Humanoid Robot Soccer Locomotion and Kick Dynamics

Open Loop Walking, Kicking and Morphing into Special Motions on the Nao Robot

Alexander Buckley
B.Eng (Computer) (Hons.)

In Partial Fulfillment of the Requirements for the Degree
Master of Science

Presented to the Faculty of Science and Engineering
of the National University of Ireland Maynooth
in June 2013

Department Head: Prof. Douglas Leith
Supervisor: Prof. Richard Middleton

Acknowledgements

This work would not have been possible without the guidance, help and most of all the patience of my supervisor, Rick Middleton.

I would like to thank the members of the RoboEireann team, and those I have worked with at the Hanilton institute. Their contribution to my work has been invaluable, and they have helped make my time here an enjoyable experience.

Finally, RoboCup itself has been an exceptional experience. It has provided an environment that is both challenging and rewarding, allowing me the chance to fulfill one of my earliest goals in life, to work intimately with robots.

Abstract

Striker speed and accuracy in the RoboCup (SPL) international robot soccer league is becoming increasingly important as the level of play rises. Competition around the ball is now decided in a matter of seconds. Therefore, eliminating any wasted actions or motions is crucial when attempting to kick the ball.

It is common to see a discontinuity between walking and kicking where a robot will return to an initial pose in preparation for the kick action. In this thesis we explore the removal of this behaviour by developing a transition gait that morphs the walk directly into the kick back swing pose. The solution presented here is targeted towards the use of the Aldebaran walk for the Nao robot.

The solution we develop involves the design of a central pattern generator to allow for controlled steps with realtime accuracy, and a phase locked loop method to synchronise with the Aldebaran walk so that precise step length control can be activated when required. An open loop trajectory mapping approach is taken to the walk that is stabilized statically through the use of a phase varying joint holding torque technique. We also examine the basic princples of open loop walking, focussing on the commonly overlooked frontal plane motion.

The act of kicking itself is explored both analytically and empirically, and solutions are provided that are versatile and powerful. Included as an appendix, the broader matter of striker behaviour (process of goal scoring) is reviewed and we present a velocity control algorithm that is very accurate and efficient in terms of speed of execution.

Contents

List of Figures

List of Tables

1. Introduction

The motivation for this project has come from participation in the RoboCup Soccer Standard Platform league [1]. In this competition, common practice for kicking the ball has been to first line up the kick, then cease the walking motion, and then execute a kick. So far, the main alternatives to this appear to be dribbling actions. While tactically useful, particularly when the ball is close to opponents, dribbling lacks the desired range and accuracy in open play. The objective of this project has been to develop a practical system which allows the Nao robot to make a direct transition from walking into kicking, 'Instride Kicking'.

We would like to be able to walk into a kick, but the Aldebaran walk is a black box system, and has a delay in the control system (command input to command actualization – velocity changes and stopping). There is a 'kill walk hard' command but determining a specific point in the walk cycle is difficult to do using kinematics alone. The solution, then, involves developing a method to accurately determine the walk cycle phase, and development of a walk gait generator that can be activated to take some steps that are realized without delay to charge the ball for a kick without stopping.

In this thesis, we choose to use the Aldebaran walk engine. This choice was made for several reasons. First, the Aldebaran walk achieves a reasonable compromise between stability and moderate speed. Secondly, we wish to benefit from any improvements made by Aldebaran in their walking system over time. In addition, prior to 2011 this was the walk engine chosen by the RoboEireann team in the RoboCup SPL league. This also allows us to isolate the problem of forward walking without having to compensate for high speed sharp turns. The Aldebaran walk can be used to compete for the ball against the opposing team, and then the custom walk can be activated to charge towards the ball when a shot oppourtunity arises.

While developling this walk, the fundamentals of open loop walking are explored and a paramterisation of the frontal plane motion and joint stiffness is presented to improve speed and stability.

To summarize, a walking system was developed with the following motivations in mind:

- Circumvent the black box limitations of stepping out of the Aldebaran walk into a kick (no access to data such as walk cycle phase, step length/height and other such walk paramters).
- Gait that is very close to the Aldebaran walk to allow for bumpless transfer between walks.
- Parametrized in a way that allows for further development; expansion to closed loop or gait modification in a way that is superior to Aldebaran's design.

To achieve these goals the Nao robot was kinematically modelled in MATLAB and most design work was done in simulation prior to moving to the hardware platform. To begin, a study of the Aldebaran walk was performed to understand its design so it could be duplicated. From there some improvements were made to the parameterization and stability so that final stage targeting steps could be performed at higher speed than Aldebaran's (charging the ball).

These two walks are then stitched together through the use of a Phase Locked Loop and parameter detection Bumpless Transfer System. With this accomplished it is then possible to design a dynamic kicking gait that is merged with the walking gait.

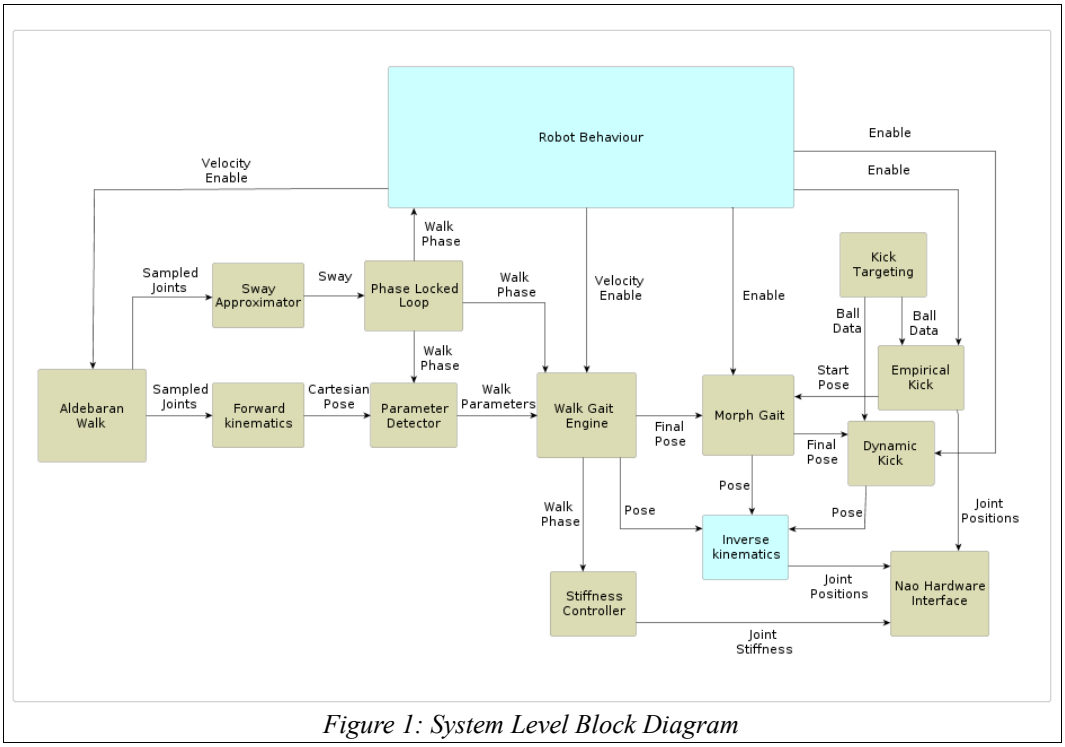


Figure 1: System Level Block Diagram

Figure 1 presents a system level block diagram for this design. The blocks marked in blue are the input and output components to the system that are included as appendices.

1.1. Contributions of Thesis

1. **Frontal plane parametrization of an open loop walk.**
2. **Phase varying joint torque control signal to provide static disturbance rejection.**
3. **A bumpless transfer method for overcoming the restrictions introduced by the delayed control response in the black box Nao walking engine to allow for walking directly into special motions or seamlessly switching between walking engines.**
4. **Parametrized kick swing for maximized shot power and dynamic targeting.**
5. **An empirical method for error reduction for an empirically defined (LUT) kick engine.**
6. **A velocity profile transfer function for use in lining up kick target points with the Aldebaran walking engine (delayed response) quickly and accurately without overstepping.**

1.2. Thesis Outline

This thesis contains 4 subject chapters. They are summarized as follows:

Chapter 2 reviews the literature pertaining to humanoid walking and kicking.

Chapter 3 focuses on the design of an open loop walk engine. It covers the topics of robot modelling and simulation, kinematics, analysis of the Aldebaran walk engine, and the design of a custom walk engine. This chapter also includes a simple massless Zero Moment Point method, the analysis of frontal plane motion and a joint stiffness method for disturbance rejection.

Chapter 4 outlines the design of a bumpless transfer system used to switch between the Aldebaran walk engine and the walk engine designed for this thesis. The components of this bumpless transfer system are a Phase Locked Loop (PLL) and a parameter matching algorithm.

Chapter 5 covers the design of a transition (from walk to kick) gait and a parametrized dynamic kick swing.

This thesis also includes two larger appendices that serve as the input and output to this system:

Appendix L provides an exact analytical solution to the inverse kinematic problems for the Nao robot. This solution relates the robots torso and feet (six degrees of freedom) to the real world coordinate system (existing methods are torso centric).

Appendix M deals with the continued development of an empirical kick design along with a targeting system. The Appendix concludes with an analysis of the robot soccer striker behaviour that controls the approach to a kick using velocity profile transfer functions.

Introduction

2. Literature Review

2.1. Humanoid Walking

Research in the area of bipedal walking has been around for some time with the earliest design dating back to 1888 when Fallis developed the first walking bipedal toy capable of a passive stable gait [2]. Nearly a century later McGeer's research [3] developed dynamically stable passive bipeds capable of walking without any control effort. In the late 60's, Vukobratovic's [4] research introduced the Zero-Moment Point (ZMP), a dynamic property of balance widely used in modern walking control strategies. The WL-5 [5], developed in the 1972 by Kato et al. at Waseda University, was the first active bipedal robot. Research that began on simple bipedal platforms (legs only) has since evolved into some very sophisticated anthropomorphic (humanoid) robots ([6][7][8][9][10][11]). But given the recent explosion of research in this area, and the availability of smaller, cheaper humanoid robots and the common use of the Biloid kits [12] or Dynamixel motors [13] to construct one's own, walking has become a very popular project topic, work that is being taken in many different directions.

With the plethora of documentation of this work there is now a great sea of literature available. There is a lot of really good work out there which we will review, but along side this many projects focus on specific features of bipedal walking, without considering broader requirements and inherent compromises. The most common background material presented being the dynamic system equations of the inverted pendulum model. A paper by Pobil [14] takes serious issue with this matter and states 'current practice of publishing research results in robotics makes it extremely difficult not only to compare results of different approaches, but also to asses the quality of the research presented by the authors'. He goes on to point out that 'this situation is impeding solid progress in the field and jeopardizing the credibility of robotics research'. For this reason, we attempt a broad review of the issues that we have found to be important in designing a walk engine for a humanoid robot.

First of all, there is the issue of benchmarking. There exists no formal benchmark definition of an anthropomorphic robot's physical construction for all projects to then relate to each other. Behnke wrote [15] about the necessity of a benchmark and calls for one. 'Benchmarking robotics research is inherently difficult. Typically, results are reported only for a specific robotic system and a self-chosen set of tasks'. He points out competitions are the best place to compare systems. 'Such robot competitions provide a standardized test bed for different robotic systems. All participating teams are forced to operate their robots outside their lab in a different environment at a scheduled time. This makes it possible to directly compare the different approaches for robot construction and control'. The closest thing we have today is the Standard Platform League in RoboCup [16]. This issue of benchmarking is briefly touched on in the context of animation and various morphologies of animated characters by Coros [17]. Kahn [18] discusses the matter of benchmarking anthropomorphic physiology but takes the issue in the direction of human interaction, behavioural psychology and social philosophy. There is a need for one, but defining a humanoid robot benchmark in terms of walking gait becomes a difficult philosophical problem very quickly. Do we look for a definition of the average human's gait? This could filter out differences introduced by somatotypes (ectomorphism, mesomorphism and endomorphism) but it would still leave us with the problem that men and women and people of different races have different gaits, as pointed out by Schafer [19] in his thorough analysis of gait. Progress is being made however as some researchers have begun using a Froude number to normalize walking velocities and a robot's ability to stably adapt to sloped terrains. But more broadly, if we chose to define a benchmark, walking under load, slow walking/speed walking and running all present different gaits. It is common knowledge that olympic short track athletes do not tend to perform as well over great distances. So this takes us in the direction of the next important matter to consider, purpose.

The purpose of walking is not just simply to get from A to B. It may also be to get from A to B while at the same time accomplishing something. Consider the skipping behaviour of 10 year old girls, where the purpose may simply be the act itself and no destination is significant (which is not a

silly example if we look towards the future of robotics entertainment). Broadly we could classify the application of humanoid robots into five major groups; medical, labour (home or industrial environment), entertainment, niche markets and military. The necessity for anthropomorphism varies considerably within these groups and so will their gait requirements. Medical applications will have the most strict requirements in the area of prosthesis. In order to interface with the existing human neural controller and to duplicate the role of missing body parts we can assume a perfect human match is the goal. However, human augmentation to expand our sensory perception or to allow for human action that is beyond human limits (wheeled people? omnivision?) is a likely future endeavour. The purpose of humanoid robots as a labour device is to replace man in his current chores, in places such as the home or where humans currently work in environments that are unhealthy. In this role the robot is required to use human tools or to operate in an environment that is already configured to the human physiology (kitchens, industrial environment, crash test dummies etc.). In entertainment there could be a lot of variance. Play companions for children, dancing robots, actors or sport robots will have their own specific gait requirements. Niche markets may include things like receptionist, teachers, waiters and hosts. Here mobility may take a back seat to behaviour and appearance. Finally there is the military. Here their purpose is logistics, supply and support. Here robust performance and durability will be significant and gait requirements too will vary.

With such a plethora of intended applications for anthropomorphic robots, are we going to be building them on a task by task application specific basis? Humans already play all of these roles, and it has been shown we are able to solve these varying locomotion and gait challenges using the same very low level neural control (section 2.1.7.9.) that adapts well to changes in restrictions of motion. We can consciously lower our torso height while walking and consequently a different gait emerges. Humans are very capable of learning to adapt to new gait challenges (skating, aerobics, mountain hiking etc.). It would seem then that the most important thing is adaptability and a high degree of flexibility in parameter values. The unifying aspect to all gaits is the requirement of maintaining balance. Having said that, walking is commonly likened to a continuously controlled fall. Each step begins and ends in the same unstable position, making use of the instability as a motive force. Static walks can be paused at any point, but more sophisticated dynamic walks are for a larger portion in the walk cycle, instantaneously unstable. It is as though we consciously force ourselves into a position of instability and then our nervous response system reacts to maintain balance. We do think before we act (plot some mental picture of limb trajectories) and then learned muscle responses take over to maintain our locomotion. So how do we model this incredible adaptability of the human body to maintain locomotion while simultaneously satisfying other objectives? What we really want is an all purpose engine that can self reconfigure its parameters to suit any humanoid robot, even if limited in degrees of freedom, and still be able to maintain locomotion while performing a vast array of other tasks. Though we are getting close, we are not there yet. For the moment, this research still produces models that are appropriate to specific target platforms. As such, not all existing models are meant to apply to all robot hardware designs. Some hardware designs necessitate the use of highly complex Lagrangian and Newtonian mechanics whereas other robots do not. Knowing which models suits which platforms is important before trying to force a particular set of equations to drive a given piece of hardware. Furthermore, most research has been performed on simple models of robots without a torso, with fully anthropomorphic robots being a very modern phenomenon, requiring more from a model than exists in the historically simple ones. Torso attitude, muscle and tendon tension and elasticity, and swing-limb dynamics play an important role in human locomotion and these are properties that do not exist in the simple, yet popular, passive rigid body compass gait model [20] or active inverted pendulum model. Yet in contrast, Collins [21] pointed out that “the Cornell and Delft bipeds demonstrate that walking can be accomplished with extremely simple control. These robots do not rely on sophisticated real-time calculations or on substantial sensory feedback such as from continuous sensing of torques, angles, or attitudes”. He notes that “no other robots have done particularly better at generating human like gaits even when using high-performance motors, a plethora of sensors, and sophisticated control.” So it would seem there does exist simplicity in modelling when appropriately applied and that there are competing walk control paradigms.

To clarify some of the existing misconseptions, we will take a top down approach and briefly review the issues we believe are crucial when designing a humanoid walk engine.

2.1.1. Properties of a Humanoid Gait

2.1.1.1. Walk Phases

Phases can be a little confusing as the terminology may either be in reference to support mode (single leg support or dual leg support phase) or leg motion (stance and swing phase Figure 2))

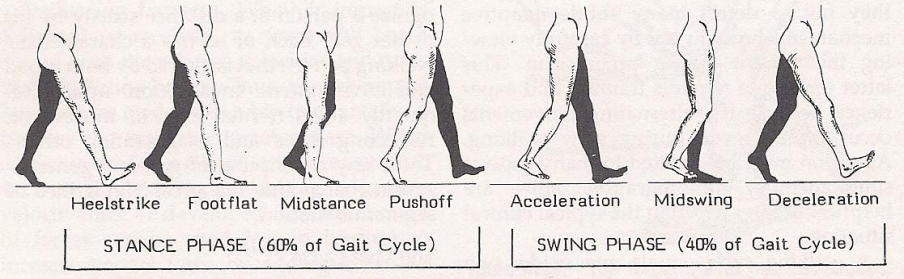


Figure 4.19. The complete gait cycle: stance and swing. Walking is a purposeful disturbance in body equilibrium during which alternating leg displacement sustains body weight.

Figure 2: [19] Leg Cycle Phases

Alternating leg swing phases will correspond with single support phase but stance phase is not the same as dual support phase, stance phase will include dual support but also partial single support. Walking engines themselves may or may not make implicit use of leg phases or support phases. Some walk engines are coded as a state machine implicitly that will perform different actions during different phases. The walk engine designed in this thesis uses one single state that is responsible for mapping the complete gait cycle which will be referred to as the 'walk cycle phase'. The only switching that occurs is between the forward and reverse trajectory mappings of the feet which is analogous to the stance and swing leg phases.

2.1.1.2. Ground Reaction Forces

Ground reaction forces (GRFs) develop during gait as a result of the force applied to the ground when the foot is in contact with it. GRF is equal and opposite to the force that the foot applies on the ground. Since GRF is an external force acting on the body during locomotion, it is of great interest to gait analysis [22].

2.1.1.3. Center Of Pressure (COP)

The Center of Pressure (CoP) is a point on the foot/ground surface where the net ground reaction force actually acts. Regardless of the state of stability of the robot, the CoP may never leave the support polygon [23].

2.1.1.4. Zero Moment Point (ZMP)

Surprisingly there exists some misunderstanding regarding the Zero Moment Point (ZMP). It has been interpreted as being nearly identical to the COP. To rectify this Vukobratovic wrote a paper entitled 'Zero-Moment Point – Proper Interpretation' [24] to clarify the concept. In brief, it is defined as either:

Interpretation 1: *ZMP is the point on the walking ground surface at which the horizontal components of the resultant moment generated by active forces and moments acting on human/humanoid links are equal to zero*.

Interpretation 2: *ZMP is the point on the floor at which the moments around x and y axes generated by reaction force and moment are zero*.

The misinterpretation has to do with the idea that the ZMP is always contained within the support polygon and that if there exist any non zero forces outside the support polygon then the ZMP will remain on the edge of the support polygon. There will be then some rotation around this pivot point. Goswami introduced the Foot Rotation Indicator (FRI) to account for these forces. Vukobratovic points out that this is an unnecessary new concept and that the COP and the ZMP are not the same thing, they are identical only when they both lie within the support polygon. He explains:

"*In the synthesis of various gait types one calculates ZMP position in the ground plane it may happen that the ZMP position falls outside the support polygon. In that case the calculated point does not represent the real ZMP but a hypothetical point at which ZMP would be if the support polygon was large enough to encompass it. As this is not the real case the calculated ZMP is outside the support polygon and the point thus calculated is an imaginary ZMP, as that point is not actually ZMP*

but represents the appearance of a perturbation moment and the beginning of rotation of the mechanism as a whole about the foot edge, yielding to its fall. Of course, the distance from the IZMP to the support polygon is proportional to the intensity of the perturbation moment."[24]

2.1.1.5. Humanoid Dimensions

Humanoid planes go by numerous names, commonly; Frontal, Sagittal and Axial (Figure 3)

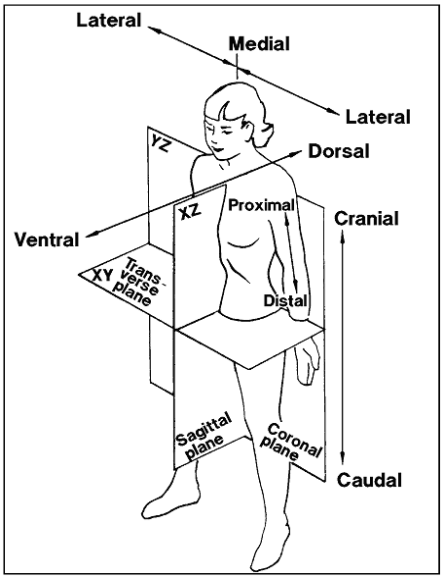


Figure 3: [25] Humanoid Planes

The convention of axes labelling is x projects forward from the torso (normal to sagittal) and the y axis is the lateral dimension. Polarity of the axes follow the Right Hand Rule.

2.1.2. Degrees Of Freedom / Dimensions, Kinematics, Simplified Models

2.1.2.1. DOF / Dimensions

Degrees of freedom (DOF) in mechanics refers to the number of independent parameters that define a systems configuration. In robotics DOF can refer to freedom of motion in different dimensions in a few different contexts. First, the degrees of freedom a robot has often represents the number of actuators or joints the robot has. This is one of the most significant properties of any particular robot as it will dictate how restricted the robot is in its ability to generate a complex gait. With one DOF a device could locomote by dragging itself across a floor. Bipedal locomotion is considerably more complex. The human body contains 230 moveable joints. Some robots are built to model the large number of DOF found in the human body that contribute to locomotion and interacting with the environment [26][27], with a more common approach being to use a reduced number of DOF that are necessary to achieve walking. The most important DOF used in locomotion are the hip, knee and ankle joints. The simplest design widely used in robotics research is the 2-dimensional planar biped. The planar biped typically has a 1 DOF hip and 1 DOF knee joint together with point feet [28]. When physically implemented, a boom is typically used to constrain the motion to the sagittal plane [29][30][31]. The most common active 3-dimensional bipedal designs use 10 to 12 DOF for the legs, 3 at each hip, 1 DOF for each knee and 2/3 for the ankle [32][33][34][35][36]. Several bipedal robots use innovative hip designs to improve mobility or reduce the DOF. The HRP-2 robot utilizes a cantilever hip structure to allow cross-legged walking (one foot in front of the other) [34]. The Nao robot uses a 45 degree mounted and coupled hip joint to achieve the yaw motion in each leg from a single actuator. Recent research [26][27] has been aimed at duplicating a human's high number of DOF using complex actuation mechanisms that emulate muscles. The additional complexity and large DOF drastically increase the complexity of the control strategy.

The other application of the concept of DOF is the freedom of movement of objects or parts. In this thesis we treat the robot as being 3 oscillating parts, the torso and each foot. Typically 6 DOF is desired for motion, that is 3 translation axis (X, Y, Z) and 3 rotations axis (pitch, roll, yaw). The Nao robot's torso and feet are independently free to move in 6 DOF but not simultaneously to the coupled

hip joint. This is considered 'under actuation' and restricts motion. The robot as a whole can be considered one object that has freedom of movement in different dimensions. As a whole a humanoid robot is free to move in 6 DOF, X-Y-yaw (the 3 common locomotive velocities that may be independent or simultaneous: 'omnidirectional'), Z (running, jumping) and pitch/roll (motion relative to the ground surface normal).

Increasing the number of DOF in models adds considerable complexity and drastically affects the significance of models. The simplest model, the compass walker [20], considers motion in only 2 DOF and is very often used to represent bipedal motion in its most basic sense.

A robot's dimension or shape will also have a drastic impact on its motion. Short legs /long legs, ankle height offset, hip width, existence of torso or arms and distribution of masses will all impact what would be a natural or preferred gait pattern. Existing robots do differ widely in this regard. One of the most significant differences between robots is the feet, in terms of actuation and shape. Robots may or may not have articulated toes that play a role in step to step transitions. Some robots simply have ball point feet, others have curved soles to allow a rocking motion. Most robots with flat soles have rectangular feet, the Nao is interesting in that it has irregular curved outline, much like a shoe print. The Nao's feet are also proportionally large, comparable to clown shoes.

Subtle differences between robot's DOF and dimensions can and do have a drastic impact on a robots motion in terms of what is ideal and what the limitations are. This most significantly affects modelling and control of the gait pattern.

2.1.2.2. Kinematics

Kinematics is the relationships between the positions, velocities and accelerations of the links of a manipulator**.**

2.1.2.2.1. Forwards Kinematics (FK)

Forward kinematics is the simple problem of using the known limb lengths and the joint angles in a serial link manipulator to compute the location and orientation of an end effector. This is a transformation from the base origin to the end effector via offsets and rotations. The two formal conventions are the popular Denavit-Hartenberg Representation [37], and the lesser known Craig's Convention [38]. Using these methods the offsets and rotational matrices can be summed together. Another simple method is to process each link and rotation as they appear according to their mounting sequence. Either by beginning at the base and working outward until the end effector is reached, or by working with a theoretically zeroed manipulator and working backwards from the end effector, manipulating an endpoint around each new offset rotation until the base is reached.

2.1.2.2.2. Inverse Kinematics

Inverse kinematics (IK) is the opposite of forward kinematics, the location and orientation of the end effector in reference to the base and link lengths are known, but the joint angles are not. The IK problem can be solved analytically or iteratively. As the analytical solution quickly becomes a very difficult geometry problem with increased number of links and joints, iterative methods are often preferred though they are not exact solutions as analytical ones are.

The iterative method is the application of Euler's method formally known as the 'Jacobian Transpose' method as it makes use of an inverse Jacobian matrix to determine the partial derivatives in each Cartesean coordinate that comprise the error vector between each iterations forward kinematic computation and the target point. For an excellent explanation and derivation from first principles see Welman's 1993 thesis [39] that details the application of the Jacobian method to articulated limbs.

Iterative methods can suffer from a variety of problems such as there being no solution if the target is outside the work space, multiple possible solutions (some more ideal than others), aliasing of end effector motion trajectories, problems at orthogonal axis, computational overhead with complex manipulators, and a trade off between algorithm speed and accuracy.

Existing documented methods (bHuman [40]) for the Nao robot were relative to the robot's torso (base) and did not relate the feet (end effectors) to a world space origin. This is not a problem in the absence of any pitch or roll in the robots torso, but torso attitude control was desired for this project. Attempts to apply the Jacobian Transpose method proved to be difficult given the robot's mechanically coupled hip joint (under actuation). Therefore, an exact analytical solution was worked out for the Nao robots entire body. Due to the under actuation in the hips, the torso and both feet are not able to move freely in their own 6 DOF. One degree of freedom must be sacrificed somewhere to solve the problem. Torso attitude was determined to be more important than foot yaw and therefore incorrect foot yaw solutions are returned when there exists torso roll. This solution is attached as Appendix L.

2.1.2.3. Modelling

Modelling is the most important aspect of a robots motion design. It is generally reckless to begin working on the motion of a mechanical system directly without prior simulation of control output as common model errors or code bugs can lead to violent jerky commands that can potentially destroy the device itself, or cause harm and even death (out of control large manipulators or vehicles). A complete dynamic mass model of a humanoid robot can become a large, very complex, problem. Often robots are modelled purely kinematically or very simply (reduced DOF, mass approximations or conceptual based models). In different fields, a researcher's purpose affects the underlying assumptions that are made, and the types of approximations that are applied to the problem. This can lead to misinterpretation or misapplication of existing models or methods (applying force control methods to position control robots). Computer animation developers may wish to apply dynamics in their work to create more realistic looking motions and interactions, whereas a researchers in the medical rehabilitation field would be interested in calculating tensile forces and torques in the design of a prosthetic limb. These two different goals will generate different dynamic solution methods, one concerned with computation speed, the other with accuracy. Simple models can be used to effectively represent complex systems, whereas even complex models may not be truly complex enough to be very accurate (a complete dynamic mass model of the Nao robot would likely overlook its highly plastic and low manufacturing tolerance properties (though the Microsoft Robot Developer Studio [41] which supports the Nao, did well to model the non-ideal properties of the joints). Very often models only work given self imposed simplifications and assumptions. These simplifications should be well understood when applyed to a model in practice. Overlooking system or environmental properties can drastically affect results of practical application just as much as outright errors. Applying the wrong models to a hardware platform that the model was not intended for can also be wasted effort.

Common representations use idealized joints and idealized motors to power motions. For example, the knee joint is usually modelled as a single degree of freedom joint with an idealized motor that can move the calf. In reality, the joint consists of four bones coupled together through flexible tendons and muscles, cushioned by cartilage and powered through the contraction of hundreds (thousands) of antagonistic and synergistic muscle fibers [42]. The simplest model is the 'compass gait' model [20] that has been used to research passive walkers on an incline. The most commonly referenced model is the inverted pendulum [43]. The most complex models calculate the full body Lagrangian and Newtonian dynamics [44]. One of the most interesting thing about the existing research is that modelling is almost always done in the sagittal plane with few researchers even considering motion in the frontal plane.

The modelling that will be done in this thesis will be massless kinematic modelling and inverse kinematics for the purpose of creating a model that can be simulated to test gait properties, body part Cartesean trajectories, ZMP behaviour, and to be able to reverse engineer the Aldebaran walk given joint sample data.

2.1.3. Actuation Type

Humanoid robots are typically driven by three types of actuators; electric motors, hydraulics and pneumatics. Their differences play a significant role in the appropriate method of control and types of joint motions that are possible.

2.1.3.1. Electric motors

Electric motors are most commonly used to power active bipeds; AC motors, DC motors or servomotors. Some robots have custom motors designed to improve application performance (Honda humanoid robots [45][33]). Electric motors can be direct drive but are most commonly geared. Direct drive coupling allows for high speeds but low torques. Geared drives often make use of high gear ratios to deliver greater holding torques. Standard gearing mechanisms suffer from the backlash problem while other high end (expensive) options like harmonic drives do not [46]. Waseda University's WABIAN-2 robot combines a DC motor with harmonic drive [35]. Alternatively, KAIST's HUBO robot uses harmonic drives in the lower body, as backlash in these joints can affect the overall stability of the system [33]. With reasonable manufacturing tolerances, geared electric motors are capable of a high degree of position accuracy. The major drawback of using electric motors is their high mechanical impedance (limited compliance) and drive train losses when coupled with geared drive mechanisms [46]. The Nao robot uses geared electric motors with high gear ratios and low precision plastic gears that suffer significantly from backlash and rapid mechanical drift due to continuous use.

2.1.3.2. Pneumatics

Pneumatic actuators are force driven piston mechanisms regulated by the compression of air. These mechanisms resemble muscles more so than electric motors and are referred to as Pneumatic Artificial Muscles (PAM). The most common variation used in lower body joints is the McKibben muscle (braided PAM) [46]. Delft University has produced several robots (Mike [47], Max [48] and Denise [49][50]), which use McKibben muscles. The advantage of pneumatic drive systems is the inherent compliance and high power to weight ratio [51]. The major drawback of pneumatic actuators is a non-linear response and poor position accuracy [46].

2.1.3.3. Hydraulics

Hydraulic actuators are similar to pneumatics but instead use compressed oil. Hydraulics have a higher power density and are more precise than pneumatic drives, but are more costly,messy and require more maintenance. The non-linear properties of hydraulic actuators make joint level control difficult [46]. Impedance control schemes are used to reduce the non-linear effects [52]. The SARCOS humanoid CB is an example of hydraulic actuator use [53].

2.1.4. Control Type

There are two main types of humanoid robot control: position and force.

2.1.4.1. Position Controlled

Position controlled robots are typically built with high gear ratio electric motors. These motors are cheaper, smaller, and provide adequate torque. Geared motors have a high mechanical impedance that creates poor inherent damping, yet on the other hand they are useful for rejecting small disturbances. Position controllers can track walking patterns very accurately and can simplify gait pattern generators to footstep planning. They do however lead to poor force control and do not handle larger disturbances as well [46]. Holding torque may be parametrized to improve the compliance properties as it is done with the Nao robot. However, end effector position control on the Nao robot is somewhat limited as the joint position sensors are mounted on the drive side of the joint and do not capture the lack of precision or backlash.

There are many robot designs [32][54][34][55] that make use of position control[56][57][58] to perform stiff and accurate trajectory control.

2.1.4.2. Force Controlled

Force-controlled robots are usually constructed with more direct-drive actuation [46]. As suggested by Pratt [59], there are three major benefits of full-body force control. First, the low impedance allows the controller to exploit the natural dynamics of the system, such as a passive swing leg. By doing so, it may be possible to significantly reduce the control effort and simplify the controller. Second, full-body force control allows for the distribution of impact forces throughout the body. This makes force controlled robots more able to compensate for larger disturbances than position controlled. Finally, the compliance offered by force control allows the system to comply with unknown ground geometry. Pneumatic and hydraulic drives system are common in force controlled robots. Electric motors can be used, however, they have a low force to weight ratio. Hybrid systems have been developed to combine the advantages of different actuators [60].

Dynamic locomotion, using torque-based control, has also been widely studied using methods such as virtual model control [61], impedance control [62], and dynamics filtering [63]. Designing force controllers can be difficult given the complex dynamics of the robot.[46]

2.1.5. Locomotion

Robots can be classified as either employing static or passive-based locomotion. The differentiation has to do with the behaviour of the ZMP during the walk cycle.

2.1.5.1. Static Locomotion

Static walkers are the most basic. If the walk is stopped at any time it will remain stable. In static walkers, the COM never leaves the support polygon [64]. Bipedal controllers are typically simple state machines that shift the COM over one leg, move the opposite leg forward, shift the COM over the advanced leg, move the first leg forward, repeat. Fallis' [2] original walking toy was a static walker.

2.1.5.2. Dynamic Locomotion

Dynamic walks are not always stable while on one leg, the COM can fall outside the support polygon during the transition from one foot to the other. A dynamic walk could be described as a continuously controlled fall. There are three groups of dynamic walks; passive, active and minimally active.

2.1.5.2.1. Passive

Passive dynamic walkers are purely mechanical devices which are capable of walking without any active power, relying on the natural dynamics of the system for gait generation. In contrast to more common robots, which actively control every joint angle at all times, passive-dynamic walkers do not control any joint angle at any time [21]. McGeer first demonstrated this concept on a simple mechanical system consisting of two straight legs connected by a hinge at the hip [3]. This design is considered to be the simplest walking model and is referred to as the 'compass gait model'. The design was expanded by adding knees for more anthropomorphic gait, and to allow foot clearance during the recovery phase [65]. McGeer showed that passive dynamics is not only suitable for the swing leg motion, but for the stance leg motion as well. He built models and prototypes that can walk down a shallow slope with no actuation and no control. Later, Garcia et al. performed an analysis of the speed, efficiency and mechanical design of 2D passive walkers [66]. Collins et al. built the first 3D passive bipedal robot which used swinging arms to counteract the lateral instability [67]. More complex prototypes [68] showed that passive designs are capable of producing remarkably human-like gait without any active power [50]. There have been efforts to embed passive dynamic systems into a nonlinear full-body controller [69]. McGeer's early work has led to recent biomechanical insight into the inherent passive properties of human gaits [70].

Passive dynamic walks have two significant features: inherent mechanical gait stability and low energy consumption. Garcia et al. showed they can be very efficient, even reducing energy consumption to zero, like an ideal rolling wheel [71]. Another nice feature of a passive walker is that it does not specify a target gait trajectory [72]. When the walk is disturbed, the perturbation is eliminated gradually over many steps through the support transfer that occurs at the end of each step. For these systems, stability analysis is often expressed using a measure of gait sensitivity [73]. These design however can only withstand small gait disturbances.

2.1.5.2.2. Active

The passive mechanical designs relied on gravitational power and sloped inclines or booms with active treadmills. To walk on level ground without external force inputs, actuation is required. Most robots today are actively powered.

The first theoretical attempts to describe the mechanics of an active bipedal robot were made over 40 years ago [74][4]. Kato et al. from Waseda University, Japan (1974), are generally acknowledged to have been the first to design and implement a successful active walker [64]. It relied on a static gait pattern, with dynamic walking only being achieved in the late 1980s [75].

2.1.5.2.3. Minimally Active or 'Virtual Passive Dynamics'

Early study into passive dynamics was confined to the scenario of walking on a slope, using gravity as a source of power. This research gave rise to new methods focused towards mimicking the work done by gravity [76] and was helpful in characterizing the nature of a passive gait cycle [77] in terms of stability and energy consumption. More recent work extends passive dynamics principles with the aim of using minimal actuation as a replacement for gravity, allowing passive-based robots to walk on level ground [21][78][79]. This has produced walkers that are more energy efficient. These robots are classed as 'minimally active' or 'virtually passive'.

Delft University has a few minimally active robots designed specially to take advantage of the natural system dynamics [49]. Denise [80], uses minimal pneumatic actuation at the hip joint to achieve walking on level ground. Flame, uses series elastic actuators (SEA) [81]. Another approach to designing minimally actuated walkers is to use actuation at the ankle joints [21][82][83][84], often in addition to the hip joint. Meta, another energy efficient walker from Delft University, uses two motors at the ankle in addition to the two motors at the hip joint [83]. Ankle actuation has been shown to improve the versatility of some walkers by allowing variable walking speeds [84]. These minimally actuated bipeds have shown similar energy efficiency properties as the passive walkers [85]. The difference between fully actuated systems and the minimally active ones is that the joints are not actuated through the whole gait cycle, only at certain phases such as push off [86].

Most of the initial research was confined to the sagittal plane. The difficulty of extending these models to 3D was unstable lateral dynamics caused by an out of phase sway velocity with the foot

landing timing [72]. This is a problem, as small disturbances could destabilize a passive walker. These obstacles were overcome in part by modifying the mechanical design to include improved compliance and some 3D minimally active bipeds have been realized [21][80][49]. These systems still share the same sensitivity to large gait disruptions as purely passive walkers [86].

2.1.6. Problem Space

The two main conceptual problems spaces that exist to work in are the Cartesian motion trajectories of body parts, or the angular joint position signals over the course of a walk cycle 'Joint Space'.

2.1.6.1. Joint Space Trajectories

Describing and calculating joint trajectories is one way to control servo motor driven humanoid robots. It is however the most non-intuitive work space as joint signals are irregular high order polynomials that change significantly with the smallest gait adjustments. Focusing on a specific joint motion signal is required in the medical industry when replacing a specific joint or series of joints in prosthesis as their work is in part concerned with specific muscle actuation patterns. It is a very complex, bottom up, approach to modelling the motion of an entire humanoid robot, though is attempted. Takanishi's research group in Waseda University presented the humanoid robot WABIAN, where the trajectories of the arms, legs and ZMP were described by Fourier series [87]. The coefficients were found in simulation as a way to ensure the ZMP stability. Ude et al. described the joint trajectories as b-spline wavelets[88]. A drawback of this joint space approach is that a detailed and valid model of the target system has to be identified, and changes in the target system require a redesign of the model [89]. It has also been shown that it only works for a fixed set of gait parameters [90], changes in the simplest property such as velocity results in considerably different joint space trajectories and then control models fail.

Aside from attempting to build a gait upwards from what is normally the output, joint space trajectories do have more practical uses in the field of robotics. In order to model a more human like gait, robots can be driven via joint space motion capture data from human subjects [91][92][88]. Repetition of joint signals (measured) over successive walk cycles with little variance can also be a demonstration of stability in compliant systems (limit cycles)[93][77]. Joint signal analysis is also a key component in the calculation of Lagrangian dynamics in force controlled robots.

2.1.6.2. Cartesian limb trajectories

The Cartesian workspace is considerably more intuitive. Trajectories of key body parts (feet, hands, torso etc.) are plotted spatially. The important consideration here is coordinate systems. The main frames of reference are the world space, full body, torso and end effector. The world space is the highest level of abstraction that defines the global environment that all entities relate to. Within the world space a robot is defined as having a location and orientation. This location would be the origin of the robot's full body coordinate system within which all body part locations can be defined. For a humanoid robot this origin is centred between the feet and lies directly below the collinear location of the spine and pelvic center when the robots joints are all zeroed (*Denavit–Hartenberg* convention). The torso is also commonly used as a reference base for the arms and legs and is the origin for partial body inverse kinematic algorithms. The goal of many robotic motion applications is to relate the coordinate frames of end effectors to any of the other frames of reference, the torso, full body robot space or another object defined in the world coordinate system. Cameras have their own reference frame that in non-swaying multi-legged (4 or more), or wheeled robots, is fixed in at least one dimension with the robots full body coordinate system. This not the case for humanoid robots. Despite the more intuitive nature of Cartesian space in comparison to joint space, the relativity of position, orientation and motion between different frames of reference can become surprisingly challenging. Euler angles and Tait–Bryan are the standard conventions to transform orientations from one frame of reference to another.

2.1.7. Popular Models and Methods

There exists a wide range of walking control strategies and gait generation methods to achieve bipedal locomotion. Some of the most popular strategies are briefly reviewed in this section. It important to keep in mind not all models or methods were meant for all platforms or purposes. Also this research is still in its youth, this problem is by no means an ideally solved one and there exist some competing theories.

Models are also generally 'simple models', and it should be kept in mind that these simple models are being applied to very complicated moving machines. But that, by no means, is to say there are not a number of very advanced platforms and models currently being professionally developed (AAUBOT[6], PETMAN[7], ROBOY[8], ASIMO[9], HRP-2 PROMET[10], iCUB[11]).

2.1.7.1. Compass Gait - 'Simplest Model'

Garcia et al.'s irreducibly simple, uncontrolled, 2D, two-link model is considered to be the simplest model of bipedal motion (Figure 4). This model is the simplest case of the passive-dynamic models pioneered by McGeer. It has two rigid massless legs hinged at the hip, a point-mass at the hip, and infinitesimal point-masses at the feet [70].

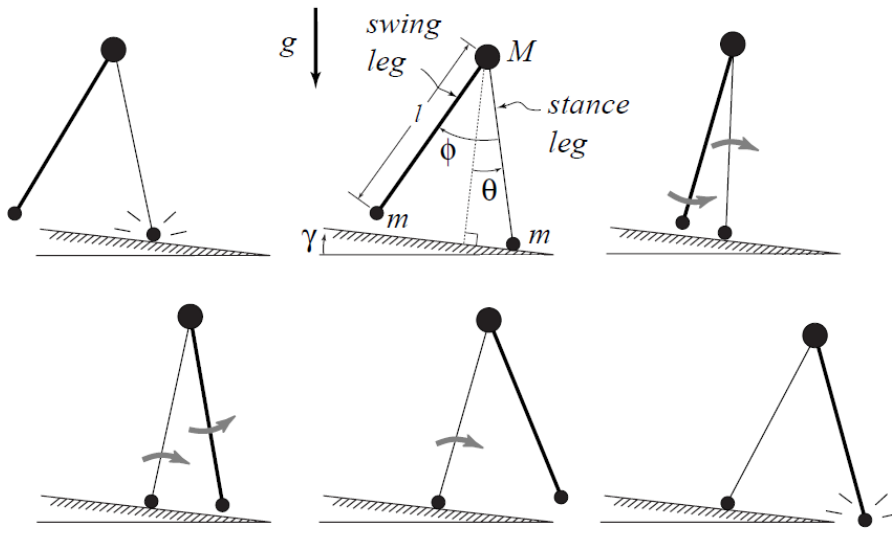


Figure 4: [70] A typical passive walking step. The new stance leg (lighter line) has just made contact with the ramp in the upper left picture. The swing leg (heavier line) swings until the next heel strike (bottom right picture). The top center picture gives a description of the variables and parameters that we use. Θ is the angle of the stance leg with respect to the slope normal. {phi}is the angle between the stance leg and the swing leg. M is the hip mass, and m is the foot mass. l is the leg length, γ is the ramp slope, and g is the acceleration due to gravity.

2.1.7.2. Six Determinants of Gait Theory

Saunders et al. defined walking as the translation of the center of mass through space in a manner using the least energy expenditure [20]. They identified six determinants that affect the energy expenditure and mechanical efficiency; variations in pelvic rotation, pelvic tilt, knee flexion at midstance, foot and ankle motion, knee motion, and lateral pelvic displacement. The “six determinants of gait” theory proposes that walking efficiency is enhanced by reducing the displacement of the COM (Figure 5). The assumption is that there is a an energetic cost to raising and lowering the COM. However, the drawback of walking with a level path for the COM, is that the joints must realize larger motions. The supporting knee must also be flexed during swing phase as opposed to passively locked, so that heavy extension torque is needed to support body weight. The high torque and large joint motion leads to a more than doubling of energy expenditure compared with normal walking [94]. Carey has confirmed there is a higher metabolic cost for walking in this way [95]. This model is also inconsistent with empirical data [94].

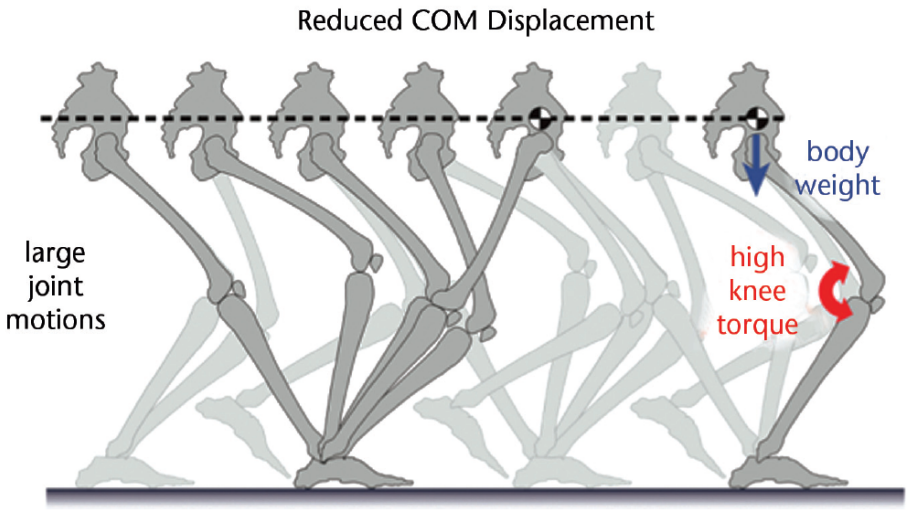


Figure 5: [94] Six Determinants of Gait Walking Model

2.1.7.3. Inverted Pendulum

The inverted pendulum model (Figure 6) is by far the most widely cited, documented and implemented dynamic walking model. In this conceptual model, the stance leg is modelled as a rigid inverted pendulum that pivots around the ankle which allows the body COM to move in an arc conserving mechanical energy [94]. Ideally no work is required to move the body, and no torque is needed in the knee to support its weight. Longer and faster strides also require no increased energy expenditure. The swing leg as well appears to move like a pendulum, whose motion ideally requires no work. Kuo [94] claims this model is supported by empirical metabolic expenditure data.

It is useful but is it that accurate? Some researchers are beginning to see issues related to the discontinuity between walk cycles and the energy expenditure during the signal low point [43] or the need for compliance [96][97] during this impact part of the walk cycle phase (compliance here would smooth out the trajectory between cycles).

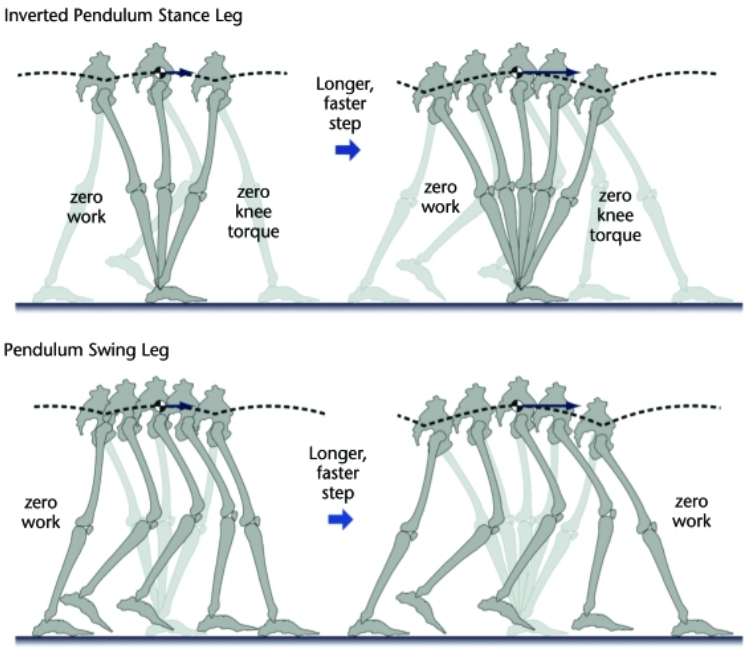


Figure 6: [94] Inverted Pendulum Model

Borghese et al. [98] sought to find properties of a walk that are consistent across the normal range of speeds. They found the following parameters changed monotonically with speed: step length increased, gait cycle duration and the percentage duration of stance decreased with increasing speed. They observed that the kinematics of the leg began to differ significantly from what was predicted by the ideal inverted pendulum model. They found that the rotation velocity of the hip around the ankle is sinusoidal during swing, however velocity is nearly constant during stance phase at all explored

speeds. The model works at slower velocities but begins to fail as velocity increases. The model tends to be accurate during swing phase but not during dual support. These observations suggest then the inverted pendulum does not model the step to step transitions, a fact that has also been cited in examinations of conservation of energy between walk cycles [94]. The model does capture various properties of a walk, such as conservation of momentum during swing phase and the rising and falling of the COM, but as a rigid body, the model creates heavy impact dynamics during foot landing. The model then maybe just be too simple or incomplete as opposed to invalid. Kuo [43] suggests the single-support phase never behaves exactly as an inverted pendulum and that the exceedingly difficult to determine properties of soft tissues (elastic tendons) are responsible for maintaining the COM trajectory.

Regardless of it's accuracy in describing the complete walk cycle, the inverted pendulum simple model remains the most popular cited and implemented paradigm today. It is also commonly used to build walk engines for position controlled robots despite its purpose being to model system dynamics and ankle torques [40].

2.1.7.4. 3D Linear Inverted Pendulum / Cart-on-a-Table

The inverted pendulum model was extended to 3D (IP-3D) by Zijlstra et al. [99]. This model resulted in an over estimation of the COM trajectory amplitude [100]. Hayot et al. [101] then proposed a hybrid model combining the inverted pendulum and the 'six determinants of gait' model. What was found from this combination was that as the vertical displacement is flattened, the CoM trajectory becomes more like the real one in comparison to the IP-3D [100], despite the consequential estimations of energy and work [102].

Following this, Kajita, et al. [103] kept the inverted pendulum principles and flattened the vertical displacement entirely. The 'Linear Inverted Pendulum Model' (LIPM). The LIPM (Figure 7) is the projection of a pendulum onto a horizontal plane. This motion in a horizontal plane has also been referred to as the "cart-on-a-table" model. The cart-on-a-table model (Figure 8) has also been put forth by Kajita [104] and has since been expanded by Suleiman et al. [105]. Another extension to the LIPM is the AMPM, or Angular Momentum inducing inverted Pendulum Model [106], which generates momentum by applying a torque about the center of mass (COM)[46].

What happened to the rising and falling nature of a human walk?

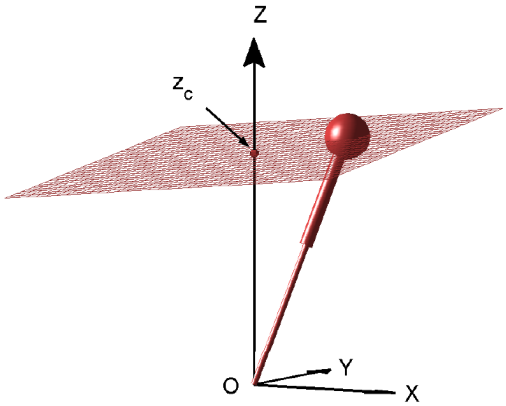


Figure 7: [104] A pendulum under constraint

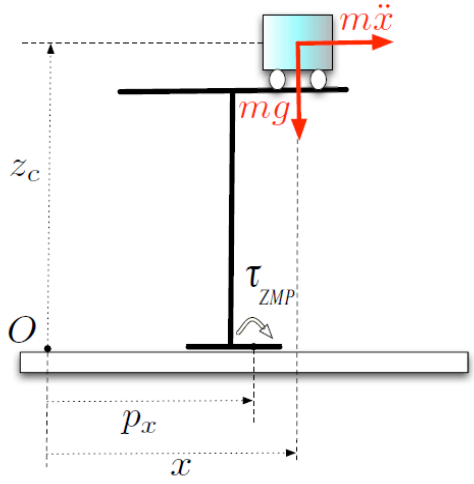


Figure 8: [104] Cart on a table model

2.1.7.5. Spring Mass Model

Historically, walking has been characterized as an inverted pendulum gait, where the exchange of elastic and potential energy powers the motion of the COM as it vaults over rigid legs. With the analysis of Ground Reaction Forces (GRF) it has been found that vaulting over rigid legs cannot truly reproduce the results of walking [107]. Instead of the large vertical amplitudes suggested by the rigid inverted pendulum model, the upper body shows comparably small vertical amplitudes during walking [108][109]. This discrepancy is also reflected in the forces acting on the centre of mass [110] [111].

To account for the apparent flaws in the inverted pendulum model, Geyer [112] introduced the Spring-Mass model (Figure 9). This model replaces the rigid legs in the simplest model with spring loaded legs. Geyer analyzes the GRF and reports that his results suggest that the two fundamental gaits of walking and running are much less different than generally assumed; with the same compliant stance-leg behaviour found in running, a bipedal spring–mass model could reproduce the stance dynamics observed in walking, and that walking is, like running, a bouncing gait. In this model the single and double support phases in walking replaces the flight and stance in running. His results also show that the multi-peak GRF patterns show that walking and running are just two out of the many possible stable solutions to bipedal locomotion of the same mechanical system. Each solution represents a separate domain in the parameter space that are isolated by a gap in system energy and locomotion speed. Geyer concludes that this could explain why both gaits are perceived as distinct gaits in human locomotion, even though they represent the same mechanical concept that is based on compliant leg behaviour. [112]

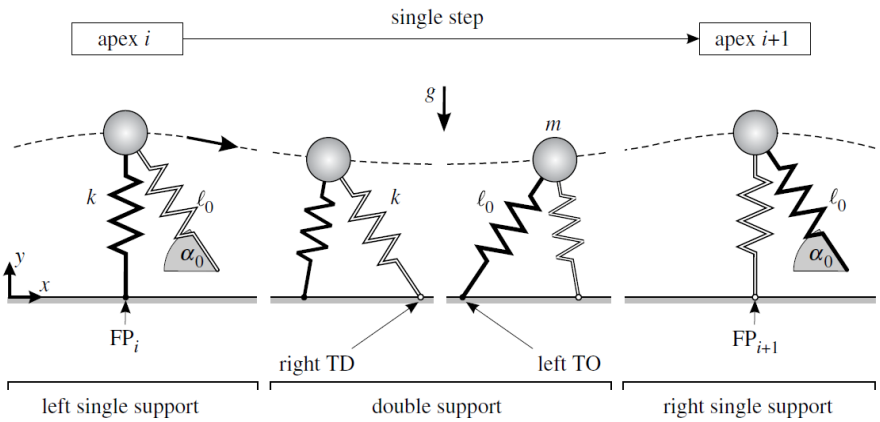


Figure 9: [97] Spring Mass Model Spring Mass Model

Figure 10 shows a comparison of ground reaction forces between the rigid inverted pendulum model, the spring mass model and empirical data, where in the figure:

A) A simple inverted pendulum produces a GRF pattern (red) that is a poor approximation of the GRF observed in walking humans (black).
B) GRFs developed during walking are consistent with an inverted pendulum model, if the model includes a leg spring.
C) The parabolic pattern of vertical ground reaction forces (F_y) and the sine-wave pattern of horizontal forces (F_x) in running are produced by a simple spring–mass model, and absolute values of forces can be matched with appropriate values for leg spring stiffness and angle of attack.[112]

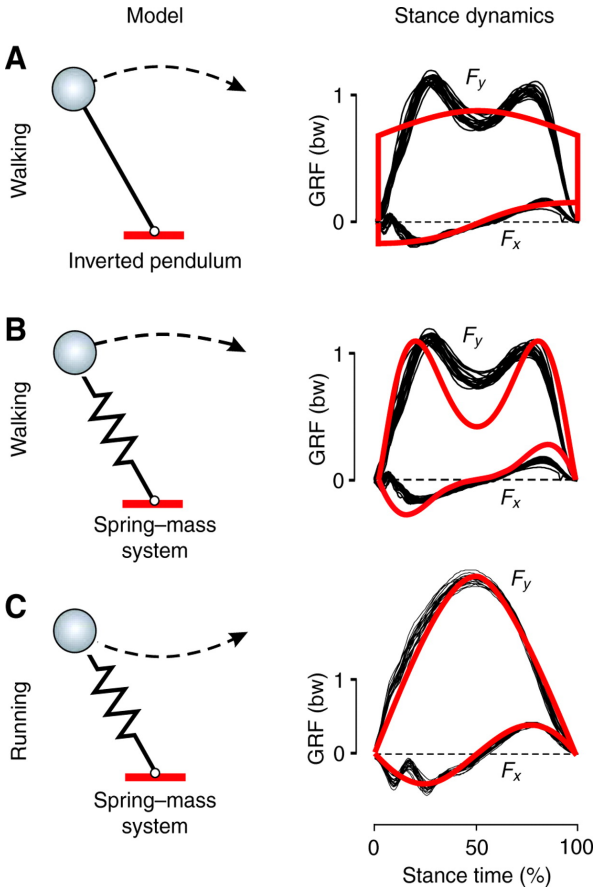


Figure 10: [112] Comparison of Inverted Pendulum and Spring Mass Modelled and Empirical GRF Data

Geyer's results also challenge the view that walking efficiently depends on how close the COM trajectory resembles an inverted pendulum. Classically, walking efficiency is quantified by the percentage recovery, a parameter that determines how much of the stride energy is recovered using the inverted pendulum's compensating exchange of gravitational potential and kinetic energies [113][114][115]. Geyer points out that for an ideal stiff-legged walk, the percentage recovery would be 100%, yet walking experiments show that it reaches 70% at best [113][116]. The difference lies in the dual support stance where reversing the COM in the vertical disturbs the motion of consecutive inverted pendulum arcs [117][3]. Geyer explains the bipedal spring–mass model shows that double support phases are necessary to achieve the walking dynamics and that low percentage recovery (15–35%) does not indicate inefficient walking. The spring–mass model recovers 100% of the stride energy by transiently storing in the leg springs the energy that would otherwise be lost during double support. Other experimental findings also support such an elastic contribution [118]. Therefore Geyer argues, walking efficiency seems to depend less on how close the COM trajectory follows inverted pendulum arcs, but more on how much of the stride energy can be stored elastically when redirecting the COM in double support.[112]

Other more complex walking models use detailed representations of the leg components that include springs, dampers, multisegments or neuromuscular structures [119][120][121][122]. Though these models describe the dynamics of walking and demonstrate that compliant leg behaviour is a fundamental component of walking much better than the inverted pendulum can, they can be quite complex as conceptual models. Hence, despite being inaccurate, the stiff-legged motion remains the mechanical paradigm for the walking gait [115][123].[112]

2.1.7.6. Zero Moment Point Based Methods

2.1.7.6.1. Zero Moment Point Criterion

Popular techniques to achieve bipedal walking have been trajectory generation and control strategies based on the ZMP criterion [124]. The criterion states that the biped is statically stable if the ZMP is kept within the region of foot support at every time instant. The ZMP location is commonly used as a feedback mechanism to achieve stable gait via either foot pressure sensors or online modelling. Methods like Huang et al. [125] may incorporate a maximized 'stability margin' (distance from the ZMP point to the boundary of the support region).

There are a couple of drawbacks to such methods. First of all, the strategy does not account for non-level surfaces or unexpected impact forces. Secondly, they are energy inefficient as they actively maintain a continuous trajectory through support polygons, which does not make use of the natural dynamics of a humanoid. This results in a static, non human looking, gait. Despite the drawbacks, some of the most popular humanoid robots utilize some variation of ZMP based feedback for locomotion (Honda ASIMO [45], Aldebaran's Nao [126]).

2.1.7.6.2. ZMP Preview Control

Kajita [104] designed a ZMP controller called 'preview control' based on combining Katayamas [127] preview servo controller with the LIPM . This method plots a signal of ideal ZMP locations (foot center) and a damped controller then adjusts the COM location to follow this reference signal (Figure 11). Each step then becomes a sequence of reference ZMP values. A ZMP tracking controller (based on the table-cart model) is used to generate the desired future COM trajectory. The preview controller uses this future reference signal (within a specified time period) to generate the adjusted current COM trajectory to remain dynamically stable. More recent methods use more accurate modelling techniques (instead of 3D LIPM) to compute ZMP based trajectories on-line [128]. Online future trajectory mapping is known to be very computationally expensive, but has been solved efficiently with quadratic programming [129]. This method can be modified to include footstep placement [130][131] by considering only a fixed set of foot placements at each step and searching over a fixed horizon [46].

This method has been implemented on the Nao robot by Strom [132] of Bowdoin College. He reports that 'Using our system of coordinate frames coupled with preview control, we are currently able to achieve maximum forward walking speeds of 10.5 cm/s, which is comparable to the maximum walk speed of the Aldebaran walk engine. However, at such speeds, the robot is not very stable. In practice, we prefer a gait which has a maximum speed of 7 cm/s, with a step frequency of 1Hz, which is much less prone to falling, even during large changes in the motion vector'.

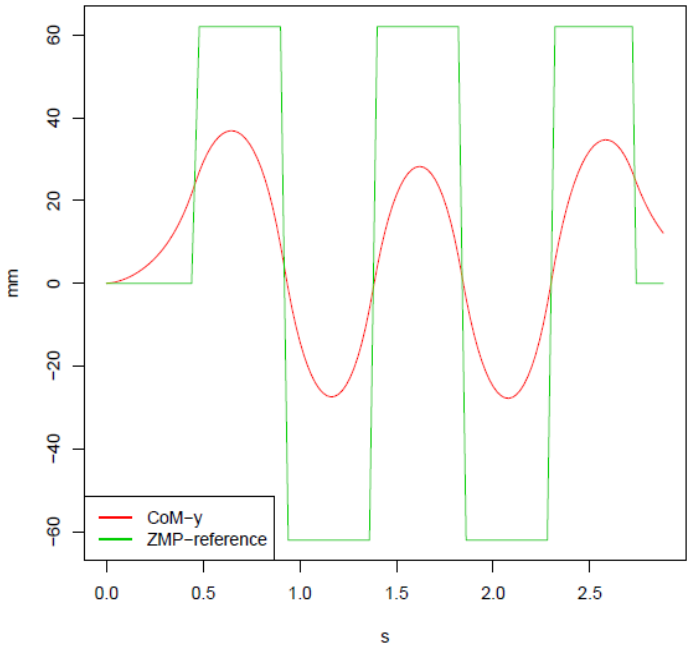


Figure 11: [132] Lateral CoM movement given a reference ZMP plotted during a sequence of five steps

2.1.7.6.3. Theory of Capture Points

Pratt's theory of Capture Points (CP) [133] (previously considered by Townsend [134]) is an interesting foot placement method based on the ZMP. It carries with it somewhat of a lengthy description, but in summary it plots a trajectory toward possible stable foot placement positions (Figure 12). It does so by making use of the following definitions;

Definition 1 (Capture State). *State in which the kinetic energy of the biped is zero and can remain zero with suitable joint torques.*

Definition 2 (Safe Feasible Trajectory). *Trajectory through state space that is consistent with the robot's dynamics, is achievable by the robot's actuators, and does not contain any Falling States.*

Definition 3 (Capture Point). *For a biped in state x, a Capture Point, p, is a point on the ground where if the biped covers p, either with its stance foot or by stepping to p in a single step, and then maintains its Center of Pressure to lie on p, then there exists a Safe Feasible Trajectory that ends in a Capture State.*

Definition 4 (Capture Region). *The set of all Capture Points.*

Definition 5 (Two-Step Capture Point). *A point on the ground, p, such that if the biped swung its swing leg to cover p with its foot and maintained its Center of Pressure to lie on p, then there exists a Safe Feasible Trajectory, such that at some state along the trajectory, there exists a Capture Point.*

Definition 6 (Two-Step Capture Region). *The set of all Two-Step Capture Points.*

Definition 7 (N-Step Capture Point). *A point on the ground, p, such that if the biped swung its swing leg to cover p with its foot and maintained its Center of Pressure to lie on p, then there exists a Safe Feasible Trajectory, such that at some state along the trajectory, there exists an N-1-Step Capture Point.*

Definition 8 (N-Step Capture Region). *The set of all N-Step Capture Points.*

With these defined concepts, it is then possible to take steps to a secure location or recursively compute stable steps towards a location outside the current region of stability. By placing a foot at this location an idealized system will come to rest without any feedback control. Also, multiple steps can be planned if the capture region is outside the current range of motion. A walk constructed this way is not an inherently oscillatory one which has the advantage of being able to walk on irregularly placed stepping stones.

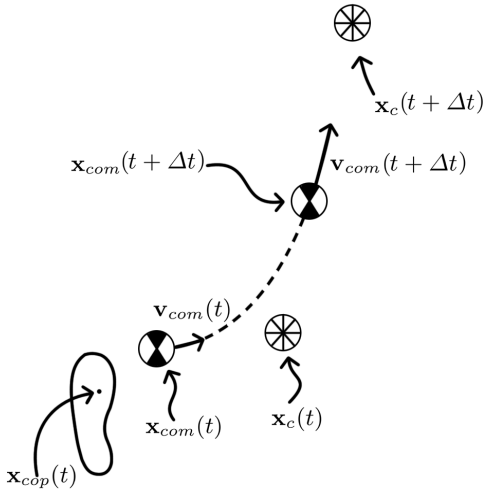


Figure 12: [133] Evolution of the Center of Mass and a Capture Point from time t to t+Δt. xcop(t) is the location of the Center of Pressure; xcom(t) and vcom(t) are the location and velocity of the Center of Mass; and xc(t) is the location of the Capture Point.

Englsberger et al. built on top of Pratt's CP concept [135] by developing two robust controllers based on LIPM dynamics. The CP end-of-step controller (CPS) responds to perturbations in real-time and adjusts the ZMP of the biped to shift the CP and regain stability. A second CP tracking (CPT) controller was also developed to realign the CP to its ideal trajectory if the biped experiences perturbations while walking. Both controllers were demonstrated in simulation and on the physical DLR biped. A similar approach was also developed in [136] which integrates the CP concept with Model Predictive Control (MPC). The MPC control scheme was developed to improve ZMP preview control for robustness to strong perturbations [129]. The CP-MPC control scheme was also demonstrated on the DLR biped.

Recently, a more comprehensive approach using CP for foot placement and gait synthesis has been proposed. De Boer et al. [137][138] focused on the ground/foot interaction to develop a robust and energy efficient walking control strategy known as the capturability framework. Pratt demonstrated this strategy on the force-controlled compliant lower body biped M2V2 [139][140]. While the capturability framework is philosophically similar to the idea behind FPE, there are several key differences. The FPE approach uses simple local controllers to form complete gait cycles and can be used on position-controlled joints without any complex actuation systems.

2.1.7.7. Foot Placement Estimator

Another similar step planning based method is the Foot Placement Estimator (FPE), introduced by Wight et al. [31]. It is an extension of Pratt's CP to use the inverted pendulum dynamics (conservation of angular momentum) to predict the location of the capture point. It is used to build complete gait cycles to achieve dynamically stable walking, as in [141]. The FPE equation evaluates the ground location where the total energy after swing foot impact is equal to the peak potential energy. If steps are taken before or after FPE location, post impact energy of the system causes the robot to fall. The solution to the FPE equation can also be used as a means of balance recovery in the event of disturbances. Utilizing the FPE as a measure of balance allowed them to create dynamically balanced gait cycles in the presence of external disturbances, including gait initiation and termination, without any precalculated trajectories [141].

There are two limitations to this approach, first, the theory assumes that the legs are massless and it only considers the 2D dynamics in the sagittal plane. Secondly, the derivation of FPE is built on the very simple compass model.

2.1.7.8. Limit Cycle Walking

According to Hobbelen [73], the main problem with tightly controlling the ZMP around a nominal trajectory, such that the robot is attempting to maintain instantaneous stability throughout the walk cycle, is that it places unnecessary constraints on the gait and high system control demands. Another method is called 'Limit Cycle Walking' where walking can be obtained without *any effort*, that is, sustained local stability requires extra actuation and tight feedback.

Hobbelen formally defines the relatively new paradigm `Limit Cycle Walking' as:

Limit Cycle Walking is a nominally periodic sequence of steps that is stable as a whole but not locally stable at every instant in time.

Hobbelen continues, 'with nominally periodic sequence of steps we mean that the intended walking motion (in the ideal case without disturbances) is a series of exact repetitions of a closed trajectory in state space (a limit cycle)'. As the trajectory is not locally stable at every time instant, the conventional trajectory control necessity of making all points on the trajectory attracted to their local neighbourhood in state space is removed. The motion is stable as a whole because neighbouring trajectories approach the nominal trajectory over the course of multiple steps. This type of stability is called `cyclic stability' or `orbital stability' [142] and is stable by damping out disturbances over time as opposed to relying on tight controller feedback.

Hobbelen explains that conventional controllers treat violent step-to-step transitions as unwanted disturbances that are deviations from a target trajectory that have to be rejected. Limit Cycle Walking increases speed range and it uses the violent step-to-step transitions at high speed as a way to create stability instead of having to fight them as disturbances. As walk speed increases, these induced deviations grow in size and occur at a higher frequency, that in turn increases the demands on the trajectory controller. This then limits a walk speed to the control bandwidth. The control bandwidth limitation on walking speed does not exist in Limit Cycle Walking as it does not depend on continuous stabilizing control to reject deviations. Conversely, step to step transitions are the main source of stabilization and the `bandwidth' of this stabilizing effect naturally increases with increasing walk cycle speed. Furthermore, Limit Cycle Walking is necessary for large disturbance rejection. Large disturbance rejection in walking is not possible with the use of high feedback gains and sustained local stability as it creates unsuitably high actuation torques that will eventually lead to a saturation of actuators and loss of contact between the feet and the floor, violating the condition of local stability. Limit Cycle Walking uses lower actuation torques and does not depend on full contact between the feet and the floor. The key premise is that the gait is allowed to reject disturbances over an extended amount of time, as long as falling is avoided. All passive walkers can be classified as Limit Cycle Walkers.

Hobbelen criticizes Honda's ASIMO application of Target ZMP Control and claims it 'is a recognition of the fact that Limit Cycle Walking is necessary for increasing disturbance rejection'. Hobbelen feels that ASIMO’s tight trajectory tracking is 'excessive and unnecessary' and recommends 'a reduction of this constraint altogether'.

Limit Cycle Walkers have two ways to increase disturbance rejection:

(1) Minimizing the divergence of motion throughout a step. Example: use of arced feet instead of point or flat feet.
(2) Increasing the stabilizing effect of step to step transitions. Example: foot placement adjustments.

The design in this thesis will introduce a walk phase varying joint stiffness method to correct gait errors both throughout dual support and stance phases. The walk design in this thesis also falls into the classification of a Limit Cycle Walker.

2.1.7.9. Central Pattern Generators

Another established method of determining reference trajectories (joint), has come from studying vertebrate animals [143][144][145]. Grillner [146] suggested that there are a number of neuronal modules in the spinal cord, which he referred to as central pattern generators (CPGs), that can be made to produce a rhythmic output. Central pattern generators are circuits which are able to produce periodic signals in a self-contained way, that is, without having any periodic input [89]. Collectively, these neural networks are able to produce rhythmic movements, such as swimming, walking, and hopping, even if isolated from the brain [147]. The use of CPGs in bipedal locomotion was first inspired by Taga's work [148][149] where he demonstrated the use of CPG's for robust and adaptive gait generation with a high-DOF robot [150]. Advantages of using CPG models in robotics are that they drastically reduce the computational overhead, mathematical complexity and dimensionality of the walking control problem existing in some other models. This approach also does not require detailed information of the system dynamics to achieve walking. At a high level, gait synthesis can be formed by simply generating a few low level control signals [151]. In order to build systems with similar properties to neural oscillators found in animals, several mathematical models have been proposed [152][153][154].

Muscle activity is achieved by combination of a few basic patterns, each occurs at a different phase of the gait cycle. The weights of distribution to different muscles may change as a function of the gait requirements, allowing for adaptive control (walking at different speeds, running, walking under load, direction changes etc.). Co-ordination body parts emerges from the coupling of these rhythmic patterns [155].

Matsuoka proposed a mathematical model of CPGs and demonstrated that the combination of simple neural models can generate the neural activities for biped locomotion [156]. This model has been applied to several biped simulations [150], as well as physical robots [157]. One of the difficulties in the application of the CPG model to robots is determining the weights of neural connections [89]. Consequently genetic algorithms have often been used to solve this problem [158] [159]. The types of neural networks used for gait creation vary from multilayer perceptrons, recurrent neural networks [160] to Cerebellar Model Arithmetic Controllers (CMAC) [161]. An example of development in this area is Miller's hierarchical control strategy which combines simple pattern oscillators and learning with CMAC neural networks [162][163].

A few key examples of bipedal locomotion with CPGs are [164][165][166].

2.1.7.10. Models Conclusion

The models and methods reviewed are the popular paradigms currently applied to the state of the art research. This is by no means a complete list, there are many other techniques or hybrid systems such as geometric reduction [167][168], hybrid zero dynamics [169][170], optimization methods [171] and other less common approaches. There are also two other major classifications of methods that are large areas of research unto themselves; Modelling from Empirical Data (Motion Capture) [172][173][174][175] and Machine Learning.

2.1.8. Lateral Motion

A very interesting and surprising aspect to the state of current humanoid robotics gait design is that little if any focus is placed on motion in the frontal plane. This includes every document cited in this thesis. It is a property of motion that does not come up in discussion with any significance other than timing issues. In Mah's [176] 1994 pattern analysis of gait he states that 'most movements in gait occur in the sagittal plane' which was an *opinion* based on observations. It is quite likely this has been taken to heart by many researchers since. This may also be a product of developing research from the bottom up, beginning with the most simple models and slowly adding more complexity or by starting with 2D and attempting to perfect a 2D problem before expanding to include further dimension. The problem here is that simple models that are verified are only strictly valid given the assumptions made

or self imposed constraints. Adding more complex aspects to a problem or model may or may not prove earlier assumptions to be invalid. In the case of humanoid robots, the case can be made that this does occur. To illustrate this point we can observe the evolution of design from both simple walkers to anthropomorphic (addition of torso) and from 2D to 3D.

Bipedal motion research first began using platforms called 'walkers'. These were machines without legs (earlier without knees). Some examples (Figures 13 – 16) demonstrate this evolution. Figures 13 and 14 show the redundant leg method used to eliminate the impact of the lateral plane evolving from point feet to forward rolling soles. Figure 15 then shows the inclusion of wide based feet to provide stability and then further expansion in Figure 16 to use round lateral soles to generate smooth lateral rocking motions. These are all different methods that allow researchers to apply their models to hardware platform while continuing to work in the sagittal plane only.

Figure 13: [177] Point foot redundant leg walker

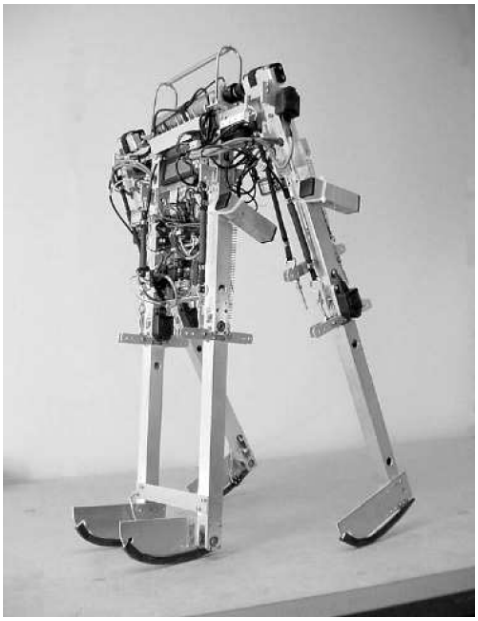


Figure 14: [177] Rolling foot redundant leg walker

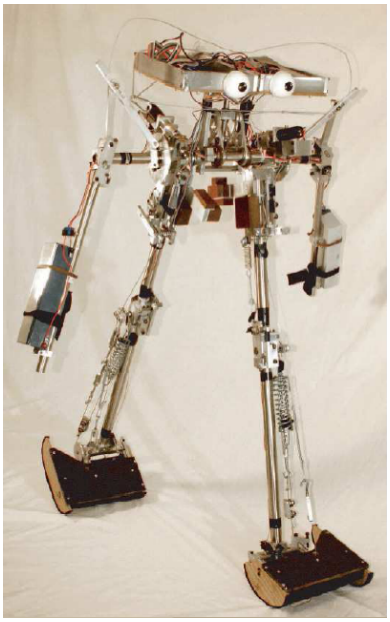


Figure 15: [177] Wide foot walker

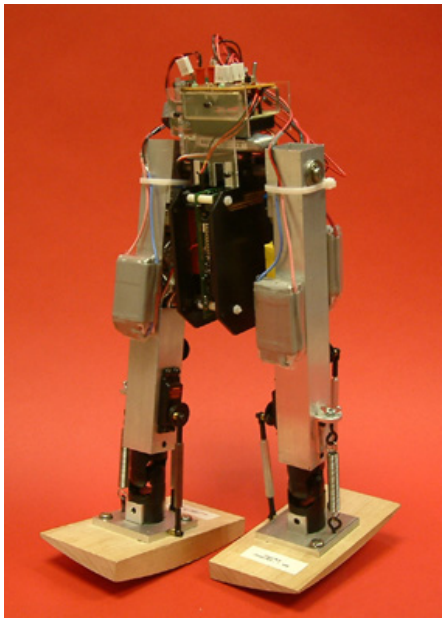


Figure 16: [178] Rolling sole walker

Kuo's highly influential work entitled 'Stabilization of Lateral Motion in Passive Dynamic Walking'[72] looked at the lateral motion during this era. He proposed various lateral stabilization strategies (Figure 17) consisting of ankle torque, angular momentum about the hips, theoretical torso stabilization or step width modification. This was of course based on the simplest walker model and one without knees. These compensation strategies are also defined by Roffer [40] as; CoM Balancing, Rotation Balancing, Phase Balancing, and Step-Size Balancing.

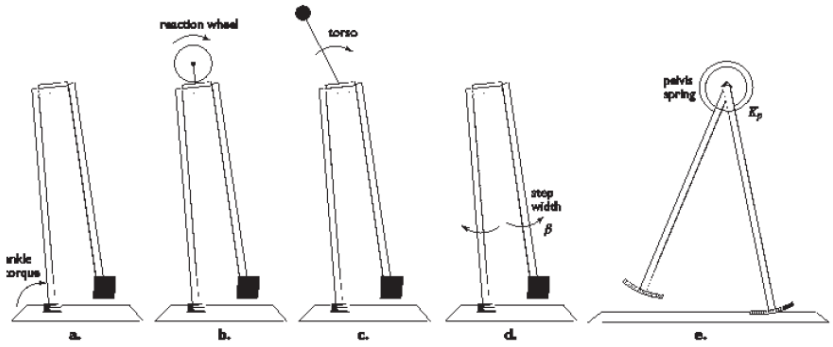


Figure 17: [72] Five possible stabilization methods. Ankle torque (a), reaction wheel (b), and torso motion (c) all exert control over trajectory of roll motion. Lateral step width control (d) indirectly affects roll motion, as does torsional spring (e) mounted at hip.

The best way to get an overview of what happened next historically, is to watch all of the RoboCup final matches for every humanoid league year including the the SPL Nao league. It is quite clear that in the early humanoid days the torso method (c) in Figure 17 was applied quite literally to imply counter balancing. It was most common to observe early anthropomorphic robots roll the torso in opposition to the swing foot. This of course lead to very top heavy motion that is very unstable, though technically able to locomote slowly. What can be observed from that point up to the present day is a convergence around keeping the torso vertically upright and shifting it laterally, along with the commonly published LIPM.

With the inclusion of torsos on humanoid robots, this early simple model has not been readdressed. Given the limit of DOFs, the early walkers could do nothing but rock back and forth like an inverted pendulum. The inclusion of more DOF, such as knees and ankle/hip roll joints, has allowed for greater range of motion. Yet the range of motion due to underactuated constraints has not been revisited in reference to premises of the early simple models.

The linear inverted pendulum model was not derived until the 2D nonlinear model was expanded to 3D, as the 3D version was found to be very unstable. Linearizing the problem gave the appearance of reduced instability without addressing it directly, while also violating the original premise that the 'bent-knee-bent-hip model is proven to be energetically and trajectorially inaccurate' from which the 2D nonlinear model was founded on. It is very self defeating in a circular logical sense.

An example of recent work performed on the Nao robot by Alcaraz-Jimenez et al. [179], entitled "Lateral Disturbance Rejection for the Nao Robot' (which received the Best Paper Award of the RoboCup Symposium 2012), also cites 'the importance of lateral stability is often overlooked' though in their work they take the approach of closed loop stability applied to the LIPM, by adding a lateral COM position offset without first addressing the validity of the open loop target trajectory.

These models that make their way from simulation to hardware are often allowed to just rock back and forth on their own with little design methodology or they are suspended from moving laterally by a support rod. Humanoid motion exist simultaneously in both planes and therefore motion in both dimensions will have to be stable. No matter how stable the motion is in one plane, the overall stability is going to be affected if one plane is unstable or marginally stable. In 2D simulation this will not be an issue but for a 3D hardware platform that is not stabilized with a boom, stability analysis and results are going to be marred.

A recent example of current work being performed solely in the sagittal plane and being greatly affected by overlooking the frontal plane is Ames' [180] 'Dynamically Stable Bipedal Robotic Walking with NAO via Human-Inspired Hybrid Zero Dynamics', where motion capture was used to create a gait that has the look of a human walk and is stable. Ames posted a video [181] of this walk showing multiple points of view. As seen from the front it rocks back and forth in the same manner as the early highly underactuated walkers. Figure 18 is a screen captured frame that shows the walk reaching its stability limit at the sway zenith and is teetering on the outside edge of the foot. Here the stability of the presented work is being limited not by anything done in the sagittal plane, but by ignoring motion in the frontal. With any increase in walk cycle speed this robot will fall.

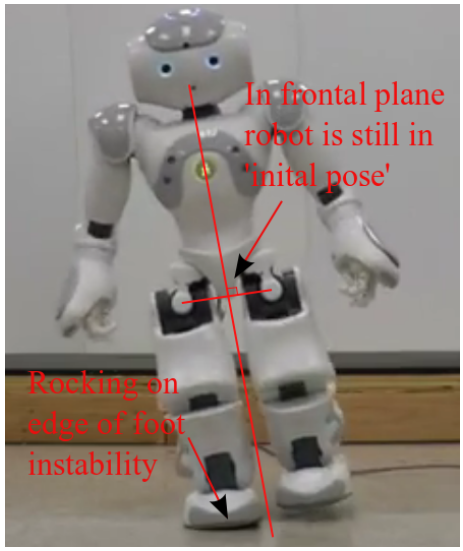


Figure 18: [182] Top heavy unstable rocking motion

Typically robots are not built with flexible spines which would allow for twisting motions. It is now being done in animation. For example, in his gait design, Boulic [183] uses motion capture to model a fully actuated spine (Figure 19 and 20). In Figure 21 we can see that lateral motion and rotations in the hips have little impact on the upper torso. This change in shape also shifts the COM position to some degree. With rigid spine robots, leaning inward would be a decent approximation.

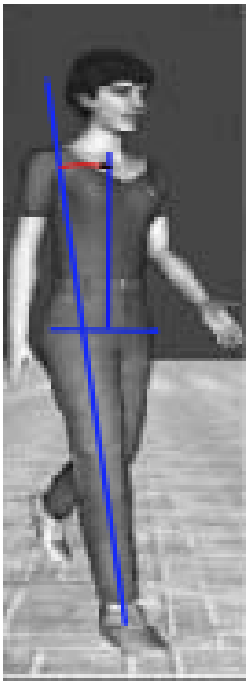


Figure 19: [183] Motion Capture

Figure 20: [183] Simulation

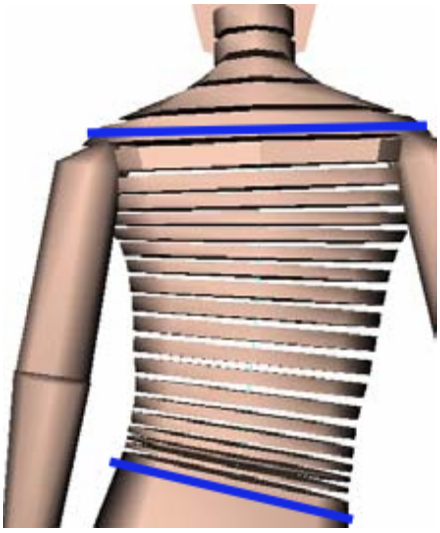


Figure 21: [183] Flexible Spine

Given the COM trajectory and 'bent-knee-bent-hip' energy requirements conflict between the Inverted Pendulum and LIP models, there seems to be an assumption that because the COM does not move up and down too much, as shown by empirical data, then no part of the robot's torso can. Indeed as seen from the side, in rigid simple models the hips and torso move together. In the frontal plane, this is not the case at all. It is quite simple to lean inward to compensate for the vertical displacement of the hip to maintain a more level COM trajectory (Figures 22 - 25). This then allows for the stance leg to be fully straightened during stance phase (Inverted Pendulum Model) and for the COM not to bob up and down between stances (LIPM or bent-knee-bent-hip model theory from empirical data).

Most robots are not yet manufactured with much degree of flexibility in the spine. Most are like the Nao having a completely rigid spine. Given this restriction it is still possible to bridge the conflict between the inverted pendulum and bent-knee-bent-hip model (LIPM), while preserving the premisses from each one and not creating the errors each one cites in the other. By viewing motion in the frontal plane it is possible to do this (straiten the supporting leg during stance phase while maintaining the COM trajectory height between steps) as shown in Figure 25.

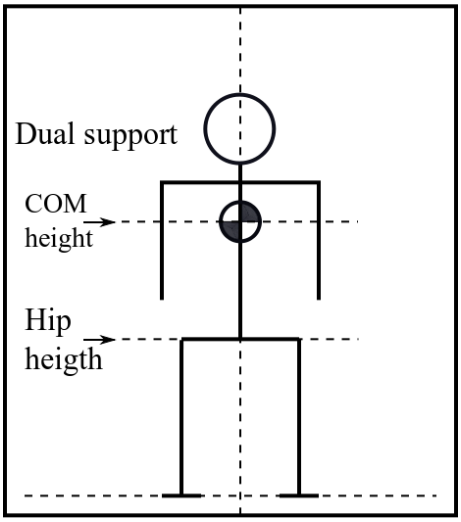


Figure 22: Dual Support Stance

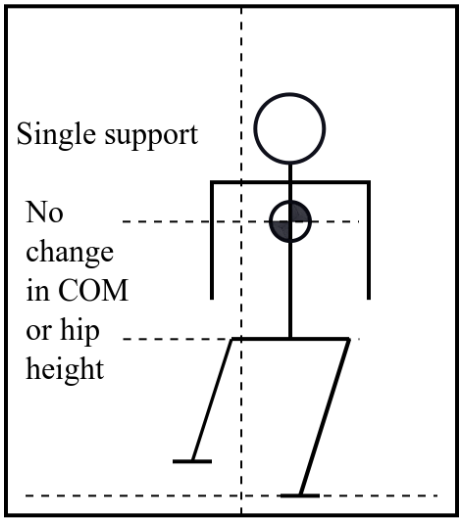


Figure 23: Linear Inverted Pendulum

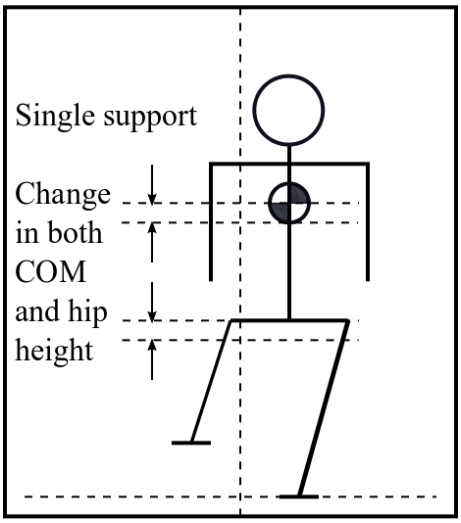


Figure 24: Inverted Pendulum

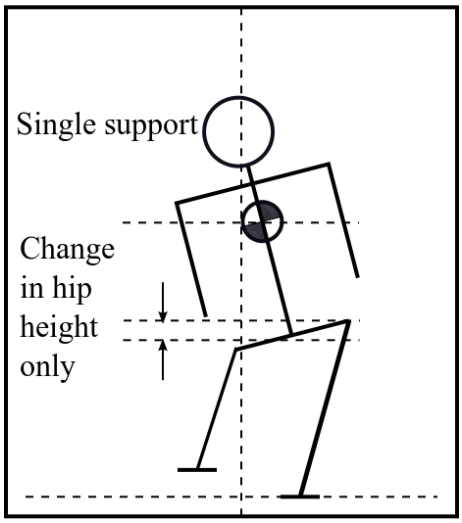


Figure 25: Empirical Data Matching

Essentially, this is a simultaneous application of two of the possible stabilization methods in Kuo's proposal. Lateral shifting of the COM and angular momentum created around the COM. Increasing walk cycle speed amplifies the lateral ZMP motion, so the faster the walk, the more laterally unstable a robot becomes. By employing a method such as this, the robot should be able to move more quickly relative to gaits with no lateral gait compensation.

Biomechanical research also supports the importance of lateral motion with Cairns et al. [184] noting 'The racewalkers in this study displayed an increased amount of hip flexion during the swing phase of racewalking, which was significantly greater at the competitive velocity as compared to the other gait conditions'.

To demonstrate the significance of the frontal plane, the design of the gait in this thesis will be done in the frontal plane only (exception being forward foot motion) by introducing the *Non-Inverted Pendulum Frontal Model*.

2.1.9. Walk Cycle Frequency

Stride length and stride frequency have been thoroughly documented [185]. Stride length is a larger contributing factor to velocity than stride frequency [186]. As velocity increases, stride length also increases while stride frequency stays relatively the same [187][186][188]. However, the limit to stride length's influence on velocity is emphasized when reaching maximum velocities. Attempting to reach top velocities shows increases in stride frequency rather than increases in stride length which holds relatively constant [189][187][186]. Essentially, once a reasonable kinematic stride length limit has been reached, the only way to increase velocity further is to move faster.

In the course of their medical research of elderly gait characteristics, Barak et al. [190] found that as velocity increases (step length) there is a proportional increase in hip sway. This makes sense as the ZMP spends more time in the region of the supporting foot, more time is purchased for the swing foot to travel a greater distance at a fixed speed. They also found that as walk cycle frequency increases, the sway magnitude decreases, demonstrating that stepping more quickly in the sagittal plane also increases the speed of motion in the frontal plane, as would be expected for synchronized timing in the motion planes.

Helbostad et al. [191] found in similar studies that there is a relationship between both lateral trunk accelerations and step width with walking speed. Though it is difficult to determine if they mean speed of motion or velocity. As far as application to the Nao robot is concerned, the feet are so wide that it is not possible to bring them much closer together than they are in the robots initialized position without risking them making contact with each other.

It is interesting that lateral motion has not been addressed significantly in humanoid robotics, since, as the walk cycle frequency increases, the resultant lateral ZMP position increases for a given sway width. Increasing speed amplifies lateral instability, and therefore, it is worth considering gait influences on lateral momentum, as increasing speed will inevitably lead to a fall to the side regardless of how well motion in the sagittal plane is developed.

2.1.10. Stiffness on the Nao Robot (Holding Torque)

Despite the Nao robot being strictly a position controlled robot, it does come with a joint stiffness setting. This is a unitless value, not an actual torque reference point. It presumably acts as an output 'trim dial' to reduce the holding torque between 0 and 100% of it maximum value. Walking

motions with firm rigid torques poorly reject disturbances or inherent gait design flaws. In the first RoboCup SPL Nao league (2008), Kulk designed an open-loop walk that keeps the stiffness of the joints as low as possible to both conserve energy and to increase the stability of the walk [192]. The Nao basically could not move around effectively with the packaged Aldebaran walk engine without reducing the stiffness from its maximum value, nor did one lowered value for each joint work well either. Kulk heuristically determined a set of values for each joint that gave the gait more of an elastic nature. The following version 2 of the Aldebaran walk required further empirical tuning as the version 1 values did not port over well to the modified gait design. With the introduction of the 3rd version of the Aldebaran walk, a single stiffness value could be used. A value of 100% did work relative to earlier velocities, but reducing it allowed for faster walking and significant energy savings.

Stiffness in literature generally relates to spring stiffness, something which has been studied in vertebrates and humans in particular. Fareley [193] reports that 'the stiffness of the leg spring remains nearly the same at all speeds in a variety of animals including humans (though the stiffness can be changed with different gaits)'. Whereas the results of others such as Seyfarth et al. [194] demonstrates that humans do change their leg stiffness at different speeds.

DC servo motors poorly emulate spring-damper system properties. There is little to no reference-able in depth analysis of holding torque in regards to position controlled robots. If there was, it would very much be a platform dependant property. In the RoboCup Standard Platform League, also very little attention has been paid to the subject of holding torque. Either way, if it is changed it appears to be something that is held constant for a given walk speed, with no one having attempted to vary the holding torque during the walk cycle using a position controlled robot other than Kulk [192].

One very interesting piece of work from the Wrighteagle group (Xue et al. [195]) is an attempt to model the Nao position servo motors as elastic joints via the unit-less stiffness control input. They are attempting to gain force control over the joint by doing so. This could provide the option of creating a hybrid position-force control walk engine.

2.1.11. Turning

Turning is not addressed in this thesis as it is a problem all of its own and only small angular changes are required in the run up to a kick (the kick itself is designed to compensate for angles). Turning in the walk designed here simply makes use of adjusting the foot yaw, though it is assumed that a proper omnidirectional high speed walk would have to spend unequal periods during the walk cycle on each foot (inside and outside the turn) and is likely to have to bank inward somewhat to compensate for the centripetal forces on tight turns.

2.1.12. Stability

There are two definitions of humanoid stability outlined by Pratt [133]:

Definition 1 (Fall): *When a point on the biped, other than a point on the feet of the biped, touches the ground*.

This is very forgiving and is thus the most widely applied criteria of stability, 'not falling'.

Turning now to the question of whether the biped *will* fall, the state space of the dynamics of the biped can be considered. Taking the robots dynamics in the general form of $\dot{x} = f(x,t)$, where $x \in \Re^N$ is the state vector. We define a subset $F \subset \Re^N$ which includes all configurations of the robot for which some part of the robot other than the feet is touching the ground. This then allows us to create a basin of attraction of *F*, which we will call the “Basin of Fall", that defines all of the states of the robot that eventually lead to a fall. So, first we define a Basin of Fall:

Definition (Basin of Fall). *Subset of state space that leads to a fall.*

$$B \subset \Re^N, x(t) \in B \Rightarrow \exists \Delta t \geq 0 \; s.t. \; x(t+\Delta t) \in F$$

This then allows us to use a second definition of stability:

Definition 2 (Stable). *A biped is stable if and only if the state of the robot is not inside the Basin of Fall*.[133]

Unfortunately the basin of fall subspace is quite impractical to map out with real hardware.

2.1.13. Assessment

Full physics developed simulation environments provide a lot of detail that can be readily captured by mere virtue of being a generated environment. Researchers working with simulated models have a lot of detail readily available. Real robots in the real world are quite the opposite. Generally, walking gait quality is in the eye of the beholder, humanoid robot developers will make most of their stability assessments by observing the robot walking. It is a highly dimensional problem space and with some experience, developers can visually observe what properties may be creating

instability, or, extrapolate what would happen if parameter x was altered. Testing with instruments typically is a project unto itself to build a test environment analytically or empirically. For example, instantaneous torso orientation measurements are challenging. Torso orientation will not provide ground contact information. Setting up a motion capture system is a big job for measuring isolated properties. The test space is also formidably large and a true test of stability is to test a robot against various weights approaching from all angles, at various velocities, contacting the robot over all of the many body surface points, during all instances of the walk cycle phase [73]. No one does this as it would be very expensive, numerically (in simulation) as well as experimentally. That said, there is large degree of safety provided in robotics research as the assessment criteria is vague to nonexistent with the definition of stable creating a very large grey zone with systems operating within this range laying at extreme ends being the difference between 'good' and 'bad'. 'Avoiding a fall' is a legitimate measure of stability but one with limited practical value [73].

Honest qualitative measures may be the best to capture the overall nature of stability. There are many parameters and abstract considerations to a walking gait, one or more will be the limiting factor in performance or stability, therefore, making the best measure the forthcoming nature of authors, as 'very stable' may also just mean 'walks extremely slowly'. Even when doing so it still results in quantifying one aspect to the motion and nothing else. Quite often when results are presented it is what is not being said that is more significant than what is. Being self critical may be the fairest assessment of all.

That said, there are means of measuring the properties of some models applied to hardware platforms.

2.1.13.1. Froude Number

To provide a decent comparison between different walking systems that can vary considerable in construction size, proportions and weight a Froude number [73] is used. Froude number *Fr*, which is the walking speed divided by the square root of gravity times leg length:

$$Fr = \frac{v}{\sqrt{gl}}$$

2.1.13.2. Efficiency

The energy efficiency of bipedal gait is quantified by the specific cost of transport (c_t) [73]. This dimensionless number gives the amount of energy that the biped uses per distance travelled per weight of the walker:

$$c_t = \frac{Used\ energy}{Weight \cdot Distance\ travelled}$$

To make a fair comparison between walking systems, cost of transport should be determined using similar walking speeds as determined by the Froude number.

2.1.13.3. Poincare Return Maps

Cyclic stability of a Limit Cycle Walker in particular is analyzed by observing its motion on a step-to-step basis. One step is considered as a function or `mapping' from the walker's state at a definite point within the motion of a step (for instance the moment just after heel strike) to the walker's state at the same point in the next step [73]. This mapping is generally called a Poincare map in nonlinear dynamics and the definite point within the motion is defined by the intersection of the motion with the Poincare section. With regard to walking, the mapping was termed the `stride function' by McGeer [3].

If a walking gait exhibits a cyclic pattern, then the biped realizing such a gait will return to the same state at the end of each cycle. One can consider a lower dimensional subspace *S* of the system state-space, the Poincare return map, which is intersected by the cyclic motion [196]. The intersection point is called a fixed point. For a stable periodic walking gait, the system state-trajectories return to approximately the same state after every step. One can make a Poincare map at, for example, every start of a step, just after heel strike. The stride function determines a transition between the current state and the state after one cycle. The stability of the cyclic motion can be analyzed by perturbing the initial fixed point and checking if it converges to the fixed point after a number of cycles [196].

2.1.13.4. ZMP Stability Margin

Limit cycle stability is only suitable for walks that are periodic in nature. Many designs such as ZMP based methods are not necessarily periodic. For such systems a 'ZMP Stability Margin' can be used. The ZMP Stability Margin is the distance from the ZMP to the nearest edge of the convex hull of the support polygon [133]. The stability margin is a measure that can also be used at run time to make

gait corrections.

2.1.14. Proposed walk design

Human walk is robust, not through sensory feed back mechanisms alone, but in our ability to intentionally modify our gait parameters to compensate for various types of footwear (stilettos, locked ankle ski boots, snow shoes, etc.) or to adjust for injury or load bearing (back pack, pulling a cart, balancing a sack of grain on ones head, etc.). Central pattern generator research has shown the existence of low level controllers in the spine, but we can also force gait characteristics by choice (walking with a squat, imitating the walk of others). If we assume every humans core walk engine is the same, then we must also assume it has evolved to be a very generalized one that can adapt given sustained constraints. What we can extract from this is that detailed parametrization and parameter flexibility is significant to multipurpose use. Our goal here is to create a gait that is fully parametrized given the available degrees of freedom the Nao has.

Various elements from different models will used be to design a hybrid system. Taking inspiration from central pattern generator concepts, the gait will be designed by moving the various degrees of freedom in the torso and feet driven by a central oscillator. The gait trajectories will be plotted using a novel frontal plane non-inverted pendulum (sway height / width, torso roll) to alleviate the conflict between the '6 determinants of gait' theory and the inverted pendulum model and provide greater speed by compensating for the lateral forces that occur when the walk cycle frequency is increased. A ZMP method will be used in simulation to observe the effect of the introduced parameters. The hardware version will use a novel walk cycle phase varying technique to provide open loop stability that dampens out disturbances over the complete walk cycle. The result will be an active dynamic limit cycle walk for an underactuated position controlled robot.

Motion in one plane does not interfere directly with motion in orthogonal planes, though the phase must be synchronized. As the research is performed in the frontal plane alone, the results could be applied in conjunction with many other common sagittal plane works to allow for increased walk cycle frequency and stability and be extended to consider closed loop methods.

2.1.15. Conclusion

In summary, there are a wide range of issues one should be aware of when developing or conducting research with humanoid robots. There are a number of different and sometimes conflicting models and various classes of hardware platforms and locomotion strategies. Project specific requirements will often dictate what is the appropriate design methodology, the most important being intended purpose. Different applications will require varying degrees of anthropomorphism, velocity, stability, and behaviour. For example being able to walk on non-level terrain, or accurate foot placements (randomly placed stepping stones). Position controlled robots offer a high degree of accuracy but do not perform as well with larger unexpected disturbances. Force-controlled robots adapt to larger disturbances but require more complex dynamic mass modelling and do not maintain precise control as well (though great progress has recently been made in force feedback control to grasp objects).

Sometimes the hardware itself is fixed, in that case the performance measure relative to the intended purpose may come with with its own inherent limitations. In the case of RoboCup, which is the application environment of this thesis, the position controlled Nao robot is fixed and for the foreseeable future is intended to walk as quickly as possible on level terrain, with falls being undesirable but not harmful to humans as could be the case in other application domains. Precision placed footsteps are only required when near the ball for a kick. The Nao robot is underactuated in the hips, has a rigid spine and irregular shaped flat feet. It also lacks toe actuation. In the Standard Platform League the LIPM is applied nearly universally, with a recent league convergence around the use of the bHuman team's walk engine. The predominant gait characteristic is walking while maintaining a vertically upright torso posture throughout the walk cycle. There are a few odd unique walks coming from teams such as Leipzig [197] that are developing their research by way of machine learning.

Our needs here, as far as walking is concerned, are: walking forward stably (never falling or even appearing close to a fall) at least as fast as the Aldebaran walk engine; strafing and minor orientation corrections, for as many steps as is typical during the 'line up' phase of kicking a ball in robot soccer, and to do so at a state of the art performance level. Also, to leave the design with further development compatibility (flexible parameters, engine is not dependant on any specific static values).

2.2. Humanoid Kicking

Kick kinematics has been studied to improve coaching performance optimization and to avoid injury [198]. This investigation may provide a useful reference to coaches or teachers for approaching optimal instep kick and may be applied to the area of robotic kicking. Ahmad Rasdan [199] used biomechanical analysis to identify the variables such as velocity, acceleration, distance and the angle of the knee that would influence the player's kicking force from image capture data. Chen et al. [177] observed that instep kicks with arm swaying caused higher ball velocity and body stretch.

Vasquez et al. [200] defines the kicking movement as a series of rotational movements where the aim is to produce, through the kinematic chain of body segments, high angular velocity to the foot. They define 6 stages of an instride kick; The approach, Plant-foot forces, Swing-limb loading, Hip flexion and knee extension, Foot contact, and Follow-through.

Properties of each stage include:

1. The approach
    - A diagonal approach produces greater swing-limb velocity for great ball speed.
    - A 45 degree angle approach produces the greatest peak velocity
    - Elite players take longer strides than novices as they approach the ball.
2. Plant-foot forces
    - There is a direct relationship between the direction that the plant foot faces and the direction in which the ball travels.
3. Swing-limb loading
    - Cocking of the kicking limbs.
    - The opposite arm to the kicking leg is raised and pointed in the kicking direction to counter balance the rotating body.
4. Hip flexion and knee extension
    - The thigh is swung forward and downward with a concomitant forward rotation of the lower leg.
    - As the forward thigh movement slows, the lower leg begins to accelerate.
    - The knee extensors then powerfully contract to swing the leg and foot forwards towards the ball.
5. Foot contact
    - At the point of contact 15% of the kinetic energy of the swing limb is transferred to the ball. The rest is dissipated to the hamstring muscle group to slow the limb down.
    - At the instant of the impact, the hip and knee are slightly flexed and the foot is moving upwards and forwards.
6. Follow-through
    - The follow-through serves two purposes; to keep the foot in contact with the ball for longer; and to guard against injury.
    - Longer contact time will maximize the transfer of momentum and thus increase its speed.

The role of the arms is described as:

- Essential in helping to maintain body balance.
- Arms extend horizontally during forward motion of the kicking leg. This helps to keep the center of mass over the supporting foot and also increases the inertia of the trunk.
- It also increases rotational resistance around the spine.
- As the kicking foot strikes the ball, the opposite arm moves forward and upward to help maintain balance.

They also point out the role of the moment arm forces where the greater distances from the center of the ball, to the center of the active joints in the kick, the longer the lever system is acting and results in a faster speed of kick (basic laws of orbiting bodies).

Regarding the matter of ball approach angle, Scurr [201] reports contradicting conclusions that altering an individual’s self-selected approach angle does not improve kicking accuracy or ball velocity during a kick, at least for recreational players assessed.

Plagenhoef [202] describes the optimal kicking motion as follows: “For performing an instep kick (last step of the kick), one should place the supporting foot at the side, and slightly behind, the ball. The kicking leg is first taken backwards swing with flexed knee. Then, in the following forward swing, the kicking leg should be carried out in a whip-like manner, i.e. forward rotation (both acceleration, and then, deceleration) should begin with the hip, followed by the rotations of knee and ankle. Such a timely control of the kicking leg is vital for the quality of instep kick”. This whip-like movement was confirmed by a German study [203] that analyzed the instep kick of professional football players.

Ussually, analysis of a kick is 2D and is done in sagittal plane only where the results are similar to the following Figure 26.

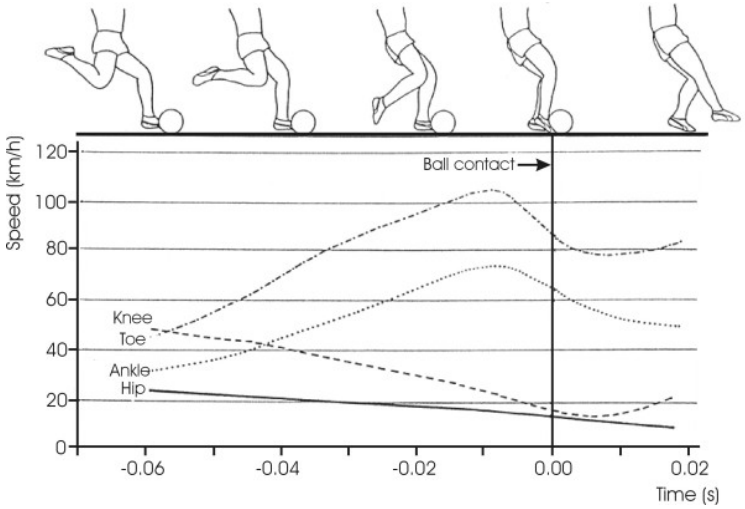


Figure 26: [203] Speed-time excursions of hip, knee, ankle and toe of a German national soccer player

A 2D study done by Roberts & Metcalfe [204] analyzed the transverse plane and made it clear that the path of the kicking foot was like an S-curve to the ball, not a straight line as was assumed (shown in Figure 27). They concluded that approaching a ball at an angle would benefit hip rotation, which was important for the lateral foot movement during a kick (it is still uncommon for cheaper robots to be constructed with the ability to rotate the hips relative to the shoulders).

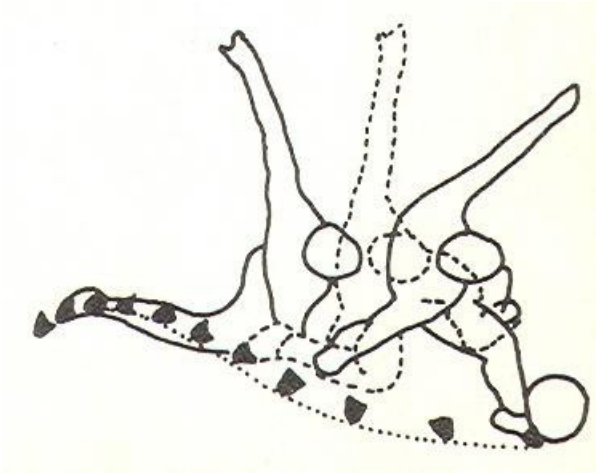


Figure 27: [204] Top view of the kicking foot movement during a maximal instep kick

Research has developed simple leg/ball velocity models but they prove to be of little practical use for coaching as analysis requires the use of motion capture technology [205]. However, the study of proper gait has been effective in reducing injury and increasing ball velocity [205]. For example, analysis has also proved that the maximal angle between thighs of the last step or the movement amplitude of the last step is highly correlated to the ball release speed [206]. 45-meter kicks have

greater movement amplitude at the hip as well as the last step length than those of 25-meter kicks. However, there is no difference existing in knee extension between both kicks.

There was little 3D analysis done prior to 2000 and it was questionable whether 2D analysis could describe full-body movement without losing important characteristics [205]. Early 3D analysis [207] did not show any significant difference in results as compared to the previous 2D work as they focused on capturing the motions of the legs only. The first full-body 3D analysis by Shan and his colleague [208], using more sophisticated motion capture, revealed as significant impact on kick performance by torso motion. They sited the significant characteristics of a kick being as follows (shown in Figure 28):

Formation of the dynamic tension arc involved:

1. Kick-side hip over-extension and knee flexion.
2. Trunk twist towards non-kick-side.
3. Non-kick-side shoulder extension and abduction.

The release of the kick leg arc consisted of:

1. Quasi whip-like control sequence of the kicking leg.
2. Upper trunk flexion and twist towards kick-side.
3. The non-kick-side shoulder flexion and abduction during kick.

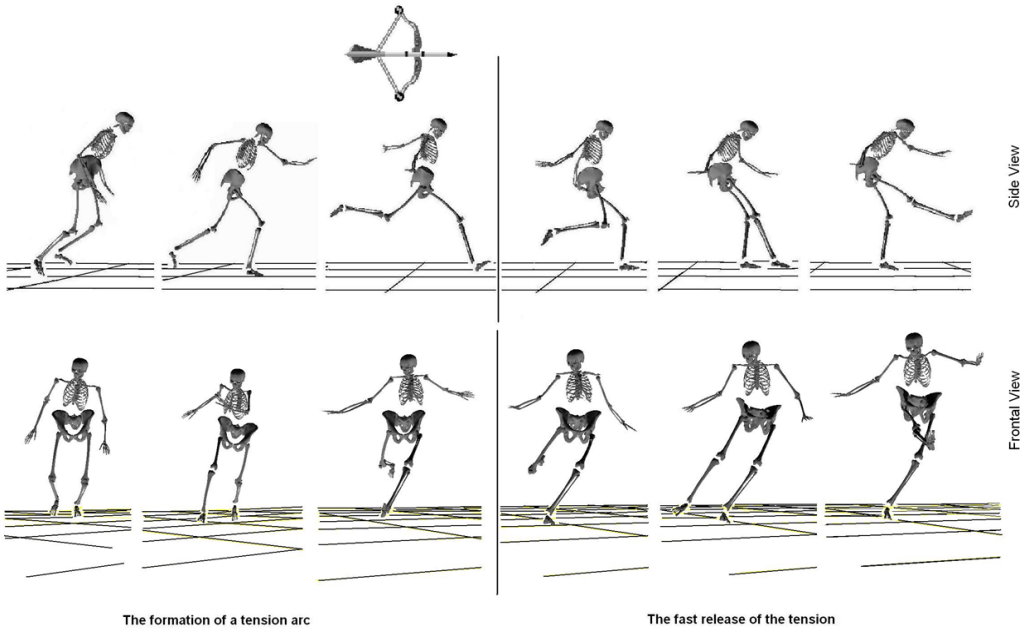


Figure 28: [208] The main feature of a maximal instep kick, a process consisting of tension arc formation and its fast release

The model that has been derived by Shan for kicking a soccer ball is:

$$V_{Ball} = 1.23 V_{Foot} + 2.72$$

Or:

$$V_{Ball} = \frac{V_{Foot} M (1+e)}{M+m}$$

Where V = velocity of ball and foot, respectively; M = effective striking mass of the leg; m = mass of the ball; and e = coefficient of restitution [209][210].

These simplified models were determined not to be sufficiently practical as they have to determine foot velocity by use motion capture technology [205]. Shan states that soccer kicking studies have revealed certain insight, scientific investigation so far has little impact on practitioners and that future studies should aim at developing user-friendly methods for evaluating training effect and improving learning efficiency for soccer athletes and coaches.

Another study by Ismail et al. [211] analyzed the kicking of professional a Malaysian footballer. They attempted to identify the variables such as velocity, acceleration, distance and the angle of the knee using motion capture as shown in Figure 29 and 30. Their results showed similar motion profiles but an interesting thing they found was that the highest optimum force was achieved from a three step run up to the ball.

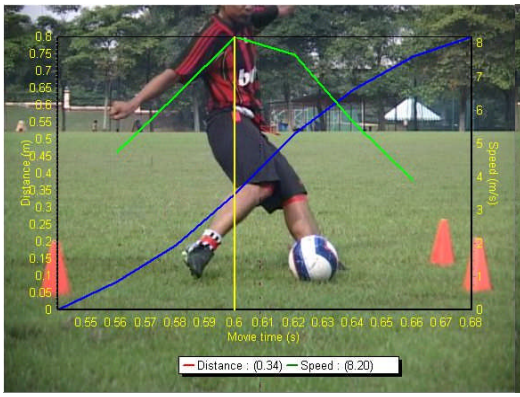


Figure 29: [211] Velocity and Angle vrs Time

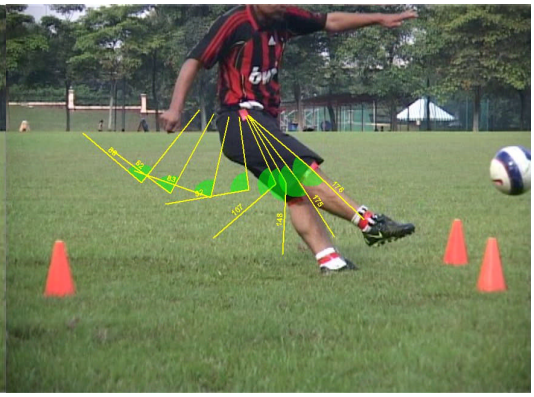


Figure 30: [211] Knee Angle of Kick

In this thesis, a kicking swing will be defined using cubic splines that captures these properties of a human kick.

2.3. The NAO Robot

Humanoid robots are humanoid in that they have a bipedal walking system, a torso, arms and a head. They differ however in the materials used to make them, their actuators, degrees of freedom, dimensions, as well as differences in available motive power, computational power and their vast array of equipped sensors. These difference will make for distinctiveness in their potential usefulness or abilities, and conversely, their limitations. Aside from all of the specifications in these areas (hard plastic frame, electric motors, Pentium processor etc.), the features that make the Aldebaran robot most distinct are two things: proportionally large odd shaped feet (see Figure 31), and the coupling of two of the hip rotation axis into a single joint known as the HipYawPitch (see Figure 32). As much as this wide foot area heavily impacts the dynamics, it wont be addressed here. What is of significance to this project is the mechanical coupling of the hips.

Humanoid robots are commonly found to have 3 DOF aligned with the Cartesian axes in each separate hip. The Nao robot has, in place of the Z rotation axis, an axis of rotation that is midway (45 degrees) between the Z and Y axis. It is this axis that allows the robot to perform a yaw rotation of the leg. Yaw motions of the leg are crucial for actions such as turning. This mechanical coupling means that the robot is under actuated, that is, it is not possible to independently specify the 6DOF of each foot with respect to the robot torso. In many cases, this limitation does not appear to be particularly significant when playing robot soccer. However, it prohibits more advanced actions such as ones which require a different yaw angle in each foot relative to the other, with respect to the torso frame.

A further consequence of the hip joint coupling is the fact that the robot is not symmetrical. In the hip Y-joint, one actuator will be driven by the main drive rod and the other will be geared to it. This means that there will be one extra mechanical connection on one side, and therefore more backlash. If symmetry is an issue, some backlash compensation method will be required.

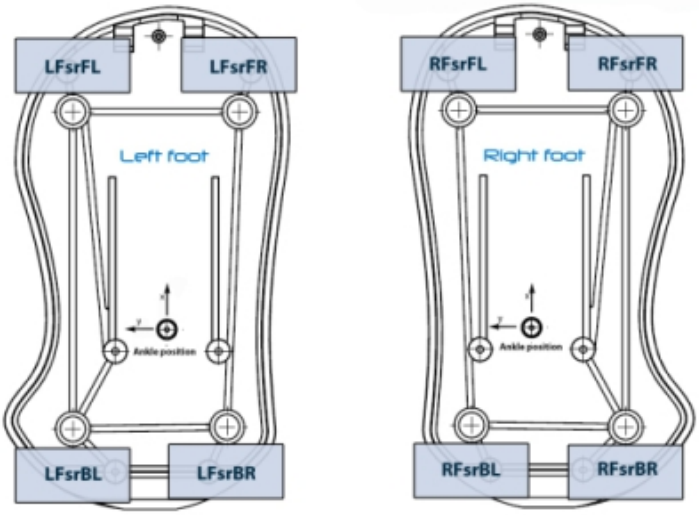


Figure 31: [212] Nao Foot Shape

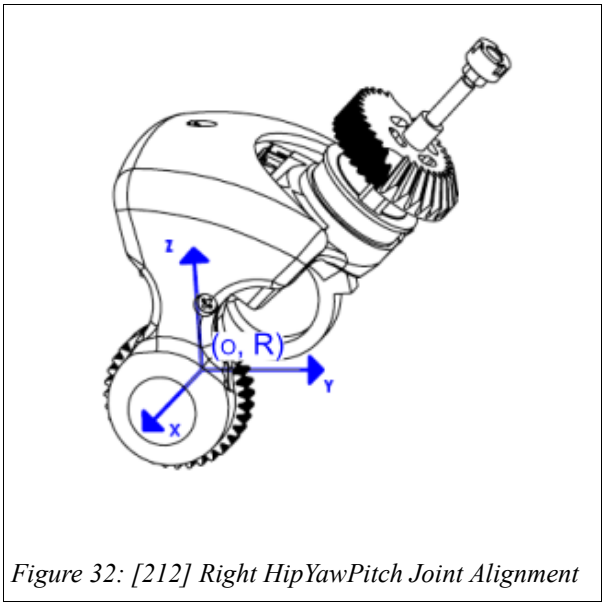


Figure 32: [212] Right HipYawPitch Joint Alignment

Useful kinematic properties of the Nao robot for the purposes of this thesis are attached in the appendices:

Appendix A - Nao's 22 Degrees of Freedom and Joint Names
Appendix B - Nao Links Measurements

3. Open Loop Humanoid Walking Engine for the Nao Robot

3.1. Objectives

The primary purpose of this chapter is to design and develop an open loop humanoid walking engine that takes control of the robot during the final high speed charge up to a full swing in-stride kick. This walk engine will be smoothly spliced between the Aldebaran walk and the kick to allow for realtime stepping control. The design features that are required are:

1. Be able to display a gait that is identical to the Aldebaran walk. (for smooth switching)

2. Preferably be able to walk faster than the Aldebaran walk, accelerating up to the kick.

Features 1 and 2 are contradictory in nature as we will assume that the Aldebaran system is already configured to operate at its maximum stable velocity for game performance. Squeezing any more velocity out of that specific gait will come at the cost of already marginalised stability. Therefore, the design will have to adopt a new gait with additional parameters to walk faster while maintaing stability. This then leads to a 3rd feature:

3. A walk engine design that is flexible and can be configured to perform different gaits.

To accomplish this, the robot is defined in every sample pose as being three parts, the torso and both feet, that are defined in their own 6 DOF's. Any gait then is the description of the cartesean spacial trajectories of these body parts.

4. The walk should permit small omnidirectional corrections during kick approach.

The gait design is able to simultaneously turn, strafe and walk straight. It is however not expected to take high speed sharp turns or other trajectories while challenging opposing robots to 'acquire' the ball. It is assumed that the behaviour system has determined it is time to directly charge the ball and kick it. Therefore, only the forward velocity is assesed for the given walk gait configuration. Omnidirectionality is verified at slower speed for the purpose of ensuring that the algorithm is robust and error free.

To acomplish these objectives, this chapter is broken down into the following major sections:

Section 3.2. Modelling. Developing any motion control system directly on hardware is potentially hazardous to the hardware itself as long as design errors may exist. A model is built to use as a development platform to verify control algorithms are robust and do not produce any obviously dangerous outputs. A modelled environment also has the advantage that some properties can be measured or viewed easily in simulation that can be very difficult with the real robot itself. Humanoid models are often torso centric in nature and as such are limited in use. This section outlines a world coordinate massless kinematic model with simulated gravity (that is, the support foot is kept level with the ground for all poses).

Section 3.3. Reverse engineering the Nao. To be able to merge a special motion or another walk with the Aldebran system, the Aldebaran gait must be known and duplicated. In this section joint sample recordings are taken from the robot and are fed into the Nao model to observe the gait characteristics in simulation.

Section 3.4. Walk engine design. This section covers the design of a dynamic (gait paramters that are modifyable during runtime) walk based on the central pattern generator concept (all body parts follow oscilating trajectory signals that are driven by a central clock). Existing open source walk engine code generally violate standard programming conventions such as code commenting and using self evident variable names. This makes porting difficult, especially if undisclosed dependancies and calibration methods exist. The algorithm presented is meant to be user friendly and keeps implementation in mind. First, each gait property employed is explained. The algorithm that follows can be coded into a programming language such as C++ in the order that it appears. It is not an abstract mathematical model, it is a complete algorithm, containg all the required maths that will perform a humanoid walk.

Section 3.5. ZMP. For the purposes of assessing stability in simulation, a massless ZMP calculation is covered. The ZMP is used as a design tool, but that is not to say that any part of the walk engine implicty uses a ZMP calculation. The walk engine developed is not a direct ZMP based method like many others, however, the ZMP is an abstract property of any walk that can be used to make assesment such as stability margin.

Section 3.6. Parameter simulation. The gait design methodolgy in this thesis differs significantly from the popular gait paradigm of focusing on the sagittal plane. The gait design here ignores the sagittal plane (aside from the obvious forward and reverse foot motions) and focusses primarly on the frontal plane. Two commonly overlooked parameters (tosro roll and height) are studied, and their effect in a feedforwad sense is analyzed in simulation, demonstrating proof of concept.

Section 3.7. Hardware stiffness. Finally, the hardware performance is verified and a time varying stiffness control method to improve stabilization is introduced.

These sections pertain to the components highlighted in blue in the system design (Figure 33):

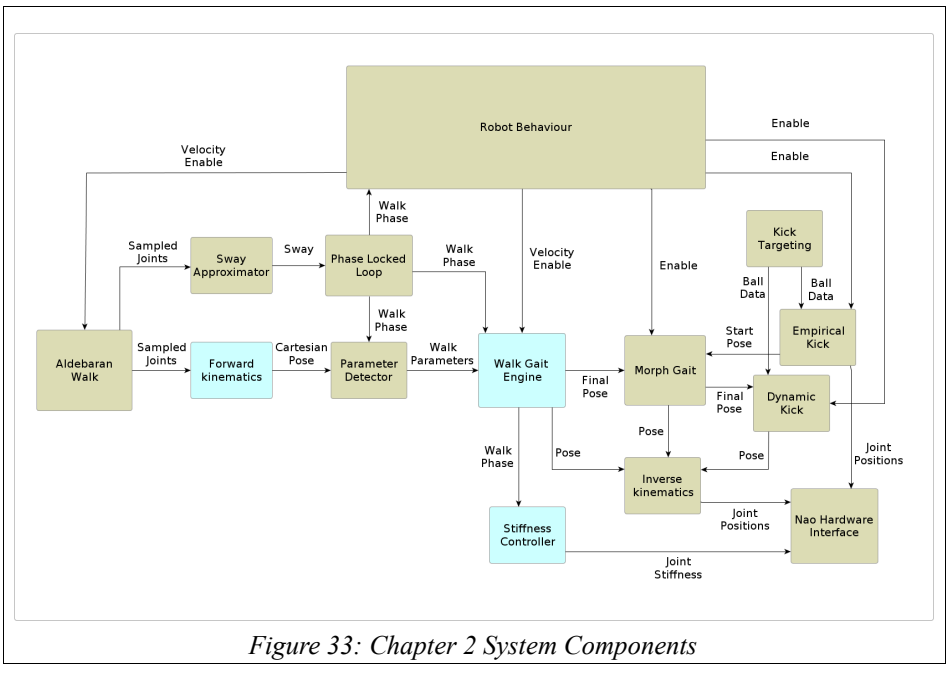


Figure 33: Chapter 2 System Components

3.2. Nao Robot Modelling in MATLAB

MATLAB was used to build a kinematic model of the NAO robot. This was done with two purposes in mind. First, to simulate the motion of the robot for analysis purposes using joint data sampled from the robot while walking. Secondly, to build a walking engine that produces its own joint values in simulation. The simulated robot was constructed within its own coordinate frame and then placed within a world coordinate frame to allow the robot to step forward simulating motion. This allows for plotting of spatial trajectories of significant body parts, feet, hip, COM, head, etc. The first version of the Aldebaran documents (no longer accessible) describes the robot kinematics as a modified Denavit-Hartenberg configuration. This configuration is where a robot that has all joint values equal to zero, defining a pose with the arms stretched out forward as shown in Figure 34. We will refer to this pose as a 'zeroed robot'. Another useful and commonly used pose is the 'initial position', seen in Figure 35.

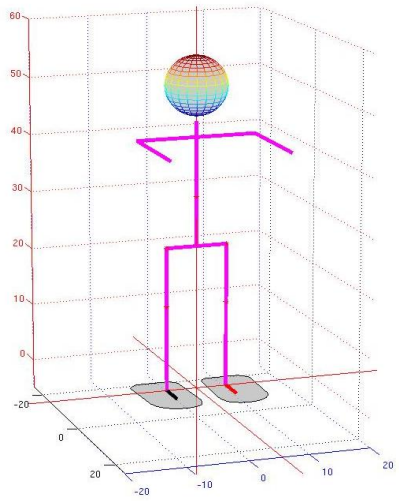


Figure 34: Robot 'Zeroed' Pose

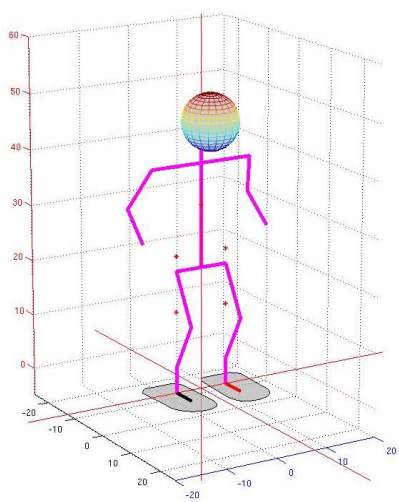


Figure 35: Robot 'Initial' Pose

As the orientation of the arm joints are somewhat nonintuitive, the rotation matrices required to manipulate each joint on the robot from the zeroed position are attached as Appendix D.

The robot model was also built with the following features for analysis purposes:

- A scaled camera with the visible region projected to the ground plane
- The robot's feet were modelled as the support polygon that encloses the irregular shape of the Nao's feet
- An inclinometer that traces the robots torso kinematic pitch and roll over time
- An inertial vector that projects the measured accelerometer data on to the ground plane

Simulation of a robot walking forward is achieved by recording the location of the center of one of the feet and accumulating a world position value. This accumulated value is then added to all the plotted data points to allow the robot to move forward in the world coordinate frame. By doing so the spacial trajectories of pertinent body parts can be plotted and analyzed for their characteristics. An example of a simulation of the modelled robot is shown in Figure 36.

The orientation of the robot is set by taking the robot's support foot and the ground to be parallel planes. For this purpose, the support foot is determined to be the closer foot in the Y dimension to the robot’s torso. Note that this definition will not work if the robot is tipped in the direction of the other foot far enough for it to make contact with the ground. Each foot has a vector that represents the X and Y dimensions that are attached to the foot center. The measured angular difference between these vectors and the world coordinate axis are used to level the robot by rotating all body data points around the foot center. By doing so, this introduces some error into the simulation in comparison to reality as it is quite visibly observable that the early versions of the Aldebaran walk had very poor

ground contact. However the goal here is not to asses the actual spatial trajectories of the body parts during the incidental hardware run that the data was captured from, but to determine the target trajectories of the Aldebaran designers. The simulation therefore represents what would have happened if the Aldebaran open loop walk performed as desired and the ground contact was flawless.

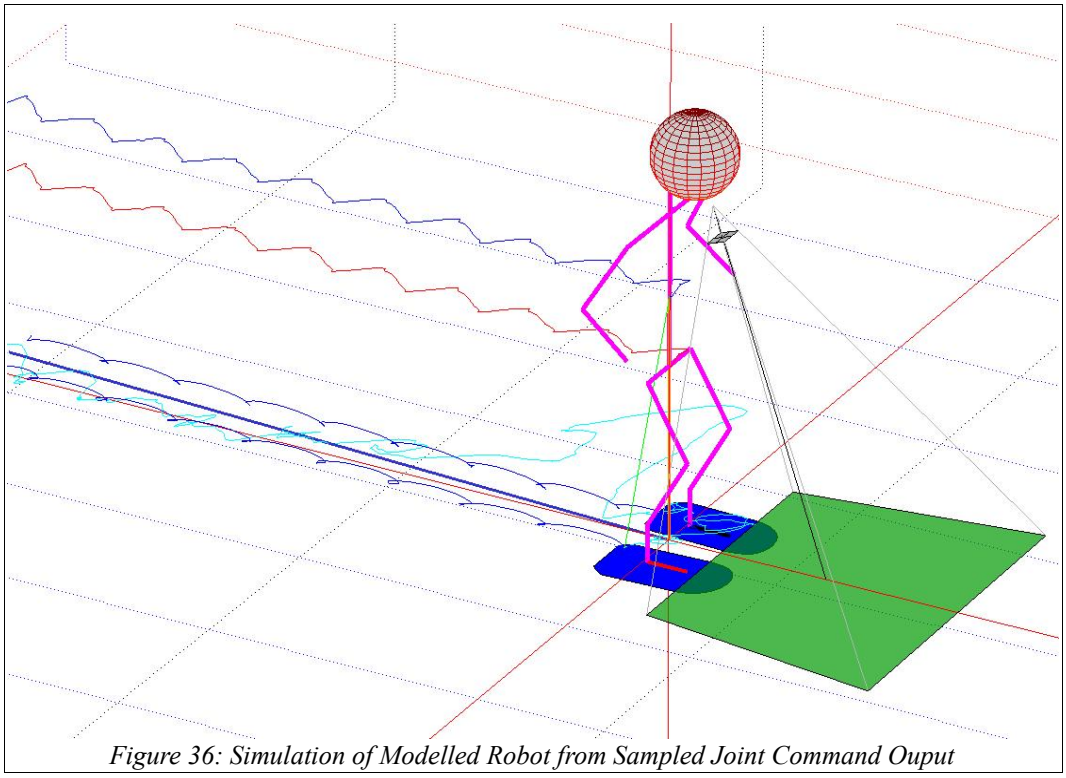


Figure 36: Simulation of Modelled Robot from Sampled Joint Command Ouput

3.2.1. Forward Kinematics

The following forward kinematics method was used to simulate the Nao robot given joint sample data:

Steps:

1. Define the zeroed robot body data points:
(Construction of kinematic model attached as Appendix C)

Result:

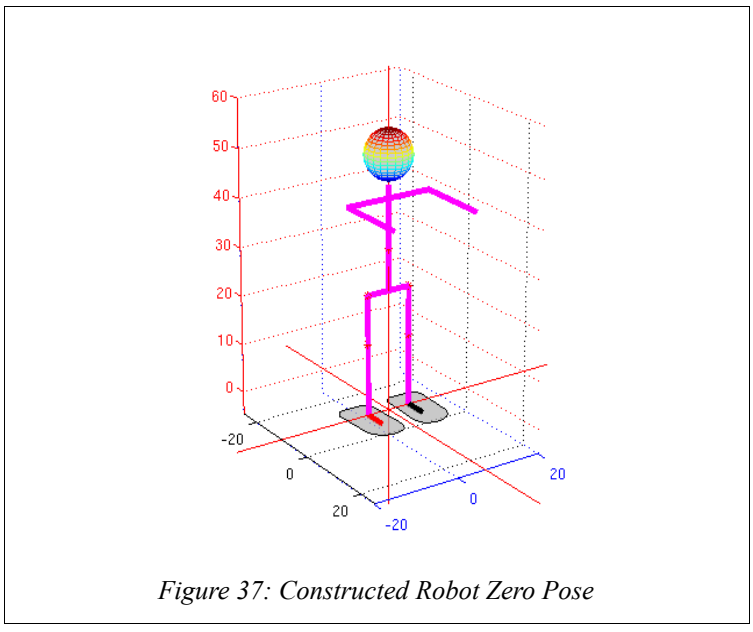


Figure 37: Constructed Robot Zero Pose

2. Record zero foot positions:

$$footLockL = LeftFootCenter \tag{1}$$

$$footLockR = RightFootCenter \tag{2}$$

3. Rotate all body data points:
(Only the legs are shown. The solution for the entire modelled robot is attached as Appendix I as a MATLAB script).

$$\begin{aligned} LeftAnkle &= R_{\phi y}(\theta_{KneePitch\,L})(LeftAnkle - LeftKnee) + LeftKnee \\ LeftAnkle &= R_{\phi y}(\theta_{HipPitch\,L})(LeftAnkle - LeftHip) \\ LeftAnkle &= R_{\phi x}(\theta_{HipRoll\,L})\,LeftAnkle \\ LeftAnkle &= R_{\phi yzL}(\theta_{HipYawPitch\,L})\,LeftAnkle + LeftHip \end{aligned} \tag{3}$$

$$\begin{aligned} RightAnkle &= R_{\phi y}(\theta_{KneePitch\,R})(RightAnkle - RightKnee) + RightKnee \\ RightAnkle &= R_{\phi y}(\theta_{HipPitch\,R})(RightAnkle - RightHip) \\ RightAnkle &= R_{\phi x}(\theta_{HipRoll\,R})\,RightAnkle \\ RightAnkle &= R_{\phi yzR}(-\theta_{HipYawPitch\,L})\,RightAnkle + RightHip \end{aligned} \tag{4}$$

$$\begin{aligned} LeftKnee &= R_{\phi y}(\theta_{HipPitch\,L})(LeftKnee - LeftHip) \\ LeftKnee &= R_{\phi x}(\theta_{HipRoll\,L})\,LeftKnee \\ LeftKnee &= R_{\phi yzL}(\theta_{HipYawPitch\,L})\,LeftKnee + LeftHip \end{aligned} \tag{5}$$

$$\begin{aligned} RightKnee &= R_{\phi y}(\theta_{HipPitch\,R})(RightKnee - RightHip) \\ RightKnee &= R_{\phi x}(\theta_{HipRoll\,R})\,RightKnee \\ RightKnee &= R_{\phi yzR}(-\theta_{HipYawPitch\,L})\,RightKnee + RightHip \end{aligned} \tag{6}$$

Result:

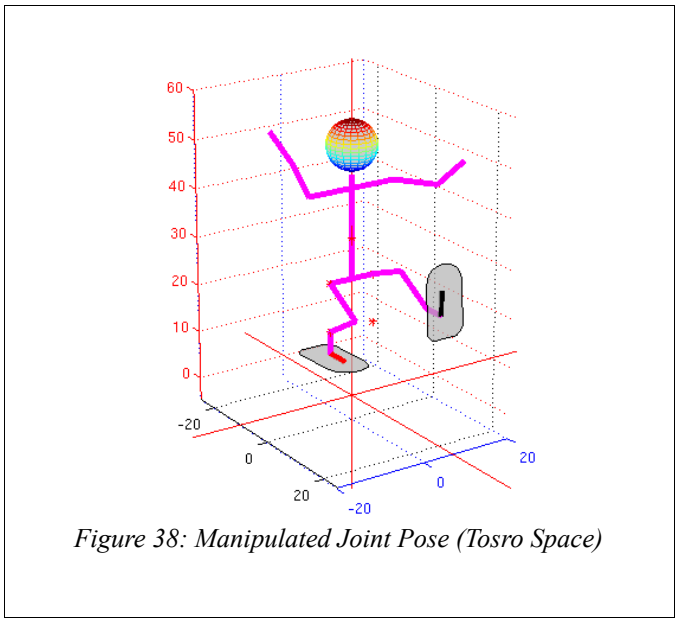


Figure 38: Manipulated Joint Pose (Tosro Space)

4. Set support mode

$$supportMode = \begin{Bmatrix} \text{Left} & : & \text{abs}(RightAnkle_y) & > & \text{abs}(LeftAnkle_y) \\ \text{Right} & : & \text{abs}(RightAnkle_y) & <= & \text{abs}(LeftAnkle_y) \end{Bmatrix} \quad (7)$$

5. Drop robot into support location:

$$TranslationVector = \begin{Bmatrix} \text{RfootLock - RightFootCenter} & : & \text{supportMode} & = & \text{right} \\ \text{LfootLock - LeftFootCenter} & : & \text{supportMode} & = & \text{left} \end{Bmatrix} \quad (8)$$

$$BodyDataPoints_{0 \to n} = BodyDataPoints_{0 \to n} + TranslationVector \quad (9)$$

Result:

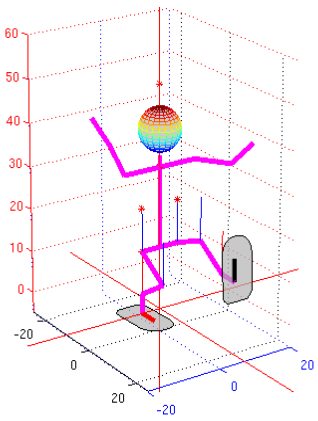


Figure 39: Translated to Locked Support Foot Location

6. Level on support foot:

For the purpose of analyzing the Aldebaran walk, ground contact is achieved by comparing the supporting foot normal with the world coordinate Z axis or 'field normal'. The angular difference found in terms of pitch and roll is then used to rotate the robot around the center of the supporting foot.

3.2.2. Inverse Kinematics

Connecting multiple joints together with a robotic manipulator (an origin and a target point) requires the use of an inverse kinematics method to solve for the manipulator's joint angles. In this case we are connecting the hip joints to the ankle locations via the leg and we require the leg joint angles to define an output pose. An iterative forward kinematics based method may be used or an exact geometrical solution. This project was begun with the use of an iterative algorithm to solve for the leg joints (excluding the ankle joints) provided by Micheal Quinlin [213]. This was sufficient for the more simple process of defining poses to take walking strides that involved no torso incline, no foot attitude or turning motion. Introducing a body roll parameter into the walk, or more complex foot orientations that would be required for kicking, meant that the inverse kinematics algorithm would have to be expanded to include the ankles. In the attempt to do so, numerous different geometrical solutions were worked out to satisfy the ankle problem that would always fail in some capacity if any body roll was introduced. Given the highly likely possibility that there was a simple bug in the implementation of the algorithm or a slight error in the maths, the only course left was to start at the beginning and map out the entire problem from top to bottom, replacing the iterative component with an exact solution until the error was found. As there was no bug in existence to be found, an entire

exact solution for full body pose inverse kinematics was inadvertently worked out. As hindsight is 20/20, it was established that there was no error in the mathematics after triple checking. The simple fact is, the problem can not be solved. The torso and the two feet separately can not be defined in their own six degrees of freedom to form any arbitrary pose. The offending constraint is the mechanical coupling in the HipYawPitch joint. It is possible to define any pose with any 6 DOF in the simulated robot, but the end product is always different values for the HipYawPitch Left and HipYawPitch Right whenever some body roll is introduced. This can not be physical realized on the actual hardware given both these joints are physically geared to the same drive motor. This then means something else must give way, it is not possible define to both a body roll and symmetrical foot yaw orientations.

This constraint raises a few issues in the realm of 'what is more important, body roll or foot orientation?'. That answer would seem to be application dependent. An erratic torso roll signal that is compensating for foot yaw changes while walking and turning would likely lead to stability problems when trying to define a specific value of orientation change per step. Also, further development of this system to include closed loop control would likely include body roll as a controlled variable. The closed loop system would have difficulty if it could not realize its correction values. However the development of an angled kick making use of inverse kinematics may be desired, a kick that lines up the swinging leg directly in line with the shot angle. Yielding some body roll in order to accomplish this kind of a kick could be sought after. But again, mixing such a swing with dynamic stability would create conflict. It was therefore decided that the body roll takes precedence over defining specific foot yaw values. Other than making said angled kick difficult, the only constraint placed on a walking system would be that a turning velocity could not be specifically set. For example a turning velocity of 10 degrees per step would result in a velocity near, but not exactly, 10 degrees, with the error magnitude being proportional to the body roll magnitude. This is favored as this should not disturb the robots continuity of motion and generate any upper body trajectory error leading to imbalance.

The final solution is to run the full body inverse kinematics algorithm, determine the supporting foot and then rerun the calculation for the leg that connects the airborne foot, first seeding it with the HipYawPitch value determined for the supporting leg (equal but opposite) or split the error between the two and run both legs calculations. In discussion with Jeff de Haas from the B-Human team, it was learned the B-Human team encountered the same obstacle and solved it by reiterating the legs in the same way. They solve the problem slightly differently by introducing a new parameter that is the ratio of favouritism between each leg. They can assign all of the error to one leg, or distribute it over both legs in accordance to the set ratio.

The exact solution worked out for this thesis is attached as the final appendix, Appendix L. It does differ though from what most RoboCup SPL teams will be using for the Nao robot. The solution is not for the torso coordinate system only but for the body as a whole (excluding arms).

3.3. Analysis of the Aldebaran Walking System

The approach taken to designing an open loop walk in this project was to model it after the one created by the robots manufacturers. The goal was to parametrize the motion of the robots limbs and to find mathematical functions that describe how these limbs move over time. The simulator built was used to reverse engineer the nature of signal trajectories being mapped by the Aldebaran walk engine. This was performed to consider whether these signals make sense and to identify periodic functions that may be used to represent them. In addition, it may be possible to consider alterations to these functions to improve performance.

3.3.1. Observations

The result of this analysis yielded the following significant observations:

1. As seen from above, the torso and COM of mass trajectories appear to be sawtooth functions with loops on the peaks. (Figure 40)

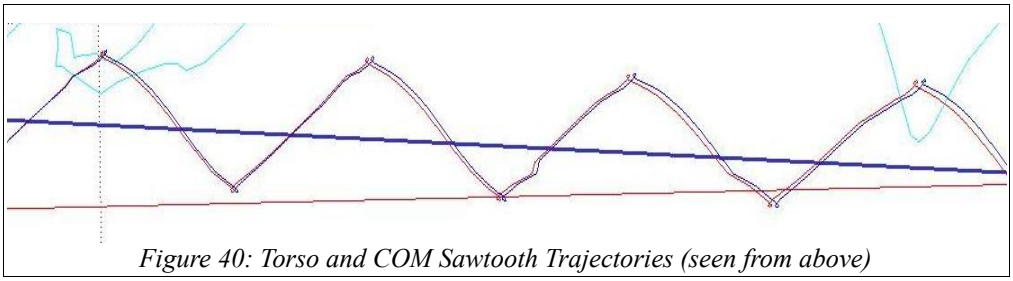


Figure 40: Torso and COM Sawtooth Trajectories (seen from above)

2. As seen from another perspective, there is an observable change (increase) in height of the robot very subtly over one stride that is quickly removed during the loop (drop). This change in height of the COM starts at 30.91 cm and rises to approximately 31.2 cm. For all intents and purposes, this signal can be considered 'flat' with some subtle increase in height while the robots torso moves laterally. (Figure 41)

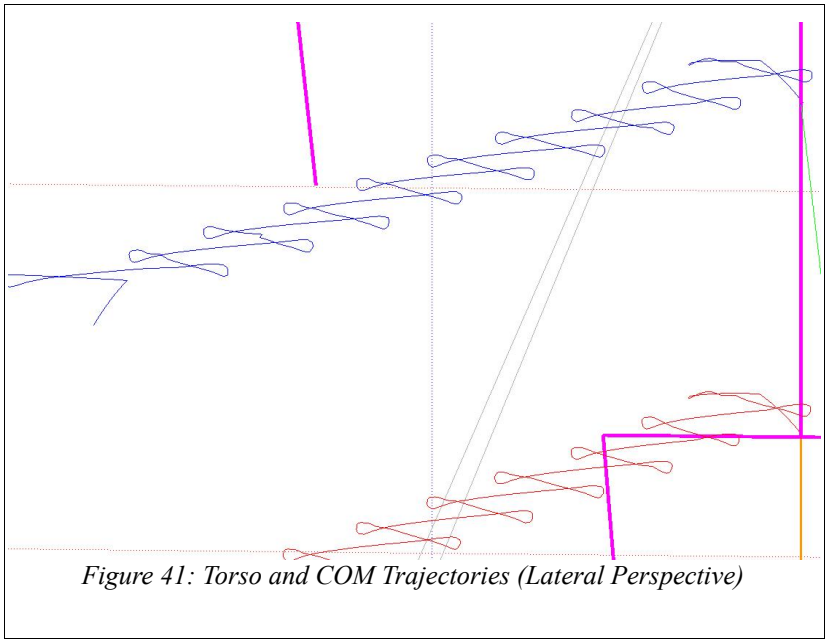


Figure 41: Torso and COM Trajectories (Lateral Perspective)

3. A torso kinematic inclinometer (Figure 42) shows little to no change in kinematic pitch and roll (degrees) over a 10 step walk. This was accomplished by subtracting the location of the hip center from the neck in the model producing a 3 dimensional vector that represents the torso attitude.

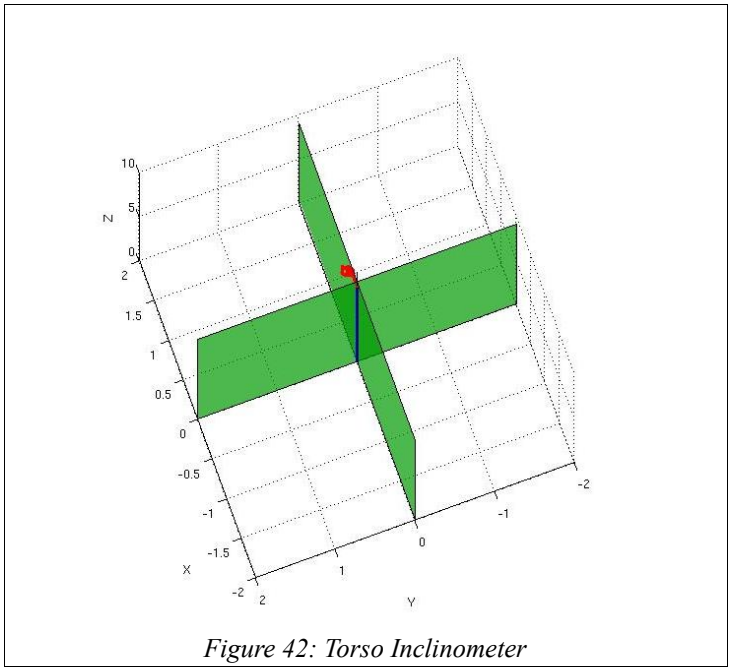


Figure 42: Torso Inclinometer

4. This differs somewhat from the inertial unit readings of the pitch and roll (degrees). This data taken from the same walk shows a build up of oscillations ultimately leading to a fall to the side which is typical of the first version of the Aldebaran walk. In this trial the robot fell to

the left side near the 7.5 second mark and was quickly caught and rebalanced for the remainder of the 10 steps. Though it is noisy, there is a discernible 10 degree peak to peak oscillation in both dimensions with similar periods. Combining these dimensions has the effect of generating an increasing orbital error in the walk gait around the z axis that quickly leads to a fall. (Figure 43)

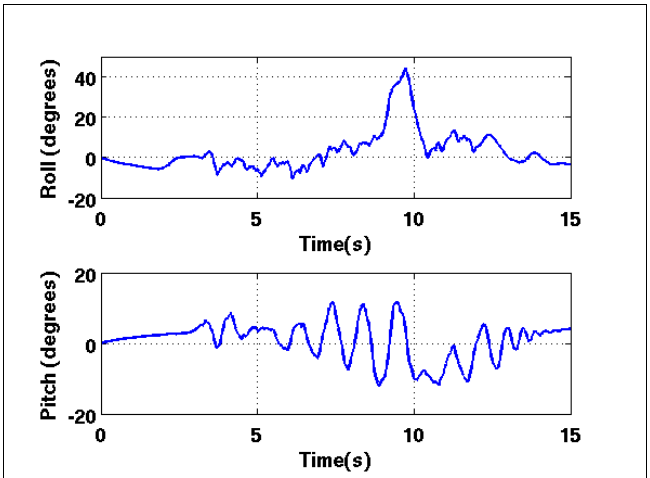


Figure 43: Nao Walk Sampled Inertial Unit Data

5. In Figure 44 these pitch and roll vectors are combined and projected to the ground plane (red signal) along with the measured sway (black). This projected inertial signal represents the ZMP location at each instant. The signal is simply generated by creating a vertical unit vector that represents gravity and attaching it to the location of the inertial unit inside the robot. This vector is then pitched and rolled according to the measured inertial unit reading and then extended to a point where it intersects the ground plane in the model. It is quite clear from this perspective that once this signal goes beyond the outside edge of the foot (limit of the support polygon), a fall occurs rapidly.

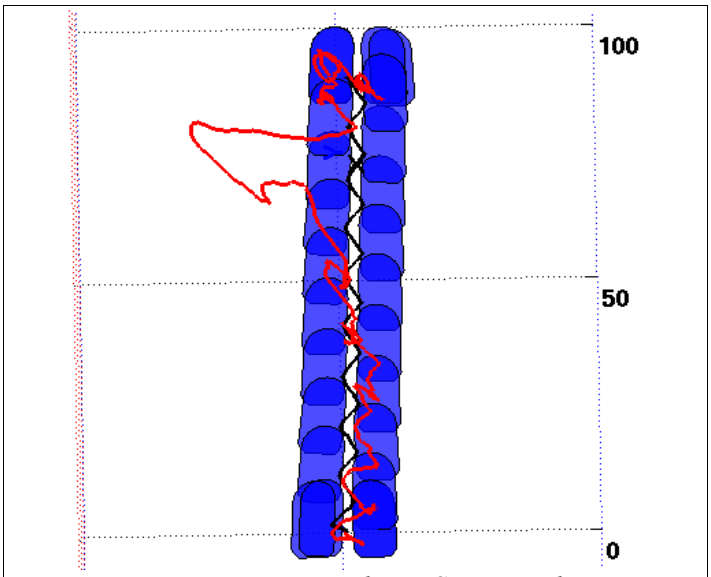


Figure 44: ZMP and Foot Step Record

Analysis of the foot trajectories are not required as the first version of the Aldebaran documentation identified them as cycloids.

3.3.2. Discussion

Saw tooth sway signal. It would appear from the sway signal observed that Aldebaran initially implemented a state machine nature of a walk controller. This is one that switches between support modes to shift body parts around as a sequence of events. That is: shift weight to a single support foot, move other foot forward, shift weight to other foot, move first foot forward etc. This differs from a coupled oscillators approach which does not include state switching behaviour. Though it is simple to implement a controller, the drawback to this method is the discontinuous nature of the spatial signal trajectories of the body parts at the peaks. This is not to say that state machines can not be used to implement continuous signals, but that it motivates the designers to break the gait itself down in rudimentary steps that are not fluid, as was the typically nature of early walking systems. This control function does not relate well to the properties of real moving objects where inertia will continue to propel the body forward despite the fact that the target trajectory has come to an abrupt end. This may have been factored into the design and the sway magnitude stops short in anticipation of the fact that there will be an overrun. The signal also shows an even spacing in data points, so there is no acceleration/deceleration attempted. However, a sine wave would inherently have these properties, it would be smooth and decelerate into the peaks and accelerate through the dual support phase. It would therefore lend itself more naturally to control of real moving objects.

Loops at sway peak. The loops at the peak are difficult to understand. As they are produced through observation derived from joint sample data, it is difficult to reverse engineer the intention of the walk designers. They could be intentional, some sort of method for dealing with the signal discontinuities at the peaks. They could be there for some inertial control purpose. Or perhaps they do not exist in the walk engine output and they are a result of damping introduced when the stiffness for the walk is calibrated. Either way, they are very likely to be undesirable. Orbital motions that develop around the Z axis while the robot is walking tend to amplify and lead to a fall. Perhaps the loops are intended to counter these orbital motions. However, they could have the opposite effect in an open loop design by introducing orbital motions if they had not yet occurred. It was therefore decided to remove them from the walk trajectory.

Flat torso trajectory (no change in height). This is something that is quite interesting. It is easy to observe humans bob up and down as they walk. That then begs the question, is this a desirable property of a humanoid walk? Closer observation reveals a human is at its highest point during single support phase and at the lowest during dual support. This too is in contrast to the inverted pendulum approach to defining signal trajectories. It is worth then exploring this idea, perhaps a non-inverted pendulum design is better. At a minimum it would have the properties of allowing for a longer step length if the hips were to drop at the step peak. Furthermore, raising during single support phase would allow for more clearance room for the swinging leg, that could have some effect on the inertial moment dynamics.

3.4. Design of a Central Pattern Generator for the Nao Robot

The walk developed for this project was designed so that it would be both suitable to be used on its own and, given the correct parameter settings, be able to reproduce the Aldebaran walk gait so that smooth switching between walking engines is possible. To be considered a suitable walk, the following design features were implemented.

3.4.1. Design Features

- Velocity control .
- Smooth acceleration .
- The ability to apply step changes at any point in the walk cycle.
- Omnidirectional.
- As fast as the current Aldebaran walk.
- Statically stable with passive disturbance rejection.
- Also to design a gait with the aim of the camera being a pivot point, with little or no translation. (Balance is achieved by swinging the hips outward).

Signal generators were developed in MATLAB to allow the modelled robot to walk forward with the desired properties, the strafing (side stepping) and turning control were implemented on the actual robot once the system was ported to the robot. The parameters used to describe the humanoid gait suited for the Nao robot are listed in Table 1 along with values empirically found that can walk as fast as the Aldebaran top velocity. The maths and control algorithms required to describe this parametrized motion are presented in the following subsection 3.4.2.. These functions define the six degrees of freedom desired for each body part, the torso and both feet. A full body inverse kinematics algorithm then connects these parts together and yields the joint angles.

Parameter/Variable	Symbol Used	Value
Walk Cycle Phase	ϕ_W	$f(\phi_I)$
Cycle Phase Increment	ϕ_I	0.0085104 rad/samp (Nao top speed)
Torso Pitch	Tr_β	2°
Torso Roll	Tr_γ	$f(Hp_Y)$
Hip Translation	$Hp_{(X,Y,Z)}$	
Forward Displacement	Hp_X	0 cm
Sway	Hp_Y	2.3 cm
Height	Hp_Z	22.3 cm
Stride Length	St_L	0 ↔ 12 cm
Step Width	St_W	5 cm + $St_{W\Delta}$
Side Step Length	St_L	Max 5 cm
Step Height	St_H	1.8 cm
Foot Pitch	Ft_β	0°
Foot Roll	Ft_γ	0°
Left Foot Lift Time	$Ft_{L\ Lift}$	0.375
Left Foot Land Time	$Ft_{L\ Land}$	0.6
Right Foot Lift Time	$Ft_{R\ Lift}$	0.875
Right Foot Land Time	$Ft_{R\ Land}$	0.1
Left Airborne Cycle Phase	ϕ_{LA}	$f(\phi_W)$
Left Contact Cycle Phase	ϕ_{LC}	$f(\phi_W)$
Right Airborne Cycle Phase	ϕ_{RA}	$f(\phi_W)$
Right Contact Cycle Phase	ϕ_{RC}	$f(\phi_W)$
Joint Stiffness	J_τ	$f(\phi_W)$ range 0.7 ↔ 1.0

Table 1: Walk Parameter and Variable Symbols with Aldebaran Matching Values

3.4.2. Function Generators and Control Algorithm Maths

To describe the motion of a complex machine such as a humanoid robot at a level higher than mere playback of recorded joint values, analytical functions or algorithms are required. The parameters presented in Table 1 set the operating limits of these functions. The incidental values for any given sample during the walk cycle are calculated using the following methods.

3.4.2.1. The Walk Cycle Phase Generator

The modulated walk cycle phase (ϕ_W) is the core of the walking engine and drives all other functions. It is incremented by the phase increment (ϕ_I) and therefore ϕ_I controls the speed of the walk. Velocity is set by changing either ϕ_I or the Stride Length (St_L) or both.

Phase increment:

$$\phi_W^* = \phi_W + \phi_I \tag{10}$$

Walk cycle modulation:

$$\phi_W = \phi_W^* \bmod 1 \tag{11}$$

3.4.2.2. Torso Pitch

The torso pitch is set statically to a slight forward lean, a value settled upon through empirical tuning. Two degrees is used to here to match that of the Aldebaran walk.

$$Tr_\beta = \frac{\pi}{90} rads \tag{12}$$

3.4.2.3. Torso Roll

The torso roll and the hip height are generated with a non-inverted pendulum signal that is synchronized with the walk cycle. They are both calculated such that the camera focal point can follow a trajectory that does not sway in hope of aiding the vision processing. This is done with trigonometric roll of the torso in synchronization with the hip sway. This may place undesirable constraints on the walk parameter configuration and therefore a roll reduction factor (R_r) is introduced to dampen the amount of roll if desired. The torso length T_ℓ is the height difference between the camera focal point and the hip center while the robot is zeroed. From Figure 45, simple trigonometry gives:

$$Tr_\gamma = \frac{\pi}{2} - acos\left(\frac{Hp_Y}{T_\ell}\right) R_r \tag{13}$$

3.4.2.4. Hip Forward Displacement

The forward displacement of the hip center relative to the robot global origin is also set to a static value. The walk designed here has no forward offset.

$$Hp_x = 0 \tag{14}$$

3.4.2.5. Hip Sway

Here a sine wave is chosen for its smooth properties and the fact that it begins and ends at the origin in one cycle.

$$Hp_{y(i)} = Hp_y \sin(2\pi\phi_W) \tag{15}$$

3.4.2.6. Hip Height

Again, the hip height is generated with a non-inverted pendulum signal that is synchronized with the walk cycle. (see Figure 45)

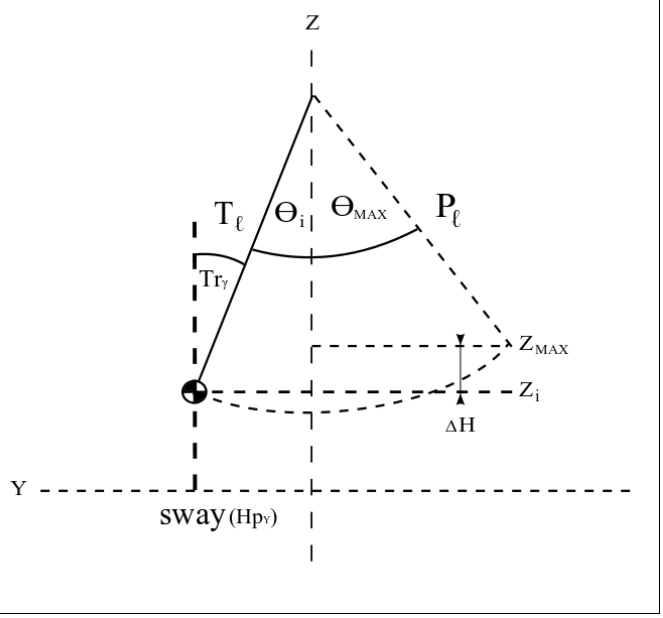


Figure 45: Pendulum Model for Torso Attitude and Signal Trajectory Mapping

Solution:

$$\theta_{MAX} = asin(\frac{Hp_Y}{P_\ell}) \tag{16}$$

$$\theta_{[n]} = \theta_{MAX} \sin(2\pi\phi_W) \tag{17}$$

$$z_{MAX} = \cos(\theta_{MAX}) P_\ell \tag{18}$$

$$z_{[n]} = \cos(\theta_{[n]}) P_\ell \tag{19}$$

$$H_\Delta = z_{[n]} - z_{MAX} \tag{20}$$

$$Hp_{Z[n]} = Hp_Z - H_\Delta \tag{21}$$

3.4.2.7. Body Part Synchronization and Timing

There are several factors involved in determining the location of the feet. They can be summarized as follows:

1. Initial leading foot choice.
2. Foot lift off and landing timing (switching between forward and reverse signals).
3. Acceleration related issues (splay).
4. Step error correction.
5. Option to use in-stride kick.
6. Side stepping and turning control algorithms.

When building the walk engine it was these issues that were most problematic. We will discuss first the solution to these problems, followed by the algorithm and equations presented in subsection 3.4.2.9.. This way the algorithm will be more readily understandable and the equations can be presented together in the order they should be processed.

Initial leading foot choice. This is a somewhat minor point, but important practically. To begin walking either foot can step out forward first. This just becomes a bit of a practical coding issue that can lead to more work than is desired for a beta version of a walking engine. There is the potential for fairly complex decision logic in the code. For example, the walk sway function is a sine function that is leading right. To lead left an inverted signal (-sine) would be required. A number of the other sinusoidal functions would require this switch. Therefore for simplicity, and speed of development, the walk developed here is leading right only. This just leaves a single switch to implement in the right foot cycloid that halves the first stride length. The only issue this would create in a completely general walking system is when attempting to use the walking system as a single 'step-to' position when trying to step to the left. It is still achievable, however, the robot would simply rock back and forth to get supported on to the right leg first before stepping and it takes more time than is necessary to achieve such a motion.

Foot lift off and landing timing. Each body part follows a signal that is periodic in nature. To synchronize the motion they are bound together as coupled oscillators. Two methods were devised to control the footing timing in relation to the torso motion during the walk cycle. The first method was an attempt to duplicate the fluid motion of a human that is governed by sinusoidal signals that are proportional to the torso motion and thus would require little, if any, optimization. The second method uses fixed liftoff and landing times and is more robotic in nature but easier to work with. Both methods work as follows:

Following some inspiration gleaned from observing toddlers learning to walk, it would appear that humans place their foot in the direction they are moving. This is best observed watching a toddler trying to stay in one position while concentrating on some other task. Before they learn to stand still you can see they are constantly moving on the spot, stepping in the direction they are falling. You can try this yourself, try to step into a location that is not the direction you are moving and see what happens. As the governing signal used in this walking engine is a sine function, there is only one point in the signal that points in the direction of the next foot fall position. To duplicate this observed behaviour, we set the foot landing timing as the point at which the tangent to the torso trajectory in world coordinates is in line with the next zenith of its own trajectory. Conversely, the foot liftoff is the tangent that is in line with the preceding sway zenith (this also makes sense as it is the one point in the signal that has the most power to tug the body mass forward away from where it is planted – the body inertial vector pointing directly away from the last foot fall).

Instead of computing derivatives at each sample point and determining if the slope coincides with a forward extrapolated position, the tan function itself was scaled (* 0.636) and used as a trigger point. The derivative at this 0.636 intersection point is very close to, but not exactly in line with the zenith. This tangent function has the nice property of always being proportional to the step width and length even if they are changed while in motion. Figure 46 shows a plot of these intersecting functions in action over an accelerating walk trajectory. The figure demonstrates that the slope line calculated from the intersection sample of both the sine function and the tan function * 0.636 using the backwards difference approximation draws a line to the zenith point of the sine function (red line for the next zenith, black line for the preceding one). This is the method that is used by the model in the MATLAB walking simulation.

Simulated Timing Method:

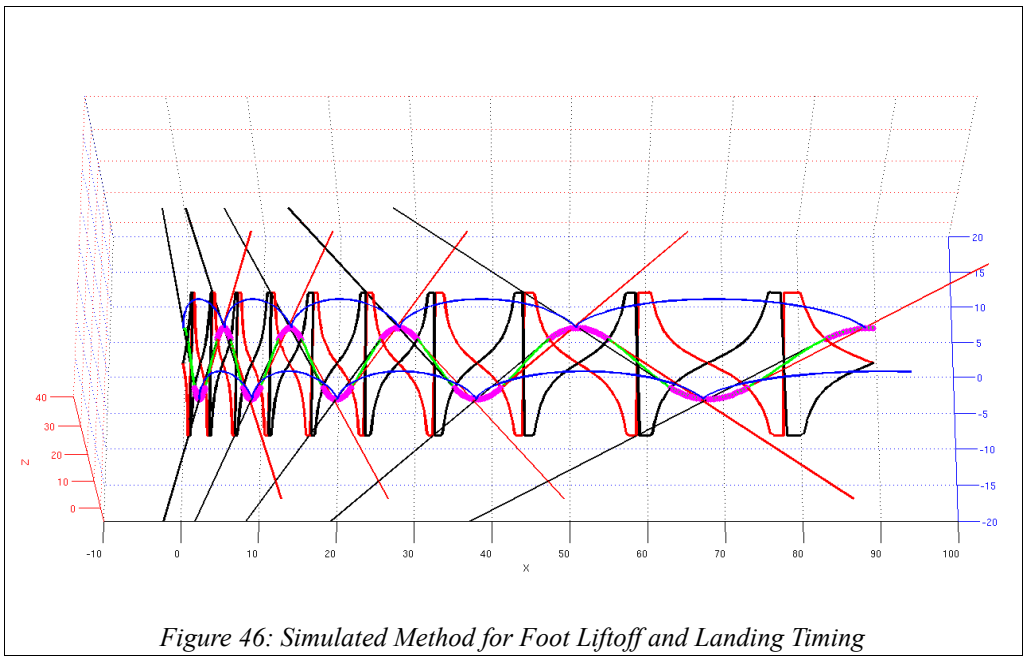


Figure 46: Simulated Method for Foot Liftoff and Landing Timing

Figure 47 shows a close up of these functions over one period. The mauve pearl region is the single support phase and the green line region is the dual support phase. The switching occurs at the intersection of the clipped tangent functions (black and red).

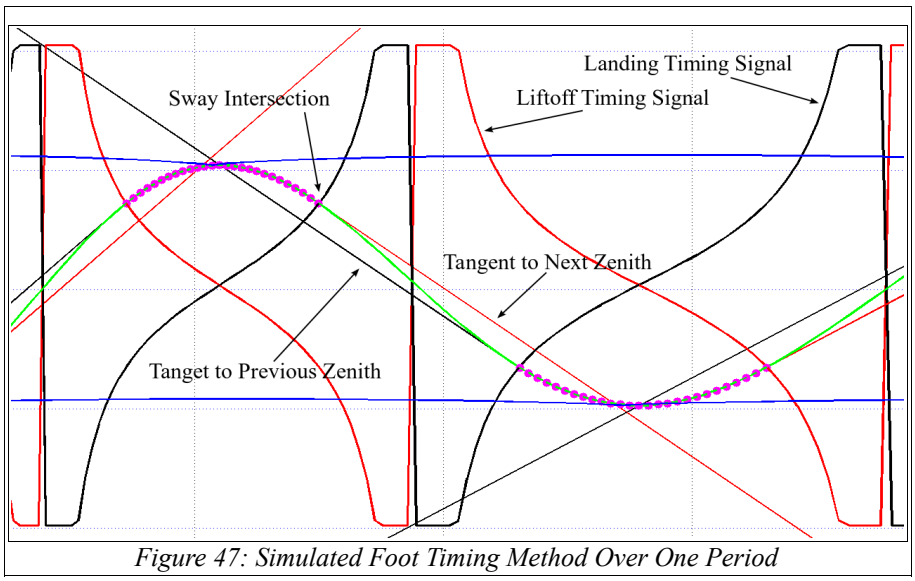


Figure 47: Simulated Foot Timing Method Over One Period

However, this method proved to introduce more disturbance than is desirable, in fact as much disturbance as possible. This makes sense as this is one of its inherent properties. Human legs are more suited to absorbing impact than the Nao robot legs and children have the two years to invest into using their on board organic adaptive controller to optimize their stiffness control transfer functions. Migrating a motion control project from simulation to hardware brings with it a variety of hurdles that must be overcome. Therefore this timing method was replaced with one that attempts to minimize the impact of the foot fall. This leads to a walk that is observably less 'human' and one that spends a lot more time in the dual support state. This simulation method may perhaps be more suitable for running or walking on stilts where there would be less room for gait variances or force controlled robots that are more able to compensate for high impact forces.

The method that was decided upon was to hard code points in the walk cycle phase to lift off (begin cycloid) and land (begin linear backwards motion through the robots global coordinate system). Both functions require their own driving phase signal that is coupled to the torso phase and each foot has their own set. This does require an extrapolation of the walk cycle phase beyond the 1.0 modulated limited so that upper and lower trigger points can be set. One foot step occurs during the walk cycle phase and the other foot bridges two consecutive phases. This method is shown in Figure

48 along with the phase coupling. With these points defined, suitable values were then determined heuristically in the lab.

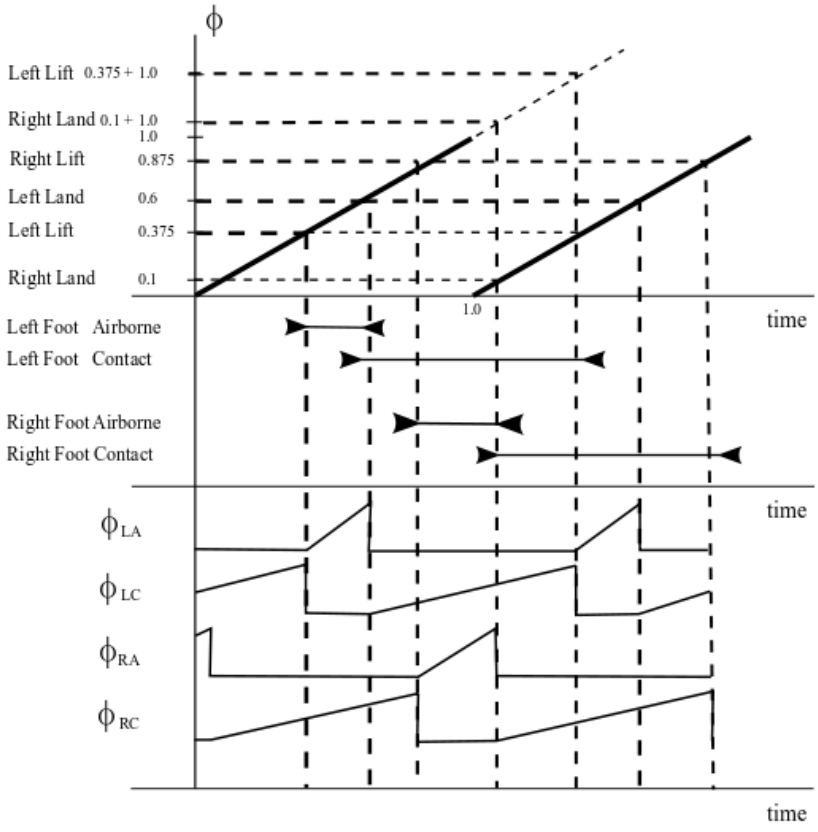


Figure 48: Hardware Method for Foot Liftoff and Landing Timing

Acceleration. Velocity change is achieved in two ways, either by increasing the step length or by increasing the signal generator's phase increment over one walk cycle. Increasing the speed by ramping up the phase increment results directly in increased step frequency and thereby increased speed. Mid stride step length adjustments require more and we consider these now. Linear acceleration over the entire walk cycle, as opposed to one that comes in spurts during single support phase, was the desired result but proved to be problematic. It is difficult to keep the feet synchronized with each other, and with the torso and keep their trajectories centred around the robot Y axis. Linear acceleration was achieved in the first version of the robot model however it came with certain drawbacks.

First of all, though the torso trajectory appears to be good, the synchronization with the feet had a long repeating pattern over many strides. The feet never broke away from their work space but with each stride they would always be in a different position relative to one another in a long repeating pattern. This gait therefore could be made to function but had several undesirable features.

Secondly, the algorithm was complicated and not very robust. It was prone to the usual discrete systems problems that arise when changing period lengths of signals (the first version of the system was defined with a signal generator that had a defined number of samples per period as opposed to a phase generator). Once the linear acceleration was achieved, the algorithm proved too sensitive to handle changes to the other properties of the system such as the first foot liftoff and landing timing method. As a result that goal was abandoned as the ability to change any parameter during run time was considered to be more important. It is worth pursuing this further in future work, as ramping velocities up to near running likely would not respond well to jerky motions. The torso position data in the world coordinate systems was captured and is shown in Figure 49. Smooth torso signals such as these would be ideal in a final product assuming that there was also some good symmetry to the foot trajectories as well.

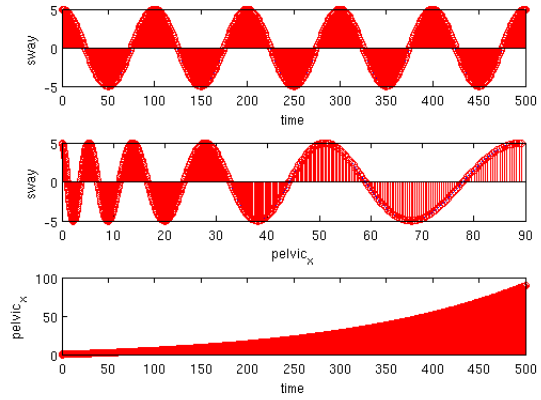


Figure 49: Simulated Torso Acceleration (world coordinate system)

A second method was therefore developed that would only implement step length changes during the single support phase. The foot cycloid algorithm was rewritten to its current form. This has the somewhat undesirable effect of accelerating in spurts but is acceptable. The problem with accelerating during dual support phase is that as the foot cycloid signal increases in magnitude, the feet have to move away from each other. This then makes the leading foot move forward instead of backwards during acceleration and this is not acceptable while in contact with the ground. A problem still does arise when the cycloids grow in size during single support phase. Splaying occurs as one foot travels further forward while the other foot travels further backwards. This is shown spatially in Figure 50 and plotted versus time (samples) in Figure 51.

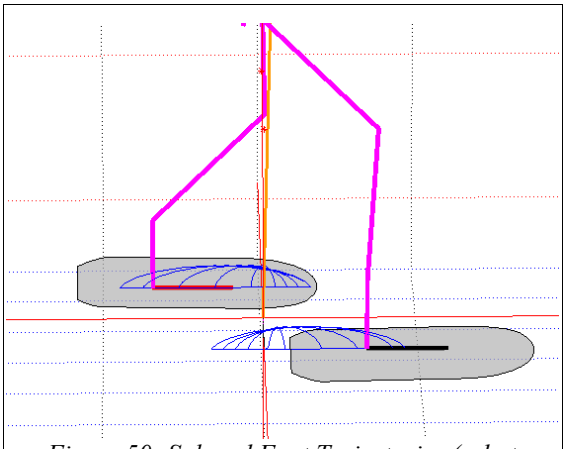


Figure 50: Splayed Foot Trajectories (robot coordinate system)

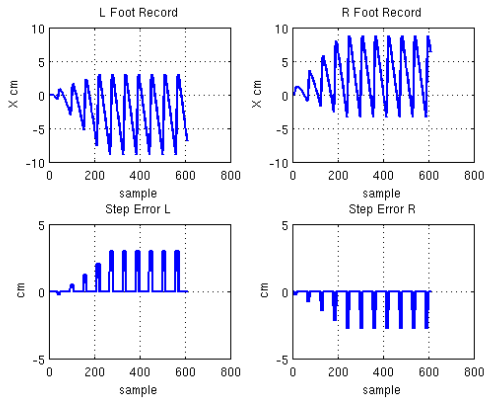


Figure 51: Foot Position and Error (uncompensated)

Step error correction. The problem of splaying step error as a result of acceleration was fixed by measuring the error during each step's liftoff sample and adding it to the target step length. The acceleration itself however is a continuous source of error so this method is always chasing the changes trying to keep up. This is shown in Figure 52. There is an improved performance and the error is eliminated completely when the acceleration period comes to an end. A gain value is therefore introduced to work against the effects of acceleration and rectify any error quickly. Setting it too high however can cause some over compensation. Figure 53 and 54 shown error correction gain values of 5 and 15 respectively.

The step error is determined by comparing the last known position of the foot with where the foot should be, given the newest step length value (shifted to be centred on robot Y axis) which could have been updated at any time during the previous dual support phase. The error is then distributed over the foots airborne phase.

Error correction function:

$$StepErrorL = \frac{-St_L}{2} - Ft_{Lx(i-1)} \tag{22}$$

$$numFlightSamples = \frac{\phi_{L\,Land} - \phi_{L\,Lift}}{\phi_I} \tag{23}$$

$$Ft_{Lx(i)} = Ft_{Lx(i)} + stepErrorGain\left(\frac{StepError_L}{numFlightSamples}\right) \tag{24}$$

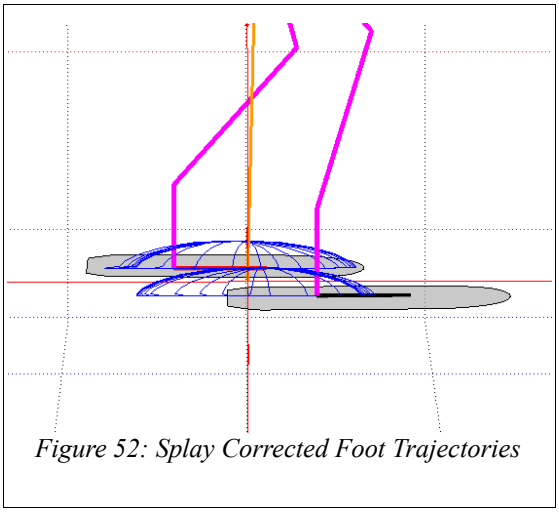


Figure 52: Splay Corrected Foot Trajectories

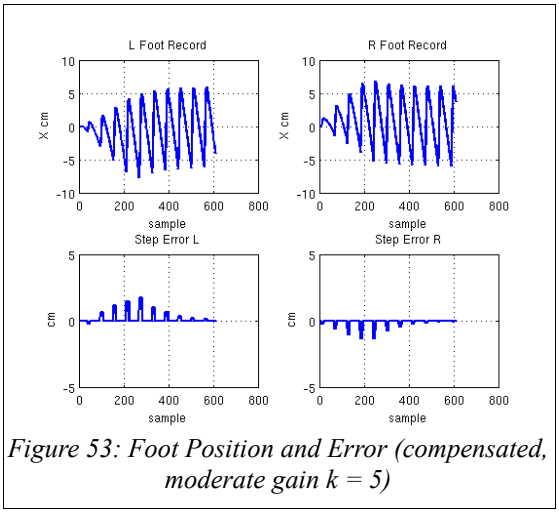


Figure 53: Foot Position and Error (compensated, moderate gain k = 5)

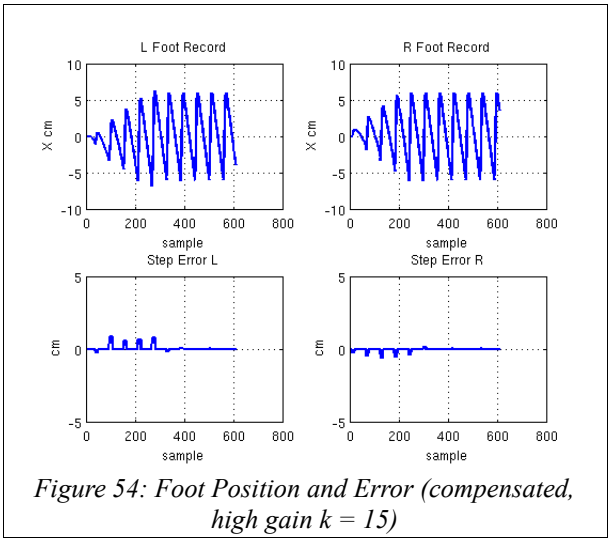


Figure 54: Foot Position and Error (compensated, high gain k = 15)

Option to use in-stride kick. Each step can be modified to act as a quick snap kick by replacing the foot cycloid with a linear function that dynamically targets the center of the ball midway through the step (the offset vector between the toe and the ankle is added to the ball data). So as not to disturb the rhythm and trajectory of the walk, the kick is performed in the same time period as the planned step and the foot landing position is the same as the regular walking cycloid placement. The three key positions (lift-off, contact and landing) are computed during the first sample of the step and then latched so that the forward motion of the robot during the step does not alter the ball location data and thus affect the momentum of the kick/step. This dynamic targeting function is shown in the following Figure 55.

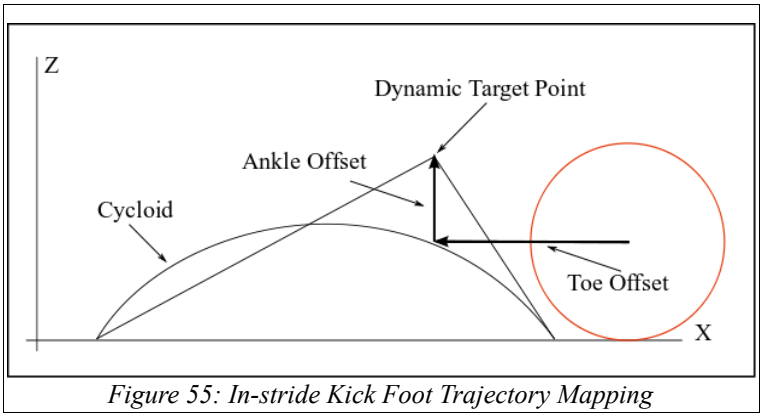


Figure 55: In-stride Kick Foot Trajectory Mapping

Side stepping algorithm. The side stepping is the only component of the system that is not purely an incidental function driven by the phase generator, relying on data from no more than the last sample. The algorithm must make use of some memory lasting one full walk cycle or bridging two. This kind of action requires some modification to the gait that once initiated, must be undone or corrected for at some future point. The motion can be broken down into concepts of a leading foot and a trailing foot. The leading foot performs the desired action such as laterally stepping to a new position outside the forward trajectory or orientating to a new heading and then at a later point the trailing foot must follow in the same way when the opportunity arises. In the interim, the robot will also have to shift its weight during dual support in a different way than in a forward walk. So there exists the potential for older memorized actions to conflict with current commands, such as changing the velocity or direction. To ensure this does not occur, the algorithm has the properties of a shift register. The act of side stepping is broken down into its most atomic elements or actions and a register is created for each one. As the process proceeds, velocity values are passed from one register to the next incrementally during the full stepping cycle to completion. The output becomes a summation of all these registers. This implementation also prevents new values from overwriting old values.

The act of side stepping is broken down into the following sub-actions and registers:

With a new lateral velocity command, the robot steps outward with the leading foot and lands, straddling the distance. While this is being performed, the step length value is split across both the leading foot and trailing foot. This gives us two values to hold in registers during the leading step.

During the dual support phase, the robot shifts its weight back to its normal position relative to the leading foot. So the existing leading foot step length value is shifted from the leading foot to the trailing foot. This gives us 3 registers: a value to transfer, a leading foot value to reduce to zero, and a trailing foot value to increase up to the step length.

Finally, the trailing leg must return to the normal stride width. That gives us a trailing value to shift on the launch sample and a register to reduce to zero during flight.

The tricky part here is that when the trailing foot lifts off it becomes the leading foot for the next step, which could be of any value or direction. Therefore, all registers have to exist in duplicate, tracking either leading left or leading right steps.

Unlike the rest of the system, this stepping action was developed on the hardware once everything else was operating. One of the final dynamic test runs was recorded and is shown in Figure 56 and 57. These figures are meant to demonstrate two things: the robustness of the algorithm successfully managing a mid stride direction change (command signal is in black) and the debugging method itself. For most debugging problems all the pertinent variables were logged during a hardware test and then plotted. MATLAB was used as it is well-stocked with signal analysis tools. There are two reasons for doing this. First, a buggy motion algorithm can be very difficult to diagnose by eye, the result is usually a violent fast-paced erratic motion. The joint command output should be shut off in early development stage but these events pop up even when everything looks like it checks out and an active output trial is attempted. Secondly, these motions are periodic in nature, and even if not, could simply be activated repeatedly. This then means all variables can be analyzed for their periodic nature, smoothness, amplitude etc. Using a plotting tool as a virtual oscilloscope will almost always help identify algorithm errors very quickly. Generally, there is a certain amount of continuity to working algorithms. When algorithm errors occur, there is often a large spike visible in the 'virtual oscilloscope' that may be very useful in debugging.

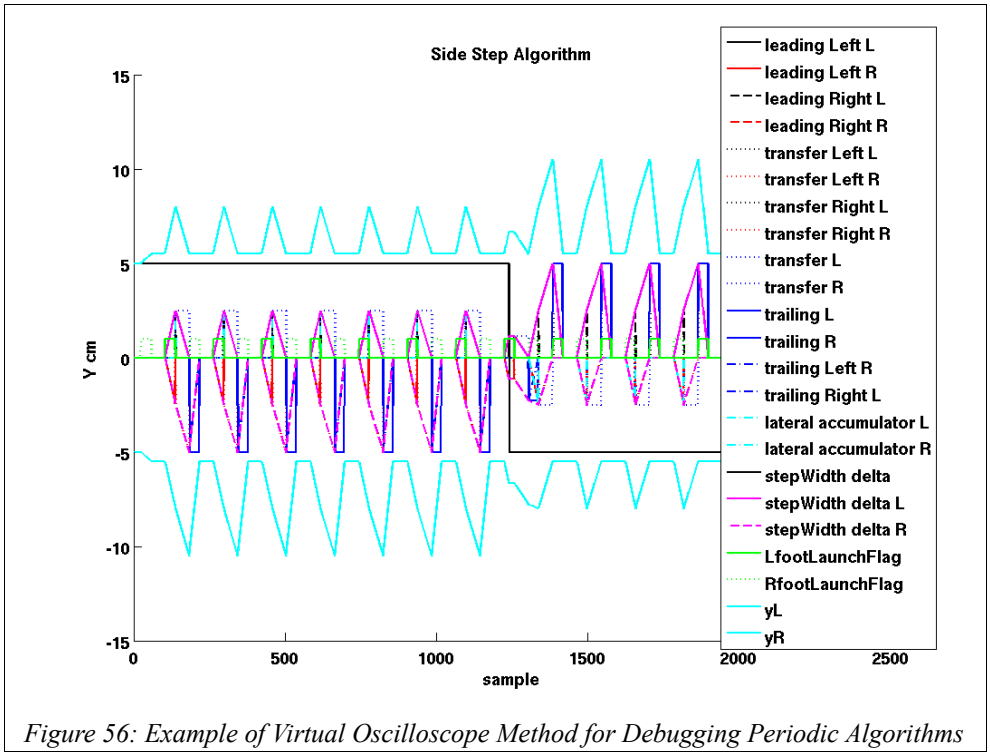


Figure 56: Example of Virtual Oscilloscope Method for Debugging Periodic Algorithms

The robustness of the algorithm is demonstrated during the step where the command to change direction occurs. The algorithm is capable of dynamically compensating for changes mid stride without historical values having an adverse effect. A close up of this action is shown more clearly in Figure 57.

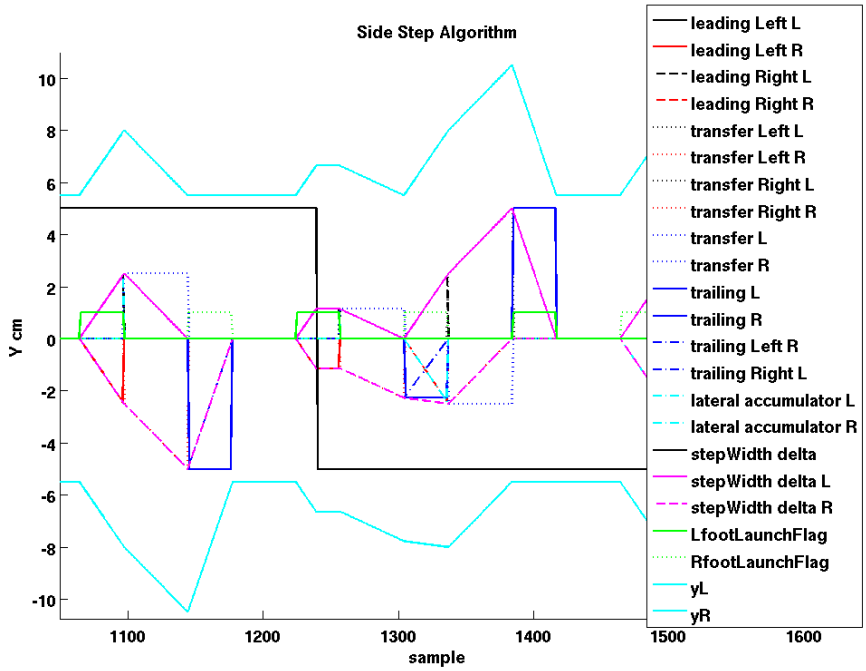


Figure 57: Demonstration of Side Step Algorithms Ability to Change Direction Mid Stride While Correcting for Historical Motion

Turning algorithm. The turning algorithm is a simple matter of changing the orientation of the feet while stepping and interpolating though the orientation change magnitude over the airborne step cycle. As for the side stepping, the velocity is split over both feet. This is also a must as the feet orientations are mechanically coupled. The output value will interpolate between whatever orientation the foot has at the start of the step up to whatever the current target value is.

If there is any existing torso roll, the turn velocity will not be accurate.

3.4.2.8. Foot Trajectory Algorithms

Each foot is controlled separately with similar equations. They differ slightly in their timing in relation to the torso cycle phase. While the feet are airborne, they follow cycloid trajectories travelling forward, and a simple linear function to back-track. All functions are defined in the robot's global coordinate frame.

Special operations are performed during lift off and land samples inside code blocks latched using the lock and key method that ensures they only run once. For example, operations such as determining the foot splay error, setting control flags and shift register clocking for side stepping and turning algorithms, as previously discussed.

The foot control algorithm has four major components, they are:

1. Evaluate the foot launch triggers and perform the 'run once' operations if required such as setting flags and data shifting operations.

2. Increment the active foot trajectory function; the forward cycloids, reversing linear trajectory or the optional in-stride snap kick. These can be broken down further into sub algorithms:

   (a) Forward Cycloids

   i. Update cycle phases

   ii. Apply smooth acceleration value to target step length

   iii. Plot cycloid trajectories

   iv. Add splay error compensation value

(b) Reversing Linear Trajectory

i. Update cycle phases
ii. Plot reverse linear trajectory

(c) Optional In-stride Snap Kick

i. Latch initial, termination and ball contact values in forward dimension on first execution sample
ii. Increment either pre or post-contact linear trajectories
iii. Increment sinusoidal foot height trajectory

3. Increment the turning and side stepping algorithms.

4. Evaluate and perform the foot landing operations if necessary.

3.4.2.9. Foot Trajectory Functions and Equations

The foot trajectory algorithm is computed as follows:

note: The following algorithm is a combination of mathematical equations and numerical methods that use numerous flags and variables. For the purpose of notation clarity, flags will have the prescript Fl_. The names will be as self-explanatory as possible, as is the convention when writing computer code. Also, reference to left or right occurs frequently and will use the notation L or R as a postscript.

Other notation used:
$Ft_{L(x,y,z)}$ / $Ft_{R(x,y,z)}$ → *Left and Right foot robot space location*

1. Evaluate the foot launch triggers and perform the 'run once' operations if required such as setting flags and data shifting operations.

Launch sample operations:

Left foot:

Check the run once latch has been cleared to open:

$$if\ (runOnceL = true) \tag{25}$$

Check the walk cycle phase has entered the foot launch trigger time period:

$$if\ ((\phi_W \geq Ft_{\text{L Lift}}) \text{and} (\phi_W < Ft_{\text{L Land}})) \tag{26}$$

Activate the left foot airborne trajectory flag:

$$Fl_{footLaunchL} = true \tag{27}$$

Compare the last known left foot location with the desired step length and record error:

$$StepErrorL = \frac{-St_L}{2} - Ft_{Lx(i-1)} \tag{28}$$

Shift the side step algorithm data:

$$trailing_R = -2\,transfer_R \tag{29}$$

$$transfer_R = 0 \tag{30}$$

$$transferRight_L = 0 \tag{31}$$

$$transferRight_R = 0 \tag{32}$$

Close the left foot run once latch:

$$Fl_{runOnceL} = false \tag{33}$$

Right foot:

Check the run once latch has been cleared to open:

$$if\,(runOnce_R = true) \tag{34}$$

Check the walk cycle phase has entered the foot launch trigger time period:

$$if\,((\phi_W \geq Ft_{\text{R Lift}})\,\text{or}\,(\phi_W < Ft_{\text{R Land}})) \tag{35}$$

Activate the right foot airborne trajectory flag:

$$Fl_{LaunchR} = true \tag{36}$$

Compare the last known right foot location with the desired step length and record error only if this is not the first step:

$$StepErrorR = \begin{Bmatrix} 0 & : & Fl_{firstStep} & = & true \\ \dfrac{-stepLengthDesired}{2} - previousSample_{x\,R} & : & Fl_{firstStep} & = & false \end{Bmatrix} \tag{37}$$

Shift the side step algorithm data:

$$trailing_L = -2\,transfer_L \tag{38}$$

$$transfer_L = 0 \tag{39}$$

$$transferLeft_L = 0 \tag{40}$$

$$transferLeft_R = 0 \tag{41}$$

Close the right foot run once latch:

$$Fl_{runOnceL} = false \tag{42}$$

2. Increment the active foot trajectory function; the forward cycloids, reversing linear trajectory or the optional in-stride snap kick. These can be broken down further into sub algorithms:

(a) *Forward Cycloids*

i. Update airborne cycle phases

Left foot:

$$\phi_{LA} = \frac{\phi_W - Ft_{\text{L Lift}}}{Ft_{\text{L Land}} - Ft_{\text{L Lift}}} \tag{43}$$

Right foot:

Buffer the walk cycle phase:

$$\phi_W^* = \phi_W \tag{44}$$

Extend the buffered cycle phase beyond the modulated limit if required:

$$if\ (\phi_W^* < (Ft_{R\,Land} + 0.1))\ \{\phi_W^* = \phi_W^* + 1.0\} \tag{45}$$

Update the right foot airborne cycle phase:

$$\phi_{RA} = \frac{\phi_W^* - Ft_{\text{R Lift}}}{(Ft_{\text{L Land}} + 1.0) - Ft_{\text{R Lift}}} \tag{46}$$

ii. Apply smooth acceleration value to target step length

For both feet (calculated independently):

Evaluate the current cycle phase step size:

$$footCyclePhaseIncrement = \phi_{A(i)} - \phi_{A(i-1)} \tag{47}$$

Set an exponentially increasing interpolation step size that eliminates existing error over the remainder of the existing stride:

$$stepLengthFactor = \frac{footCyclePhaseIncrement}{1 - \phi_{A(i-1)}} \tag{48}$$

Interpolate output step length between current and updated value:

$$stepLengthActual = stepLengthActual + stepLengthFactor * (stepLengthDesired - stepLengthActual) \tag{49}$$

iii. Plot cycloid trajectories

Left foot: (formula is the same for right foot)

Height:

$$radius_L = \frac{StepLengthActual_L}{2\pi} \tag{50}$$

$$heightFactor_L = \frac{St_H}{2\,radius_L} \tag{51}$$

$$Ft_{\text{L z}} = radius_L(1 - \cos(2\pi\phi_{LA}))\,heightFactor_L + Cn_{FootHeight} \tag{52}$$

Width:

$$Ft_{\text{y L}} = St_w \tag{53}$$

Length:

$$Ft_{\text{L x}} = radius_L((2\pi\phi_{LA}) - \sin(2\pi\phi_{LA})) - \frac{StepLengthActual_L}{2} \tag{54}$$

iv. Add splay error compensation value

$$numFlightSamples = \frac{\phi_{L\,Land} - \phi_{L\,Lift}}{\phi_I} \tag{55}$$

$$Ft_{L\,x} = Ft_{L\,x} + stepErrorGain\left(\frac{StepError_L}{numFlightSamples}\right) \tag{56}$$

(b) *Reversing Linear Trajectory*

i. Update cycle phases

Left foot:

Buffer the walk cycle phase:

$$\phi_W^* = \phi_W \tag{57}$$

Extend the buffered cycle phase beyond the modulated limit if required:

$$if\ (\phi_W^* < (Ft_{L\,Lift} + 0.1))\ \{\phi_W^* = \phi_W^* + 1.0\} \tag{58}$$

Update the left foot ground contact cycle phase:

$$\phi_{LC} = \frac{\phi_W^* - Ft_{\text{L Land}}}{(Ft_{\text{L Lift}} + 1.0) - Ft_{\text{L Land}}} \tag{59}$$

Right foot:

$$\phi_{RC} = \frac{\phi_W - Ft_{\text{R Land}}}{Ft_{\text{R Lift}} - Ft_{\text{R Land}}} \tag{60}$$

ii. Plot reverse linear trajectory

Left foot: (formula is the same for right foot)

$$Ft_{Lx} = (1 - \phi_{CL})\, stepLengthActual_L - \frac{stepLengthActual_L}{2} \tag{61}$$

(c) *Optional In-stride Snap Kick*

i. Latch initial, termination, and ball contact values in forward dimension on first execution sample.

Left foot (right foot uses the same functions):

Check snap kick command signal:

$$if\ (Fl_{doAKickLeft} = true) \tag{62}$$

Open run once latched function block:

$$if\ (Fl_{openKickLatchL} = true) \tag{63}$$

Record last known grounded foot location:

$$InitalPosition_L = Ft_{x[n-1]} \tag{64}$$

Calculate regular stride landing location:

$$CycloidTermination_{L\,x} = radius_L((2\pi) - \sin(2\pi)) - \frac{stepLengthActual_L}{2} \tag{65}$$

Latch input targeting data (shifted to the foot center location):

$$ContactPosition_{L\,x} = BallTargetData_x - Cn_{ToeVector\,x} + Cn_{BallRadius} \tag{66}$$

Close the latched section:

$$Fl_{openKickLatchL} = false \tag{67}$$

ii. Increment either pre or post-contact linear trajectories.

$$Ft_{Lx}=\begin{Bmatrix} InitalPosition_{Lx}+\phi_{AL}\dfrac{ContactPosition_{Lx}-InitalPosition_{Lx}}{0.5} & : & \phi_{AL} & < & 0.5 \\ ContactPosition_{Lx}+(\phi_{AL}-0.5)\dfrac{CycloidTermination_{Lx}-ContactPosition_{Lx}}{0.5} & : & \phi_{AL} & >= & 0.5 \end{Bmatrix} \tag{68}$$

iii. Increment sinusoidal foot height trajectory.

$$heightFactor_L=\frac{Cn_{BallRadius}}{2\ radius_L} \tag{69}$$

$$Ft_{Lz}=heightFactor_L(1-\cos(2\pi\phi_{AL}))radius_L+Cn_{FootHeight} \tag{70}$$

3. Increment the turning and side stepping algorithms.

Turning algorithm:

Split the rotational magnitude per step across both feet:

$$footOrientationChangePerStride=\frac{robotOrientationChangePerStride}{2} \tag{71}$$

Interpolate through the stepping phase:

$$Ft_{\alpha L}=initialOrientation_L+\phi_{AL}(footOrientationChangePerStride-initialOrientation_L) \tag{72}$$

Assign the inverse orientation to the opposite foot:

$$Ft_{\alpha R}=-Ft_{\alpha L} \tag{73}$$

Side stepping algorithm:

Leading Left foot airborne:

If there exists some leftward lateral velocity and the left foot is airborne:

$$if\ ((Vr_{stepWidth\Delta}>0)\&\&(Fl_{footLaunchL}==true)) \tag{74}$$

Split the velocity across both feet and interpolate through step phase:

$$leadingLeft_L=\phi_{AL}\frac{stepWidth_\Delta}{2} \tag{75}$$

$$leadingLeft_R=\phi_{AL}\frac{stepWidth_\Delta}{2}(-1) \tag{76}$$

Buffer the step velocity:

$$lateralAccumulator_L=leadingLeft_L \tag{77}$$

Leading Right foot airborne:

If there exists some rightward lateral velocity and the right foot is airborne:

$$if\ ((stepWidth_\Delta<0)\&\&(Fl_{footLaunchR}==true)) \tag{78}$$

Split the velocity across both feet and interpolate through step phase:

$$leadingRight_L = \phi_{AR} \frac{stepWidth_\Delta}{2}(-1) \tag{79}$$

$$leadingRight_R = \phi_{AR} \frac{stepWidth_\Delta}{2} \tag{80}$$

Buffer the step velocity:

$$lateralAccumulator_R = leadingRight_R \tag{81}$$

Transfer left, both feet stationary:

Create phase variable for dual support mode (leading left):

$$\phi_{DL} = \frac{\phi_W - Ft_{\text{L Land}}}{Ft_{\text{R Lift}} - Ft_{\text{L Land}}} \tag{82}$$

If there exists a value that requires transfer and both feet are stationary:

$$if\ ((transfer_L > 0)\&\&(Fl_{footLaunchL} == false)\&\&(Fl_{footLaunchR} == false)) \tag{83}$$

Do the transfer of accumulated side stepped value:

$$transferLeft_L = (1 - \phi_{DL})\, transfer_L \tag{84}$$

$$transferLeft_R = (1 - \phi_{DL})\, transfer_L \tag{85}$$

Transfer right, both feet stationary:

Create phase variable for dual support mode (leading right):

$$\phi_{DR} = \frac{\phi_W - Ft_{\text{R Land}}}{Ft_{\text{L Lift}} - Ft_{\text{R Land}}} \tag{86}$$

Again, if there exist a value that requires transfer and both feet are stationary:

$$if\ ((transfer_R < 0)\&\&(Fl_{footLaunchL} == false)\&\&(Fl_{footLaunchR} == false)) \tag{87}$$

Do:

$$transferRight_L = (1 - \phi_{DR})\, transfer_R \tag{88}$$

$$transferRight_R = (1 - \phi_{DR})\, transfer_R \tag{89}$$

Trailing left, opposite foot airborne:

If there exists a trailing value from a leading left side step and the right foot is airborne:

$$if\ ((trailing_L < 0)\&\&(Fl_{footLaunch\,R} == 1)) \tag{90}$$

Laterally retract the trailing right foot by the accumulated trailing value:

$$trailingLeft_R = (1 - \phi_{AR})\, trailing_L \tag{91}$$

Trailing right, opposite foot airborne:

If there exists a trailing value from a leading right side step and the right foot is airborne:

$$if\ ((trailing_R > 0)\&\&(Fl_{footLaunch\,L} == 1)) \tag{92}$$

Laterally retract the trailing left foot by the accumulated trailing value:

$$trailingRight_L = (1 - \phi_{AL}) trailing_R \tag{93}$$

Side step output:

Accumulate all registers:

$$stepWidth_{\Delta L} = leadingLeft_L + leadingRight_L + transferLeft_L + transferRight_L + trailingRight_L \tag{94}$$

$$stepWidth_{\Delta} R = leadingLeft_R + leadingRight_R + transferLeft_R + transferRight_R + trailingLeft_R \tag{95}$$

Update foot's lateral position:

$$Ft_{Ly} = Ft_{Ly} + stepWidth_{\Delta L} \tag{96}$$

$$Ft_{Ry} = Ft_{Ry} + stepWidth_{\Delta R} \tag{97}$$

4. Evaluate and perform the foot landing operations if necessary.

Left Foot:

If the foot landing latch is open and the step phase is complete:

$$if\ (Fl_{runOnceEnd\ L} == 1)\ \&\&\ (\phi_{AL} >= 1) \tag{98}$$

Close the foot landing latch:

$$Fl_{runOnceEnd\ L} = false \tag{99}$$

Deactivate the foot airborne trajectory flag:

$$Fl_{footLaunch\ L} = false \tag{100}$$

Open the foot lift off latch:

$$Fl_{runOnceStart\ L} = true \tag{101}$$

Reset the step error register:

$$StepError_L = 0 \tag{102}$$

Load side step transfer register from buffer:

$$transfer_L = lateralAccumulator_L \tag{103}$$

Shift transfer values to each foots register:

$$transferLeft_L = transfer_L \tag{104}$$

$$transferLeft_R = -transfer_L \tag{105}$$

Clear used registers:

$$lateralAccumulator_L = 0 \tag{106}$$

$$leadingLeft_L = 0 \tag{107}$$

$$leadingLeft_R = 0 \tag{108}$$

$$trailing_R = 0 \tag{109}$$

$$trailingRight_L = 0 \tag{110}$$

Load the opposite foot initial orientation with the opposite of the current value:

$$initialOrientation_R = -footCurrentOrientation_L \quad (111)$$

Clear the left foot orientation register:

$$footCurrentOrientation_L = 0 \quad (112)$$

Reset the variables used by the acceleration function:

$$footCyclePhaseIncrement_L = 0 \quad (113)$$

$$stepLengthFactor_L = 0 \quad (114)$$

Right Foot:

The right foot landing sample operations are much the same:

$$if\ (Fl_{runOnceEndR} == 1)\ \&\&\ (\phi_{AR} >= 1) \quad (115)$$

$$Fl_{runOnceEndR} = false \quad (116)$$

$$Fl_{footLaunchFlag\ R} = false \quad (117)$$

$$Fl_{runOnceStartR} = true \quad (118)$$

$$StepError_R = 0 \quad (119)$$

$$Fl_{firstStep} = false \quad (120)$$

$$transfer_R = lateralAccumulator_R \quad (121)$$

$$transferRight_L = -transfer_R \quad (122)$$

$$transferRight_R = transfer_R \quad (123)$$

$$lateralAccumulator_R = 0 \quad (124)$$

$$leadingRight_L = 0 \quad (125)$$

$$leadingRight_R = 0 \quad (126)$$

$$trailing_L = 0 \quad (127)$$

$$trailingLeft_R = 0 \quad (128)$$

$$initialOrientation_L = -footCurrentOrientation_R \quad (129)$$

$$footCurrentOrientation_R = 0 \quad (130)$$

$$footCyclePhaseIncrement_R = 0 \quad (131)$$

$$stepLengthFactor_R = 0 \quad (132)$$

3.4.2.10. The Arms

The arms do not play a complex role in this walking engine. They interpolate through periodic signals that are synchronized with either feet or the torso. During the first step they must interpolate from the initial pose.

The x dimensional motions track saw tooth wave forms, this provides small jerks at the signal peak when they change directions as a way of compensating for their low mass.

The y dimensional motions sway like the torso but in the opposite direction. This to a large degree removes the swaying motion from the arms spatial trajectories.

The range of motion is not dynamic however. Ideally the signal peaks would be proportional to the robots stride length. They are however parametrized by range of motion.

3.4.2.10.1. Arm Function Initialization:

Abbreviations used:

(L/R)SR → (Left/Right) Shoulder Roll
(L/R)SP → (Left/Right) Shoulder Pitch
(L/R)ER → (Left/Right) Elbow Roll

The arm parametrization and initialization table is attached as Appendix H.

On the first step:

$$if\ (Fl_{FirstStep} == True) \tag{133}$$

Load signal starting point with initial pose values:

$$\begin{aligned} RSP_{MIN} &= RSP_{Init} \\ RSR_{MIN} &= RSR_{Init} \\ RER_{MIN} &= RER_{Init} \\ LSP_{MIN} &= LSP_{Init} \\ LSR_{MIN} &= LSR_{Init} \\ LER_{MIN} &= LER_{Init} \end{aligned} \tag{134}$$

3.4.2.10.2. Arm Signal Trajectories:

Right Arm:

If the left foot is airborne, right arm swings forward :

$$if\ (Fl_{FootLaunchL} == True) \tag{135}$$

$$RShoulderPitch = RSP_{MIN} + \phi_{AL}(RSP_{MAX} - RSP_{MIN}) \tag{136}$$

$$RElbowRoll = RER_{MIN} + \phi_{AL}(RER_{MAX} - ReER_{MIN}) \tag{137}$$

Otherwise the right arm swings backwards:

$$if\ (Fl_{footLaunchL} == false) \tag{138}$$

$$RShoulderPitch = RSP_{MAX} - \phi_{CL}(RSP_{MAX} - RSP_{MIN}) \tag{139}$$

$$RElbowRoll = RER_{MAX} - \phi_{CL}(RER_{MAX} - RER_{MIN}) \tag{140}$$

Left Arm:

If the right foot is airborne, left arm swings forward :

$$if\ (Fl_{footLaunchR} == true) \tag{141}$$

$$LShoulderPitch = LSP_{MIN} + \phi_{AR}(LSP_{MAX} - LSP_{MIN}) \tag{142}$$

$$LElbowRoll = LER_{MIN} + \phi_{AR}(LER_{MAX} - LER_{MIN}) \tag{143}$$

Otherwise the right arm swings backwards:

$$if\,(Fl_{footLaunchR} == false) \tag{144}$$

$$LShoulderPitch = LSP_{MAX} - \phi_{CR}(LSP_{MAX} - LP_{MIN}) \tag{145}$$

$$LElbowRoll = LER_{MAX} - \phi_{CR}(LER_{MAX} - LER_{MIN}) \tag{146}$$

The shoulders both roll with sinusoidal signal in the opposite direction as the torso sway.

This eliminates the swaying motion from the arms, they follow a more linear spacial trajectory then.

$$RShoulderRoll = -SR_{MIN} + (SR_{MAX}\sin(2\pi\phi_W)) \tag{147}$$

$$LShoulderRoll = SR_{MIN} - (SR_{MAX}\sin(2\pi\phi_W)) \tag{148}$$

3.5. Zero Moment Point

An inertial vector that hangs like a pendulum (in red) and ZMP signal (green) was added to the model for visual feedback and parameter analysis during simulation (Figure 58). The Zero Moment Point (ZMP), the point at which the inertial vector and ground forces meet and negate each other, was determined by calculating an inertial vector and projecting it to the ground plane and finding the intersection using a plane-line intersection mfile found on MathWorks.com posted by Nassim Khaled [214].

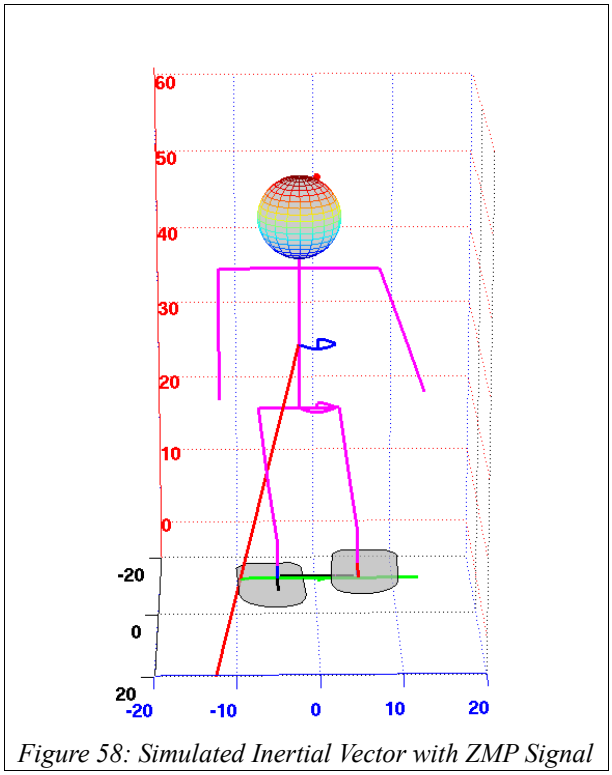


Figure 58: Simulated Inertial Vector with ZMP Signal

The inertial vector and ZMP signal proved to be quite an accurate model as the parameter values used in simulation to yield a good looking signal also worked best on the hardware.

The inertial vector can be calculated without the use of a mass model or any calculation of forces. The robot is controlled at one level of abstraction higher than the motor system and can be considered an open loop position control where the motors are black box components. We are not concerned with how much force the motors must generate to achieve a desired pose. We assume that the robot's designers have built the robot such that any reasonable motion request will be implemented. We can therefore consider the transfer function of this mechanical component as near $k = 1$ (with some time delay) and any set position will be achieved during the prescribed time frame

(within practical limits obviously and setting aside the impact of lowered joint holding torque). This open loop position control can be expanded to closed loop making use of forward kinematics and the inertial unit that provides orientation. Disturbance rejection can be achieved via 'go forward', 'go backwards' etc. corrections. A more comprehensive mass/force model walking engine can be developed as Aldebaran does provide the mass of each body, the associated COMs, motor types/torque, gear transfer ratios and foot pressure sensors for feedback if desired. This leaves the user with the job of empirically determining the backlash and friction forces and hope that the motor stiffness setting (a parameter ranging from zero to one, without units) really does represent the maximum and minimum motor torque.

We can however analyze the ZMP, which is a mass/inertia related concept, without the use of a mass model with a similar such position control assumption. This is even simpler in simulation as there will be no time delay, position changes are implemented instantaneously with each sample update. The ZMP is calculated by finding the inertial force vector and determining where it intersects the ground plane. What is of interest to us is where this location is, specifically, we want to ensure it is within the robots support polygon.

Wikipedia provides a few interpretations of inertia, one of which is as follows:

> Interpretations
> **Mass and inertia**
> Physics and mathematics appear to be less inclined to use the original concept of inertia as "a tendency to maintain momentum" and instead favor the mathematically useful definition of inertia as the measure of a body's resistance to changes in momentum or simply a body's inertial mass.
> This was clear in the beginning of the 20th century, when the theory of relativity was not yet created. Mass, *m*, denoted something like amount of substance or quantity of matter. And at the same time mass was the quantitative measure of inertia of a body.
> The mass of a body determines the momentum *p* of the body at given velocity *v*; it is a proportionality factor in the formula:
> $$p = mv$$
> The factor *m* is referred to as inertial mass.
> But mass as related to 'inertia' of a body can be defined also by the formula:
> $$F = ma$$
> Here, *F* is force, *m* is mass, and *a* is acceleration.
> By this formula, the greater its mass, the less a body accelerates under given force. Masses *m* defined by formula (1) and (2) are equal because formula (2) is a consequence of formula (1) if mass does not depend on time and velocity. Thus, "mass is the quantitative or numerical measure of body's inertia, that is of its resistance to being accelerated".
> This meaning of a *body's inertia* therefore is altered from the original meaning as "a tendency to maintain momentum" to **a description of the measure of how difficult it is to change the momentum of a body**.
>
> Source: http://en.wikipedia.org/wiki/Inertia

In this application there is no difficulty in changing the robots momentum (within practical limits). The robot will provide enough force X to move mass X to where we want it to be. Therefore we can ignore the mass and focus on the acceleration vector only to find ZMP location. Practically however, as the servos drive the joints hard to their desired locations, large (jerky) reaction forces are created (and therefore irregular accelerations). So the ZMP is also erratic. To use this method beyond simulation some smoothing/filtering would certainly be required.

The location of the center of mass to which we attach the acceleration vector is also approximated. It is treated as a fixed point inside the robot's torso frame. This approximation is based on the assumption that the location will not vary too much given the symmetrical nature of the walk poses. Any existing inaccuracy in the oscillating ZMP signal can be partially compensated with a small gain factor during the system calibration if need be.

3.5.1. Inertial Vector and ZMP Calculation

$$t_s = 10\text{ms} \tag{149}$$

$$\boldsymbol{a} = \frac{\boldsymbol{COM}_i - 2\boldsymbol{COM}_{i-1} + \boldsymbol{COM}_{i-2}}{t_s^2} \tag{150}$$

$$g = -980\, cm/s^2 \tag{151}$$

$$\boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ -(a_z + g) \end{bmatrix} \tag{152}$$

$$\boldsymbol{ZMP} = -\boldsymbol{a} + \boldsymbol{COM} \tag{153}$$

3.6. Simulation of Torso Roll and Height Variance

Using the model, we simulated the walk to analyze the performance improvements that are possible given the parameters of peak torso roll (γ_{pk}) and range of torso height variance (ΔHp_Z). For comparison purposes we simulate the Aldebaran walk at its most stable higher end velocity of 0.10 m/s. The model is configured to walk like the Aldebaran system by assigning the parameter values; torso height (Hp_Z): 22.3 cm, sway peak (Hp_Y): 2.3 cm, stride length (St_L): 10 cm, output samples per period of: 100 (walk cycle frequency: 1 Hz), and no change on torso height as the robot walks (LIPM).

3.6.1. Aldebaran Benchmark

Using the Aldebaran parameters, we increased the walk cycle frequency and observed the effects on the lateral motion (Figure 59) until the theoretical lateral tipping point is reached (stability margin = 0) as shown in Figure 60. These results are used as a benchmark for comparison. On the Nao robot the lateral 10 cm point corresponds to the outside edge of the foot when the stride width matches the hip width. The stability margin is the distance between the ZMP location and the outside edge of the foot.

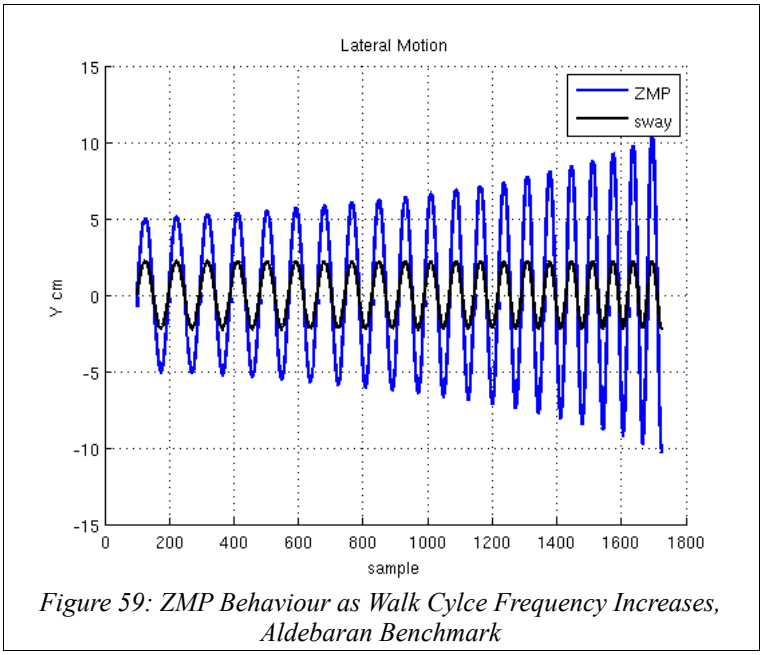


Figure 59: ZMP Behaviour as Walk Cylce Frequency Increases, Aldebaran Benchmark

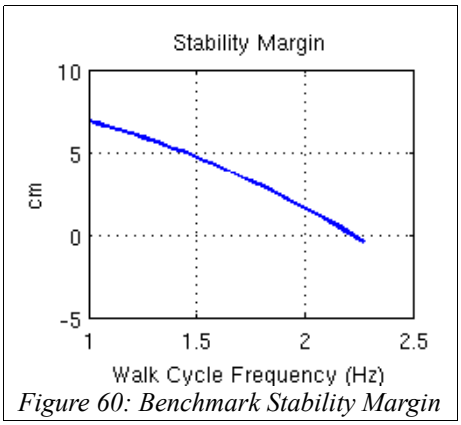


Figure 60: Benchmark Stability Margin

Figures 61 and 62 show the development of the velocity and Froude number as walk cycle frequency increases up to the stability margin limit:

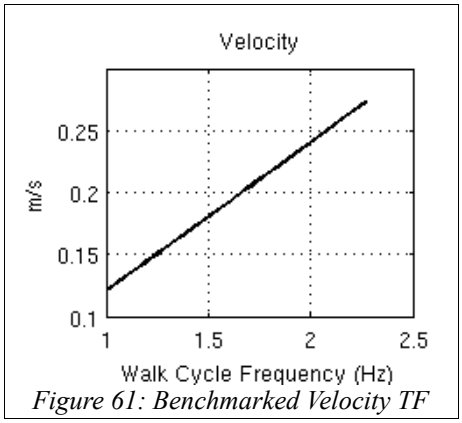


Figure 61: Benchmarked Velocity TF

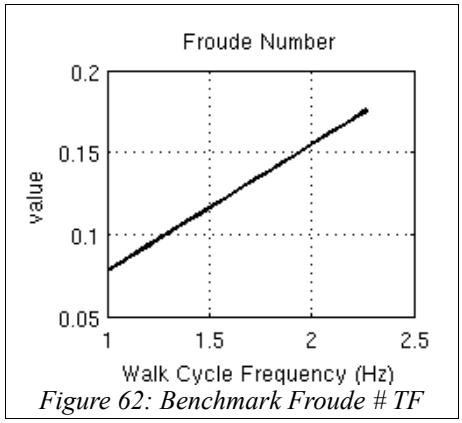


Figure 62: Benchmark Froude # TF

Benchmark results:

- Maximum walk cycle frequency: 1.72 Hz (58 samples per cycle)
- Maximum velocity: 0.207 m/s
- Maximum Froude number: 0.133
-

3.6.2. Torso Roll Simulation

Figure 63 demonstrates the simulated effects of added roll compensation on the stability margin. A range of trial roll angles were used up to 5.5 deg limit in 0.55 deg increments. This limit was chosen as this is the point where the center of the robots head reaches the sagittal plane for the given hip sway peak of 2.3 cm.

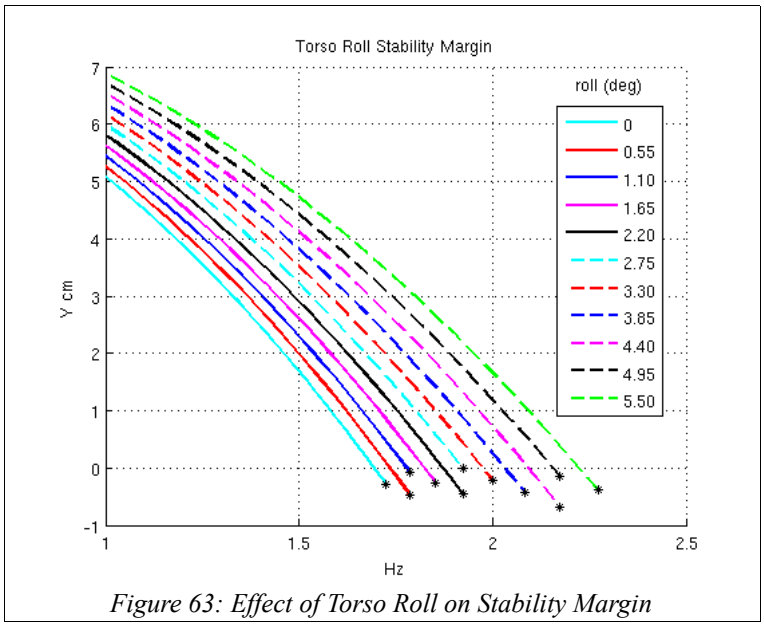


Figure 63: Effect of Torso Roll on Stability Margin

From these trails the effect on maximum performance measures were determined as a function of peak roll value $f(\gamma_{pk})$ (Figure 64).

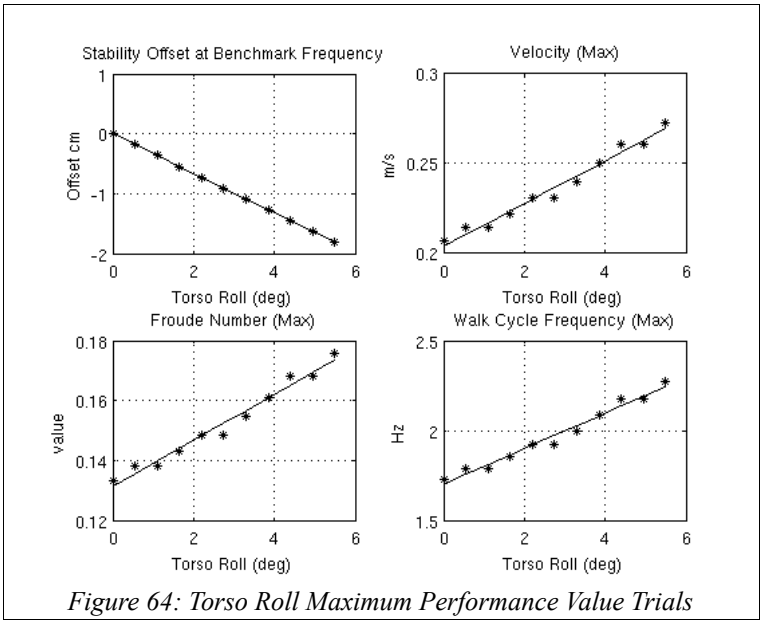


Figure 64: Torso Roll Maximum Performance Value Trials

From this we can evaluate the theoretically maximum stable (stability margin = 0) limit transfer functions as a function of peak roll value (given benchmark sway = 2.3, height = 22.3):

Maximum velocity:	$velocity_{MAX} = 0.012\gamma_{pk} + 0.2$	(154)
Maximum Froude number:	$Froude\ Number_{MAX} = 0.077\gamma_{pk} + 0.13$	(155)
Maximum walk cycle frequency:	$Cycle\ Frequency_{MAX} = 0.099\gamma_{pk} + 1.7$	(156)

Torso roll - ZMP offset transfer function for stabilizing at benchmark walk cycle frequency:

$$ZMPoffSet = -0.33\,roll_{pk} + 0.0015 \tag{157}$$

Compensation results with limited maximum roll (5.5 deg):

- Maximum walk cycle frequency: 2.27 Hz (44 samples per cycle)
- Maximum velocity: 0.273 m/s
- Maximum Froude number: 0.175

Which represents an increase of:

- Maximum walk cycle frequency: 31.8 %
- Maximum velocity: 31.8 %
- Maximum Froude number: 31.8 %

3.6.3. Varying Torso Height Simulation

The effects of varying the torso height during the walk cycle had an interesting and unexpected result. With each trail peak value of height variance, the reduction of the ZMP peak reduced much like that of the torso roll method as shown in Figure 65. Here 11 trials peak values were used up to a 4 cm limit.

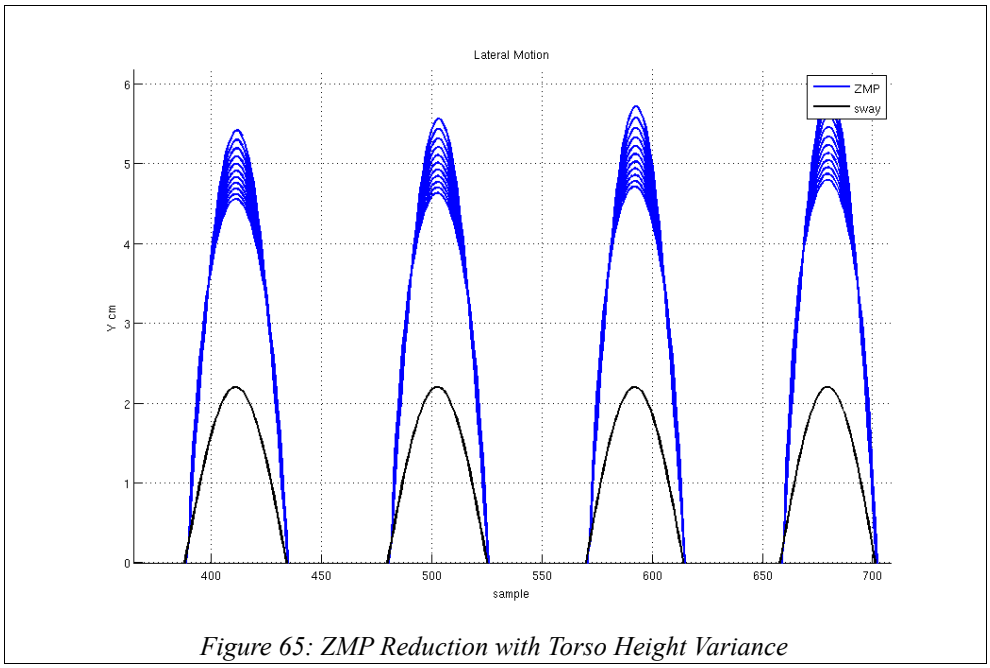
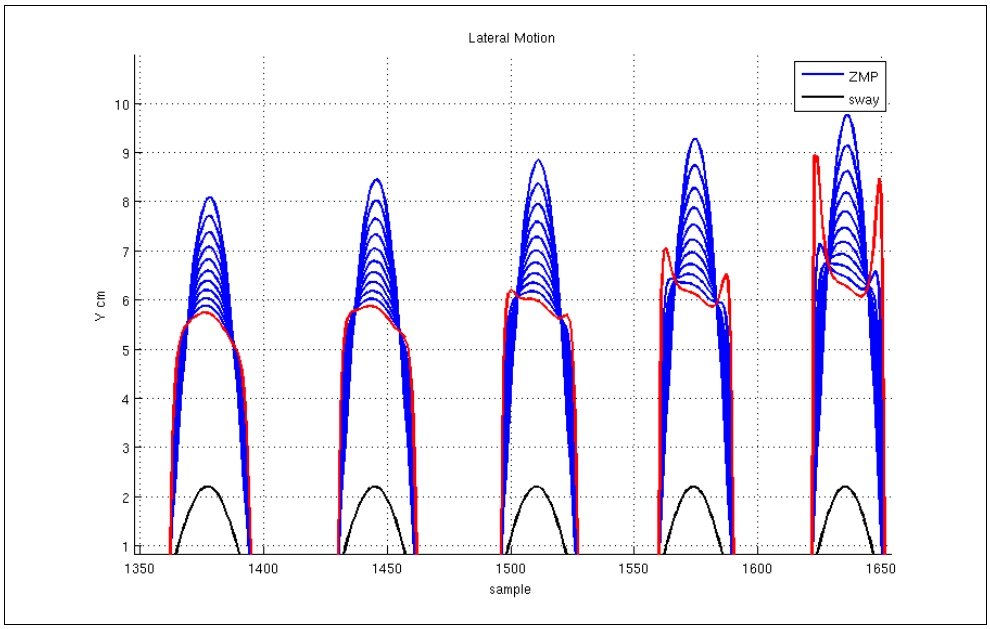


Figure 65: ZMP Reduction with Torso Height Variance

However, as the walk cycle frequency increases towards the higher end, the signal characteristics begin to change. Harmonics appear that very quickly shoot up beyond the 10 cm stability boundary that limit further speed increases despite the large existing stability margin at the sway peak.



The evolution of the stability margin versus walk cycle frequency is shown in Figure 66 for a range of Δhp_Z values where the premature termination is noticeable.

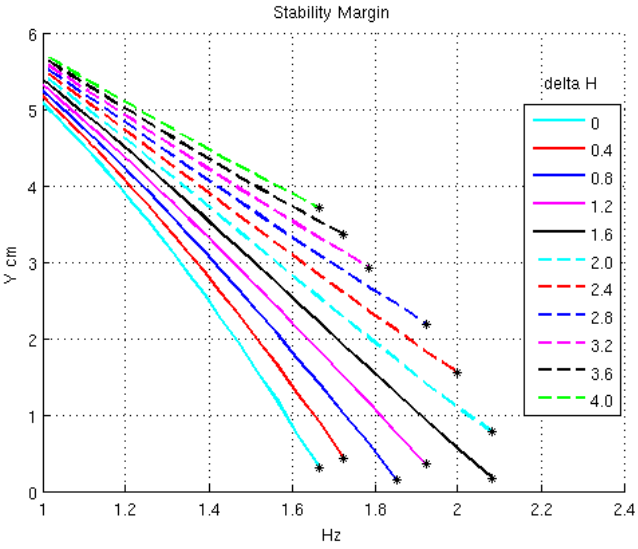


Figure 66: Effect of Height Variance on Stability Margin

The effect on theoretical maximum stable values have a nonlinear response to Δhp_Z (Figure 67).

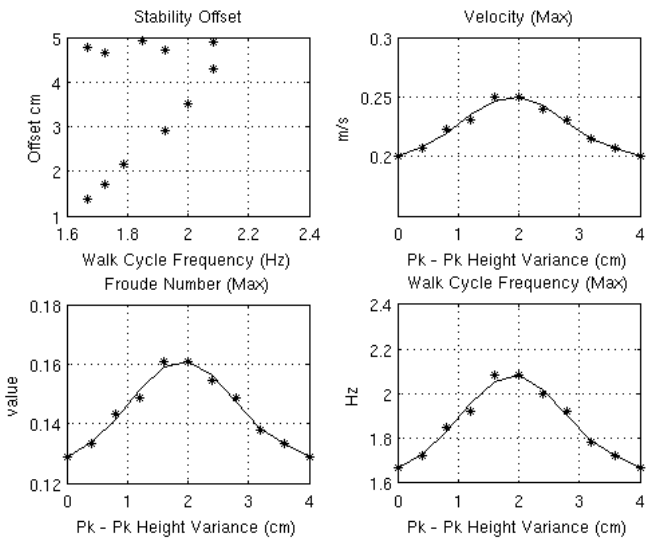


Figure 67: Torso Height Variance Maximum Performance Value Trials

To avoid this undesirable harmonic influence, we therefore chose limited the Δhp_Z to a maximum of 2 cm (Figure 68) as any value further reduces both the stability and maximum velocity.

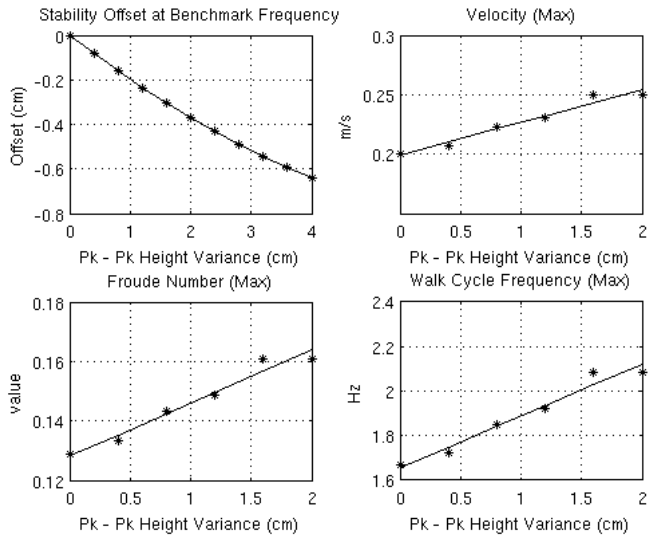


Figure 68: Limited Torso Height Variance Maximum Performance Value Trials

From this we can evaluate the theoretically maximum stable (stability margin = 0) limit transfer functions as a function of Δhp_Z (given benchmark sway = 2.3, height = 22.3):

Maximum velocity: $$velocity_{MAX} = 0.0277\,\Delta hp_Z + 0.1989 \quad (158)$$

Maximum Froude number: $$Froude\ Number_{MAX} = 0.0178\,\Delta hp_Z + 0.1281 \quad (159)$$

Maximum walk cycle frequency: $$Cycle\ Frequency_{MAX} = 0.2308\,\Delta hp_Z + 1.6579 \quad (160)$$

Δhp_Z - ZMP offset transfer function for stabilizing at benchmark walk cycle frequency:

$$ZMPoffSet = -1.4868\,\Delta hp_Z - 1.1716 \quad (161)$$

Compensation results with limited maximum Δhp_Z (2 cm):

- Maximum walk cycle frequency: 2.08 Hz (48 samples per cycle)
- Maximum velocity: 0.254 m/s
- Maximum Froude number: 0.161

Which represents an increase of:

- Maximum walk cycle frequency: 20.8 %
- Maximum velocity: 20.8 %
- Maximum Froude number: 20.8 %

3.6.4. Combined Varying Torso Height and Roll Simulation

We simulated the walk with the combined parameters, γ_{pk} and Δhp_Z, using the values in Table 2, to determine if they could be superimposed or if there would be some complex relationship between them. The results are shown in Figure 69.

Trial #	1	2	3	4	5	6	7	8	9	10	11
γ_{pk} (deg)	0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Δhp_Z (cm)	0	0.55	1.1	1.65	2.2	2.75	3.3	3.85	4.4	4.95	5.5

Table 2: Combined Paramter Trial Values

The results show that there is a complex relationship between the parameters. Harmonics again developed with increasing Δhp_Z even within the previous limited range of 0 – 2.0 cm. Figure 69 Demonstrates that with increased walk cycle frequency and combined torso roll, harmonics begin to develop after Δhp_Z increases beyond 1.2 cm.

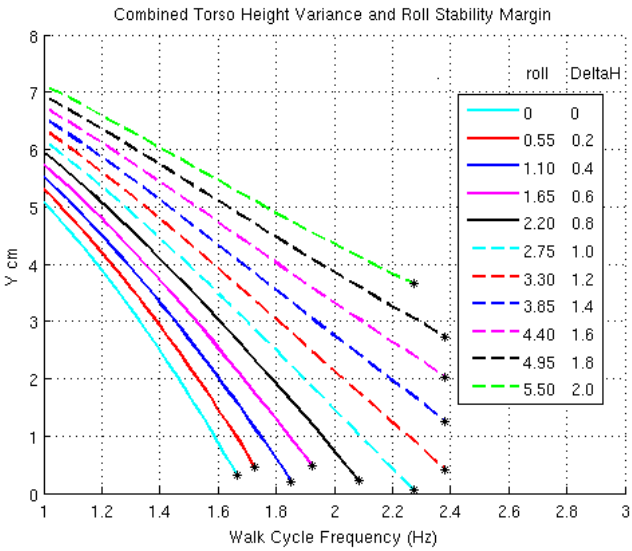


Figure 69: Effect of Simultaneous Height Variance and Torso Roll on Stability Margin

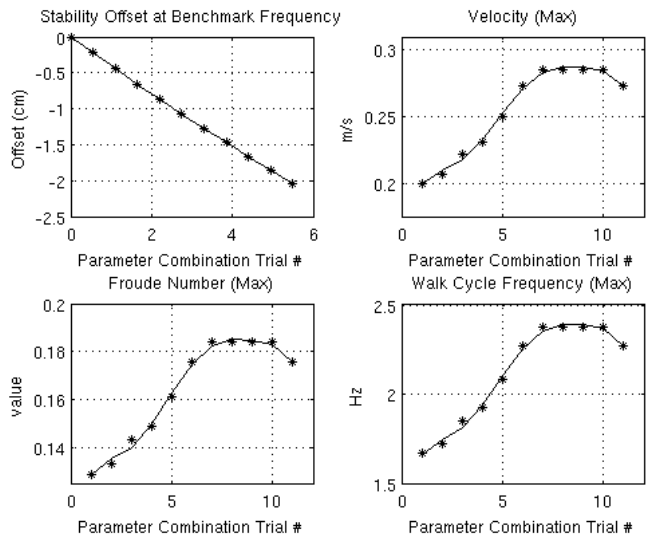


Figure 70: Combined Parameter Maximum Performance Value Trials

Theoretical maximum compensation results with γ_{pk} = 3.3 deg and Δhp_Z = 1.2 cm:

- Maximum walk cycle frequency: 2.38 Hz (42 samples per cycle)
- Maximum velocity: 0.286 m/s
- Maximum Froude number: 0.184

Which represents an increase of:

- Maximum walk cycle frequency: 38.1 %
- Maximum velocity: 38.1 %
- Maximum Froude number: 38.1 %

3.6.5. Maintaining Stability with Increased Walk Cycle Frequency

Later laboratory observations during experimentation with the hardware showed that the robot is most stable with a sway peak of 2.3 cm with benchmark gait parameter values. Measurements of the Aldebaran system also revealed they used a fixed value of approximately 2.3 cm. This translates to ZMP signal peak stability margin of 5 cm, which turns out to be the center of the foot axis. The robot should theoretically maintain its stability until the ZMP reaches the edge of the foot. However numerous other factors come into play, such as swinging limb dynamics, foot landing impact forces, reduced joint stiffness that can effect the stability behaviour of the ankle joints when the ZMP is not centred in the foot, etc.. Figure 71 shows a comparison of the effective parameter space for each parameter, independently and combined.

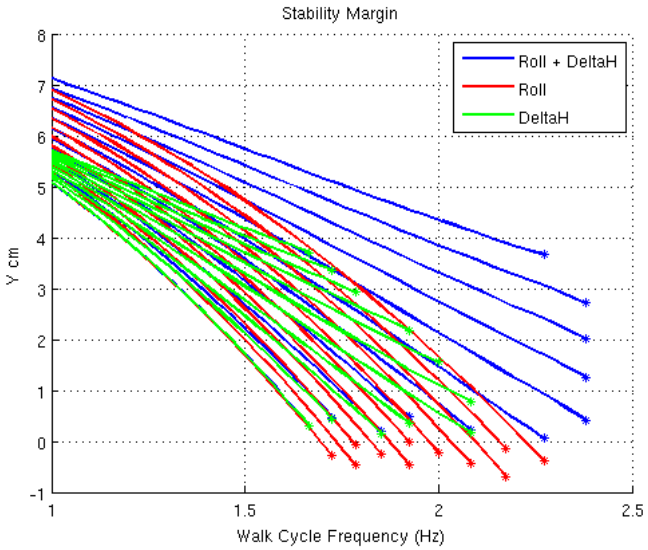


Figure 71: Comparison of Independant and Combined Paramter Space

Increasing the step length increases velocity, but increased walk cycle frequency increases motion speed in both the sagittal and frontal plane. We would like to determine the parameter values γ_{pk} and Δhp_Z as a function of the Walk Cycle Frequency (W_f) such that a specific Stability Margin (S_m) is maintained (5 cm from the foot edge) while the robot walks faster.

$$\gamma_{pk} = f\{S_m, W_f\} \tag{162}$$

$$\Delta hp_Z = f\{S_m, W_f\} \tag{163}$$

$$[\gamma_{pk}, \Delta hp_Z] = f\{S_m, W_f\} \tag{164}$$

First, we evaluate the measured S_m response as a systems of 3rd order polynomial functions of W_f.

$$\begin{aligned} S_{m(n)} &= A_{(n)} W_f^{\,3} + B_{(n)} W_f^{\,2} + C_{(n)} W_f + D_{(n)} \\ &\vdots \\ S_{m(N)} &= A_{(N)} W_f^{\,3} + B_{(N)} W_f^{\,2} + C_{(N)} W_f + D_{(N)} \end{aligned} \tag{165}$$

Where n → N for N = 11 trials in each parameter space.

The coefficients A, B, C and D were found using MATLAB's 3rd order least squares polynomial fit for each parameter trial to yield the coefficient matrix (coefficient values in Table 7, Appendix G) :

$$\begin{bmatrix} A_n & B_n & C_n & D_n \\ \vdots & \vdots & \vdots & \vdots \\ A_N & B_N & C_N & C_N \end{bmatrix} \tag{166}$$

The columns in this coefficient matrix were simulated as a function of parameter value (Figure 72) yielding the following transfer functions for each parameter.

Torso roll:

$$\left[A_{\gamma(n\Rightarrow N)},\ B_{\gamma(n\Rightarrow N)},\ C_{\gamma(n\Rightarrow N)},\ D_{\gamma(n\Rightarrow N)}\right] = f\left\{\gamma_{pk(n\Rightarrow N)}\right\} \tag{167}$$

Torso change in height:

$$\left[A_{\Delta hp(n\Rightarrow N)},\ B_{\Delta hp(n\Rightarrow N)},\ C_{\Delta hp(n\Rightarrow N)},\ D_{\Delta hp(n\Rightarrow N)}\right] = f\left\{\Delta hp_{Z\,(n\Rightarrow N)}\right\} \tag{168}$$

Both torso roll and change in height:

$$\left[A_{\gamma\,\Delta hp_Z(n\Rightarrow N)},\ B_{\gamma\,\Delta hp_Z(n\Rightarrow N)},\ C_{\gamma\,\Delta hp_Z(n\Rightarrow N)},\ D_{\gamma\,\Delta hp_Z(n\Rightarrow N)}\right] = f\left\{\gamma_{pk\ n\Rightarrow N}\ ,\ \Delta hp_{Z\ n\Rightarrow N}\right\} \tag{169}$$

These coefficient functions were then found again using MATLAB's polyfit tool. These functions appeared to be higher order so an evaluation of goodness of fit was performed for various N values.

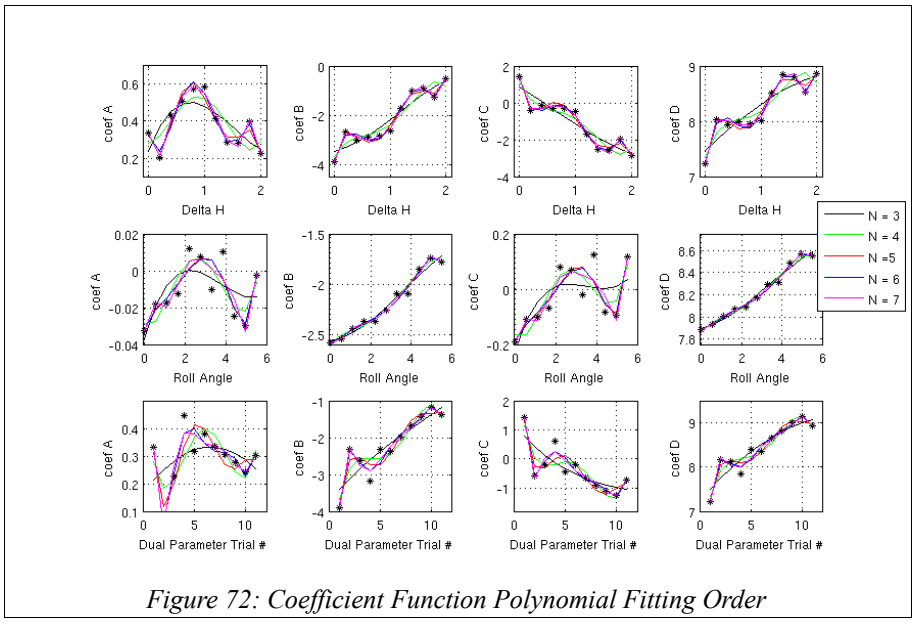


Figure 72: Coefficient Function Polynomial Fitting Order

The goodness of fit was evaluated by observing both the Mean Squared Error (Figure 73) and absolute values (Figure 74).

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\left(Y_{FIT} - Y_{TRUE}\right)^2$$

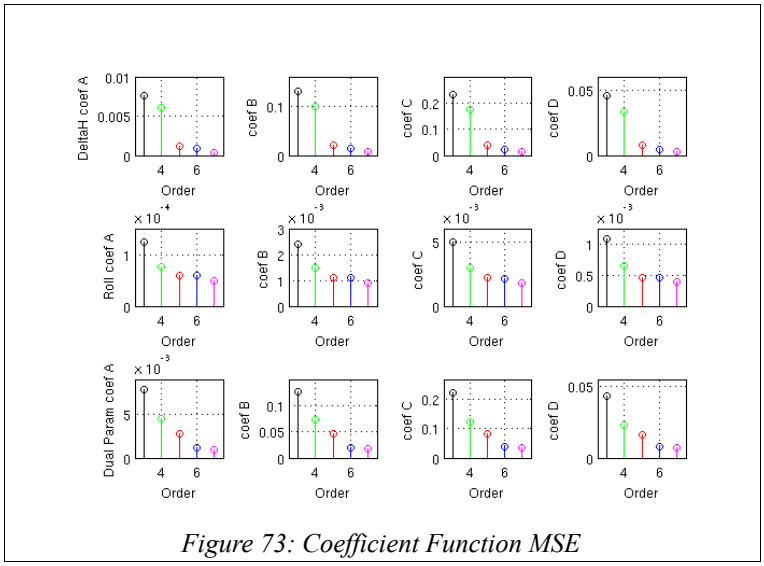


Figure 73: Coefficient Function MSE

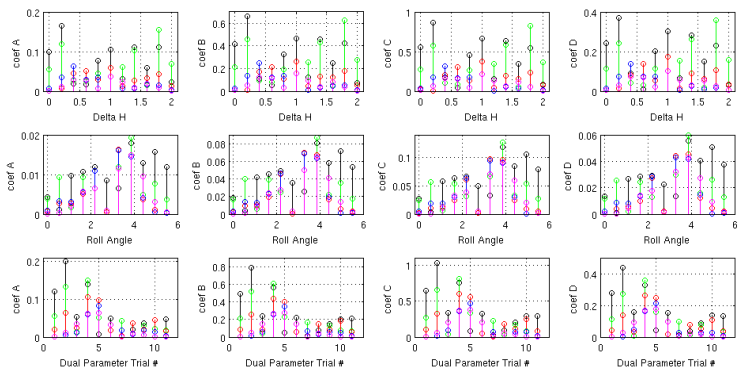


Figure 74: Coefficient Function Absolute Error

From these results, a parameter transfer function order of N = 5 appears to be the best choice in both the MSE and absolute sense. The coefficient ($\alpha_{n\rightarrow N}$, $\beta_{n\rightarrow N}$, $c_{n\rightarrow N}$, $d_{n\rightarrow N}$) values are listed in Table 8, Appendix G.

We therefore have the coefficients equations as a function of parameter (λ) value.

$$A = \sum_{n=0}^{N}\alpha_n\lambda^n \qquad B = \sum_{n=0}^{N}\beta_n\lambda^n \qquad C = \sum_{n=0}^{N}c_n\lambda^n \qquad D = \sum_{n=0}^{N}d_n\lambda^n \tag{170}$$

For the parameter options: $\lambda \rightarrow \lambda_{(\gamma pk)}$ or $\lambda_{(\Delta hpZ)}$ or $\lambda_{(\gamma pk + \Delta hpZ)}$

Substituting back into equation 165, we have the following equation:

$$S_m = \left(\sum_{n=0}^{N} \alpha_n \lambda^n\right) W_f{}^3 + \left(\sum_{n=0}^{N} \beta_n \lambda^n\right) W_f{}^2 + \left(\sum_{n=0}^{N} c_n \lambda^n\right) W_f + \left(\sum_{n=0}^{N} d_n \lambda^n\right) \tag{171}$$

Where $\alpha_{(n\to N)}$, $\beta_{(n\to N)}$, and $c_{(n\to N)}$ are known empirically within the trial limits. We would like to solve the equations for any parameter value within the trial limits given W_f and S_m.

By expanding the equation and collecting terms we derive a Stability Margin function in terms of λ to the fitting order N (5):

$$S_m = \sum_{n=0}^{N} \left(\alpha_n W_f{}^3 + \beta_n W_f{}^2 + c_n W_f + d_n\right) \lambda^n \tag{172}$$

We can now solve for λ using a modified Newton-Raphson method that suits each compensation paramter combination (γ_{pk} , Δhp_Z, $\gamma_{pk} + \Delta hp_Z$):

$$\begin{aligned}
&for\ i = 1:1:Iteration_{MAX} \\
&\quad S_{m\,\mathrm{APPROX}} = \sum_{n=0}^{N} \left(\alpha_n W_f{}^3 + \beta_n W_f{}^2 + c_n W_f + d_n\right) \lambda_i{}^n \\
&\quad error = S_{m\,\mathrm{APPROX}} - S_{m\,\mathrm{TARGET}} \\
&\quad if\ |error| < precision_{SM} \\
&\qquad return\ \lambda_i \\
&\quad else \\
&\qquad h = Kp \cdot error + Kd \cdot \frac{d}{di} error \\
&\qquad \lambda_{i+1} = \lambda_i + h \\
&\quad if\ i = iteration_{MAX} \\
&\qquad convergence\ failure \\
&end
\end{aligned} \tag{173}$$

In the case where both γ_{pk} and Δhp_Z are used, λ is returned as an index value into the following vectors based on initial test values (Table 2):

$$\gamma_{pk} = 0.55\,\lambda - 0.55 \quad \text{and} \quad \Delta hp_Z = 0.2\,\lambda - 0.2$$

The algorithm parameters used are:

Initial guess: $\lambda_{i=1} = 0$

$\mathrm{Precision}_{SM}$: 0.01 cm

$\mathrm{Iteration}_{MAX}$: 1000

Step size parameters (Table 3)

	γ_{pk}	Δhp_Z	$\gamma_{pk} + \Delta hpZ$
Kp	-1.5	-1.2	-1.2
Kd	0.25	0.1	0.2

Table 3: Newton's Method Step Size Parameters

Building and tuning this algorithm proved to be quite challenging. The entire compensation parameter space occupies areas that responded very differently to an iterative convergent method. Maintaining very small stability margins showed to be very unstable and higher target values very under damped. To ensure stability, the entire parameter space need to be tested (indicated in Figure 137, Appendix F).

Figure 75 illustrates this conflicting tuning problem where higher gain is desired to speed up convergence as hundreds of iterations are required on one end, but existing instability makes it problematic on the other:

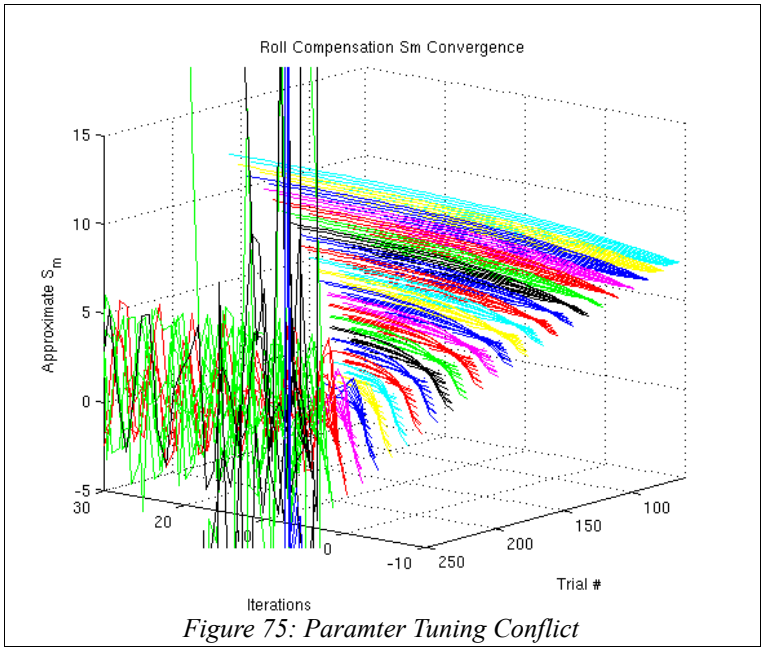


Figure 75: Paramter Tuning Conflict

Various methods were attempted to counteract the instability such as adding proportional, integrative, and derivative components to change the step size behaviour. Gains were made in convergence speed but the instability was a continuing issue. In the end, removing the $f(x)/f'(x)$ component from the Newton-Raphson method step size proved to be the solution, requiring then only an internal PD error controller. This converges quickly and stably, balancing both extremes with the appropriately tuned values and an initial seeding value of $\lambda = 0$. The algorithm converges between 4 to 25 iterations with very little overshoot on its most under damped slice of the parameter space. Results shown in Figures 138 – 141, Appendix F.

3.6.6. Simulation Results

The performance improvements using the torso roll and varying height strategy was measured by inserting the derived Stability Margin function into the walking model and observing the effect on the ZMP as the walk cycle frequency is increased. Figures 76 to 78 demonstrate the effect of removing 2 samples per period from the walk cycle frequency, beginning with 100 samples per period, until the compensation limit of each strategy is reached. Each parameter is compared to the uncompensated approach.

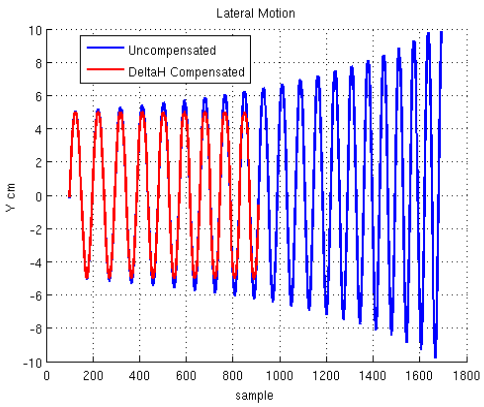


Figure 76: ZMP Development with Torso Height Variance Compensation

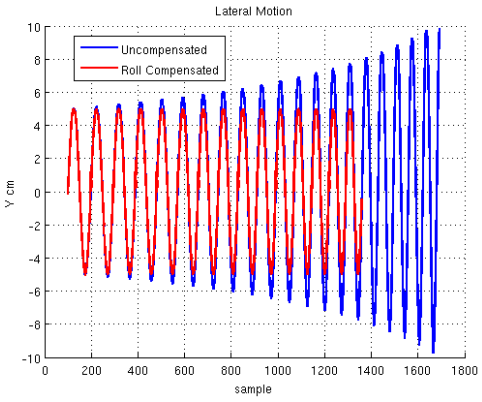


Figure 77: ZMP Development with Torso Roll Compensation

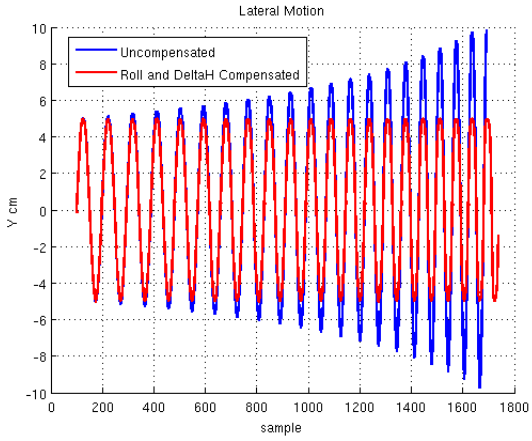


Figure 78: ZMP Development with both Torso Roll and Height Variance Compensation

Figure 79 shows the results compared to one another. Each compensation method ideally allows for a speed increase of up to:

Varying the torso height:	25 %
Rolling the torso inward:	51 %
Both parameters together:	79 %

The significant gain here is that while this approach maintains a comfortable stability margin of 5 cm up to 1.79 Hz, the robot should have definitely fallen by 1.64 Hz without compensation.

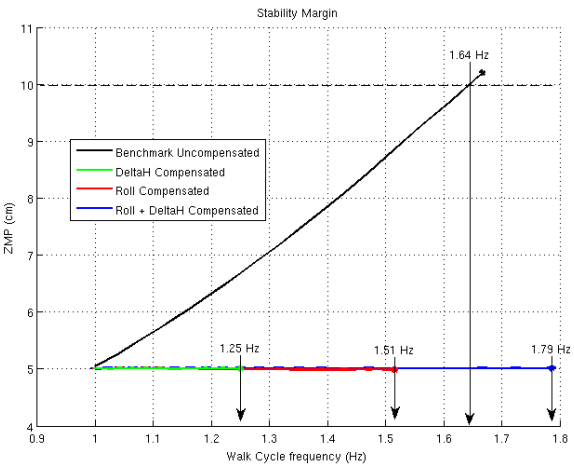


Figure 79: Maximum Walk Cycle Frequency Comparison

3.7. Stiffness Control

Once the system was migrated to the physical robot, walking was only achieved at a slow pace, or at a higher speed for only a few steps. These problems could be solved by correctly specifying the joint stiffnesses. A walk typically requires some sort of impact damping. The first version of the Nao walk required a different value for each joint as demonstrated by Jason Kulk [192]. Later versions of the Nao walk could use the same value for each joint but certainly never set to 1.0. To explore this issue further it seemed appropriate to use different values at different points in the walk, such as damping on impact and springing into the single support phase.

As this then requires different values at different times, applying another smooth sinusoidal signal to avoid sharp switching points made sense. Therefore a sinusoidal signal is created in the same way the torso height variation is (as shown in Figure 45) . It peaks at the sway peak which is the center of single support phase and reduces to a minimum value that occurs during dual support phase. This has the property of correcting for pose errors with a stiff leg when on one foot and also damping out impact generated errors when the foot lands.

The idea here is similar to other ZMP methods such as 'ZMP Preview Control' that generate regions of attraction for the ZMP at sway peak of the walk cycle. The difference here is that there is no implicit use of control system and that an additional region of attraction is created at the midpoint of the walk cycle. Here regions of attraction are generated by smoothly alternating between damper and spring like properties (Figure 80).

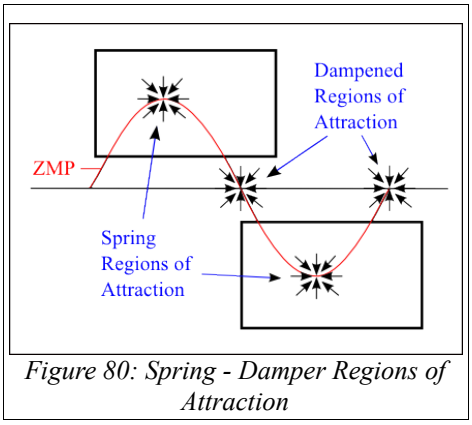


Figure 80: Spring - Damper Regions of Attraction

When the ZMP is directly over the center of the supporting foot in single support, the ankle motor will not be required to act to maintain the pose. If the ZMP lies at some other location in side the foot area, the ankle servo motors will react to maintain the pose up to the holding torque limit. During dual support, when the holding torque is dropped, the robot will buckle and collapse in on itself, creating a region of attraction at the midpoint between the supporting feet. This will provide gait correction by rigidly controlling the single support pose at the sway peak, and damping out gait errors at the center of dual support. By increasing and decreasing the joint stiffness as a function of the walk cycle, these regions will grow and recede along the ZMP path as the walk cycle progresses. This strategy will also transition smoothly between each gait correction zone, creating a balance between peak zones at any instant during the walk cycle.

Stiffness calculation:

Create a pendulum arm length proportional to the hip sway peak:

$$K_{P\ell} = 2.5 \tag{174}$$

$$P_\ell = K_{P\ell} Hp_Y \tag{175}$$

Update the pendulum motion with the walk cycle phase:

$$H_\Delta = P_\ell \cos\left(asin\left(\frac{Hp_{y[n]}}{P_\ell}\right)\sin(2\pi\phi_W)\right) - P_\ell \cos\left(asin\left(\frac{Hp_{y[n]}}{P_\ell}\right)\right) \tag{176}$$

Set maximum range of pendulum motion:

$$H_{\Delta MAX} = P_\ell - P_\ell \cos(asin(\frac{Hp_{y[n]}}{P_\ell})) \quad (177)$$

Use the ratio of pendulum position to maximum range of motion (in Z dimension) to set the value of all of the joints:

$$allJointStiffness = Stiff_{MIN} + \frac{H_\Delta}{H_{\Delta MAX}}(Stiff_{MAX} - Stiff_{MIN}) \quad (178)$$

3.7.1. Evaluation

The effectiveness of this approach was evaluated on the physical robot. This property of the gait is however, very difficult to measure with instruments. In the literature, the conventional methods are the Stability Margin criteria and the Poincare return maps used to evaluate limit cycles. With the appropriate model, these measures can be taken in simulation where all physical properties are implicitly known. These two measures however do not lend themselves well to all robot platforms, nor can they alone measure all gait properties. The Nao does have foot pressure sensors but they are very noisy and nonlinear, making them difficult to work with. Furthermore, ground contact with the Nao is intermittent when operating at the speed limit (our region of concern). Building a Poincare map does not reveal useful information when measuring joint angles as this is an active position controlled robot, where the joint angles are tightly controlled in comparison to passive walkers. When the robot is tipped over on the edge of its foot under rigid control, a Poincare map that is built using joint angles does not reveal the stability properties we are looking for, that is, its ability to recover from large disturbances (the tendency to wobble and recover).

This method of evaluating joint angles is perhaps more appropriate for passive walkers or any that are not possessing high gear ratio electric motors, or for simply small disturbance rejection where tipping does not occur. Also, when the robot is tipping on the edge of it's foot, the ZMP has left the support polygon and has become theoretical in nature and much harder to determine, rendering ZMP methods of little use. This leaves us with the basic definition of stability from literature that is a massive grey zone. The formal definition to date is 'either the robot did or did not fall'. 'The robot really looks like it is going to fall any second or with the slightest additional perturbation but does not' still falls under the definition of stable, though when observed it is readily obvious that the gait is poorly parametrized. There however exists no formal definition of stability that captures the subtleties of a humanoid gait. In bio-medical practice, where measurements can not capture abstract gait properties, visual observation and practitioners experience of limb motion is relied upon. The motivation behind this stiffness approach is to improve this difficult to quantify abstract property of the gait.

The improvement we are looking for is overall gait correction at its limit of performance such that the maximum velocity could be increased. All robots will begin to wobble backwards and forwards and from side to side as the stability margin approaches zero while increasing the walk cycle frequency. The Nao robot has round shaped feet that make it somewhat unique. Typically robots to date are constructed with rectangular feet. This in theory improves gait stability in the way that a cylindrical column may reject perturbations more so than a rectangular one, by oscillating orbitally as opposed to toppling over. This effect is readily observable on the Nao robot. The Aldebaran walking system very often recovers from pivoting violently, presumably better then it would with rectangular feet. There is however a limit to the effectiveness of this property alone. After working with the Nao robot for 4 years, experience has shown that once these orbital oscillation begin to appear, they will more than likely grow with each step when the robot is driven at a high speed. Occasionally, it surprisingly recovers from very steep inclines. It appears to be a function of walking velocity, which is a combination of step length and motion speed. The method we use in practice to find a comfortable balance between speed and stability for RoboCup competitions, is to manually strike the robot in the shoulders at random points in the walk cycle with a force strong enough to tip the robot on the edge of its feet (~10 – 15 degree roll) and then observe the robots ability to recover. With repetition, a balance can be found between speed and stability (collisions between robot occur frequently during RoboCup matches).

When applying this method to the robot in conjunction with the varied stiffness, the results were very good. The problem of accumulating orbital oscillations were drastically reduced, in fast almost complete elimination. The method observably was able to dampen out disturbances over a few steps and return to a consistent gait. When falls occur, there would be more of a tendency to just go strait

down to one side than to wobble for a while first. This is a property that is difficult to quantify as manually striking the robot is not consistently measurable without constructing a very elaborate experimental apparatus. Despite this, we used the same *conservative* test criteria of manually disturbing the robot while walking and observed for the same performance. Finding a parameter tuning balance between increased velocity and zero falls. Beginning with the Aldebaran 10 cm/s velocity (1 Hz frequency and 10 cm step length), improvements were made up to 13 cm/s (1.11 Hz and 12 cm step length) which is 30% faster. There was certainly room for further velocity gains, but the same level of conservatism was used as would be while tuning for a RoboCup challenge.

At this point, the walking component of the thesis was set aside to continue working towards a fluid kick motion. In hindsight, we recognize the robot's torso does have a naturally oscillating motion that should be discernible in the inertial unit measurements. It may be possible to observe and measure limit cycle stability with this data, and this we leave to future work.

3.8. Summary

The walk engine designed was completed with the main objectives satisfied:

- Fully parametrized and able to duplicate the gait of the Aldebaran walk.
- Able to walk as fast or faster than the Aldebaran walk.
- Statically stable.
- Room for further improvements with the introduction of new parameters (value added).
- Can vary velocity in real time (accelerate or decelerate into a kick).
- Omnidirectional enough to allow for flexibilty in the behavioural targeting system.

Figure 81 is a system level block diagram for the designed walk. The diagram includes implemented components in blue and propossed expansion blocks in pink that could potentially allow for high speed turning. Included static paramter values are shown in green. These are ones that would do well to be tuned as a function of walk cycle frequency via machine learning.

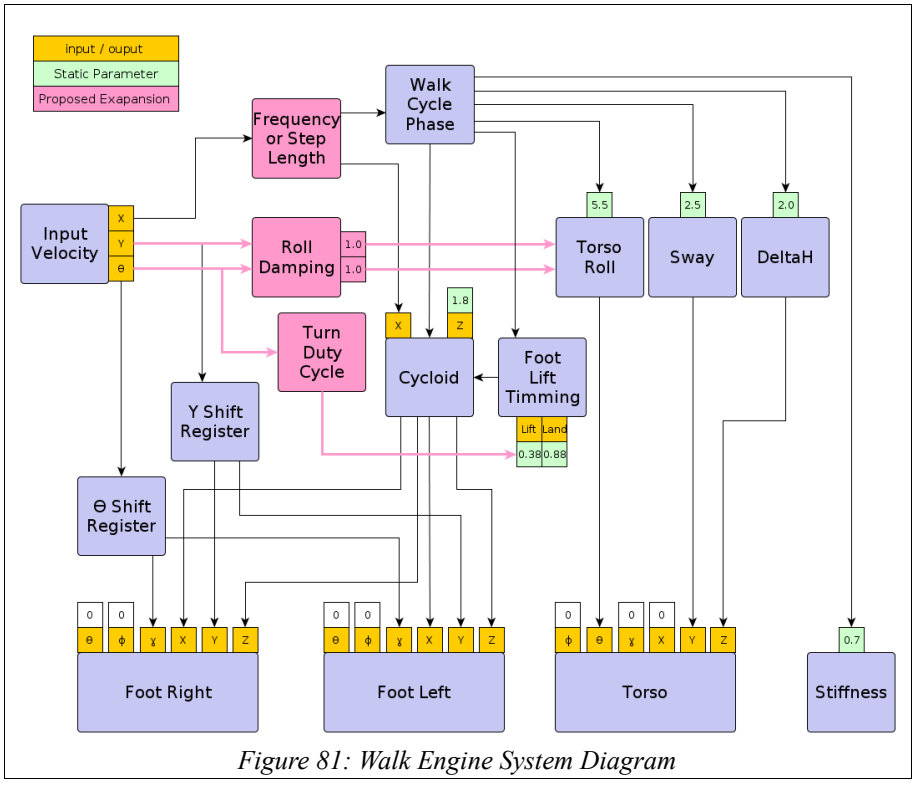


Figure 81: Walk Engine System Diagram

With this chapter completed, we are now ready to merge it with the Aldebaran system to achieve precise real time stepping control when required during a charge to the ball.

4. Walking Engine Bumpless Transfer System

4.1. Objectives

With the walk gait generator that is able to charge the ball for a kick completed, we want to smoothly merge it with the Aldebaran walk. The challenge here is that the Aldebaran walk is a closed system, that is, we do not have access to the internal code or required data such as the walk cycle phase and gait paramters such as step length, step height, torso displacement relative to foot trajectories etc. In order to transition smoothly between walks, the trajectories of the body parts during the Aldebaran walk must be known. Knowing the paramters of the Aldebaran walk would allow us to drive our robot model online in a way that produces limb trajectories that are matching. Therefore we require a method of monitoring the Aldebaran walk system to detect these properties. This will be our bumpless transfer system that is comprised of components that detect the sway, the walk gait paramters and the walk cycle phase (indicated in blue in Figure 82). The gait parameters are a set of periodic signals that can be monitored using peak detectors and the walk cycle phase can be extracted using a Phased Locked Loop (PLL).

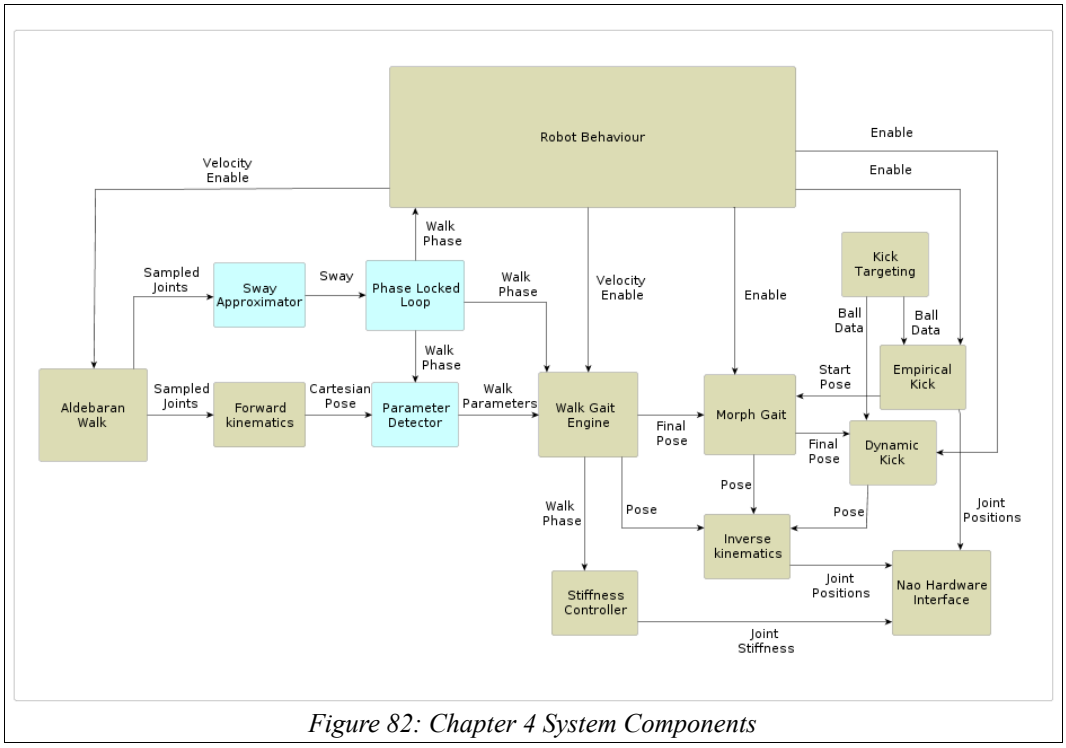


Figure 82: Chapter 4 System Components

4.2. Motivation

It is difficult to determine at what point the robot is in its walk cycle given only sampled joint information. Forward kinematics can be used to construct the robot's pose at each sample but connecting these poses together into a signal can be either inaccurate or noisy. A signal that is very useful in tracking the robots walk cycle is the lateral location of the torso, the robot's 'sway'.

Identifying the robot's current sway position can be attempted by building a spatial model of the the robot and tracking its torso location or by analyzing some other signal (for example hip roll). As the signal is a periodic one, we can identify some instantaneous phase within the signal period. For example identifying transition points by checking to see if the last amplitude value was greater than the current or looking for +ve/-ve transitions to either identify peaks or zero crossings. Unfortunately noise exists which renders such methods unusable as very frequently past values would appear to be transition points even though they are really not. To compound the issue, building a kinematic model to track the location of the torso spatially usually requires some concept of a supporting foot to either seed the model or to use as an origin for a frame of reference for the forward kinematics. Levelling the model on the support foot itself can be a difficult problem. Also, as the robot has two feet that it keeps transferring its weight to and from, there is a switching problem that occurs as we toggle between the supporting foot modes. This can be done at a variety of points, when one foot lands, or when a foot lifts off the ground, at the zenith of the sway peak or at the sway zero crossing. All will produce somewhat differing signals, even if just slightly. Let us say the point we want to transition at is the zero crossing, the worst time to perform the support mode switching would be at that same point. Figure 83 shows a computed sway signal that is the result of tracking the robots lateral torso position via kinematics.

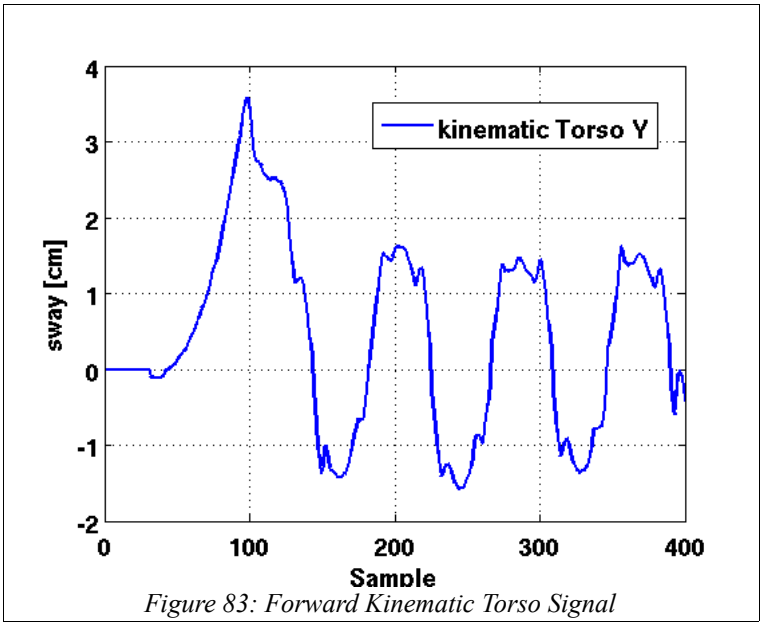


Figure 83: Forward Kinematic Torso Signal

There are many methods we could attempt to get around this noise problem. Low pass filtering is one way we could try to remove the noise but this would create a delay in the signal which is just as detrimental as the noise itself. The Nao robot allows us to read the joint output command buffer. This produces the best noise free signals but is not a true representation of where things are once you take the joint stiffness into consideration. Either way, some very accurate model of the robot is required to identify the walk cycle phase using kinematics alone.

Fortunately, there is a method employed in electronic systems that can solve this problem. It is the phase locked loop (PLL). It can be found for example, in communication systems such as signal repeater stations. They rely on a PLL to accurately detect the carrier signal that is inside the received noisy signal. It then uses this phase information to generate its own carrier signal at full power and noise free. This method would not only work well for this bumpless transfer problem, but allow us to worry less about the quality of the periodic signal we are monitoring as long as it is not delayed.

4.3. PLL Gait Synchronization

A PLL is a closed loop controller that drives a signal generator in reference to some set point signal (sinusoid) by manipulating some variable (instantaneous frequency of the signal generator). The transfer function blocks that make up a simple PPL are; a Phase Detector, a Loop Filter and a Voltage (or Digital) Controlled Oscillator (VCO/DCO). A block diagram for this PPL is shown in Figure 84:

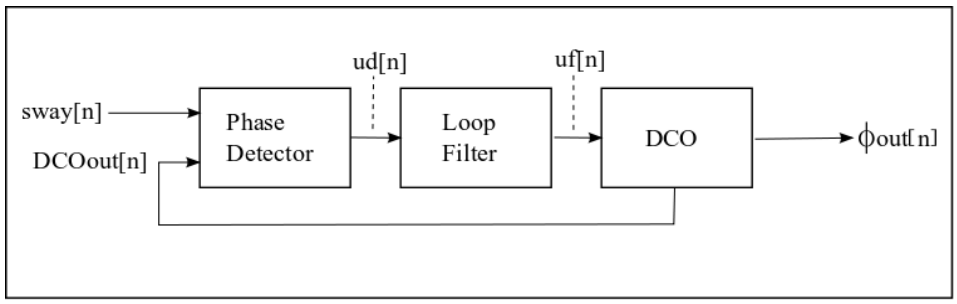


Figure 84: Walk Cycle Phase Locked Loop Block Diagram

The PLL implemented here follows that of the Digital PPL implemented by Hans Eklund [215] in his masters thesis, but with a slightly modified VCO. The transfer functions for these blocks are as follows:

4.3.1. Phase Detector

Finding the phase difference between two signals is quite simple. We multiply the instantaneous values of each signal together and the difference will be part of the result. This is basic signal modulation where the result of the signal mixing is two side band signals centred around the higher frequency signal as follows:

$$u_d = \sin(2\pi f_1 t)\cos(2\pi f_2 t) = \frac{1}{2}\left[\sin(2\pi(f_1 - f_2)t) + \sin(2\pi(f_1 + f_2)t)\right] \tag{179}$$

In our system this is the measured sway signal multiplied by the fed back DCO output:

$$u_d = sway\ DCO_{out} \tag{180}$$

The signal we want is the difference or lower side band. The instantaneous value of this lower side band represents the signal difference between the two input signals we need to drive the DCO. We can therefore use a low pass filter to block both noise and the higher frequency sideband.

4.3.2. Loop Filter

The filter we use is a discrete low pass IIR-filter with the following z-domain form:

$$F(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}} \tag{181}$$

Here the discrete filter coefficients are calculated as:

$$\begin{aligned} a_1 &= -1 \\ b_0 &= \frac{T_s}{2\tau_1}\left(1 + \frac{1}{\tan(T_s/2\tau_2)}\right) \\ b_1 &= \frac{T_s}{2\tau_1}\left(1 - \frac{1}{\tan(T_s/2\tau_2)}\right) \end{aligned} \tag{182}$$

This gives us the difference equation to use as a digital filter:

$$u_{f[n]} = b_0 u_{d[n]} + b_1 u_{d[n-1]} - a_1 u_{f[n-1]} \tag{183}$$

τ_1 and τ_2 were empirically found to be $\tau_1 = 1$, $\tau_2 = 0.9$ in this application.

T_S is the Nao sample period of 10ms.

4.3.3. Digital Controlled Oscillator

The DCO is is a phase driven component of the control loop. It is a signal generator that provides feedback for the loop. The phase is calculated from our filtered difference signal with the following relationship:

$$\phi_{[n]} = \phi_{[n-1]} + \frac{2\pi(f_c + K_d\, u_{f[n]})}{f_s} \tag{184}$$

This phase is then used to drive both the walking engine while it runs in parallel with the running black box walking system and to generate the PPL feed back signal.

The feedback signal:

$$DCO_{out[n]} = \sin(\phi_{[n]}) \tag{185}$$

The parameters here are:

$f_S \rightarrow$ The sample frequency ($1/T_S$ the Nao sample period)

$K_d \rightarrow$ The loop gain. Empirically found to be 0.3

$f_C \rightarrow$ The center frequency. The center frequency here assumes the walking system 'speed' or number of sample per step is constant and is evaluated offline by analyzing a recorded sway signal using the following method in MATLAB:

```
%% evaluate input signal natural frequency
timeMax = length(sig) * 0.01;
time = 0:0.01:timeMax - 0.01;
N = length(time)+1;

Ts = 0.01; % sample period
fs = 1/Ts;  % sample frequency

% FFT analysis
SIG = fft(sig);
freq = fs/N * (0 : N/2);

figure()
clf;
plot(freq, 2/N * abs(SIG(1 : N/2+1)));
% result from reading plot (natural frequency) -> 1.7 Hz
```

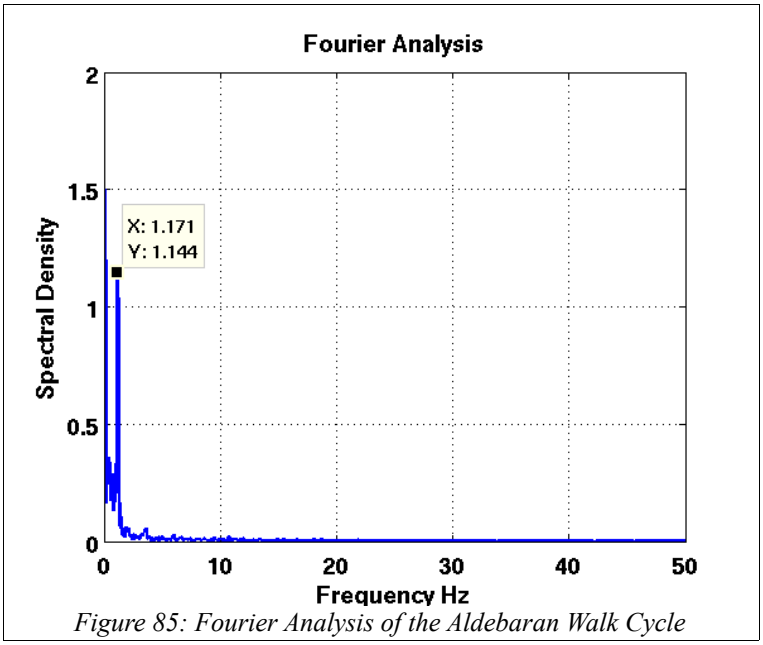


Figure 85: Fourier Analysis of the Aldebaran Walk Cycle

The Nao walk was determined to have a cycle frequency of 1.17 Hz then (2010) and later (2011) 1.7 Hz.

Keep in mind these results were taken from a particular walk calibrated using the calibration values used by the RoboCup team RoboEireann to achieve an optimal fast and stable walk. These sample rates are fixed during a walk, but variable in configuration.

There is also some signal conditioning that takes place inside the DCO. The phase calculated is integrated and should be unwrapped:

$$while\,(\phi_{[n]} > \pi) \quad \left\{ \phi_{[n]} = \phi_{[n]} - (2\pi) \right\} \tag{186}$$

The out going phase ϕ_{out} is normalized, -π to +π => 0 to 1.

$$\phi_{out[n]} = 0.5 + \frac{\phi_{[n]}}{2\pi} \tag{187}$$

4.4. Parameter Value Detection and Matching

There is a second source of discontinuity during transition between walks, one that is much more threatening than the noisy signal problem. This is a bump that occurs when switching between different process controllers. For example, a physical process that is first stabilized by a human controller and then switched to automatic control. An autopilot is one such automatic process controller that can be switched on and off in an aeroplane. Many industrial process switch between human operators or controllers. A bump in such instances could be either be destructive or dangerous.

The problem is that the controller that takes over from the first could come online with its internal variables at any given state, likely the initial settings or possibly whatever state it was in when last deactivated. In a control systems sense, if the feedback path of the second controller was not online, the controller would determine a different value for the manipulated variable than the first controller that has already stabilized the process. So to solve this problem what has to happen is the secondary controller that is going to take over must be running in parallel with the first controller so its internal state variables can match that of the online contoller only with its output disconnected from the system. This way the second controller is actually tracking the actions of the online controller. In our problem we have two phase generators, the second phase generator must take over from the first. However, these phase generators are driving a complex set of actions or equations. The incidental values of the step length, height, sway, arm positions etc. must be known in order for a smooth takeover to be performed. Therefore some duration of parameter matching must occur before one walk controller can hand over to another smoothly.

The parameters and method used to detect them are as follows:

4.4.1. Sway Peak

The sway peak is found using a signal peak detector that is reset twice on every cycle at some constant points. These constant points are set to be just prior to the signal peaks so that a value is captured for each sway zenith, not just the largest sway value that has occurred over its running time. The input sway signal is rectified and fed into a latch that captures the peak value around the signal zenith points as shown in Figure 86:

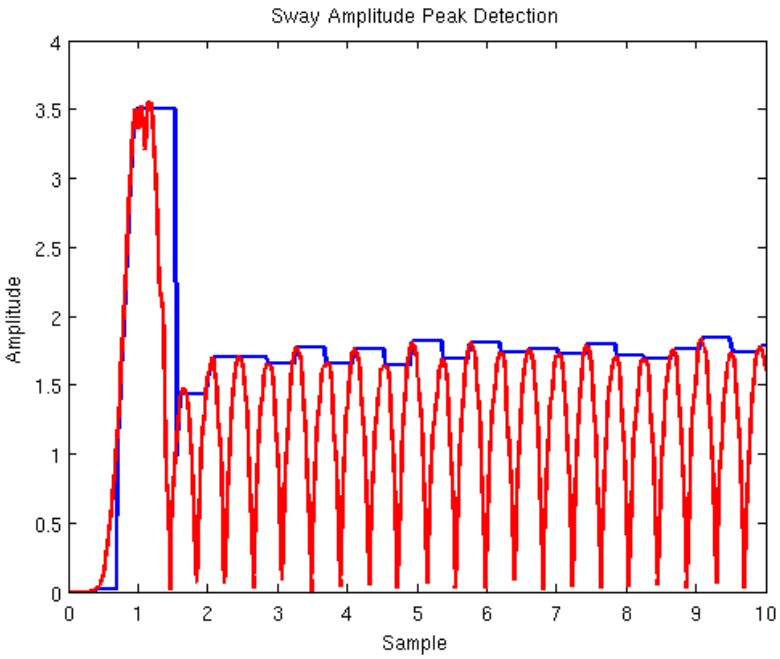


Figure 86: Sway Amplitude Peak Detector

The solution is a two step operation; calculate the sway signal, then process the sway with a peak detector:

4.4.1.1. Sway Calculation

The sway is measured by taking the average of multiple methods that build an approximate trapezoidal representation of the robots legs. At any given time each joint could be trying to realize the defined walking gait and also responding to some error in a closed loop scenario. Also, lowered joint stiffness values could obscure the ideal joint angle for the prescribed walk. So forward kinematics or joint angle trigonometry that relies on just one single joint or a select few tends to produce erratic signals. By using multiple methods simultaneously, the errors from each individual get dampened out resulting in the clean sinusoidal from Figure 86. Figure 87 illustrates this trapezoid method:

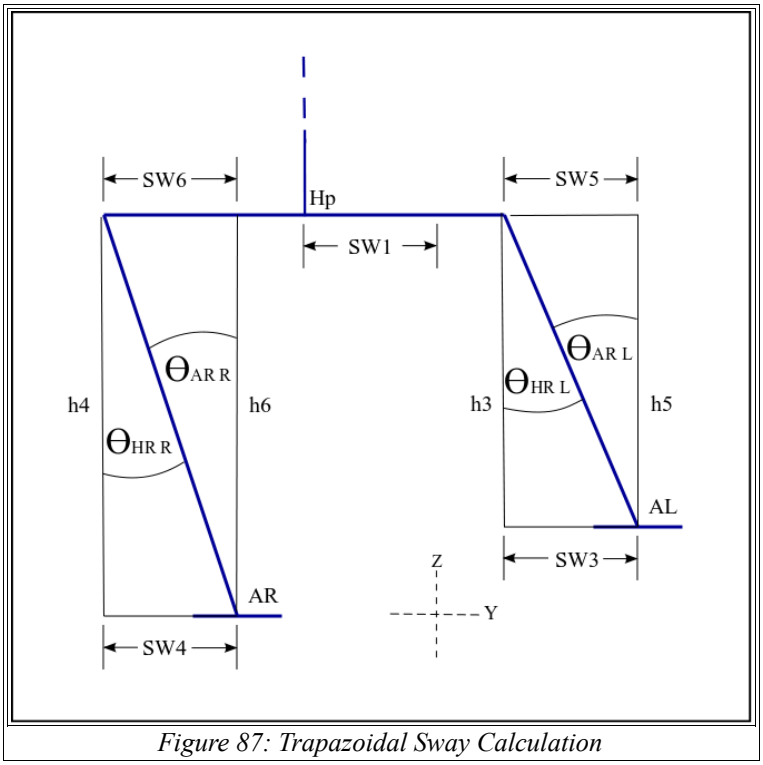


Figure 87: Trapazoidal Sway Calculation

Solution:

The sway of the running walk is evaluated as the average of 6 calculations:

First, the result found directly by the forward kinematics.

$$SWY1 = Hp_Y \tag{188}$$

The average lateral difference between the feet and hip also using the forward kinematic data:

$$SWY2 = \frac{(Hp_{Ly} - Ft_{Ly}) + (Hp_{Ry} - Ft_{Ry})}{2} \tag{189}$$

And triangles built using each leg roll motor:

$$\begin{aligned} SWY3 &= \tan(-\theta_{HRL})(Hp_z - Ft_{Lz}) \\ SWY4 &= \tan(-\theta_{HRR})(Hp_z - Ft_{Rz}) \\ SWY5 &= \tan(\theta_{ARL})(Hp_z - Ft_{Lz}) \\ SWY6 &= \tan(\theta_{ARR})(Hp_z - Ft_{Rz}) \end{aligned} \tag{190}$$

Finally, the sway average filter:

$$sway = \frac{(SWY1 + SWY2 + SWY3 + SWY4 + SWY5 + SWY6)}{6} \tag{191}$$

Figure 88 demonstrates this filtering method producing a final sway signal (rectified for observation purposes) that is a very good fit to the true value. The result is roughly 2.3cm at each peak. Though the Aldebaran walk does not have a sway parameter, during the process of calibrating the built walk engine to match the performance of Aldebarans, a sway value of 2.3cm was used as it appeared to be most stable. None of the individual signals produce the correct value over time. Each signal joint based signal has an asymmetrical characteristic about the x axis and the forward kinematic based signals (ankleDiff and HipCentery) look to be the worst. Which makes sense as error is accumulated and grows every time a another joint is included in the serial kinematic chain.

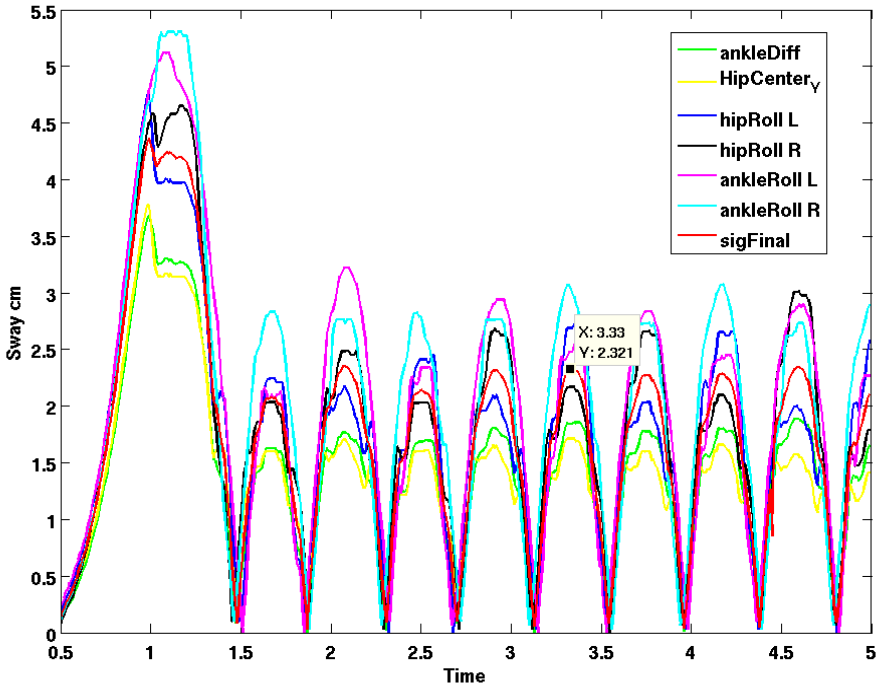


Figure 88: Comparison of Various Sway Signal Calculations

4.4.1.2. Sway Peak Detector

The sway peak detector operates as follows:

Rectify the the sway signal:

$$rect_{sig[n]} = fabs(sig_{[n]}) \tag{192}$$

Detect peak:

$$sway_{pk[n]} = \begin{Bmatrix} sway_{rct[n]} & : & sway_{rct[n]} & > & sway_{pk[n-1]} \\ sway_{pk[n-1]} & : & sway_{rct[n]} & <= & sway_{pk[n-1]} \end{Bmatrix} \tag{193}$$

Reset peak detector prior to opening latch:

$$sway_{pk[n]} = \begin{Bmatrix} 0 & : & (\phi_{out[n]} > 0.9) \;\; || \;\; (\phi_{out[n]} < 0.01) \\ 0 & : & (\phi_{out[n]} > 0.4) \;\&\&\; (\phi_{out[n]} < 0.50) \end{Bmatrix} \tag{194}$$

Latch operation:

$$latch_{[n]} = \begin{Bmatrix} sway_{pk[n]} & : & (\phi_{out[n]} > 0.01) \;\&\&\; (\phi_{out[n]} < 0.2) \\ sway_{pk[n]} & : & (\phi_{out[n]} > 0.50) \;\&\&\; (\phi_{out[n]} < 0.6) \\ latch_{[n-1]} & : & \phi_{out[n]} = \text{all other} \end{Bmatrix} \tag{195}$$

Output, torso lateral peak:

$$Tr_{y\,pk} = latch_{[n]} \tag{196}$$

4.4.2. Torso Height

The torso height is set as a constant value. It is a constant in the walk configuration used for this project. A walk that varied its height would require parameter detection such as the latest Aldebaran walk. This latest walk by Aldebaran is not used by RoboEireann as it is not stable.

4.4.3. Torso X Displacement

The forward torso displacement is found by assessing the midpoints of the foot cycloids and taking the average values. It relies on the values found in the step length detector (described in subsection 4.4.4.)

Solution:

Find step length mid points:

$$\begin{aligned} MidPoint_L &= \frac{(Ankle_{Pk\,POS\,L} + Ankle_{Pk\neg L})}{2} \\ MidPoint_R &= \frac{(Ankle_{Pk\,POS\,R} + Ankle_{Pk\neg R})}{2} \end{aligned} \tag{197}$$

Take the opposite of the average value:

$$Tr_x = -\frac{(MidPoint_L + MidPoint_R)}{2} \tag{198}$$

4.4.4. Step Length

The step length detector is also a peak detector that is reset each cycle. However, the foot location signal may not be as symmetrical as the sway, so a peak detector for both the positive and negative peak is implemented and the difference between them is found. The final value is then the average of both legs step length.

The following Figure 89 demonstrates the algorithm converging over a few steps, along with the torso X displacement:

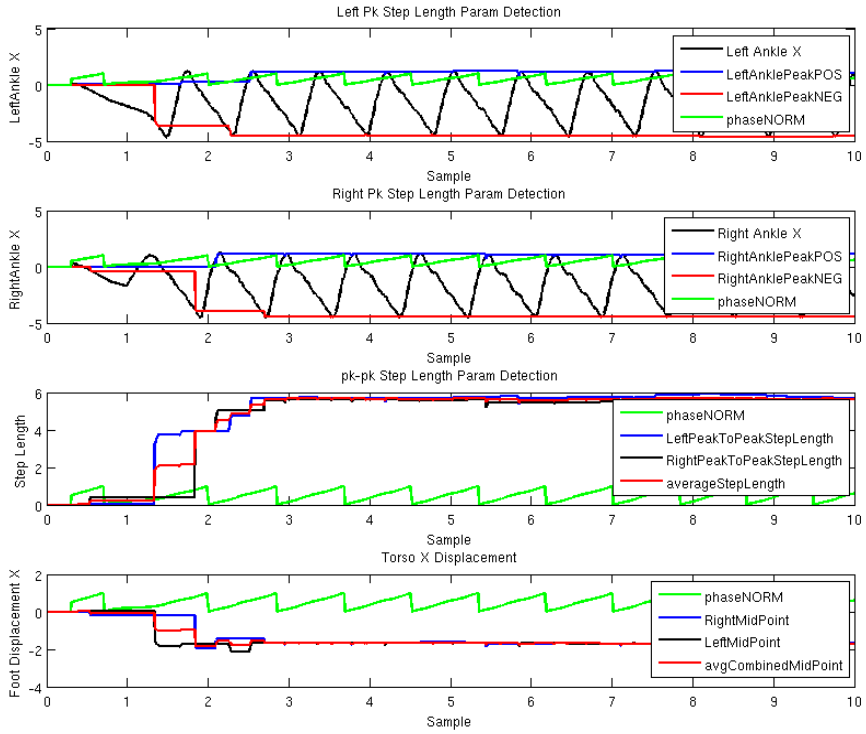


Figure 89: Offline Step Length and Torso Offset (cycloid trajectory center) Parameter Detectors

Walking gaits can vary considerably in terms of when in the walk cycle the feet lift off and land. So identification of phase point for each peak must be found by analyzing the foot trajectory signals and calibrating the phase point for the opening and closing of the latches. The following is a table of these latch trigger phases, listing the phase at which to reset the tracking signal, initiate the latch throughput and terminate the latch throughput.

Parameter	Phase
$\text{Reset}_{\text{POS L}}$	0.68
$\text{Reset}_{\text{POS R}}$	0.18
$\text{Reset}_{\text{NEG L}}$	0.26
$\text{Reset}_{\text{NEG R}}$	0.76
$\text{Track}_{\text{INIT POS L}}$	0.70
$\text{Track}_{\text{INIT POS R}}$	0.20
$\text{Track}_{\text{INIT NEG L}}$	0.28
$\text{Track}_{\text{INIT NEG R}}$	0.78
$\text{Track}_{\text{TERM POS L}}$	0.73
$\text{Track}_{\text{TERM POS R}}$	0.23
$\text{Track}_{\text{TERM NEG L}}$	0.31
$\text{Track}_{\text{TERM NEG R}}$	0.81

Table 4: StepLength Parameter Detection Latch Reset and Enable Phases

Solution (the same method is used for both feet):

Track increasing values for:

Peak positive:

$$track_{POS[n]} = \begin{Bmatrix} Ft_x & : & Ft_{x[n]} & > & track_{POS[n-1]} \\ track_{POS[n-1]} & : & Ft_{x[n]} & <= & track_{POS[n-1]} \end{Bmatrix} \tag{199}$$

Peak negative:

$$track_{\text{NEG}[n]} = \begin{Bmatrix} Ft_{x[n]} & : & Ft_{x[n]} & > & track_{\text{NEG}[n-1]} \\ track_{\text{NEG}[n-1]} & : & Ft_{x[n]} & <= & track_{\text{NEG}[n-1]} \end{Bmatrix} \tag{200}$$

Reset evaluator each cycle near but before the peak value:

Reset positive:

$$track_{POS[n]} = \begin{Bmatrix} 0 & : & (\phi_{out[n]} > Reset_{POS}) \;\&\&\; (\phi_{out[n]} < Track_{INIT\,POS}) \end{Bmatrix} \tag{201}$$

Reset negative:

$$track_{\text{NEG}[n]} = \begin{Bmatrix} 0 & : & (\phi_{out[n]} > Reset_{\text{NEG}}) \;\&\&\; (\phi_{out[n]} < Track_{INIT\,\text{NEG}}) \end{Bmatrix} \tag{202}$$

Latch values:

Latch positive:

$$peak_{POS[n]} = \begin{Bmatrix} track_{POS[n]} & : & (\phi_{out[n]} > track_{INIT\,POS}) \;\&\&\; (\phi_{out[n]} < track_{TERM\,POS}) \\ peak_{POS[n-1]} & : & \text{all other} \end{Bmatrix} \tag{203}$$

Latch negative:

$$peak_{\text{NEG}[n]} = \begin{Bmatrix} track_{\text{NEG}[n]} & : & (\phi_{out[n]} > track_{INIT\,\text{NEG}}) \;\&\&\; (\phi_{out[n]} < track_{TERM\,\text{NEG}}) \\ peak_{\text{NEG}[n-1]} & : & \text{all other} \end{Bmatrix} \tag{204}$$

Find peak to peak values:

$$\begin{aligned} Sl_{pk-pk\,L[n]} &= peak_{POS\,L[n]} - peak_{NEG\,L[n]} \\ Sl_{pk-pk\,R[n]} &= peak_{POS\,R[n]} - peak_{NEG\,R[n]} \end{aligned} \tag{205}$$

And then the average step length:

$$Sl_{AVG[n]} = \frac{(Sl_{pk-pk\,L[n]} + Sl_{pk-pk\,R[n]})}{2} \tag{206}$$

4.4.5. Step Height

The step height detector operates the same way with the slight difference that the peak detector tracks both the left and right foot simultaneously (Figure 90).

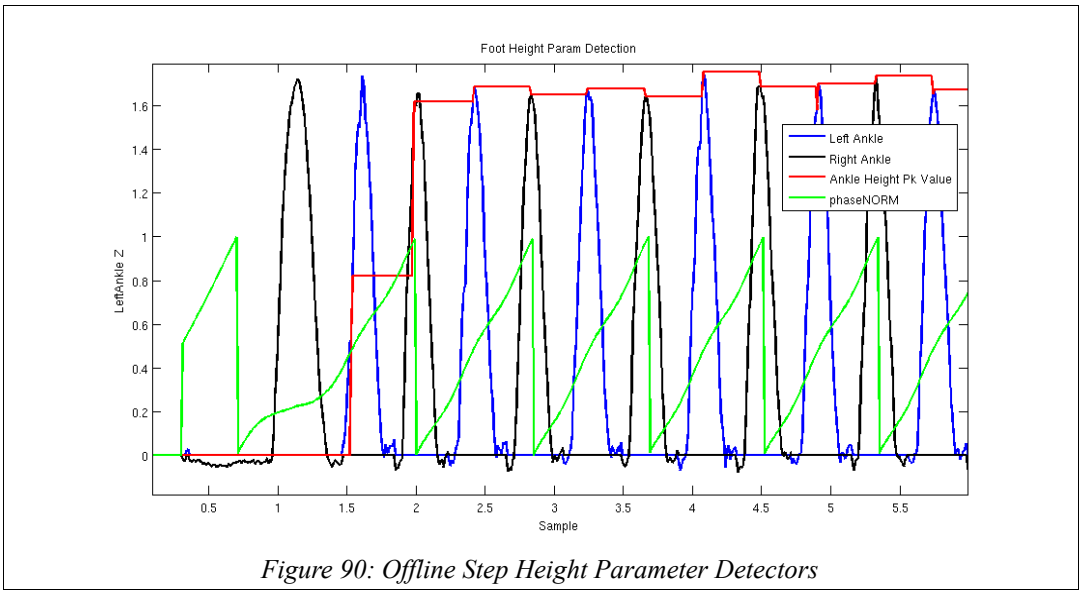


Figure 90: Offline Step Height Parameter Detectors

Table of parameters:

Parameter	Phase
reset $_{L}$	0.95
reset $_{R}$	0.45
track $_{INIT\,L}$	0.97
track $_{INIT\,R}$	0.47
track $_{TERM\,L}$	1.00
track $_{TERM\,R}$	0.50

Table 5: StepHeight Parameter Detection Latch Reset and Enable Phases

Solution:

Track only increasing values:

$$track_{[n]} = \left\{ \begin{array}{l} track_{[n]} = \left\{ \begin{array}{lllll} Ft_{zL[n]} & : & Ft_{zL[n]} & > & track_{[n-1]} \\ track_{[n-1]} & : & Ft_{zL[n]} & <= & track_{[n-1]} \end{array} \right\} : Ft_{zL} > Ft_{zR} \\ track_{[n]} = \left\{ \begin{array}{lllll} Ft_{zR[n]} & : & Ft_{zR[n]} & > & track_{[n-1]} \\ track_{[n-1]} & : & Ft_{zR[n]} & <= & track_{[n-1]} \end{array} \right\} : Ft_{zL} < Ft_{zR} \end{array} \right\} \tag{207}$$

Reset tracking signal in off cycle:

$$track_{[n]} = \begin{Bmatrix} 0 & : & (\phi_{out[n]} > reset_L) \;\&\&\; (\phi_{out[n]} < track_{INIT\,L}) \\ 0 & : & (\phi_{out[n]} > reset_R) \;\&\&\; (\phi_{out[n]} < track_{INIT\,R}) \end{Bmatrix} \quad (208)$$

Latch in peak values:

$$Ft_{z[n]} = \begin{Bmatrix} track_{[n]} & : & (\phi_{out[n]} > track_{INIT\,L}) \;\&\&\; (\phi_{out[n]} < track_{TERM\,L}) \\ track_{[n]} & : & (\phi_{out[n]} > track_{INIT\,R}) \;\&\&\; (\phi_{out[n]} < track_{TERM\,R}) \\ Ft_{z[n-1]} & : & \text{all other} \end{Bmatrix} \quad (209)$$

4.4.6. Step Width

The step width is assumed to be a constant value and is set manually during calibration. Typically the hip width is the same as the step width.

4.5. Implementation and Performance Analysis

The performance of this system was first tuned and tested in simulation. The kinematic data was recorded from the Aldebaran walk and the PPL was simulated in MATLAB. Over the course of the project this PLL was used to synchronize the walking gait with two different systems on the robot itself; the Aldebaran walk and the B-Human walk.

4.5.1. Simulation

The PLL was first tested offline in MATLAB after recording data from a walk. The data was padded with a couple of random length pulses to make sure the data was not coincidentally in phase with the signal generator to begin with and to give it a bit of a kick with broad spectrum frequency components. There the filter parameters and loop gain etc were tuned. Results are shown in Figure 91:

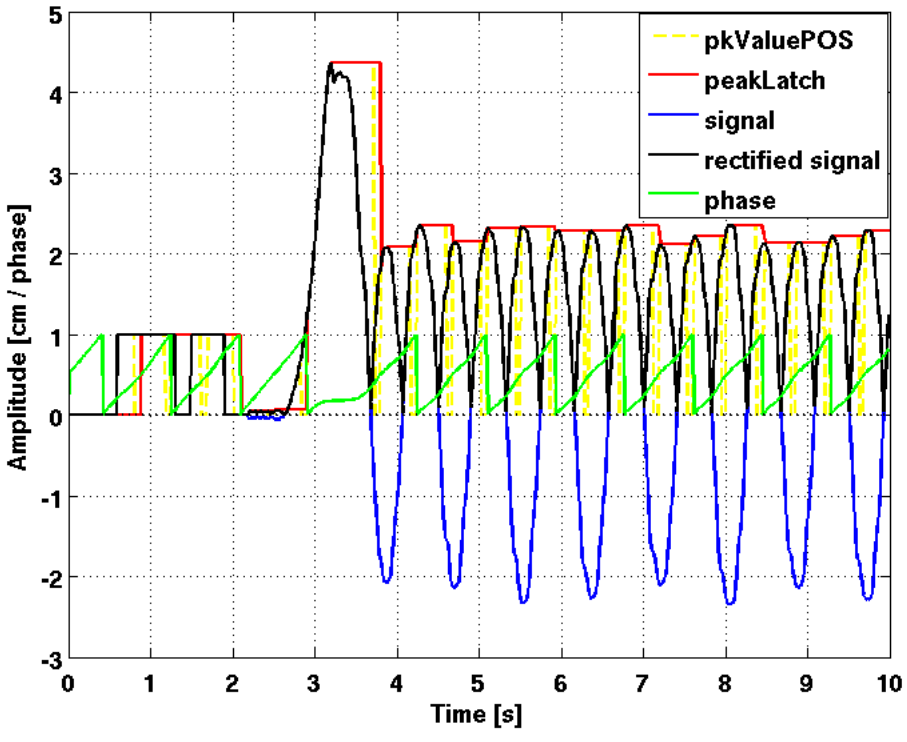


Figure 91: Offline PLL Calibration

From this plot we can see the controller running at it natural frequency inside the high frequency padding period to start and then responds very quickly with the phase locking in over just one period of sway data.

The next figure (92) is a comparison of an offline sway signal generated from the PLL system output phase with the original measured sway. The noticeable jumps at the peaks here is the signal generator responding to the peak detector updates.

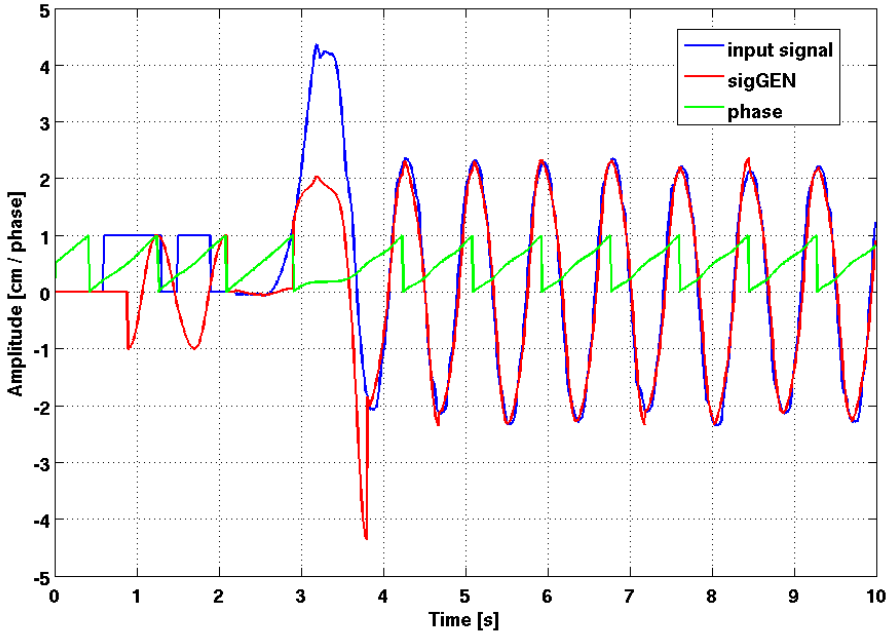


Figure 92: Offline Signal Generator Analysis

4.5.2. Hardware Performance Analysis

Once tuned offline, the PLL and parameter matching system was ported to C++ and run on the physical robot. The main point of interest is the acquisition speed, that is, how rapidly the system locks in and matches the gait. The results here (Figure 93 to 95) show a hardware trial over a period of 3 walking cycles. We can observe the PLL locks in quite quickly with both the step height and length parameter detection methods converging within these 3 walking cycles.

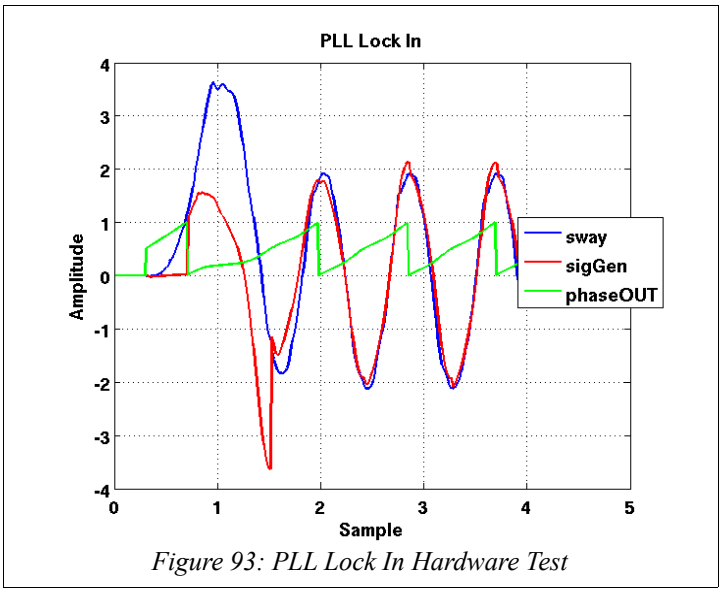


Figure 93: PLL Lock In Hardware Test

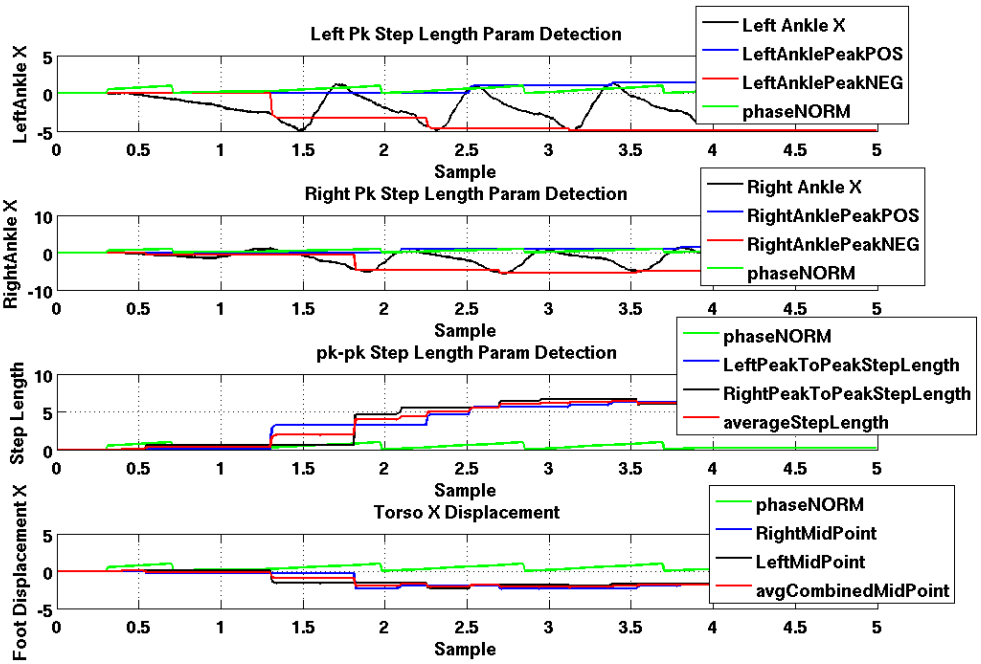


Figure 94: Online Step Length and Torso Offset (cycloid trajectory center) Parameter Detectors

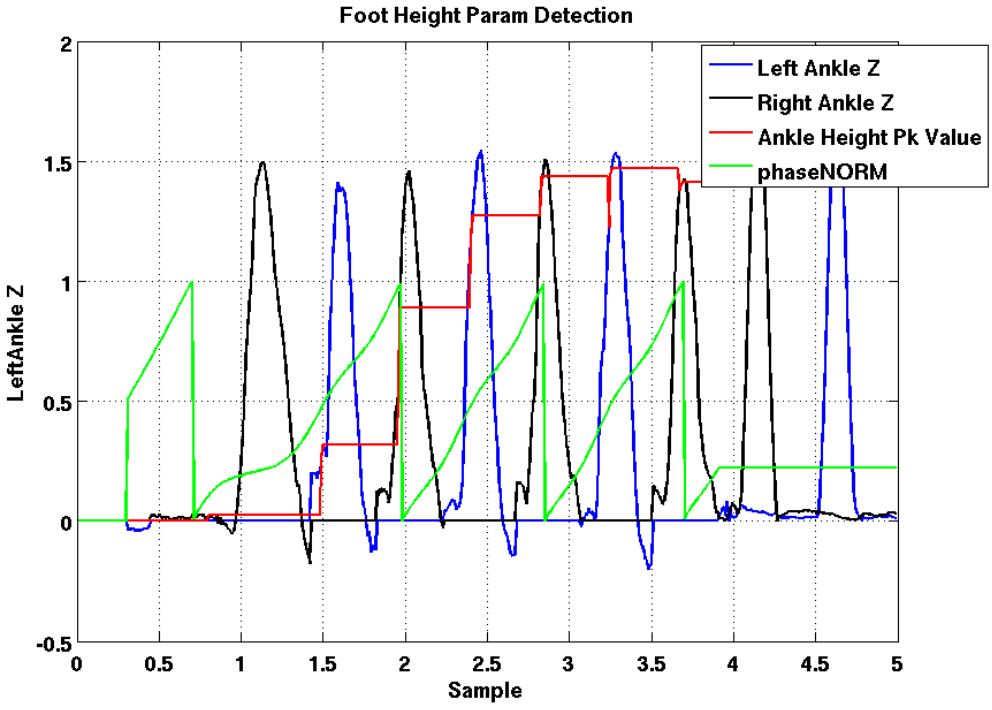


Figure 95: Online Step Height Parameter Detectors

The system was then set to run for a long duration (30 transitions), switching back and forth between the Aldebaran walk and the controlled walk that takes over as shown in Figure 96.

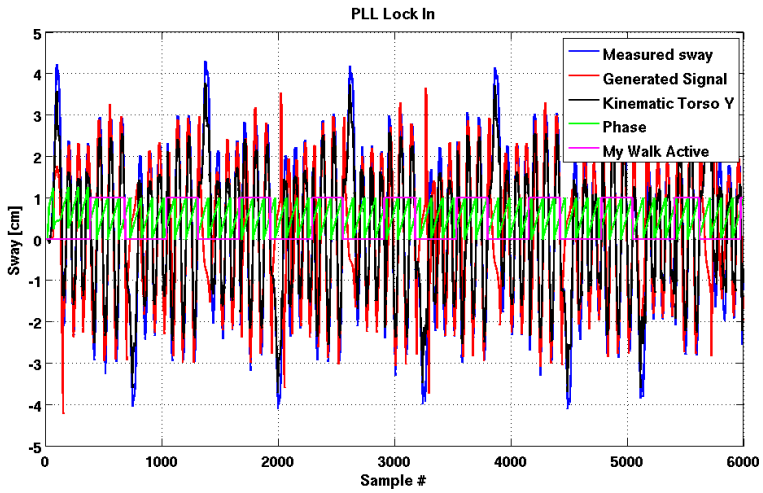


Figure 96: Extended Running Tests Switching Back and Forth Between Walking Engines

This test was performed on a short track, with the Aldebaran system taking three steps, the custom walk takes three steps and then the robot was quickly replaced at the top of the track while the Aldebaran system reengages. As it can be seen in Figure 97, the Aldebaran walk first must reset from the intital position, taking one initial step requiring a larger sway. This provided an opourtunity to move the robot though it has to be as fast as manually posssible as it had already bugen to walk during placement and the first step was manulaly stabilised in nessesary. This introduced a random perturbation into the test that made for a good test of robustness as there would be some existing instability and irregularity in the Aldebaran walk while it is being monitored and transfering.

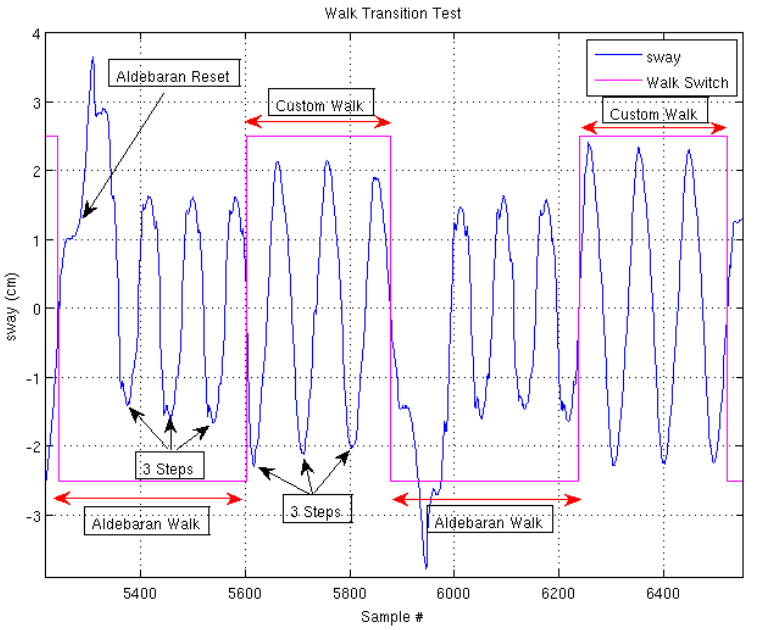


Figure 97: Walk Cycles During Transition Test

To evaluate the existence of a bump we chose to mesure the motion in a Hip Roll joint over a window of s = -5 to 12 samples, with the walk control transfer occuring at s = 0 (Figure 103).

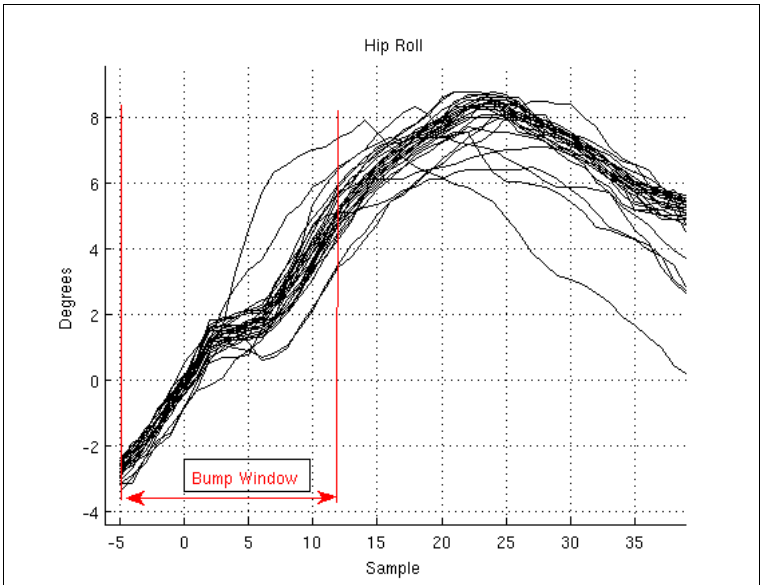


Figure 98: Hip Roll Joint Trajectoies During 30 Control Transfers

From these signals we computed the velocity, accelerattion and jerk (Figure 51). This revealed a minor bump occuring 30 ms after the transition, as the new walk takes over:

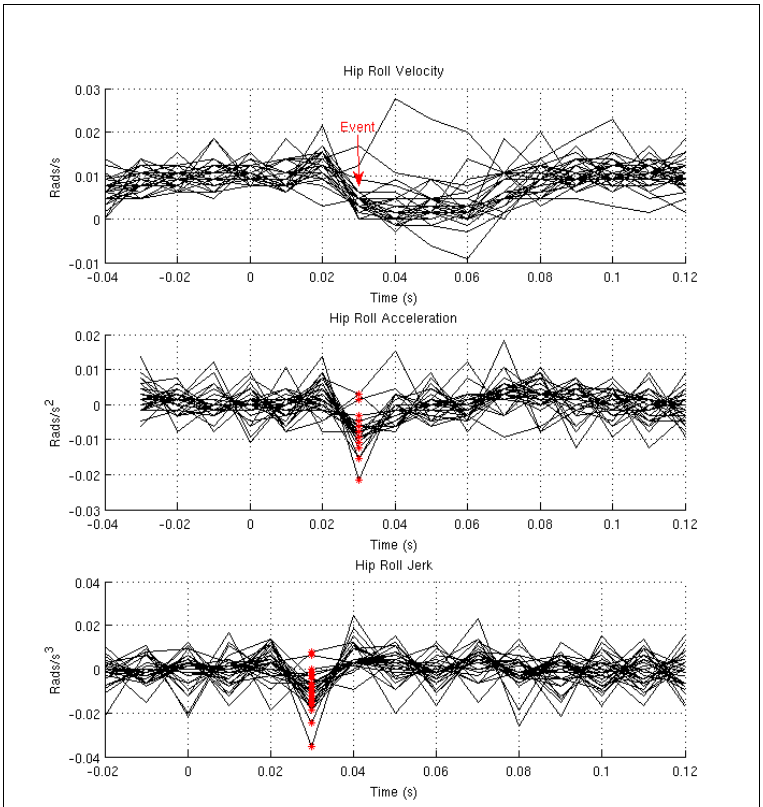


Figure 99: Motion Analysis During Transfer

We were able to reduce to reduce the bump to the following magnitiude:

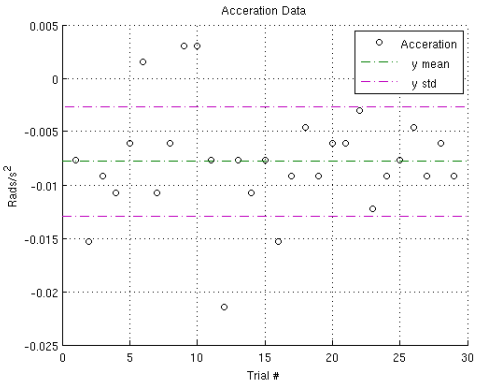


Figure 100: Bump Acceleration Data

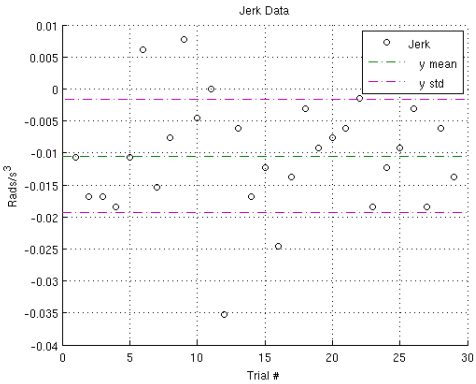


Figure 101: Bump Jerk Data

Acceleration:
mean -7.77 x 10^{-3} rad/s^2
std -5.14 x 10^{-3} rad/s^2

Jerk:
mean -10.5 x 10^{-3} rad/s^3
std - 8.85 x 10^{-3} rad/s^3

At this magnitude, the disturbance is quite inaudible. It is also highly likely that this event is a result of activating the varying joint stiffness system.

4.6. Discussion and Conclusion

In conclusion, the final product works very well. It is able to quickly and accurately determine the phase of the Aldebaran walk cycle and use the phase information to drive a secondary walking engine, match its parameters and perform a smooth bumpless transfer at the Aldebarans top velocity consistently. Significant changes to the Aldebaran walk would call for some calibration adjustments as there are some static values within the system such as the center frequency (f_c) of the walk controller and the values listed in Table 4 and 5 that set the tracking reset and latch window edge phases. This is just a matter of reviewing the kinematic signals for the torso and foot positions offline. Something would have to change a lot though as these windows can tolerate a considerable amount of variation.

Finally, a byproduct of this PLL is its usefulness with other applications that require phase knowledge of the walk cycle other than switching to another walk or special motion. The time varying stiffness control method presented in Chapter 3 could be used alone with this PLL which would be one of the more portable stand alone products of this project. The components themselves are each quite small and could be implemented with ease. The stiffness control method could be applied to many walks, inlcuding the Aldebaran black box system as the stiffness is something that can be controlled in real time on a sample by sample basis.

4.7. Summary

With this bumpless transfer system complete, we are able to smoothly stitch our ball charging gait to the Aldebaran walk. This allows for accelerating towards the ball to kick and take accurately controlled steps. The next component that is required is one that can merge two different motions, such as a walk and a kick. This 'morphing gait' and kicking is covered in the next chapter.

5. Stepping into Kicking and Dynamic Kick Swing

5.1. Objectives

This chapter continues with the process of walking into a kick. This is morphing from a walk gait into a special motion. The walk termination point occurs midway through the dual support phase. In theory we could splice into any point in the walk phase to peform other motions, however we only use and have tested the point when the torso sway crosses the sagittal plane. The morph gait stiches a smooth transition trajectory between this pose and the first sample pose of the intended special motion. We assume that this first sample pose of the special motion is a statically stable one. For this morphing gait to be stable we are making the same assumption as in the one employed in the Theory Of Capture Points [133] method (section 2.1.7.6.3.). This states that if a motion begins and ends with a stable pose, the motion as a whole may be acomplished without falling even though any particular sample point along the way may not be statically stable.

This process is therefore separated into two parts, the transition from the walk to a static back swing pose ('morphing') and the dynamic kick swing implementing a cubic splines method for dynamic targeting and parametrization of the swing itself for maximum foot swing velocity. These are the two final core components (Figure 102):

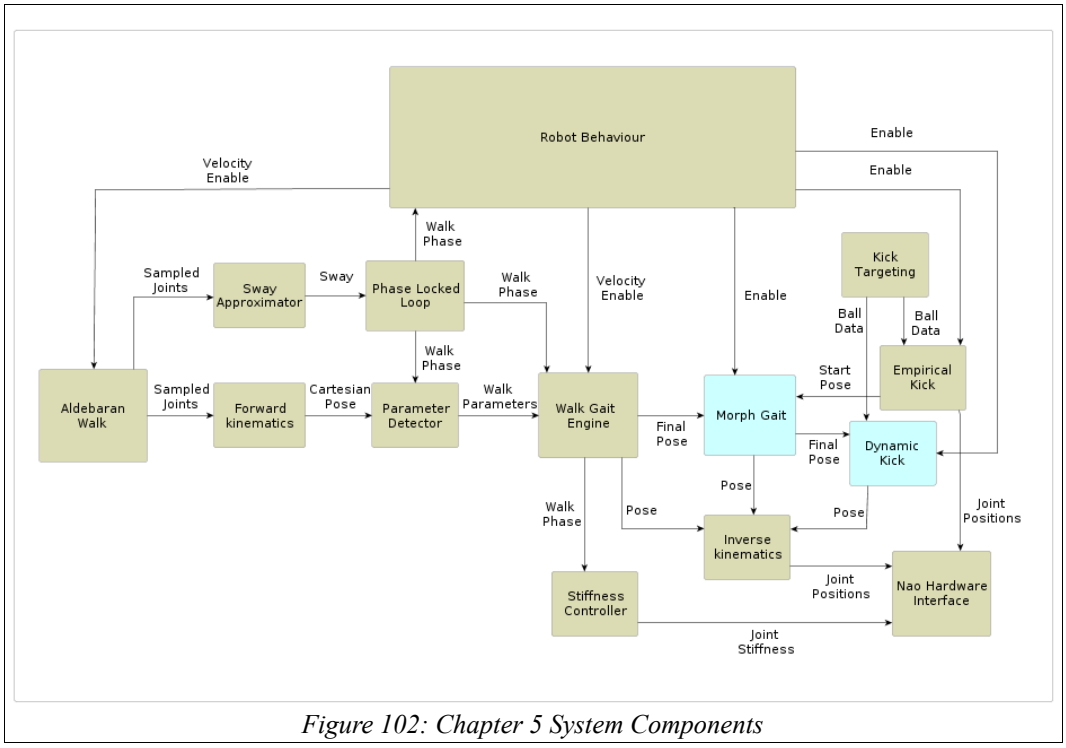


Figure 102: Chapter 5 System Components

5.2. Transition from walk to static back swing pose

The inverse kinematics allows us to define any target pose by specifying the position and attitude of both the feet and the torso. What we need to do is map a trajectory from the walking trajectory into this defined pose. The kick pose here is a statically defined one but the walking trajectory can enter this process with some variability. The morphing trajectory will then have to be a dynamic one. This is accomplished then by interpolating from whatever the final walking pose was through to the first back swing pose for the kick.

There are two coordinate systems we can use to define these trajectories, the robot's own coordinate system or the world coordinate system. Often these are one and the same but for some applications there can be some differences and dependencies on other factors. What we have to be aware of here is the ground contact of the feet. This back swing pose has the robot standing on one foot. To stand on one foot we could place the foot under the torso in the robot coordinate system, but that would relate the robot coordinate system more towards the torso frame, the feet would be moving relative to the upper body. Given that we are trying to kick the ball and the targeting data is in the world coordinate system we then want to position the torso over the foot location instead. This means then, the feet are stationary and we do not have to be concerned with the smoothness of the connection when splicing the walking and morphing trajectories, as the feet will either be remaining stationary (supporting foot) or departing from a stationary position (kick swing foot). A linear interpolation will do just fine here. The torso, however, is initially already in motion, mostly laterally, it would be best then to interpolate preserving the smoothness in the motion. A cubic spline interpolation is appropriate therefore. This method of interpolation will be discussed later in subsection 5.3.1. Figure 103 shows the control of the robot sway through each phase. Figure 104 and 105 are the signal trajectories mapped for the torso swinging foot (stationary foot does not move).

The use of the cubic spline also allows us to define some point midway (or any other phase) through the morph phase. This is used as a means for shaping the lateral portion of the trajectory. The sway in this motion is much greater than the walking sway and therefore will push the ZMP outward beyond the support polygon. However, the act of hoisting the swinging foot upward creates an opposite force that will counter-balance the sway. Should there be any uneven balance, this midpoint could be used to offset the difference. It has not been used in practice in the lab, it was found the two motions do counter each other well.

The final property of these trajectories is the launch time for the swing foot. As the motion is very similar to the walk itself, the sway trigger point for the foot lift off from the walk is used here as well. This then means the foot will have a different phase variable than the torso and the two are are offset from each other.

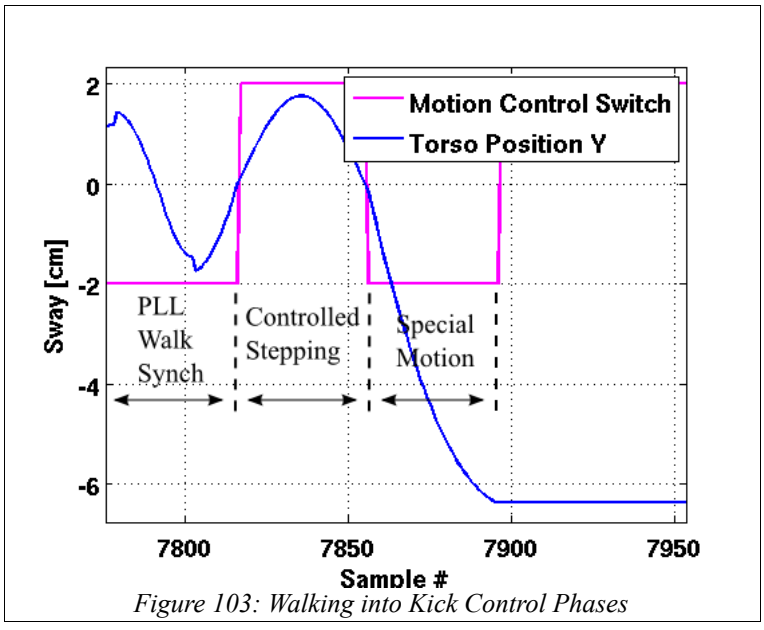


Figure 103: Walking into Kick Control Phases

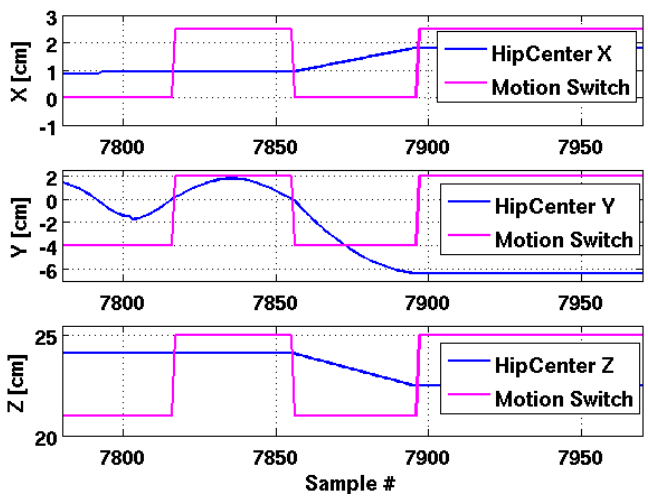


Figure 104: Torso Signal Trajectory Mapping, Morphing from Walks into Kick

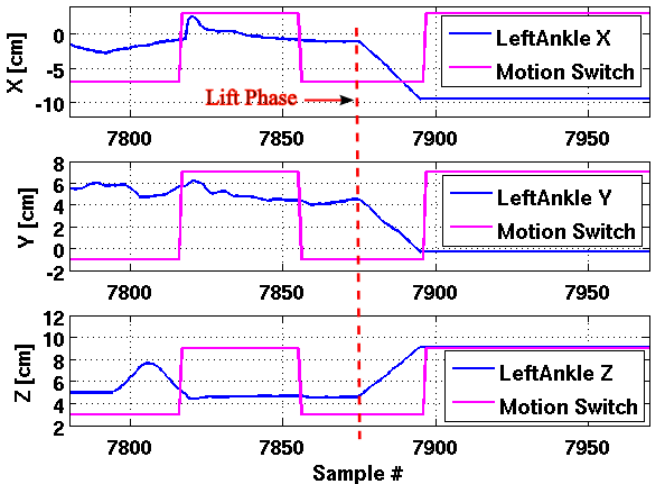


Figure 105: Kick Foot Signal Trajectory Mapping, Morphing from Walks into Kick

Solution to mapping this signal trajectories from the walk through to the back swing pose:
Initialization operations:

Set morph phase increment:

$$\phi_{MI} = 0.017 \tag{210}$$

Set swing foot back swing pitch:

$$Ft_{SWING\,\beta} = -\frac{\pi}{3} \tag{211}$$

Set mid point side indicator:

$$swaySideSign = \begin{Bmatrix} 1 & : & kickLeg & = & \text{left} \\ -1 & : & kickLeg & = & \text{right} \end{Bmatrix} \tag{212}$$

Set sway midpoint value:

$$Hp_{y\,midPt} = swaySideSign\ Hp_y \sin\left(\frac{\pi}{2}\right) \tag{213}$$

Load shoulder initial positions:

$$\begin{aligned} LShoulderPitch_{kick[\phi_M=0]} &= LShoulderPitch_{[n-1]} \\ RShoulderPitch_{kick[\phi_M=0]} &= RShoulderPitch_{[n-1]} \end{aligned} \tag{214}$$

If kick leg is left:

Swing foot roll:

$$Ft_{SWING\,\gamma[\phi_M=0]} = Ft_{\gamma\,L[n-1]} \tag{215}$$

Swing foot pitch:

$$Ft_{SWING\,\beta[\phi_M=0]} = Ft_{\beta\,L[n-1]} \tag{216}$$

$$Ft_{SUPPORT(x,y,z)[\phi_M=0]} = Ft_{L(x,y,z)[n-1]} \tag{217}$$

$$Ft_{SWING(x,y,z)[\phi_M=0]} = Ft_{R(x,y,z)[n-1]} \tag{218}$$

Assign pose paramter values:

Target hip back swing position (Note: adjustments to ensure static stability may be made by minor modifications to this position):

$$Hp_{[\phi_M=1]} = \begin{bmatrix} 1.8 \\ -6.5 \\ 22.5 \end{bmatrix} \tag{219}$$

This swing foot position sets the ball line up (Y), the swing potential energy source (Z) and is variable.

$$Ft_{SWING(x,y,z)}[\phi_M=1] = \begin{bmatrix} -9.5 \\ 5 \\ 10 \end{bmatrix} \tag{220}$$

Target support foot position relative to torso in designed kick (past walk step may be slightly different).

$$Ft_{SUPPORT(x,y,z)[\phi_M=1]} = \begin{bmatrix} 0.0 \\ -5.0 \\ 4.6 \end{bmatrix} \tag{221}$$

Shoulder target positions:

$$\begin{aligned} LShoulderPitch_{kick[\phi_M=1]} &= \frac{\pi}{6} \\ RShoulderPitch_{kick[\phi_M=1]} &= \frac{\pi}{6} \end{aligned} \tag{222}$$

If kick leg is right:

$$Ft_{SWING\,\gamma[\phi_M=0]} = Ft_{\gamma\,R[n-1]} \tag{223}$$

$$Ft_{SWING\,\beta[\phi_M=0]} = Ft_{\beta\,R[n-1]} \tag{224}$$

$$Ft_{SUPPORT(x,y,z)[\phi_M=0]} = Ft_{R(x,y,z)[n-1]} \tag{225}$$

$$Ft_{SWING(x,y,z)[\phi_M=0]} = Ft_{L(x,y,z)[n-1]} \tag{226}$$

$$Hp_{[N]} = \begin{bmatrix} 1.8 \\ 7.5 \\ 22.5 \end{bmatrix} \tag{227}$$

$$Ft_{SWING(x,y,z)[\phi_M=1]} = \begin{bmatrix} -9.5 \\ -5 \\ 22 \end{bmatrix} \tag{228}$$

$$Ft_{SUPPORT(x,y,z)[\phi_M=1]} = \begin{bmatrix} 0.0 \\ 5.0 \\ 4.6 \end{bmatrix} \tag{229}$$

$$LShoulderPitch_{kick[\phi_M=1]} = 2\frac{\pi}{3} \tag{230}$$

$$RShoulderPitch_{kick[\phi_M=1]} = \frac{\pi}{6} \tag{231}$$

(end right leg)

Load initial points from past walk system values:

$$Hp_{(x,y,z)[n-1]} \rightarrow \text{previous torso position} \tag{232}$$

Load knot values for cubic spline interpolations:

HipCenter (y):

$$knots_{Hp\,y} = \begin{bmatrix} Hp_{y[n-1]} \\ Hp_{y\,midPt} \\ Hp_{y[\phi_M=1]} \end{bmatrix} \tag{233}$$

HipCenter(z):

$$knots_{Hp\,z} = \begin{bmatrix} Hp_{z[n-1]} \\ \dfrac{Hp_{z[n-1]} + Hp_{z[\phi_M=1]}}{2} \\ Hp_{z[\phi_M=1]} \end{bmatrix} \tag{234}$$

Also the second derivatives at each knot (none applied at this time):

$$knots_{2\,nd\,deriv} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{235}$$

Knot phase vector:

$$knot_{phases} = \begin{bmatrix} 0 \\ 0.5 \\ 1 \end{bmatrix} \tag{236}$$

Morph Phase Operations:

Increment the morphing phase:

$$\phi_M = \phi_M + \phi_{MI} \tag{237}$$

Saturate the phase for safety:

$$if\ (\phi_M > 1.0)\ \ \{\phi_M = 1.0\} \tag{238}$$

Increment foot motion phase after walking trigger point (phase = 0.375)

$$\phi_{MF} = \begin{Bmatrix} \frac{\phi_M - Ft_{L\,Lift}}{1.0 - Ft_{L\,Lift}} & : & \phi_M & >= & Ft_{L\,Lift} \\ 0 & : & \phi_M & < & Ft_{L\,Lift} \end{Bmatrix} \tag{239}$$

Give the swing foot pitch value its own phase cycle that begins shortly after liftoff. Otherwise the toe may dig into the ground as it begins to pitch downward.

$$\phi_{MF\,\beta} = \begin{Bmatrix} \frac{\phi_M - (Ft_{L\,Lift} + 0.2)}{1.0 - (Ft_{L\,Lift} + 0.2)} & : & \phi_M & >= & Ft_{L\,Lift} \\ 0 & : & \phi_M & < & Ft_{L\,Lift} \end{Bmatrix} \tag{240}$$

Linear interpolations:

$$value[n] = initialValue_{[\phi_M = 0]} + phase[n](finalValue_{[\phi_M = 1]} - initialValue_{[\phi_M = 0]}) \tag{241}$$

Supporting foot position and attitude:

$$\begin{gathered} Ft_{SUPP\,y[n]} = linearInterp(Ft_{SUPP\,y[0]}, Ft_{SUPP\,y[N]}, \phi_{MF[n]}) \\ Ft_{SUPP\,\beta[n]} = linearInterp(Ft_{SUPP\,\beta[\phi_M = 0]}, Ft_{SUPP\,\beta[\phi_M = 1]}\phi_{MF\,\beta[n]}) \\ Ft_{SUPP(x,y,z)[n]} = linearInterp(Ft_{SUPP(x,y,z)[\phi_M = 0]}, Ft_{SUPP(x,y,z)[\phi_M = 1]}, \phi_{MF[n]}) \end{gathered}$$

Swinging foot position and attitude:

$$\begin{gathered} Ft_{SWING\,y[n]} = linearInterp(Ft_{SWING\,y[\phi_M = 0]}, Ft_{SWING\,y[\phi_M = 1]}, \phi_{MF[n]}) \\ Ft_{SWING\,\beta[n]} = linearInterp(Ft_{SWING\,\beta[\phi_M = 0]}, Ft_{SWING\,\beta[\phi_M = 1]}, \phi_{MF\,\beta[n]}) \\ Ft_{SWING(x,y,z)[n]} = linearInterp(Ft_{SWING(x,y,z)[\phi_M = 0]}, Ft_{SWING(x,y,z)[\phi_M = 1]}, \phi_{MF[n]}) \end{gathered} \tag{242}$$

Torso attitude and x dimension position:

$$\begin{gathered} Tr_{y[n]} = linearInterp(Tr_{y[\phi_M = 0]}, Tr_{y[\phi_M = 1]}, \phi_{M[n]}) \\ Tr_{\beta[n]} = linearInterp(Tr_{\beta[\phi_M = 0]}, Tr_{\beta[\phi_M = 1]}, \phi_{M[n]}) \\ Tr_{x[n]} = linearInterp(Tr_{x[\phi_M = 0]}, Tr_{x[\phi_M = 1]}, \phi_{M[n]}) \end{gathered}$$

Cubic interpolations (usage next subsection):

Torso height:

$$Hp_{z[n]} = splineInterp(3, knot_{phases}, \phi_M, knots_{Hp\,z}, knots_{2\,nd\,deriv}, 3, 3) \tag{243}$$

Torso sway:

$$Hp_{y[n]} = splineInterp(3, knot_{phases}, \phi_M, knots_{Hp\,y}, knots_{2\,nd\,deriv}, 3, 3) \tag{244}$$

If kick leg is left:

$$\begin{gathered} Ft_{(x,y,z)\,L[n]} = Ft_{SWING(x,y,z)[n]} \\ Ft_{(x,y,z)\,R[n]} = Ft_{SUPP(x,y,z)[n]} \\ Ft_{\beta\,L[n]} = Ft_{SWING\,\beta[n]} \end{gathered} \tag{245}$$

Else if kick leg is right:

$$\begin{aligned} Ft_{(x,y,z)R[n]} &= Ft_{SWING(x,y,z)[n]} \\ Ft_{(x,y,z)L[n]} &= Ft_{SUPP(x,y,z)[n]} \\ Ft_{\beta R[n]} &= Ft_{SWING\,\beta[n]} \end{aligned} \tag{246}$$

Arms:

$$\begin{aligned} LShoulderPitch_{[n]} &= linearInterp(LShoulderPitch_{kick[\phi_M=0]}, LShoulderPitch_{kick[\phi_M=1]}, \phi_{M[n]}) \\ RShoulderPitch_{[n]} &= linearInterp(RShoulderPitch_{kick[\phi_M=0]}, RShoulderPitch_{kick[\phi_M=1]}, \phi_{M[n]}) \end{aligned} \tag{247}$$

5.3. Parametrized Kick swing using inverse kinematics

The kick swing is designed to have a few key properties, these are: (i) dynamic (adjusting for ball placement relative to the robot), (ii) capable of aiming in different directions (limited range but enough to account for line up error in the behavioural targeting system) and (iii) to be as fast and powerful as possible (matching that of the RoboEireann pose based design). The components of this design are a cubic spline dynamic trajectory mapping method and pendulum natured parametrized kick swing.

5.3.1. Cubic Splines

The kick swing is algorithmically similar to the morphing to back swing pose and involves just the swinging foot. However, the trajectory has considerably more properties to it. The trajectory mapped is entirely dynamic, its starting and terminating points are variable and the foot must pass through or avoid certain points along its route. So instead of just 2 points, the beginning and the end that we are concerned with, the interpolation through the trajectory must pass through numerous points along the way. We are therefore going to have to fit some function to a series of points that is smooth. Now, fitting a polynomial of the appropriate order to these points may come to mind but that would be subject to the Runge phenomenon [216] where wild oscillations can occur. As we are attempting to kick within some range of angles there is a very high likelihood this will occur.

Fortunately, there is a solution to this problem, we can define the trajectory using cubic splines. A cubic spline is a piecewise cubic polynomial fitted to the data points and is only valid for the section between three points. It is therefore blind to any higher frequency components in the overall data set that a single polynomial could identify. A cubic spline is defined by the data points, known as 'knots' and the second order derivatives at each knot. These second order derivatives control the amount of curvature that exists when passing through each knot and thereby limiting the wild oscillations that could otherwise occur. The second order derivatives that join the signal trajectory between points are then dynamically computed by the algorithm. Cubic splines may also come in various flavours, the B-spline, the Hermite spline and the 'successive over relaxation'. This B-spline gives an interpolation that will be an approximation of the knots but not necessarily passing through them resulting in less curvature. The Hermite spline passes through the knots but takes sharp turns at them providing a more piecewise linear fit and limited oscillations. The successive over relaxation is more a standard cubic fit piecewise. Figure 106 is an example of such splines in use.

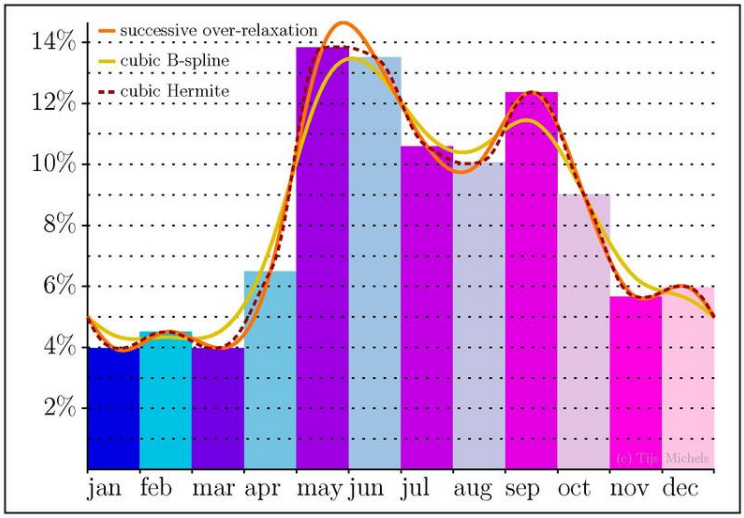


Figure 106: [217] Various Cubic Spline Flavors

Two cubic spline algorithms are used in this project, for simulation and development the spline.m that is packaged with MATLAB and a C++ package written and shared by John Burkardt [218] that is used in on the robot in the systems final version.

Each dimension, X, Y and Z, is mapped independently as a function of swing phase. The knots in each dimension are a product of the parametrization of the kick swing design. Figure 107 is a simulation of this kick swing trajectory mapping:

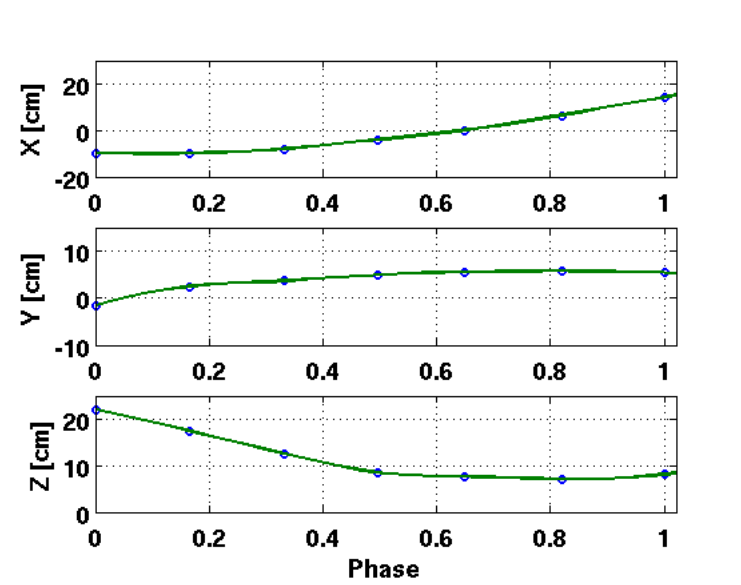


Figure 107: Simulated Kick Swing Trajectory Mapping

5.3.2. Parametrized Dynamic Kick Swing

There is a common misconception among the RoboCup SPL (Nao) League members that there is no need to take a big wind-up in the kick to maximize shot power as setting the foot travel velocity from point A to B (cocked position and contact) will take care of everything and no more than that will help. Consequently, you will see most kicks are a linear foot trajectory from point A to B. There are some things wrong with that belief. First of all, we can gain more momentum by finding a way to make the foot accelerate between point A and B. It is true there is no point in defining a linear trajectory that accelerates, that simply setting the target maximum velocity is enough and the robot's lower level motor control system will accelerate on its own to achieve that velocity. What is overlooked is that this is a real machine, it has mass, a limited availability of potential energy in the motors and a variety of opposing forces such as friction and gravity. Acknowledging these environmental properties can help make a difference. For example, defining a trajectory that is inherently accelerating (centripetal trajectory) and by using gravity as a motive force.

Another overlooked property is the behaviour of Aldebaran's motor command system. There is no limited velocity range for the motor command input API. The motors will attempt to realize any given command, as fast as it is capable of until saturation is reached. If a motor is told to move from one of its angular limits to the other in 10ms it will not calculate the motion profile and return an error and refuse to comply if this motion is not physically realizable, it will simply try and do it, overcoming whatever resistive force it encounters. Therefore the motor velocity is limited by the load the motors experience alone. So, we should be able to gain more shot power by relieving some of the motor load and gravity can assist here.

Finally, putting effort into the full body posture of the kicking gait can make a difference on contact. If the robot pivots on the supporting foot upon contact, power is being lost due to the law of conservation of energy. The robot is heavy (relative to the scale of this foot to ball force transfer problem) and if it rotates, kick swing energy is being used up to spin the robot. Upon the kick snap a moment arm is created. Therefore bracing against a counter-revolutionary force will also help [219]. The arms play a role here.

So, to perform beyond the velocity limits of setting the foot down a linear trajectory from point 'A' to 'B', the kick swing can be modelled as a pendulum, using gravity as an additional source of potential energy and by generating centripetal forces that translate into linear acceleration as it tangentially disembarks from the pendulum trajectory, finally terminating in a counter revolusionary pose.

5.3.2.1. Kick Swing Design Design Method

The kick swing as mentioned is a smooth 3D trajectory that is dynamically mapped using cubic splines. There are various concerns or properties that go into building a kick swing such as power, speed, ground clearance, angled ball contact, the legs workspace limits etc. These properties will then give us some points of interest in each dimension that we will use as the knots in the cubic spline interpolation. Each separate dimension will have points of interest, where certain behaviour must be realized, that do not have any impact on the other dimensions, but our knots will be defined three dimensionally. It is not strictly necessary that each dimension be defined with the same number of knots computationally, but it does make things tidier. We will define a set of points with their significant elements governed by the kick swing parameters and then use those to fill in the missing elements in the other dimensions. What ties all the dimensions together is that each dimension is the function of phase, a phase that represents the progress of the swing (rotation angle) as it moves through an abstract orbital path or ideal pendulum-like motion we use to represent the swing. We will design the kick in two planes, the Z-X plane first and then the X-Y plane. The Z-X plane controls the kick power and leg length issues such as unfolding and ground clearance. On the Z-X plane we will use an abstract orbital path to represent the kick swing and on this orbital path the leg will have the properties of a pendulum responding to gravity. The basic premise here is that we will raise the foot up very high and drop it, and then redirect the acquired kinetic energy to travel down the ball target shot vector by relying on the centripetal inertial properties of orbiting bodies.

5.3.2.1.1. The Z-X Plane (Kick Power)

To start we will introduce this orbital path as a circle centred around the hip point as shown in Figure 108. There will be numerous points of interest along this circle, the first being the clearance point that ensures there will be no foot – ground contact. The length of the leg at this point (ℓ_{CLR}) will be the radius of this circle. This parameter is defined as a clearance height and the leg length is then the difference between the clearance height and the hip height. It will be located straight below the hip position and will have the same X value as the hip.

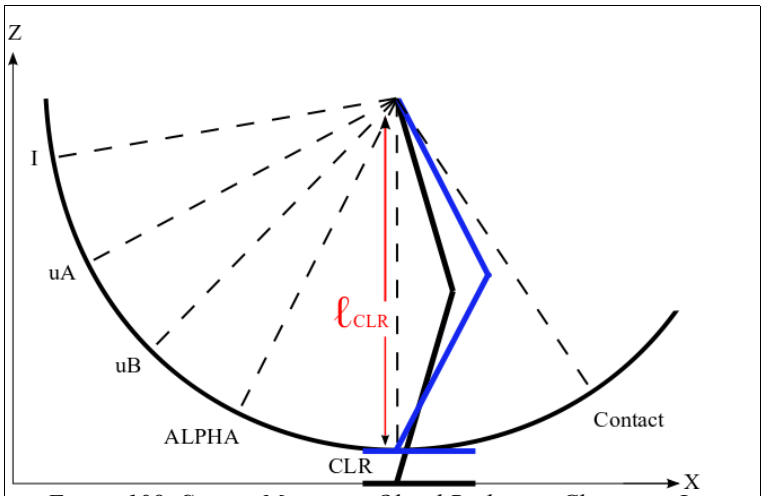


Figure 108: Setting Maximum Obital Radius as Clearance Leg Length

The remaining set of points can be found using straightforward trigonometry. We will need either two lengths or a length and an angle. The next point of interest is a point we will call ALPHA, seen in Figure 109. This is the point we will say that the foot pitch must be level by, as it is approaching the ground and will begin to risk toe contact. From this point forward until release the foot will travel in a constant orbit. This point is set by introducing the parameter 'X at ALPHA' and is a distance behind the clearance point. This distance and the leg length allows us to build our first triangle and provides the angle at ALPHA (θ_{ALPHA}).

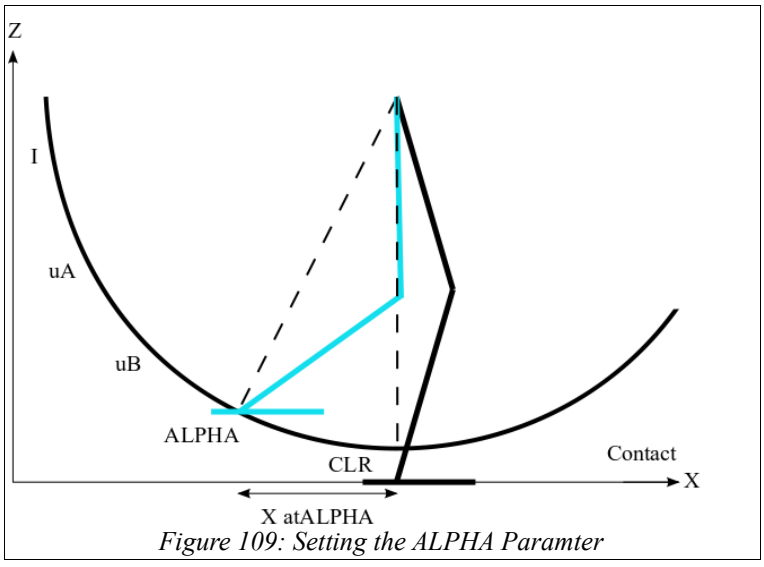


Figure 109: Setting the ALPHA Paramter

The next set of points I, uA, and uB govern the back swing. It is during this phase that the leg will unfold. We will find these points by using the leg length and the angle at each position. We already have θ_{ALPHA} and the initial angle(θ_I) and contact angle($\theta_{Contact}$). We can derive these from their X-Y coordinates that are the dynamic inputs, the cocked foot location and ball contact position that is calculated from the ball position (more on that in the X-Y plane discussion). Our angles here are relative to the initial angle as the phase range is from θ_I to $\theta_{Contact}$. We use the θ_I as an offset and subtract it from the θ_{ALPHA}, θ_{CLR} and $\theta_{Contact}$. The remaining angles θ_{uA} and θ_{uB} are a trisection between θ_I and the θ_{ALPHA} shown in Figure 110. This will tie the linear increase in the leg length to the constant rotation around the orbital path.

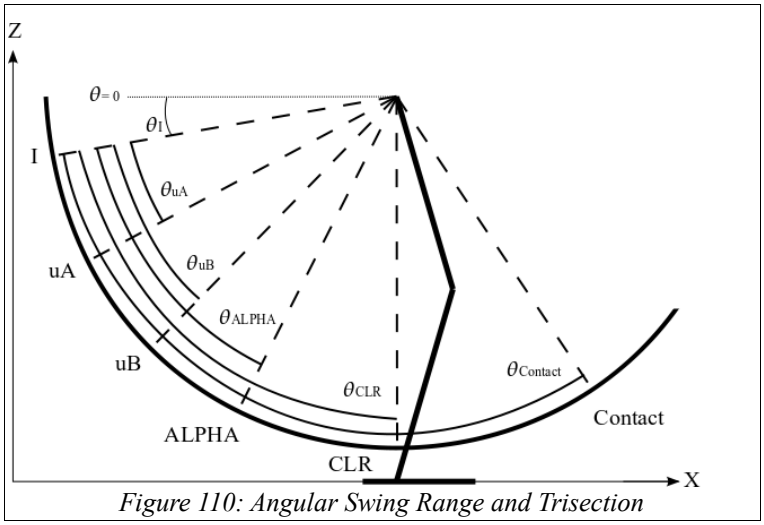


Figure 110: Angular Swing Range and Trisection

All we need then to build our last two triangles is the leg length as it unfolds to point ALPHA. The leg will increase in length, interpolating linearly(Figure 111). We will need the phase at ALPHA which is found from the angle we already have at ALPHA and its ratio inside the full swing angle range. This gives as an interpolation phase range, 'phase at ALPHA'.

We will also find an additional source of power here by using the leg tension. Constant rotational velocity around our abstract circle while simultaneously increasing the leg length (orbital radius) will build up momentum as we know that bodies with equal mass travelling at higher orbits with equal rotation velocity will have more momentum. So as well as absorbing the potential energy from the height drop, we will be amplifying it by increasing the moment arms length. Although this is not a point mass suspended as a bob, the center of mass of the leg will travel outward as the leg unfolds.

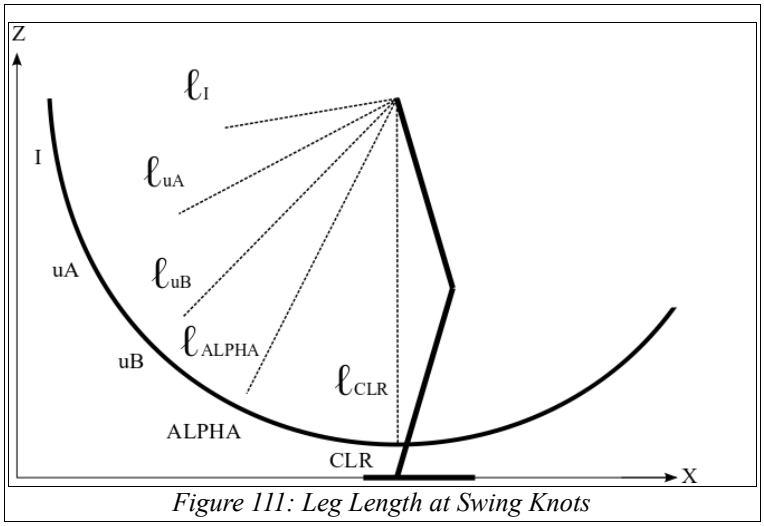


Figure 111: Leg Length at Swing Knots

With all these points defined, we will generate the following trajectory in Figure 112. These points will be used as a partial set of elements for our cubic spline knots.

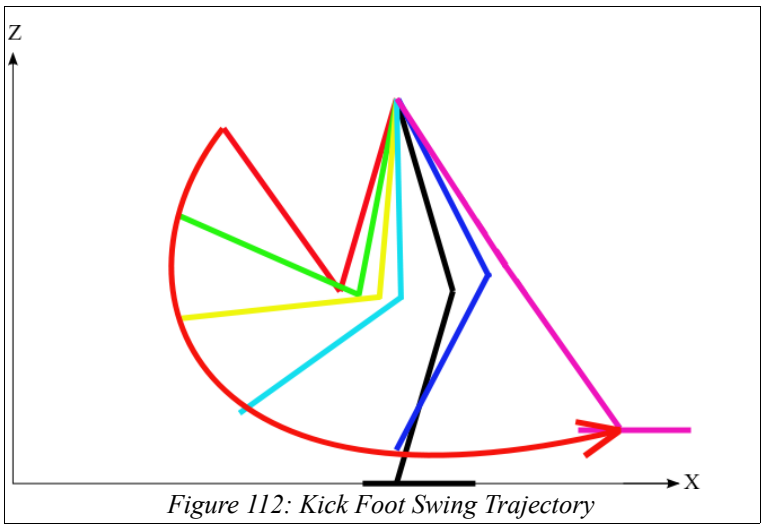


Figure 112: Kick Foot Swing Trajectory

To summarize, the basic principle is shown in Figure 113. The swing trajectory converts the potential energy from the drop in height to kinetic energy. This creates a moment arm orbiting a circle that both preserves the acquired energy and makes use of the leg tension to redirect the momentum vector around to become inline with the shot vector. The foot then accelerates as it is released on a tangent to the orbital path ultimately transferring as much power as possible into the ball as the knee snaps in to a fixed position.

What we are also doing here is making use of both the knee and hip pitch motors as the leg unfolds. Both actuators acting on the foot will allow it to reach a higher velocity. A continually expanding leg will maintain the knee motor velocity.

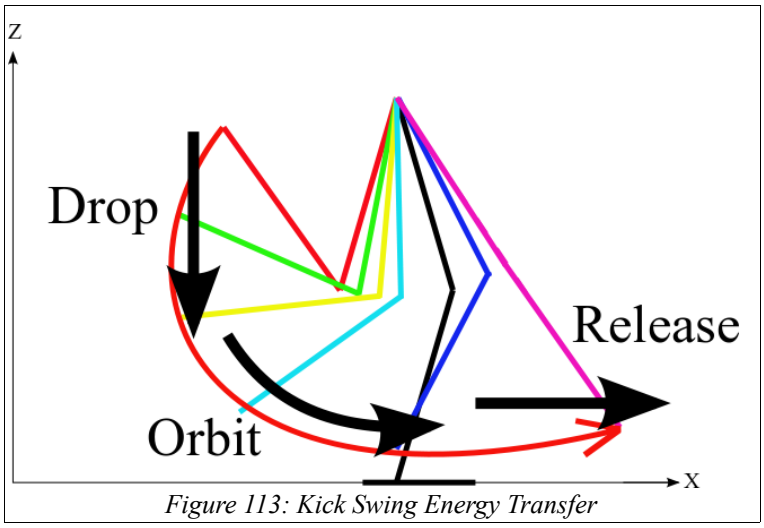


Figure 113: Kick Swing Energy Transfer

Putting this all together we have our parametrized power kick swing in the Z-X plane(Figure 114).

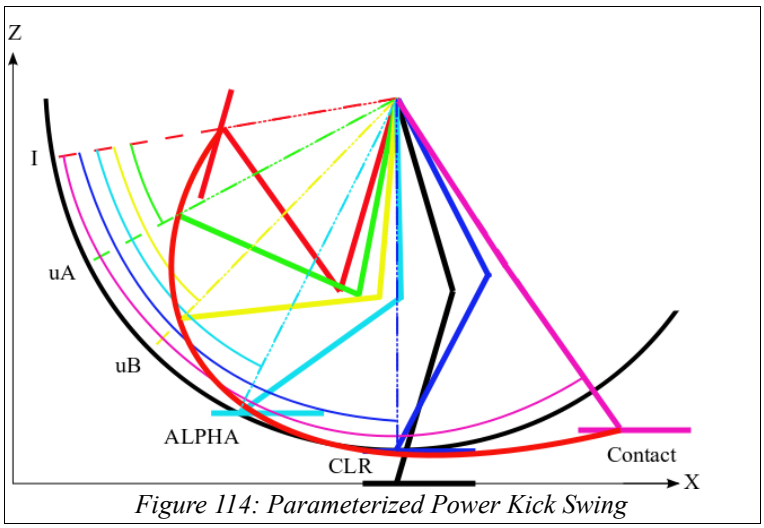


Figure 114: Parameterized Power Kick Swing

5.3.2.1.2. The X-Y Plane (Kick Swing Targeting)

The X-Y plane is where we shape the swing trajectory to contact the ball and to target the direction of the kick. This is achieved by creating additional knots that represent significant points such as the ball center, the foot toe center, the ball contact position, and a supporting foot clearance position. We then map a trajectory from the the initial position that will hit all the points smoothly.

Graphical depiction of the various factors involved is complex, therefore we will build it up one piece at a time. We will begin with the representation of the foot or toe. It turns out that a circle fits quite well to the front edge of the robot's foot and we can there represent it as such. The center of this circle has an offset vector from the foot origin directly below the ankle axis. We will call this point the 'toe'. Figure 115 demonstrates the toe and the representation of the foot as a circle around the toe. The measured toe vector is [5.75 0.2 0] for the left foot and [5.75 -0.2 0] for the right foot.

You will notice here too that the foot shape is not the Nao foot shape but the convex hull of the foot, which is the same as the support polygon around it. This was constructed by hand in Photoshop followed by extraction of the relevant data points from the resultant image and saving them in a look up table.

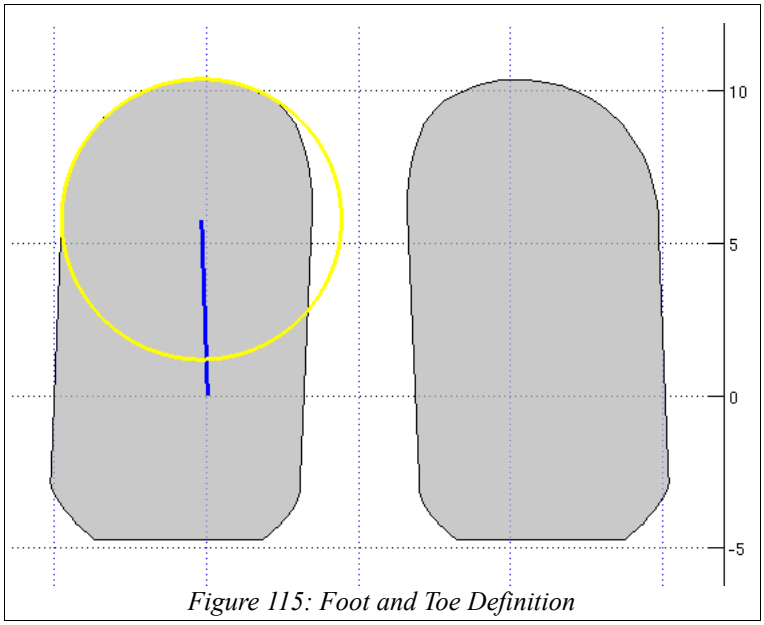


Figure 115: Foot and Toe Definition

Next we define the ball – toe contact position. Here we can use the basic spherical contact solution: add the radius of the foot circle and the ball together to form a contact vector [(R_{Ball} + R_{Foot}) 0 0] in the glabal coordinate system. This vector is then rotated around the ball location to be inline with the shot vector and then rotated again by 180 degrees to set the contact position as shown in Figure 116.

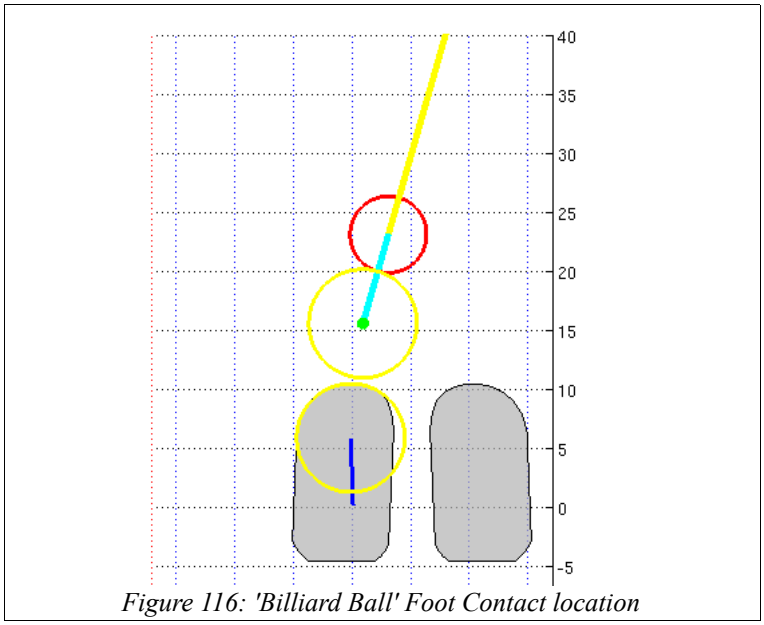


Figure 116: 'Billiard Ball' Foot Contact location

We must now define a target zone that the ball has to be in to perform a kick and limit it. This will be the workspace of the foot and is limited by the kinematic resolution range and clearance around the supporting foot. This is done by drawing a box that represents the kinematic range and then removing a portion of this range to ensure that the kicking foot clears the supporting foot. The corner points then give us lines we can use as decision boundaries as shown in Figure 117. This decision boundary is not implemented in real time in the robot as yet. It is however already part of the simulation graphic. The current behavioural system for lining up a kick uses a simple targeting box that is about the size of the ball itself. This box is large enough for the kick line up behaviour system to get the ball into quickly without any final shuffling. In the current system, the targetting box lies directly ahead of the foot initial position and therefore there is no issue with foot collisions. However, the concept can be easily extended to larger targetting regions, provided that careful attention is paid to avoiding foot-foot collisions.

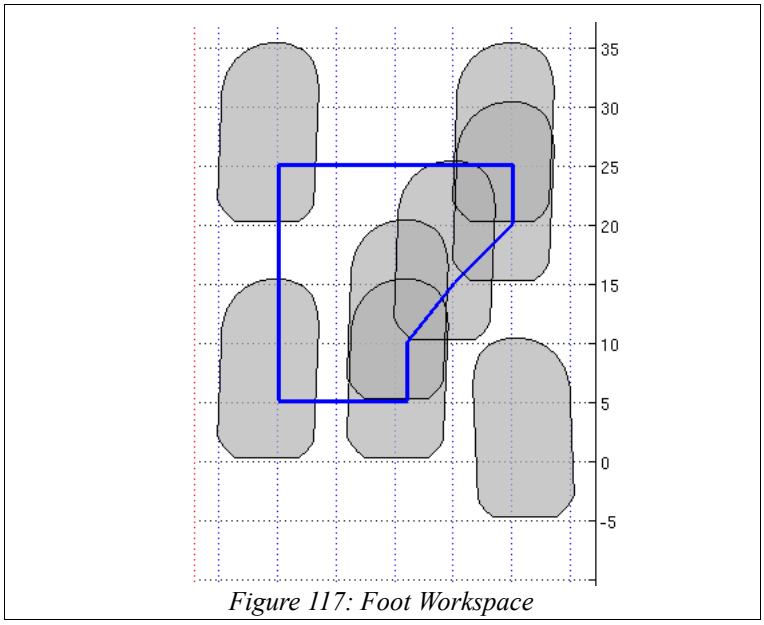


Figure 117: Foot Workspace

To get good force transfer between the foot and the ball, we will want to place the center of the foot (the end of the toe vector) in the same location as the ball center. So we can then add the toe vector to the kinematic range box and offset it to yield the final ball target region (Figure 118).

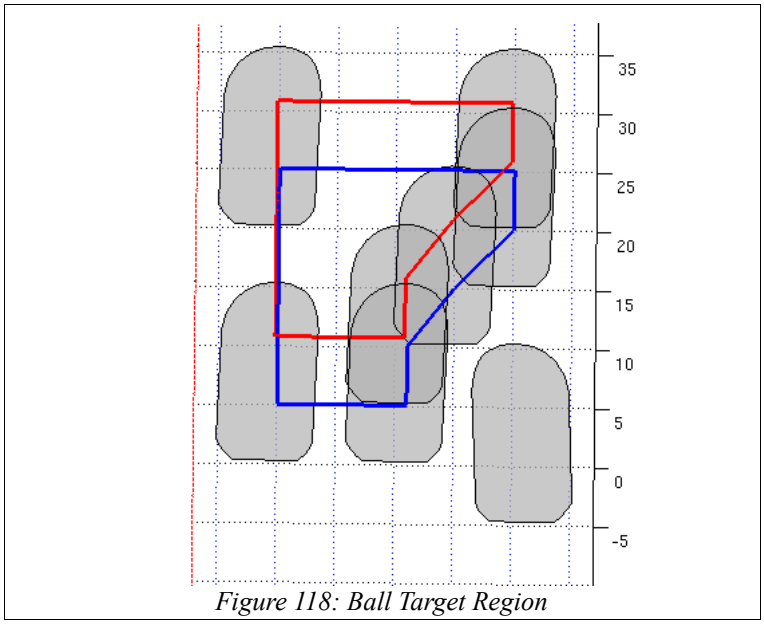


Figure 118: Ball Target Region

So now we can put both these items together. When the ball is within the target zone we can use the ball position and shot angle to dynamically generate the ball contact position (Figure 119). This value, along with the ball position gives the Y coordinates for the cubic splines.

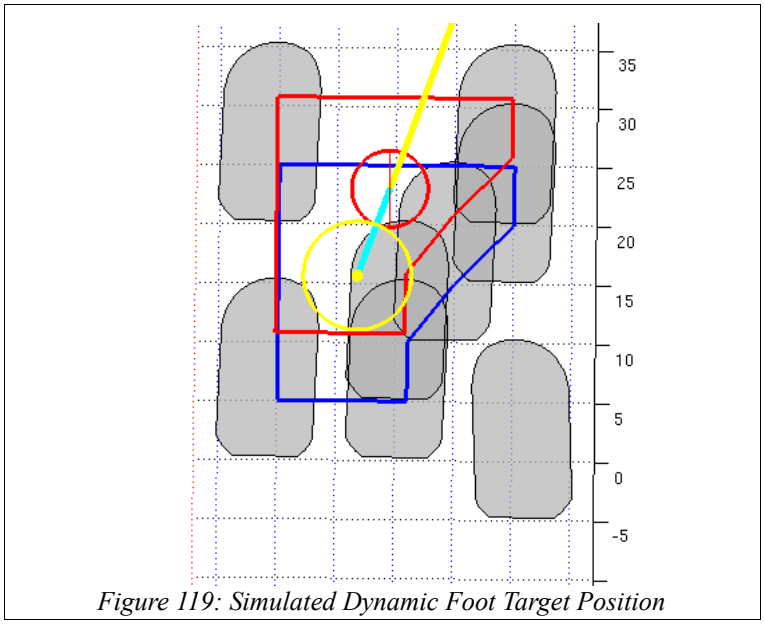


Figure 119: Simulated Dynamic Foot Target Position

The final remaining point of interest is the clearance position. That is set to be the same distance outward as the ball contact position. The points uA, uB and ALPHA are then a fraction of this location, mapping out a linear trajectory to the clearance point. These additional points help reduce oscillations without having to get into the second derivatives and still provide a route that is similar to the desired curve of an angled kick.

5.3.2.1.3. The kick Swing Trajectory

With each knot parametrized, we can then plot the kick swing trajectory as a function of kick swing phase. However there are a couple of further issues to address. First of all, the trajectory we have calculated is the foot flight path. The inverse kinematics resolves to the ankle position. So we take this trajectory and offset it by both the toe vector length and the foot height to yield the final trajectory shown in Figure 120.

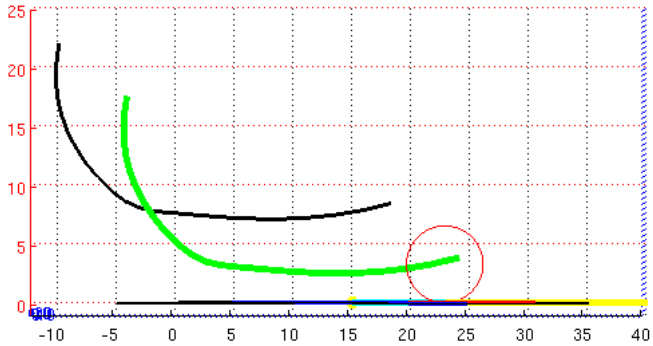


Figure 120: Simulated Foot and Ankle Swing Trajectory

The second issue is the behaviour of this kick design at sharp angles. Figure 121 is an example of an angle of -80 degrees given the random ball position set for the previous demonstration. The trajectory mapped runs outside of the foot workspace.

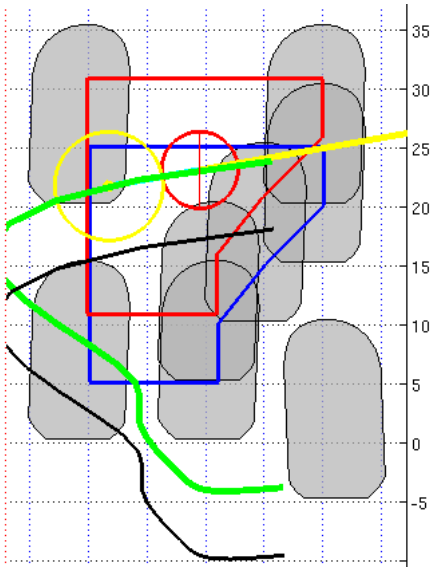


Figure 121: Excessive Cubic Spline Oscillation Example 1

Moving the ball to the far corner does not really help much as seen in Figure 122:

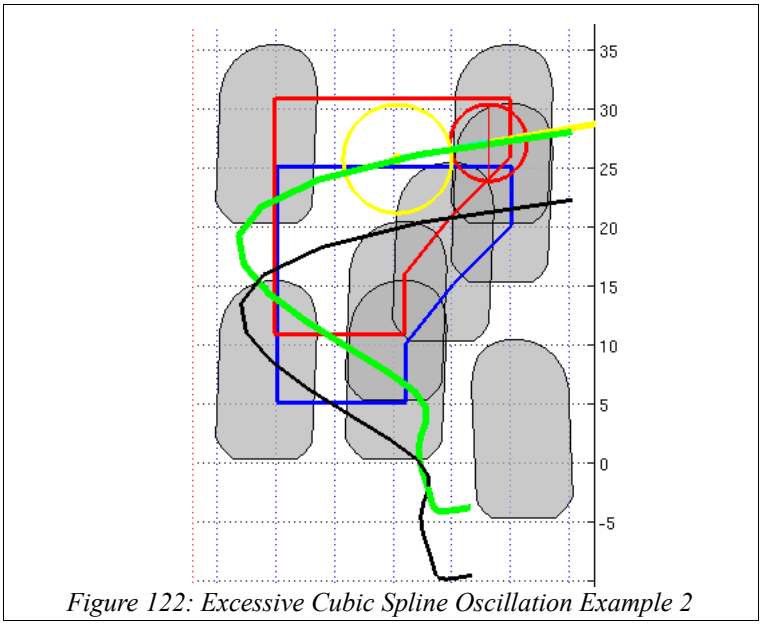


Figure 122: Excessive Cubic Spline Oscillation Example 2

It is also worth pointing out here, (Figure 123), the affect of the clamping at the back swing knots by removing them from the Y signal mapping:

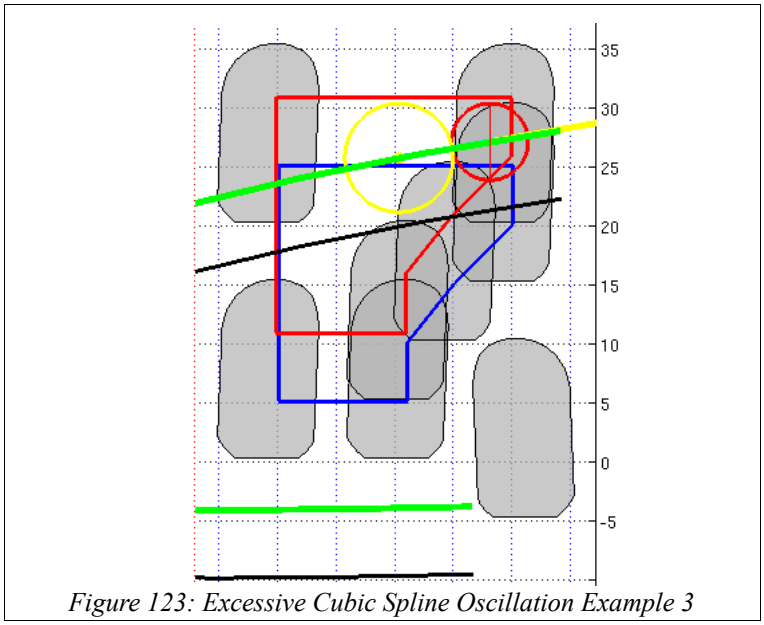


Figure 123: Excessive Cubic Spline Oscillation Example 3

This is also with the 2 second derivatives set to be zero. So the clamping is required but will only be effective up to a limit. Certain ball positions (Figure 124) will have a more oscillatory effect than others and the clamping will reduce this. However, some very sharp turns over a short period would simply be to fast too be realizable on the actual hardware. In this example, the angle -80 is possible kinematically but the robot really would not be able to makes such quick turns.

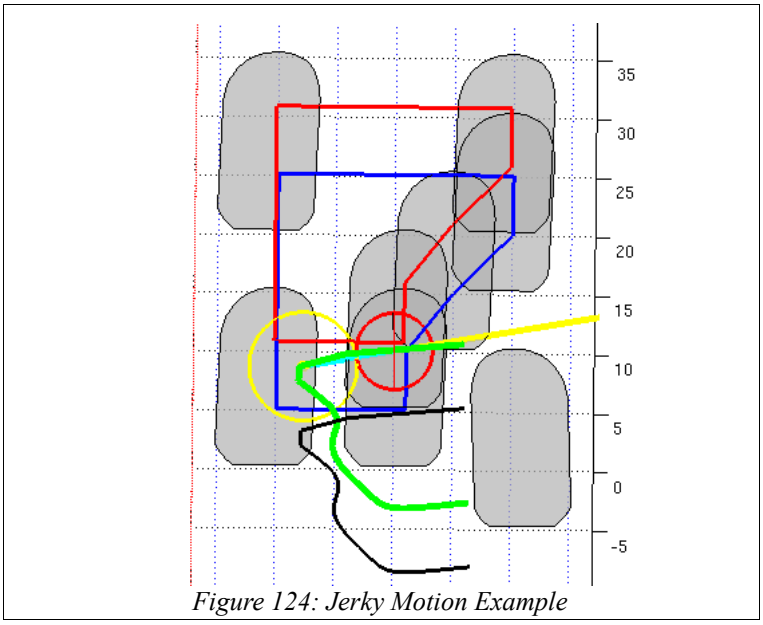


Figure 124: Jerky Motion Example

Further work could be done here to develop some more complex clamping strategy. Perhaps by checking for the oscillation peak location and iteratively generating a clamp positions that would keep it in the workspace.

Moving forward, it would seem that the best approach for now is to limit the ball angle. Some more examples can put this into perspective a little more. Figure 125 demonstrates that -60 degrees works at our original ball position (this ball position is near in line with the foot and is a very realistic one the behaviour line up system would aim for, but we would like to work with cases when the initial condition is post ball line up).

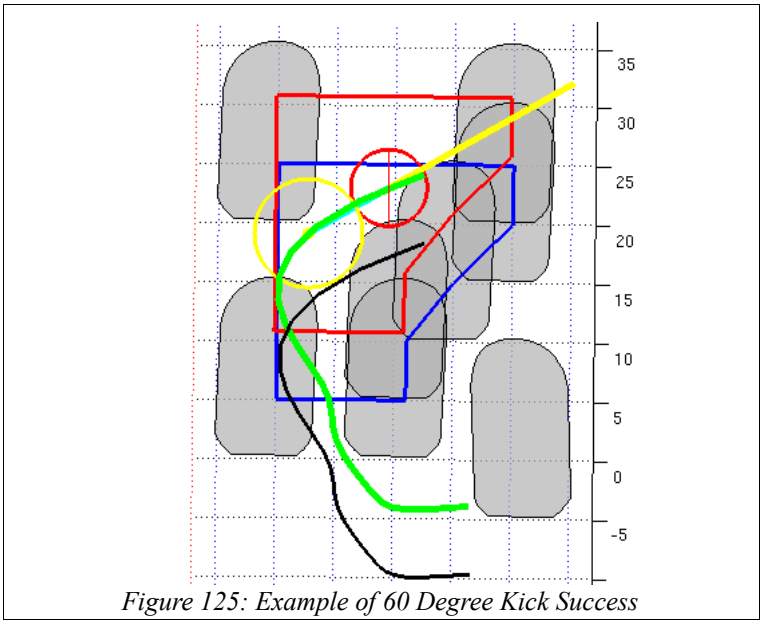


Figure 125: Example of 60 Degree Kick Success

However, we can make that angle fail again just by moving the ball (Figure 126).

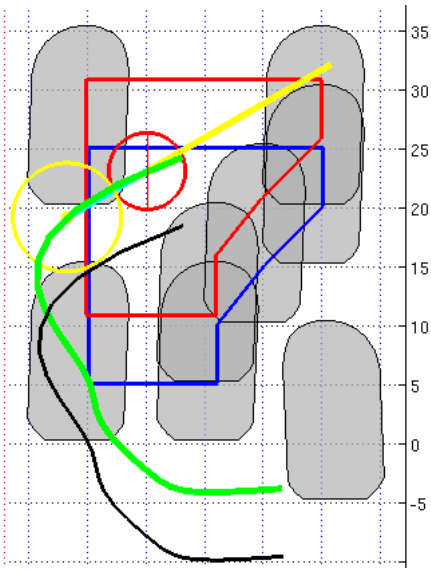


Figure 126: Example of 60 Degree Kick Failure

It seems best to limit the ball position and shot angle to something like -35 degrees, lateral limit of y = 10 and keeping the ball back so it is no closer than x = 20. These are the values in the current developed system and without the upper right corner non rectangular section that requires the decision boundary to clear the supporting foot. These limiting values look to be quite near to the operating limit as shown in simulation (Figure 127).

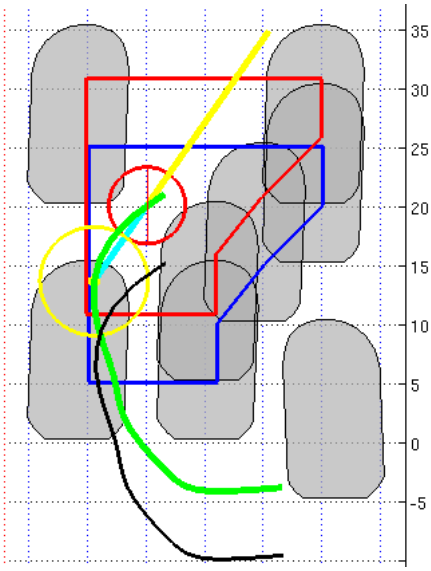


Figure 127: Chosen Target Region Limit

Finally, it is noticeable from this last figure that the trajectory at the end does not quite follow the billiard ball contact vector. This was tidied up in the hardware version by creating a final clamping knot that is just a short distance past the ball location down the shot vector.

5.3.3. Kick Swing Trajectory Calculation

The kick swing trajectory is a two part process. First, the cubic spline knots must be calculated given the kick parameters and ball position, then the foot position interpolates its way through the plotted cubic spline trajectory.

5.3.3.1. Knot Value Calculation

The knot values are mapped out in the first sample of the kick swing cycle. These operations are as follows.

Set swing speed (phase increment):

$$\phi_{SI} = 0.0255 \tag{248}$$

Convert input data from vision system from meters to cm for this system.

Define some used constants:

Toes (ankle center to 'foot ball' center):

$$\begin{aligned} ToeOffset_L &= \begin{bmatrix} 5.75 \\ 0.2 \\ 0 \end{bmatrix} \\ ToeOffset_R &= \begin{bmatrix} 5.75 \\ -0.2 \\ 0 \end{bmatrix} \end{aligned} \tag{249}$$

Foot height offset vector:

$$ankleOffset = \begin{bmatrix} 0 \\ 0 \\ 4.6 \end{bmatrix} \tag{250}$$

Radii for both foot and ball:

$$\begin{aligned} R_{FOOT} &= 4.6 \\ R_{BALL} &= 3.25 \end{aligned} \tag{251}$$

Height parameters (trying to get some top spin on the ball here by bracketing the midpoint)

$$\begin{aligned} targetHeight &= 3.5 \\ contactHeight &= 2.5 \\ clearanceHeight &= 3 \end{aligned} \tag{252}$$

Follow through foot pitch (more top spin):

$$Ft_{SWING\,\beta[\phi_s=1]} = -5\frac{\pi}{180} \tag{253}$$

Set target position knot:

$$Target = \begin{bmatrix} Ball_x \\ Ball_y \\ targetHeight \end{bmatrix} \tag{254}$$

CLR knot z:

$$CLR_z = clearanceHeight \tag{255}$$

Ball contact vector:

$$ballToeContact = R_{\phi z}(\theta_{Shot\ Angle} - \pi) \begin{bmatrix} (R_{FOOT} + R_{BALL}) \\ 0 \\ 0 \end{bmatrix} \tag{256}$$

Contact knot:

$$Contact = Target + ballToeContact \tag{257}$$

If kick leg = left

Load foot pitch from memory:

$$Ft_{SWING\,\beta[\phi_s=0]} = Ft_{\beta\,L[n-1]} \tag{258}$$

Load initial foot position from memory:

$$Ft_{SWING[\phi_s=0]} = Ft_{L[n-1]} \tag{259}$$

Assign correct hip side from memory:

$$Hp_{SWING} = Hp_{L[n-1]} \tag{260}$$

Shift back target point to ankle with correct toe:

$$target = target - ToeOffset_L \tag{261}$$

Again with contact location:

$$contact = contact - ToeOffset_L \tag{262}$$

Else if kick leg = right (same as above)

$$Ft_{SWING\,\beta[\phi_s=0]} = Ft_{\beta\,R[n-1]} \tag{263}$$

$$Ft_{SWING[\phi_s=0]} = Ft_{R[n-1]} \tag{264}$$

$$Hp_{SWING} = Hp_{R[n-1]} \tag{265}$$

$$target = target - ToeOffset_R \tag{266}$$

$$contact = contact - ToeOffset_R \tag{267}$$

(end right leg)

Set ALPHA position parameter:

$$XatALPHA = Hp_{SWING\,x} - 4 \tag{268}$$

Set a clamping knot just ahead of target point to straighten out trajectory along contact vector:

$$Follow = target - (0.0001\ ballToeContactVector) \tag{269}$$

Set clearance location directly under the kick hip:

$$CLR_x = Hp_{SWING\,x} \tag{270}$$

Set leg length to clearance point:

$$\ell_{CLR} = Hp_{SWING\,z} - CLR_z \tag{271}$$

Use Pythagoras to find leg lengths to knots we have enough information for:

$$\begin{aligned}
\ell_{init} &= \sqrt{(((Hp_{SWING\,x} - Ft_{x[\phi_s=0]})^2) + ((Hp_{SWING\,z} - Ft_{z[\phi_s=0]})^2))} \\
\ell_{Contact} &= \sqrt{(((Hp_{SWING\,x} - contact_x)^2) + ((Hp_{SWING\,z} - contact_z)^2))} \\
\ell_{Target} &= \sqrt{(((Hp_{SWING\,x} - target_x)^2) + ((Hp_{SWING\,z} - target_z)^2))} \\
\ell_{Follow} &= \sqrt{(((Hp_{SWING\,x} - Follow_x)^2) + ((Hp_{SWING\,z} - Follow_z)^2))}
\end{aligned} \tag{272}$$

Use trigonometry to find swing angles to these same positions:

$$\begin{aligned}
\theta_{init} &= acos\left(\frac{Hp_{SWING\ x} - Ft_{SWING\ x[\phi_s=0]}}{\ell_{init}}\right) \\
\theta_{CLR} &= \frac{\pi}{2} \\
\theta_{ALPHA} &= acos\left(\frac{Hp_{SWING\ x} - XatALPHA}{\ell_{CLR}}\right) \\
\theta_{Contact} &= \pi - acos\left(\frac{contact_x - Hp_{SWING\ x}}{\ell_{Contact}}\right) \\
\theta_{Target} &= \pi - acos\left(\frac{target_x - Hp_{SWING\ x}}{\ell_{Target}}\right) \\
\theta_{Follow} &= \pi - acos\left(\frac{Follow_x - Hp_{SWING\ x}}{\ell_{Follow}}\right)
\end{aligned} \tag{273}$$

Set full swing range:

$$\theta_{Range} = \theta_{Follow} - \theta_{init} \tag{274}$$

Use angles to find phases to position (this here sets the constant rotation velocity)

$$\begin{aligned}
\phi_{ALPHA} &= \frac{(\theta_{ALPHA} - \theta_{init})}{\theta_{Range}} \\
\phi_{CLR} &= \frac{(\theta_{CLR} - \theta_{init})}{\theta_{Range}} \\
\phi_{Contact} &= \frac{(\theta_{Contact} - \theta_{init})}{\theta_{Range}} \\
\phi_{Target} &= \frac{(\theta_{Target} - \theta_{init})}{\theta_{Range}} \\
\phi_{Follow} &= \frac{(\theta_{Follow} - \theta_{init})}{\theta_{Range}}
\end{aligned} \tag{275}$$

Trisect for the remaining locations:

$$\begin{aligned}
\phi_{uA} &= \frac{\phi_{ALPHA}}{3} \\
\phi_{uB} &= 2\frac{\phi_{ALPHA}}{3}
\end{aligned}$$

Interpolate missing leg lengths:

$$\begin{aligned}
\ell_{uA} &= \ell_{init} + \frac{\phi_{uA}}{\phi_{CLR}}(\ell_{CLR} - \ell_{init}) \\
\ell_{uB} &= \ell_{init} + \frac{\phi_{uB}}{\phi_{CLR}}(\ell_{CLR} - \ell_{init}) \\
\ell_{ALPHA} &= \ell_{init} + \frac{\phi_{ALPHA}}{\phi_{CLR}}(\ell_{CLR} - \ell_{init})
\end{aligned} \tag{276}$$

Interpolate missing angles:

$$\begin{aligned}
\theta_{uA} &= \theta_{init} + (\phi_{uA}\theta_{Range}) \\
\theta_{uB} &= \theta_{init} + (\phi_{uB}\theta_{Range})
\end{aligned} \tag{277}$$

Calculate missing knot x values:

$$\begin{aligned}
uA_x &= -\ell_{uA}\cos(\theta_{uA}) \\
uB_x &= -\ell_{uB}\cos(\theta_{uB})
\end{aligned} \tag{278}$$

Calculate missing knot z values:

$$\begin{aligned} uA_z &= Hp_{SWING\ z} - \ell_{uA}\sin(\theta_{uA}) \\ uB_z &= Hp_{SWING\ z} - \ell_{uB}\sin(\theta_{uB}) \\ ALPHA_z &= Hp_{SWING\ z} - \ell_{CLR}\sin(\theta_{ALPHA}) \end{aligned} \tag{279}$$

Offset foot trajectory to ankle position for inverse kinematics:

$$\begin{aligned} ALPHA &= ALPHA + ankleOffset \\ CLR &= CLR + ankleOffset \\ Contact &= Contact + ankleOffset \\ Target &= Target + ankleOffset \\ Follow &= Follow + ankleOffset \end{aligned} \tag{280}$$

Positions uA and uB will however have a pitched foot and therefore the ankle offset needs rotation:

Interpolate the ankles pitch at these positions:

$$\begin{aligned} \theta_{Ft_\beta uA} &= linearInterp(\phi_{uA}, Ft_{SWING\beta[\phi_s=0]}, 0) \\ \theta_{Ft_\beta uB} &= linearInterp(\phi_{uB}, Ft_{SWING\beta[\phi_s=0]}, 0) \end{aligned} \tag{281}$$

Rotate ankle vector and add to uA and uB:

$$\begin{aligned} uA &= uA + Rz(\theta_{Ft_\beta uA})\, ankleOffset \\ uB &= uB + Rz(\theta_{Ft_\beta uB})\, ankleOffset \end{aligned} \tag{282}$$

Now for the X-Y plane set some clamping knots:

$$\begin{aligned} uA_y &= \frac{1.3}{3}(Ft_{SWING\ y[\phi_s=0]} + (contact_y - Ft_{SWING\ y[\phi_s=0]})) \\ uB_y &= \frac{2}{3}(Ft_{SWING\ y[\phi_s=0]} + (contact_y - Ft_{SWING\ y[\phi_s=0]})) \\ ALPHA_y &= \frac{2.25}{3}(Ft_{SWING\ y[\phi_s=0]} + (contact_y - Ft_{SWING\ y[\phi_s=0]})) \\ CLR_y &= Ft_{SWING\ y[\phi_s=0]} + (contact_y - Ft_{SWING\ y[\phi_s=0]}) \end{aligned} \tag{283}$$

Finally load vectors for use in cubic spline interpolation function:

$$swingKnotPhases = \begin{bmatrix} 0 \\ \phi_{uA} \\ \phi_{uB} \\ \phi_{ALPHA} \\ \phi_{Clearance} \\ \phi_{Contact} \\ \phi_{Target} \\ \phi_{Follow} \end{bmatrix} \tag{284}$$

$$swingKnotValues_X = \begin{bmatrix} Ft_{SWING\ x[\phi_S=0]} \\ uA_x \\ uB_x \\ ALPHA_x \\ Clearance_x \\ Contact_x \\ Target_x \\ Follow_x \end{bmatrix}$$

$$swingKnotValues_Y = \begin{bmatrix} Ft_{SWING\ y[\phi_S=0]} \\ uA_y \\ uB_y \\ ALPHA_y \\ Clearance_y \\ Contact_y \\ Target_y \\ Follow_y \end{bmatrix} \quad (285)$$

$$swingKnotValues_Z = \begin{bmatrix} Ft_{SWING\ z[\phi_S=0]} \\ uA_z \\ uB_z \\ ALPHA_z \\ Clearance_z \\ Contact_z \\ Target_z \\ Follow_z \end{bmatrix}$$

Use spline_cubic_set function to generate 2nd derivative. We are setting all 2nd derivatives as zero here. In this application we are not trying to fit some process-like data and capture the characteristics, we just want to hit all the points with minimal curvature between them. Exploring these 2nd derivative values further could improve performance and reduce curvature.

$$\begin{aligned} swingKnot2ndDerivs_X &= \text{spline\_cubic\_set}(numSwingKnots_X, swingKnotPhases, swingKnotValues_X, 0,0,0,0) \\ swingKnot2ndDerivs_Y &= \text{spline\_cubic\_set}(numSwingKnots_Y, swingKnotPhases, swingKnotValues_Y, 0,0,0,0) \\ swingKnot2ndDerivs_Z &= \text{spline\_cubic\_set}(numSwingKnots_Z, swingKnotPhases, swingKnotValues_Z, 0,0,0,0) \end{aligned} \quad (286)$$

5.3.3.2. Swing Phase Operations

Increment swing phase:

$$\phi_S = \phi_S + \phi_{SI} \quad (287)$$

Increment foot along cubic trajectory (see Appendix M for usage):

$$\begin{aligned} Ft_x &= \text{spline\_cubic\_val}(8, swingKnotPhases, \phi_S, swingKnotValues_X, swingKnot2ndDerivs_X, \ldots) \\ Ft_y &= \text{spline\_cubic\_val}(8, swingKnotPhases, \phi_S, swingKnotValues_Y, swingKnot2ndDerivs_Y, \ldots) \\ Ft_z &= \text{spline\_cubic\_val}(8, swingKnotPhases, \phi_S, swingKnotValues_Z, swingKnot2ndDerivs_Z, \ldots) \end{aligned} \quad (288)$$

Set foot pitch, level by phase ALPHA and pop up at end:

$$\begin{aligned} &if\,(\phi_S < \phi_{ALPHA})\left\{\theta_{Ft_\beta uB} = linearInterp\left(\frac{\phi_S}{\phi_{ALPHA}}, Ft_{SWING\beta[\phi_S=0]}, 0\right)\right\} \\ &else\,if\,(\phi_S > \phi_{Contact})\left\{\theta_{Ft_\beta uB} = linearInterp\left(\frac{\phi_S - \phi_{Contact}}{\phi_{Follow} - \phi_{Contact}} 0, \theta_{Ft_\beta Follow}\right)\right\} \\ &else\left\{\theta_{Ft_\beta uB} = 0\right\} \end{aligned} \tag{289}$$

Assign swing foot to correct kick leg:

$$\begin{aligned} if\,(KickLeg = \text{Left})&\begin{Bmatrix} Ft_{x\,L} & = & Ft_x \\ \theta_{Ft_{L\beta}uB} & = & \theta_{Ft_\beta uB} \end{Bmatrix} \\ else\,if\,(KickLeg = \text{Right})&\begin{Bmatrix} Ft_{x\,R} & = & Ft_x \\ \theta_{Ft_{R\beta}uB} & = & \theta_{Ft_\beta uB} \end{Bmatrix} \end{aligned} \tag{290}$$

Interpolate arm swings:

$$\begin{aligned} if\,(KickLeg = \text{Left})&\begin{Bmatrix} \theta_{ShoulderPitch\,L} & = & linearInterp(\phi_S, \theta_{SWING\,ShoulderPitch\,L[\phi_S=0]}, \theta_{SWING\,ShoulderPitch\,L[\phi_S=1]}) \\ \theta_{ShoulderPitch\,R} & = & linearInterp(\phi_S, \theta_{SWING\,ShoulderPitch\,R[\phi_S=0]}, \theta_{SWING\,ShoulderPitch\,L[\phi_S=1]}) \end{Bmatrix} \\ else\,if\,(KickLeg = \text{Right})&\begin{Bmatrix} \theta_{ShoulderPitch\,L} & = & linearInterp(\phi_S, \theta_{SWING\,ShoulderPitch\,L[\phi_S=0]}, \theta_{SWING\,ShoulderPitch\,R[\phi_S=1]}) \\ \theta_{ShoulderPitch\,R} & = & linearInterp(\phi_S, \theta_{SWING\,ShoulderPitch\,R[\phi_S=0]}, \theta_{SWING\,ShoulderPitch\,R[\phi_S=1]}) \end{Bmatrix} \end{aligned} \tag{291}$$

5.4. Results

This kicking system was built and tested as a stand alone motion control project. As was the primary goal of this thesis, to use the Aldebaran walking system and walk directly into a kick, focusing on the motion control nature of the problem.

The system was tested with a static dummy ball location and a forced 2 cm final correction step. The system was tested in two ways. First by having the robot take 3 Aldebaran walk engine controlled steps at full velocity and switch directly into taking 3 custom designed steps (a design that matched the Aldebaran system) and then one final potentially disruptive ball placement correction step followed by morphing into a large and fast kick swing. The system was also tested by having the robot take 3 full speed zero step length Aldebaran steps and then one single custom built correction step morphing into the same big kick. The check here with the two tests is if forward velocity has an impact on the static stability. In both cases the robot was able to perform a stable and consistent motion, showing no visible signs of risking a fall in 30 trials.

To evaluate the strength of the kick we took 4 forward kick trials and determined the velocity possible at the Nao's operational limit (reduced the swing phase duration until the Nao motion system mechanically failed). Figure 128 shows the recorded forward swing motion given a simulated ball target point of 17 cm.

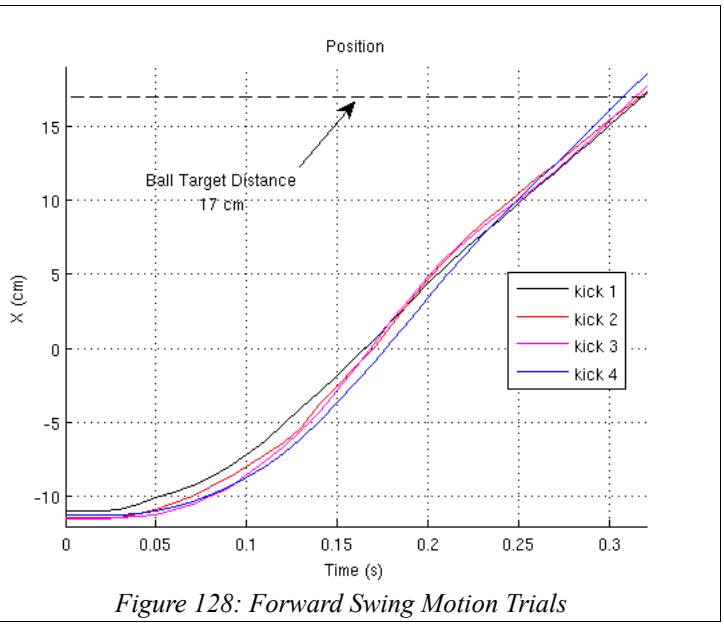


Figure 128: Forward Swing Motion Trials

From this we determined the swing velocity and acceleration data by differentiating the recorded position data (Figure 129 and 130)

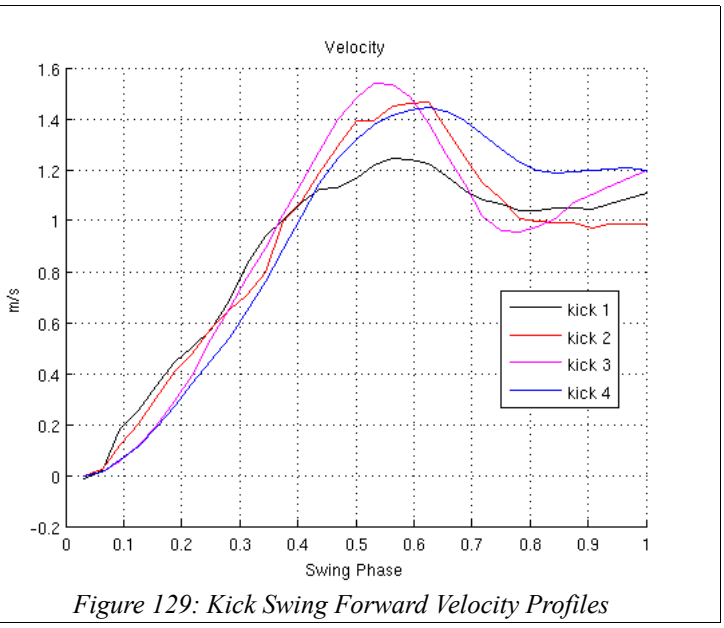


Figure 129: Kick Swing Forward Velocity Profiles

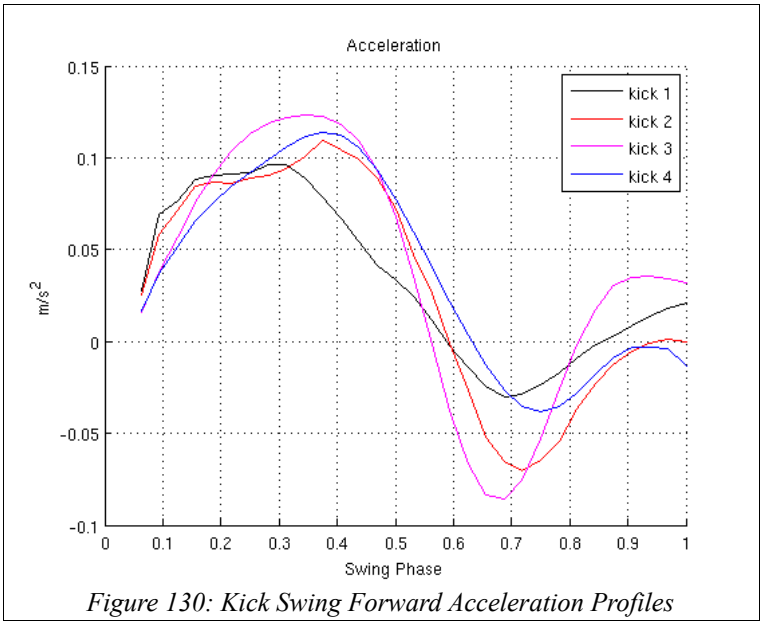


Figure 130: Kick Swing Forward Acceleration Profiles

The velocity profile compares very well with Kollath's study [203] (Figure 26), showing a peak foot velocity just after the swing foot passes the planted foot.

The mean foot velocity at chosen contact point was found to be 1.12 m/s (std 0.101 m/s).

Using Shan's velocity model:

$$V_{Ball} = \frac{V_{Foot} M_{Leg} (1+e)}{M_{Leg} + m_{Ball}}$$

With the following masses:

Ball 0.055 kg
Thigh 0.38976 kg
Tibia 0.29163 kg
Ankle 0.13415 kg
Foot 0.16171 kg

The ball velocity at contact was found to be 3.94 m/s.

The results show that it would be more ideal to move the target point forward such that contact is made just after the swing foot clears the supporting foot. The Nao's feet are very long but it could be possible with a long stride. At that point, the mean foot velocity recorded is 1.42 m/s which is 26.9% faster.

5.5. Summary

With this chapter complete, the entire sequence of charging out of the Aldebaran walk towards a ball and kicking it with a dynamic swing is possible and concludes this project. Driving this system with a higher level behaviour module is covered in Appendix M.

6. Conclusions and Future Work

6.1. Thesis Conclusion

The primary goal of this thesis was to walk directly into a kick using the Aldebaran walk engine. To accomplish this, we required a bumpless transfer from the Aldebaran walk to a new walk design that could make the final few steps. Then the design and development of a dynamic kick along with a morphing gait to transition from the walk was possible. This motion generation system was built as a stand alone module and tested for its stability and high speed performance. The results of this work were outlined in the conclusion of each chapter and demonstrate the systems design to specifications.

This analytical kicking system was designed to complement the existing RoboEireann empirical kick and provide another option. Design work was also continued on that project as outlined in Appendix M. For both kicks, targeting systems were developed along with a striker behaviour algorithm to deliver the robot to the required target position. To complete the project, this behaviour was modified to support either kicking system with each one having subtly different requirements.

We have been able to show in simulation, given the tested torso peak height and introduction of torso roll and height variance parameters, that the frontal plane stability can be significantly increased with the inclusion of a feed-forward Empirical Stability Margin Model. Simulation results show that a 79% increase in walk cycle frequency should be possible while maintaining a consistent lateral stability.

Using the Nao robot hardware, we were able to determine that by raising and lowering the joint stiffness with the walk cycle phase, it is possible to correct the gait over successive walk cycles against larger disturbances that lead to an orbital wobble that frequently results in a fall. This property is a difficult to quantify one as wobbling while walking still falls into the formal definition of stable. The characteristics of the gait were visually observed while applying the same conservative tuning method as would be for a RoboCup match, finding a balance between velocity and observable gait behaviour while being manually disturbed. To achieve the same level of robustness (resilience to moderate pushes) we were able to increase the velocity by 30%.

A bumpless transfer system was designed to switch from the Aldebaran motion system into another motion during the midpoint of dual support phase with a minimized bump. We quantified this bump as a -10.5×10^{-3} rad/s^3 jerk in the hip roll motor.

It had been expected that walking at the top possible speed into a full swing in stride kick was going to be a challenging stability problem given past work has shown that moving from a stationary intial possition into a tall kick pose is very unsteady without closed loop compensation. It came as a surprise that it was far more stable (never falling in 50 trials) to walk directly into the cocked kick pose quickly. This makes sense as this pose is very much like a walking stride pose, just more exaggerated. Using a dynamic cubic splines trajectory generator we were able to achieve similar swing motion properties to those shown to be ideal in bio-mechancal studies. Given a static target point used in the RoboEireann targeting system, we achieved a foot swing velocity of 1.12 m/s (std 0.101 m/s) and a corresponding ball velocity of 3.94 m/s using Shan's velocity transfer model. We also found that this velocity could be increased by 26.9% by moving the target point closer to the robot to where contact is made as the swing foot passes through the frontal plane.

6.2. Thesis Contributions

The various modules developed in this thesis have uses on their own or in combinations.
The modules that have been developed include:

1. A statically stable omnidirectional open loop walk engine design.
2. A Phase Locked Loop to extract the phase from black box walking systems (for example, the Aldebaran walk).

3. A walk cycle phase varying approach to setting joint stiffness.
4. A morphing gait engine that stitches together the end of a walk to the start of a special motion.
5. An analytical kick swing system.
6. An expanded empirical kick system that dynamically compensates for final ball placement error.
7. A targeting system for empirical angle kicks.
8. A versatile striker behaviour algorithm that performs very well, even with hard to control walks such as the Aldebaran walk.
9. An exact inverse kinematics solution that operates in the robots global coordinate system, including body pitch and roll instead of just the torso frame.

The phase locked loop itself can have a few uses. The Aldebaran walk has always come with a 'kill walk hard' command that has in practice been difficult to implement as determining the end of a walk cycle accurately has not been possible. With the PLL it is now possible to halt the walk in a known stable position.

The PLL could also be used to indicate at what point it may be suitable to perform various special motions such as a defensive leg dive when using the Aldebaran walk. The center of the walk would be a good time to trigger a defender save by dropping into a leg splayed pose if the ball was detected to be travelling past laterally.

A very practical, and simple to implement, use of the PLL would be for Aldebaran walk users to explore the phase varying stiffness control method with the latest version of the Nao walk.

The morphing gait module can be used with any walking system once the walk cycle phase has been extracted to morph into the start of any special motion smoothly. An example of practical use for the RoboEireann team would be to stitch together the B-Human walk with the existing empirical kick system. This kick has a different back swing pose for any arbitrary kick angle. The morphing gait has both the input and output body part locations set dynamically. Mapping out the body part locations of the empirical kick would allow for this dynamic connection.

The empirical kick system and its targeting performs excellently. This can be of use for many teams that are still struggling with the kicking problem.

Finally, the striker behaviour velocity profiles should also be of use to many RoboCup SPL teams. It is very robust and does an excellent job of converging on the ball target location very quickly, even if the delayed Aldebaran walking system is being used.

6.3. Future Work

Continuing work in this direction could come in a variety of forms. A more novel direction would be to walk directly into new special motions. Motions such as defender actions or different style of kicks or passes. There is also the in-stride kick that was developed as part of the walking engine. This could be explored further and be used as a means to dribble dynamically without stopping to kick the ball. Or to pass while walking.

The most obvious step forward with the walk design could be to apply machine learning techniques to the currently static parameters of the walk and learn more appropriate values as a function of walk cycle frequency.

What would be of interest to myself would be to explore the open loop properties of a running gait. It can be difficult to abruptly initiate something as dynamic as running. Some form of ramping up of velocities and stability is a good idea. It would also be interesting to explore the idea of varying the torso height and roll in both the walk and the a run and develop a transition between them. As I have observed there are some properties to walking such as; damping more towards the center of dual support phase and less during single support. An increase in height during single support to allow for the clearance of the airborne leg and a decrease going into dual support to allow for a long stride length. However once this maximum stride length has been reached and speed limit has been reached, no further increased velocity is possible. Running on the other hand, by my observations, is characterized by propelling oneself forward into the air to achieve greater stride lengths. To be able to do so, the joint stiffness increases during the propulsion phase going into 'dual support' (which does not actually occur with any ground contact) and then decreasing stiffness going into the landing which is single support. So it would seem to me these height and stiffness properties of a walk and a run are opposite from one another. The pendulum length of the walk engine developed is the value that controls the change in height. Setting this value negative creates an inverted signal. Four torso trajectories with different values of pendulum lengths (one being negative,) are plotted in Figure 131 and shown as a cross section in Figure 132.

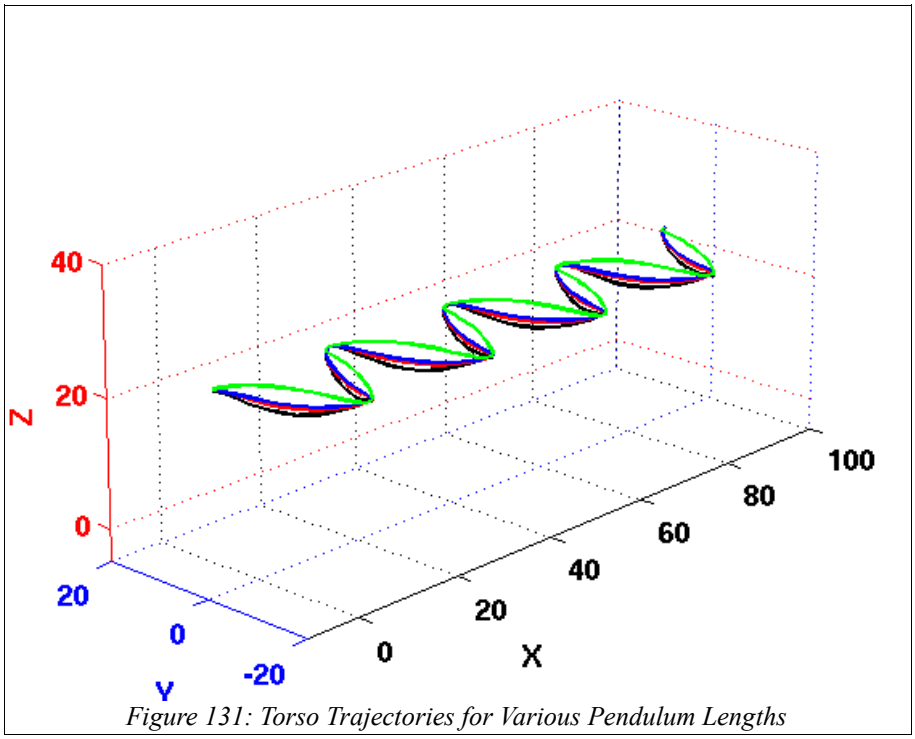


Figure 131: Torso Trajectories for Various Pendulum Lengths

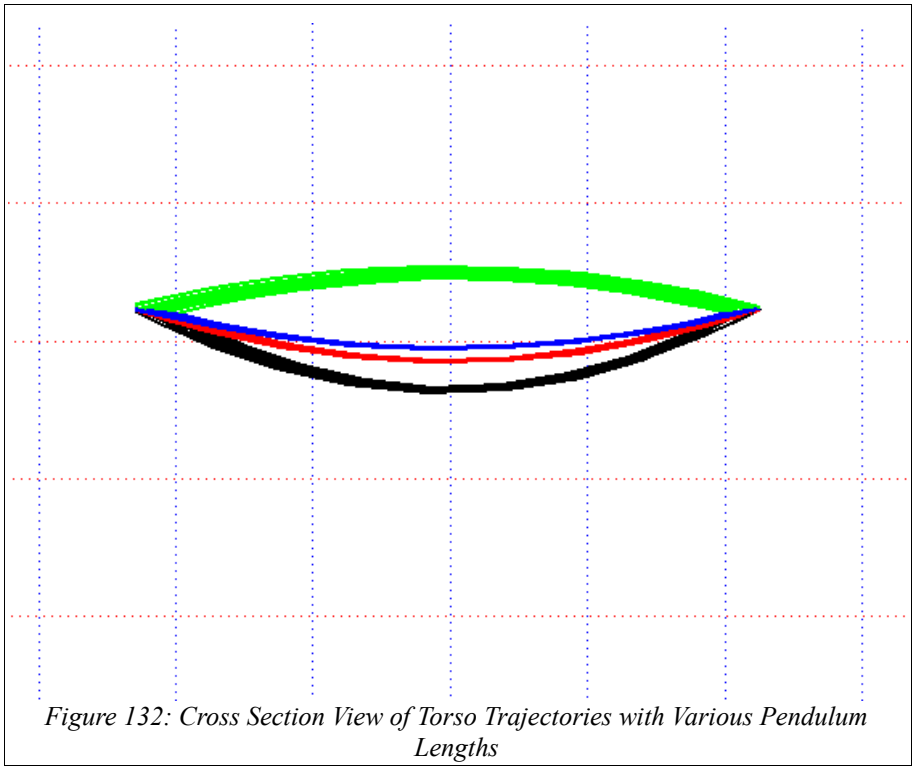


Figure 132: Cross Section View of Torso Trajectories with Various Pendulum Lengths

My idea would be to begin with a signal that is low initially and ramp the pendulum length from a positive value down to a negative one to create the trajectory shown in pink in Figure 133, thereby simultaneously inverting the torso height trajectory and the stiffness signal dampen and spring phase.

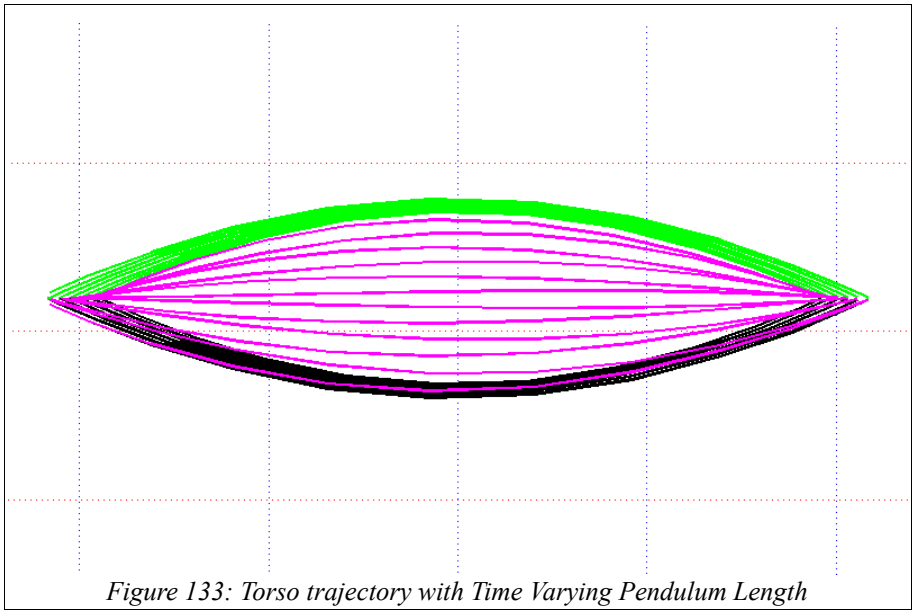


Figure 133: Torso trajectory with Time Varying Pendulum Length

Appendix A - Nao's 22 Degrees of Freedom and Joint Names

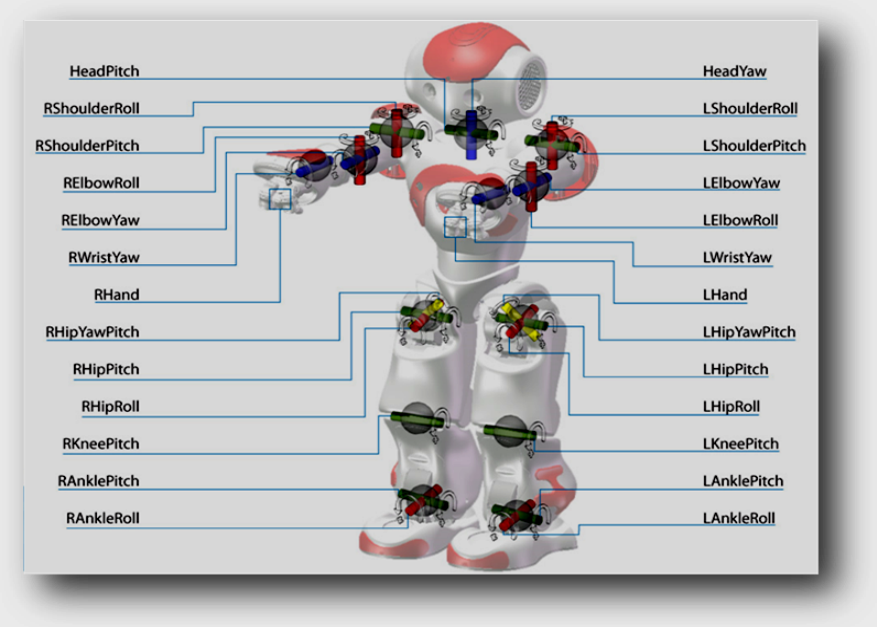


Figure 134: [212] Nao Joint Names

Appendix B - Nao Links Measurements

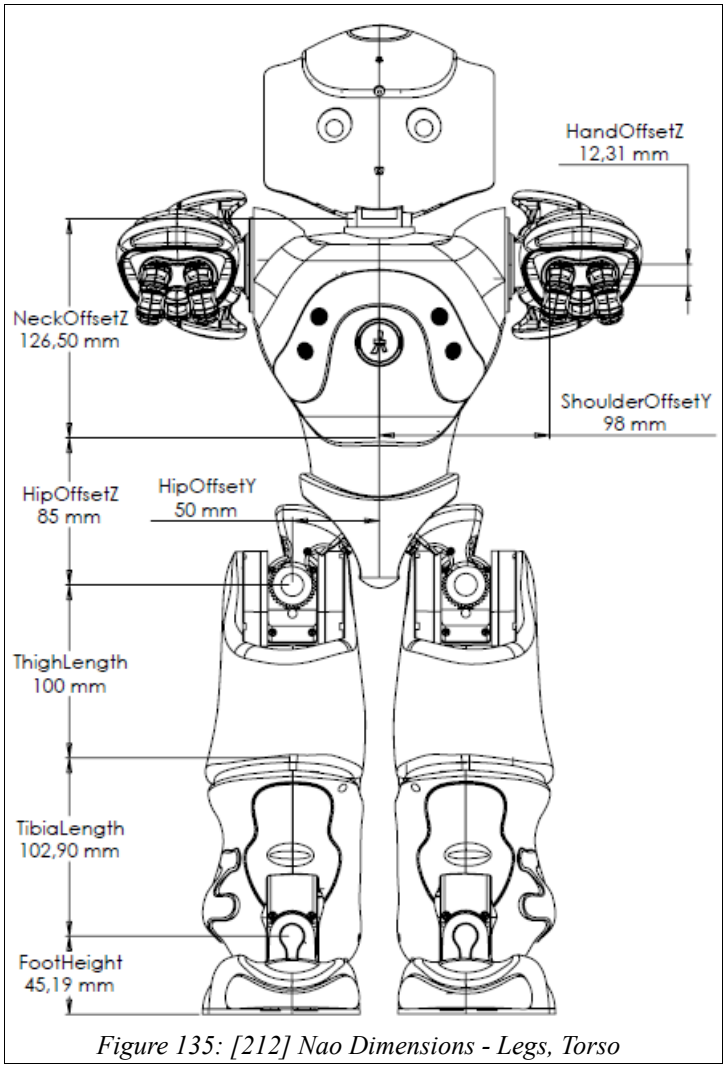


Figure 135: [212] Nao Dimensions - Legs, Torso

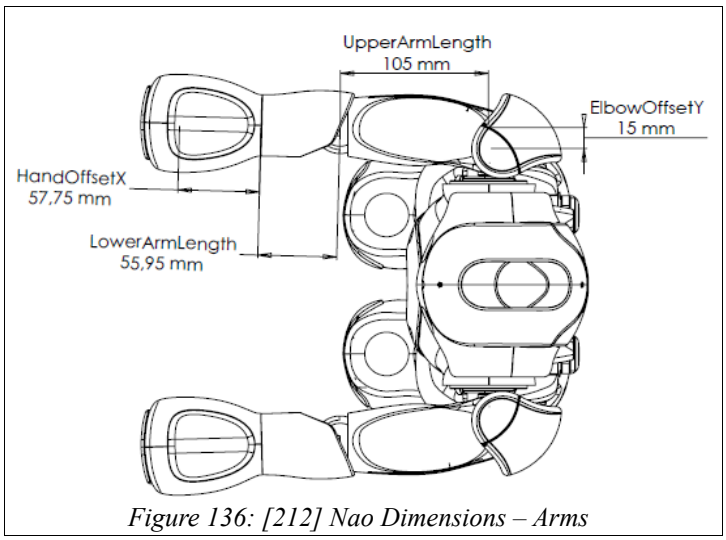


Figure 136: [212] Nao Dimensions – Arms

Appendix C - Kinematic Model of Nao Robot in Zeroed Pose

Body link offsets (cm):

$$
\begin{aligned}
NeckOffsetZ &= 12.65 \\
ShoulderOffsetY &= 9.8 \\
ShoulderOffsetZ &= 10.0 \\
HipOffsetZ &= 8.5 \\
HipOffsetY &= 5.0 \\
ThighLength &= 10.0 \\
TibiaLength &= 10.0 \\
FootHeight &= 4.6 \\
HeadOffsetZ &= 6
\end{aligned}
$$

Define zeroed robot pose data points:

$$
\begin{aligned}
BodyCenter &= [\, BodyOrigin_x \; BodyOrigin_y \; (BodyOrigin_z + HipOffsetZ + ThighLength + TibiaLength + FootHeight) \,] \\
HipCenter &= [\, BodyCenter_x \; BodyCenter_y \; (BodyCenter_z - HipOffsetZ) \,] \\
LeftHip &= [\, BodyCenter_x \; (BodyCenter_y + HipOffsetY) \; (BodyCenter_z - HipOffsetZ) \,] \\
Leftknee &= [\, LeftHip_x \; LeftHip_y \; (LeftHip_z - ThighLength) \,] \\
LeftAnkle &= [\, Leftknee_x \; Leftknee_y \; (Leftknee_z - TibiaLength) \,] \\
LeftFootCenter &= [\, LeftAnkle_x \; LeftAnkle_y \; (LeftAnkle_z - FootHeight) \,] \\
RightHip &= [\, BodyCenter_x \; (BodyCenter_y - HipOffsetY) \; (BodyCenter_z - HipOffsetZ) \,] \\
Rightknee &= [\, RightHip_x \; RightHip_y \; (RightHip_z - ThighLength) \,] \\
RightAnkle &= [\, Rightknee_x \; Rightknee_y \; (Rightknee_z - TibiaLength) \,] \\
RightFootCenter &= [\, RightAnkle_x \; RightAnkle_y \; (RightAnkle_z - FootHeight) \,] \\
Sternum &= [\, BodyCenter_x \; BodyCenter_y \; (BodyCenter_z + ShoulderOffsetZ) \,] \\
LeftShoulder &= [\, Sternum_x \; (Sternum_y + ShoulderOffsetY) \; Sternum_z \,] \\
LeftUpperArm &= [\, (LeftShoulder_x + UpperArmLength) \; LeftShoulder_y \; LeftShoulder_z \,] \\
LeftLowerArm &= [\, (LeftUpperArm_x + LowerArmLength) \; LeftUpperArm_y \; LeftUpperArm_z \,] \\
RightShoulder &= [\, Sternum_x \; (Sternum_y - ShoulderOffsetY) \; Sternum_z \,] \\
RightUpperArm &= [\, (RightShoulder_x + UpperArmLength) \; RightShoulder_y \; RightShoulder_z \,] \\
RightLowerArm &= [\, (RightUpperArm_x + LowerArmLength) \; RightUpperArm_y \; RightUpperArm_z \,] \\
NeckCenter &= [\, BodyCenter_x \; BodyCenter_y \; (BodyCenter_z + NeckOffsetZ) \,] \\
HeadCenter &= [\, NeckCenter_x \; NeckCenter_y \; (NeckCenter_z + HeadOffsetZ) \,]
\end{aligned}
$$

Appendix D - Rotation Matrices Used For Manipulations

Rotation around the X axis: $R_{\phi x}(\Theta_x)$

$$R_{\phi x}=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\Theta_x) & -\sin(\Theta_x)\\ 0 & \sin(\Theta_x) & \cos(\Theta_x)\end{bmatrix}$$

Rotation around the Y axis: $R_{\phi y}(\Theta_y)$

$$R_{\phi y}=\begin{bmatrix}\cos(\Theta_y) & 0 & \sin(\Theta_y)\\ 0 & 1 & 0\\ -\sin(\Theta_y) & 0 & \cos(\Theta_y)\end{bmatrix}$$

Rotation around the Z axis: $R_{\phi z}(\Theta_z)$

$$R_{\phi z}=\begin{bmatrix}\cos(\Theta_z) & -\sin(\Theta_z) & 0\\ \sin(\Theta_z) & \cos(\Theta_z) & 0\\ 0 & 0 & 1\end{bmatrix}$$

Rotation around the Y-Z axis right hip: $R_{\phi yzR}(\Theta_{yzR}, u)$

$$u=\begin{bmatrix}0 & 0.7071 & 0.7071\end{bmatrix}$$

$$R_{\phi yzR}=\begin{bmatrix}u_x^2+(1-u_x^2)*\cos(\theta_{yzR}) & u_x*u_y*(1-\cos(\theta_{yzR}))-u_z*\sin(\theta_{yzR}) & u_x*u_z*(1-\cos(\theta_{yzR}))+u_y*\sin(\theta_{yzR})\\ u_x*u_y*(1-\cos(\theta_{yzR}))+u_z*\sin(\theta_{yzR}) & u_y^2+(1-u_y^2)*\cos(\theta_{yzR}) & u_y*u_z*(1-\cos(\theta_{yzR}))-u_x*\sin(\theta_{yzR})\\ u_x*u_z*(1-\cos(\theta_{yzR}))-u_y*\sin(\theta_{yzR}) & u_y*u_z*(1-\cos(\theta_{yzR}))+u_x*\sin(\theta_{yzR}) & u_z^2+(1-u_z^2)*\cos(\theta_{yzR})\end{bmatrix}$$

Rotation around the Y-Z axis left hip: $Rf_{yzL}(\Theta_{yzL}, u)$

$$u=\begin{bmatrix}0 & -0.7071 & 0.7071\end{bmatrix}$$

$$R_{\phi yzL}=\begin{bmatrix}u_x^2+(1-u_x^2)*\cos(\theta_{yzL}) & u_x*u_y*(1-\cos(\theta_{yzL}))-u_z*\sin(\theta_{yzL}) & u_x*u_z*(1-\cos(\theta_{yzL}))+u_y*\sin(\theta_{yzL})\\ u_x*u_y*(1-\cos(\theta_{yzL}))+u_z*ssin(\theta_{yzL}) & u_y^2+(1-u_y^2)*\cos(\theta_{yzL}) & u_y*u_z*(1-\cos(\theta_{yzL}))-u_x*\sin(\theta_{yzL})\\ u_x*u_z*(1-\cos(\theta_{yzL}))-u_y*\sin(\theta_{yzL}) & u_y*u_z*(1-\cos(\theta_{yzL}))+u_x*\sin(\theta_{yzL}) & u_z^2+(1-u_z^2)*\cos(\theta_{yzL})\end{bmatrix}$$

Rotation around the an arbitrary axis of rotation: $R_{\phi ARB}(\Theta_{ARB}, v_{ARB})$ - source wikipedia

$$scale=\sqrt{v_{ARBx}^2+v_{ARBy}^2+v_{ARBz}^2}$$

$$u=\frac{v_{ARB}}{scale}$$

$$R_{\phi ARB}=\begin{bmatrix}\cos(\theta_{ARB})+u_x^2*(1-\cos(\theta_{ARB})) & u_x^2*(1-\cos(\theta_{ARB}))-u_z*\sin(\theta_{ARB}) & u_x*u_z*(1-\cos(\theta_{ARB}))+u_y*\sin(\theta_{ARB})\\ u_y*u_x*(1-\cos(\theta_{ARB}))+u_z*\sin(\theta_{ARB}) & \cos(\theta_{ARB})+u_y^2*(1-\cos(\theta_{ARB})) & u_x*u_z*(1-\cos(\theta_{ARB}))-u_x*\sin(\theta_{ARB})\\ u_z*u_x*(1-\cos(\theta_{ARB}))-u_y*\sin(\theta_{ARB}) & u_z*u_y*(1-\cos(\theta_{ARB}))+u_x*\sin(\theta_{ARB}) & \cos(\theta_{ARB})+u_z^2*(1-\cos(\theta_{ARB}))\end{bmatrix}$$

Appendix E - Rotation matrix used by each joint

HeadPitch	$R\phi_Y$
HeadYaw	$R\phi_Z$
ShoulderRoll	$R\phi_Z$
ShoulderPitch	$R\phi_Y$
ElbowRoll	$R\phi_Z$
ElbowYaw	$R\phi_X$
RHipYawPitch	$R\phi_{yzR}$
LHipYawPitch	$R\phi_{yzL}$
HipPitch	$R\phi_Y$
HipRoll	$R\phi_X$
KneePitch	$R\phi_Y$
AnklePitch	$R\phi_Y$
AnkleRoll	$R\phi_X$

Table 6: Joint Rotation Matrix Symbols

Appendix F - Newton's Method Parameter Tuning

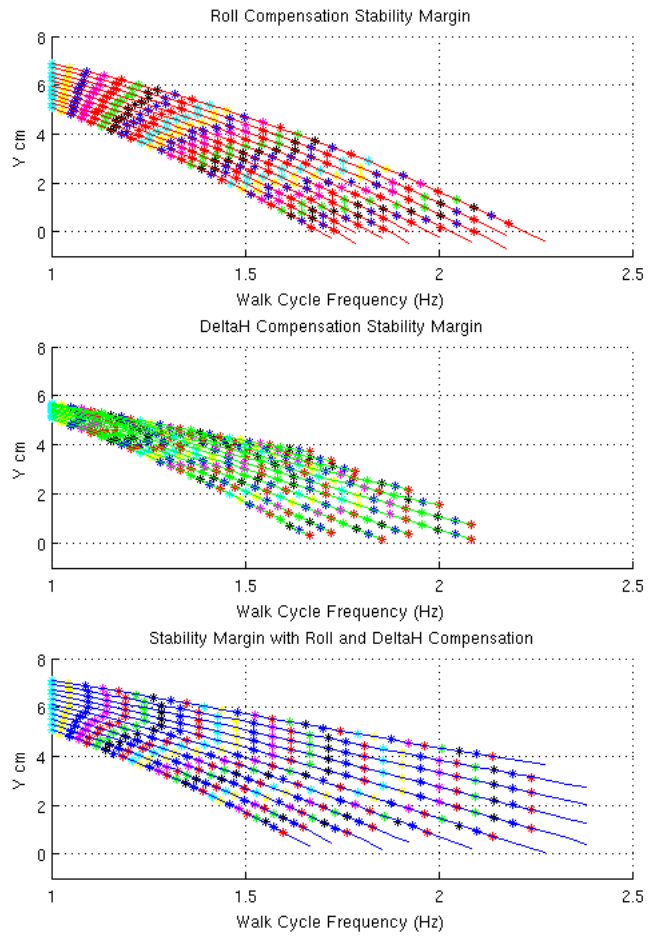


Figure 137: Parameter Space Test Values

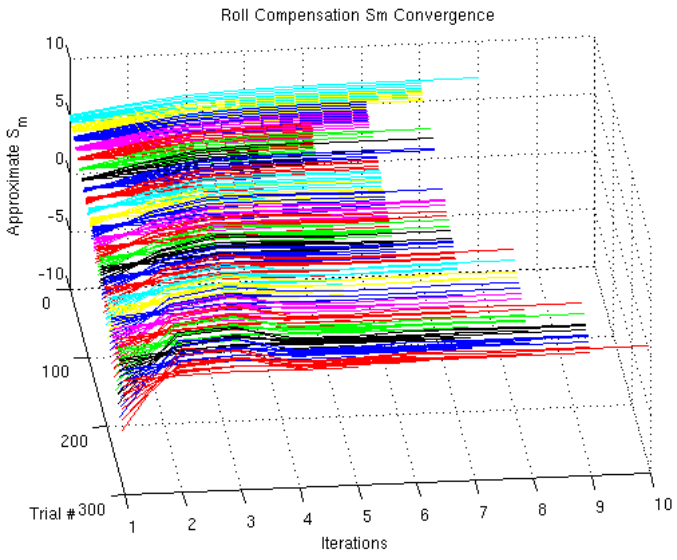


Figure 139:
Figure 138: Balanced Underdamped and Overdamped Convergnce with Roll

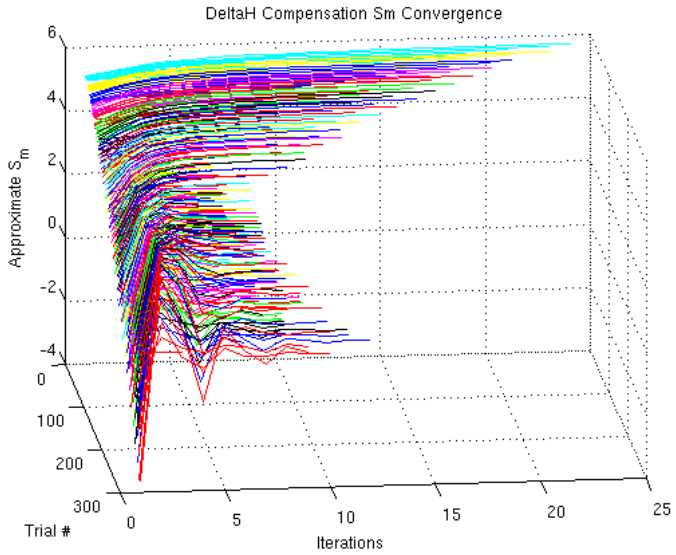


Figure 140: Balanced Underdamped and Overdamped Convergnce with DeltaH

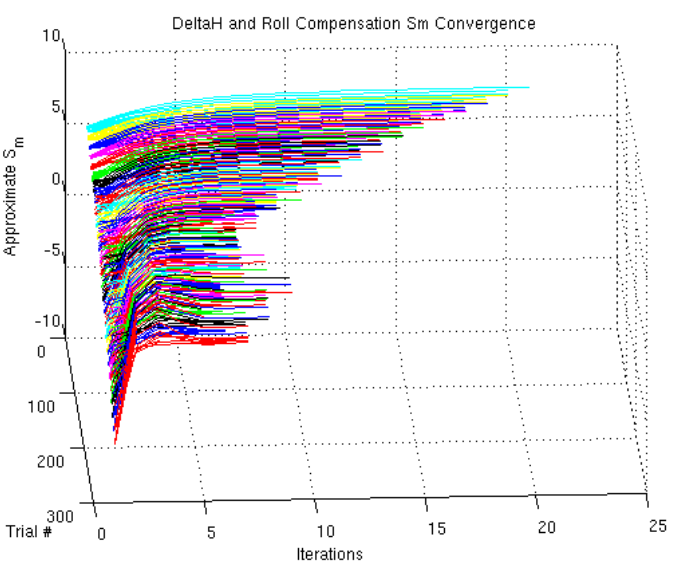


Figure 141: Balanced Underdamped and Overdamped Convergnce with Roll and DeltaH

Appendix G - Stability Margin Function Coefficients

Powers	3	2	1	0
Trial #				
1	-0.032516	-2.5805	-0.19064	7.8812
2	-0.018025	-2.5401	-0.11015	7.9276
3	-0.017679	-2.444	-0.10279	8.0054
4	-0.012522	-2.3685	-0.067213	8.0706
5	0.012023	-2.3756	0.081142	8.0857
6	0.0075606	-2.2634	0.068103	8.1723
7	-0.010482	-2.0962	-0.01909	8.2916
8	0.010108	-2.0944	0.12365	8.3065
9	-0.024501	-1.8499	-0.081748	8.4839
10	-0.02977	-1.7375	-0.094616	8.5698
11	-0.0024081	-1.7753	0.11545	8.5485
DeltaH				
Powers	3	2	1	0
Trial #				
1	0.33385	-3.8982	1.4094	7.2313
2	0.20179	-2.6691	-0.40178	8.0328
3	0.43213	-3.0049	-0.11429	7.9291
4	0.51003	-2.8904	-0.30216	7.998
5	0.57723	-2.8558	-0.29909	7.9604
6	0.58726	-2.6565	-0.48705	8.0028
7	0.41326	-1.7214	-1.6876	8.5065
8	0.28431	-1.0322	-2.5216	8.8396
9	0.28025	-0.90947	-2.5603	8.8127
10	0.39879	-1.2815	-1.9713	8.5268
11	0.22439	-0.52259	-2.8411	8.8618
Roll + DeltaH				
Powers	3	2	1	0
Trial #				
1	0.33318	-3.9017	1.4218	7.2229
2	0.050432	-2.3317	-0.59558	8.1797
3	0.22754	-2.6153	-0.20923	8.1182
4	0.44507	-3.1708	0.62218	7.8361
5	0.3149	-2.3295	-0.43101	8.3917
6	0.38112	-2.3834	-0.20848	8.3614
7	0.33513	-1.974	-0.65044	8.6445
8	0.30614	-1.6856	-0.8945	8.8282
9	0.27438	-1.4176	-1.0992	8.9919
10	0.24185	-1.1728	-1.2645	9.1366
11	0.30157	-1.3834	-0.73812	8.944

Table 7: Stability Margin Function Coefficients

Roll						
Powers	5	4	3	2	1	0
α	0.0005353	-0.00614	0.022967	-0.03485	0.03336	-0.03190
β	-0.002482	0.028679	-0.10861	0.16342	0.033488	-2.5826
c	0.0037144	-0.04316	0.16486	-0.2485	0.21601	-0.18861
d	-0.001792	0.020893	-0.08027	0.12119	0.047036	7.8806
DeltaH						
Powers	5	4	3	2	1	0
α	-0.045656	0.48117	-1.7725	2.5806	-1.0939	0.33558
β	0.18273	-1.9372	7.2271	-11.01	6.4439	-3.9109
c	-0.23977	2.553	-9.5999	14.935	-9.2062	1.4331
d	0.10321	-1.1027	4.1688	-6.5605	4.0953	7.2182
Roll + DeltaH						
Powers	5	4	3	2	1	0
α	-0.000283	0.009412	-0.11514	0.62344	-1.3937	1.1887
β	0.0010844	-0.03636	0.44855	-2.4718	6.0227	-7.7843
c	-0.001366	0.046153	-0.57349	3.1897	-7.7923	6.4475
d	0.0005653	-0.01926	0.24098	-1.349	3.4068	4.9878

Table 8: Coefficients as a Function of Parameter Value

Appendix H - Arm Parametrization and Initialization Table

$SP_{Range} = 30$
$SR_{Range} = 15$
$ER_{Range} = 20$
$LSP_{Init} = 120$
$LSR_{Init} = 15$
$LER_{Init} = -80$
$RSP_{Init} = 120$
$RSR_{Init} = -15$
$RER_{Init} = 80$
$LSP_{MIN} = 117$
$LSP_{MAX} = LSP_{MIN} - SP_{Range}$
$RSP_{MIN} = 117$
$RSP_{MAX} = RSP_{MIN} - SP_{Range}$
$LER_{MIN} = 0$
$RER_{MIN} = 0$
$LER_{MAX} = -ER_{Range}$
$RER_{MAX} = ER_{Range}$
$SR_{MIN} = 25$
$SR_{MAX} = SR_{MIN} + SR_{Range}$

(292)

Table 9: Arm Parameterization and Initialization Table

Appendix I - Forward kinematics for an entire simulated robot in MATLAB

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% NAO JOINT ROTATIONS  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Alexander Buckley Hamilton Institute, NUIM 2011 %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% All rotations are performed on robot zeroed position,
% not relative to the last incremental position!!!
% Do not change the order of any operation!
% Rotations must begin at extremities and work back to body center
% Also the order of rotation matrices in multiplications is important

%% Neck Rotation (do yaw first then pitch)
HeadPitchM = Ry(eye(3),HeadPitch,'radians');
HeadYawM   = Rz(eye(3),HeadYaw,'radians');
HeadCenter   = ( HeadYawM   * (HeadCenter-NeckCenter)')'+NeckCenter;
HeadCenter   = ( HeadPitchM * (HeadCenter-NeckCenter)')'+NeckCenter;

% bottom CAMERA
botCamFocalPoint  = ( HeadYawM   * (botCamFocalPoint-NeckCenter)')'+NeckCenter;
botCamFocalPoint  = ( HeadPitchM * (botCamFocalPoint-NeckCenter)')'+NeckCenter;
botCamPlaneCenter = ( HeadYawM   * (botCamPlaneCenter-NeckCenter)')'+NeckCenter;
botCamPlaneCenter = ( HeadPitchM * (botCamPlaneCenter-NeckCenter)')'+NeckCenter;
% bottom camera sensor frame corners:
botCamPlaneTopRightCrnr = ( HeadYawM   * (botCamPlaneTopRightCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneTopRightCrnr = ( HeadPitchM * (botCamPlaneTopRightCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneTopLeftCrnr  = ( HeadYawM   * (botCamPlaneTopLeftCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneTopLeftCrnr  = ( HeadPitchM * (botCamPlaneTopLeftCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneBtmLeftCrnr  = ( HeadYawM   * (botCamPlaneBtmLeftCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneBtmLeftCrnr  = ( HeadPitchM * (botCamPlaneBtmLeftCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneBtmRightCrnr = ( HeadYawM   * (botCamPlaneBtmRightCrnr-NeckCenter)')'+NeckCenter;
botCamPlaneBtmRightCrnr = ( HeadPitchM * (botCamPlaneBtmRightCrnr-NeckCenter)')'+NeckCenter;
% 4 value vectors for the corners of sensor plane:
xsensorbtm     =     [     botCamPlaneBtmRightCrnr(X)     botCamPlaneBtmLeftCrnr(X)     botCamPlaneTopLeftCrnr(X)
botCamPlaneTopRightCrnr(X) ];
ysensorbtm     =     [     botCamPlaneBtmRightCrnr(Y)     botCamPlaneBtmLeftCrnr(Y)     botCamPlaneTopLeftCrnr(Y)
botCamPlaneTopRightCrnr(Y) ];
zsensorbtm     =     [     botCamPlaneBtmRightCrnr(Z)     botCamPlaneBtmLeftCrnr(Z)     botCamPlaneTopLeftCrnr(Z)
botCamPlaneTopRightCrnr(Z) ];
% X - Y cross hairs on cam plane
botcamYmidLft = ( HeadYawM   * (botcamYmidLft-NeckCenter)')'+NeckCenter;
botcamYmidLft = ( HeadPitchM * (botcamYmidLft-NeckCenter)')'+NeckCenter;
botcamYmidRht = ( HeadYawM   * (botcamYmidRht-NeckCenter)')'+NeckCenter;
botcamYmidRht = ( HeadPitchM * (botcamYmidRht-NeckCenter)')'+NeckCenter;
botcamXmidTop = ( HeadYawM   * (botcamXmidTop-NeckCenter)')'+NeckCenter;
botcamXmidTop = ( HeadPitchM * (botcamXmidTop-NeckCenter)')'+NeckCenter;
botcamXmidBtm = ( HeadYawM   * (botcamXmidBtm-NeckCenter)')'+NeckCenter;
botcamXmidBtm = ( HeadPitchM * (botcamXmidBtm-NeckCenter)')'+NeckCenter;

%% left elbow rotation
% Left lower arm
LeftElbowRollM = Rz(eye(3),LElbowRoll,'radians');
LeftElbowYawM   = Rx(eye(3),LElbowYaw,'radians');
LeftLowerArm   = ( LeftElbowRollM * (LeftLowerArm-LeftUpperArm)')'+LeftUpperArm;
LeftLowerArm   = ( LeftElbowYawM  * (LeftLowerArm-LeftUpperArm)')'+LeftUpperArm;
%% left shoulder rotation
% Left upper arm  - includes lower arm
LeftShoulderPitchM = Ry(eye(3),LShoulderPitch,'radians');
LeftShoulderRollM  = Rz(eye(3),LShoulderRoll,'radians');
LeftUpperArm   = ( LeftShoulderPitchM * LeftShoulderRollM * (LeftUpperArm-LeftShoulder)')'+LeftShoulder;
LeftLowerArm   = ( LeftShoulderPitchM * LeftShoulderRollM * (LeftLowerArm-LeftShoulder)')'+LeftShoulder;
%% right elbow rotation
RightElbowRollM = Rz(eye(3),RElbowRoll,'radians');
RightElbowYawM   = Rx(eye(3),RElbowYaw,'radians');
RightLowerArm   = ( RightElbowRollM * (RightLowerArm-RightUpperArm)')'+RightUpperArm;
RightLowerArm   = ( RightElbowYawM  * (RightLowerArm-RightUpperArm)')'+RightUpperArm;

%% right shoulder rotation
RightShoulderPitchM = Ry(eye(3),RShoulderPitch,'radians');
RightShoulderRollM  = Rz(eye(3),RShoulderRoll,'radians');
RightUpperArm   = (RightShoulderPitchM * RightShoulderRollM * (RightUpperArm-RightShoulder)')'+RightShoulder;
RightLowerArm   = (RightShoulderPitchM * RightShoulderRollM * (RightLowerArm-RightShoulder)')'+RightShoulder;

%%  Left Ankle Rotation
LAnklePitchM = Ry(eye(3),LAnklePitch,'radians');
LAnkleRollM  = Rx(eye(3),LAnkleRoll,'radians');
LeftFootCenter     = ( LAnklePitchM * LAnkleRollM * (LeftFootCenter-LeftAnkle)')'+LeftAnkle;
LfootOrigin         = ( LAnklePitchM * LAnkleRollM * (LfootOrigin-LeftAnkle)')'+LeftAnkle;
LfootFrntLeftCrnr   = ( LAnklePitchM * LAnkleRollM * (LfootFrntLeftCrnr-LeftAnkle)')'+LeftAnkle;
LfootFrntRghtCrnr   = ( LAnklePitchM * LAnkleRollM * (LfootFrntRghtCrnr-LeftAnkle)')'+LeftAnkle;
LfootBackLeftCrnr   = ( LAnklePitchM * LAnkleRollM * (LfootBackLeftCrnr-LeftAnkle)')'+LeftAnkle;
LfootBackRghtCrnr   = ( LAnklePitchM * LAnkleRollM * (LfootBackRghtCrnr-LeftAnkle)')'+LeftAnkle;
LfootPointer        = ( LAnklePitchM * LAnkleRollM * (LfootPointer-LeftAnkle)')'+LeftAnkle;

for i=1:length(RfootXdata)
temp   =   (   RHipPitchM   *   RHipRollM   *   RHipYawPitchM   *   ([Rfootdata(i,X)   Rfootdata(i,Y)   Rfootdata(i,Z)]-
RightHip)')'+RightHip;
    Rfootdata(i,X) = temp(X);
    Rfootdata(i,Y) = temp(Y);
    Rfootdata(i,Z) = temp(Z);
end

for i=1:length(LfootXdata)
          temp   =   (   LAnklePitchM   *   LAnkleRollM   *   ([Lfootdata(i,X)   Lfootdata(i,Y)   Lfootdata(i,Z)]-
LeftAnkle)')'+LeftAnkle;
    Lfootdata(i,X) = temp(X);
    Lfootdata(i,Y) = temp(Y);
    Lfootdata(i,Z) = temp(Z);
```

```
end

%% Left knee Rotation
LKneePitchM = Ry(eye(3),LKneePitch,'radians');
LeftAnkle           = ( LKneePitchM * (LeftAnkle-Leftknee)')'+Leftknee;
LeftFootCenter      = ( LKneePitchM * (LeftFootCenter-Leftknee)')'+Leftknee;
LfootOrigin         = ( LKneePitchM * (LfootOrigin-Leftknee)')'+Leftknee;
LfootFrntLeftCrnr   = ( LKneePitchM * (LfootFrntLeftCrnr-Leftknee)')'+Leftknee;
LfootFrntRghtCrnr   = ( LKneePitchM * (LfootFrntRghtCrnr-Leftknee)')'+Leftknee;
LfootBackLeftCrnr   = ( LKneePitchM * (LfootBackLeftCrnr-Leftknee)')'+Leftknee;
LfootBackRghtCrnr   = ( LKneePitchM * (LfootBackRghtCrnr-Leftknee)')'+Leftknee;
LfootPointer        = ( LKneePitchM * (LfootPointer-Leftknee)')'+Leftknee;

%Lfootdata          = ( LKneePitch * (Lfootdata-Leftknee)')'+Leftknee;
for i=1:length(LfootXdata)
    temp = ( LKneePitchM * ([Lfootdata(i,X) Lfootdata(i,Y) Lfootdata(i,Z)]-Leftknee)')'+Leftknee;
    Lfootdata(i,X) = temp(X);
    Lfootdata(i,Y) = temp(Y);
    Lfootdata(i,Z) = temp(Z);
end
%% Left Hip Rotation
LHipRollM = Rx(eye(3),LHipRoll,'radians');
LHipPitchM  = Ry(eye(3),LHipPitch,'radians');
LHipYawPitchM  = R45L(eye(3),LHipYawPitch,'radians');
Leftknee            = ( LHipPitchM * LHipRollM * LHipYawPitchM * (Leftknee-LeftHip)')'+LeftHip;
LeftAnkle           = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LeftAnkle-LeftHip)')'+LeftHip;
LeftFootCenter      = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LeftFootCenter-LeftHip)')'+LeftHip;
LfootOrigin         = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootOrigin-LeftHip)')'+LeftHip;
LfootFrntLeftCrnr   = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootFrntLeftCrnr-LeftHip)')'+LeftHip;
LfootFrntRghtCrnr   = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootFrntRghtCrnr-LeftHip)')'+LeftHip;
LfootBackLeftCrnr   = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootBackLeftCrnr-LeftHip)')'+LeftHip;
LfootBackRghtCrnr   = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootBackRghtCrnr-LeftHip)')'+LeftHip;
LfootPointer        = ( LHipPitchM * LHipRollM * LHipYawPitchM * (LfootPointer-LeftHip)')'+LeftHip;

for i=1:length(LfootXdata)
       temp  =  (  LHipPitchM  *  LHipRollM  *  LHipYawPitchM  *  ([Lfootdata(i,X)  Lfootdata(i,Y)  Lfootdata(i,Z)]-
LeftHip)')'+LeftHip;
    Lfootdata(i,X) = temp(X);
    Lfootdata(i,Y) = temp(Y);
    Lfootdata(i,Z) = temp(Z);
end

%% Right Ankle Rotation (rotates all foot vertices)
RAnklePitchM = Ry(eye(3),RAnklePitch,'radians');
RAnkleRollM  = Rx(eye(3),RAnkleRoll,'radians');
RightFootCenter     = ( RAnklePitchM * RAnkleRollM * (RightFootCenter-RightAnkle)')'+RightAnkle;
RfootOrigin         = ( RAnklePitchM * RAnkleRollM * (RfootOrigin-RightAnkle)')'+RightAnkle;
RfootFrntLeftCrnr   = ( RAnklePitchM * RAnkleRollM * (RfootFrntLeftCrnr-RightAnkle)')'+RightAnkle;
RfootFrntRghtCrnr   = ( RAnklePitchM * RAnkleRollM * (RfootFrntRghtCrnr-RightAnkle)')'+RightAnkle;
RfootBackLeftCrnr   = ( RAnklePitchM * RAnkleRollM * (RfootBackLeftCrnr-RightAnkle)')'+RightAnkle;
RfootBackRghtCrnr   = ( RAnklePitchM * RAnkleRollM * (RfootBackRghtCrnr-RightAnkle)')'+RightAnkle;
RfootPointer        = ( RAnklePitchM * RAnkleRollM * (RfootPointer-RightAnkle)')'+RightAnkle;


for i=1:length(RfootXdata)
          temp   =   (   RAnklePitchM   *   RAnkleRollM   *   ([Rfootdata(i,X)   Rfootdata(i,Y)   Rfootdata(i,Z)]-
RightAnkle)')'+RightAnkle;
    Rfootdata(i,X) = temp(X);
    Rfootdata(i,Y) = temp(Y);
    Rfootdata(i,Z) = temp(Z);
end

%% Right knee Rotation
RKneePitchM = Ry(eye(3),RKneePitch,'radians');
RightAnkle          = ( RKneePitchM * (RightAnkle-Rightknee)')'+Rightknee;
RightFootCenter     = ( RKneePitchM * (RightFootCenter-Rightknee)')'+Rightknee;
RfootOrigin         = ( RKneePitchM * (RfootOrigin-Rightknee)')'+Rightknee;
RfootFrntLeftCrnr   = ( RKneePitchM * (RfootFrntLeftCrnr-Rightknee)')'+Rightknee;
RfootFrntRghtCrnr   = ( RKneePitchM * (RfootFrntRghtCrnr-Rightknee)')'+Rightknee;
RfootBackLeftCrnr   = ( RKneePitchM * (RfootBackLeftCrnr-Rightknee)')'+Rightknee;
RfootBackRghtCrnr   = ( RKneePitchM * (RfootBackRghtCrnr-Rightknee)')'+Rightknee;
RfootPointer        = ( RKneePitchM * (RfootPointer-Rightknee)')'+Rightknee;

%Rfootdata          = ( RKneePitch * (Rfootdata-Rightknee)')'+Rightknee;
for i=1:length(RfootXdata)
    temp = ( RKneePitchM * ([Rfootdata(i,X) Rfootdata(i,Y) Rfootdata(i,Z)]-Rightknee)')'+Rightknee;
    Rfootdata(i,X) = temp(X);
    Rfootdata(i,Y) = temp(Y);
    Rfootdata(i,Z) = temp(Z);
end

%% Right Hip Rotation
RHipRollM = Rx(eye(3),RHipRoll,'radians');
RHipPitchM  = Ry(eye(3),RHipPitch,'radians');
RHipYawPitchM  = R45R(eye(3),RHipYawPitch,'radians');
Rightknee              = ( RHipPitchM * RHipRollM * RHipYawPitchM * (Rightknee-RightHip)')'+RightHip;RightAnkle
= ( RHipPitchM * RHipRollM * RHipYawPitchM * (RightAnkle-RightHip)')'+RightHip;
RightFootCenter     = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RightFootCenter-RightHip)')'+RightHip;
RfootOrigin         = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootOrigin-RightHip)')'+RightHip;
RfootFrntLeftCrnr   = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootFrntLeftCrnr-RightHip)')'+RightHip;
RfootFrntRghtCrnr   = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootFrntRghtCrnr-RightHip)')'+RightHip;
RfootBackLeftCrnr   = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootBackLeftCrnr-RightHip)')'+RightHip;
RfootBackRghtCrnr   = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootBackRghtCrnr-RightHip)')'+RightHip;
RfootPointer        = ( RHipPitchM * RHipRollM * RHipYawPitchM * (RfootPointer-RightHip)')'+RightHip;
for i=1:length(RfootXdata)
       temp  =  (  RHipPitchM  *  RHipRollM  *  RHipYawPitchM  *  ([Rfootdata(i,X)  Rfootdata(i,Y)  Rfootdata(i,Z)]-
RightHip)')'+RightHip;
    Rfootdata(i,X) = temp(X);
    Rfootdata(i,Y) = temp(Y);
    Rfootdata(i,Z) = temp(Z);
end
```

Appendix J - Solution for three example velocity profile transfer functions

```
calculateVelocity( minVelocity,  maxVelocity,  minDistance,  maxDistance, inputDistance)
{
   if(fabs(inputDistance) < minDistance)        # lower saturation
      outPutVelocity = minVelocity
   else if(fabs(inputDistance) > maxDistance)    # upper saturation
      outPutVelocity = maxVelocity
   else
        ## b = y1 -m*x1  (gain offset)
        b = minVelocity - ((maxVelocity - minVelocity) / ( maxDistance - minDistance))*minDistance
        ## y = mx+b
        outPutVelocity = (fabs(inputDistance) * ((maxVelocity - minVelocity) / ( maxDistance - minDistance))) + b

   if(outPutVelocity > 1.0)
      outPutVelocity = 1.0                              # clip at maximum
   if( inputDistance < 0)
      outPutVelocity = outPutVelocity * (-1)            # go opposite direction with negative distances

   return outPutVelocity
}

computeVelocityX(DistanceX)
{
   MIN_DISTANCE_X = 0.22 # was 0.20
   MAX_DISTANCE_X = 0.45
   MIN_VELOCITY_X = 0.1
   MAX_VELOCITY_X = 1.0
     velocityX = calculateVelocity(MIN_VELOCITY_X, MAX_VELOCITY_X, MIN_DISTANCE_X, MAX_DISTANCE_X,
DistanceX)
   return velocityX
}

computeVelocityY(DistanceY)
{
   MIN_DISTANCE_Y = 0.1
   MAX_DISTANCE_Y = 0.4
   MIN_VELOCITY_Y = 0.2 #0.2
   MAX_VELOCITY_Y = 0.6

      velocityY = calculateVelocity(MIN_VELOCITY_Y, MAX_VELOCITY_Y, MIN_DISTANCE_Y, MAX_DISTANCE_Y,
DistanceY)
   return velocityY
}

computeVelocityThetaForEncircleBall(DistanceTheta)
{
   MIN_DISTANCE_THETA = 5 * pi/180
   MAX_DISTANCE_THETA = 45 * pi/180
   MIN_VELOCITY_THETA = 0.25
   MAX_VELOCITY_THETA = 0.75
     velocityTheta = calculateVelocity(MIN_VELOCITY_THETA, MAX_VELOCITY_THETA, MIN_DISTANCE_THETA,
MAX_DISTANCE_THETA, DistanceTheta)
   return velocityTheta
}
```

Example of Call:

```
if(ball.model.angle()* pi/180 > ORIENTATION_LOCK_ERROR_BOUND)
        rotationVelocity = computeVelocityThetaForEncircleBall(ball.model.angle())
else
        rotationVelocity = 0.0
```

Appendix K - Striker Behaviour

```
requires "../outputHelper/motion.bs"

func calculateVelocity(float minVelocity, float maxVelocity, float minDistance, float maxDistance, float inputDistance) : float
   .float outPutVelocity
   .float b

   if(math::fabs(inputDistance) < minDistance)        # lower saturation
      outPutVelocity = minVelocity
   else if(math::fabs(inputDistance) > maxDistance)    # upper saturation
      outPutVelocity = maxVelocity
   else
      b = minVelocity - ((maxVelocity - minVelocity) / ( maxDistance - minDistance))*minDistance                  # b = y1 -m*x1
(gain offset)
       outPutVelocity = (math::fabs(inputDistance) * ((maxVelocity - minVelocity) / ( maxDistance - minDistance))) + b # y =
mx+b  (velocity profile)

   if(outPutVelocity > 1.0)
      outPutVelocity = 1.0                     # clip at maximum (i don't trust naoqi)
   if( inputDistance < 0)
      outPutVelocity = outPutVelocity * (-1)        # go oposite direction with negative distances

   return outPutVelocity

func computeVelocityX(float DistanceX) : float
   .float MIN_DISTANCE_X = 0.22 # was 0.20
   .float MAX_DISTANCE_X = 0.45
   .float MIN_VELOCITY_X = 0.1
   .float MAX_VELOCITY_X = 1.0
   .float velocityX
    velocityX = calculateVelocity(MIN_VELOCITY_X, MAX_VELOCITY_X, MIN_DISTANCE_X, MAX_DISTANCE_X,
DistanceX)
   return velocityX

func computeVelocityY(float DistanceY) : float
   .float MIN_DISTANCE_Y = 0.1
   .float MAX_DISTANCE_Y = 0.4
   .float MIN_VELOCITY_Y = 0.2 #0.2
   .float MAX_VELOCITY_Y = 0.6
   .float velocityY
     velocityY = calculateVelocity(MIN_VELOCITY_Y, MAX_VELOCITY_Y, MIN_DISTANCE_Y, MAX_DISTANCE_Y,
DistanceY)
   return velocityY

func computeVelocityTheta(float DistanceTheta) : float
   .float MIN_DISTANCE_THETA = 5 * math::pi/180
   .float MAX_DISTANCE_THETA = 45 * math::pi/180
   .float MIN_VELOCITY_THETA = 0.25
   .float MAX_VELOCITY_THETA = 0.75
   .float velocityTheta
     velocityTheta = calculateVelocity(MIN_VELOCITY_THETA, MAX_VELOCITY_THETA, MIN_DISTANCE_THETA,
MAX_DISTANCE_THETA, DistanceTheta)
   return velocityTheta


   ##############################################
   ########## MISSILE FROM DISTANCE #############
   ##############################################
task gotoStartMissilePosition(bool kickOff)
   for(; true;)
      if(input.ball.model.abs() < 1) ## found it!
         break
      TRACES("APPROACHING BALL")
      .float missileXvelocity
      missileXvelocity = calculateVelocity(0.2, 0.8, 0.22, 0.55, input.ball.model.x)
      motion::walkObstacle2(missileXvelocity, 0.0, input.ball.model.angle() * 0.8) # correct this code
      #headControl::lookAroundWithBall()
      headControl::lookAtBallModel()
      yield



   ##############################################
   ############## ENCIRCLE BALL #################
```

```
    ##############################################
    .float ySpeed
    .float rotationVelocity
    .float angleToTarget

    .float ORIENTATION_LOCK_ERROR_BOUND = 0 * math::pi/180
    .float direction = math::sgn(input.angleToGoal)
    for(; true;)
        if(kickOff)
            angleToTarget = input.angleToKickoffPoint
        else
            angleToTarget = input.angleToGoal
        if((math::fabs(angleToTarget) < 0.2) or (math::fabs(angleToTarget) < 0.60 and !input.game.penaltyShoot) or (input.role ==
RoboEireann::Roles::GOALIE and math::fabs(angleToTarget) < 0.78))
            break

        if(input.ball.model.abs() > 0.6)
            yield TaskState::failure

        TRACES("ENCIRLING BALL")
        TRACEF("KICK:", angleToTarget)
        TRACEF("Goal:", input.angleToGoal)
        TRACEF("KOP:", input.angleToKickoffPoint)

        ySpeed = -1.0 * direction # hmmm consider use of computeVelocityTheta
        if(math::fabs(angleToTarget) > math::pi / 2 and math::sgn(angleToTarget) != direction)
            ySpeed = -ySpeed

        if(input.ball.model.angle()* math::pi/180 > ORIENTATION_LOCK_ERROR_BOUND) # remove error bound?
            rotationVelocity = input.ball.model.angle() * 3.9
        else
            rotationVelocity = 0.0

        motion::walkObstacle2(0.0, ySpeed, rotationVelocity ) #input.kickParameters.encycleRot)

        #headControl::lookAroundWithBall()
        headControl::lookAtBallModel()
        yield


    ##############################################
    ############## MISSILE APROACH ###############
    ##############################################

task missileToKick(bool kickOff)
    ######## variable declarations #######

    ### empirical kick vars ###
    .float angleToTarget
    .float angleToTargetDeg
    .float xSpeed
    .float ySpeed
    .float thetaSpeed
    .float missilePhaseTerminationDistanceX = 0.22 # conservative - pull this back for dynamic kick so set switchable
    .float missilePhaseTerminationDistanceY = 0.2

    ### dynamic kick vars ###
    .RoboEireann::Vector2f dynamicKickIdealTargetV
    dynamicKickIdealTargetV.x = 100.0
    dynamicKickIdealTargetV.y = 100.0
    .RoboEireann::Vector2f dynamicKickTargetingError

    .float LEG_DECISION_RANGE = 0.35

    .float NEAR_SIDE_BOX_EDGE_X = 0.01
    .float FARSIDE_BOX_EDGE_X   = 0.02
    .float INSIDE_BOX_EDGE_Y    = 0.01
    .float OUTSIDE_BOX_EDGE_Y   = 0.05

    .float IDEAL_TARGET_POINT_X = 0.17
    .float IDEAL_TARGET_POINT_Y = 0.05


    .float ballDistance
    .float thetaBallIdeal
    .float thetaBallActual
    .float correctionTheta
    .bool dynamicKickLeg
    .bool X_Locked = false
```

```
.bool Y_Locked = false

.RoboEireann::Vector2f dynamicKickIdealTargetV_L
.RoboEireann::Vector2f dynamicKickIdealTargetV_R
dynamicKickIdealTargetV_L.x = IDEAL_TARGET_POINT_X
dynamicKickIdealTargetV_L.y = IDEAL_TARGET_POINT_Y
dynamicKickIdealTargetV_R.x = IDEAL_TARGET_POINT_X
dynamicKickIdealTargetV_R.y = IDEAL_TARGET_POINT_Y


.bool USE_DYNAMIC_KICK = true #      <<<<<<<< _____________________________Mode Switch
if(USE_DYNAMIC_KICK)

   ##############################################
   ######### MISSILE TO ANALYTIC KICK ###########
   ##############################################

   for(; true;)

      output.motionSystemBehaviourOutputData.BALL_X                = input.ball.model.x
      output.motionSystemBehaviourOutputData.BALL_Y                = input.ball.model.y
      output.motionSystemBehaviourOutputData.BALL_ERROR_X          = dynamicKickTargetingError.x
      output.motionSystemBehaviourOutputData.BALL_ERROR_Y          = dynamicKickTargetingError.y
      output.motionSystemBehaviourOutputData.SHOT_ANGLE            = input.angleToGoal
      output.motionSystemBehaviourOutputData.TARGET_POINT_X_IDEAL  = dynamicKickIdealTargetV.x
      output.motionSystemBehaviourOutputData.TARGET_POINT_Y_IDEAL  = dynamicKickIdealTargetV.y
       output.motionSystemBehaviourOutputData.TARGET_BOX_X_EDGE_FAR     =     dynamicKickIdealTargetV.x     +
FARSIDE_BOX_EDGE_X
      output.motionSystemBehaviourOutputData.TARGET_BOX_X_EDGE_NEAR        =    dynamicKickIdealTargetV.x    -
NEAR_SIDE_BOX_EDGE_X
       output.motionSystemBehaviourOutputData.TARGET_BOX_Y_EDGE_INNER      =    dynamicKickIdealTargetV.y    -
INSIDE_BOX_EDGE_Y
       output.motionSystemBehaviourOutputData.TARGET_BOX_Y_EDGE_OUTTER   =   dynamicKickIdealTargetV.y   +
OUTSIDE_BOX_EDGE_Y

      ## steering control ##
      if(input.ball.model.x > LEG_DECISION_RANGE)
         ##### missile centering ball between feet #####
         correctionTheta = input.ball.model.angle()
      else##### missile centering ball lateraly on target point #####
         ballDistance = math::sqr((input.ball.model.x * input.ball.model.x)+(input.ball.model.y * input.ball.model.y)) # pythag
         thetaBallIdeal = math::arcSin(math::fabs(dynamicKickIdealTargetV.y) / ballDistance)
         thetaBallActual = math::atan2(input.ball.model.y , input.ball.model.x)
         correctionTheta = thetaBallIdeal - thetaBallActual
         ### set kick leg and target point
         if(input.ball.model.y > 0)
            dynamicKickLeg = false #Left
            output.motionSystemBehaviourOutputData.KICK_LEG = 0
            dynamicKickIdealTargetV.x = dynamicKickIdealTargetV_L.x
            dynamicKickIdealTargetV.y = dynamicKickIdealTargetV_L.y
         else
            dynamicKickLeg = true #Right
            output.motionSystemBehaviourOutputData.KICK_LEG = 1
            dynamicKickIdealTargetV.x = dynamicKickIdealTargetV_R.x
            dynamicKickIdealTargetV.y = dynamicKickIdealTargetV_R.y

      dynamicKickTargetingError.x = input.ball.model.x - dynamicKickIdealTargetV.x
      dynamicKickTargetingError.y = math::fabs(input.ball.model.y) - math::fabs(dynamicKickIdealTargetV.y)

      ##### BREAK IF INSIDE CAPTURE BOX ####
               if(  (dynamicKickTargetingError.x  <  FARSIDE_BOX_EDGE_X)  and  (dynamicKickTargetingError.x  >
-NEAR_SIDE_BOX_EDGE_X) and (input.ball.seen)  )
         X_Locked = true
      else
         X_Locked = false
               if(  (dynamicKickTargetingError.y  <  OUTSIDE_BOX_EDGE_Y)  and  (dynamicKickTargetingError.y  >
-INSIDE_BOX_EDGE_Y) and (input.ball.seen)  )
         Y_Locked = true
      else
         Y_Locked = false
      if(X_Locked and Y_Locked)
         break

      if(input.ball.model.abs() > 1.5 or (math::fabs(angleToTarget) > 1.0 and input.role != RoboEireann::Roles::GOALIE))
         yield TaskState::failure
      TRACES("MISSILE TO ANALYTIC KICK")

      ## forward motion control ##
            xSpeed = calculateVelocity(0.2, 0.8, 0.24, 0.55, input.ball.model.x) # minVelocity, maxVelocity, minDistance,
```

```
maxDistance, inputDistance

        ## strafing control ##
        #ySpeed = computeVelocityY(dynamicKickIdealTargetV.y - input.relativeGoalPos.y)   ## recompute this calc is wrong - use goal info to line up: goal - ball - robot

        ### OUTPUT ###
        motion::walkObstacle2(xSpeed, 0.0, -correctionTheta * 0.9)
        output.motionSystemBehaviourOutputData.control = RoboEireann::controlCommand::WALKING_TO_BALL
        headControl::lookAtBallModel()
        yield



   else
      ################################################
      ######## MISSILE TO EMPIRICAL KICK ##########
      ################################################
      for(; true;)

         if(kickOff)
            angleToTarget = input.angleToKickoffPoint
         else
            angleToTarget = input.angleToGoal

                              if(input.ball.model.x  <  missilePhaseTerminationDistanceX  and  math::fabs(input.ball.model.y)  <  missilePhaseTerminationDistanceY)
            break

         if(input.ball.model.abs() > 1 or (math::fabs(angleToTarget) > 1.0 and input.role != RoboEireann::Roles::GOALIE))
            yield TaskState::failure
         TRACES("MISSILE TO EMPIRICAL KICK")
               xSpeed  =  calculateVelocity(0.1,  1.0,  0.25,  0.55,  input.ball.model.x)  #  minVelocity,  maxVelocity,  minDistance,  maxDistance, inputDistance
            motion::walkObstacle2(xSpeed, 0.0, input.ball.model.angle() * input.kickParameters.missileRot) # missileRot is not used for penaltyShoot
         headControl::lookAtBallModel()
         yield




   ################################################
   ##### stop and look around before line up ####  performance seems fine without
   ################################################

   #.int startTime = input.time
   #for(; !kickOff and input.time - startTime < 2000; ) # remove
   #    headControl::lookAroundFast()
   #    motion::stand()
   #    yield




   ################################################
   ############### LINE UP PHASE ################
   ################################################
   if(kickOff)
      angleToTargetDeg = input.angleToKickoffPointDeg
   else
      angleToTargetDeg = input.angleToGoalDeg

   .float kickAngle = angleToTargetDeg
   .bool kickLeg = false # left is true, use right leg per default
   if(kickAngle < 0)
      kickLeg = true
      kickAngle = -kickAngle



        if(USE_DYNAMIC_KICK)
      ################################################
```

```
    ########## LINE UP ANALYTIC KICK ############
    ##############################################

      for(; true; )

      dynamicKickTargetingError.x = input.ball.model.x - dynamicKickIdealTargetV.x
      dynamicKickTargetingError.y = math::fabs(input.ball.model.y) - math::fabs(dynamicKickIdealTargetV.y)

      output.motionSystemBehaviourOutputData.BALL_X        = input.ball.model.x
      output.motionSystemBehaviourOutputData.BALL_Y        = input.ball.model.y
      output.motionSystemBehaviourOutputData.BALL_ERROR_X          = dynamicKickTargetingError.x
      output.motionSystemBehaviourOutputData.BALL_ERROR_Y          = dynamicKickTargetingError.y
      output.motionSystemBehaviourOutputData.SHOT_ANGLE            = input.angleToGoal
      output.motionSystemBehaviourOutputData.TARGET_POINT_X_IDEAL          = dynamicKickIdealTargetV.x
      output.motionSystemBehaviourOutputData.TARGET_POINT_Y_IDEAL          = dynamicKickIdealTargetV.y
      output.motionSystemBehaviourOutputData.TARGET_BOX_X_EDGE_FAR      =      dynamicKickIdealTargetV.x      +
FARSIDE_BOX_EDGE_X
      output.motionSystemBehaviourOutputData.TARGET_BOX_X_EDGE_NEAR   =      dynamicKickIdealTargetV.x      -
NEAR_SIDE_BOX_EDGE_X
      output.motionSystemBehaviourOutputData.TARGET_BOX_Y_EDGE_INNER  =      dynamicKickIdealTargetV.y      -
INSIDE_BOX_EDGE_Y
      output.motionSystemBehaviourOutputData.TARGET_BOX_Y_EDGE_OUTTER   =   dynamicKickIdealTargetV.y   +
OUTSIDE_BOX_EDGE_Y


       output.motionSystemBehaviourOutputData.control = RoboEireann::controlCommand::BALL_IN_RANGE
       output.motionRequest.motion = RoboEireann::Motion::motionSystem
       headControl::lookAtBallModel()
       yield


    ##############################################
    ########## LINE UP EMPIRICAL KICK ############
    ##############################################
  else


    #  kick line up phase parameters
      .float TARGET_POINT_ERROR_BOUND_X = 0.01    # note this is length from point .ie box dimension is twice this
value
    .float TARGET_POINT_ERROR_BOUND_Y = 0.01
      .float TARGET_POINT_OFFSET_X = 0.02          # this acounts for GDTP origin difference between initial pose and
walking is there is one
     .float TARGET_POINT_OFFSET_Y = 0.01          # or any other such similar error like residual sway. this will shift the
target point location
    .float HYSTERESIS_ABORT_THRESHOLD_X = 0.3
    .float HYSTERESIS_ABORT_THRESHOLD_Y = 0.3

    #.RoboEireann::Vector2f targetV = input.targetVectorGenerator.getTargetVector3(kickLeg, kickAngle) # improve with orig
function
    .RoboEireann::Vector2f targetV = input.targetVectorGenerator.getTargetVector(kickLeg, kickAngle)
    #targetV.x += TARGET_POINT_OFFSET_X



    #.float xError = 0.01
    #.float yError = 0.005
    .int startTimeForliningUpStop = input.time
    for(; true; )

      .RoboEireann::Vector2f error = input.ball.model - targetV
                        if(math::fabs(error.x)  <  TARGET_POINT_ERROR_BOUND_X  and  math::fabs(error.y)  <
TARGET_POINT_ERROR_BOUND_Y and input.time - input.ball.lastSeen < 500)
          break

                    if(input.ball.model.x  >  HYSTERESIS_ABORT_THRESHOLD_X  or  math::fabs(input.ball.model.y)  >
HYSTERESIS_ABORT_THRESHOLD_Y or input.time - input.ball.lastSeen > 3000 or(math::fabs(angleToTarget) > 1.0 and
input.role != RoboEireann::Roles::GOALIE))
          yield TaskState::failure

      if( input.time - startTimeForliningUpStop > input.liningUpStopTime)
         yield TaskState::failure

      TRACES("KICK LINE UP PHASE")
      TRACEF("Kickangle", kickAngle)
      TRACEF("Targetvector x: ", targetV.x)
      TRACEF("Targetvector y: ", targetV.y)

      if(math::fabs(error.x) > (TARGET_POINT_ERROR_BOUND_X - 0.003))
```

```
            xSpeed = computeVelocityX(error.x)
        else
            xSpeed = 0.0  ## dont do this, continue to correct until both are satisfied simultaneously

        if(math::fabs(error.y) > (TARGET_POINT_ERROR_BOUND_Y - 0.004))
            ySpeed = computeVelocityY(error.y)
        else
            ySpeed = 0.0

        motion::walkObstacle2(xSpeed, ySpeed, 0.0)
        headControl::lookAtBallModel()
        yield


    ##############################################
    ########## PERFORM EMPIRICAL KICK ############
    ##############################################
    if(USE_DYNAMIC_KICK)
        for(; true;)
            TRACES("PERFORM EMPIRICAL KICK")
            if(kickOff)
                motion::kick(kickLeg, kickAngle, 120, 0)
            else
                motion::kick(kickLeg, kickAngle, 55, 0)
            headControl::lookAt(input.relativeGoalPos)
            if(taskState(motion::kick) == TaskState::done)
                break
            yield




    ##############################################
    ############## HIGH LEVEL TASK ###############
    ##############################################

task gotoBallAndKick(bool kickOff)

    for(; true;) : 2
        gotoStartMissilePosition(kickOff)
        if(taskState(gotoStartMissilePosition) == TaskState::done)
            break
        if(taskState(gotoStartMissilePosition) == TaskState::failure)
            continue
        yield


    for(; true;)
        missileToKick(kickOff)
        if(taskState(missileToKick) == TaskState::done)
            break
        if(taskState(missileToKick) == TaskState::failure)
            yield TaskState::failure
        yield

    .int startTime = input.time
    for(; !kickOff and input.time - startTime < 2000;)
        motion::stand()
        headControl::lookAroundFast()
        #headControl::lookAtBallModel()
        yield
```

Appendix L - Full Body Inversekinematics Exact Solution for the Nao Robot

This solution is defined in the robots global coordinate system, not in its torso frame. In this coordinate system, the robots joints are found as a function of the 6 DOF (x, y, z, pitch – β, roll – γ, yaw - α) of each body part, the torso and both feet:

$$\begin{bmatrix} LeftHipYaw \\ LeftHipRoll \\ LeftHipPitch \\ LeftKnee \\ LeftAnklePitch \\ LeftAnkleRoll \\ RightHipYaw \\ RightHipRoll \\ RightHipPitch \\ RightKnee \\ RightAnklePitch \\ RightAnkleRoll \end{bmatrix} = f\left\{ \begin{bmatrix} Hp_X & Hp_Y & Hp_Z & Hp_\beta & Hp_\gamma & Hp_\alpha \\ Foot_{LX} & Foot_{LY} & Foot_{LZ} & Foot_{L\beta} & Foot_{L\gamma} & Foot_{L\alpha} \\ Foot_{RX} & Foot_{RY} & Foot_{RZ} & Foot_{R\beta} & Foot_{R\gamma} & Foot_{R\alpha} \end{bmatrix} \right\}$$

The algorithm is broken down into a few functions that are performed on each leg separately. They are as follows:

Function 1:

$$inversKinemalexCanNotResolve = mapTargetToHipFrame\left(V_T, Hp, \theta_{pelvicRoll}, \theta_{pelvicPitch}, leg\right) \quad (293)$$

To begin we must map the foot target points (V_T) from the robot global coordinate system into their respective hip frames (Figure 142). Before this is performed the torso attitude must be accounted for.

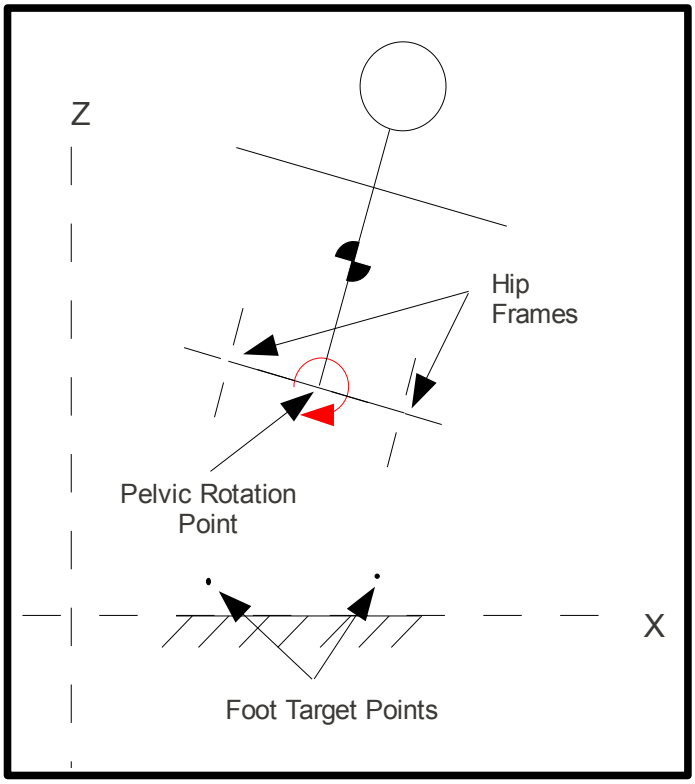


Figure 142: Inverse Kinematics Targets

However we want to to work with properly aligned frames in future calculations such as atan2() etc. The target points themselves will be manipulated relative to the location of the hips (left or right) in global coordinate frame first. This will keep the torso frame aligned with the global coordinate frame (Figure 143).

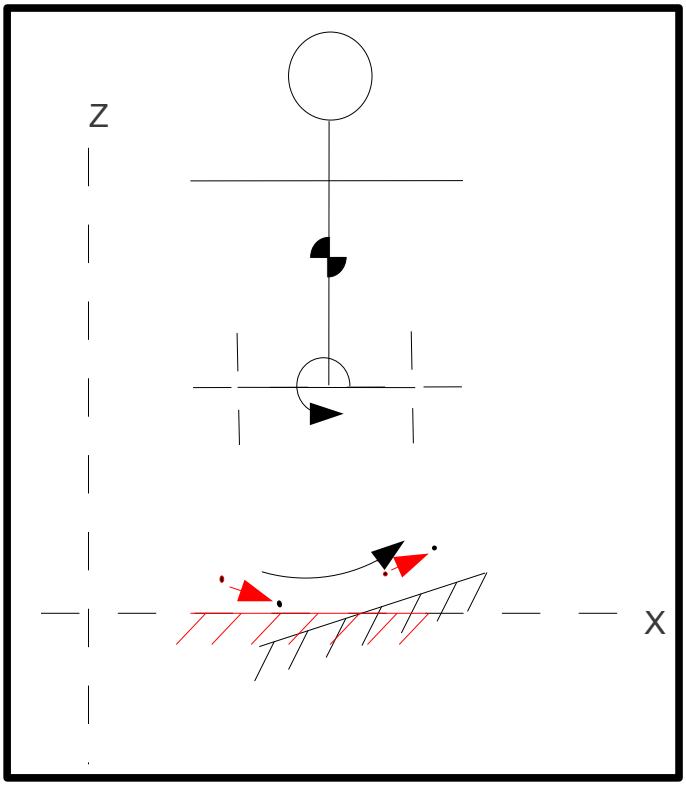


Figure 143: Torso Space Relative Motion

Solution:

Roll target points around pelvic roll in the robot global coordinate frame:

$$V_T = R_{\phi x}(-\theta_{pelvicRoll})(V_T - Hp) + Hp \tag{294}$$

Pitch target points around pelvic pitch (the order of these two operations can not be reversed):

$$V_T = R_{\phi y}(-\theta_{pelvicPitch})(V_T - Hp) + Hp \tag{295}$$

Compensated for left or right hip offset from groin:

$$Hp_y = \begin{Bmatrix} Hp_y + 5\,cm & : & leg & = & \text{left} \\ Hp_y - 5\,cm & : & leg & = & \text{right} \end{Bmatrix} \tag{296}$$

Map target points to Hip frame :

$$V_T = V_T - Hp \tag{297}$$

Check if unreachable and abort algorithm throwing error signal if so (length of leg can not be greater than 20 cm)

$$\ell_T = pythag(V_T) \tag{298}$$

$$inversKinemalexCanNotResolve = \begin{Bmatrix} true & : & \ell_T & >= & 20\,cm \\ false & : & \ell_T & < & 20\,cm \end{Bmatrix}$$
$$return(inversKinemalexCanNotResolve) \tag{299}$$

Function 2:

$$getKneeAngle(\theta_{KneePitch}, \theta_A, V_T, leg) \tag{300}$$

Once the target points are placed in the hip frames we begin by finding the knee angle that will allow the legs to hit their targets. The leg is then treated as one single vector.

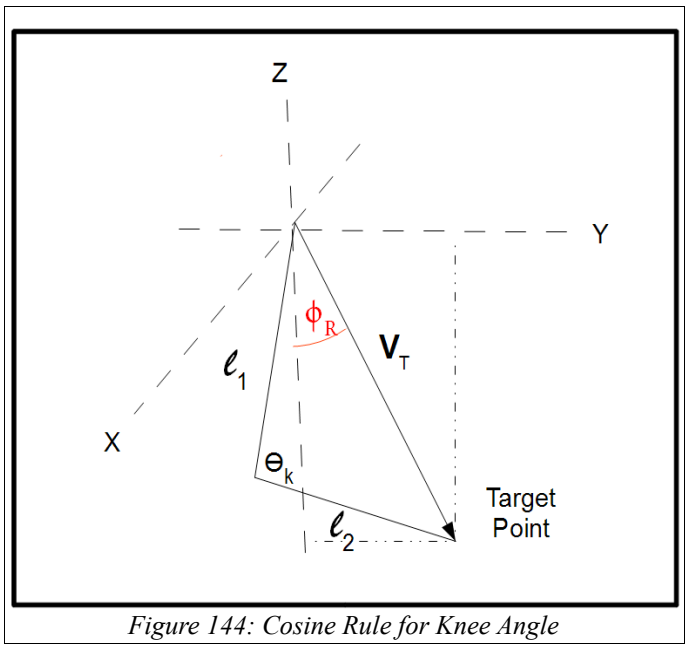


Figure 144: Cosine Rule for Knee Angle

Solution:

The leg at any orientation can be treated as a two dimensional triangle (Figure 144) as we know all sides and can therefore use the cosine rule. Note the knee angle is defined from the straight leg position so we take the supplement:

$$\begin{aligned} \ell_1 &= ThighLength = 10\,cm \\ \ell_2 &= TibiaLength = 10\,cm \end{aligned} \tag{301}$$

$$\theta_{KneePitch} = \pi - acos\left(\frac{\ell_1^2 + \ell_2^2 - \ell_T^2}{2\,\ell_1\,\ell_2}\right) \tag{302}$$

Achieving this position will also require some hip pitch. We can use the triangle to find this other interior angle and bank it for use later (θ_A). This allows us to treat the leg in its current unknown state as a vector hitting the target point as shown in Figure 145.

$$\theta_A = -acos\left(\frac{\ell_1^2 + \ell_T^2 - \ell_2^2}{2\,\ell_1\,\ell_T}\right) \tag{303}$$

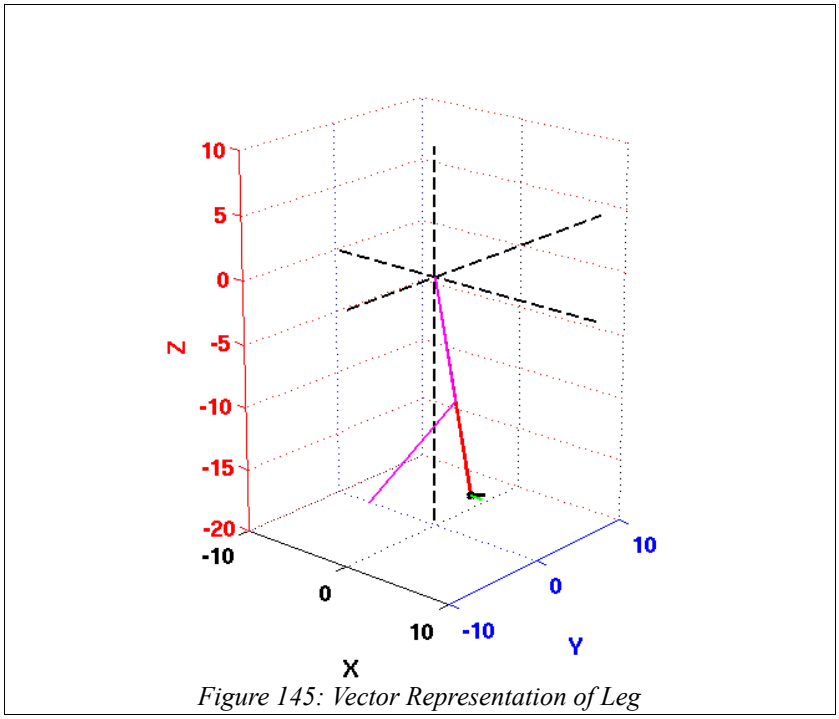


Figure 145: Vector Representation of Leg

Function 3:

$$getInitialHipPitchAndRoll(\theta_{HipPitchInitial}, \theta_{HipRollInitial}, V_T) \tag{304}$$

The next thing we can do is find a hip pitch and roll value that would allow this target vector to hit the target point with the assumption that there is no yaw orientation in the target pose (Figure 146). This is accomplished by projection on the the hip axis.

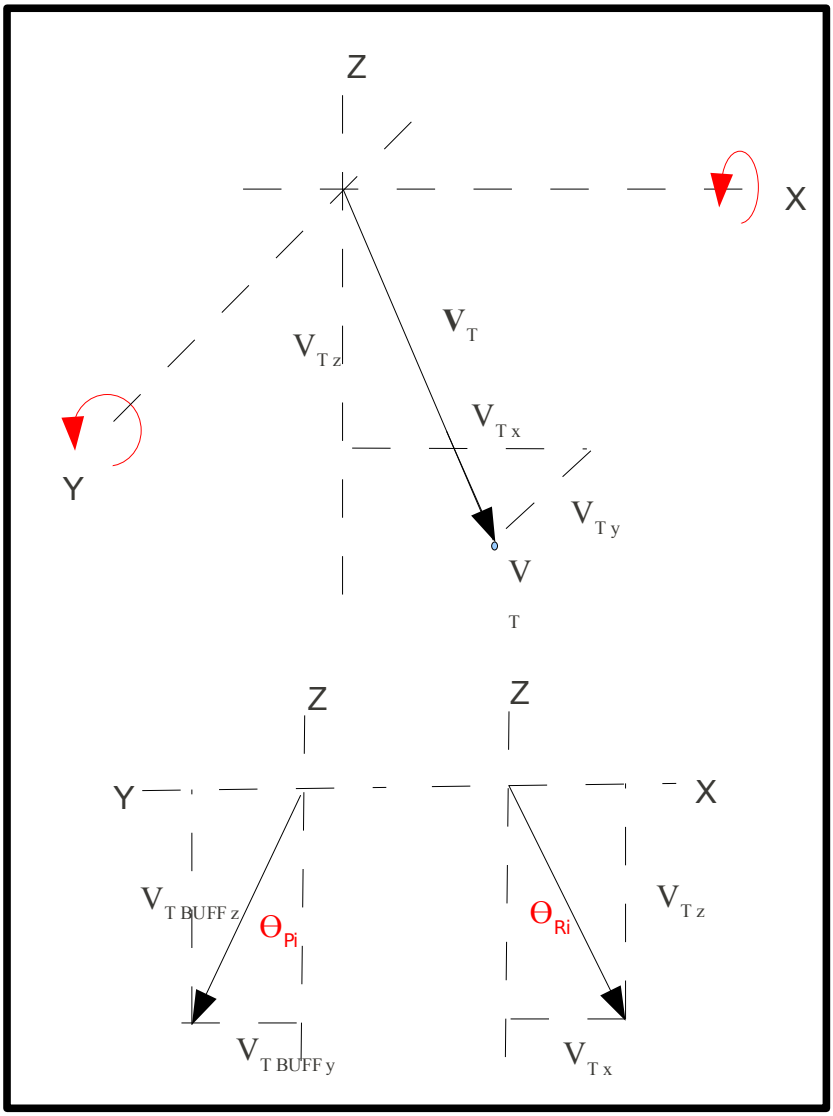


Figure 146: Initial Hip Pitch and Roll Projection

Solution:

The hip roll can be evaluated directly from its projection on the the Z-X plane:

$$\theta_{HipRollInitial} = atan2(V_{Ty}, -V_{Tz}) \tag{305}$$

We evaluate the vector without in it being obscured by the hip roll. Therefore the roll value is removed from the target vector and then evaluated for its pitch value against the Z-Y plane:

$$V_{T\,BUFF} = R_{\phi x}(-\theta_{HipRollInitial})\,V_T \tag{306}$$

$$\theta_{HipPitchInitial} = atan2(-V_{T\,BUFF\,x}, -V_{T\,BUFF\,z}) \tag{307}$$

Appendix L - Full Body Inversekinematics Exact Solution for the Nao Robot

Function 4:

$$getHipYawPitch(\theta_{HipYawPitch}, \theta_{aligment}, V_T, leg, \theta_{footOrientationDesired}, \theta_{pelvicRoll}, \theta_{pelvicPitch}) \quad (308)$$

Finding the HipYawPitch value is the hard part. To begin, the algorithm relies on two abstract properties of a two link manipulator.

First, given some initial foot orientation represented as a vector, there is field of vectors the vector belongs to. Any combination of hip pitch and roll values will move the the ankle to some new location and the equal but opposite value applied to the ankle joints will return the ankle orientation to the initial vector field. We can say then, hip pitches and rolls do not disturb the ankle yaw as we can correct for them.

Secondly, by only manipulating the hip pitch and roll, we can transcribe a whole sphere with the vector end point, 2 degrees of freedom is enough. The HipYawPitch axis lies at the same point as the pitch and roll axis and itself can transcribe a circle. Therefore, any manipulations made to the HipYawPitch joint will move the target vector to reach a different point, but we will be able to then correct for the translation with the hip roll and pitch as the circle lives on the sphere.

Putting these two facts together, we know we can set the orientation of the foot via the HipYawPitch joint and then correct for the unwanted ankle translation afterwards without messing up the foot yaw with the proper pitch and roll values for both the hips and the ankles.

To begin, we take the target vector with its existing pitch and roll and manipulate the it with the HipYawPitch to achieve the desired global coordinate foot yaw.

We will use the vector V_C to represent the current orientation, vector V_D to represent the desired orientation (yaw only at this point) and their corresponding angles Θ_D and Θ_C (Figure 147).

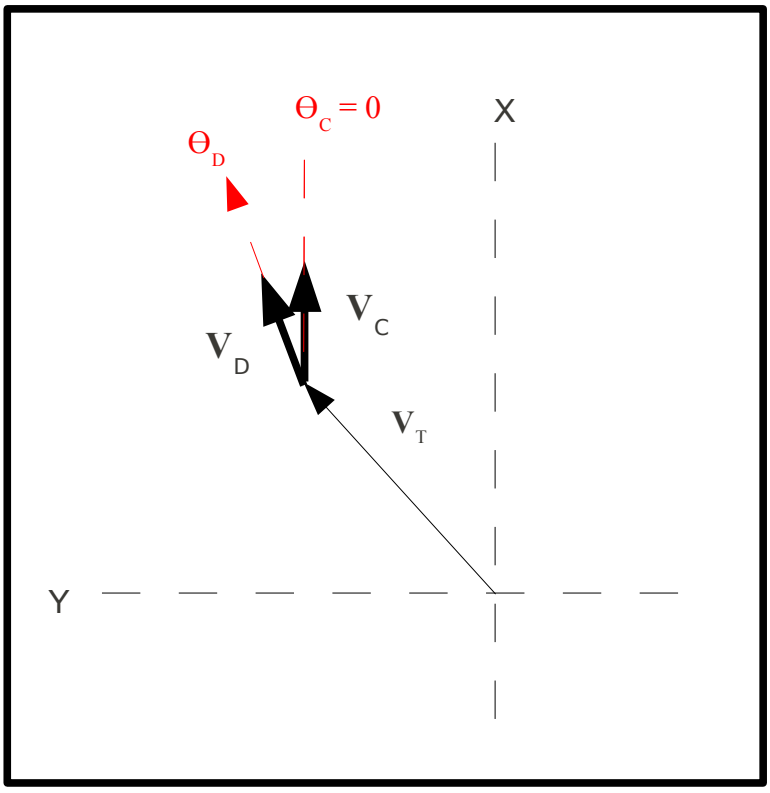


Figure 147: Desired and Current Foot Orientation Representation

Solution:

First define a unit vector:

$$U = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (309)$$

And perform the foot yaw to achieve the desired orientation:

$$V_D = R_{\phi z}(\theta_{footYaw},)U \tag{310}$$

The current orientation is defined as a unit vector point straight ahead as our leg is currently after applying some hip pitch and roll (the ankle joints are ignored as at they will be leveled independently in the end):

$$V_C = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{311}$$

Now, as with the target points, we have to compensate for the torso attitude in our desired orientation vector:

$$V_D = R_{\phi y}(-\theta_{pelvicPitch}) * R_{\phi x}(-\theta_{pelvicRoll}) * V_D \tag{312}$$

All three vectors can now be mapped into the confusing HipYawPitch frame that is mounted at a slanted angle (45 degrees to both the Z and Y axis when rotated about the X axis), Figure 148:

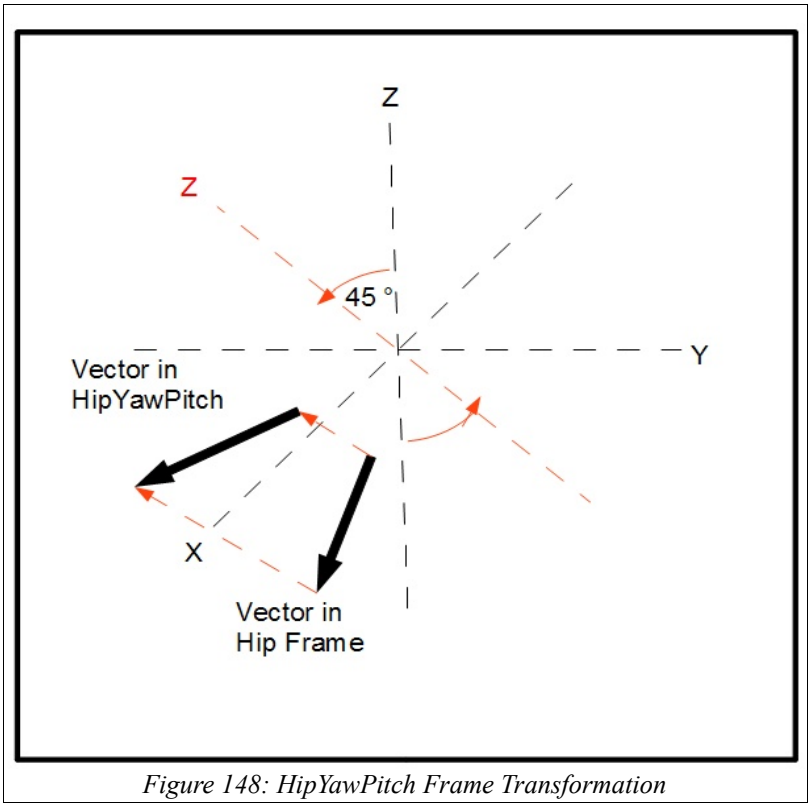


Figure 148: HipYawPitch Frame Transformation

Solution:

Each leg has the opposite frame roll:

$$\theta_{FrameRoll} = \begin{Bmatrix} \frac{\pi}{4} & : & leg & = & \text{left} \\ \frac{-\pi}{4} & : & leg & = & \text{right} \end{Bmatrix} \tag{313}$$

Roll all three vectors to put then in the HipYawPitch frame:

$$V_{D\,hyp} = R_{\phi x}(-\theta_{FrameRoll})\,V_D \quad (314)$$

$$V_{C\,hyp} = R_{\phi x}(-\theta_{FrameRoll})\,V_C \quad (315)$$

$$V_{T\,hyp} = R_{\phi x}(-\theta_{FrameRoll})\,V_T \quad (316)$$

Now conceptually what we want to do is manipulate the target vector (V_T) via the HipYawPitch such that the ankle is relocated to a new position (V_A) where the orientation of the attached V_C will be the same as V_D as shown in Figure 149. Unfortunately, simply using atan2 at this point will not work as the problem is a three dimensional one, not 2D.

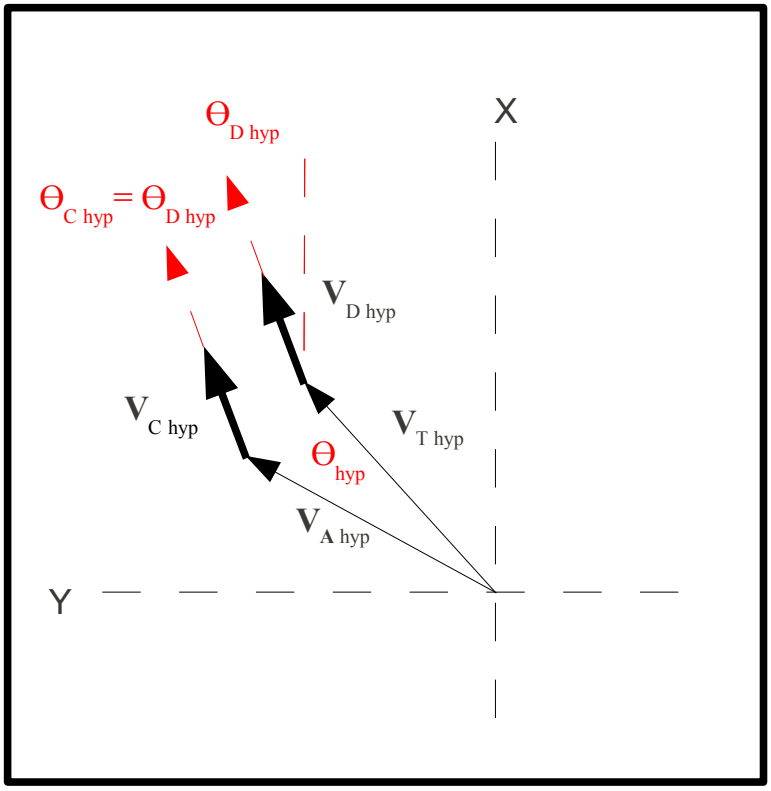


Figure 149: Foot Orientation Correction (abstract 2D analogy)

Our calculations however will be relative to the orientation of the target vector so we will have to temporarily align everything with the X and Y axis. Atan2 will satisfy this problem:

$$\theta_{aligment} = atan2(V_{T\,hyp\,x}, V_{T\,hyp\,y}) \quad (317)$$

$$V_{D\,hyp} = R_{\phi z}(-\theta_{aligment})\,V_{D\,hyp} \quad (318)$$

$$V_{C\,hyp} = R_{\phi z}(-\theta_{aligment})\,V_{C\,hyp} \quad (319)$$

$$V_{T\,hyp} = R_{\phi z}(-\theta_{aligment})\,V_{T\,hyp} \quad (320)$$

Plane alignment method to find HYP:

What we have thus far (seen in the following Figure 150) is a target vector representing the current position of the leg (red), a vector representing the current orientation of the foot (green) and a vector representing the desired orientation of the foot (mauve).

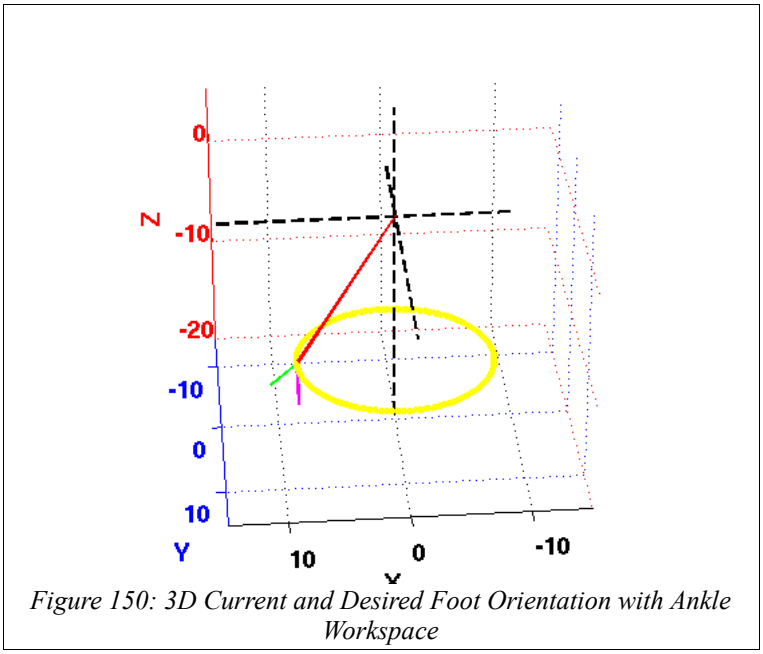


Figure 150: 3D Current and Desired Foot Orientation with Ankle Workspace

What we need to do is rotate the target vector around the Z axis such that the attached orientation vector is pointing in the same direction and the desired vector. Now the orientation information is relative to the target vector, meaning the resultant orientation vector after manipulation does not have to lay with the same vector field as the desired vector, but has to lie on the same plane described the target point vector and the desired orientation vector. It will however be different plane, one that is parallel. So, to find this other plane we can encode the orientation information as a normal (black line) to the first plane (shown in Figure 151) by finding the cross product:

$$N = V_{T\,hyp} \times V_{D\,hyp} \tag{321}$$

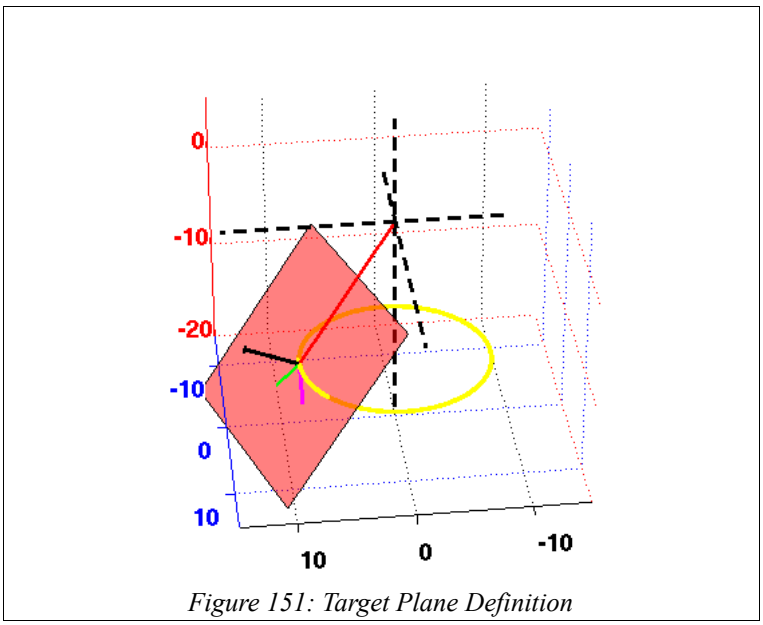


Figure 151: Target Plane Definition

As the normals to both planes will exist in the same vector field, the desired orientation will then exist in any plane described by this normal. We also know that any vector will exist on a plane if its inner product with the plane normal is zero. Therefore, if we take our current orientation vector and manipulate it somehow with HipYawPitch (here the Z axis), it will lie on some other plane that has the same normal as the first plane.

We know from our first plane the dot product equation:

$$N \cdot V_{Dhyp} = 0 \tag{322}$$

And that some manipulation of the current orientation(ϕ) will achieve the desired orientation:

$$\phi_{V_{Dhyp}} = R_{\phi z}(\theta_{HYP})\phi_{V_{C\,hyp}} \tag{323}$$

If we correctly manipulate the current orientation vector, the following equation is also true:

$$N \cdot (R_{\phi z}(\theta_{HYP}) V_{C\,hyp}) = 0 \tag{324}$$

This equation is then solved for the angle θ_{HYP}.

Finally, this Z axis rotation value that transforms a vector in this frame will transform any vector in this frame in the same way. So, by rotating the target vector we get the new location of the ankle(V_A)(blue) as it lies on the second plane (Figure 152).

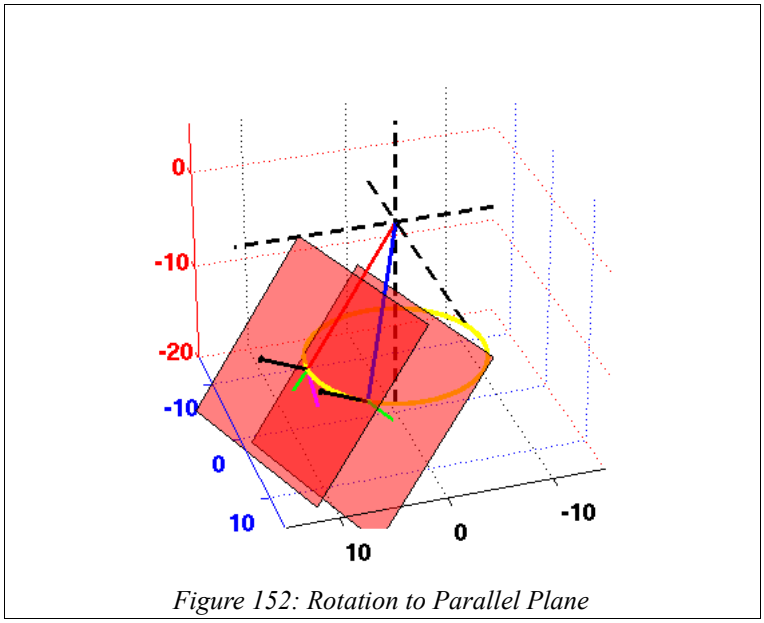


Figure 152: Rotation to Parallel Plane

Solution:

$$\begin{aligned} V1 &= V_{T\,hyp} \\ V2 &= V_{Dhyp} \\ V3 &= V_{C\,hyp} \end{aligned} \tag{325}$$

Normal to plane described by target vector and the desired orientation vector (cross product):

$$N = \begin{bmatrix} (V1_y \times V2_z - V1_z \times V2_y) \\ (V1_z \times V2_x - V1_x \times V2_z) \\ (V1_x \times V2_y - V1_y \times V2_x) \end{bmatrix} \tag{326}$$

Solution to equation (324) for theta:

$$\theta_{HYP} = acos\left(\frac{N_x V1_x + N_y V1_y + N_z V1_z - N_z V3_z}{\sqrt{(N_x V3_x + N_y V3_y)^2 + (N_x V3_y - N_y V3_x)^2}}\right) - \text{atan2}\left|(N_x V3_y - N_y V3_x),(N_x V3_x + N_y V3_y)\right| \tag{327}$$

$$\theta_{HipYawPitch} = \theta_{HYP} \tag{328}$$

Function 5:

$$setAverageHipYawPitch(\theta_{\text{HipYawPitch L}}, \theta_{\text{HipYawPitch R}}) \tag{329}$$

If the HipYawPitch left and right values are not the same:

$$if\ (|\theta_{\text{HipYawPitch L}} - \theta_{\text{HipYawPitch R}}| > 0.001) \tag{330}$$

Then set the average value:

$$sign = \begin{cases} 1 & : \ \theta_{\text{HipYawPitch L}} >= 0 \\ -1 & : \ \theta_{\text{HipYawPitch L}} < 0 \end{cases} \tag{331}$$

$$\theta_{\text{HipYawPitch L}} = \frac{1}{2} sign(|\theta_{\text{HipYawPitch L}}| + |\theta_{\text{HipYawPitch R}}|) \tag{332}$$

$$\theta_{\text{HipYawPitch R}} = -\theta_{\text{HipYawPitch L}} \tag{333}$$

Function 6:

$$RealignHipFrame(V_T, V_A, \theta_{HipYawPitch}, V_{T\,hyp}, \theta_{aligment}, leg) \tag{334}$$

Now that the HipYawPitch has been determined, we can reposition the target vector to the newly manipulated location, the ankle vector(V_A).

$$AnkleVector_{hyp} = R_{\phi z}(\theta_{HYP}) V_{T\,hyp} \tag{335}$$

Remove x axis alignment we performed earlier:

$$V_{A\,hyp} = R_{\phi z}(\theta_{aligment}) V_{A\,hyp} \tag{336}$$

$$V_{T\,hyp} = R_{\phi z}(\theta_{aligment}) V_{T\,hyp} \tag{337}$$

Finally, we want to return the vectors to the Hip Pitch and Roll frame. However, before doing so we must account for the fact that the act of manipulating the HipYawPitch motor changes the orientation of the other leg joint axes as they are physically mounted on top of the HipYawPitch in a serial chain. We know though the Hip Pitch axis will always be pointing in the same direction as the foot orientation. Prior to rolling the vectors back into the Hip frame we have to rotate the hip frame according to our found HipYawPitch angle (Figure 153).

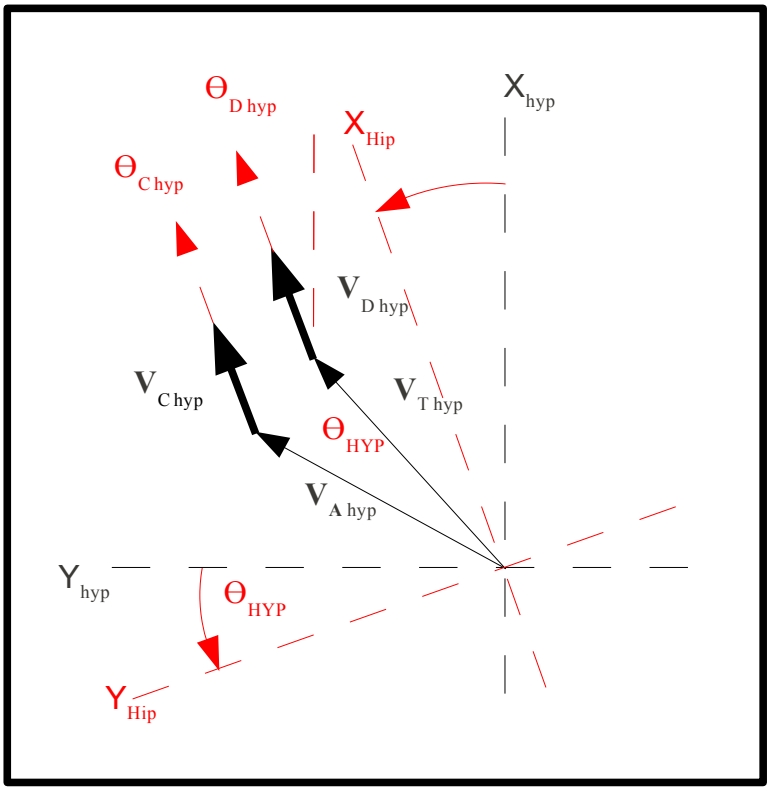


Figure 153: Hip Frame Relocation

Solution:

$$\theta_{FrameRoll} = \begin{cases} \frac{\pi}{4} & : \; leg = \text{left} \\ \frac{-\pi}{4} & : \; leg = \text{right} \end{cases} \tag{338}$$

$$V_T = R_{\phi x}(\theta_{FrameRoll}) R_{\phi z}(-\theta_{HYP}) V_{T\,hyp} \tag{339}$$

$$V_A = R_{\phi x}(\theta_{FrameRoll}) R_{\phi z}(-\theta_{HYP}) V_{A\,hyp} \tag{340}$$

After performing all of these manipulations we have the orientation of the foot correctly in the global coordinate system, but the target point has moved to a new position inside the hip frame that was reorientated when we moved the HipYawPitch joint. We have manipulated both the ankle position and the target position in the algorithm and have the result shown in Figure 154.

To summarize we have transformed the target vector ($T_{V\,HIP}$)in the Hip frame to become the new ankle position($V_{A\,hyp}$) in the HipYawPitch frame while aligning the attached orientations vectors V_D and V_C. We then put the these vectors back into the Hip frame, however it is not the same Hip frame that we began with. We can see two thing have happened in Figure 154; The location of the current ankle position ($V_{A\,HIP}$ red) now occupies the former location of the target point ($V_{T\,HIP}$ black), and that the target point itself has shifted to a new location($V_{T\,HIP}$ red).

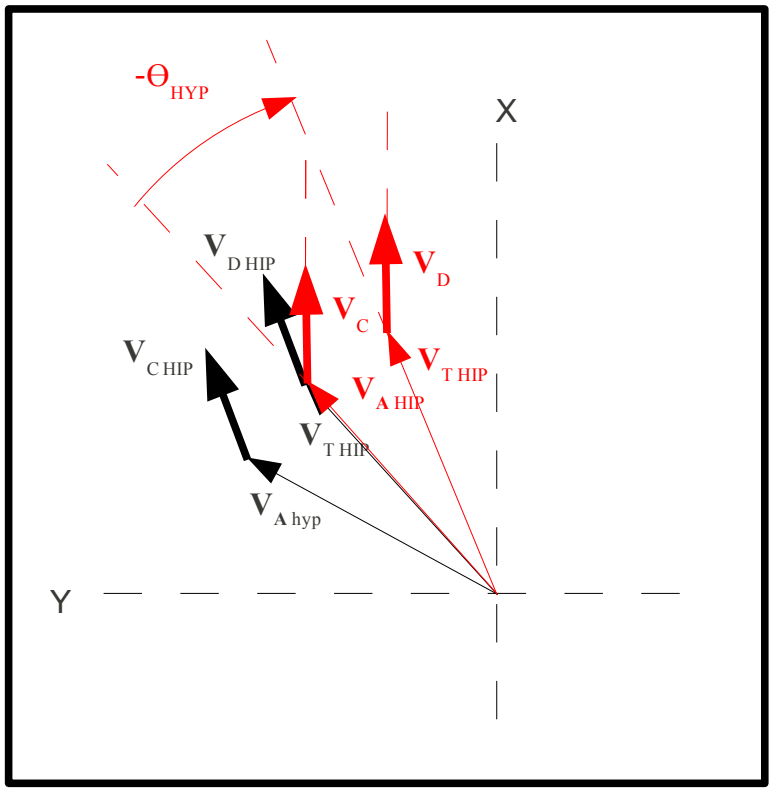


Figure 154: Post HYP Manipulation Target and Ankle Relocation

So now we just need to find the pitch and roll difference between the two final vectors.

Function 7:

$$getFinalHipPitchAndRoll(\theta_{HipPitch}, \theta_{HipRoll}, \theta_{HipPitchInitial}, \theta_{HipRollInitial}, \theta_{A}, V_{T}, AnkleVector) \quad (341)$$

The final values for the Hip Pitch and Roll motors can not be found by taking the absolute angles of the current vectors as the current Hip frame is not the hip frame that we began with. However, we did record the initial Hip Pitch and Roll values. We just need to find the difference between the current ankle position and the new target point and add them the the initial values.(Figure 155). We will also add the Hip pitch angle we banked earlier (θ_A) when we represented the leg triangle as a single vector.

Appendix L - Full Body Inversekinematics Exact Solution for the Nao Robot

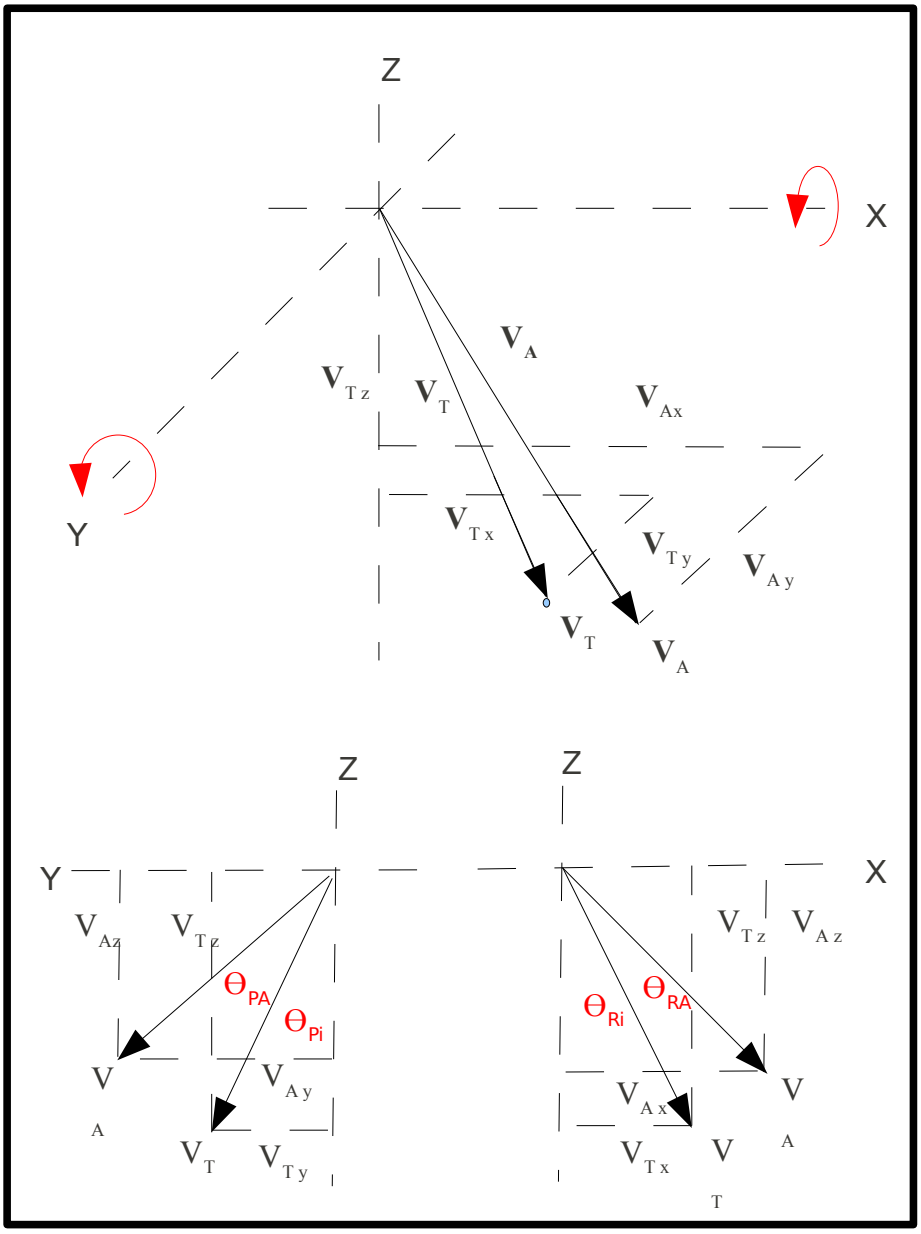


Figure 155: Additional Hip Pitch and Roll Projections

Solution:

First we must find the roll angles and evaluate the difference between them:

$$\theta_{targetVectorsRollValue} = atan2(V_{Ty}, -V_{Tz}) \tag{342}$$

$$\theta_{ankleVectorsRollValue} = atan2(V_{Ay}, -V_{Az}) \tag{343}$$

$$\theta_{HipRollDifference} = \theta_{targetVectorsRollValue} - \theta_{ankleVectorsRollValue} \tag{344}$$

Then given the joints are mounted on top of each other, we must perform this roll operation on the vectors to be able to measure them properly in the pitch axis:

$$V_T = R_{\phi x}(-\theta_{targetVectorsRollValue})\, V_T \tag{345}$$

$$V_A = R_{\phi x}(-\theta_{ankleVectorsRollValue})\, V_A \tag{346}$$

Then again evaluate the difference:

$$\theta_1 = atan2(-V_{Tx}, -V_{Tz}) \tag{347}$$

$$\theta_2 = atan2(-V_{Ax}, -V_{Az}) \tag{348}$$

$$\theta_{HipPitchDifference} = \theta_1 - \theta_2 \tag{349}$$

The final hip joint values are then sum of all past operations:

$$\theta_{HipPitch} = \theta_{HipPitchInitial} + \theta_{HipPitchDifference} + \theta_A \tag{350}$$

$$\theta_{HipRoll} = \theta_{HipRollInitial} + \theta_{HipRollDifference} \tag{351}$$

Function 8:

$$levelAnkle(\theta_{AnklePitch}, \theta_{AnkleRoll}, \theta_{HipYawPitch}, \theta_{KneePitch}, \theta_{HipPitch}, \theta_{HipRoll}, \theta_{pelvicRoll}, \theta_{pelvicPitch}, \theta_{AnklePitchTarget}, \theta_{AnkleRollDesired}, leg) \tag{352}$$

Finally, we level the ankle using a modified Euler's Method. This method is defined as:

"Euler angles are a means of representing the spatial orientation of any frame (coordinate system) as a composition of rotations from a frame of reference (coordinate system)."

--- http://en.wikipedia.org/wiki/Euler_angles

Essentially, what this method does is allow us to manipulate incrementally one frame so that it is aligned with a target frame without having to make use of some other known coordinate system. One frame is defined as a series of rotations with respect to another. This is accomplished by generating intermediate frames as we manipulate the ankle, in this case, one rotation at a time.

The key to this process is to create a vector known as the 'line of nodes'. This line of nodes is the intersection of planes transcribed by each degree of freedom. For example, if you were to roll (rotation about the x axis) any Cartesean frame, the Z and Y axis would be free to move around on plane that is always orthogonal to the axis of rotation. Now, a pitch is a rotation around the Y axis. Therefore, when manipulated, its Z and X axis describe some other plane. As we can see then, the Z axis is free to move along both planes. However, if the Z axis lies somewhere on the first plane, it would first have to be manipulated (rolled in this example) up to the position where both planes intersect before it can then move through the second plane (pitch). If our abstract target frame was the global coordinate itself, then this line of nodes would be the global X axis. What the Euler method allows us to do is go directly from the current orientation to the target orientation without reference to the global coordinate system.

This method must be tailored to fit the specification of a particular application. In our case the order in which the ankle motors are mechanically mounted is significant. This applied method is shown in Figure 156, manipulating the ankle's current orientation after all the other leg operations have been performed to achieve some pitch and roll values. This is somewhat abstract so a better picture (acquired from Wikipedia) is shown in Figure 157 demonstrating the concept. This figure represents a yaw roll yaw (Z-X-Z) rotation.

Appendix L - Full Body Inversekinematics Exact Solution for the Nao Robot

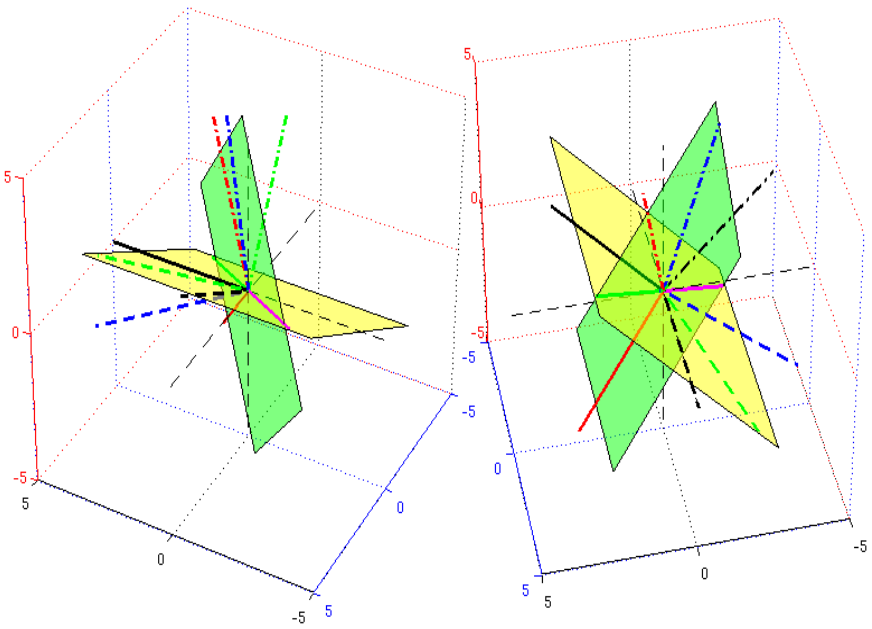


Figure 156: Eulars Method Applied To Both Ankles

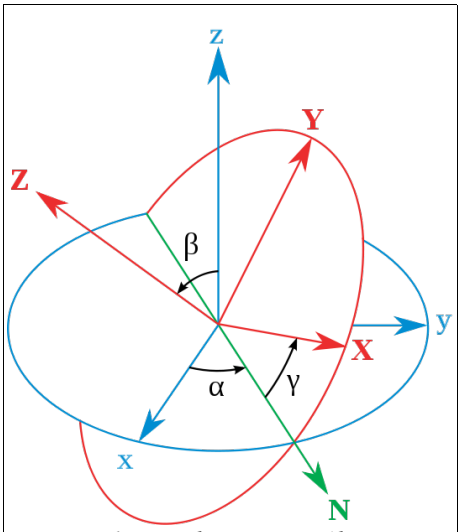


Figure 157: Eulars Frame Alignment Method

source ---- http://en.wikipedia.org/wiki/File:Gimbaleuler2.svg

First create the target frame built with unit vector to represent each axis:

$$TargetAxisX = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$TargetAxisY = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad (353)$$

$$TargetAxisZ = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Then orientate this frame into the target position as defined in the global coordinate system:

$$\begin{aligned} TargetAxisX &= R_{\phi x}(\theta_{AnkleRollDesired})\, R_{\phi y}(\theta_{AnklePitchTarget})\, TargetAxisX \\ TargetAxisY &= R_{\phi x}(\theta_{AnkleRollDesired})\, R_{\phi y}(\theta_{AnklePitchTarget})\, TargetAxisY \\ TargetAxisZ &= R_{\phi x}(\theta_{AnkleRollDesired})\, R_{\phi y}(\theta_{AnklePitchTarget})\, TargetAxisZ \end{aligned} \tag{354}$$

Then do the same for the current ankle orientation after the other leg joints have acted upon it:

$$\begin{aligned} AnkleAxisX &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ AnkleAxisY &= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ AnkleAxisZ &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \tag{355}$$

$$R_{\phi yz} = \begin{Bmatrix} R_{\phi yz\,L}(\theta_{HipYawPitch}) & : & leg & = & \text{left} \\ R_{\phi yz\,R}(\theta_{HipYawPitch}) & : & leg & = & \text{right} \end{Bmatrix} \tag{356}$$

$$\begin{aligned} AnkleAxisZ &= R_{\phi x}(\theta_{pelvicRoll})\, R_{\phi y}(\theta_{pelvicPitch})\, R_{\phi yz}\, R_{\phi x}(\theta_{HipRoll})\, R_{\phi y}(\theta_{HipPitch})\, R_{\phi y}(\theta_{KneePitch})\, AnkleAxisZ \\ AnkleAxisY &= R_{\phi x}(\theta_{pelvicRoll})\, R_{\phi y}(\theta_{pelvicPitch})\, R_{\phi yz}\, R_{\phi x}(\theta_{HipRoll})\, R_{\phi y}(\theta_{HipPitch})\, R_{\phi y}(\theta_{KneePitch})\, AnkleAxisY \\ AnkleAxisX &= R_{\phi x}(\theta_{pelvicRoll})\, R_{\phi y}(\theta_{pelvicPitch})\, R_{\phi yz}\, R_{\phi x}(\theta_{HipRoll})\, R_{\phi y}(\theta_{HipPitch})\, R_{\phi y}(\theta_{KneePitch})\, AnkleAxisX \end{aligned} \tag{357}$$

The line of nodes will be orthogonal to the normal of both planes. First find the normals to the planes described by the axes that are free to move around the axis of rotation (cross product). Then again take the cross product of both normals.

$$N1 = \begin{bmatrix} ((AnkleAxisX_y \times AnkleAxisZ_z) - (AnkleAxisX_z \times AnkleAxisZ_y)) \\ ((AnkleAxisX_z \times AnkleAxisZ_x) - (AnkleAxisX_x \times AnkleAxisZ_z)) \\ ((AnkleAxisX_x \times AnkleAxisZ_y) - (AnkleAxisX_y \times AnkleAxisZ_x)) \end{bmatrix} \tag{358}$$

$$N2 = \begin{bmatrix} ((TargetAxisX_y \times TargetAxisY_z) - (TargetAxisX_z \times TargetAxisY_y)) \\ ((TargetAxisX_z \times TargetAxisY_x) - (TargetAxisX_x \times TargetAxisY_z)) \\ ((TargetAxisX_x \times TargetAxisY_y) - (TargetAxisX_y \times TargetAxisY_x)) \end{bmatrix} \tag{359}$$

$$planeIntersection = \begin{bmatrix} (N1_y \times N2_z) - (N1_z \times N2_y) \\ (N1_z \times N2_x) - (N1_x \times N2_z) \\ (N1_x \times N2_y) - (N1_y \times N2_x) \end{bmatrix} \tag{360}$$

The normal calculated this way will give us the inverse line of nodes so we just flip it over:

$$lineOfNodes = -planeIntersection \tag{361}$$

Now the pitch value can be determined. We take the angle between the X axis (we could use Z axis too) and the line of nodes. The angle can be found by building a triangle in 3 dimensions and using the cosine rule. Note this is what makes the algorithm user friendly, as trying to apply atan2() here would be cumbersome.

$$\begin{aligned} a &= AnkleAxisX \\ b &= lineOfNodes \\ c &= lineOfNodes - AnkleAxisX \end{aligned} \tag{362}$$

$$\begin{aligned} la &= \sqrt{(a_x^2 + a_y^2 + a_z^2)} \\ lb &= \sqrt{(b_x^2 + b_y^2 + b_z^2)} \\ lc &= \sqrt{(c_x^2 + c_y^2 + c_z^2)} \end{aligned} \tag{363}$$

As there are two directions to move in, we need to evaluate which direction the target is in relative to current position. As the height of the axis endpoint will change, we can simply check which is greater and use that as an indicator if we need to pitch up(-ve) or down(+ve).

$$\theta_{AnklePitch}=\begin{Bmatrix} -acos(\frac{(la^2+lb^2-lc^2)}{(2\ la\ lb)}) & : & TargetAxisX_z & > & AnkleAxisX_z \\ acos(\frac{(la^2+lb^2-lc^2)}{(2\ la\ lb)}) & : & TargetAxisX_z & < & AnkleAxisX_z \\ 0 & : & TargetAxisX_z & = & AnkleAxisX_z \end{Bmatrix} \quad (364)$$

The ankle frame can now be manipulated into its intermediary position with its axes located on the roll plane. However this time we will use the rotation matrix that allows us to rotate around any arbitrary axis of rotation. This axis of rotation is given to the matrix function as an input vector, here the AnkleAxisY:

$$\begin{aligned} partialManipulationFrameX &= R_{\phi ARB}(AnkleAxisY, \theta_{AnklePitch})\ AnkleAxisX \\ partialManipulationFrameY &= R_{\phi ARB}(AnkleAxisY, \theta_{AnklePitch})\ AnkleAxisY \\ partialManipulationFrameZ &= R_{\phi ARB}(AnkleAxisY, \theta_{AnklePitch})\ AnkleAxisZ \end{aligned} \quad (365)$$

The roll value is found in the same way, build the triangle between the intermediary axis and target:

$$\begin{aligned} a &= TargetAxisZ \\ b &= partialManipulationPitchedFrameZ \\ c &= TargetAxisZ - partialManipulationFrameZ \end{aligned} \quad (366)$$

$$\begin{aligned} la &= \sqrt{(a_x^2+a_y^2+a_z^2)} \\ lb &= \sqrt{(b_x^2+b_y^2+b_z^2)} \\ lc &= \sqrt{(c_x^2+c_y^2+c_z^2)} \end{aligned} \quad (367)$$

Finally the roll angle can be found:

$$\theta_{AnkleRoll}=\begin{Bmatrix} -acos(\frac{la^2+lb^2-lc^2}{2\ la\ lb}) & : & TargetAxisZ_y & > & partialManipulationFrameZ_y \\ acos(\frac{la^2+lb^2-lc^2}{2\ la\ lb}) & : & TargetAxisZ_y & < & partialManipulationFrameZ_y \\ 0 & : & TargetAxisZ_y & = & partialManipulationFrameZ_y \end{Bmatrix} \quad (368)$$

A useful final check can be to actually perform this final roll operation on the ankle frame and then subtract it from the target frame. The result should be zero.

$$\begin{aligned} finalAnkleAxisX &= R_{\phi ARB}(lineOfNodes, \theta_{AnkleRoll})\ partialManipulationFrameX \\ finalAnkleAxisY &= R_{\phi ARB}(lineOfNodes, \theta_{AnkleRoll})\ partialManipulationFrameY \\ finalAnkleAxisZ &= R_{\phi ARB}(lineOfNodes, \theta_{AnkleRoll})\ partialManipulationFrameZ \end{aligned} \quad (369)$$

$$if\ |(|TargetAxisX - finalAnkleAxisX|)\ \text{or}\ (|TargetAxisY - finalAnkleAxisY|)\ \text{or}\ (|TargetAxisZ - finalAnkleAxisZ|)|\ \{\ \text{Throw Error}\ \} \quad (370)$$

Conclusion:

In summary, the algorithm is run by calling each of these defined functions one at a time and operating on each independent foot as follows:

$$\begin{aligned} &inversKinemalexCanNotResolve = mapTargetToHipFrame(V_{TL}, Hp, \theta_{pelvicRoll}, \theta_{pelvicPitch}, 'l') \\ &inversKinemalexCanNotResolve = mapTargetToHipFrame(V_{TR}, Hp, \theta_{pelvicRoll}, \theta_{pelvicPitch}, 'r') \\ &if\ (inversKinemalexCanNotResolve = true)\ \{\ Abort\ Algorithm\ \} \\ &getKneeAngle(\theta_{KneePitchL}, \theta_{AL}, V_{TL}, 'l') \end{aligned} \quad (371)$$

$$getKneeAngle(\theta_{KneePitch\,R}, \theta_{AR}, V_{T\,R}, 'r')$$
$$getInitialHipPitchAndRoll(\theta_{HipPitchInitialL}, \theta_{HipRollInitialL}, V_{T\,L})$$
$$getInitialHipPitchAndRoll(\theta_{HipPitchInitialR}, \theta_{HipRollInitialR}, V_{T\,R})$$
$$getHipYawPitch(\theta_{HipYawPitchL}, V_{\text{T hyp L}}, \theta_{\text{aligment L,}} V_{T\,L}, 'l', \theta_{footYaw\,L}, \theta_{pelvicRoll}, \theta_{pelvicPitch})$$
$$getHipYawPitch(\theta_{HipYawPitchR}, V_{\text{T hyp R}}, \theta_{\text{aligment R,}} V_{T\,R}, 'r', \theta_{footYaw\,R}, \theta_{pelvicRoll}, \theta_{pelvicPitch})$$

At this point there are some choices can be made. Either to split the error across both feet of perhaps keep the support foot as desired and all the error can go into the other during single support.

$$setAverageHipYawPitch(\theta_{\text{HipYawPitch L}}, \theta_{\text{HipYawPitch R}})$$

$$RealignHipFrame(V_{T\,L}, V_{A\,L}, \theta_{HipYawPitchL}, V_{T\,hyp\,L}, \theta_{aligmentL}, 'l')$$
$$RealignHipFrame(V_{T\,R}, V_{AR}, \theta_{HipYawPitchR}, V_{T\,hyp\,R}, \theta_{aligmentR}, 'r')$$
$$getFinalHipPitchAndRoll(\theta_{HipPitchL}, \theta_{HipRollL}, \theta_{HipPitchInitialL}, \theta_{HipRollInitialL}, \theta_{AL}, V_{T\,L}, V_{A\,L})$$
$$getFinalHipPitchAndRoll(\theta_{HipPitchR}, \theta_{HipRollR}, \theta_{HipPitchInitialR}, \theta_{HipRollInitialR}, \theta_{AR}, V_{T\,R}, V_{A\,R})$$
$$levelAnkle(\theta_{AnklePitchL}, \theta_{AnkleRollL}, \theta_{HipYawPitchL}, \theta_{KneePitchL}, \theta_{HipPitchL}, \theta_{HipRollL}, \theta_{pelvicRoll}, \theta_{pelvicPitch}, \theta_{AnklePitchL}, \theta_{AnkleRollL}, 'l')$$
$$levelAnkle(\theta_{AnklePitchR}, \theta_{AnkleRollR}, \theta_{HipYawPitchR}, \theta_{KneePitchR}, \theta_{HipPitchR}, \theta_{HipRollR}, \theta_{pelvicRoll}, \theta_{pelvicPitch}, \theta_{AnklePitchR}, \theta_{AnkleRollR}, 'r')$$

This algorithms output can be seen simulated in Figure 158 and 159 with the following parameters in Table 10. The different perspectives demonstrate the foot pitch and roll is in the global coordinate system.

Body Pitch	30°
Body Roll	30°
Foot orientation	45°
L Foot Pitch	-45°
L Foot Roll	0°
R Foot Pitch	0°
R Foot Roll	30°

Table 10: Sample of Inversekinematic Test Parameters

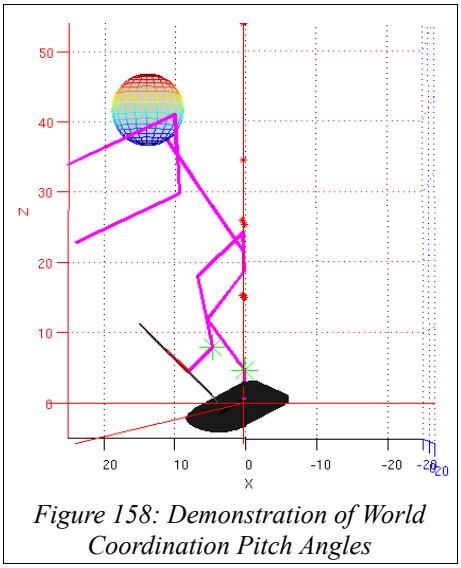


Figure 158: Demonstration of World Coordination Pitch Angles

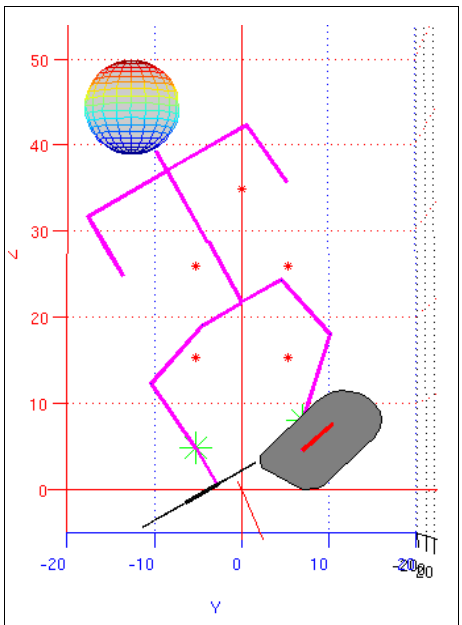


Figure 159: Demonstration of World Coordination Roll Angles

The result was that the torso orientation was achieved as with the feet pitch and roll values, however all things can not be satisfied so the final foot yaw values in the global coordinate frame were:

Left Foot Yaw: 35.05° and Right Foot Yaw: -58.81°

As we can see error was not split evenly with the target value being 45°. This is a function of splitting the difference in the HipYawPitch frame, where again, all angles in the Hip frame are obscured.

Appendix M - Kick Targeting Systems and Goal Scoring in the RoboCup SPL (Nao)

1.1 Introduction

The process of kicking a ball is one of the most vital components of a RoboCup soccer system. There are teams in the league that perform well in many respects but falter at the kicking stage and this directly impacts the team's competitiveness. It is actually true that this is more of problem among teams than it is not. Overall, my opinion based on observations over four years is that this aspect of team performance at the competition frequently leaves a lot to be desired. It's possible the cause of difficulties here is that the work required in developing a kick system in terms of time and effort is underestimated. A lot of work has gone into the RoboEireann kicking system and that is what this chapter will cover.

The act of kicking the ball in a RoboCup soccer system is the one component that relies on all other system components being functional as it has dependencies on all of them. Vision, localization (be it primitive vision based or world modelling), locomotion and behaviour. Consequently the performance, accuracy, stability and signal quality of each of these systems will have an impact on the performance of the kicking system. There are differences in the design and nature of each of these other components, therefore, the kicking system may have to be designed differently to compensate for such differences. For example, navigation using a world modelling localization system or in its absence a visual only navigation method, continually checking for landmarks, would impact how the striker behaviour must be designed and the scale of the final error that would have to be tolerated. Noisy vision data or walking errors (one that veers to one side, command delays, omnidirectional) become issues that must be accounted for.

Kicks can be categorized as two kinds; (i) a static pose based design, much like animation, that is designed in the laboratory or a simulator that we can call 'empirically defined kick', (ii) an inverse kinematics swing trajectory design that we will call 'analytically defined'. The empirically defined kick appears to be the most common in the standard platform league. This is apparent through observing the number of teams that spend time shuffling around the ball during the final line up stage. An analytical kick could be characterized as being more dynamic and would be able to compensate for slight differences in ball placement, of the order of 5 centimetres. Most teams up to and including the 2011 competition do some lateral shuffling and consequently it is reasonable to assume they are trying to line up a static animated kick swing.

These kicks will constitute a termination point in the behaviour code, since once reached, the whole system resets to initial conditions. The only other termination points being a fall or game reset to kick off positions. The rest of the system could be considered a payload delivery system and all other software modules being payload delivery components. What we will cover here is the behaviour component referred to as 'striker behaviour' in the league. This is basically a target acquisition system. We will review the kick targeting systems and the difficulties that arise with differing methods. We will also cover the design and deployment of a kick targeting system for both the empirical kick and the analytic kick.

We will begin with a discussion of the nature of the kick problem and the issues that arise, followed by a brief review of the existing RoboEireann kick system, and then continuing from that work to present the current targeting system design for the RoboEireann omnidirectional kick. We will then cover the expansion of the kick design to eliminate the final error, striker behaviour, design and tuning method for an empirical kick using the Nao walk engine.

1.2 Kick Design in the RoboCup SPL League

Ultimately the act of goal scoring could be summarized as 'walk up to the ball and kick it into the goal'. However, there are many potentially conflicting facets to this. Walking up to the ball should be as rapid as possible, but also as accurate as possible. It is also important to arrive at the ball location with a suitable orientation, to avoid (if possible) unnecessary delays in reorienting the robot for a kick. Kicking the ball into the goal is a complex task, particularly given that there are opposing agents designed to prevent this. There is a goal keeper robot in the goal which the ball will have to pass by. The ball must travel between the keeper and one of the posts, so the area we shoot for is actually much smaller then the goal itself. This means that accuracy is very important. Furthermore, the keeper can react by trying to save the goal so the shot will need to be a powerful one in order to beat the keeper. There are also defending robots that are continuously converging on the ball location so the whole process of kicking will have to be executed as fast as possible since often goal scoring opportunities exist for a short time window. Speed and accuracy in opposition to each other will create for us a cost function to manage. Not only that, but as this ball approach is actually a series of objectives that must be satisfied, there are multiple cost functions that will accumulate errors and delays.

What will make this a more complex issue is the differing types of kicks that are available and how much final ball placement error they can tolerate. This will dictate how much breathing room there is in preceding stages that may give rise to cumulative errors. Aside from the kicks themselves, it will be true for all systems that reducing the walk to ball process to as few stages as possible will be best. Then there are the kicks themselves, they will place differing demands on the ball approach process given what nature of a kick they are capable of performing. In the RoboEireann kick system, we can break the kicks down into 4 types; (i) a straight shooting empirical kick, (ii) an omnidirectional empirical kick, (iii) a straight shooting analytical kick and (iv) an analytical kick that is capable at shooting with some range of angles (we will ignore the existence of the 'side tap' kick as it has no fundamental difference in properties to these other 4, it is simply executed at a right angle). The other two properties that will make the walk to the ball process more difficult are the stochastic nature of both the localization system and the quality of the walk used. In both cases, despite many attempts to minimise this, there will be unpredictable variations from the ideal in the available data. To begin we will set aside these other matters, and the analytical kick, to first address the difference between an omnidirectional kick and a straight shooting one. We will assume for now these other matters are constants and that the analytical kick in theory gives us the most breathing room during the final targeting phase. We will then address these issues when we reach the point in the process where they have an impact.

To start then with our cost functions, there is the major difference between straight shooting kicks and omnidirectional ones. Assuming we are on the correct side of the ball, we can break the line up process down into parts. First moving to a position to line up the shot angle at some point back behind the ball, then walk straight to the ball, and when right on top of the ball shuffle to place the ball on the target point relative to the robot within some margin of error (target box). Focusing on the first stage, positioning to lining up the shot, there are two associated objectives. The robot must achieve some location on the field and orientate itself to be in line with the shot angle. This is very much like the problem of 'going to ready positions' with one big difference being the size of the acceptable error box the robot needs to get into. Typically teams will aim for an area the size of a square foot and some reasonable amount of rotation error when going to ready positions. That problem alone has shown to be quite a difficult one for developing teams. Lining up a kick with an initial error of one foot is marginally acceptable. This would most likely lead to a missed shot or require considerably more shuffling at the final line up stage. From the point of view of our accumulating cost functions, simply realizing this position will be a major source of either delay or accuracy. Objectives will have to be satisfied for both position and orientation. Though omnidirectional kicking is obviously better, it is much much better as simply performing this first act is major waste of time. Being able to walk directly to the ball from any obtuse angle eliminates one of the steps in shot line up.

Our next source of a trade off between time and accuracy is the final ball approach, be it from this discussed straight shooting inline position, or from any arbitrary angle of approach given an angled kick system. With any empirical kick, at some point a decision needs to be made to commit to a kick with a fixed kicking angle. Failure to do this can very easily give rise to `live lock' type behaviours, where the robot is continually readjusting and recomputing kick angles, but never actually kicks. This in particular will be subject to a wide amount of variance given the sophistication of the behavioural code structure, such as the ability to separate the head control as an independent process that is not tied to the rest of the motion.

Being able to intelligently look around separately from controlling the locomotion functions is the ideal behaviour that would allow for last second goal checks while walking to fix up any final shot angle errors. We will leave that aside as a level of functionality that many teams may not have. It

would be common to stop, take a quick look around, and then look back at the ball and keep looking at it until the kick has been performed. Any motion thereafter would accumulate more error in terms of shooting accuracy to some degree. With a really good walk like the B-Human walk it may be negligible. If it is the Aldebaran walk and some side stepping is involved, it actually will be a large source of error. So with the different possible combinations of kick styles, walking engines used and behaviour sophistication, this can all play out in different ways, but always with something in common. As some amount of error will be incurred while walking the distance travelled between the location of the last check for or the goal (or other landmarks) and the ball kick location, putting effort into minimizing this distance can be quite important.

To make this more clearer lets consider two extremely different systems: one with a straight shooting kick, the Aldebaran walk, and a look around motion that interrupts the walking, the other with an angled kicking system with an accurate walk and a head the behaves independently from the body. With the first system, the distance to walk from the last shot angle line up position forward to the ball would dictate how much angle error is accumulated. The Aldebaran walk does not walk straight and correcting it is not a simple task. Therefore this distance should be a short as possible. Now this position could be a dynamic one and a stop to quickly look around could be performed when closer to the ball and then the realignment process could be reiterated but this would be costly in terms of time. So then we have some decisions to make: how far back can the line up point be? Is the walk bad enough to require re-correction? This issue presents itself as a cost function and at some point the error has to be accepted and it is time to focus on the ball and kick it.

Moving on to our more ideal case, with the angle kick and independent head, there will still have to be some distance behind the ball set where the head stops looking for position updates and just focuses on kick targeting. Any head search routine will take some time to complete and this time must relate to the distance the robot can travel in this time. There is also a danger of over-running the ball when looking around. At some distance there will be time for one last look before the ball is too close. From then on, all further motion accumulates error. The ultimate point here is that at some stage either the ball line up point or the selected kick angle must be latched, thereafter angle errors can not be corrected. Stopping and looking around when right on top of the ball and then taking one very accurate 'step to' with 3 degrees of freedom can provide the most accuracy, but the looking around can be expensive in terms of time. So in the end, regardless of what system used, there will be a cost function between speed and accuracy that represents maintaining the shot angle when approaching near the ball. Given whatever particular traits of the system used, putting effort into calibrating this line up stage will go a very long way to performing fast and accurate shots to the best of the system's potential. It is very easy here to waste over 5 seconds or add 20 degrees to shot error.

The final design issue to be considered is the size of the error box that is placed around the ideal kick target point. Obviously, the larger the box the faster the kick is executed and the smaller the box the longer it takes to line it up. Selection of a suitable box size can be done through repeated lab experimentation. Again, we are assuming a fixed kick type associated with an ideal finite kick targeting point. A very important issue here, when using the Aldebaran walk, is that robot translations (forward or lateral) inadvertently cause small rotations (yaw) of the robot. This undesirable yaw is more prevalent with strafing lateral motions. Therefore, for example, using a large number of small steps to very accurately attain a target point will certainly introduce a considerable amount of rotation error. Also, it is desirable to avoid using any lateral motion if possible. Performing this final line up stage, or 'shuffling', should not be performed by translation until the very last moment. When using the Aldebaran walk, we propose that the forward translation and rotational walk velocity commands should be used; but that the side translation should not be used in general. Any residual y component errors can typically be removed in the final 2 or 3 steps of the approach. Those few final steps will still be a large contributor to final shot angle error.

Finally, as far as motion is concerned, this is not meant to be the end all be all method. Doing something like plotting an optimal shortest route Bezier curve would likely be more ideal but this is turn would be subject to some errors which would need to be managed. The solution here breaks the method down into simplistic steps in a process that readily lends itself to correction methods. Complex path planning methods would lead to complex error corrections that would have to be performed by visual observation. Complex curves could potentially be difficult to calibrate by eye.

With all these motion related issues in mind, subsection 1.6.2 will cover a single method that can be used to calibrate for both best and worst case examples. The issue of the Nao walk command to output delay will also be addressed within it. But first we must consider a couple of more potentially problematic issues that relate to kicking accurately. These pertain to vision and localization.

First there is the matter of localization. Different teams will use different computational methods to localize. Any method will have a stability property that relates to how the system responds to new land marker updates. The position output will either jump around a lot if not well refined, or it will drift smoothly. In either case there will be some element of change given new information or noise.

Now the ball position may or may not be modelled as its own separate model. Regardless of the approach used to model both the ball and the robot, the ball's position must be fixed relative to the robot. Ideally, the two would be treated somehow as a single object even of they are modelled separately if this information is going to be used to line up a kick. If there is any relative motion between the robot and the ball that is caused by either noise, frequent update jumps, or separate processes (the ball and robot), then vision data should be used to target the kick. This relative motion is analogous with trying to target a moving object, which becomes a much more difficult problem. Given that the ball is actually not moving on the field, this relative motion is 100% error. The ball location will of course have to be updated, but this should be done using data from the robot's perspective only and be protected against any outside variance that may occur from independant modeling or from data such as shared ball information. If the shared ball infomation does move the ball, the robot's position needs to be updated by the same amount to maintain the relative distance from the robots perspective. Note that without some form of compensation, a ball placement error of 1 cm translates to a 20 degree shot error. For example, from the mid field sideline, a shot aimed at the center of the goal with a 20 degree shot error just misses the outside of the goal post.

The vision system introduces it's own set of problems, both in terms of image processing, and translation of image frame information to world frame. Modelling of the robot's pose can be done differently with different error properties. Ultimately for any given team, the vision data may or may not be better than the the ball modelling if it exists. Both sources of data should be checked closely and assessed for errors. At one point in team RoboEireann's development, vision was better in one dimension and therefore during one competition the vision X and world model Y was used to target the ball. But with well designed ball localization, it is more likely the vision data will have more errors. This can still be better if the world model jumps frequently or is itself subject to the problems just reviewed. Filtering the ball noise would be an obvious plan but that can introduce delay. Therefore, a trade off between filtering window size and acceptable noise levels should be looked at. Also some extrapolation of the relative ball velocity to make up for the filtering delay could be beneficially useful to get accurate data.

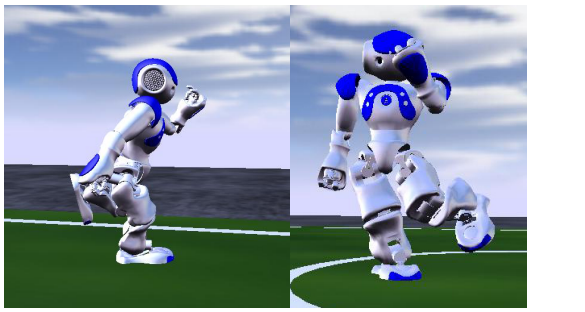
Finally, there is a ball property that is related to both localization and vision. This is the sway in the walk and how it expresses itself in the body kinematics. It would be quite probable that a team's vision system was developed alongside the localization system. Centimetre accuracy is far more than what is needed for localization, whilst this is critical for ball line up. For kicking, that magnitude of precision is unacceptable. The vision data should be checked for ~2.5cm pk 1-2Hz signal riding on top of the Y dimension. This kind of variation may be directly attributable to the swaying of the robot's head back and forth. Neither the ball nor the feet sway. Lining up the feet with the ball with this signal mixed in is very difficult. An offset can be used but it will fail at least 25% of the time. From a kicking perspective, the head, the hips, and the feet are all different parts moving relative to one another. Modifying the vision kinematics will not adversely affect the localization unless somehow it was filtered out by other means. Eliminating the sway is a matter of fixing the robot's kinematics to a particular point of origin. Selecting what is the appropriate point of origin and coordinate frame is far from trivial. One possibility that helps greatly reduce sway induced variations in the perceived ball location is to take the vision kinematic calculation down from the camera to the support foot. In other words, take the support foot as the frame of reference for vision data geometry calculations.

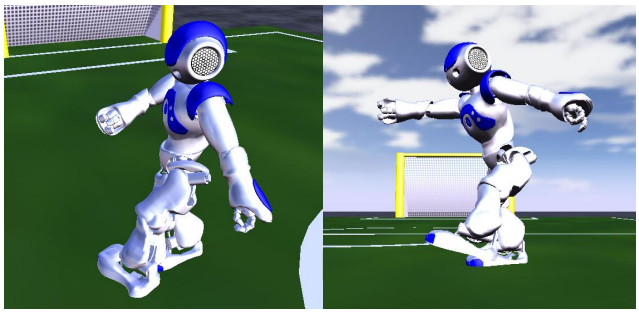
To conclude, these sources of error will pertain to static or 'empirical kicks' swing. All of the above mentioned issues contribute to shot error and execution delay. Any of these alone can contribute 1cm or more in ball placement error or upwards of 20 degrees. These all contribute to missed shots, outright swing misses of the ball, or blocked shots as too much time was taken. It is possible to use a static kick swing and get all the accumulated error down to under 1 cm and a rotation error of less than 5 degrees if care is taken to correct for motion errors and the targeting data is properly understood. Having a dynamic kick swing that adjusts for ball placement when targeting obviously eliminates several sources of error. But as of RoboCup 2011, most teams play was still observably suffering from some or all of these problems. Even teams with a dynamic kick swing may not have one that is capable of kicking at angles so even then some of the orientation problems will still arise. The striker behaviour and corresponding calibration method that will be outlined in this chapter will be assuming an omnidirectional empirical kick. This solution will also work for a unidirectional kicks just with a little bit more error that will have to be tolerated. First, the prior design of this kick will be explained. This is followed by discussion of its continued development for this thesis which includes the omnidirectional targeting system and a gait modification to allow for a wider range of ball placement error.

1.3 Review of the Existing RoboEireann Kick

In order that the following few subsections make sense, a brief review of the omnidirectional static kick swing used as a foundation for this work is in order [219]. There is more that goes into this kick than what will be covered, but in brief, the design is as follows:

Using photographs taken of human footballers as inspiration, the kick swing gait was mapped out as a series of poses that make up a kick swing for a full range of motion from 0 to 90 degrees on each leg. Poses were defined for 3 swings; straight, 45 degrees and 90 degrees with a set of 5 poses for each angle. There is also a set of two poses on each side to set up into the cocked position (A and B). Together these poses serve as a look up table to interpolate a full range of swing angles, allowing the robot to kick the ball at any angle from 0 to 180 degrees from one side to the other. These poses were hand tailored joint by joint using the Microsoft Robotics Studio. Some examples of these poses are shown in the following figures:





An abstract representation of this look up table can be expressed in the following matrix (Figure 160):

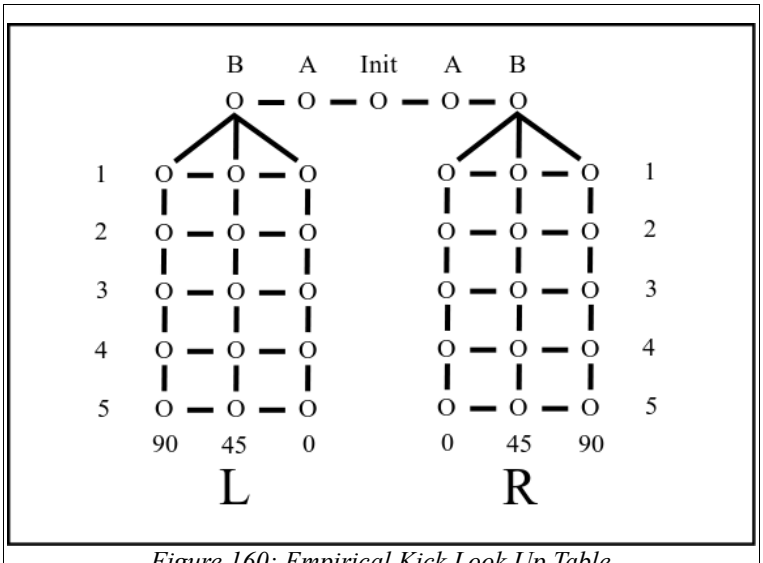


Figure 160: Empirical Kick Look Up Table

Each node in this matrix represents a full body pose. Then, given a desired kick angle, 5 poses are dynamically generated for each joint angle via linear interpolation between the range 0 to 45 or 45 to 90. The swing itself is another timewise interpolation with a 10 ms (DCM clock rate) step size between each generated output pose with the time between each major pose (1-2-3-4-5) being the governing kick swing speed (~50ms).

The accuracy and performance of this kick was evaluated with a series of trials, ball placement by hand, and a range of kick swing speeds. The results are shown in the following Figure 161. The different symbols used represent a different swing speed for the given kick angle.

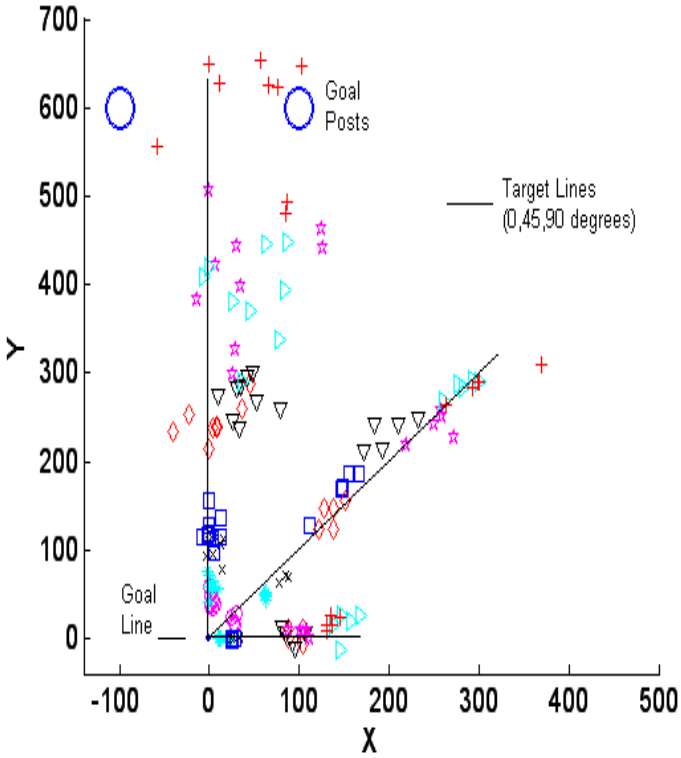


Figure 161: Empirical Kick Swing Tests Results

With the ball placement by hand, the ball placement error was in the order of less than 1.0 cm. It can be seen from the figure that with that magnitude of error, the goal keeper can reliably place the ball in the opposing goal. It should be noted as well that this was done using the first official SPL league ball that was neither perfectly spherical nor evenly weighted.

The shooting range was then evaluated with full speed swing at 10 degree increments. The area this covers is shown in Figure 162:

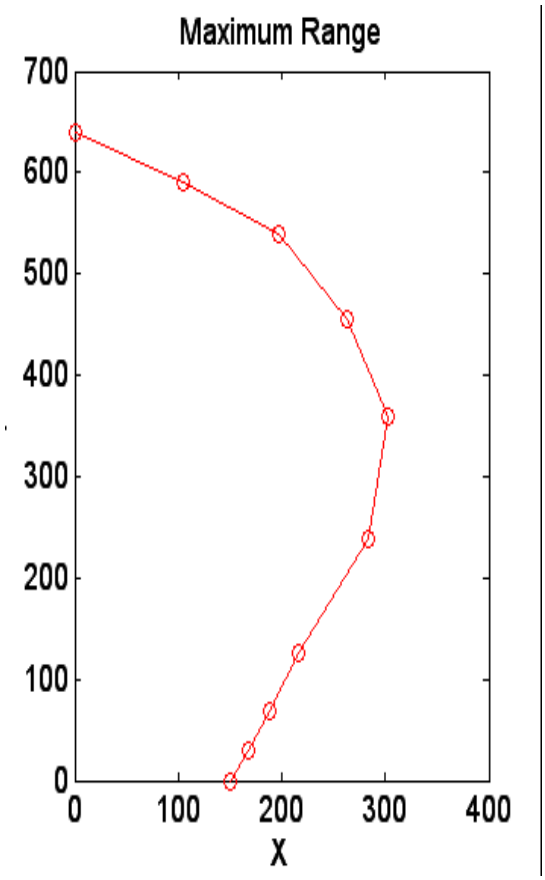


Figure 162: Empirical Kick Ball Placement Range

Given the high friction surface of the piled carpet in the RoboEireann lab, using this kick, the ball can be placed anywhere on the field (RoboCup surface) from the goal position accurately without having to alter the robots orientation, assuming that the ball placement error is well under 1 cm and the correct targeting data has been generated.

This kick swing design was credited for an undergraduate final year project [219].

1.4 The Empirical Kick Targeting System

The targeting system designed for this kick is the aspect that makes 'Empirical Kick' a suitable name. With this kick, the relationship between the kick swing angles and the resultant ball travel vectors was measured in the lab by kicking at targets. The robot was placed in the corner of the field and duct tape markers were placed out in an arc in front of the robot 1 to 3 meters away (0 to 90 degrees respectively) at 10 degree intervals. A ball was then placed in front of the robot with its distance and bearing as reported by the RoboEireann vision system displayed on a terminal and recorded. The kick was then triggered to hit the targets. This was repeated until values were found that hit the targets at every 10 degree increment.

This data was then plotted in MATLAB, as shown in Figure 163, and the basic fitting toolbox was used to find the lowest order polynomial function that best described the data.

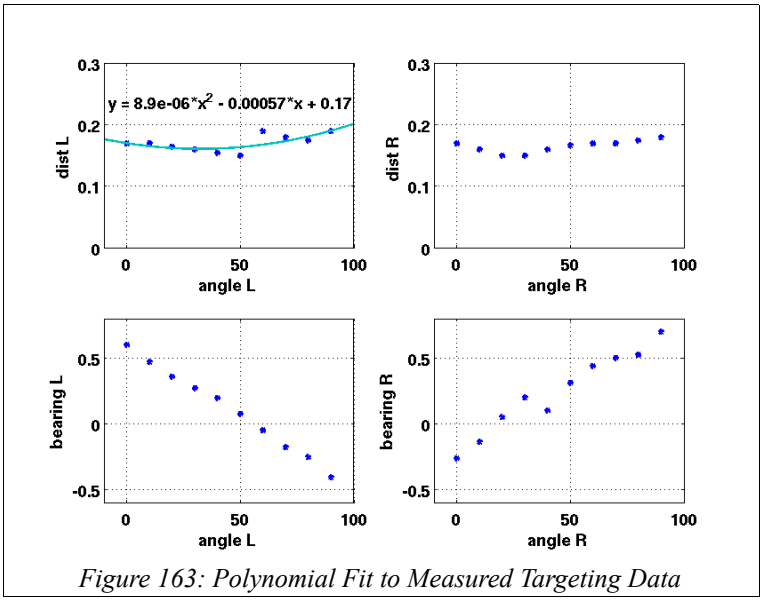


Figure 163: Polynomial Fit to Measured Targeting Data

The functions for this particular version of the kick swing were found to be:

$$\begin{aligned} bearing_R &= -2.8\text{e-}05\,\theta_{KICK}^2 + 0.013\,\theta_{KICK} - 0.24 \\ bearing_L &= -0.011\,\theta_{KICK} + 0.6 \\ distance_R &= -2.3\text{e-}07\,\theta_{KICK}^3 + 3.8\text{e-}05\,\theta_{KICK}^2 - 0.0015\,\theta_{KICK} + 0.17 \\ distance_L &= 8.9\text{e-}06\,\theta_{KICK}^2 - 0.00057\,\theta_{KICK} + 0.17 \end{aligned} \tag{372}$$

It is a good idea when using a method like this to re-plot back into the field plane in simulation and observe the results. The data should look like the following Figure 164. The red points correspond to the left kick, and the blue to the right. There should be some balance to both data sets with the straight shooting point being a little closer to the robot than the sharpest angle. Each side will not be perfectly symmetrical as the kicks swings are built manually and independently. What should be very obvious is if there are any camera calibration errors. With small misalignment in the camera mounting, large data shifts and rotations occur even at this distance. This can be corrected by using rotation matrices and shifting the data until it looks reasonably balanced. Preferably, this would be done by calibration of camera and joint offsets used in the vision geometry calculations.

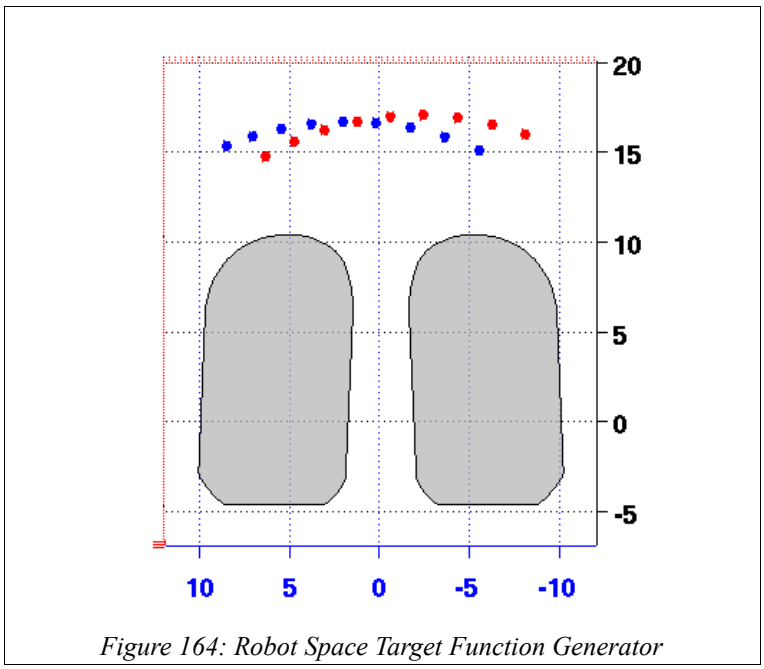


Figure 164: Robot Space Target Function Generator

With that process complete, the whole transfer function between the kick swing gait and resultant ball travel vectors are described simply. In comparison to the inverse kinematics and signal trajectory mapping method, it is a lot more consistent and produces excellent results. The empirical method clearly involves more experimentation to generate the overall calibration, but it also reduces the need for expensive online inverse kinematics computations. The problem of the foot orientations due to the mechanical coupling in the hip Y-joint is circumvented, while allowing a wide range of powerful kick swing angles shown in Figure 165.

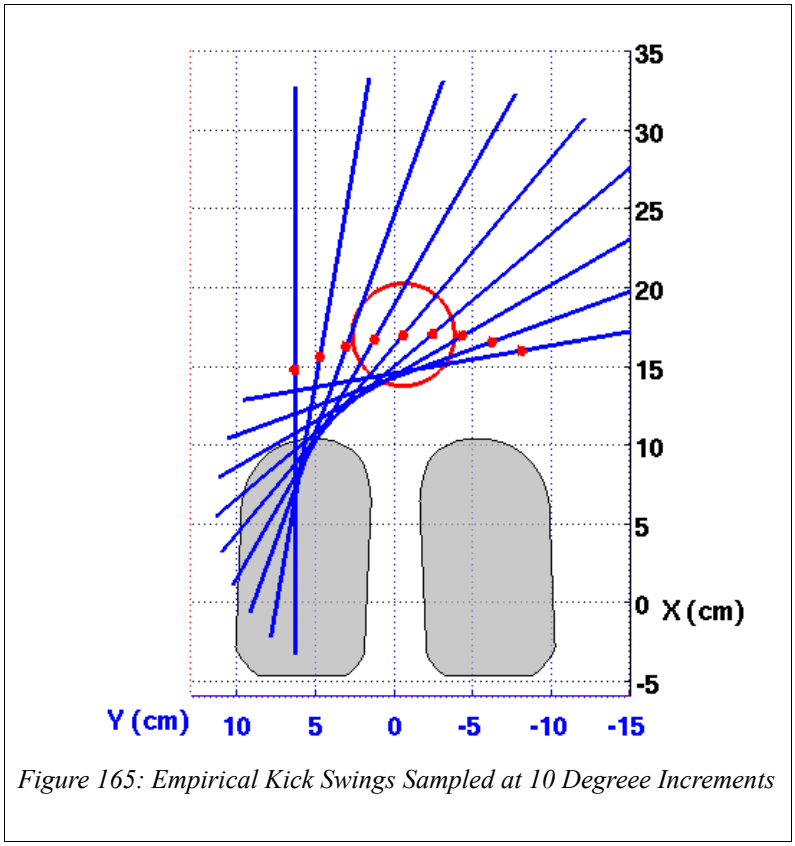


Figure 165: Empirical Kick Swings Sampled at 10 Degreee Increments

The only difference between this kick and an analytically defined one is dynamic compensation for ball placement near, but not on, the ideal target point to allow for faster and more accurate targeting. This dynamic compensation property can also be implemented with an empirical kick by extending the kicking gait and redefining the target points as target regions the kick extension operates

in, as described in the following subsection.

1.5 Expansion of the RoboEireann Kick to Eliminate Error

One obvious error source in kicking is the lining up of a static kick swing that requires the ball to be at a specific target point relative to the robot for optimal shot performance. Line up to a very tight tolerance is very time consuming and consequently some margin of error must be accepted. This error region in both dimensions creates a target box to place the ball in instead of a target point. The larger the box, the less time it takes to manoeuvre into kicking position. However, this leads to increased kick inaccuracy as the box size increases. This would be the primary difference between a statically defined kick swing and one that makes use of an inverse kinematic kick swing trajectory that dynamically compensates for the ball placement. However, kicking at angles is much more difficult with inverse kinematics. As shown in the section on inverse kinematics in chapter 2, specifying the orientation of the leg axes is problematic and would certainly clash with balance systems that define a target torso attitude. It is possible nonetheless to get the best of both worlds, powerful kicking at angles (as is the nature of the RoboEireann static kick swing) and dynamic compensation for ball placement, relative to an ideal target point, allowing for a larger target box and thus fast execution. This was accomplished by modifying the existing kick swing pose so that the swing leg is outstretched laterally as far as is possible mechanically. This second set of poses constitutes an 'extended set' and provides another dimension to the kick that can be interpolated dynamically. Therefore, the kick's look up table is expanded as portrayed in Figure 166.

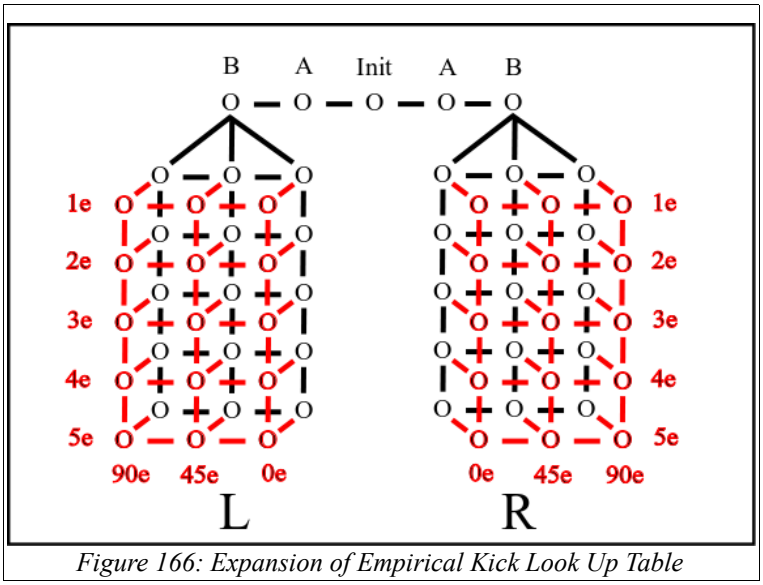


Figure 166: Expansion of Empirical Kick Look Up Table

The extension range defines the width of the target region and is a function of the kicking angle. The target box for the kick will rotate in the field plane as the angle increases so each unique target point will have its own unique target box.

The width of the extension range was found empirically by photographing the normal swing and the extended swing contact poses with a tripod mounted camera from above. A ruler was placed under the foot for the 0, 45 and 90 degree kicks. As the kick swing makes use of the hip roll, the extension range reduces with increased angle. The results were 7.3, 5.0 and 0 cm respectively. Mechanically, this extension collapses to zero as the kicking angle reaches ~90 degrees as the hip abductor has already reached its mechanical limit. The reduction of extension between the 0 and 45 degree kicks is assumed with confidence to be linear, but the region beyond that is not so clear. There is still some more room for extension beyond 45 degrees, but it is difficult to fix the robot in these interpolated poses without coding up a specific debug tool. However, other aspects of the kick experience a transition around 70 degrees, such as a drop in kick strength and having to begin using the off center part of the foot to achieve the desired angle. It is quite likely that the extension range drops to zero at some point before 90 and that it would at least not occur before 70 degrees. Therefore, a limit on extension has been set to zero at 70 degrees. Beyond this range it would be better anyway to use a side kick like that which was first introduced to the league by the B-Human team.

Also, the full 7.3cm extension in the straight kick is far more than necessary. It is simply what is physically realizable by the robot, not something that should really be done. Extending the kick leg

outwards does alter the moment arm effect in the kick and thus leads to a degraded performance. Reducing this range to a limit that is more practical provides a second point to limit the extension range to. The maximum extension required is governed by the performance of the ball line up behaviour. If the robot can be delivered to the ball quickly within the range +/- R then that R would be a suitable extension limit. For now, +5 cm (straight kick) was chosen as the range at which the robot can be aligned quickly and accurately. That provides a 2nd limiting point and a line can be drawn between them that is certainly well within the mechanical range.

The following Figure 167 shows the measured extension limit, the unknown zone and the limit that was chosen as the upper limit to the operating range:

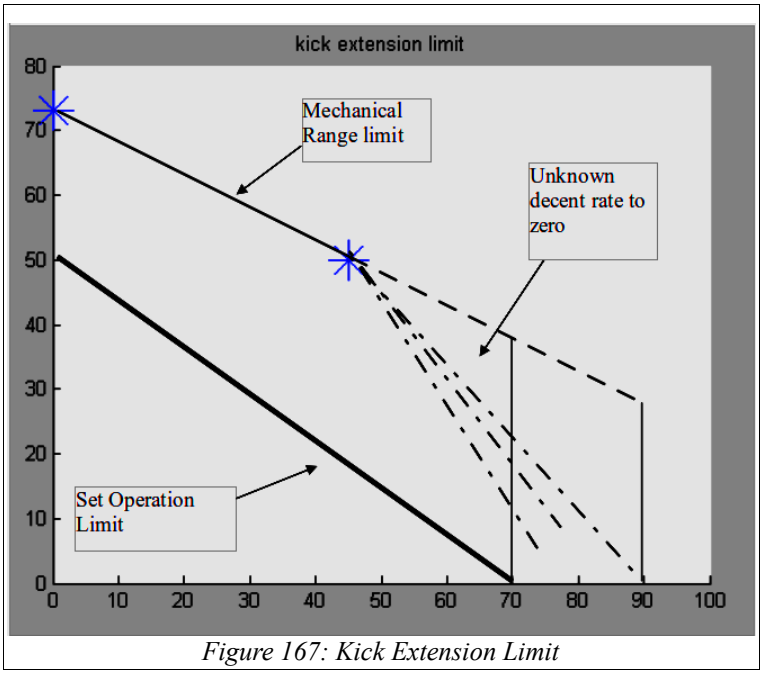


Figure 167: Kick Extension Limit

These measurements yield two functions, one for the width of the target box, (relative to the Hip Roll Frame), and the other as the range to interpolate within to calculate the required extension to target the ball. The target box length was set to a distance of 4 cm, a value chosen from practical experience that will be inline with the kick swing and provide decent ball contact.

These empirical functions are:

Dynamic targeting width:

$$TargetRegion_{WIDTH} = -\frac{5}{7}\theta_{KICK} + 5\ cm$$
$$TargetRegion_{LENGTH} = 4.0\ cm \tag{373}$$

Interpolation range maximum value:

$$KickRange_{MAX} = -\frac{2.3}{5}\theta_{KICK} + 7.3\ cm \tag{374}$$

This will provide us with the targeting regions shown in Figure 168 that collapse in width as a function of kick angle:

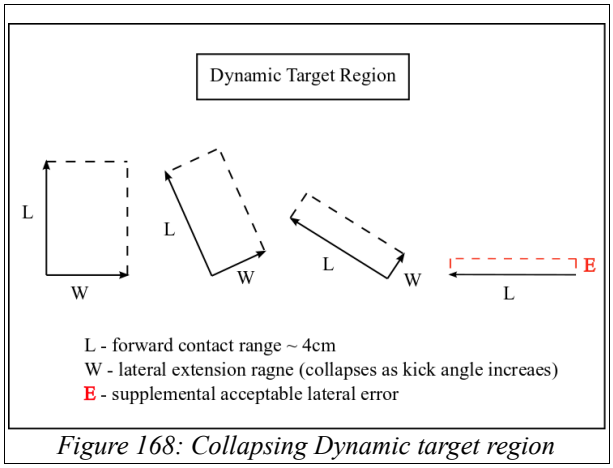


Figure 168: Collapsing Dynamic target region

Each region is defined as follows (Figure 169):
(Note the target point is the corner point to the target region, not the center point).

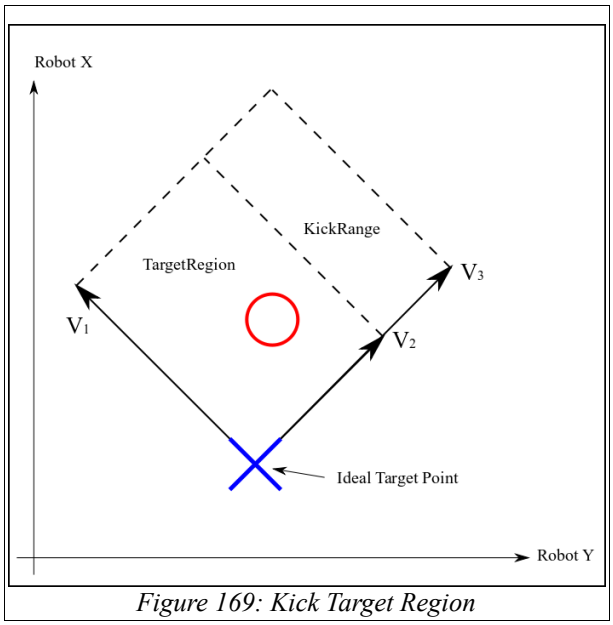


Figure 169: Kick Target Region

Where:

$$V_1 = R_{\phi z}(\theta_{KICK}) \begin{bmatrix} TargetRegion_{LENGTH} \\ 0 \\ 0 \end{bmatrix}$$
$$V_2 = R_{\phi z}(\theta_{KICK}) \begin{bmatrix} 0 \\ -TargetRegion_{WIDTH} \\ 0 \end{bmatrix} \quad (375)$$
$$V_3 = R_{\phi z}(\theta_{KICK}) \begin{bmatrix} 0 \\ -KickRange_{MAX} \\ 0 \end{bmatrix}$$

Finally, we need a method to find a value that will determine if both the ball is inside the target box, and the amount of extension that is required to eliminate the final error between the target point and the ball location. This value is calculated as follows:

First, subtract the generated target point position from the ball actual position to yield the targeting error vector (Figure 170).

$$V_{ERROR} = V_{BALL} - V_{TARGET} \tag{376}$$

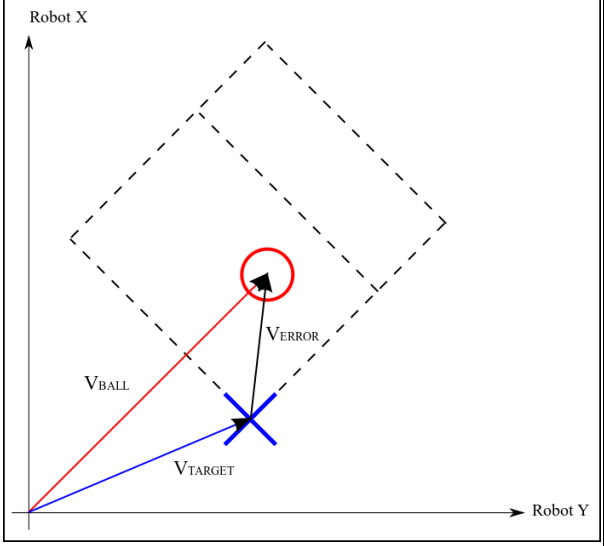


Figure 170: Kick Target Error Vector

Next the error vector is projected against the other vectors, V_1 and V_2 via vector dot product to produce scalar values, $Error_{LENGTH}$ and $Error_{WIDTH}$, representing the length of the projection along V_1 and V_2. (Figure 171)

$$\begin{aligned} Error_{LENGTH} &= V_{ERROR} \cdot V_1 \\ Error_{LENGTH} &= V_{ERROR} \cdot V_2 \end{aligned} \tag{377}$$

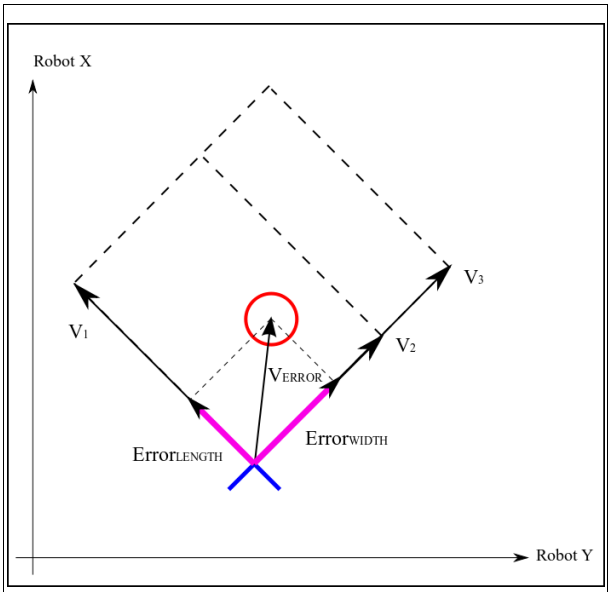


Figure 171: Target Error Projections to Target Region Dimensions

We then use these values to activate the kick swing system:

$$\begin{gathered} if\ (Error_{LENGTH} < TargetRegion_{LENGTH}\ \&\ Error_{LENGTH} > 0)\ \{LengthCaptured = True\} \\ if\ (Error_{WIDTH} < TargetRegion_{WIDTH}\ \&\ Error_{WIDTH} > 0)\ \{WidthCaptured = True\} \\ if\ (WidthCaptured\ \&\ LengthCaptured)\ \{KickEnabled = True\} \end{gathered} \tag{378}$$

The $Error_{WIDTH}$ is then used to interpolate between the normal kick swing and the extended set:

$$Swing_{OUTPUT} = Swing_{NORMAL} + \frac{Error_{WIDTH}}{KickRange_{MAX}} \left(Swing_{NORMAL} - Swing_{EXTENDED} \right) \tag{379}$$

1.6 RoboEireann Striker Behaviour

With our kick and kick targeting methods developed we require a system that will deliver the robot to within its target area. The striker behaviour is really a set of convergent algorithms and simple closed loop controllers. There are various solutions that work ideally in a simulated environment but do not translate well to the real world environment. There is always some discrepancy between a modelled world and the real world. It is often the case that the design methods make the transition from simulation to reality difficult purely based on the fundamentals of the design assumptions and techniques. The more noise, variance, and unaccounted for elements, that exist in the real world but not in the model, the more work is going be required in hardware development to mitigate these new challenges. This is true for just about every hardware application electrical or mechanical that exists to some degree or another. A robot is a very complex system and the Nao robot is not manufactured with a high degree of precision. Many obstacles will make it difficult to simply walk the robot up to a kicking position accurately and efficiently. Modelling all of the sources of errors in their entirety is unrealistic, consequently, statically defined motion planning fails. Even some nature of dynamic path planning that does not account for all the environmental realities will fail. The approach pursued here is based on a series of controllers that will combat the different sources of error and be able do so in the face of practical variations between robots. Furthermore, this division into a set of controllers, as compared to designing one large complex kick line up behaviour, permits simplified calibration of the controllers. For most of these motion control problems the main source of information is visual feedback. So calibration will require repeated testing and observation of the effects of altering parameters. To do that properly it is important to isolate variables so that cause and effect relationships can be clearly observed. To that end, compartmentalizing the control functions to be as specific and simple as possible will make algorithm development and implementation much less problematic.

What will be presented in this subsection will be a tuning method approach, a method that relates to real world observations and has parameters that can be adjusted that will have effects that are not overly complex. Therefore, the kick line up behaviour method developed here does not attempt to plot the shortest route in all cases. The system design is specifically developed with calibration and testing in mind. With frequent code system changes, hardware changes, rule changes, and wear and tear to robots, a system design plan is better if it is more robust and user friendly. Time pressures in a competition environment will dictate the need for rapid system adjustments to be made with confidence. The method presented here has always been developed on the hardware platform only, and has evolved as the environmental factors have become more understood and as they have changed with robot versions. In one form or another this method has always been a part of the RoboEireann competition system and was best presented during the Open Challenge of RoboCup 2011. This presentation won first place in the Open Challenge for its localization merits, however it also clearly demonstrated fast accurate ball line up behaviour. Part of the credit for this performance was also a fast walk engine (due to B-Human). However, if the Aldebaran walk was used this motion control system would have performed with equal accuracy, just somewhat slower when approaching the ball form a distance.

With these adaptability, portability and system changes in mind, this method has been implemented in a variety of forms: brute force if/else, a classic state machine, a hierarchical state machine, and in its current form of a costate programming structure currently being developed by Jeff de Hass of the B-Human team. Because of various noise sources in the system data, hysteresis is required in several of the algorithms. In this section, we describe several environmental challenges and solutions to counter them. The latest version of this method is attached as Appendix K. The latest version of the kick gait had to be radically modified in response to a change made by Aldebaran to the low level stiffness controller to account for the fact that the Hip Roll motor could not support the robots weight. Consequently the dynamic error compensation from the preceding subsection is not implemented in the current build as no time has been made available to redesign the extended gait. This makes no difference however, as the striker behaviour developed is designed for the worst case scenario, statically defined kick swing (straight or angled) and using the Aldebaran walking system (delay) on a poor surface such as the RoboEireann lab or the RoboCup pitch. We will treat it as the most difficult case and some properties may be redundant, given other systems and environments. In such cases, this can be easily accommodated by setting some parameters to zero or large values.

On the whole, the system design is not too complex and the focus is on the method of tuning. The correct tuning sequence is very important otherwise the whole behaviour appears to be extremely erratic and produces bizarre path planing that can be difficult to trouble shoot. As well as robustness, focus is placed on high fidelity performance. If tuned so that all errors are kept to their absolute minimum with every aspect of the behaviour and complete kick targeting and swing system, the results are extremely fast and accurate.

The system consists of two main elements: a behaviour mode, or state controller, and a velocity

profile parametrization of the motion system.

1.6.1 Behaviour State Control

To being with we will look at the behaviour algorithm that determines which motion control mode should be used. The whole method is depicted in Figure 172 and can viewed as a process or series of steps that must be accomplished to capture the ball.

The three modes of motion used are:

1. Approach
   This motion is the same as most forms of locomotion or transport systems. The two dimensions used to move are forward and steering (yaw rotations). This is good for walking in a straight line using the yaw to correct for errors.
2. Encircle Ball
   The encircle ball is the complement of the approach. It tries to walk in a circle while remaining faced towards the ball. Lateral velocity is used to move while again the angular velocity keeps the robot facing the ball.
3. Line Up
   The line up does not attempt to fix rotation errors, it uses both the X and Y velocity to translate the robot while maintaining the current orientation.

Although not impossible, it is very difficult to use both the Y velocity and Yaw simultaneously. Trying to fix lateral ball position and shot angles at the same time causes both dimensional controllers to overcompensate for each other. Also, the combination of these two motions produces a spiral path. It is hard on the mind and best not to try to aim and calibrate a spiral by visual inspection.

The algorithm is then comprised of two key distances (shown as rings) that set the transition between motion modes and an angle, $\Theta_{K\ MAX}$, that is used for an angled kicking system. There is another range, 'Last Chance to Look', that is more abstract. This distance will be highly variable given the different system used. It would be the distance or stage to either stop and do a quick look around for final localization updates, if it is a motion blocking event. Alternatively, the last time a look around should be triggered if it can be done independently and still have time to return the head to focusing on the ball for lining up without overrunning it (a risk introduced when looking away for a variety of reasons). The length of time which the robot can move for without looking around, while still maintaining good localization should be known by each designer. For very good localization systems, doing a non blocking head check is advisable during encircle ball state and not afterwards. Less accurate localization systems may require subsequent checks for landmarks. The main thing is to not interrupt the motion states so that they have to start and stop. A blocking landmark search to aid localization should occur at the transition between states where the robot velocity will be at its lowest. For example, in low accuracy localization systems, this minimum velocity occurs just prior to line up.

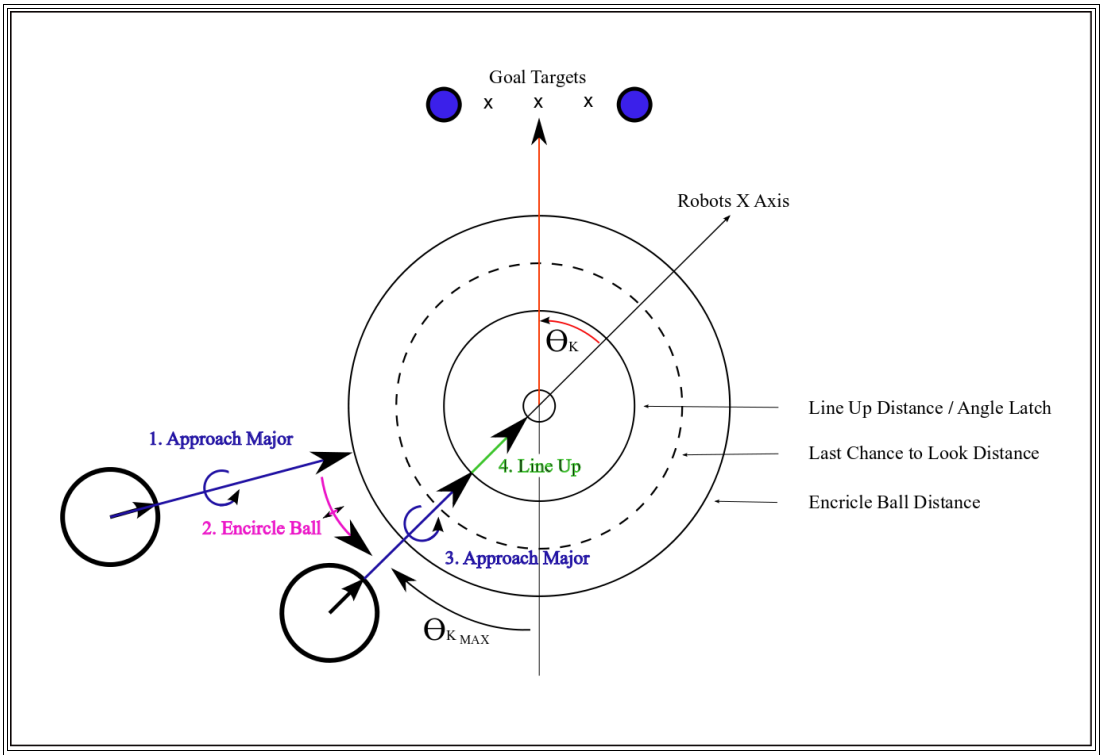


Figure 172: Striker Behaviour Modes and Transitions

Acquiring the ball is a process of satisfying a sequence of goals. At each stage (that is, state in the state machine) we check for achievement of our objective. If this has been achieved, a state transition occurs. All states have two important features: an error bound is required as exact values will never be reached, and, hysteresis will have to be implemented on each transition due to quantization noise that will exist everywhere in the system. So it should be kept in mind that every goal or target stage has a cost function associated with it and effort should be made to optimize the error bound magnitudes. They should be small for accuracy, but big enough to execute quickly, however, there should never be oscillations around target values. Oscillations around target values is a very observable common problem with some RoboCup teams. Step length near target values should be smaller than the error bound itself. That may mean decelerating near target values and the accuracy will come at the expense of speed. Conversely, large step lengths can be used and target limits replaced with target thresholds that once crossed will trigger a state transition. Here overruns occur and speed is being favoured over accuracy. Over use of this method can lead to quickly executed missed shots. Accurately calibrated steep decelerations are best.

The process then is simply like the motion modes:

1. Approach Major
   In this state the robot will accelerate to top velocity as rapidly as possible to the Line Up position while monitoring both the distance to the ball and the kicking angle. $\Theta_{K\,MAX}$ can be used as the actual error bound for the shooting angle alignment. If the encircle ball distance is reached and the kick angle is greater than $\Theta_{K\,MAX}$ then the state transitions to encircle ball. Or it will continue directly to the line up via a second Missile Approach phase (Missile Approach Minor). For non angled kick systems $\Theta_{K\,MAX}$ is set to near zero (the magnitude of acceptable shot angle error). After this point there is no yaw correction and straight shooting kicks will accumulate error with every step. Therefore, this distance should be as close to the ball without over stepping the ball when doing a full encircle from the 'wrong' side of the ball. Angled kicking systems accumulate no shot angle error during both approach stages (pre and post encircle ball distance).
2. Encircle Ball
   During this state the robot traverses a circle to achieve the the target shooting angle. Here decelerating would be a good idea for a straight shooting system to properly set a good angle. Angled kick systems do not have to decelerate. Once inside the shooting range, over run translates into straighter shots which is better anyway. Here only $\Theta_{K\,MAX}$ is evaluated in the same manner as the preceding state.
3. Approach Minor
   Approach Minor is just like Major except for a slight modification of how the ball is orientated relative to the robot. The approach keeps the ball centred in the robot's coordinate

system between the feet on approach. It would be preferable to line it up with the intended target point that will be chosen.

This can only be done by an angled kicking system, however, as any orientation changes at this point would mess up the prior completed encircle ball to fix shooting angle. So straight shooting systems do not make this modification.

For angled kicking systems, the shot angle is latched (very important) upon entry to this state and that value will be used to control the orientation of the ball relative to the robot. For an inverse kinematics analytical kick system that would be some box with an ideal point on either the left or right side. For the empirical kick system, its target function generator would yield a specific target point to hit.

Note!, this is not the actual kick angle and target point that will be used, it is just a good approximation, and effort in getting it there early will reduce the action required in the line up phase where otherwise a distance nearing 5 cm will have to be covered (for straight shots this is the distance between the robots feet and the ideal target point in front of the foot).

Switching from bringing the ball down a centered line to lining it up with a laterally offset line will bump the yaw controller up to a new target. This controller bump can only be corrected by angled kick systems after this state. (To avoid digression this solution to offset orientation control is presented at the end of this subsection)

4. Line Up

This phase is where the final accurate shuffling takes place. Upon entry to this state the current shot angle is latched and no further yaw correction will occur. Taking a look around to fix the angle with blocking head motions would in most cases be done here and the value is latched once this is complete. Independent head controllers should complete the search cycle by this point. It is important moreover that this state be executed with the least amount of steps possible and therefore it should be very close to the ball. With the asymmetrical nature of the robot HypYawPitch joint, the robot changes orientation with every lateral step. This is not a trivial matter, 5 steps with 2 degrees of error each accumulates to 10 degrees total error, which is enough to cause major errors in kick accuracy.

The Aldebaran walk has a delay in its response to velocity commands that becomes very problematic. We will address this issue separately.

The Kick Line Up Problem

As each lateral step generates rotation error we will want to execute this line up task with as few steps as possible. Ideally, one perfectly calculated step would suffice. However, we are using a unit-less velocity controlled walk (Aldebarans) not a calculated 'step to' motion. Even with units there are inaccuracies due to reduced holding torque in the joints without more sophisticated closed loop position control for airborne body parts. Therefore, it seems quite likely that in most cases two steps will be required.

The major problem faced here is that the Aldebaran walk has a control delay. This will be expressed as the robot taking one extra step after it has been commanded to stop. Because the real Nao walk velocity is unknown, it is not appropriate to make final line up steps proportional to the existing error. To solve this problem we require very accurate step length control (velocity) as a function of distance. There is also a conflict between travelling forward quickly to the ball and not overrunning it in the forward dimension which again causes conflict. The region therefore that will require a lot of accuracy in calibration is around the transfer function origin, small distances and small velocities. The design and calibration of the velocity profiles are the next major component in the overall ball acquisition system.

1.6.2 Velocity Profiles

A common problem in setting the robot's walk velocity at any time is the competing objectives between different motion states. There is a conflict between gaining ground quickly and taking small accurate steps when required. The Aldebaran walk delay only compounds this issue further. As just discussed, lining up the ball with what will be mostly lateral steps calls for precision calibration near the lower values in both dimensions. Whereas the need for speed increases rapidly as the distance from the ball increases. This conflict is best described in the velocity X dimension where it also becomes a problem to calibrate. Figure 173 demonstrates the regions where we want to operate (green and blue) in a distance to velocity transfer function.

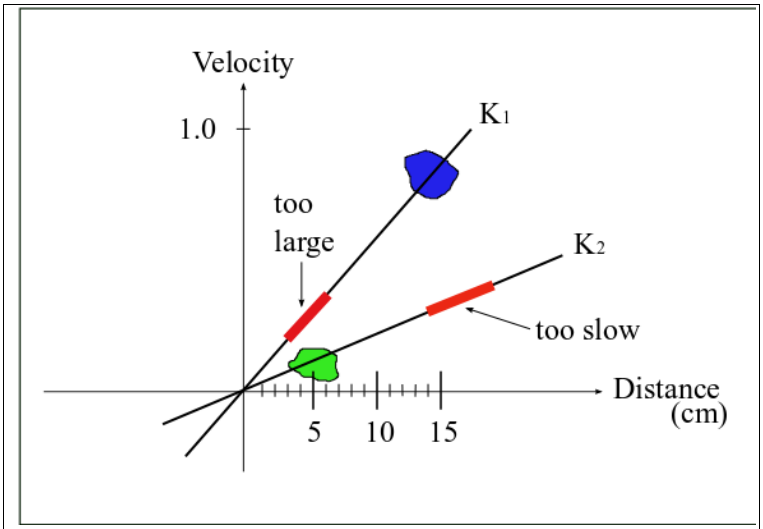


Figure 173: Mutually Exclusive Linear Gain Solutions

It should be clear then from the figure that what we need is a gain line with an offset to satisfy the problem. However, it would be unwise to parametrize the whole problem as a slope and an offset. Those are abstract concepts that do not relate well to visualizing the real world motion problem. Saturation limits will also have to be set. What will work well is maximum and minimum limits that we can use as corner points that will provide conceptual ease during calibration. We will also have 4 parameters instead of 2. Their modification will result in readily observable effects that operate in only one dimension. From these points the gain transfer function can be computed automatically to put things back in to the more abstract gain/slope domain for use in the controller. This velocity profile is shown in Figure 174:

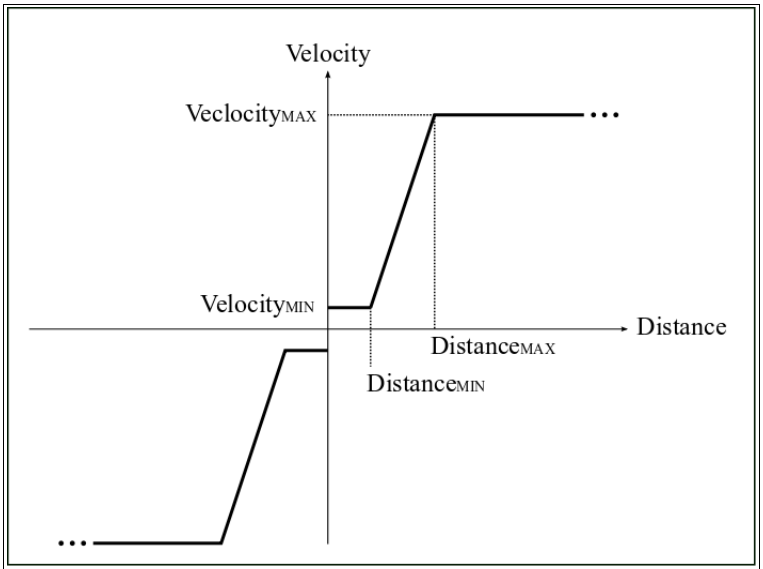


Figure 174: Velocity Profile Transfer Function

The real key to this system is its tuning. Changing any value will break the system easily if it is calibrated tightly. Tuning adjustments may have to be made to adapt to other system module changes, hardware changes or environment changes. A method has been worked out here (next subsection) to iron out any bugs and to systematically calibrate the parameters.

1.6.3 Velocity Profile Calibration

The following is a method used to calibrate the velocity profile parameters. If done correctly it can be done quite quickly. Before doing any of this the motion system's maximum velocity and gait parameters have to be calibrated. When the walk is determined to be stable at its fastest values, they should not be changed. Any changes or updates to a walking gait can require redoing some of the velocity profile calibration. However this is simple enough to do after a first pass at it.

Step 1 Straighten the walk

The Aldebaran walk does not in general walk straight 'out of the box', it will always veer of to one side. It does have a backlash calibration value but that has not been explored and may prove difficult to tune and interpret. A simple solution is just to calibrate for the error in the motion controller. There is also the delay. Here we will be using the X and Yaw velocity profiles.

The first thing is to get rid of the constant veering to one side during a forward walk. This can be compensated for with a counter rotation continuously applied. This is set as a fixed offset that is always added to the walk angular velocity. To set this value, drive the robot with the velocity X function that is part of the Naoqi motion module at maximum velocity (outside of game code). Slowly increment the added compensation value from zero until the robot is walking straight.

Next we will use the yaw or 'theta velocity' profile. As well as veering to one side, a problem occurs with the delay in the walk. Setting the gain line through the origin would mean the robot is constantly adjusting for errors but these adjustments would never actually match the actual error when it is implemented by the walk system. This creates a weaving walk trajectory that can very quickly spin out by 90 degrees off the target orientation and then the ball is lost. This triggers a search for ball routine and it all get messy as the cycle keeps repeating. Therefore there are two issues to deal with. The technique here will solve both these problems.

To begin, set the maximum distance in the X velocity profile to a low value, as shown in Figure 175. This will force the robot to always walk at the maximum speed. Then have the robot chase the ball across the length of the field (using game code motions now).

Note: Only adjustments to the X velocity profile will be shown graphically. Both the Y and Theta velocity profiles have the same form as shown in Figure 174.

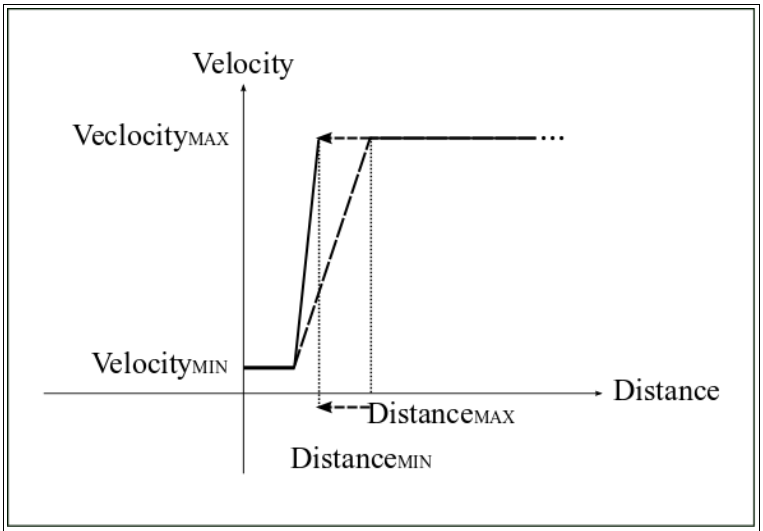


Figure 175: Velocity X Profile Calibration - straighten walk

Next switch to the theta velocity profile. We will want to set a dead band and the gain by adjusting both the maximum and minimum theta distances. Creating a dead band also creates a weave, but an opposite one that is directed towards the target orientation and therefore should not suddenly veer off sharply. The minimum velocity theta is set to zero to produce the dead band. The minimum distance theta is then adjusted to set the size of the dead band (only a small one is needed). The maximum velocity theta is then used to set the gain slope. It works in the opposite manner as a gain value, the higher the value, the lower the gain. With a wider dead band and lower gain a weave will occur as the orientation bounces back and forth between the limits. Reducing the value toward zero seems to cause something like an inversion at some point where it weaves the other way out of sync with the error and becomes unstable. Slowly reducing the dead zone width from a larger value incrementally will begin to reduce the weaving until some 'resonant' spot is found where straight walking is achieved. This is readily observable if attention is paid to it and is easy to deal with as a range of combinations will work (use small values). This tuning is really very simple and trial and error should sort it out quickly.

Step 2 Target box Convergence

Once the walk has been straightened, the first thing to calibrate will be the convergence of the line up state. There are a few issues at play here and it can get a bit tricky. First of all there is a

minimum step length that will be required for the robot to gain ground. This is a function of the ground surface material. In the RoboEireann lab the pitch is covered with a piled carpet. With very small velocities the robot will get stuck in a depression and be unable to move. Whereas a very different thing happens on the RoboCup playing surface. The Aldebaran walk suffers from slippage with small velocities and also will not be able to gain ground. It is possible for the line up system to compute very small values when only a tiny error exists outside the capture region. Therefore a lower saturation step length has to be set, and that it must be a value great enough to able to overcome surface conditions and move. We also want to make very small adjustments with the last step for accurate targeting. Therefore, the lowest value that is capable of moving must be found by trial and error.

To perform this calibration step a few system parameters will first have to be set to temporary values. Both the maximum and minimum distance values in the velocity profiles (X and Y) must be pushed outward to high values to force the system to always take smallest steps as shown in Figure 176. If an angled kick system is used, a fixed target point will have to be hard coded in place of the target function generator in order to make this tuning repeatable. Also the final line up transition distance needs to be set to a temporary high value. These changes will force the robot to always take the smallest step size toward a fixed target point. Place the robot near the ball adjusting the velocity minimum value and observe the effects and find the value where it just begins to start moving (~ 0.5 cm per step). This minimum velocity will have to be set in both the velocity X and Y profiles.

Once these minimum values have been determined, the next thing to do is find the size of the error box around the target point that is as small as possible that will converge quickly. The step length can not be larger than the box or it can just dance around it, failing to drop into it quickly. Now we will want the smallest box possible for good accuracy. To find this value, keep reducing the size of the box incrementally until oscillating around the target region occurs and then dial it back up again slightly. This final value should then be tested a good number of times to make sure there was no luck in hitting the small box and that it will work every time. The robot should walk in a very slow straight line directly towards the target point and just stop. The target point location is not the issue here, being able to calibrate the finest level of resolution possible for any given target point is the goal.

If a good walking system is used, that is enough, the minimum velocity can be considered calibrated. However, if the Aldebaran walk is used, the delay needs to be accounted for. As we just reduced the size of the target box to be a magnitude near the step length, and we know the robot will take one last step that is equal to the last step length, then we can simply double the size of the just found target box value and it should then be triggered too early. The final extra step will then hit the target point as it will be ~half the size of the box, and therefore land in the center.

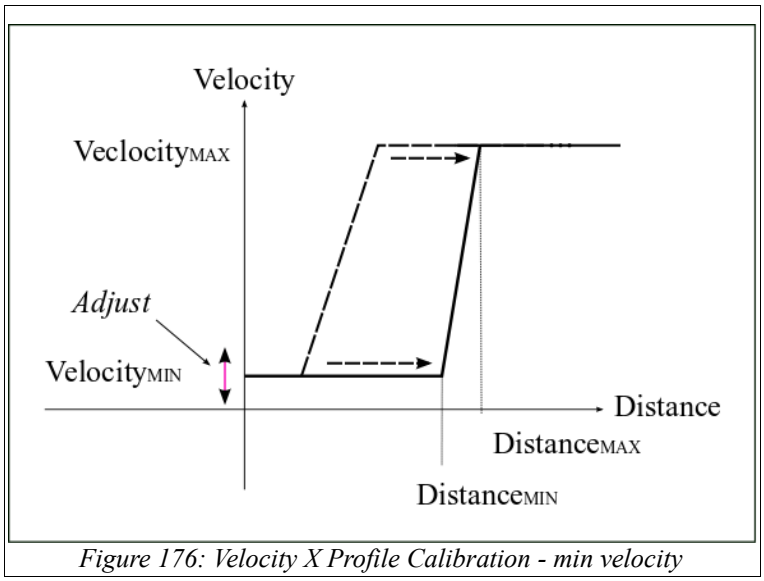


Figure 176: Velocity X Profile Calibration - min velocity

Step 3 Line Up Transition Distance

The next value to adjust will be the Line Up transition distance. This needs to be as close to the ball as possible, in order to limit the number of final correction steps, as no further orientation correction will be made. This should be a point where the robot is getting very close to the ball. About 5 cm in front of the toes. It can be visibly observed by watching for lateral steps. The robot will approach the ball and then at some point step outward. This action should be performed as late as possible.

Step 4 Minimum Velocity Range

With the minimum step length found we can next set the distance at which these values will be used (Figure 177). Again this will done in both dimensions X and Y with most attention being paid to the Y dimension. The X dimension is a matter of over running the ball. When this close to the ball the X dimension will nearly be satisfied, but the Y will not be.

To begin we will focus on the Y dimension. Here we need some realistic larger step length values. Move the maximum distance value down to something that will cause a steeper slope but not too steep. Both the minimum and maximum will have to be adjusted together to find good larger steps that are in the order of the error size but always smaller. This is the most sensitive part of the calibration as the greatest speed of execution will be determined here. Ideally we want the smallest step size to be forced with only the last step and the repeating delayed step. So only two small steps. Keep repeating this process until the robot walks up into the Line Up transition and then steps outward for the hard coded straight kick target point. This should be one large lateral step, followed by two small steps. Adjusting both the minimum and maximum distance will provide the correct Y gain slope to achieve this. With those values found the Y dimension calibration is completed. The straight shooting target point and the 90 degree point are the furthest points outward and is therefore the best to calibrate for. If at run-time the 45 degree kick was selected, then the ball should be between the feet and only two steps should then be required which is the best that can be achieved.

This all will have been executed quite slowly as the X dimension minimum distance is still set to a large value. The next thing to find is again the minimum distance in the velocity X profile and delay here is also a factor. This is the value that will control overrunning the ball. Here we want large step sizes for fast execution, but not too large as the ball will be overrun. The issue here is not a matter of going to far forward and then having to back track, but that the target line of data point is right in front of the toes so one over step, due to delay, will cause the ball to be nudged out of place and this is after the target point has been latched. There is some latitude here though, it will not require the same level of fine tuning as the Y dimension as this forward dimension will converge first usually. Setting the value to some distance just below the target data line will do ~ 3 or 4 cm back behind the ball.

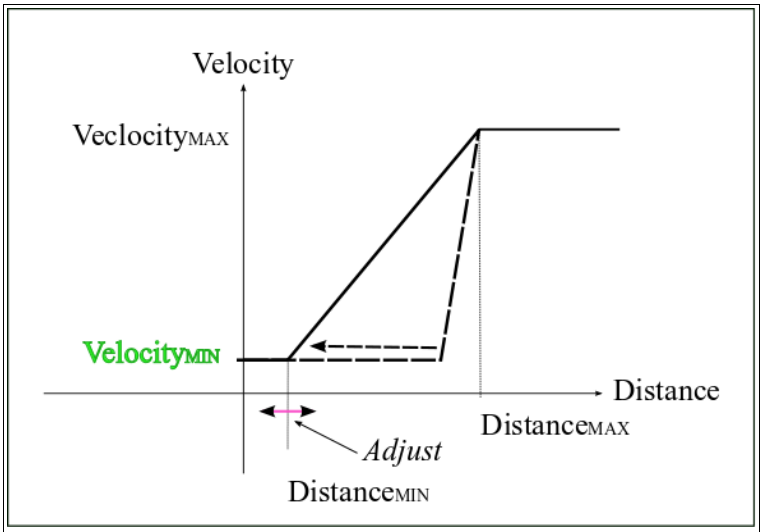


Figure 177: Velocity X Profile Calibration - min distance

Step 5 The Deceleration Range

With all those parameters locked in, that just leaves one, the maximum distance in the X dimension (Figure 178). We have at this point set the latest stage convergence speed and accuracy but all the tests will have been performed quite slowly and the effect will have been clearly observable. All we have to do now is set the declaration zone. This is just a matter of reducing the distance at which the deceleration begins. This can be set to a very steep value, just keep reducing it until ball over run is being risked, it will not affect any of the other system attributes. For really excellent 'stop on a dime' performance, the minimum distance can be increased very slightly and then reduce the maximum distance again slightly to provide an even steeper slope. This will not make the whole thing execute faster, as the Y dimension will still have to satisfied. It will appear as though the ball is going to be over run but at the last second it will suddenly slow down, terrifying the rest of the team members during a competition. As much as robots are a difficult system to work with, the silver lining

is that they will repeat the same actions over and over again reliably once properly calibrated (within noise ranges of course).

Once this deceleration value has been set, there will be no more observability when changing any other parameters (very little anyway). To do any recalibration, this value will always have to be opened up to slow the whole process down.

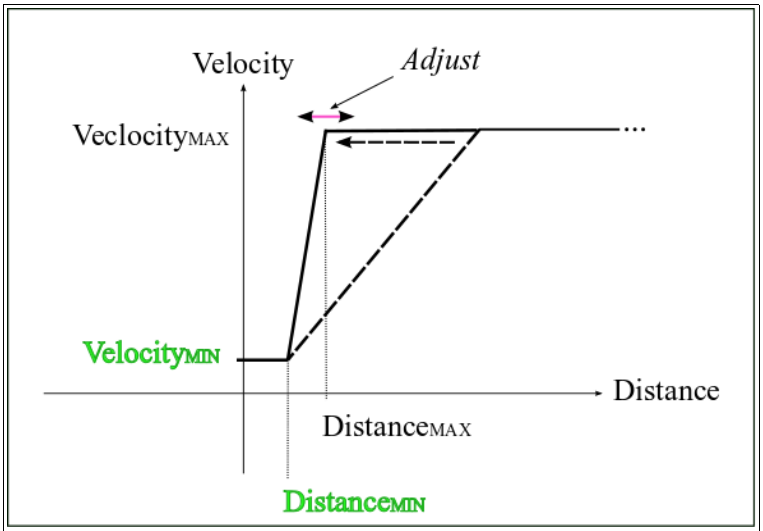


Figure 178: Velocity X Profile Calibration - max distance

One final important implementation point that has not been raised is that two velocity X profiles are used. One for the Approach Phase and one for the Line Up phase. The distances that will be passed in to the function will be different, one is the distance to the ball and the other is the size of the error between the ball and target point. This problem does not apply to the other two velocity profiles as they do not overlap. It should be noticed though that calibrating the X dimension was the easy one. Both velocity minimum values will be the same. The distance minimum value for the Line Up profile is not a hyper sensitive value, it is set roughly at 3-4 cm. We can just add the smallest targeting function distance (17 cm) to this value. So ~ 20 cm will match up well. That just leaves the maximum distance, the point to begin the declaration. Again, just add 17 cm for stop on a dime performance. For more conservative performance just add an equal value to both profiles. This may seem redundant but recall the target vector is not generated until the final Line Up phase.

Step 6 Encircle ball calibration

The last motion property to tune in the striker behaviour is the encircle ball phase. To this point we have approached the ball from the correct side, forcing a hard coded target point. Now we can remove the hard coded point and let the target function generator run. We now place the robot on the wrong side of the ball to initiate the encircling behaviour, setting the minimum kick angle to zero to force longer arcs to be traversed.

Here Y and Theta profiles can be used depending on the system. This is a matter of finding matching values that work for a particular radius. A range of velocities will work with their own combination of values. The same theta profile that is used for Approach Phase could be used. That would dictate a single lateral velocity that must be found and it may be a slow one. Better to build two instances of the theta velocity profiles. The only real use for the profile though would be to control the deceleration nicely, factoring in delay and final step sizes. This would only be an issue for straight kicking systems that are statically defined. The act of encircling is what accomplishes the setting of the shot accuracy. Effort would have to be placed on slowing down and stopping at an accurate angle inline with the goal and ball. Also, tight orientation control with the ball would be required while encircling. The same calibration technique used to line up the target point would be used here to converge on the target encircle angle. Set a smallest step size, set the error bound size and ensure the robot can walk directly into the minimized zone. Then calibrate a steep deceleration phase.

For angled kicking systems this is much easier, as simple gain transfer function will do. Also, the orientation lock on the ball does not have to be tight, oscillations are okay, it wont converge any faster or slower with a small amount of oscillation. The robot has a large range of kick angles it can swing into so no deceleration is required.

The only issue is if there is a delay in the walk. If so, the same dead band method used to straighten the walk is used to lock on to the ball while rotating around. Again oscillation due to a high

gain will not be as much of a problem as lagging rotations that can spin the robot quickly. Slowly reducing the dead zone down from a higher value should find a working value for a given rotation gain.

Another convenient way to parametrize this motion is to do it twice at different radii. With two values, a high one and a low one, interpolation can be used to generate corresponding rotation gains, Y velocities and dead band widths as a function of radius. This could be a convenient parameter to have on hand in a competition environment where alterations to the encircle radius may be desired for strategic purposes. With the radius parameter calibrated at high and low points, it could be changed at any time confidently, without worrying anything is going to break, 30 seconds before a match. This could be used to avoid opposing robots that happen to be using the same encircle distance and collisions keep occurring or multiple robots from the same team could be stacked up in different orbits around the ball for various kinds of defensive or obstructive plays.

The computer code implementation for these three velocity profiles are attached as Appendix J.

Approach To An Off center Target

As previously mentioned, this is a modification to Approach Minor state that lines up intended target points (Figure 179). This method has been implemented with the latest analytical kick system but would lend itself more suitably to the angled kick system, increasing the speed of the lateral convergence.

It is not a perfect solution however. As soon as it activates the target angle is latched and then a correction rotation occurs. That first correct rotation angle will steer the robot to a new heading and consequently change the intended kick angle. The correction angle will not translate directly into kick angles though. Right on the target line of target data points the 45 degree kick is right in between the feet and the straight shooting target point is in front of the foot which is 34 degrees to one side. That is the worst case scenario. The further back from the ball, the more the correction angle reduces. The further back to activate this the better. Another ring should be added to the behaviour diagram in Figure 172 to switch from centering the ball during Approach with a centered ball to an Approach that lines up target points. But there would be no gain in always doing it. The robot will be turning a fair bit as it walks around the field dodging obstructions and correcting for motion errors. There would be some distance that is a good compromise between being too close and too far away for it to do any good. Doing this as mentioned above, at the encircle ball ring, is a bit close but would still be pretty good.

The system should never be chasing free running target kick angle computations, the value has to be latched. If it chases a target that keeps updating the correction angle always steers the robot so that a larger target angle is generated. If the robot was walking to a shot that is straight ahead and it began to line up the ball in front of the foot, a sharper angle would be recomputed and then the robot turns away, ultimately always producing right angle kicks.

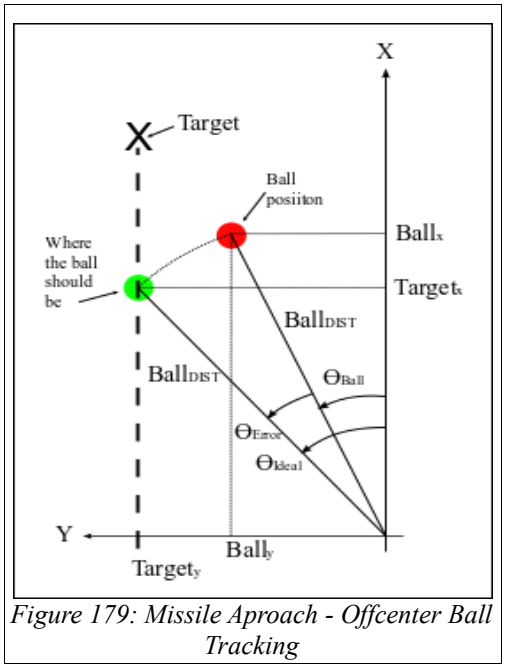


Figure 179: Missile Aproach - Offcenter Ball Tracking

Solution:

$$Ball_{DIST} = \sqrt{Ball_x^2 + Ball_y^2} \tag{380}$$

$$\theta_{Target} = \arcsin\left(\frac{fabs(Target_y)}{Ball_{DIST}}\right) \tag{381}$$

$$\theta_{Ball} = atan2(Ball_y, Ball_x) \tag{382}$$

$$\theta_{Error} = \theta_{Target} - \theta_{Ball} \tag{383}$$

1.6.4 Shot Accuracy

Finally, assess the shooting angle accuracy. First force the max kick angle to straight shooting only and verify that this is being performed. Pay no attention to whether the ball goes straight to the goal but see that the straight shooting target point itself is being properly lined up. The ball should travel out directly away from the robot. If this is not the case then there can be two known sources of problems. Some shifting offset my be required to move the set of target points around as run-time vision kinematics may differ slightly from those taken from a stationary pose during targeting calibration. If the shots are straight, then do the same, setting the kick angle to 45 degree. There is some latitude in the x dimension for the straight kick but not in the y dimension. Sorting out the y could still leave the x uncalibrated. Forcing the 45 degree will reveal if some x offset is necessary and adjust as required.

If at this point all previous tasks have been performed and the target points are being lined up erratically there is one last place to look, vision. Remove the offsets as they may have just been calibrated to attempt to adjust to some non kinematic problem. There could be a delay in the ball data. This can come in the form of vision data that is being filtered with a large window for noise or a world model ball system that models and dampens ball velocity. Other people work on these systems and they can do things that impact this system unknowingly. As an example of some such problem, in the latest RoboEireann build the ball velocity was adjusted to suit the goal keepers needs (false triggering a dive related issues). This method was applied to all the robots however, instead of just the goal keepers code.

Depending on available tools, identifying the delay can be a little tricky. One technique can be to record both the ball position and some kinematic data either be it the foot positions or simply one of the joints position data. Plot the data graphically and observe the end of the signal where the targeting system locks and the robot stops walking (do not forget to deactivate the kick swing). Do not record and check the stop walk command flag, there will most certainly be a delayed response to this command of some magnitude. Check that once the motion has actually come to a halt, if there is any continual ball motion. When the joints stop moving so should the ball. Do whatever it takes to get the ball position delay down as small as possible. A trade off may need to be made if 'vision ball' is used and the size of the ball location error bound as a result of the peak to peak noise value. Equalizing them is the best that can be done. Kalman filter modelling should be a lot more accurate and no intentional ball dampening should exist for the striker. But systems vary so both should be reviewed for what is best. It may even turn out that using one dimension from one ball method (vision) and the other dimension from the other method (modelling) works best.

The second possibility is that the vision system may have been built for localization and a fine level of accuracy was not required. In particular the sway of the robot could have been overlooked. There should be no sway in the ball data. Check the lateral ball position data signal for a ~1-2 Hz riding signal. That should be eliminated completely. Edit the vision kinematics as required to account for the upper body lateral motion. Using static kick swing means the target data is relative to the robot in the initial position when the camera has no lateral offset.

With that delay analysis competed, reiterate the target data offset calibration if necessary.

1.6.5 Conclusion

With all of this completed, the results should be very good.

Also, with all the sources of errors, they can be either positive or negative. So probabilistically, often they would cancel each other out and a perfect shot will be pulled off. In the unlucky case they could all swing positive or negative at the same time. So no one single aspect to the system should be shrugged off, leaving the error to overshadow all efforts to tune the rest of the system. However, if the same wrong thing happens over and over again, it probably is not accumulated error, there is a bug.

Is all this necessary? In RoboEireanns 2011 competition ending match there were only two shooting opportunities as a result of a myriad of other game play and code bug related issues. Both from mid field. One where the robot lined up a shot accurately and punched a defender robot that was quickly approaching from the side while swinging the kick. That contact threw the robot out of position. Less than one second would have made the difference and, given the quality of the keeper in the goal, likely the whole outcome of the game and continued ladder play was decided. The second shot was a clear shot on a break away with 20 seconds left on the clock. Given the system was only partially calibrated (keeper ball motion filtering was in the striker code), the shot just missed the goal. Again, small errors have big consequences.

References

[1]: Robocup Soccer League, 2011, http://www.robocup.org/robocup-soccer
[2]: G. FALLIS, *Fallis*, Jan. 17 1888. US Patent 376,588
[3]: T. McGeer, *Passive dynamic walking*, The International Journal of Robotics Re-search, vol. 9, no. 2, pp. 62-82, 1990
[4]: M. Vukobratovic and D. Juricic, *Contribution to the synthesis of biped gait*, Biomedical Engineering, IEEE Transactions on, no. 1, pp. 1-6, 1969
[5]: I. Kato, *The hydraulically powered biped walking machine with a high carrying capacity*, in Proc. of the 4th Int. Symposium on External Control of Human Ex-tremities, Dubrovnik, pp. 410-421, 1972
[6]: Aalborg University, *AAUBOT*, http://www.aaubot.aau.dk/
[7]: Boston Dynamics, *PETMAN*, http://www.bostondynamics.com/robot_petman.html
[8]: Artifical Intelligence Laboratory, *ROBOY*, http://www.roboy.org/
[9]: ASIMO, http://world.honda.com/ASIMO/
[10]: Kawada Industries, *HRP-2 PROMET*, http://global.kawada.jp/mechatronics/hrp2.html
[11]: Istituto Italiano di Tecnologia, *iCUB*, http://www.icub.org/
[12]: Trossen Robotics, *Biloid Kits*, http://www.trossenrobotics.com/bioloid-comprehensive-robot-kit.aspx
[13]: Robotis, *Dinemixel Motors*, http://www.robotis.com/xe/dynamixel_en
[14]: Del Pobil, A.P., *Why do we need benchmarks in robotics research?*, In: Proceedings of the IROS 2006 Workshop on Benchmarks in Robotics Research, Beijing (2006)
[15]: Sven Behnke, *Robot Competitions - Ideal Benchmarks for Robotics Research*, In Proceedings IROS-2006 Workshop on Benchmarks in Robotics Research, Beijing, October 2006
[16]: Robocup, http://www.robocup.org/
[17]: Stelian Coros, Philippe Beaudoin, and Michiel van de Panne, *Generalized biped walking control*, ACM Trans. Graph. 2010
[18]: P. H. Kahn, H. Ishiguro, B. Friedman, and T. Kanda, *What is a human? – Toward psychological benchmarksin the field of human-robot interaction*, In IEEE Proceedings of the International Workshop on Robot andHuman Interactive Communication (RO-MAN), Hatfield, UK, Sep 2006
[19]: Schafer, R C., *Clinical biomechanics : musculoskeletal actions and reactions*, Published: Baltimore : Williams & Wilkins, c1983. NLM ID: 8302418
[20]: Saunders, Inman and Eberhart , *The major determinants in normal and pathological gait*, J. Bone & Jt. Surg., 35-A(3):543-558, 1953
[21]: S. Collins, A. Ruina, R. Tedrake, and M. Wisse, *Efficient bipedal robots based on passive-dynamic walkers*, Science, vol. 307, no. 5712, pp. 1082-1085, 2005
[22]: AlGheshyan F. N., *Comparison of Ground Reaction Force in Treadmill Walking and in Overground Walking*, University of Miami Scholarly Repository, Electronic Theses and Dissertations, 2012-05-07
[23]: Goswami A. , *Postural Stability of Biped Robots and the Foot-Rotation Indicator (FRI) Point*, The International Journal of Robotics Research, 18(6):523 -533, June 1999
[24]: Vukobratovic M., Borovac B., Surdilovic D., *Zero-Moment Point – Proper Interpretation and new application*, in Proceedings of the IEEE/RAS International Conference on Humanoid Robots, 2001
[25]: NASA, *Man-stsems Integration Standards*, Technical Report, NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 1995
[26]: I. Mizuuchi, Y. Nakanishi, Y. Sodeyama, Y. Namiki, T. Nishino, N. Muramatsu,J. Urata, K. Hongo, T. Yoshikai, and M. Inaba, *An advanced musculoskeletal humanoid kojiro*, in Humanoid Robots, 2007 7th IEEE-RAS International Conferenceon, pp. 294-299, IEEE, 2007
[27]: M. Osada, N. Ito, Y. Nakanishi, and M. Inaba, *Realization of flexible motion by musculoskeletal humanoid "Kojiro" with add-on nonlinear spring units*, in HumanoidRobots (Humanoids), 2010 10th IEEE-RAS International Conference on, pp. 174-179, IEEE, 2010
[28]: S. Tzafestas, M. Raibert, and C. Tzafestas, *Robust sliding-mode control applied to a 5-link biped robot*, Journal of Intelligent & Robotic Systems, vol. 15, no. 1,pp. 67-133, 1996
[29]: C. Chevallereau and P. Sardain, *Design and actuation optimization of a 4-axes bipedrobot for walking and running*, in Robotics and Automation, 2000. Proceedings.ICRA'00. IEEE International

Conference on, vol. 4, pp. 3365-3370, IEEE, 2000
[30]: J. Pratt, C. Chew, A. Torres, P. Dilworth, and G. Pratt, *Virtual model control: Anintuitive approach for bipedal locomotion*, The International Journal of RoboticsResearch, vol. 20, no. 2, pp. 129-143, 2001
[31]: D. L. Wight, *A Foot Placement Strategy for Robust Bipedal Gait Control*, PhD thesis,University of Waterloo, 2008
[32]: K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, *The development of Honda humanoid robot*, in Robotics and Automation, 1998. Proceedings. 1998 IEEE Inter-national Conference on, vol. 2, pp. 1321-1326, IEEE, 1998
[33]: I. Park, J. Kim, J. Lee, and J. Oh, *Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot 3: HUBO)*, in Humanoid Robots, 2005 5th IEEE-RAS International Conference on, pp. 321-326, IEEE, 2005
[34]: K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi, *Humanoid robot HRP-3*, in Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ Interna-tional Conference on, pp. 2471-2478, IEEE, 2008
[35]: Y. Ogura, H. Aikawa, K. Shimomura, A. Morishima, H. Lim, and A. Takanishi, *Development of a new humanoid robot WABIAN-2*, in Robotics and Automation,2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pp. 76-81,IEEE, 2006
[36]: J. Yamaguchi, E. Soga, S. Inoue, and A. Takanishi, *Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking*, inRobotics and Automation, 1999. Proceedings. 1999 IEEE International Conferenceon, vol. 1, pp. 368-374, IEEE, 1999
[37]: Denavit, Jacques; Hartenberg, Richard Scheunemann, *A kinematic notation for lower-pair mechanisms based on matrices*, Trans ASME J. Appl. Mech 23: 215--221. 1955
[38]: Craig, J.J,, *Introduction to Robotics: Mechanics and Control*, Reading, Mass.: Addison-Wesley, 1986
[39]: Welman, C., *Inverse Kinamatics and Geometric Constraints for Articulated Figure Manipualtion*, Masters Thesis, Simon Fraser University, April 1993
[40]: Colin Graf, Er Hartl, Thomas Rofer, and Tim Laue, *A robust closed-loop gait for the Standard Platform League Humanoid*, In Proceedings of the 4th Workshop on Humanoid Soccer Robots. A workshop of the 2009 IEEE-RAS Intl. Conf. On Humanoid Robots, 2009
[41]: Microsoft, *Microsoft Robot Developer Studio*, http://www.microsoft.com/robotics/
[42]: Loken K., *Imitation-based Learning of Bipedal Walking UsingLocally Weighted Learning*, Masters Thesis, The University Of British Columbia, August 2006
[43]: Kuo, A. D., Donelan, J. M., and Ruina, A., *Energetic consequences of walking like an inverted pendulum: Step-to-step transitions*, Exercise and Sport Sciences Reviews, 33: 88-97, 2005
[44]: Christensen, R., Fogh, N., Hansen, R. H., Hansen, H., Iversen, A. M., Jensen, M. S., and Lantow, L. S., *Modelling and Control of a Biped Robot*, Technical report. Aalborg University. Project group: 07gr830 , 2007b
[45]: Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, *The intelligent ASIMO: System overview and integration*, in Intelligent Robots andSystems, 2002. IEEE/RSJ International Conference on, vol. 3, pp. 2478-2483, IEEE,2002
[46]: Benjamin Stephens, *Push Recovery Control for Force-Controlled Humanoid Robots*, doctoral dissertation, tech. report CMU-RI-TR-11-15, Robotics Institute, Carnegie Mellon University, May, 2011
[47]: M. Wisse and J. Van Frankenhuyzen, *Design and construction of mike; a 2D autonomous biped based on passive dynamic walking*, in Proceedings of internationalsymposium of adaptive motion and animals and machines (AMAM03), 2003
[48]: D. Hobbelen and M. Wisse, *Ankle joints and at feet in dynamic walking*, Climbingand Walking Robots, pp. 787-800, 2005
[49]: D. Hobbelen, T. de Boer, and M. Wisse, *System overview of bipedal robots ameand tulip: Tailor-made for limit cycle walking*, in Intelligent Robots and Systems,2008. IROS 2008. IEEE/RSJ International Conference on, pp. 2486-2491, IEEE,2008
[50]: M. Wisse, G. Feliksdal, J. Van Frankkenhuyzen, and B. Moyer, *Passive-based walking robot*, Robotics & Automation Magazine, IEEE, vol. 14, no. 2, pp. 52-62, 2007
[51]: M. Wisse and R. van der Linde, *Delft Pneumatic Bipeds*, Springer Tracts in AdvancedRobotics, Springer, 2007
[52]: G. Bilodeau and E. Papadopoulos, *A model-based impedance control scheme for high-performance hydraulic joints*, in Intelligent Robots and Systems, 1998. Pro-ceedings., 1998 IEEE/RSJ International Conference on, vol. 2, pp. 1308-1313, IEEE,1998.
[53]: S. Hyon, J. Hale, and G. Cheng, *Full-body compliant human-humanoid interaction:balancing*

in the presence of unknown external forces, Robotics, IEEE Transactionson, vol. 23, no. 5, pp. 884-898, 2007
[54]: P. Ill-Woo, K. Jung-Yup, L. Jungho, and O. Jun-Ho, *Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot - 3: HUBO)*, In 5th IEEE-RAS International Conference on Humanoid Robots, 2005., volume 12, pages 321-326. IEEE,2005
[55]: K. Akachi, K. Kaneko, N. Kanehira, S. Ota, G. Miyamori, M. Hirata, S. Kajita, andF. Kanehiro, *Development of humanoid robot HRP-3P*, In Humanoid Robots, 2005 5th IEEE-RAS International Conference on, pages 50-55, 2005
[56]: A. Takanishi, T. Takeya, H. Karaki, and I. Kato, *A control method for dynamic biped walking under unknown external force*, In IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, volume 29,pages 795-801. IEEE, 1990
[57]: K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, *Planning walking patterns for a biped robot*, IEEE Transactions on Robotics and Automation, 17(3) :280-289, June 2001
[58]: K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, *Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired ZMP*, In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System, volume 3, pages 2684-2689, 2002
[59]: J. Pratt, B. Krupp, and C. Morse, *Series elastic actuators for high fidelity force control*, Industrial Robot: An International Journal, 29(3):234-241, 2002
[60]: Shin D., Sardellitti I., Khatib O., *A Hybrid Actuation Approach for Human-Friendly Robot Design*, Robotics and Automation. ICRA '09. IEEE International Conference, May 2009
[61]: J. Pratt, P. Dilworth, and G. Pratt, *Virtual model control of a bipedal walking robot*, Proceedings of the IEEE International Conference on Robotics and Automation, 1:193-198, Apr. 1997
[62]: J. H. Park, *Impedance Control for Biped Robot Locomotion*, IEEE Transactions on Robotics and Automation, 17:870-881, Dec. 2001
[63]: K. Yamane and Y. Nakamura, *Dynamics Filter - concept and implementation of online motion Generator for human figures*, IEEE Transactions on Robotics and Automation, 19:421-432, June 2003
[64]: Vaughan C.L., *Theories of bipedal walking: an odyssey*, J Biomech. 2003 Apr;36(4):513-23. Review
[65]: T. McGeer, *Passive walking with knees*, in Robotics and Automation, 1990. Pro-ceedings., 1990 IEEE International Conference on, pp. 1640-1645, IEEE, 1990
[66]: M. Garcia, A. Chatterjee, and A. Ruina, *Efficiency, speed, and scaling of two dimensional passive-dynamic walking*, Dynamics and Stability of Systems, vol. 15,no. 2, pp. 75-99, 2000
[67]: S. Collins, M. Wisse, and A. Ruina, *A three-dimensional passive-dynamic walking robot with two legs and knees*, The International Journal of Robotics Research,vol. 20, no. 7, pp. 607-615, 2001
[68]: S. H. Collins, M. Wisse and A. Ruina, *A two legged kneed passive dynamic walking robot*, Int. J. Robotics Research 20, 607–615 July, 2001
[69]: I. R. Manchester, U. Mettin, F. Iida, and R. Tedrake, *Stable dynamic walking ove runeven terrain*, The International Journal of Robotics Research, 30(3):265-279, Jan. 2011
[70]: Garcia, M., Chatterjee, A., Ruina, A., Coleman, M., *The simplest walking model: stability, complexity, and scaling*, Journal of Biomechanical Engineering 120 (2), 281–288, 1998
[71]: M. Wisse, A. L. Schwab, R. Q. van der Linde, *A 3D Passive Dynamic Biped with Yaw and Roll Compensation*, Robotica. V 19, p.p.: 275-284. 2001
[72]: A. Kuo, *Stabilization of lateral motion in passive dynamic walking*, The International Journal of Robotics Research, vol. 18, no. 9, pp. 917-930, 1999
[73]: Hobbelen, D., Wisse, M., *Limit cycle walking*, Humanoid Robots, Human-likeMachines (2007)
[74]: McGhee, R., *Some finite state aspects of legged locomotion*, Mathematical Biosciences 2, 67–84, 1968
[75]: Zheng, Y.F., *Acceleration compensation for biped robots toreject external disturbances*, IEEE Transactions on Systems, Man,and Cybernetics 19 (1), 74–84, 1989
[76]: F. Asano, M. Yamakita, and K. Furuta, *Virtual passive dynamic walking andenergy-based control laws*, in Intelligent Robots and Systems, 2000.(IROS 2000).Proceedings. 2000 IEEE/RSJ International Conference on, vol. 2, pp. 1149-1154,IEEE, 2000
[77]: A. Goswami, B. Espiau, and A. Keramane, *Limit cycles and their stability in a passive bipedal gait*, in Robotics and Automation, 1996. Proceedings., 1996 IEEEInternational Conference on, vol. 1, pp. 246-251, IEEE, 1996
[78]: M. Spong et al., *Passivity based control of the compass gait biped*, in Proc. ofIFAC World Congress, Beijing, China, 1999
[79]: F. Asano, Z. Luo, and M. Yamakita, *Some extensions of passive walking formula to active biped robots*, in Robotics and Automation, 2004. Proceedings. ICRA'04.2004 IEEE International

Conference on, vol. 4, pp. 3797-3802, IEEE, 2004
[80]: S. Anderson, M. Wisse, C. Atkeson, J. Hodgins, G. Zeglin, and B. Moyer, *Powered bipeds based on passive dynamic principles*, in Humanoid Robots, 2005 5th IEEE-RAS International Conference on, pp. 110-116, IEEE, 2005
[81]: G. Pratt and M. Williamson, *Series elastic actuators*, in Intelligent Robots andSystems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995IEEE/RSJ International Conference on, vol. 1, pp. 399-406, IEEE, 1995
[82]: A. Kuo et al., *Energetics of actively powered locomotion using the simplest walking model*, Transactions-american Society Of Mechanical Engineers Journal Of Biome-chanical Engineering, vol. 124, no. 1, pp. 113-120, 2002
[83]: D. Hobbelen and M. Wisse, *Ankle actuation for limit cycle walkers*, The Interna-tional Journal of Robotics Research, vol. 27, no. 6, pp. 709-735, 2008
[84]: D. Hobbelen and M. Wisse, *Controlling the walking speed in limit cycle walking*, The International Journal of Robotics Research, vol. 27, no. 9, pp. 989-1005, 2008
[85]: F. Asano, Z. Luo, and M. Yamakita, *Unication of dynamic gait generation methods via variable virtual gravity and its control performance analysis*, in IntelligentRobots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ InternationalConference on, vol. 4, pp. 3865-3870, IEEE, 2004
[86]: Choudhury S., *Design and Gait Synthesis for a3D Lower Body Humanoid*, Masters Thesis, University of Waterloo, Ontario, Canada, 2012
[87]: S. Hashimoto, S. Narita, H. Kasahara et al, *Humanoid Robots in Waseda University Hadaly-2 and WABIAN*, Auton. Robots 12(1) 25–38, 2002
[88]: Ude A., Atkeson C. G., Riley M., *Planning of Joint Trajectories for Humanoid Robots Using B_Spline Wavelets*, In Proc. IEEE Int. Conf. Robotics and Automation, San Francisco, California, pp. 2223-2228, April 2000
[89]: Hein D., Hild M., Berger R., *Evolution of biped walking using neural oscillators and physical simulation*, In RoboCup 2007
[90]: Moro F. L., Tsagarakis N. G. and Caldwell D. G. , *On the Kinematic Motion Primitives (kMPs) – Theory and Application* , Front Neurorobot. ;vol. 6, Oct. 2012
[91]: Wehner S., Bennewitz M., *Optimizing the Gait of a Humanoid RobotTowards Human-like Walking*, In 4th European Conference on Mobile Robots, Mlini/Dubrovnik, Croatia. 2009
[92]: Do M., Gehrig D., Kühne H., Azad P., Pastor P., Asfour T., Schultz T., Wörner A., Dillmann R., *Transfer of Human Movements to Humanoid Robots*, 8th IEEE-RAS International Conference on Humanoid Robots, Workshop Imitation and Coaching in Humanoid Robots, 2008
[93]: Sangwan, V.; Agrawal, S.K., *Differentially Flat Design of Bipeds Ensuring Limit Cycles*, Mechatronics, IEEE/ASME Transactions on , vol.14, no.6, pp.647-657, Dec. 2009
[94]: Kuo A. D., Donelan J. M., *Dynamic Principles of Gait and Their Clinical Implications*, Journal of the American Physical Therapy Association, 90(2): 157–174. Feb 2010
[95]: Carey T., Crompton R. H., *The metabolic costs of 'bent-hip, bent-knee' walking in humans*, Journal of Human Evolution 48, pp 25-44, 2005
[96]: Iida F, Rummel j, Seyfarth A, *Bipedal walking and running with spring-like biarticular muscles*, J. Biomech. 41: 656-667, 2008
[97]: Geyer, H., Seyfarth, A., and Blickhan, R., *Compliant leg Behaviour Explains basic Dynamics of Walking and Running*, Proceedings of the Royal Society B, 273, 2861-2867. 2006
[98]: Borghese N. A., Bianchi L. and Lacquaniti F., *Kinematic determinants of human locomotion*, Journal of Physiology, 494.3, pp.863-879, 1996
[99]: Zijlstra W., Hof A., *Displacement of the pelvis during human walking: experimental data and model predictions*, Gait and Posture, vol 6, Issue 3, p 249- 267, Jan 1997
[100]: Sakka S., Hayot C, and Lacouture P., *A generalized 3D inverted pendulum model to represent human normal walking*, 2010 IEEE-RAS International Conference on Humanoid Robots Nashville, TN, USA, December 6-8, 2010
[101]: Hayot C., Sakka S., and Lacouture P., *Eip-3d: Anextended 3d inverted pendulum model to represent normal humanwalking cycle*, Gait & Posture, 2010
[102]: Kuo A. D., *The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective*, Human Movement Science 26, pp 617–656, 2007
[103]: S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, *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 InternationalConference on, vol. 1, pp. 239-246, IEEE, 2001
[104]: S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, andH. Hirukawa, *Biped Walking Pattern Generation by using Preview Control of Zero Moment Point*, in Robotics and

Automation, 2003. Proceedings. ICRA'03. IEEEInternational Conference on, vol. 2, pp. 1620-1626, IEEE, 2003
[105]: Suleiman, W., Kanehiro, F., Miura, K. & Yoshida, E., *Generating Dynamically Stable Walking Patterns for Humanoid Robots Using Quadratic System Model* , Proc. of 2010 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics, 2010
[106]: S. Kudoh and T. Komura, *C'2 continuous gait-pattern generation for biped robots*, In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, pages 1135-1140, Oct. 2003
[107]: R. J. Full and D. E. Koditschek, *Templates and anchors: neuromechanical hypotheses of legged locomotion on land*, The Journal of experimental biology, 202(Pt 23):3325 -32, Dec. 1999
[108]: Weber W., Weber E., *Mechanik der menschlichen Gehwerkzeuge*, In Dietrichsche Buchhandlungen 1836 Göttingen, Germany:Dietrichsche Buchhandlungen
[109]: Lee C., Farley C., *Determinants of the center of mass trajectory in human walking and running*, J. Exp. Biol. 201, 2935–2944, 1998
[110]: Marey E., *Le mouvement*, In Masson 1894 Paris, France:Masson
[111]: Fischer O., *vol. 25 1899 Leipzig, Germany:Sächsischen Gesellschaft der Wissenschaften*,
[112]: Roberts T. J., Azizi E., *Flexible mechanisms: the diverse roles of biological springs in vertebrate movement*, journal of Experimental Biology 214, 353-361. Feb 2011
[113]: Cavagna G., Saibene F., Margaria R., *Mechanical work in running*, J. Appl. Physiol. 19, 249–256, 1964
[114]: Mochon S., McMahon T., *Ballistic walking*, J. Biomech. 13, 49–57, 1980
[115]: Dickinson M., Farley C., Full R., Koehl M., Kram R., Lehman S., *How animals move: an integrative view*, Science. 288, 100–106, 2000
[116]: Minetti A., Ardigo L., Reinach E., Saibene F., *The relationship between mechanical work and energy expenditure of locomotion in horses*, J. Exp. Biol. 202, 2329–2338, 1999
[117]: Alexander R., *Mechanics of bipedal locomotion*, In Perspectives in experimental biology Davies P.S, pp. 493–504. Eds. Oxford, UK:Pergamon Press, 1976
[118]: Fukunaga T., Kubo K., Kawakami Y, Fukashiro S., Kanehisa H., Maganaris C., *In vivo behaviour of human muscle tendon during walking*, Proc. R. Soc. B. 268, 229–233, 2001
[119]: Siegler S., Seliktar R., Hyman W., *Simulation of human gait with the aid of a simple mechanical model*,
[120]: Gurp M.V, Schamhardt H., Crowe A., *The ground reaction force pattern from the hindlimb of the horse simulated by a spring model*, Acta Anat. 129, 31–33, 1987
[121]: Pandy M., Berme N., *Synthesis of human walking: a planar model for single support*, J. Biomech. 21, 1053–1060, 1988
[122]: Zajac F., Neptune R., Kautz S., *Biomechanics and muscle coordination of human walking part II: lessons from dynamical simulations and clinical implications*, Gait Posture. 17, 1–17, 2003
[123]: Srinivasan M., Ruina A., *Computer optimization of a minimal biped model discovers walking and running*, Nature. 439, 72–75, 2005
[124]: M. Vukobratovic and B. Borovac, *Zero-moment point thirty five years of its life*, International Journal of Humanoid Robotics, vol. 1, no. 01, pp. 157-173, 2004
[125]: Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, *Planning walking patterns for a biped robot*, Robotics and Automation, IEEETransactions on, vol. 17, no. 3, pp. 280-289, 2001
[126]: S. Kajita, M. Morisawa, K. Harada, K. Kaneko, F. Kanehiro, K. Fujiwara, andH. Hirukawa, *Biped walking pattern generator allowing auxiliary zmp control*, in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on,pp. 2993-2999, IEEE, 2006
[127]: Katatama T., Ohki T, Inoue T., Kato T., *Design of an optimal controller for a discrete-time system subject to previewable demand*, Int. J. Control, Vol. 41, No. 3, 677-699, 1985
[128]: T. Takenaka, T. Matsumoto, and T. Yoshiike, *Real time motion generation and control for biped robot - 1st report: Walking gait pattern generation*, in Intelli-gent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conferenceon, pp. 1084-1091, IEEE, 2009
[129]: P. Wieber, *Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations*, in Humanoid Robots, 2006 6th IEEE-RASInternational Conference on, pp. 137-142, IEEE, 2006
[130]: H. Diedam, D. Dimitrov, P.-B. Wieber, K. Mombaur, and M. Diehl, *Online Walking Gait Generation with Adaptive Foot Positioning Through Linear Model Predictive Control*, In Proceedings of the International Conference on Intelligent Robots and Systems, pages 1121-1126. IEEE, 2008
[131]: J. Chestnutt, M. Lau, K. M. Cheung, J. Kuffner, J. K. Hodgins, and T. Kanade, *Footstep Planning for the Honda ASIMO Humanoid*, In Proceedings of the IEEE International Conference on

Robotics and Automation, Apr. 2005
[132]: Strom, J., Slavov, G., Chown, E., *Omnidirectional Walking using ZMP and Preview Controlfor the NAO Humanoid Robot*, In Proceedings of RoboCup. 2009, 378-389.
[133]: Pratt, J., Tedrake, R., *Velocity-based stability margins for fast bipedal walking*, First Ruperto Carola symposium: fast motions in biomechanics and robots
[134]: Townsend M. A. , *Biped gait stabilization via foot placement*, Journal of Biomechanics, 18(1):21—38, 1985
[135]: J. Englsberger, C. Ott, M. Roa, A. Albu-Schaer, and G. Hirzinger, *Bipedal walking control based on Capture Point dynamics*, in Intelligent Robots and Systems (IROS),2011 IEEE/RSJ International Conference on, pp. 4420-4427, IEEE, 2011
[136]: M. Krause, J. Englsberger, P. Wieber, and C. Ott, *Stabilization of the Capture Point Dynamics for Bipedal Walking Based on Model Predictive Control*, in RobotControl, vol. 10, pp. 165-171, 2012
[137]: T. De Boer, *Foot placement in robotic bipedal locomotion*, PhD thesis, Delft University of Technology, 2012
[138]: T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, *Capturability-based analysis and control of legged locomotion, Part 1: Theory and application to three simple gait models*, The International Journal of Robotics Research, vol. 31, no. 9,pp. 1094-1113, 2012
[139]: J. Pratt and B. Krupp, *Design of a bipedal walking robot*, in SPIE Defense andSecurity Symposium, pp. 69621F{69621F, International Society for Optics and Photonics, 2008
[140]: J. Pratt, T. Koolen, T. De Boer, J. Rebula, S. Cotton, J. Car, M. Johnson, andP. Neuhaus, *Capturability-based analysis and control of legged locomotion, Part 2: Application to M2V2, a lower-body humanoid*, The International Journal ofRobotics Research, vol. 31, no. 10, pp. 1117-1133, 2012.
[141]: D. Wight, E. Kubica, and D. Wang, *Introduction of the foot placement estimator: A dynamic measure of balance for bipedal robotics*, Journal of computational andnonlinear dynamics, vol. 3, no. 1, 2008
[142]: Strogatz S. H., *Nonlinear Dynamics and Chaos*, Westview Press, Cambridge, MA, USA, 2000
[143]: Sten Grillner, *Neurobiological Bases of Rhythmic Motor Acts in Vertebrates*, Science 228(4696), 143–149 , 1985
[144]: Grillner Sten, *Neural Networks for Vertebrate Locomotion*, J-SCI-AMER 274(1), Jan 1996
[145]: Ijspeert A. J. , *The Handbook of Brain Theory and Neural Networks*, M. Arbib (Ed.). In: Locomotion, Vertebrate. 2nd edn. MIT Press, 649–654 , 2002
[146]: Grillner S., *Neurological bases of rhythmic motor acts invertebrates*, Science 228, 143–149, 1985
[147]: MacKay-Lyons M. , *Central pattern generation of locomotion: A review of the evidence*, Physical Therapy, 82(1):69–83, 2002
[148]: G. Taga, Y. Yamaguchi, and H. Shimizu, *Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment*, Biological cybernetics,vol. 65, no. 3, pp. 147-159, 1991
[149]: G. Taga, *A model of the neuro-musculo-skeletal system for anticipatory adjustment of human locomotion during obstacle avoidance*, Biological Cybernetics, vol. 78,no. 1, pp. 9-17, 1998
[150]: S. Miyakoshi, G. Taga, Y. Kuniyoshi, and A. Nagakubo, *Three dimensional bipedal stepping motion using neural oscillators-towards humanoid motion in the real world*, in Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ InternationalConference on, vol. 1, pp. 84-89, IEEE, 1998
[151]: A. Ijspeert, *2008 Special Issue: Central pattern generators for locomotion control in animals and robots: A review*, Neural Networks, vol. 21, no. 4, pp. 642-653, 2008
[152]: A.J. Ijspeert and J.-M. Cabelguen, *Gait Transition from Swimming to Walking: Investigation of Salamander Locomotion Control Using Nonlinear Oscillators*, Technical report, Swiss Federal Institute of Technology, 2002
[153]: Hillel J. Chiel, Randall D. Beer, John C. Gallagher, *Evolution and Analysis of Model CPGs for Walking I. Dynamical Modules*, Journal of Computational Neuroscience 7(2), 1999
[154]: Randall D. Beer, Hillel J. Chiel, John C. Gallagher, *Evolution and Analysis of Model CPGs for Walking: II. General Principles and Individual Variability*, Journal of Computational Neuroscience 7(2) 119–147 , 1999
[155]: Lacquaniti F, Ivanenko YP, Zago M, *Patterned control of human locomotion*, Journal of Physiology, 590(Pt 10):2189-99, May 2012
[156]: Kiyotoshi Matsuoka, *Mechanisms of Frequency and Pattern Control in the Neural Rhythm Generators*, Biological Cybernetics 56(5–6) 345–353 , 1987
[157]: Gen Endo, Jun Nakanishi, Jun Morimoto, and Gordon Cheng, *Experimental Studies of a Neural Oscillator for Biped Locomotion with QRIO*, In: IEEE 2005: International Conference on

Robotics & Automation. 2005
[158]: Akinobu Fujii, Akio Ishiguro and Peter Eggenberger, *Evolving a CPG Controller for a Biped Robot with Neuromodulation*, In: Proceedings of the 5th International Conference on Climbing and Walking Robots, Paris, France 17–24 , 2002
[159]: K. Endo, T. Maeno, and H. Kitano, *Co-Evolution of Morphology and Walking Pattern of Biped Humanoid Robot Using Evolutionary Computation - Consideration of Characteristic of the Servomotors*, In: IEEE/RSJ
[160]: Reil, T., Husbands, P., *Evolution of Central Pattern Generators for Bipedal Walking in a Real-Time Physics Environment*, IEEE Transactions on evolutionary computation, Vol. 6, No. 2, pp. 159-166. 2002
[161]: D. Katic and M. Vukobratovi, *Survey of intelligent control techniques for humanoid robots*, Journal of Intelligent & Robotic Systems, vol. 37, no. 2, pp. 117-141, 2003
[162]: W. Miller III, *Real-time neural network control of a biped walking robot*, ControlSystems, IEEE, vol. 14, no. 1, pp. 41-48, 1994
[163]: A. Kun and W. Miller III, *Adaptive dynamic balance of a biped robot using neural networks*, in Robotics and Automation, 1996. Proceedings., 1996 IEEE InternationalConference on, vol. 1, pp. 240-245, IEEE, 1996
[164]: J. Morimoto, G. Endo, J. Nakanishi, S. Hyon, G. Cheng, D. Bentivegna, and C. Atkeson, *Modulation of simple sinusoidal patterns by a coupled oscillator model for biped walking*, in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pp. 1579-1584, IEEE, 2006
[165]: L. Righetti and A. Ijspeert, *Programmable central pattern generators: an application to biped locomotion control*, in Robotics and Automation, 2006. ICRA 2006.Proceedings IEEE International Conference on, pp. 1585-1590, IEEE, 2006
[166]: S. Aoi and K. Tsuchiya, *Locomotion control of a biped robot using nonlinear oscillators*, Autonomous Robots, vol. 19, no. 3, pp. 219-232, 2005
[167]: A. D. Ames, R. D. Gregg, and M. W. Spong, *A geometric approach to three-dimensional hipped bipedal robotic walking*, In 45th Conference on Decision and Control, San Diago, CA, 2007
[168]: J. W. Grizzle, C. Chevallereau, A. D. Ames, and R. W. Sinnet, *3D bipedal robotic walking: models, feedback control, and open problems*, In IFAC Symposium on Nonlinear Control Systems, Bologna, 2010
[169]: E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris, *Feedback Control of Dynamic Bipedal Robot Locomotion*, CRC Press, Boca Raton, 2007
[170]: E. R. Westervelt, J. W. Grizzle, and D. E. Koditschek, *Hybrid zero dynamics of planar biped walkers*, IEEE TAC, 48(1):42–56, 2003
[171]: Kulk J., *Improved humanoid robot movement through impact perception and walk optimisation*, University of Newcastle Research Higher Degree Thesis, 2012
[172]: K. Yin, D. K. Pai, and M. van de Panne, *Data-driven Interactive Balancing Behaviors*, Pacific Graphics, Oct. 2005
[173]: da Silva M., Abe Y., and Popovic J., *Simulation of Human Motion Data using Short- Horizon Model-Predictive Control*, Computer Graphics Forum, 27(2):371-380, Apr. 2008
[174]: Sok K. W., Kim M., and Lee J., *Simulating Biped Behaviors from Human Motion Data*, In International Conference on Computer Graphics and Interactive Techniques, 2007
[175]: Tsai Y.-Y., Lin W.-C., Cheng K. B., Lee J., and Lee T.-Y., *Real-Time Physics-Based 3D Biped Character Animation Using an Inverted Pendulum Model*, IEEE transactions on visualization and computer graphics, 16(2):325-37, 2010
[176]: Mah CD., Hulliger M., Lee R.G., O'Callaghan I.S., *Quantitative analysis of human movement synergies: constructive pattern analysis for gait*, Journal of Motor Behavior, 26(2): p. 83-102, 1994
[177]: Chen Yo, Chang Jia-Hao, *An Investigation of Soccer Ball Velocity on Instep kick with and without Arm Swaying*, XXVIII International Symposium of Biomechanics in Sports, July 2010
[178]: Tedrake R., Fong M., Zhang, T.W., Seung H. S., *Actuating a Simple 3D Passive Dynamic Walker*, IN: IEEE International Conference on Robotics and Automation, vol. 5, pp. 4656-4661, 2004
[179]: Alcaraz-Jiménez J. J., M. Missura, Martínez-Barberá H., and Behnke S., *Lateral Disturbance Rejection for the Nao Robot*, In proceedings RoboCup Symposium 2012
[180]: A. D. Ames, E. A. Cousineau, M. J. Powell, *Dynamically Stable Robotic Walking with NAO via Human-Inspired Hybrid Zero Dynamics*, in Hybrid Systems: Computation and Control, 2012
[181]: Ames A.D., *Video*, http://www.youtube.com/watch?v=OBGHU-e1kc0/ Last accessed Feb 7 2012
[182]: http://www.youtube.com/watch?v=OBGHU-e1kc0/ Last accessed Feb 7 2012
[183]: Boulic R., Glardon P., Thalmann D., *From Measurements to Model: the Walk Engine*, Proc. of

6th Conf on Optical 3D Measurement Techniques, Zurich, Switzerland, 2003
[184]: Cairns M. A., Burdett R. G., Pisciotta J.C., Simon S. R., *A biomechanical analysis of racewalking gait*, Medicine and Science in Sports and Exercise Vol. 18, No. 4, 1985
[185]: Cunningham, Thomas J., *THREE-DIMENSIONAL QUANTITATIVE ANALYSIS OF THE TRAJECTORY OF THE FOOTWHILE RUNNING*, University of Kentucky Master's Theses. Paper 500, 2007
[186]: Sinning W. E. , H.L.F., *Lower-Limb actions while running at different velocities*, Medicine and Science in Sport, 2(1): p. 28-34, 1970
[187]: Dillman, C.J., *Kinematic Analyses of Running*, Exercise Sport Science Reviews, p. 193-216, 1975
[188]: Williams, K.R., *Biomechanics of Running*, Exercise Sport Science Reviews, p. 389-441, 1985
[189]: Weyand P. G., D.B.S., Bellizzi M. J., Wright S., *Faster top running speeds are achieved with greater ground forces not more rapid leg movements*, Journal of Applied Physiology, 2000. 89: p. 1991-1999
[190]: Barak Y, Wagenaar RC, Holt KG, *Gait characteristics of elderly people with a history of falls: A dynamic Approach*, Phys Ther. 86:1501–1510. 2006
[191]: Helbostad JL, Moe-Nilssen R, *The effect of gait speed on lateral balance control during walking in healthy elderly*, Gait Posture. 18(2):27-36. Oct 2003
[192]: Jason Kulk, , Castle University
[193]: Farley C.T., Gonzalez O, *Leg stiffness and stride frequency in human running*, J. Biomech. 1996 Feb; 29(2):181-6
[194]: Seyfarth A., Geyer H., Lipfert S., Rummel J., Minekawa Y., and Iida F., *Running and walking with compliant legs*, Fast Motions in Biomechanics and Robotics - Optimization and Feedback Control, Diehl M, Mombaur K (eds.). Springer Verlag, Berlin Heidelberg: 383-402. 2006
[195]: Xue F., Chen X., Liu J., Nardi D., *Real Time Biped Walking Gait Pattern Generator for a Real Robot*, In proceedings RoboCup 2011: Robot Soccer World Cup XV
[196]: Ando N., Balakirsky S., Hemker T., Reggiani M.,von Stryk O., *Simulation, Modeling and Programming for Autonomous Robots*, In Proceedings, Second International COnference, SIMPAR 2010. Darmstadt, Germany, November 2010
[197]: R. Tilgner, T. Reinhardt, D. Borkmann, T. Kalbitz,S. Seering, R. Fritzsche, C. Vitz, S. Unger,S. Eckermann, H. M¨uller, M. Bellersen, M. Engel, and M. Wunsch, *Team research report 2011*, Technicalreport, Leipzig University of Applied Sciences
[198]: Lees A, Nolan L., *The biomechanics of soccer: a review*, J Sports Sci. 1998 Apr;16(3):211-34
[199]: Ahmad Rasdan, Ismail , *Biomechanics Analysis And Optimization Of Instep Kicking: A Case Study To Malaysian Footballer*, In: National Conference in Mechanical Engineering Research and Postgraduate Students (1st NCMER 2010), 26-27 May 2010, FKM Conference Hall, UMP, Kuantan, Pahang, Malaysia
[200]: Vasquez M., Tan J., Saeed R., Patel A., *Kinesiology of a soccer kick*, http://www.docstoc.com/docs/74667475/Kinesiology-of-Soccer-Kick accessed Jan 23 2013
[201]: Scurr J. and Hall B., *The effects of approach angle on penalty kicking accuracy and kick kinematicswith recreational soccer players*, Journal of Sports Science and Medicine 8, 230-234, 2009
[202]: Plagenhoef S., *Patterns of Human Motion*, Englewood Cliffs, NJ: Prentice-Hall; 1971
[203]: Kollath E., *Analyse des Innenspannstoßes aus biomechanischer Sicht*, Fußballtraining, 2(5):15-20, 1983
[204]: Roberts E., Metcalfe A., *Mechanical Analysis of Kicking*, In Biomechanics I Basel, Karger Edited by Wartenweiler J, Jokl E, Hebbelinck M., 315-319,1967
[205]: Shan G. and Zhang X., *From 2D leg kinematics to 3D full-body biomechanics-the past, present and future of scientific analysis of maximal instep kick in soccer*, Sports Medicine, Arthroscopy, Rehabilitation, Therapy & Technology, 3:23, 2011
[206]: Stoner L, Ben-Sira D, *Variation in movement patterns of professional soccer players when executing a long range in-step soccer kick*, In Biomechanics VII-B. Edited by Morecki A, Fidelius K, Kedzior K, Wit A. Baltimore, University Park Press; 337-342, 1981
[207]: Rodano R, Tavana R, *Three dimensional analysis of the instep kick in professional soccer players*, In Science and Football II. Edited by Reilly T, Clarys J, Stibbe A. London, E & FN Spon, 357-361, 1993
[208]: Shan G, Westerhoff P, *Full-body kinematic characteristics of the maximal instep soccer kick by male soccer players and parameters related to kick quality*, Sports Biomech, 4(1):59-72, 2005
[209]: Lees A., *Biomechanics applied to soccer skills*, In Science and Soccer. 2nd edition. Edited by Reilly T, Williams A. London, Routledge; :123-134, 2003
[210]: Bull-Andersen T, Dorge HC, Thomsen FI, *Collisions in soccer kicking*, Sports Engin, 2:121-

126, 1999
[211]: Ismail A. R., Ali M. F.M., Deros B. M. and Johar M. S. N.M., *Biomechanics Analysis and Optimization of Instep Kicking: A Case Study to Malaysian Footballer*, National Conference in Mechanical Engineering Research and Postgraduate Students (1st NCMER 2010)26-27 MAY 2010, FKM Conference Hall, UMP, Kuantan, Pahang, Malaysia; pp. 535-542
[212]: Aldebaran Robotics, *User Documents*, http://users.aldebaran-robotics.com/
[213]: Micheal Quinlin, Personal Corespondance, RoboCup 2009, Team nubots, NewCastle University
[214]: Nassim Khaled, 'Straight Line and Plane Iintersection' mfile, 2007, http://www.mathworks.com/matlabcentral/fileexchang
[215]: Hans Eklund, Real Time Phased Locked Loops, 2006
[216]: Chris Maes, Runge's Phenomenon, 2011 http://demonstrations.wolfram.com/RungesPhenomenon
[217]: wikipedia, *http://en.wikipedia.org/wiki/B_spline*, Last accessed Apr 10, 2013
[218]: John Burkardt, SPLINE Interpolation and Approximation of Data, http://people.sc.fsu.edu/~jburkardt/cpp_src/spline
[219]: Buckley A., Humanoid Kicking, NUI Maynooth, Ireland, Undergrad Thesis, 2009