing the walk was selecting which control variables should be used where. The walk needed to be constrained correctly at all times in order to achieve its goal motion, anchors needed to be easily defined, and interpolation needed to be done in such a way that the kinematics singularity in the knee could be avoided, and such that all joint movements could be smooth.

After selecting which control variables were to be used in each phase, their values were either relatively obvious (keep the foot flat, keep the knee straight), or were derived from a mathematical model (ZMP trajectory). There were a few magic numbers here and there, but the definition of the walk's motion on the by and large worked out elegantly.

The movement of the limbs tended also to follow naturally from what was already defined. For example, the rate at which the foot was flattened against the ground after striking its heel was entirely determined by other constraints in the motion of the robot, namely that the knee needed to be straight, the centre of mass had a required trajectory, and the z-coordinate of the heel needed to finish in a location where if the foot was flat, the knee was straight.

We now see exactly how each phase's anchors and interpolation was calculated.

1.3.8.1 Determining x_a of the Support Foot

Throughout all phases the ZMP trajectory function is used to determine and interpolate the value of x_a of the support foot. We set its parameters accordingly, at the beginning of each phase, remembering that the origin is at the coordinates of the support foot's hip joint:

$$\begin{aligned} x_s &= \text{the } x_a \text{ value seen at the beginning of the phase} \\ x_f &= \text{half the stride\_length} \\ t_f &= \text{the time difference between the beginning of this phase} \\ &\quad \text{and halfway through the next rock phase} \end{aligned}$$

This is slightly different for the second half of the `rock` phase:

$$\begin{aligned} x_s &= \text{negative of half the stride\_length} \\ x_f &= \text{half the stride\_length} \\ t_f &= \text{the stride\_duration} \end{aligned}$$

1.3.8.2 Determining x_a of the Non-support Foot

Except during the motion of the swing phase, this is simply the x_a of the support foot subtracted by the `stride_length`.

1.3.8.3 Converting x_a into x_h or x_t

$$\begin{aligned} x_h &= x_a - \text{length of heel} \\ x_t &= x_a + \text{length of toe} \end{aligned}$$

1.3.8.4 The Rock Phase

1.3.8.4.1 The Front Foot The front foot was controlled at the heel in order to allow for a heel planting motion to be easily defined. The control variables in the final position and interpolation differed because controlling the angle of the foot created singularity problems during interpolation. By controlling the height of the heel throughout the interpolation, the ankle could take the values which were required to achieve that position.

A linear function was used for z_h's interpolation because the value was not expected to change greatly, and produced a smooth movement in the ankle joint despite not being complicated.

The resultant value for θ throughout interpolation was not allowed to exceed 0, since this would result in the foot digging its toe into the ground.

Final position control variables	$(x_h, \theta, \beta = 0)$
x_h	ZMP trajectory function
θ	0
Interpolation control variables	$(x_h, z_h, \beta = 0)$
x_h	ZMP trajectory function
z_h	linear function

1.3.8.4.2 The Rear Foot The rear foot was controlled at the toe in order to allow the rear foot to lift its heel if required, ready for the `lift` phase. z_t is kept equal to the front foot's z_h so that the feet are at the same vertical distance from the torso, keeping the robot upright and feet flat against the ground. z_t was interpolated linearly because the front foot's z_h was also interpolated linearly. The resultant value for θ throughout interpolation was not allowed to decrease below 0, since this would result in the foot digging its heel into the ground.

Final position control variables	$(x_t, z_t, \beta = 0)$
x_t	ZMP trajectory function
z_t	front foot's z_h
Interpolation control variables	$(x_t, z_t, \beta = 0)$
x_t	ZMP trajectory function
z_t	linear function

1.3.8.5 The Lift Phase

1.3.8.5.1 The Support Foot The support foot was controlled using the heel in order to match what was used in the previous phase. Choosing these control variables was an obvious choice given that the support foot's key constraintn is remaining flat against the ground. θ was interpolated linearly as it was expected to remain constant throughout the phase.

Final position control variables	$(x_h, \theta, \beta = 0)$
x_h	ZMP trajectory function
θ	0
Interpolation control variables	$(x_h, \theta, \beta = 0)$
x_h	ZMP trajectory function
θ	linear function

1.3.8.5.2 The Lift Foot The lift foot was controlled by the toe in order to easily define a heel lifting motion, pivoted on the toe. The final pose was chosen by finding the angle that the knee will need to have as it passes underneath the hips because it not only emulates roughly what the human gait does, but provided sensible values for the final anchor in all observed parameter combinations. z_t is equal to the support foot's z_h so that the feet are at the same vertical distance from the torso, keeping the robot upright.

Final position control variables	$(x_t, z_t, \alpha = 0)$
x_t	ZMP trajectory function
z_t	support foot's z_h
Interpolation control variables	(x_t, z_t, β)
x_t	ZMP trajectory function
z_t	support foot's z_h
β	velocity control function

The velocity control function is set the following parameters. The constant used in calculating v_f was experimentally determined by observing what worked best for various combinations of parameters to the walk.

$y_i =$ the angle β seen at the beginning of the phase
$y_f =$ the angle β calculated by the final position after inverse kinematics
$v_i = 0$
$v_f = -0.005$(lift_duration)

1.3.8.6 The Swing Phase

1.3.8.6.1 The Support Foot Again, this was an obvious choice of control variables for the support foot. Its key constraint is to remain flat on the ground. θ was interpolated linearly as it was expected to remain constant throughout the phase.

Final position control variables	$(x_h, \theta, \beta = 0)$
x_h	ZMP trajectory function
θ	0
Interpolation control variables	$(x_h, \theta, \beta = 0)$
x_h	ZMP trajectory function
θ	linear function

1.3.8.6.2 The Swing Foot The swing foot's final position was defined using the heel as it is the heel which is intended to make first contact with the ground. The heel's final z-coordinate was made equal to that of the support foot at the end of the `swing`, meaning the torso should be vertical when both feet are in contact with the ground.

The individual joint angles were interpolated for the swing foot because it had no hard constraints for its position during its motion. Instead of manually crafting what looks like a smooth swing, the interpolation functions were allowed to generate a smooth motion. A midway point in the interpolation was used to prevent the swing foot from colliding with the ground, and to allow control of the step height.

The final z_h is equal to the support foot's z_h so that the feet are at the same vertical distance from the torso when the heel strikes the ground, keeping the robot upright.

Final position control variables	$(x_h, z_h, \beta = 0)$
x_h	ZMP trajectory function
z_h	support foot's z_h
Interpolation control variables	(α, β, γ)
α, β and γ	guided velocity control function
Midway position control variables	(x_h, z_h, θ)
x_h	average of initial value and final value of x_h
z_h	$foot\_lift$ + average of initial value and final value of z_h
θ	average of initial value and final value of θ

Below are the parameters to the α guided velocity control function. The value for v_f was experimentally determined, and worked well for the range of parameters that were actually seen on the robot.

$$
\begin{aligned}
y_i &= \text{the angle } \alpha \text{ seen at the beginning of the phase} \\
y_m &= \text{the angle } \alpha \text{ given by the midway position after inverse kinematics} \\
y_f &= \text{the angle } \alpha \text{ given by the final position after inverse kinematics} \\
v_i &= \text{the velocity of } \alpha \text{ in the } joint\_velocities \text{ property of the state} \\
v_f &= -0.2
\end{aligned}
$$

Below are the parameters to the β guided velocity control function.

y_i = the angle β seen at the beginning of the phase
y_m = the angle β given by the midway position after inverse kinematics
y_f = the angle β given by the final position after inverse kinematics
v_i = the velocity of β in the *joint_velocities* property of the state
$v_f = 0$

Below are the parameters to the γ guidede velocity control function.

y_i = the angle γ seen at the beginning of the phase
y_m = the angle γ given by the midway position after inverse kinematics
y_f = the angle γ given by the final position after inverse kinematics
v_i = the velocity of γ in the *joint_velocities* property of the state
$v_f = 0$

1.4 Results

A variety of results are collected here, with the aim of demonstrating the degree to which the key goals of this project were achieved, both in the simulator and reality. Because some results are difficult to represent numerically, some qualitative observations about the walk will be included in addition to numerical data for a clearer perspective on the final outcome.

1.4.1 Open Challenge Result

This thesis was presented in short at the RoboCup 2014 SPL Open Challenge in Brazil, which was a competition open to any form of research in autonomous systems that teams were embarking upon. Each project was allowed a 3 minute presentation to describe the aims and methodology behind their project, and to demonstrate it live.

Our presentation involved a description of the motivation behind the walk, a breakdown of how the walk was constructed, and a demonstration of the walk in both the simulator and on the physical robot.

The presentations were subject to peer review in the form of preferential voting, where each team could list their ordered top 10 choices for the most impressive project. According to the technical challenge manual, the suggested criteria for voting were "technical strength, novelty, and expected impact and relevance to RoboCup" [18]. All 20 teams in the league were required to vote, with each team consisting of a number of research students and professors from universities around the world. Different preferences for the best project got different amounts of points. The 1st ranking item on a voting sheet was awarded 10 points, the next was awarded 9 points, etc until the 10th item was awarded a single point. Once all ranks were submitted, they were summed for each team and normalised such that the top score was 25.

This project was awarded 2nd place among 11 other competing teams. The final results for each team in the Open Challenge are displayed in Figure 1.10.

1.4.2 Straightened Knee

Within the simulator, the knee of the support foot was straight at all times, in accordance with the goals of the project. A plot of the angles of each joint can be seen in Figure 1.11a, demonstrating this.

However, on the robot, due to the added stability techniques, this condition was no longer strictly true at all times. The early contact detection system, which was essential to having the robot walk, meant that the `swing` phase would often finish while the knee was still bent, moving on into the `rock` phase where the bent leg then became a support foot. By the end of the `rock` phase the knee was straightened, and remained straight while it was weight bearing. In situations when the support knee ended up bent, it was generally not bent by a significant degree.

Due to time constraints, no data on the joint angles of the real robot has been collected for review here, but the above observations were clear based purely on observing the walk.

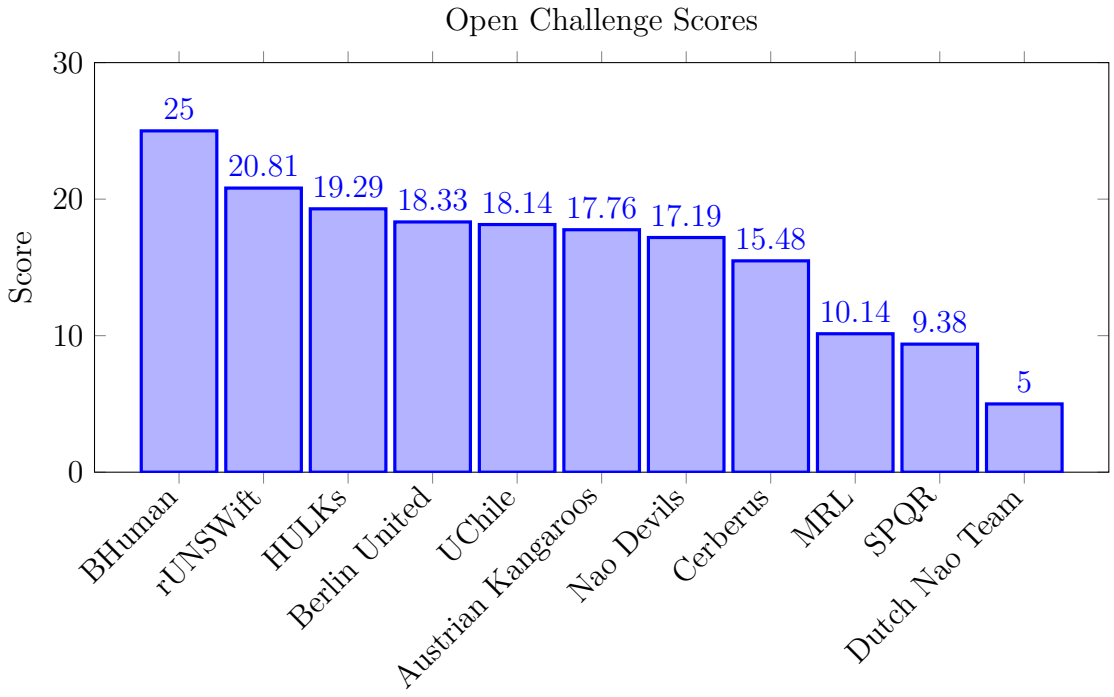


Figure 1.10: *The final preferential voting scores of the RoboCup 2014 SPL Open Challenge. This project was awarded 2nd place among 11 other competing teams*

1.4.3 Smooth Joint Movement

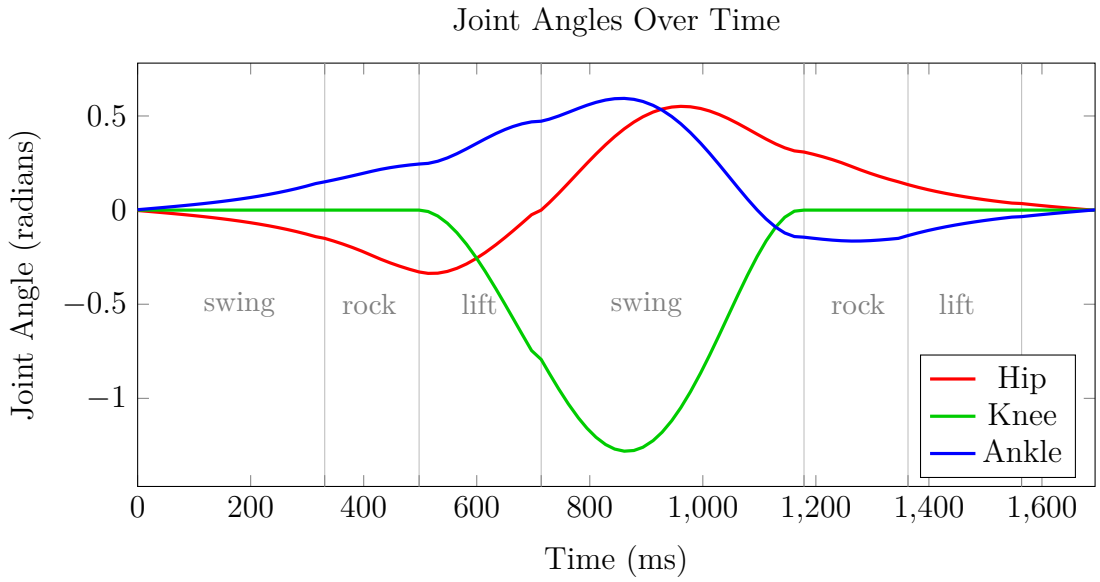
Within the simulator, the joints moved quite smoothly throughout. This is demonstrated by the plot of the instantaneous velocities of the joints shown in Figure 1.11b, where changes in velocity from frame to frame are small. The instantaneous velocities were calculated simply as the difference between two consecutive values of a joint angle, divided by the time that elapsed between them. This calculation is very sensitive to small changes in velocity, giving a scrutinous look into the rate of change of the joint angles.

On the physical robot, joint smoothness was somewhat harmed by the early contact detection system. This interrupted the expected flow of the gait, and caused joint angles to change velocities quickly, jolting the robot. The `rock` phase, which expected the knee to be straight upon entry, did not do a smooth job of straightening the knee back out during its duration. Neither did it often have much time to do so - the rock phase typically had a duration of 200ms.

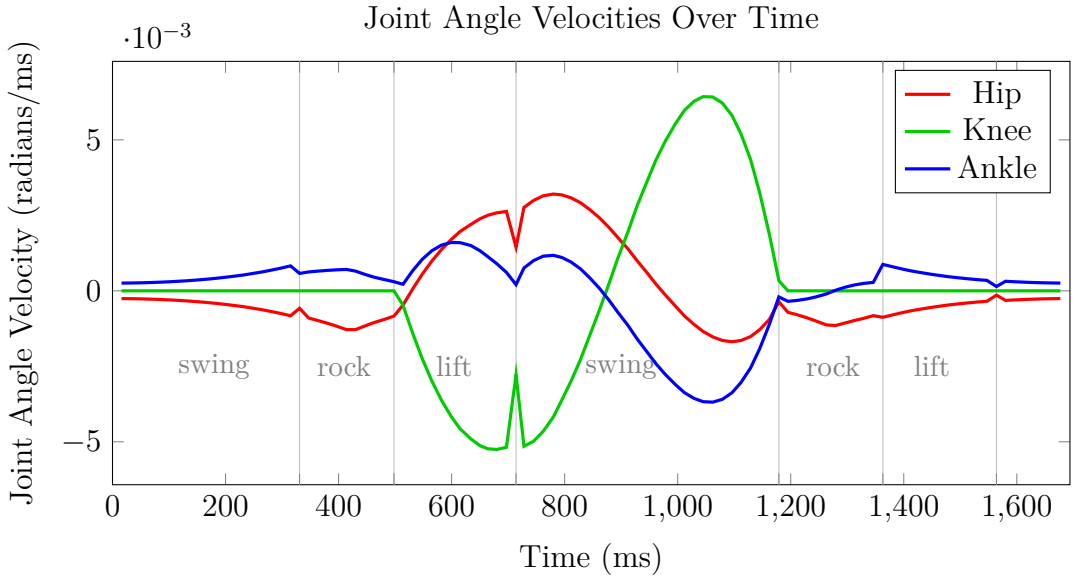
1.4.4 Stability Techniques

Stability was in general, a success, in that the robot was able to walk for a large amount of steps without falling, at a target pace of 120 mms^{-1}. The most amount of steps taken was never measured explicitly, but walks in the order of 100 steps were achieved. Unsurprisingly, prior to employing stability techniques, the robot was not able to take a single step unaided, which is a testament to its effectiveness.

Figure 1.11: *Graphs demonstrating the pattern and smoothness of joint values over time*



(a) *A plot of the joint angles for a single leg produced by the walk engine in a typically parametrised gait in simulation.*



(b) *A plot of the joint velocities for a single leg produced by the walk engine in a typically parametrised gait in simulation.*

1.4.4.1 ZMP Trajectory

From observing the effects of the ZMP trajectory in both the simulator and reality, the results appear positive. A natural looking sway was introduced into the hips along the y-axis, allowing the centre of pressure to remain in the support foot, and stopping the robot from falling the moment the swing foot is lifted from the ground. Movement along the x-axis also showed desirable properties, approaching a linear trajectory as movement speed increased, but as movement speed was decreased, the torso hesitated above the support foot during the swing phase in order to control the zero moment point and maintain balance.

In order to analytically determine the success of planning the trajectory of the ZMP however, a way of measuring its position in practice was required. Since the ZMP is theoretically equivalent to the centre of pressure on the foot, an attempt was made to use the four foot sensitive resistors to estimate the ZMP's position in the Transverse plane, interpolating between the positions of the sensors based on how much force they were experiencing. However, this technique did not give useful information, since the sensors reported positions along the rim of the foot the majority of the time, due to some sensors reporting 0 while others did not.

The technique that was used instead was to measure the trajectory of the centre of mass of the robot using the existing kinematics system developed by the rUNSWift team [19]. This system uses the angles reported by each joint to construct a three dimensional model of the robot, with the position and orientation of each limb calculated and averaged in a Euclidean fashion to find the centre of mass.

In this system, the origin is defined as the ground just below the ankle joint of the support foot, incorporating measurements from the robot's inertial unit to approximate the relative rotation of the foot to the ground. For these experiments, the origin was moved to the position directly below the centre of mass when the robot is in an upright position, and was fixed to the ground instead of following the robot. This meant that the robot moved away from the origin along the x-axis as it walked forward. The reason for placing the origin here was that it was advantageous to have it fixed in place instead of snapping between the support feet as they changed.

In order to adjust the reported values of the centre of mass to suit this new origin, the y-axis value had 50 mm either added or subtracted to it depending on whether the support foot was on the right or left respectively. The x-axis values were more difficult to treat, with the values for each step needing to be stitched together, having been offset by the stride length each time. The stitching process on the x-axis was not perfectly accurate, and affected the ZMP calculation in places close to where the support foot was changed. It is thought that this inaccuracy is a result of errors in the joint readings through the kinematics chain between the feet, which causes an unexpected change in value when the support foot is changed.

Once the centre of mass' trajectory has been expressed in terms of a consistent origin, we can calculate the ZMP of the system under the cart-table model, as expressed in Section 1.3.7.1. Recall that the ZMP along an axis can be found by the following equation:

$$p = x - \ddot{x}\frac{h}{g}$$

We already have values for h, g and x at this point, so now we must derive values for $\ddot{x}$, the

acceleration of the centre of mass. This was done by first calculating the velocity of the centre of mass at each point, and then using the same technique on that calculation to get the acceleration.

The velocity/acceleration at time t was calculated at each time step by observing the position/velocity values at time $t \pm 50ms$, then dividing their difference by $100ms$. The reason for using a window of $100ms$ was to retrieve intelligible values from the data, which contained noise. The window had the effect of averaging the velocity/acceleration over a small period of time, making the effect of noise less damaging to the results.

The end result of this process can be seen in Figure 1.12, in both the x and y axes. The intended position of the ZMP is plotted against the calculated position of the ZMP for reference, along with the position of the centre of mass.

1.4.4.1.1 Data to Support Cart-Table Model Assumptions The accuracy of the cart-table model hinges on the inaccuracy of the simplifying assumptions that it makes. One of the largest assumptions made was that of the constant height of the centre of mass throughout the motion of the walk. This assumption was tested in order to determine how warranted the use of the cart-table model was, and whether it may have been better to develop something more complex.

The constant height assumption is known to be acceptable for walks which aim to keep the torso at a constant height, but this walk had no such constraint, and in fact the torso did oscillate up and down throughout the gait both in the simulator and on the robot. This bob was a natural consequence of keeping the support knee straight; a similar characteristic can be observed in the human gait.

In order to test this assumption, the height of the centre of mass throughout the walk was plotted in order to observe its deviations away from its assumed value. This assumed value used in the final version of the walk was $h = 242.971$.

The plot can be seen in Figure 1.14. Visually we can see that the value does not deviate very far away from its median, with only the occasional spike occurring when the support foot is changed. In the larger data set that this figure was derived from, it was found that 95.0% of all values lay between 242.971 ± 2.335. The constant h's only contribution to the ZMP trajectory equations is in the ratio $k = \sqrt{\frac{g}{h}}$. Across this 95.0% interval for h, the value of k ranges from 0.00632 to 0.00638.

1.4.4.2 Early Swing Contact Detection

This mechanism of stabilisation appeared to be both a blessing and a curse. Without a doubt, the walk simply could not function without its inclusion. It successfully prevented the robot from kicking its feet into the ground as a result of destabilisation by stopping the swing phase early, and actively helped the robot recover from destabilisations, including relatively severe ones.

However, this mechanism sometimes generated a sharp jerk in the support foot as it made first contact with the ground, which tended to destabilise the robot. In general, the earlier in the swing phase that the mechanism was triggered, the sharper the jerk became.

The graph in Figure 1.15 shows the length of the step that the robot was able to make in each foot during a heel to toe walk. The robot aimed to make a step of 100mm each time, but often fell short due to instabilities. The shorter the step length achieved, the less stable the robot is deemed

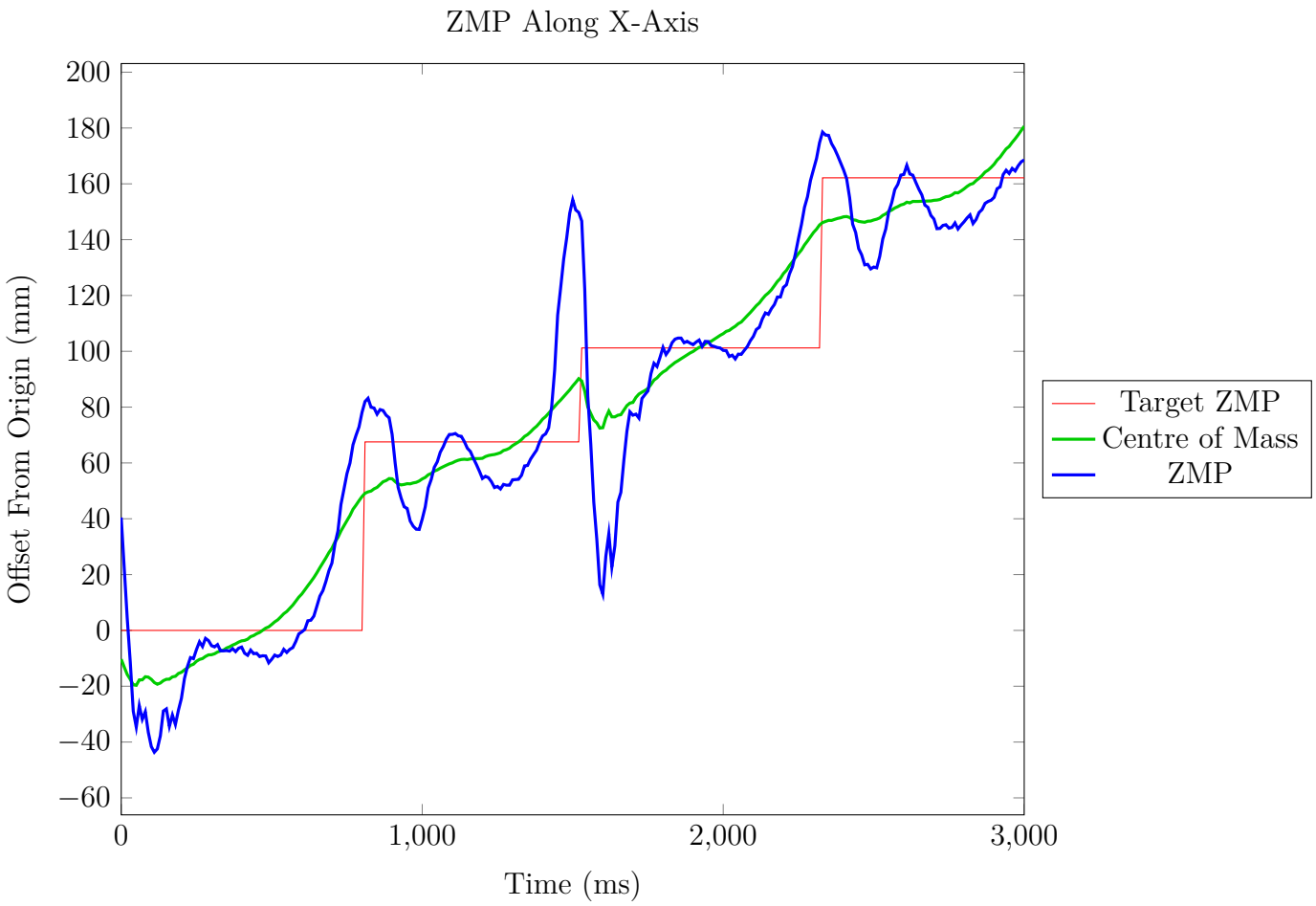
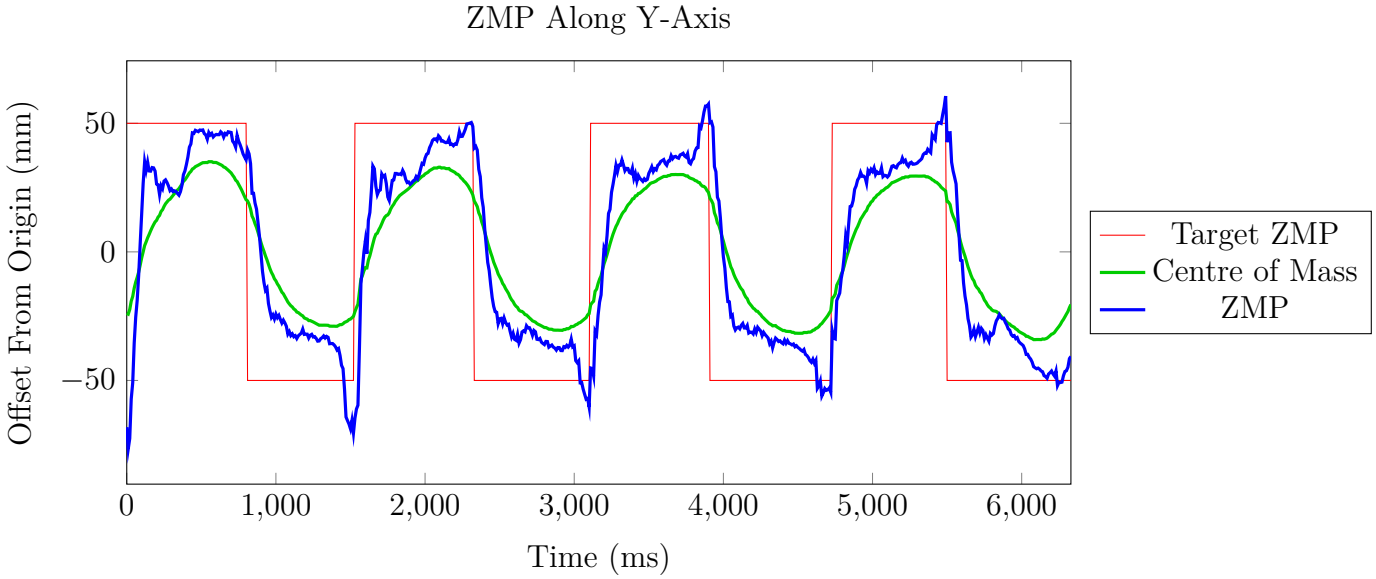


Figure 1.12: *Here we see plots of the coordinates of the Zero Moment Point during a forwards heel to toe walk, along with its target value. The ZMP was calculated based on the centre of mass' trajectory, which was recorded from the robot's existing kinematics system. The origin was defined at time 0 as the position of the centre of mass in the upright standing position, relative to the ground.*



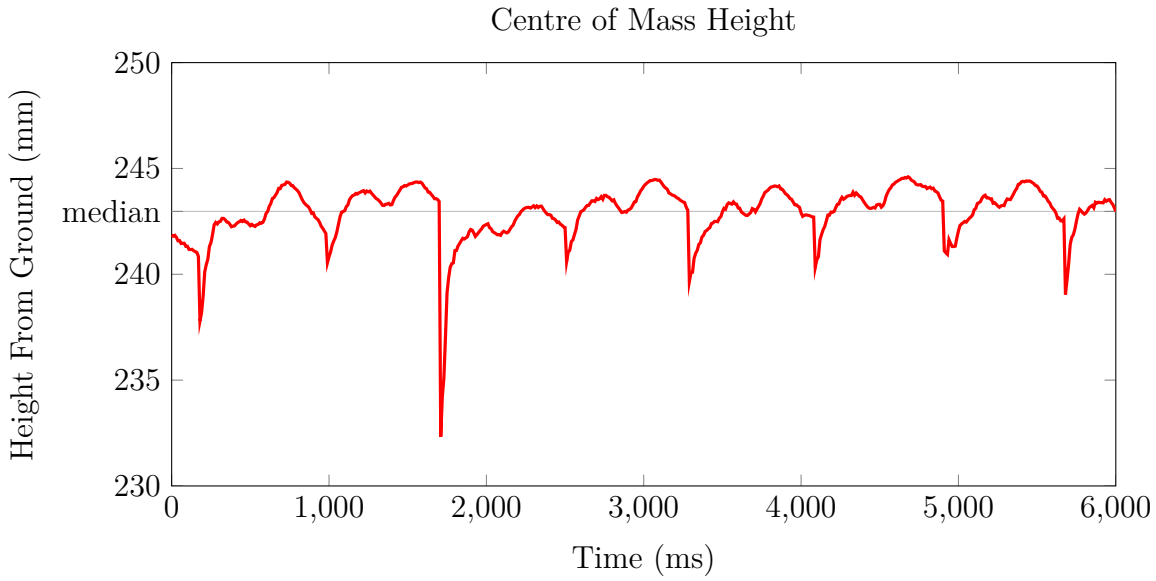


Figure 1.14: *A plot of the height of the centre of mass of the robot during a heel to toe walk, measured using the existing kinematics system. The median taken to approximate the height of the centre of mass is displayed for comparison to the real value's oscillations. It was found that 95.0% of all values lay between* 242.971 ± 2.335

to be. Comparing the step length across the left and right feet can reveal information about the lean of the robot throughout the walk.

We see the robot walks in a stable manner for up to 7 of the steps shown in each foot, but after this it destabilises significantly. The mechanism however recovers successfully, and continues walking afterwards.

1.4.4.3 Rotational Adjustments

The addition of this stabilisation technique provided a clearly noticeable improvement over the stability of the walk without it. Although the walk could take a few steps on its own, oscillations back and forth along the x-axis quickly magnified and caused a destabilisation.

Due to time constraints, data useful to the evaluation of this technique is not presented here. However, this technique was not an original work of this thesis, and so is in less need of critical analysis than other items.

1.4.5 Efficiency Comparison to rUNSWift Walk

As a way of measuring the efficiency of the heel to toe walk presented in this report, the current rUNSWift 2014 Walk was used as a baseline for comparison. The "efficiency" of a walk is expressed here as the distance travelled per unit of expended energy.

Although the distance travelled is relatively easy to approximate, energy expenditure is not, since it is not reported by the motors of the NAO. However, the exact value of energy expenditure is not required; all that is required is a measure proportional in some way to energy expenditure, then a

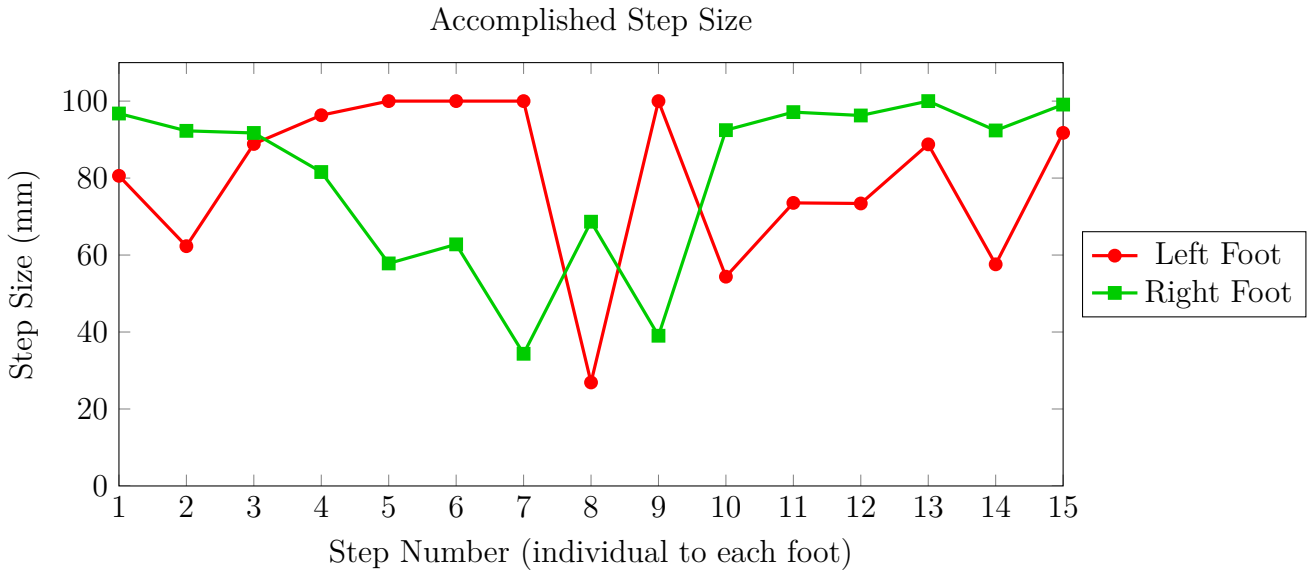


Figure 1.15: *Here we see the various lengths of the steps that the robot was able to make during a heel to toe walk. The intended step size was 100mm, but due to early swing contact detection, the step size is often shortened. At step 8 we see that the robot has become significantly destabilised, but recovers by step 10*

comparison can be made. It is assumed here that an increase in temperature in a joint motor is proportional in some way to the expenditure of energy, since energy is required to do work and invariably loses some of this energy to heat.

If we observe a robot over a period of Δt seconds, at a pace of $s\ ms^{-1}$, then we can get the distance travelled by

$$\Delta x = \Delta t s \tag{1.83}$$

If we assume that an increase in temperature ΔT is proportional to energy expenditure, then the following should be proportional to efficiency.

$$e = \frac{\Delta x}{\Delta T} \tag{1.84}$$

$$e = \frac{\Delta t s}{\Delta T} \tag{1.85}$$

To simplify the experiment, and to mitigate errors created by the assumptions above, s was kept constant at $0.12 ms^{-1}$. The robots involved in the experiment were tweaked such that they actually achieved this pace, not simply aimed for this pace. Since s was kept as a constant, then the following should be proportional to efficiency.

$$e = \frac{\Delta t}{\Delta T} \tag{1.86}$$

If we plot temperature readings over time, then the gradient over a particular time interval Δt is given by.

$$m = \frac{\Delta T}{\Delta t} \tag{1.87}$$

$$m = \frac{1}{e} \tag{1.88}$$

Hence the gradient of the graph of temperature over time is inversely proportional to efficiency, or in other words proportionally to inefficiency. This is therefore a sufficient measure for use in comparing the energy efficiency of the walks.

The following precautions were taken in order to control the variables of the experiment.

- Both walks operated at exactly the same speed in a straight line, in an attempt to make their required work as similar as possible.
- Two robots were used, each running both walks, to control for faults or oddities in the robots.
- The temperatures of all motors were allowed to cool to their natural temperatures before running an experiment.
- Tests were performed in the same air-conditioned room on the same surface.
- Each time, the robots were run non-stop until they complained about overheating in any given joint, to give the largest amount of useful data possible.

Figure 1.16 and Figure 1.17 show the results of the experiment. Because temperature readings were only available in integral values, temperature curves contained distinct steps as their values increased. In order to make the graphs more readable, only the changes in temperature reading were considered for plotting in the graphs.

The data shows that the heel to toe walk was in the order of 50% more inefficient than the existing walk. The motors labelled with "Roll", indicating that they rotate about the x-axis, were much more power hungry in the heel to toe walk when compared to the existing walk. The next biggest perpetrators are the ankle pitch motors, which consume energy almost twice as fast as the same motors in the rUNSWift walk. The knee joints appear to have been operating at comparable efficiency between the walks, with the heel to toe walk having the slight upper hand.

It appeared that Hotdog's left leg heated up more quickly than Husker's right leg, and vice versa. The reasons for this can only be speculated about, but it is hoped that in running the experiment on both robots, these unique nuances have been somewhat averaged out.

1.4.5.1 Investigation into Effect of Bent Knees on Effeciency

In order to determine whether a bent knee is at all less efficient than a straight knee in a servo driven biped, another small experiment was conducted, similar in nature to the walk temperature experiment. The joint temperatures over time of a single robot (Husker) were measured while standing in a perfectly upright position, and also while standing with a 90 degree bend at the knees. The results from this experiment have been plotted in Figure 1.18.

	Hotdog			Husker		
Joint	Heel	2014	Ratio	Heel	2014	Ratio
LAnklePitch	**31.54**	11.39	2.76	**25.98**	15.67	1.65
LAnkleRoll	**27.90**	9.97	**2.79**	**24.90**	7.31	**3.40**
LHipPitch	**48.52**	41.31	1.17	**37.89**	34.49	1.09
LHipRoll	**41.24**	7.12	**5.79**	**37.89**	7.31	**5.17**
LKneePitch	50.95	**62.68**	1.23	38.98	**41.81**	1.07
RAnklePitch	**27.90**	22.79	1.22	**29.23**	15.67	1.86
RAnkleRoll	**23.05**	8.54	**2.69**	**25.98**	6.27	**4.14**
RHipPitch	32.75	**35.61**	1.08	**38.98**	33.44	1.16
RHipRoll	**40.03**	8.54	**4.68**	**43.31**	7.31	**5.91**
RKneePitch	**35.18**	31.34	1.12	**44.39**	43.90	1.01
Average	**32.86**	22.01	1.50	**31.89**	19.57	1.63
Knee Average	43.06	**47.01**	1.09	41.68	**42.85**	1.02

Figure 1.16: *A table of temperature increase rates for both the heel to toe gait and rUNSWift 2014 walk on two different robots, Hotdog and Husker. Temperature rates are shown in degrees Celsius per 1000 seconds. For each robot, the higher of the two rates for each walk is coloured, with the overall highest rate for a joint coloured red. The ratio between the larger rate and the smaller rate for each robot's joints is also given, with the "Roll" joints' ratios bolded.*

The temperature measurements are far less consistent than those seen in the walking experiment, even though care was taken to ensure that the robot was not moved during the experiment. The gradient of the temperature over time for many joints appeared to change sharply at particular points in time, which never occurred during the walking experiment. That being said, there are some clear examples where the robot was operating more efficiently while standing than while bent. In particular, the temperatures for LHipPitch and LKneePitch did not increase at all while standing.

It is thought that the inconsistencies in the temperatures are caused by a feature of the Aldebaran NAO called "Smart Stiffness". It is also possible that the inconsistencies are caused by a servo's failed efforts to reach a precise position, actuating constantly to no avail.

Figure 1.17: *Plots of how temperature increased over time in each of the robots for each of the walks. The 2014 walk is plotted in warm colours, whereas the heel to toe walk is plotted with cool colours.*

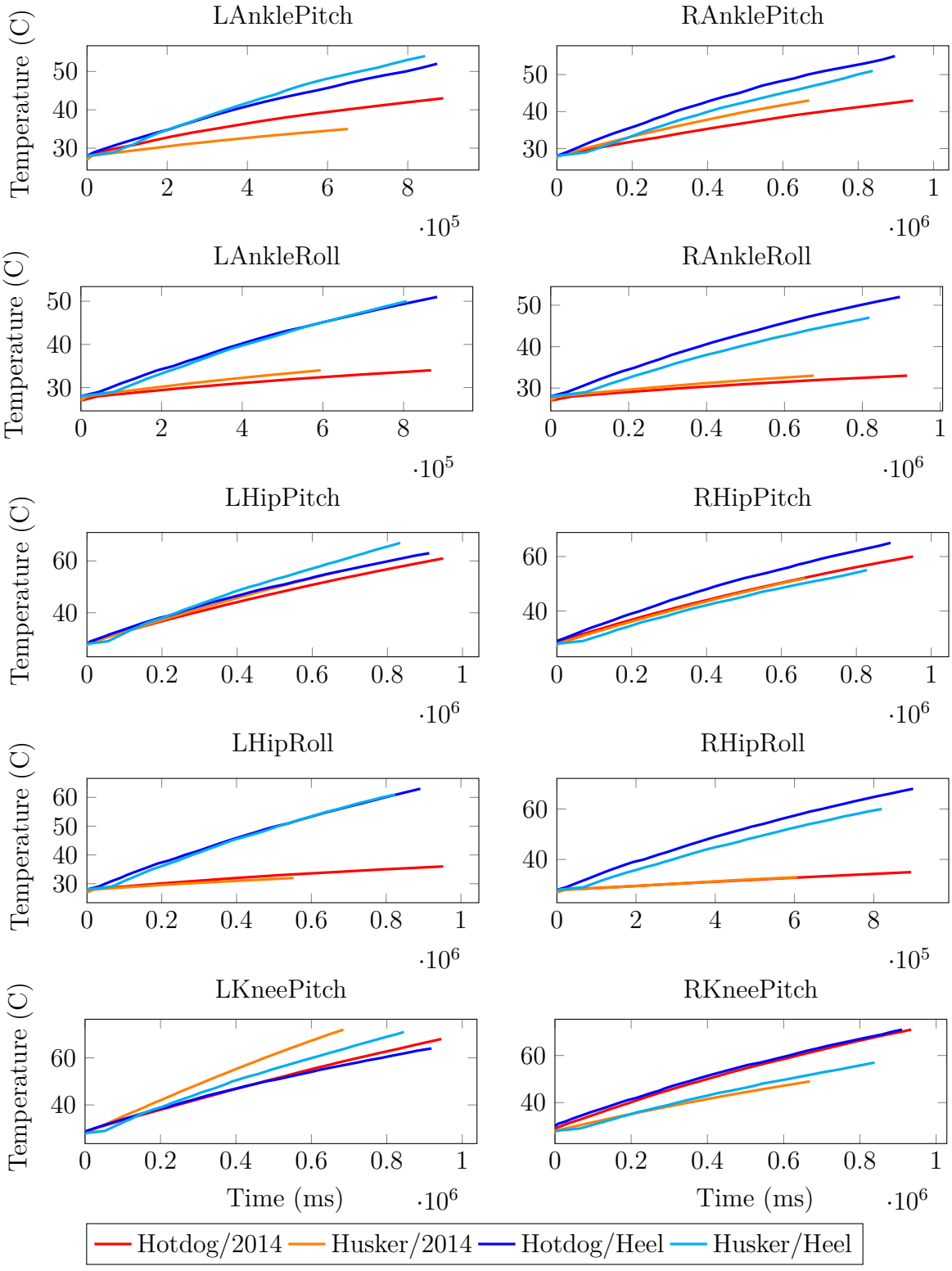
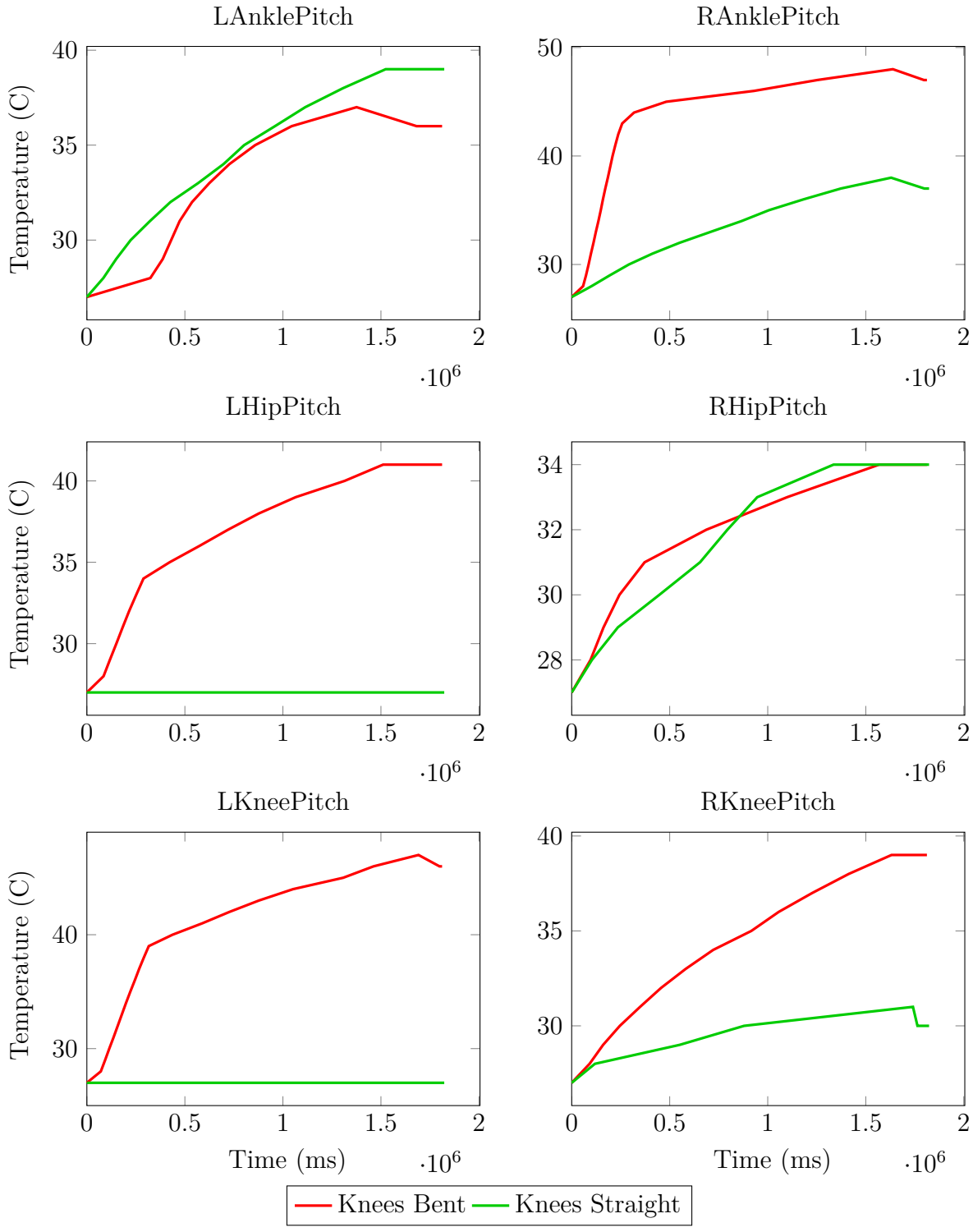


Figure 1.18: *Plots of how temperature increased over time for a particular robot in two different stances - a standing pose with straight knees, and a standing pose with knees bent at 90 degrees.*



1.5 Evaluation

From the perspective of an inverse kinematics problem, the solution presented in this report can be considered a success. This is demonstrated by the work done in the 2D simulator, which saw a heel to toe gait realised with a straight knee in the support foot, and smooth joint angles throughout. The graphs in Figures 1.11a and 1.11b display this visually. The joint angles themselves never jump suddenly from one value to another, and the joint velocities do not change by too large a degree between two given points in time.

The largest changes in velocity are found at the boundaries of the phases, where errors occurred due to rounding. The rounding occurred due to a combination of the walk engine being time-based as opposed to frame-based, and the condition that the final pose of each phase must be met exactly before continuing to the next. If the current time was just past the last allowed time for this phase, it was rounded back to be exactly the last time allowed for this phase. The final pose was calculated, then the next phase was initiated after that. This rounding caused some spikes in the velocity of the joint angles, but they generally only lasted for a single frame, and so were likely smoothed out by the servo motors as they tracked their target angle.

These graphs however depict an ideal situation where the robot is never destabilised, and each foot reaches its target on the ground at the exact intended time. It goes without saying that this almost never occurred in reality. The mechanism designed to detect early swing contact and prevent destabilisation broke assumptions made by the inverse kinematics system and reintroduced problems with the kinematics singularity in the knee. It did this by forcing the robot into the `rock` phase before the swing foot's leg had a chance to straighten its knee and reach the desired z-coordinate. Whenever the swing foot hit the ground particularly prematurely, having a significant bend at the knee, an unsightly jerking movement shortly followed. This was the result of an untamed singularity in the knee, which created a very large velocity in the knee joint. Specifically, the control variables $(x_h, z_h, \beta = 0)$ were used in the rock phase to interpolate the front foot, which broke down when the knee was not in the straight position at the beginning of the phase. Although a solution to this problem was formulated, time constraints did not permit its implementation and testing.

Aside from this undesirable property of the early swing contact detection mechanism, it was responsible for the first breakthrough in stabilisation of early versions of the walk, allowing it to take its first steps. In this respect, the technique was deemed a success, being the walk's key source of feedback regarding the environment around it. At first glance it appears the mechanism simply skips the engine into the next phase, but this in turn requires the movement speed and stride length to be adjusted, which changes the trajectory calculated for the ZMP, which then results in the dynamics of the walk being adjusted to account for the unexpected change. The technique that this mechanism was inspired by was first displayed in the rUNSWift walk, and it also behaved similarly in that it was a very simple behaviour which solved a complex problem in the dynamics of the robot.

We now observe a situation where the robot managed to restabilise. The graph in Figure 1.15 shows the length of each step taken in a walk which becomes destabilised midway, and then recovers, resuming a normal gait. We see that initially, the steps in the right foot are smaller than the right, indicating that the robot is leaning too far to the right, preventing the robot from completing a full stride on that side. The 7th step of the right foot is very short, and as a result the sudden jerk mentioned above occurred, lurching the robot leftwards, triggering an extremely small step in

the left foot. This pattern repeats for a few steps, destabilising the robot significantly. However, the step sizes gradually increase as the robot corrects its dynamics, and resumes a stable walk thereafter.

It is apparent from looking at Figure 1.15 that the step size of the gait is highly variant, especially when the robot becomes destabilised. However, despite this variance, the robot still achieves a relatively long stride length for the speed that it travels at, of up to $100mm$ at $120mms^{-1}$. For comparison, the rUNSWift walk has a stride length of 27.6mm at the same speed. Achieving a large stride length was one of the goals of the project, with the hope that this would result in a more efficient walk. This longer stride length was achieved by the hip sway that the ZMP trajectory functions introduced, allowing the ZMP to remain in a single support foot for a longer period of time.

The rotational adjustment made in the ankles and hip was also considered a successful stability technique, but was not a necessity for locomotion. The robot could walk with the mechanism turned off, but it walked much better with it activated. It too was taken from the rUNSWift walk, but was slightly modified for use in this walk. In the rUNSWift walk, only the ankle joints were adjusted to balance the robot, but since this walk relied more heavily on each foot's angle relative to the ground, a portion of this angle adjustment was moved to the torso instead. This gave a stability mechanism which worked reasonably well alongside the heel to to gait, however its modification of the ankle joints post inverse kinematics resulted in the relative positions of the feet in the rock phase being slightly inaccurate. In particular, the heel strike of the gait was practically non-existant, with the foot arriving almost parallel to the ground with each step.

As it turns out, this may have actually been a good thing. The double support phase was expected to bring stability to the robot, since by having more contact points with the ground, there are less unstable places for the ZMP to exist. However, the way in which this double support phase was architected ruined this property somewhat. Bar the very start and finish, the only points of contact that the robot had with the ground during the `rock` phase were with the tip of the heel of the front foot, and the tip of the toe of the rear foot. As can be seen in Figure 1.19, the extremities of the feet are rounded, meaning that the amount of direct contact with the ground in the feet is extremely small throughout the entirety of the `rock` phase. This caused the robot to have an unstable start to the beginning of each `rock` phase, counteracted only by the stability techniques mentioned above. Although it might seem simple to solve this problem by having at least one foot flat against the ground at all times, this is not possible if we describe a heel to toe gait with both knees kept straight. The only options are to either bend the knees or land with a flat foot, otherwise the relative position of the feet and upright position of the torso are lost, causing slipping and/or instability. Other heel to toe projects get around this problem by engineering articulated feet, which give a more substantial amount of surface area when the heel and toe are pressed against the ground, or by using complex waist joints to gain extra degrees of freedom, making two flat feet possible. For example, the WABIAN-2R seen in Figure 1.3 takes advantage of both these techniques.

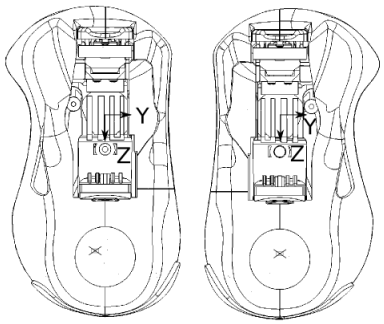


Figure 1.19: *A top-down view of the shape of the feet of the NAO.*

The `lift` phase however was a good example of a double support phase. The front foot was kept flat against the ground, and had the ZMP kept within it. The rear foot's toe was kept in contact

with the ground, but the leg's centre of mass was still moved forward by bending the knee. This allowed the robot to have two points of contact while still initiating the forwards motion of the rear leg, ready to become the swing foot in the next phase. Instabilities that occurred in the `rock` phase appeared to be somewhat rectified by the `lift` phase, making for a successful `swing` phase.

The ZMP trajectory function derived in this project provided a great improvement on the stability of the robot throughout the walk, to the point where the walk could not function without it. As mentioned above, the hip sway in the left and right directions allowed the swing foot enough time to make a long stride. Without it, the walk would have to take much smaller, shuffling steps, which would not allow us to fully realise a heel to toe walk. The use of a ZMP trajectory along the x-axis was less vital, since it was very close to a linear trajectory, but at low speeds it provided a hesitation over the support foot which was required to maintain balance.

Figure 1.12 demonstrates how the ZMP trajectory function performed on the NAO in a forwards heel to toe walk, using readings from the kinematics system developed by rUNSWift in previous years. The target ZMP is defined as the position just beneath the ankle of the support foot, which changes halfway through the `rock` phase. We can see that the ZMP along the x-axis appeared to very roughly follow its target, but often oscillated about the centre of mass instead. The reason for this is thought to be the rotational adjustment, which adjusts both the ankle and hip joints in an attempt to stabilise the robot. It has the effect of modifying the centre of mass after the ZMP trajectory function has set it, undoing its work and breaking its target. However, without this rotational adjustment enabled, the robot displayed greater instability, which is why it was used in the final product. It is possible that in attempting to move the ZMP below the ankle of the support foot, problems occur when the foot has only its heel or toe placed upon the ground, since the robot will tend to tilt forward or backwards respectively. Changing the ZMP target to be in between the two feet during the `rock` phase may have repaired this issue somewhat, but there was not enough time to pursue this avenue.

The ZMP trajectory told a more positive story in the y-axis, which was free from post-calculation adjustments. The ZMP of the robot can be seen to quickly snap from one foot to another in a fraction of a second. The centre of mass and ZMP lines appear to lag behind their target somewhat, but this is expected since the system optimistically assumed that each step would be completed in full, whereas in reality it was cut short the majority of the time. Note that the ZMP tends to fall short of its target of ± 50 each time. It is thought that this is largely due to the assumption that the centre of the torso will be the position of the centre of mass of the robot. In reality the legs of the robot contain much of its mass, and must be considered when positioning the centre of mass accurately. Additionally, the kinematics system of the robot uses measurements from the inertial unit to approximate the tilt of the robot, whereas the heel to toe walk does not. Without incorporating some kind of state estimation, the planned centre of mass will further deviate from where it was intended.

One of the largest assumptions of the Cart-Table model, that of a constant centre of mass height, was tested in order to see how inaccurate it was. This was done by recording the height of the centre of mass reported by the kinematics system, seen in Figure 1.14. Put simply, the change in height of the centre of mass was very much negligible, despite the fact that the walk made no attempt to maintain a constant torso height. This supports the potential accuracy of the ZMP trajectory functions derived in this report if more care is taken to accurately follow the paths they enscribe.

The efficiency of the rUNSWift walk and the heel to toe walk were compared by running the two

walks at identical speeds and observing the rate of increase of the temperatures of each joint. It was determined that the rUNSWift walk was more efficient that the heel to toe walk by something in the order of 30-40%.

This was chiefly due to the fact that the heel to toe walk actively used more motors than the rUNSWift walk for forward movement. Namely it used the AnkleRoll and HipRoll motors, which were responsible for controlling the hip sway of the robot, and not used at all in the forward rUNSWift walk. As a result these motors in the heel to toe walk used up to 6 times the energy of the same motors in the rUNSWift walk.

The AnklePitch joints also expended a significantly higher amount of energy in the heel to toe walk than in the rUNSWift walk, on average about 75% more. This is likely due to the fact that the heel to toe walk was far less stable than the rUNSWift walk, particularly in the forward and backward directions, meaning that the ankles had to do more work to counteract these instabilities. Also, the feet spend a non-trivial amount of time on their toes during the heel to toe walk, and if there is a load bearing down on the toe then a large amount of torque is required at the ankle to support this.

HipPitch and KneePitch motors used comparable amounts of energy in both walks, with the heel to toe walk having a very slight edge over the rUNSWift work in the KneePitch joints. Given that the knees of the heel to toe walk were subjected to much sharp movement, it was possible that the knees may consume more energy than the rUNSWift walk. This points towards the fact that when the knees were kept straight in the heel to toe walk, they consumed less energy, making up for the sudden jerks and levelling out the energy consumption between the walks.

In order to confirm this theory, a supplementary experiment was performed where a robot was allowed to stand still with straight knees and then again with bent knees, with temperatures being recorded each time. The conclusion was that the straight knee pose was far more efficient, but strangely the temperature data was far less consistent than that recorded during the walk comparison, even though other variables like duration of the experiment were similar. Figure 1.18 shows some examples of the inconsistent data, showing that a single joint's energy consumption changed sharply at various points through time despite never moving.

The explanation given to this observation was a feature of the NAO called "Smart Stiffness". "Stiffness" is a property that can be defined for each servo motor in the NAO, able to be changed at a rate of 100Hz. It corresponds to the maximum torque that each motor is permitted to output. A value of 0 means the motor is essentially off, and a value of 1 means that the motor is working at full capacity. The Smart Stiffness feature essentially attempts to detect when the robot is in a static state, and automatically turns down the stiffness in each motor according to how much torque is required to maintain the current pose. This feature would not trigger during a walk since the robot is constantly in a dynamic state, but during this test the joint stiffnesses will have been set to whatever the Smart Stiffness feature saw fit.

In the ideal case this means that the stiffnesses of unnecessary motors are set to 0, as was seen with the LHipPitch and LKneePitch joints in the experiment, where these motors saw no temperature increase. However if the stiffness setting was set very low, and the servo motor was a very small distance away from its target, its possible that the motor would continually attempt to actuate while never being able to overcome its internal friction reach its goal. It is believed that this is why some joints in symmetric positions in the robot saw highly contrasting temperature increases (see LHipPitch vs RHipPitch).

As mentioned before, a 3 minute presentation and demonstration on this project was made at the

RoboCup 2014 SPL Open Challenge, achieving a 2nd place ranking amongst 11 projects. Despite the failure to improve on the efficiency of a typical bipedal walk, this result demonstrates the merit of the research presented in this report, and the relevance of the project to peers in a global context.

1.6 Future Work

As with any piece of research, there is always more to be done. Here is a collection of thoughts on future directions that one could take if they wished to go down a similar avenue of research.

1.6.1 Knee-stretch Walk

Experiments confirmed that keeping the knee straight required far less energy under load than to keep the knee bent. Although this project aimed to take advantage of this through a heel to toe walk, issues were created because the feet of the robot were clearly not designed to bear weight on just the heel or the toe. Other gaits are possible which still keep the knee joints straight while having at least one foot flat against the ground at all times. This could be achieved by having the swing foot land flat against the ground instead of with the heel. Taking this approach may better realise the efficiency benefits of a gait with straight knees.

1.6.2 Cleaner Inverse Kinematics

The control variable transformations derived in this paper successfully circumvented the singularity problems encountered in the uninterrupted heel to toe gait. However, some control variable formulations unnecessarily constrained a degree of freedom, notably in the case of $(x_h, z_h, \beta = 0)$. This was what caused the early heel contact mechanism to have an extremely unsmooth motion. Had the β variable been freed, this would not have been an issue. Freeing this variable is not difficult either, as it has a very similar derivation to that of (x_t, z_t, β). Time simply ran out, and it would be a relatively straight forward task to reformulate the control variable selections which removed degrees of freedom in a way that restores their freedom. This would make for a more versatile inverse kinematics system.

1.6.3 More Precise Centre of Mass Planning

In this report, the assumption was made that the position of the torso approximated the position of the centre of mass. This turned out to be an assumption that was only just passable, which prevented the dynamic balance system from realising its potential. Getting around this assumption however is not a trivial task. Some achieve it by using iterative methods, repeatedly adjusting a pose in a way that moves the centre of mass closer to where it should be until it is close enough [10]. Others prefer to use a servo controller to aim the current measurement of the centre of mass along its intended path [12]. And some have used machine learning to determine the motion required to achieve a centre of mass trajectory [4]. No doubt there are other techniques waiting to be uncovered also. Whatever method is attempted, it would be an interesting research topic and would have clear applications to anything involving the dynamic movement of a biped.

1.6.4 Smarter Stiffness

Aldebaran have shown with a degree of success that energy savings can be made in the static state by adjusting the stiffness or maximum torque settings of the servo motors. It is possible that

something like this might be developed for the dynamic state also. In the swing phase of a typical walk, the exact position of the foot is generally non essential. It is possible then that the servo motors in the leg are doing unnecessary work to hit target angles with unneeded accuracy. By adjusting the stiffnesses automatically to appropiate values, the robot might be able to focus its energy in the places it counts while relaxing in the areas that are not vital, resulting in a more efficient locomotion.

1.6.5 A New Way To Move

Passive-dynamic walkers are extremely efficient, and servo-based bipeds give great control. Although a relatively ambitious idea, trying to create an actuator which can harness the power of both would be a very interesting topic of research. It is hypothesised that if an actuator with a very low amount of friction at the joint could be created, the inertial and gravitational motions of a passive-dynamic walker could be taken advantage of. Furthermore, if the actuator still delivered the control of a servo motor when required, the robot could be made much more versatile than something which can only walk forwards. We as humans display both efficiency and control in our movements, and the development of such an actuator is a necessary step towards these benefits.

1.7 Conclusions

In this report we aimed to create a bipedal walk which was more efficient than those typically exhibited in robotics, characterised by the rUNSWift walk. The approach taken was to emulate nature's work on the human gait, designing the gait around a heel to toe motion. Certain aspects of this gait were identified as being key to its efficiency. Having a straightened knee in the support leg was a major goal and dictated the nature of the gait to a large degree. The other key goals were to have smooth moving joints, to walk with a long stride length, and to stabilise the walk for use in a physical robot. Each of these goals provided their own challenges, and each of these were largely overcome.

Keeping a straight knee throughout the walk posed a problem in terms of a kinematics singularity that caused issues when a given leg was near full extension. A solution to this common problem is presented, whereby we transform the control variables of the inverse kinematics in such a way that the singularity is no longer an issue. Due to time constraints, there was a small singularity which occasionally created jerkiness in the final walk, but the solution to this problem was known and uses techniques documented in this report.

The joint movements were made smooth by crafting polynomial based interpolation functions which allowed enough expressibility to make them so. These functions also had to be fit to appropriate control variable transformations in inverse kinematics to achieve the desired smoothness. The same singularity issue above created some unsmoothness occasionally, but again, the solution is known and documented.

The walk moved with a large stride length, made possible by the stability techniques developed. A hip sway allowed the ZMP of the system to remain in a single foot for a longer period of time, allowing a longer step to be taken.

A simplified model for the dynamics of the robot was used to create a closed-form ZMP trajectory function, which could plan a safe motion for the centre of mass from one point to another, keeping the ZMP at a target position throughout. Assumptions made in controlling the centre of mass meant that the planned trajectories were not accurately realised, but despite this the technique still provided a stability gain in the robot.

The final heel to toe gait was able to operate on an Aldebaran NAO Humanoid Robot, with the help of some stability techniques inspired by the rUNSWift walk. The robot was able to walk forwards for a very large amount of steps, with a fall only happening every so often.

The efficiency of the heel to toe gait was compared against that of the rUNSWift walk. It was concluded that the heel to toe gait was less efficient than the rUNSWift walk, due to a combination of many factors. The hip swap used several motors in the robot that the rUNSWift walk does not use, increasing overall motor use considerably. The ankle motors did more work in the heel to toe walk due to a greater level of instability and because weight was often applied at the toe instead of on a flat foot. Additionally, the singularity problem created large joint velocities in the knee which contributed to inefficiencies.

A supplementary experiment was performed to determine whether keeping the knees straight under load was more efficient than keeping the knees bent. It was concluded that it was more efficient to keep the knees straight, but care must be taken to ensure that the servo motors do not unnecessarily apply torque to chase unnecessarily accurate goals.

The heel to toe walk was presented at the RoboCup 2014 SPL Open Challenge, where it placed 2nd amongst 11 research projects from teams around the globe. This can be observed as a form of peer review, which demonstrated the merit and relevance of the work presented in this thesis.

So, if researchers in the field of passive-dynamic walkers found that emulating a human-like gait was key to optimising efficiency, why were efficiency benefits not realised in this experiment? In part it is due to an imperfect execution of the concept, but even if the above mentioned problems are resolved, it is believed that emulating a human gait on a servo-based robot has limited returns in the way of efficiency. There are some aspects of the gait that may be useful, for example the straightening of the knee may be made to work if the rest of the gait is kept more conventional. But at the core of the issue is that servo-based bipeds and passive-dynamic walkers have competely different morphologies. Passive-dynamic walkers are efficient because they allow gravity and inertia to drive their movements, and it is believed that the human walk is efficient for similar reasons. Servo-based bipeds such as the NAO are unable to take advantage of these benefits by design - the motor joints possess far too much friction to allow inertia or gravity to drive them. As a result, it is concluded in this report that if we are to create efficient bipeds by emulating nature, we should begin by emulating nature's morphology first. As for making servo-based bipeds more efficient, we conclude that emulating nature's motions will not necessarily provide nature's benefits.

Chapter 2

Striker Behaviour

2.1 Introduction

The rUNSWift code has several levels of abstraction in order to modularise the code base, allowing parallel development in multiple areas without significant interference. The behaviour module sits atop the stack, pulling together the information from the localisation module, vision module and other teammates, combining it to form a plan of action. This plan of action is then delivered to the motion module, allowing the plan to be realised.

The behaviour module is divided into several roles, the most involved of which is the Striker. The Striker is the only player apart from the Goalie who interacts with the ball, and is responsible in short for getting the ball into the goal. In order to be effective, the Striker must approach, line up, and shoot the ball in as efficient a manner as possible. It must also take into consideration its position on the field, its teammates and its opponents when deciding what the best course of action is.

This report describes a few of the key aspects of the striker which made it a dangerous adversary in the RoboCup 2014 SPL league in Brazil, where the rUNSWift team dominated the competition and took home the 1st place trophy.

2.2 Methodology

2.2.1 Hystereses Over Decisions

In a noisy and continuous environment, making decisions is difficult. A "decision" here means a choice between a finite set of options, for example, which foot to kick with, whether to aim for the goal or clear the ball, and whether to avoid an opponent or go for the ball.

In general one might define a set of conditions which, if met, would point to a particular decision. For example, in order to decide which foot to kick with, we choose the foot closer to the ball. However, this approach causes issues, since we have a hard decision boundary defined over a noisy value. Imagine that the ball is roughly equidistant from both feet. Due to noise, the decision we make on which foot to kick with may flip back and forth, potentially causing the robot to make no progress as it twitches from side to side.

A solution to this problem was to define a Hysteresis over decisions with hard boundaries, such that the flow of logic was smoothed over time. A Hysteresis is where the decision of a system depends not only on the current value, but also historical values, stored as a state within the Hysteresis. Two types of Hysteresis were developed - one that worked temporally, and one that worked spatially.

2.2.1.1 Temporal Hysteresis

When observing a noisy decision boundary manually, made at say a rate of 100Hz, it might be clear which choice should be chosen purely based on the relative frequency of the decisions. If a condition evaluates to `true` 9 times out of 10, a rational thinker would assume that `true` is a good bet. Additionally, if the relative frequency of decisions is roughly equal, you would assume that either option is satisfactory, and would prefer to choose the decision that you are already pursuing.

This behaviour is modelled in a very simple but effective fashion. The state of the hysteresis is defined as an integer x, and a boolean t. When a condition evaluates to true, the value of x rises by a predetermined amount u, and when it evaluates to false it falls by d. An upper and lower bound on x are defined as x_u and x_d respectively. Whenever $x >= x_u$, we set $t = true$, and whenever $x <= x_d$, we set $t = false$. The Hystereses always returns t as the decision which should be made based on the condition it sits upon.

In this way, it takes a repeated amount of `true` or `false` evaluations in order to change the outcome of the decision. Whichever of the two is more frequent in a given situation will eventually dominate and hit the upper or lower bound, and this will happen faster when the relative frequency of one option to the other is higher. If there is an indecision then the existing condition will not be changed. The degree to which decisions are made smooth can be changed by adjusting the range between x_u and x_d, and a bias towards one decision or the other can be made by adjusting the relative values of u and d.

2.2.1.2 Spatial Hysteresis

Often decisions are made based on the range in which a continuous value lies within, for example the distance of the robot from the centre of the goal. We might wish to change the behaviour of the robot when is within shooting range of the goal compared to when it is not. Or in more complex

situations, we may want to have multiple decision boundaries to distinguish between being close to the goal, far from the goal, and out of range from the goal.

Although a temporal hysteresis is adequate for binary decisions, it is not entirely appropriate for multiway decisions[1]. There is an easy way to form a Hysteresis over continuous values however, which is quicker to respond and handles multiway decisions elegantly.

Let's say that we have a number of (ordered) decision boundaries $x_1 ... x_n$. The state of the Hysteresis is defined by a single integer b, which defines which decision boundary we currently lie ahead of. i.e., a value of $b = 2$ means that we are reporting to be between values x_2 and x_3. We also define a threshold value t which increases the size of the current decision boundary. Whenever we receive a new spatial value s and report the region in space we will take to be relevant, we first check the boundary we are currently within. If the new value s lies between the extended bounds of our region $x_b - t < s < x_{b+1} + t$, then we continue reporting the same value b. Otherwise, we determine between what values of x our new value s lies, and update b to be this value.

Increasing the size of t makes decisions more smooth. t should be set to the maximum expected deviation due to noise from the true value of the spatial variable, which will stop noise from affecting decisions.

2.2.2 Strategy Decision Structure

Our Striker strategy took into account many different aspects of the environment in order to determine what course of action was best. This could easily turn into a list of undecipherable if statements, so care was taken to structure the decision hierarchy in such a way that it would be clear and maintainable, as well as flexible.

The Striker's intention was boiled down to the following variables, which were set by the strategy component. The rest of the Striker code read from these variables in order to determine what to do.

- `kick_mode`: This was one of `KICK`, `DRIBBLE`, or `TURN_DRIBBLE`
- `kick_target`: The global (x, y) coordinates of where the ball needs to end up
- `kick_hard`: A flag determining whether we should kick as hard as we can in the direction of the `kick_target`, or try to get the ball to land at the `kick_target`

The strategy code therefore just needs to come up with values for these variables somehow. This decision making process was divided into two parts: the high-level target, and the target adjustments. The high-level target was an expression of intent without taking into consideration other opponents on the field. Target adjustments took into account the robot's immediate surroundings, and adapted the target in order to better achieve its goal.

2.2.2.1 High-Level Target

The following outcomes were decided between using a variety of decision variables, most having been passed through Hystereses. The outcomes were prioritised in the following order, with the first decision evaluating to true being executed.

[1] That didn't stop us from using it that way, though.

- **Penalty Kick**: It's a penalty shootout. Aim to the edge of the goals, either to the left or right.
- **Kick-off play**: It's a kick-off. Determine whose kick-off it is. If it's our kick-off, wait 10 seconds and get the ball into the position that the Supporter role is currently moving to.
- **Corner Play**: The angle to the goal is too sharp, because we're in one of the opponent's corners. Attempt to place the ball near the penalty spot.
- **Pass upfield**: We're too far from the goal to reliably shoot. Try to place the ball in the direction of the goal but a safe distance back, so that it doesn't go out.
- **Goal center**: We're within range, but we're too far to do anything fancy. Shoot for the centre of goal as hard as possible.
- **Goal lazy angle**: We're well within range, to the point where there's no need to turn all the way to face the centre of the goal. We calculate a position to shoot for which requires minimal change of heading while still getting the ball in. This might be straight ahead as we are, or it might mean aiming a little bit towards the centre to clear the left or right post.

Each of these high-level goals decided whether they allowed target adjustments on their instructions or not. For example, if we were very close to the goal, target adjustments were deactivated.

2.2.2.2 Target Adjustments

Whether to apply an adjustment or not was decided upon through conditions which were passed through Hystereses. They were evaluated in the following order, but sometimes multiple adjustments could occur on the same target.

- **Close Quarters Combat**: An opponent has been detected near the ball. Forget trying to shoot to the goal, get in front of their shot and `TURN_DRIBBLE` the ball away from their feet.
- **Ronaldo**: There's an opponent occluding our shot to target. Adjust our aim to avoid the opponent, and change a `KICK` to a `DRIBBLE` instead if our adjusted heading is no longer aimed at the goals.
- **Ball Contended**: There are opponents near the ball. Change a `KICK` to a `DRIBBLE` instead

2.2.3 Efficient Ball Approach

The existing Striker behaviour approached the ball by first walking directly towards it. When it arrived within a particular range of the ball, it decided whether it needed to perform a circle strafe around the ball. Once the robot was satisfied with its heading relative to the ball, it moved left/right and forward/back to align the kick foot with the ball. Once this was aligned, a kick was initiated.

This year's Striker aimed to improve upon the efficiency of this method, taking advantage of the upgraded walk engine's improved agility and control.

Instead of approaching the ball directly, the position in which we would like to finally kick the ball was calculated and aimed for. That is, the position which puts the kick foot directly in front of the ball to begin with. This meant that in ideal situations, the robot was able to just walk up to the ball and kick it, spending almost no time stopping and lining up the ball.

If the robot needed to walk around the ball to kick it in a particular direction, it drew a circle around the ball and approached the tangent to this circle which was nearest to the desired kick position. Once the robot was close to the ball, it engaged in a motion which attempted to move around the ball and rotate to face the correct direction at the same time, in combinations which gave optimal progress per second.

In order to achieve this, a vector representing the distance and direction to travel around the ball in an arc at this point in time was calculated, and the total amount of rotation required to face the correct direction was also calculated. These x, y, and θ values were thrown into a three dimensional vector, which had a maximum magnitude defined for each dimension. The vector was scaled back such that all dimensions were less than or equal to their limits. The robot then moved in the direction of this vector, turning by theta at the same time.

The effect of this calculation was that if a large distance was yet to be travelled in order to get to the target kick position, the turning component diminished, and the robot powered forwards to its position quickly. As it neared its final location, the turning component became more significant, and the robot quickly turned to face the ball as it made its last few steps to get into position. This resulted in a very efficient ball approach, even if we were required to walk all the way around the ball to get to our kick position.

2.2.4 Dribbling the ball

Because lining up the ball could still take a long time at the best of times, a dribble was developed to be able to boot the ball inaccurately but without hesitation.

The dribble worked by simply calculating the vector from the centre of the intended contact toe to the ball, and moving in this direction. If the position of the ball was accurate, this resulted in the ball moving away from the position of the toe of the robot. Therefore, to line up a dribble, all that had to be done was to get one of the toes of the robot behind the ball in the correct direction, then trigger the dribble behaviour. While walking around the ball, this condition was met well before the robot arrived at the intended kick position, and resulted in a very quick attack on the ball from a difficult angle.

This simple behaviour resulted in an omni-directional dribble which was activated in a wide variety of situations, some even unexpected, which not only got the ball moving quickly but got the robot moving towards the ball's destination very quickly too.

2.2.5 Kicking an accurate distance

For several of the Striker's high level goals, the robot is required to adjust the power of its kick to place the ball at a set of co-ordinates, instead of the usual policy of "boot the living daylights out of it". This was especially important in the **Pass Upfield** target, which was in danger of kicking the ball out if kicked too hard.

The power of the kick was tweaked by a "power" constant which adjusted the speed of the movement of the foot to make contact with the ball in a slower or faster way. From there the ball would begin rolling at a different inital speed, and eventually come to a stop due to rolling friction.

In order to better understand the relationship between this constant and the distance that the ball travelled, an experiment was performed where the robot repeatedly kicked a ball at a variety of different power settings, the distance that the ball travelled being measured each time. 5 different power settings were tried and repeated 3 times each. The distances for each of these settings were averaged, and the relationship between power and distance had a straight line fit to it by the least square error method.

When we wished to kick the ball a particular distance away, the line of best fit was used to calculate the best power to achieve it. In this way we could confidently kick the ball to a desired distance, with an error of about $\pm 1m$.

Event	Value (per Hour)	Percent Rank
Goal	20.77	100%
Ball Out	8.90	73.7%
Illegal Defender	5.44	100%
Leaving the Field	5.93	10.5%
Request for Pickup	17.80	94.7%
Pushing	5.44	47.4%
Goals/Outs Ratio	2.33	84.2%

Figure 2.1: *Statistics were collected via the logs on all the games played in the tournament. These statistics were normalised by dividing each by play-time, and rUNSWift's results are displayed here. The percentage rank of our score relative to others is given to give the figures context.*

2.3 Results and Evaluation

2.3.1 Game Statistics

Statistics taken from the game controller logs in competition are shown in Figure 2.1. As can be seen, we scored the most goals per minute, and managed to stay on the field more than about 90% of other teams.

We kicked the ball out a considerable amount, however this is believed to be due to missed shots on goal. If we take a look at the ratio between our goals scored per hour and outs per hour, we ranked in the top 15% of all teams.

Unsurprisingly, we had the most Illegal Defender calls per hour, which was due to an incessant bug which repeatedly caused the Striker to believe it was allowed to enter the centre circle before the opponent had a chance to get to the ball, which was false. This bug was fixed in the later stages of the tournament.

We were one of the teams who most frequently requested to pickup our robots, which is a testament to their poor condition and to the crazy bugs dwelling in our code.

We had a below average figure for pushing calls, which is excellent considering that we were also very aggressive on the ball.

2.3.2 Kick Distance

Figure 2.3.2 shows the data used in the kick distance experiment, used to determine the relationship between the kick power constant and the actual distance of the ball. As can be seen, the data was very consistent, and matched the straight line reasonably well. It may have fit a line of higher order more closely, but it was decided that it was best not to overfit the relationship.

2.3.3 Competition Result

We came first overall in the competition, scoring 5:0 against MRL in the quarter final, 5:0 against B-Human in the semi-final, and 5:1 against HTWK in the finals. This incredible goal difference in

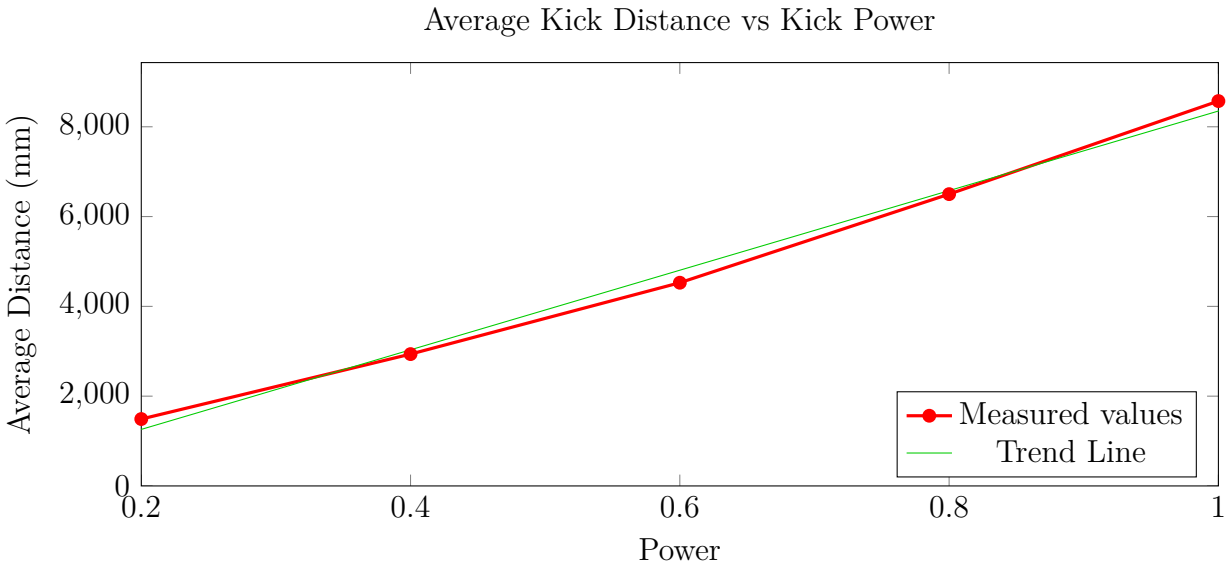


Figure 2.2: *A graph showing the relationship between the power constant and the distance of the ball.*

even the most difficult games in the competition is proof that this year's Striker behaviour, and the rest of the team, did an excellent job.

2.4 Future Work

2.4.1 Passing Upfield to a Teammate

At present, the Striker strategy does not take into account the position of teammates except for in the kick-off play. This means that when the Striker passes upfield, it just gets it near the goal and hopes that a teammate will be there to take it. It would be better to either signal to the teammate ahead of time that we're planning to put the ball in a particular position near the goal, or pick a position based on where existing teammates are already.

2.4.2 Teammate Strategies

The current Striker is quite the ball hog. There are situations in the game where passing to another teammate who has a clearer shot on goal is desirable. Passing to a teammate could be considered an option amongst taking a shot on goal, and a measure to determine which option is best could be formulated.

2.5 Conclusions

rUNSWift's impressive performance at the RoboCup 2014 SPL tournament is a testament to the Striker behaviour's success, but of it would be nothing without all the excellent modules beneath it, providing the clear, reliable and reactive data which was necessary to strategise and play.

Chapter 3

References

1. LIEBERMAN, Daniel E and BRAMBLE, Dennis M. The evolution of marathon running-capabilities in humans. *Sports Medicine*. 2007. Vol. 37, no. 4-5p. 288–290.

2. XINJILEFU, X, FENG, Siyuan, HUANG, Weiwei and ATKESON, Christopher G. Decoupled state estimation for humanoids using full-body dynamics. In : *Robotics and automation (iCRA), 2014 iEEE international conference on*. IEEE, 2014. p. 195–201.

3. CHO, Baek-Kyu, KIM, Jung-Hoon and OH, Jun-Ho. Online balance controllers for a hopping and running humanoid robot. *Advanced Robotics*. 2011. Vol. 25, no. 9-10p. 1209–1225.

4. COLLINS, Steve, RUINA, Andy, TEDRAKE, Russ and WISSE, Martijn. Efficient bipedal robots based on passive-dynamtteic walkers. *Science*. 2005. Vol. 307, no. 5712p. 1082–1085.

5. HENGST, Bernhard. rUNSWift walk2014 report. In : *RoboCup 2014 standard platform league*. 2014.

6. DONELAN, Peter. Kinematic singularities of robot manipulators. *Advances in Robot Manipulators. 2010. P.* 2010. P. 401–416.

7. ZARRUGH, MY and RADCLIFFE, CW. Predicting metabolic cost of level walking. *European Journal of Applied Physiology and Occupational Physiology*. 1978. Vol. 38, no. 3p. 215–223.

8. MCGEER, Tad. Passive dynamic walking. *the international journal of robotics research*. 1990. Vol. 9, no. 2p. 62–82.

9. MCGEER, Tad. Passive walking with knees. In : *Robotics and automation, 1990. proceedings., 1990 iEEE international conference on*. IEEE, 1990. p. 1640–1645.

10. GRAF, Colin and RÖFER, Thomas. A center of mass observing 3D-lIPM gait for the roboCup standard platform league humanoid. In : *RoboCup 2011: robot soccer world cup xV*. Springer, 2012. p. 102–113.

11. KAJITA, Shuuji, KANEHIRO, Fumio, KANEKO, Kenji, YOKOI, Kazuhito and HIRUKAWA, Hirohisa. The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation. In : *Intelligent robots and systems, 2001. proceedings. 2001 iEEE/rSJ international conference on*. IEEE, 2001. p. 239–246.

12. KAJITA, Shuuji, KANEHIRO, Fumio, KANEKO, Kenji, FUJIWARA, Kiyoshi, HARADA, Kensuke, YOKOI, Kazuhito and HIRUKAWA, Hirohisa. Biped walking pattern generation by

using preview control of zero-moment point. In : *Robotics and automation, 2003. proceedings. iCRA'03. iEEE international conference on*. IEEE, 2003. p. 1620–1626.

13. GODDARD, Ralph E, ZHENG, Yuan-Fang and HEMAMI, Hooshang. Control of the heel-off to toe-off motion of a dynamic biped gait. *Systems, Man and Cybernetics, IEEE Transactions on*. 1992. Vol. 22, no. 1p. 92–102.

14. CZERNIECKI, Joseph M. Foot and ankle biomechanics in walking and running: a review. *American journal of physical medicine & rehabilitation*. 1988. Vol. 67, no. 6p. 246–252.

15. ENDO, Gen, MORIMOTO, Jun, NAKANISHI, Jun and CHENG, Gordon. An empirical exploration of a neural oscillator for biped locomotion control. In : *Robotics and automation, 2004. proceedings. iCRA'04. 2004 iEEE international conference on*. IEEE, 2004. p. 3036–3042.

16. CHANNON, PH, HOPKINS, SH and PHAM, DT. Derivation of optimal walking motions for a bipedal walking robot. *Robotica*. 1992. Vol. 10, no. 02p. 165–172.

17. *NAO software 1.14.5 documentation*. Aldebaran, 2014. http://doc.aldebaran.com/1-14/family/index.html

18. COMMITTEE, RoboCup Technical. *RoboCup standard platform league (nAO) technical challenges*. 2014.

19. LUKE TSEKOURAS, Zijie Mei (Jacky), Jaiden Ashmore and SAMMUT, Claude. Team rUNSWift university of new south wales, australia. In : *RoboCup 2014 standard platform league*. 2014.