THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF ELECTRICAL ENGINEERING & TELECOMMUNICATIONS
SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Development of a Dynamic Gait for a Simulated Humanoid Robot

Christopher Stephen Hing

Bachelor of Engineering (Software Engineering)

November, 2002

Supervisor: Prof. Claude Sammut

Table of Contents

1 Introduction 6
1.1 Literature Survey 6
1.1.1 Purely Dynamic Walkers 7
1.1.2 Dynamic Walkers 10
1.1.3 Other Strategies 16
1.2 Thesis Contents 17
2 GURoo 18
2.1 The GURoo Robot 18
2.2 Simulator 19
2.3 Control 21
2.4 Proposed Walk 24
3 Simulated Centre of Gravity 26
3.1 Analysis 26
3.2 Implementation 27
3.3 Velocities and Accelerations of the Centre of Gravity 29
4 Simulated Feet 32
4.1 Analysis 32
4.2 Implementation 33
4.3 Centre of Foot and the Centre of Gravity 35
4.3.1 Analysis 35
4.3.2 Implementation 36
4.4 Foot Contact with Ground 37
5 Simulated Walk 39
5.1 Initiation and Balance 39
5.1.1 Analysis 39
5.1.2 Implementation 40
5.2 Walking Finite State Machine 50
5.3 Knee Straighten 52
5.3.1 Analyisis 52
5.3.2 Implementation 54
5.4 Foot Strike 57
5.4.1 Analyisis 57
5.4.2 Implementation 59
5.5 Foot on Ground 61
5.5.1 Analyisis 61
5.5.2 Implementation 63
5.6 Toe Off 65
5.6.1 Analyisis 65
5.6.2 Implementation 66
6 Results and Interpretation 69
6.1 Simulated Walk 69
6.1.1 Complete Walking Cycle 69
6.1.2 Stability 73
6.1.3 Efficiency 75
6.1.4 Speed 76
6.1.5 Comparison against Original Walk 76
6.2 Practical Application 77

7 Conclusions and Future Work 78
7.1 Conclusions 78
7.2 Future Work 79
8 Bibliography 80

List of Equations, Figures and Tables

Equation 2.3-1 Example of setting the desired joint velocity 22
Equation 2.3-2 pDist proportional controller 23
Equation 2.3-3 Proportional control for intermediate desired angle 23
Equation 3.2-1 Centre of Gravity Calculation pseudo-code 28
Equation 3.3-1 Velocity of the centre of gravity 30
Equation 3.3-2 Acceleration of the centre of gravity 30
Equation 4.3-1 Line intersecting COF parallel to AC 37
Equation 4.3-2 Line intersecting COG parallel to CB 37
Equation 4.3-3 Point P 37
Equation 4.3-4 Distance between two points 37
Equation 5.1-1 Walk initiation ankle roll angle 41
Equation 5.1-2 Walk initiation left ankle roll velocity 42
Equation 5.1-3 Walk initiation right ankle roll velocity 43
Equation 5.1-4 Walk initiation right hip roll velocity 43
Equation 5.1-5 Walk initiation left hip roll velocity 43
Equation 5.1-6 Walk initiation left ankle pitch velocity 44
Equation 5.1-7 Walk initiation right ankle pitch velocity 44
Equation 5.1-8 Walk initiation right hip pitch velocity 44
Equation 5.1-9 Walk initiation left hip pitch velocity 45
Equation 5.1-10 Foot pickup left ankle pitch angle 46
Equation 5.1-11 Foot pickup left ankle pitch velocity 46
Equation 5.1-12 Foot pickup left hip pitch velocity 47
Equation 5.1-13 Foot pickup left ankle roll velocity 47
Equation 5.1-14 Foot pickup left hip roll velocity 47
Equation 5.1-15 Foot pickup right hip roll velocity 47
Equation 5.1-16 Foot pickup right ankle roll velocity 47
Equation 5.1-17 Foot pickup right hip pitch angle 49
Equation 5.1-18 Foot pickup right hip pitch velocity 49
Equation 5.1-19 Foot pickup right ankle roll velocity 49
Equation 5.1-20 Foot pickup right ankle roll velocity 49
Equation 5.3-1 Knee straighten state right knee angle 54
Equation 5.3-2 Knee straighten state right knee velocity 54
Equation 5.3-3 Knee straighten state desired right ankle pitch 55
Equation 5.3-4 Knee straighten state left ankle pitch angle 56
Equation 5.3-5 Knee straighten state left ankle roll velocity 57
Equation 5.4-1 Foot strike state hip pitch angle control 61
Equation 5.5-1 Foot on ground state left ankle velocity control 64
Equation 5.5-2 Foot on ground state right ankle angle control 64
Equation 5.5-3 Foot on ground state right hip pitch angle control 64
Equation 5.6-1 Toe off state left knee angle control 67
Equation 5.6-2 Toe off state left hip pitch angle control 67

Figure 1.1-1 Honda's ASIMO and P3 Humanoids 12
Figure 1.1-2 M2 biped 14
Figure 1.1-3 Miniature Humanoids: Sony's SDR-4X, Fujitsu's HOAP, ZMP Corp's Pino 16
Figure 2.1-1 The GURoo Robot 18
Figure 2.2-1 GURoo Simulator 20

Figure 2.2-2 Simulator and Global Coordinate Axis (front on view) 21
Figure 3.1-1 Balance Beam and Centre of Gravity 26
Figure 3.2-1 Initial state of robot in simulation 28
Figure 3.2-2 Centre of Gravity in the Y axis (static walk) 29
Figure 3.3-1 Graph of acceleration, velocity and displacement of the Centre of Gravity... 31
Figure 4.2-1 Coordinates of right foot in relation to ankle, taken from GURoo's load file 33
Figure 4.2-2 Schematic of GURoo's feet 33
Figure 4.2-3 Looking at GURoo's feet from below... 34
Figure 4.3-1 Centre of Foot and Centre of Gravity 36
Figure 5.1-1 Initial state of robot in simulation 39
Figure 5.1-2 Phases of walking initiation 40
Figure 5.1-3 Walk initiation roll angles (front on).. 41
Figure 5.1-4 Walk initiation pitch angles (side on).. 44
Figure 5.1-5 Leaning to the left... 45
Figure 5.1-6 Foot pickup pitch angles (side on).. 48
Figure 5.1-7 Right leg lifted from ground 50
Figure 5.2-1 States of the walking finite state machine.. 52
Figure 5.3-1 During the knee straighten phase 54
Figure 5.3-2 Knee straighten state pitch angles (side on).. 55
Figure 5.3-3 Knee straighten pitch angles in final state (side on)............................. 56
Figure 5.4-1 Entering right foot strike state.. 58
Figure 5.4-2 Leaning torso forwards 59
Figure 5.5-1 Entering the right foot on ground state 62
Figure 5.5-2 During the right foot on ground state 63
Figure 5.6-1 During the left foot toe off state 66
Figure 6.1-1 Walk in action 72
Figure 6.1-2 Dynamic walk Centre of Gravity in Z 73
Figure 6.1-3 Instability due to spine bending.. 75

Table 2.3–1 Explanation of symbols in Equation 2.3-1 22
Table 5.1–1 Explanation of symbols in Equation 5.1-1 42
Table 5.1–2 Explanation of symbols in Equation 5.1-2 42
Table 5.1–3 Explanation of symbols in Equation 5.1-6 44
Table 5.1–4 Explanation of symbols in Equation 5.1-10 46
Table 5.1–5 Explanation of symbols in Equation 5.1-17 49

1 Introduction

Central to the success of a humanoid robot is its ability to be mobile. The chief advantage of a humanoid robot over many others is its potential to perform tasks in a human's complex environment. The bipedal locomotion that humanoid robots will eventually possess will allow for manoeuvrability in a similar fashion to people, and they will therefore be more suited, than any other robot, to tasks that humans can perform.

The importance of human-like mobility to a humanoid robot is also reflected by the large amounts of research conducted at present towards different walking strategies. Researchers are striving towards efficient, robust and stable gaits that at the same time reflect human-like fluidity and gracefulness.

This thesis sets out to develop a bipedal gait for a simulated humanoid robot, the GURoo.

1.1 Literature Survey

As outlined in [1], there are three classes of bipedal walkers that reflect the varying strategies of gait synthesis.

The first of these is the "static" walker. The successful control of this type of walker relies on a purely pre-programmed gait. In essence, each joint is explicitly told where to be at a particular point in time, and the walk is required to be mapped out in advance. Stability of a gait of this type requires that the centre of mass of the walker to be within the polygon that encloses the supporting foot, or feet. It is uncommon to see modern robots employing this strategy in order to walk. Due to the fact that the walk needs to be pre-programmed, it is not very robust and would have difficulty coping with disturbances.

The other remaining classes of walkers, according to [1], are the “purely dynamic” and “dynamic” walkers.

1.1.1 Purely Dynamic Walkers

“Purely dynamic” walkers are those that require extremely little control in order for them to walk, and rely largely on the “natural” physics (mostly the effects of gravity) of the walker in order to produce a stable gait. This class of walkers demonstrate the fact that a walk need not have any gait planning, and requires very little energy.

McGeer was the pioneer of the idea of “passive dynamics” with respect to bipedal walkers in [2], and argued that the motion of human legs could be likened to a pair of coupled pendula, and as such, could walk down a slope in a fashion similar to a rimless, spoked wheel would roll down an incline, given sufficient starting momentum.

He argued that the Wright brothers mastered gliding before adding an engine to an aeroplane, and thus relied on the natural physics of the object in order to first gain an understanding of flight. Walking, he suggested should be studied in a similar fashion. He argues that a walk should be able to be performed with a minimal amount of energy, especially by a robot in the shape of a person, seeing that people do it so naturally.

McGeer used the illustration of a spoked wagon wheel, with all but two of its spokes removed, and a pivot pin placed through the remaining two spokes at the axis of the wheel. This resembled a simple bipedal walker with large semi-circular feet. When placed on an inclined plane, as one of the walker’s feet rolls down the incline, the other foot would be lifted clear and swung forward, to catch the robot as the initial foot completes its roll, and thus continue the rolling motion. The initial foot would then be lifted clear, and the cycle would continue.

McGeer further demonstrated that the size of the foot, although having to remain curved, could be reduced for the model to become more practical for humanoid purposes. However, doing so would require the constraint that no momentum be lost when the support foot is changed (otherwise, the rear foot would not have enough momentum to be lifted forwards, and the cycle would collapse).

McGeer was able to build a purely dynamic walker that was constrained to fore-aft movement (via use of a crutch like double leg, could only move forwards and backwards, not side to side) that could walk a number of steps down an inclined plane. However, he found that if the period of swing of the pendula like legs became either too long or too short, the cycle required for stability was lost, and the robot would fall. He also proposed that by "pumping" the legs from the hip, the period could be controlled, and additionally, the robot could be made to walk on level ground without the aid of a slope.

Following in [3], in pursuit of a more anthropomorphic gait, McGeer suggested that the addition of knees removed the problem of toe stubbing that was experienced when the non-support foot was being swung through. He also suggests that the gait is more human-like, and would stand up to perturbations as the disturbance could be recovered from over a number of cycles (or steps). The knees were to fit well in with the passive dynamic model that was proposed, since the natural pendulum motion of the leg due to gravity would automatically cause the knee to contract when the leg is lifted off the ground, and would again automatically swing forward against its stop when the leg was to be placed.

Coleman and Ruina in [4] developed a simple straight-legged biped toy that illustrates McGeer's principles of passive dynamics, however this toy was not constrained to the 2-dimensions as was McGeer's. Coleman and Ruina suggested through this toy that McGeer's principles could also be used to stabilise lateral motion, rather than just the frontal motion as mentioned before.

In order to produce stability in side-to-side movement, the walker's legs were developed with a mass distribution similar to that of laterally extended balance bars for tightrope walkers. The centre of mass, however, was placed much closer to the foot than the hip, unlike a person's. As the toy sways to one side, the weight of the other leg creates a nearly inverted pendulum with a large rotational inertia, causing the walker to sway back the opposite way.

What is special about this toy, and different to others that were invented over a century ago (such as Fallis' in 1888), is that this one is not capable of standing upright when not moving. The only stable state of this toy can achieve is when the legs are in periodic motion, whilst travelling down an inclined plane.

In [5], Ohta, Yamakita and Furata added a slight control to a McGeer type walker (again, constrained to forwards and backwards movement) in order to produce a stable downhill walk, using passive dynamics. McGeer in [2] discussed the fact that a walk was in fact a repetitive cycle, in which the feet and legs moved in a periodic motion in order to support the body. Each step taken was simply an iteration of the cycle. Ohta et al in [5] made use of this fact, and placed a small actuator in the hip joints of the walker, in order to ensure that appropriate conditions were met at the start of each step. The actuator either slowed down or sped up the swing leg movement by applying an appropriate torque, to ensure that the period of the swing leg remained within the allowable range. The team were able to produce a walker that could walk stably down an inclined plane, which had no other control than the actuator on the hip.

Collins, Wisse and Ruina, in [6] developed the first passive-dynamic walker with human-like motions. The team began with a McGeer biped walker, and added a number of features to the robot to increase its 3-dimensional stability. Shaped feet were introduced that guided lateral motion. Soft springy heels were added in an attempt to reduce instability when the heel of the swing leg made contact with the ground. (These were later removed, as they seemed too compliant). Yaw movements created from the swinging legs were nullified by the use of counter-swinging weighted arms. Finally, to reduce side-to-side lean, the arms were also engineered to move freely outwards in a lateral direction at the same time it moved forwards. The passive-dynamics, as well as kneecaps proposed by McGeer were still utilised.

Collins et al were able to successfully produce an uncontrolled biped that walked stably down an inclined plane due to the forces of gravity and the robot's natural passive-dynamics.

All robots that had been developed thus far did not represent a proper human type model, as most concentrated on the dynamics of the legs and therefore only modelled the legs. In [7], Haruna, Ogino, Hosoda and Asada develop a simulation of a passive-dynamic robot that has a torso, and is able to produce a stable gait. The addition of the torso required the team to introduce torque control in the form of a simple proportional derivative (PD) controller between the torso and supporting leg, in order for the torso to stay upright.

1.1.2 Dynamic Walkers

As suggested in [1], dynamic walkers lie somewhere between the two extremities of "purely dynamic" and "static" walkers as discussed above. The largest proportion of today's humanoid robots employ dynamic methods in order to walk, as this method combines some of the control techniques of static walkers, while at the same time, makes use of some of the efficiencies of passive-dynamic walking.

Dynamic walking robots, unlike static robots, are able to adjust their gait depending on the situation that the robot finds itself in. The robot does not require for a walk to be pre-programmed, and is able to adjust a number of variables in order to maintain balance in its current situation.

Unlike passive dynamic walkers, a dynamic walker usually requires that a large amount of control be used to produce a stable motion. The robot does not rely heavily on its passive-dynamics for frontal and lateral stability, but rather, often uses various control mechanisms within the ankles and hips. Passive-dynamics are usually only used within this class of walkers to increase efficiency. The design of the robots is focussed more strongly towards replicating human-like features in terms of limbs, proportions and mass [10, 13], rather than passive-dynamics.

Stability of dynamic walkers relies on the ability of the robot to calculate its Zero Moment Point (ZMP), otherwise referred to as the Centre of Pressure. The idea of ZMP was first introduced by Vukobratovic in [8], and is often reiterated in numerous more recent papers such as [1, 9, 10]. The ZMP of a robot is the point on the floor at which the three-dimensional moment generated by the reaction force and reaction torque is equal to zero in the forward and sideways directions. Dynamic walkers need to control the ZMP to be within the convex hull of the foot-support area in order to remain stable. As a result, the problem of humanoid stability and balancing can be represented as an inverted pendulum problem – the mass at the end of the inverted pendulum needs to be kept within the bounds of the supporting feet, in order for the robot to not topple over. In order for the robot to walk, the feet must be placed in a position that allows the ZMP to also be moved in the desired direction.

Unlike static walkers, the Centre of Mass does not have to be directly over the supporting foot area. The Centre of Mass can move from the supporting foot area, so long as the ZMP remains within it.

There have been many successful attempts in creating dynamic biped gaits based on the principle of controlling the ZMP. Most however, are quite mathematically complex, and require an in depth knowledge of dynamics and related calculations.

An example can be seen when Napoleon, Nakaura and Sampei in [9] propose that a humanoid robot can be modelled as a double mass inverted pendulum. The first mass of the inverted pendulum would represent the waist and below, the second mass would represent the torso, arms and head. They state that by controlling the ZMP of this double mass inverted pendulum, humanoid balance can be achieved.

Lim, Kaneshima and Takanishi in [11] were also able achieve walking stabilisation through control of the ZMP of the robot. They were able to successfully implement their walking strategy on Waseda University's WABIAN-RIV robot. In this particular dynamic walk, the trunk and waist motion compensate the movements generated by the lower limbs, in order to keep the ZMP in its desired trajectory.

Honda's latest and most famous robots, Asimo and P3 both make use of dynamic walk techniques based on ZMP principles [10, 12]. Honda's use of ZMP in order to stabilise walking robots have allowed them to achieve feats such as walking up and down staircases. Hirai et al in [10] also discuss other designs used in order to create a stable motion for the Honda bipeds, such as their findings on the "Centre of Actual Total Ground Reaction Force" (C-ATRGF). This is the point on the foot on which the ground reaction force is exerted, and should be equal to the ZMP of the robot in perfect walking conditions when the ground is regular. However, in order to achieve a walk that was stable on irregular terrain, they had to take into account the fact that the ZMP and C-ATRGF were not equal, and would therefore have to adjust the latter accordingly.

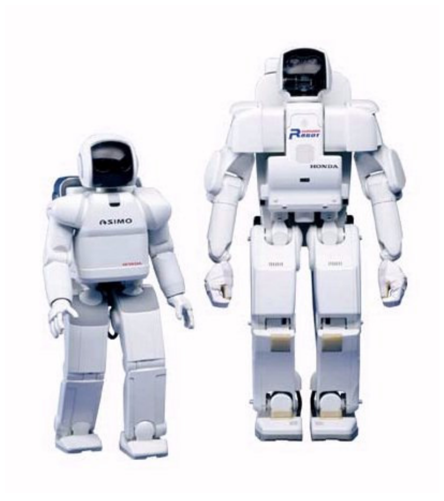


Figure 1.1-1 Honda's ASIMO and P3 Humanoids

Hirai et al in [10] also discussed the addition of an impact absorption mechanism in the feet in order to absorb the impact landing force. The weight of the Asimo, P2 and P3 robots make this absorption necessary in order to reduce the transmission of impact forces to prevent vibration in various leg controls.

Furuta et al in [13] proposed and implemented a number biped locomotion control strategies also based on ZMP principles. The robot in use was Mk.5.

One strategy was based on the double mass inverted pendulum principle as discussed above. Using this strategy, they were able to perform a stable walk using Mk.5, in which the feet were cleverly placed so the ZMP of the pendulum could move forward without toppling the robot.

Another interesting strategy presented in [13] again used ZMP for stability, however, the gait was generated by firstly computing the required initial and final states for a single step, and then, by use of interpolation by polynomials, the trajectories of joint angles between the states could be discovered.

The idea of generating a dynamic gait by the use of phases was also discussed by Furuta et al in [13]. They suggested that a step could be broken down into a number of phases, and by simply repeating the phases on each foot in turn, a controlled gait could be created. Each phase of the step could contain a specific task, thus breaking down the complexity of the walk.

Furuta suggested four phases. The first is the double support phase, in which both legs are on the ground and the robot accelerates forwards. The second is the sole support phase, in which a single leg supports the robot on its foot. This is followed by the phase in which that same foot supports the robot on its toe as it moves forward. The last phase is a shock absorption phase in which the opposite foot strikes the ground, but then absorbs the impact, to be able to return to the double support phase.

The notion of breaking a walk down into phases was also explored by researchers at the MIT Leg Lab [14, 15]. Using a simulation of their bipedal robot, M2, they were able to create a walking algorithm that was quite simple, due largely to the fact that the walk was broken down into a much simpler number of repeated individual phases.

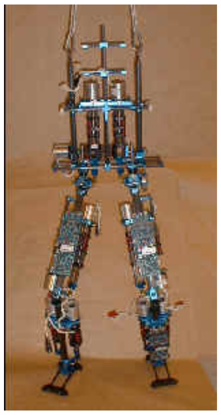


Figure 1.1-2 M2 biped

Parseghian in [14] created a finite state machine of eight states in order to simplify dynamic bipedal walking. The states were as follows: foot clearance, tibia vertical, foot strike, opposite toe off, opposite foot clearance, opposite tibia vertical, opposite foot strike, and finally, toe off. Each state was responsible for controlling the robot in such a manner that it would create the appropriate condition to continue on to the next state. By repetition of these states, a simple and stable walk could be achieved.

Essentially, there are only four states in this series, rather than eight. Half are simply copies of the initial four states, and have been modified simply for use on the opposite foot.

What seems particularly remarkable about this research is that unlike other control methods, which rely on complex ZMP and entire robot dynamics calculations, this method is able to control and servo each joint independently, depending on the current state of the robot and its centres of mass and pressure. The global dynamics of the robot were not crucial in creating a stable gait.

Parseghian was able to conclude from his research that it was unnecessary to derive complicated dynamics equations to compel a biped to walk; rather, each joint could be controlled as if it were decoupled.

The control strategies of the M2 robot were further simplified by use of passive-dynamics. Within certain states, the natural dynamics of the robot were used in such a manner that no control was necessary on particular joints. Pratt et al in [15] describes the natural dynamics of the robot that were used in order to simplify the walking algorithm. Kneecaps, compliant ankles, and passive swing legs are used in the passive dynamic strategy of the robot.

Supporting the biped using a straight leg is far more efficient than if the leg was bent, and the kneecap allows the controller to constantly servo the leg against it, and will therefore be guaranteed straight.

Compliant ankles within the support leg allow the biped to naturally move forward due to momentum. As the centre of mass of the robot moves forwards due to momentum, the compliant ankle allows the foot to stay on the ground yet at the same time, allows the leg to rotate forwards – this requiring no control. However, in order to manage the overall velocity of the robot, torque can be applied to the ankle in order to slow or speed up the leg rotation.

A passive swing leg will naturally swing forward due to gravity, again requiring no control. However if needed to be sped up, it can be done so using a hip pitch controller with a small amount of torque. The knee will also automatically straighten as the leg swings forwards, due to natural momentum.

M2's lateral stability came in the form of ankle roll controllers, which where able to correct the centre of mass to a desired position. A controller to place the foot either more to the left or the right of the robot prior to impact also allowed for a more robust walk.

The walking simulation of the M2 robot (available in downloadable video [16]) looks extremely human-like in many respects. It is both fast and stable, and control is greatly simplified due to the ability to control each joint individually, without the concern of the robot's global dynamics. Compared to many modern day humanoids relying on ZMP principles, including Honda's Asimo and P3, the miniature humanoids of Sony (Sony SDR-4X), Fujitsu (HOAP), and Japan

Science and Technology Corp. / ZMP Corp. (PINO), the simulation of M2 has by far, the most visibly elegant, yet simple, walk.

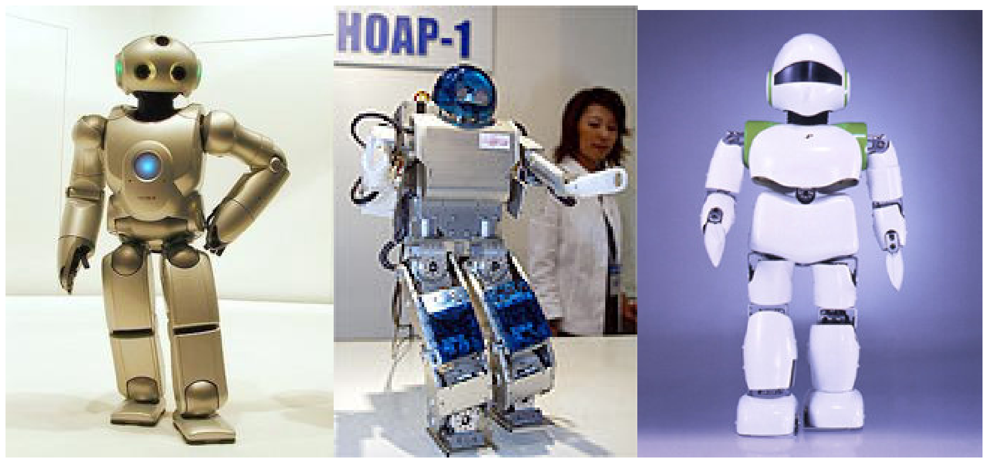


Figure 1.1-3 Miniature Humanoids: Sony's SDR-4X, Fujitsu's HOAP, ZMP Corp's Pino

1.1.3 Other Strategies

There have been other strategies that have been researched in able to realise a stable bipedal locomotion, a number of them are briefly mentioned below.

Benbrahim and Franklin in [17] were able to use neural networks and learning techniques in order to generate a gait for a bipedal robot constrained in sideways movement. A cerebellar model arithmetic processor (CMAC) neural network and a self-scaling reinforcement-learning algorithm were used in order to achieve their goal.

Zhou and Meng in [18] proposed a fuzzy reinforcement-learning algorithm for biped gait synthesis. They argued that in the real world, people use a fuzzy signal to evaluate a human's walk, such as "nearly fallen over", "faster", "slower", etc. They found that fuzzy evaluative feedback was able to improve the performance of a gait synthesizer.

Endo, Yamasaki, Maeno and Kitano in [19] used a genetic algorithm to create an energy efficient bipedal gait. It was successfully implemented on a PINO robot. Interestingly, the PINO robot did not have enough torque to be able to be controlled by ZMP methods, however, using a genetic algorithm approach, a stable walk was realized.

1.2 Thesis Contents

The remainder of this thesis is set out as follows:

Chapter 2 gives some background information on the GURoo robot, the simulator, the methods of control for moving the robot, and also discusses the desired walk to be implemented.

Chapter 3 discusses the various calculations that were necessary in order to provide the required centre of gravity data.

Chapter 4 discusses the various calculations that were necessary in order to provide the required foot data.

Chapter 5 discusses the walking algorithm and control methods.

Chapter 6 presents the results and their interpretations.

Chapter 7 concludes this thesis, and makes some suggestions for future work.

2 GURoo

2.1 The GURoo Robot

The GURoo (Grossly Underfunded Roo), was developed by the University of Queensland's Department of Information Technology and Electrical Engineering for entrance into the humanoid league of the Robot World Cup (Robocup) commencing in 2002. At 1.2 metres in height, the GURoo has 8 upper joints (head, neck and arms) controlled by standard RC servo motors, and 15 lower joints (hip and legs) controlled by geared DC motors, resulting in a total of 23 joints that make it possible for the robot to move freely with minimum external aid. The arms, however, serve no functional purpose and do not aid in the robot's movements. Furthermore, a CMOS camera mounted in its head provides the robot with monocular vision.

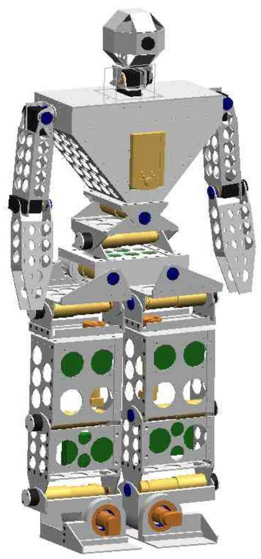


Figure 2.1-1 The GURoo Robot

A Compaq Ipaq pocket PC running Windows CE provides the necessary software to allow central controlling of the GURoo, as well as permitting rapid debugging in acting as an external interface. Mounted to the robot's chest, the Ipaq is able to receive information from the sensors located throughout the GURoo's body and generate the appropriate joint velocities, by running 1 kHz control loops, to enable it to perform the movements required.

The concern of this paper is with the operation of the lower joints, being the hips, which are able to perform pitch, roll and yaw rotations; the knees, both being able to alter pitch; and two ankles, each with the ability to rotate in roll and pitch. This gives the legs 12 degrees of freedom with which to move, and combined with the 70W geared DC motors and large torsion springs in the hips, these elements of the robot's lower half work together to provide sufficient torque, swing and velocity to power the robot's movements.

2.2 Simulator

The robot's simulator is based on Scott McMillan's open source Dynamechs [20] package. The Dynamechs package allows for robots to be simulated using the articulated body method.

Once a robot has been set up within Dynamechs, the package is then able to model the dynamics of the robot, given the movements of the joints. An interface allows the movements of the robot to be seen whilst the simulator is running (for example, Figure 5.1-1).

Through Dynamechs, the University of Queensland built a simulator of the GURoo robot through which various control methods for the robot could be developed. The simulation of the robot takes into account fine details of the robot including the actual masses of all the limbs of the robot, the friction between joints, and the means through which the actual motors are controlled. The simulation of GURoo, according to researchers at UQ, matches quite well to the actual performance of the robot.

There are two classes within the simulator that are of importance regarding this thesis: the Humanoid class, and the Central class. Their interactions are shown below (Figure 2.2-1).

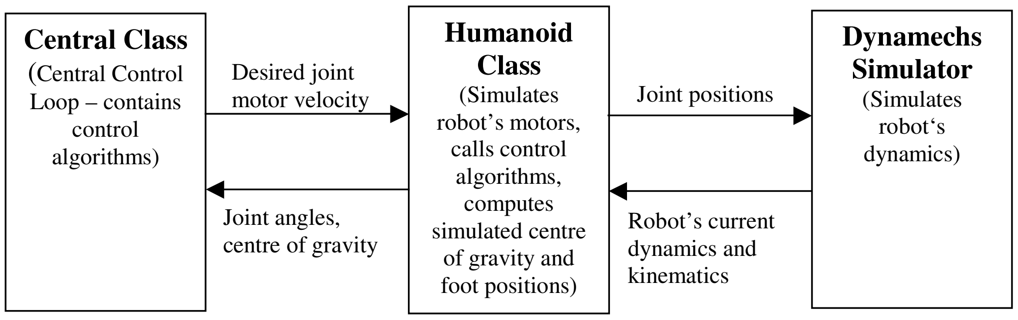

Central Class
(Central Control Loop – contains control algorithms)
Desired joint motor velocity
Joint angles, centre of gravity
Humanoid Class
(Simulates robot's motors, calls control algorithms, computes simulated centre of gravity and foot positions)
Joint positions
Robot's current dynamics and kinematics
Dynamechs Simulator
(Simulates robot's dynamics)

Figure 2.2-1 GURoo Simulator

The Humanoid class is basically the interface between the algorithms that control the robot, and the backend of the Dynamechs simulator. The class is responsible for informing the simulator as to what state the joints should currently be in, given the control algorithm. Additionally, the class is the access mechanism through which the control algorithm can obtain information on the current state of the robot in simulation.

The Central class represents the Central Control Loop of the robot, and as such, contains all the various control algorithms for robot movement, such as walking, balancing, waving etc. The simulator accesses this class fifty times a second (50Hz), in order to calculate the required velocities needing to be sent to the robot's motors. All the calculations for walking and balance control are contained within this class. The information required to run the algorithms is accessed through the Humanoid class.

The implementation of a dynamic walk algorithm was added to the Central class. This algorithm is discussed in detail in chapter 5.

Various modifications and additions were made to the Humanoid class in order to aid in developing the gait.

The graphical interface of the simulator showing the robot in motion (Figure 2.2-2) was edited to show additional information such as the centre of gravity drawn as a white crosshair, the centre of the feet and the actual position of the feet. Methods were also added that allowed the control algorithms within the Central class to query joint positions.

Other additions to the Humanoid class such as methods to compute the centre of gravity and foot positions will be discussed in more detail in chapters 3 and 4.

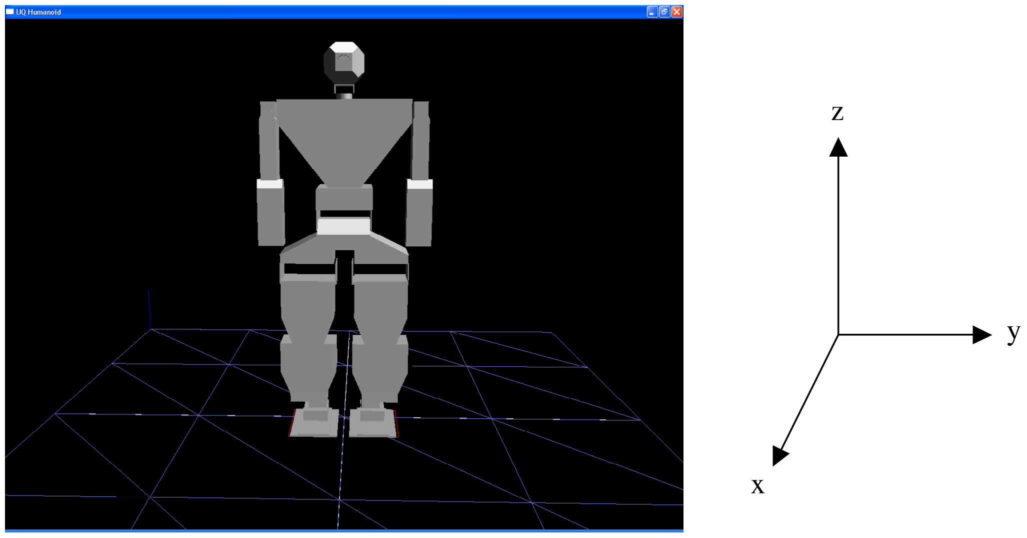


Figure 2.2-2 Simulator and Global Coordinate Axis (front on view)

Figure 2.2-2 also illustrates the three-dimensional global coordinate axis within the simulator. The x-axis points forwards, the y-axis points to the left of the robot, and the z-axis points upwards. Many results to queries made to the Dynamechs simulator for the Cartesian coordinates of various points are returned in the global coordinate system.

2.3 Control

The way in which the robot is controlled was understood by examining existing control algorithms within the Central class.

The UQ simulator is set up in such a way, that each time an algorithm within the Central Control Loop was run, it updates a Move object. A Move object contains, amongst other details, variables containing the desired velocity of each joint of the robot, as well as a link to the algorithm that should be called next time the Central Control Loop is run.

Therefore, to make the robot move, the desired velocity of the joints of the robot has to be updated within the Move object. The Move object is then accessed by various other parts of the simulator, in order to move the robot as instructed.

```
&move.left_leg.hip_pitch->desired_joint_vel =
  SRAD2ENC(deg2rad(desiredLeftHipPitchVel));
```

Equation 2.3-1 Example of setting the desired joint velocity

Figure	Meaning / Value
`move`	The Move object that contains all desired joint velocities etc.
`left_leg`	The left leg object within the Move
`hip_pitch -> desired_joint_vel`	The desired hip pitch of the left leg
`SRAD2ENC`	A convenience method for converting radians to encoder values, which are the input to the servo controllers.
`deg2rad`	A convenience method for converting degrees to radians.
`desiredLeftHipPitchVel`	The desired left hip pitch velocity calculated by the control algorithm, in degrees.

Table 2.3–1 Explanation of symbols in Equation 2.3-1

The control of the GURoo robot is quite different to many other humanoids. Rather than supplying a torque or position to the motor controllers, an angular velocity is required instead. The motor controllers then generate the required amount of torque to move the joint at the desired speed.

Therefore, in order to move a joint, its desired position was firstly calculated. From that desired position, a desired angular velocity was then computed. The desired angular velocity had to be such that if applied to the joint, it would move it to the desired position.

If the distance between the current and desired joint positions was large, then the velocity also had to be large. Similarly, if the difference was small, then the velocity had also to be small. In order to achieve such a method for calculating velocity, a very simple proportional controller was used (Equation 2.3-2). A proportional method, over proportional-derivative or proportional-integral-derivative methods, was used due to its simplicity (there is no need for calculating integrals and derivatives of change in angles). The velocity returned by the controller would be proportional to the difference between current and desired joint position.

```
float pDist(float k, float currDist, float desDist) {
      return (k * (desDist - currDist));
}
```

Equation 2.3-2 pDist proportional controller

Inputs to the controller are the current position, the desired joint position, and a constant. The output is the desired velocity.

In order to make the controller work effectively, the constant, k, has to be carefully tuned. The higher the value of k, the quicker the joint will move towards the desired position. However, a high value of k might also result in the joint overshooting its desired position. Extremely bad choices of k could also result in large oscillations of joint angle around the desired position. Logging and graphing joint positions and trial and error can lead to finding appropriate values of k.

In some cases, to make the control more stable, an intermediate desired angle was used (Equation 2.3-3). This angle was also calculated proportionally, and use of it meant that the joint velocity did not have to be calculated from the desired final angle, but rather the intermediate desired angle. Therefore, the joint did not have to aim for such a large increases, and control of joint angles would be smoother. The intermediate desired angle, when proportionally controlled, would increase in a manner that was proportional to the difference between the current joint position and the final desired angle.

```
desiredAngle = currentAngle + pDist(k, currentAngle,
                     desiredFinalAngle);
```

Equation 2.3-3 Proportional control for intermediate desired angle

Effectively, this allowed for two proportional controllers over a joint's movement. The first calculated the desired angle, and the second calculated velocity. The first would focus on controlling the rate in which the intermediate desired angle changed in order to reach the final angle. The second would focus on creating appropriate velocities for the joint motor, in order to meet the intermediate desired angle (see Equation 5.1-17 and its explanations for an example). The two controllers could be tuned separately for their individual tasks.

It is also worth noting that use of an intermediate desired angle controller prevents the desired angle from increasing too rapidly, such that the velocity controller cannot keep up. The controller is set up in such a way that the current joint position is added to, in order to calculate the desired angle. This means that even if a desired angle is not met during a movement, the subsequent calculation of the desired angle will take this into consideration, as the desired angle is calculated from the current joint position, not the previous desired angle. This prevents the desired angle from increasing too rapidly.

The above-mentioned method for controlling the simulated version of GURoo only works for the leg joints. Joints above the hips cannot be controlled using the simulator as yet. For this reason, the head rolls uncontrollably forwards in simulation (compare Figure 5.1-1 with Figure 5.1-5, for example).

2.4 Proposed Walk

The original walk that is already implemented on the robot is a purely static walk. The robot has no knowledge of the current state of its body. The only knowledge the robot does have is the elapsed time since the walk has begun.

The walk is broken down into a number of phases, and the time elapsed dictates the phase the robot should be in. The phases are continually repeated so that walking is performed. When the robot enters a phase, it is simply told the final joints positions. The algorithm calculates the desired velocity of the joints based on the sin curve, maximum acceleration allowed, and time elapsed.

As with all static walks, the desired joint positions of the robot have to be entirely mapped out, before the robot begins. The walk cannot take into account and counter various disturbances, and is therefore not robust.

The proposed walk is to be dynamic, and state based. Rather than have the elapsed time determine the state as in the original walk, a combination of joint positions and centre of gravity of the robot should do so. Control of the robot should stem from the state that the robot is in, rather than the time elapsed since it started moving.

The proposed walk should constantly take into account the centre of gravity, and continue to control it so that it always in a stable position, to ensure the robot does not fall. The walk should be based on the premise that if the centre of gravity is within the supporting foot base, the robot is in a stable position. Performing this would make the walk dynamic, and more robust.

A passive-dynamic walk will not suit the GURoo robot. Passive-dynamics relies on the physical design of the robot, and requires specially shaped feet and legs etc. It is evident that the GURoo robot was not designed to be a passive-dynamic walker given the motors and control capabilities that it has. However, if passive-dynamics can be used to some extent to increase the efficiency, or simplify the walk, then they should be used.

In order to achieve this proposed walk, various calculations need to be made that have not yet been implemented on the simulator. It will be necessary to know the centre of gravity of the robot, and also to know the distance of the centre of gravity from the feet to allow us to place the centre of gravity in stable positions. The methods and calculations relating to the centre of gravity, and to the feet, are discussed in chapters 3 and 4 respectively.

3 Simulated Centre of Gravity

3.1 Analysis

The simplest method for determining whether the robot is in a stable state can be measured via the centre of gravity. If the centre of gravity of the robot is above the supporting foot base, then the robot will be stable.

It is crucial for this dynamic walk algorithm to know the robot's centre of gravity. Controlling the centre of gravity to be in a stable position is the basis of dynamic nature of the walk.

The position of the centre of gravity, being a three-dimensional Cartesian coordinate, can be separated into its x, y and z components. Each component of the centre of gravity can be calculated independently from the rest.

The method used for calculating the centre of gravity is similar to that of calculating the centre of gravity of a pole with weights on either end, pictured in Figure 3.1-1.

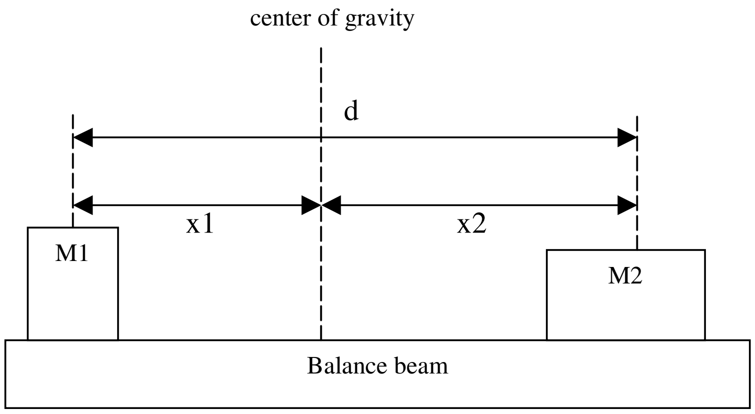


Figure 3.1-1 Balance Beam and Centre of Gravity

The centre of gravity in the above diagram is the point on the balance beam in which `x1 * M1 = x2 * M2`. Given the distance d and masses m1 and m2, it is possible to calculate the centre of gravity using simultaneous equations. The simulator provides the masses of each link, and global coordinates of the centre of each link.

Therefore, to calculate the centre of gravity of the robot for a certain axis, initially, two links of the robot are considered, and the centre of gravity for those two limbs is calculated, in the same manner as mentioned for the balance beam. The resultant centre of gravity of these two limbs can be treated as the position of a new mass equivalent to the sum of the masses of the initial two limbs. Then, the centre of gravity of this new mass and a third limb is calculated, and the process is repeated until there are no more links to consider, resulting in the centre of gravity for the robot in that particular axis.

3.2 Implementation

The method for calculating the centre of gravity was placed within the Humanoid class, as it requires access to the positions of the limbs. The Humanoid class can access these values directly from the Dynamechs simulation, as it contains pointers to the required objects.

The implementation of the method to calculate the robot's centre of gravity is outlined in pseudo-code in Equation 3.2-1. The simultaneous equations generated from the balance beam example in Figure 2.2-1 were resolved and coded into the appropriate method within the Humanoid class. The pseudo-code below illustrates the manner in which this was done. m, x, y and z represent the mass and coordinates of the current link.

```
CartesianVector getCentreOfGravity() {
  float force = 0;
  float cogX = 0;
  float cogY = 0;
  float cogZ = 0;
  for (each link) {
    cogX = ((m * (x - cogX)) / (force + m)) + cogX;
    cogY = ((m * (y - cogY)) / (force + m)) + cogY;
    cogZ = ((m * (z - cogZ)) / (force + m)) + cogZ;
    force = force + mass;
  }
  return {cogX, cogY, cogZ};
}
```

Equation 3.2-1 Centre of Gravity Calculation pseudo-code

The main difference between the pseudo-code and actual code is the manner in which pointers to the robot's links and the link's properties are established.

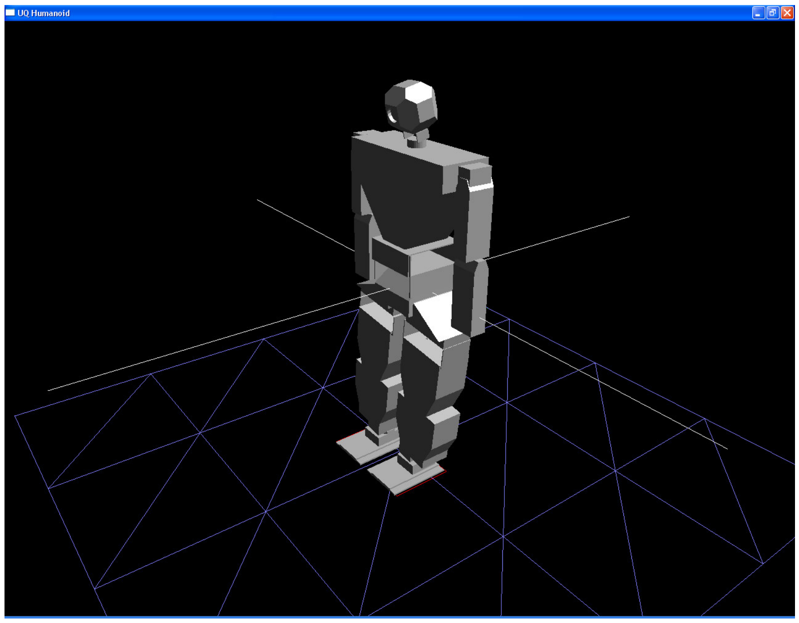


Figure 3.2-1 Initial state of robot in simulation

The graphical interface of the simulator was also edited to show the centre of gravity of the robot in three dimensions (Figure 3.2-1). The white crosshair through the robot is the centre of gravity in three dimensions. Since the centre of gravity in the z-axis played little part in the stability of the robot, the centre of gravity was therefore drawn on the ground. Drawing the centre of gravity on the ground also aided in understanding its placement in relation to the feet.

The centre of gravity of the robot was also recorded and logged in order for graphs to be produced. The graphs show the trends of the robot whilst walking. An example can be seen in Figure 3.2-2, which graphs the robot's centre of gravity in the y-axis for its static walk. It can be seen from the graph that the robot's centre of gravity is swaying from side to side.

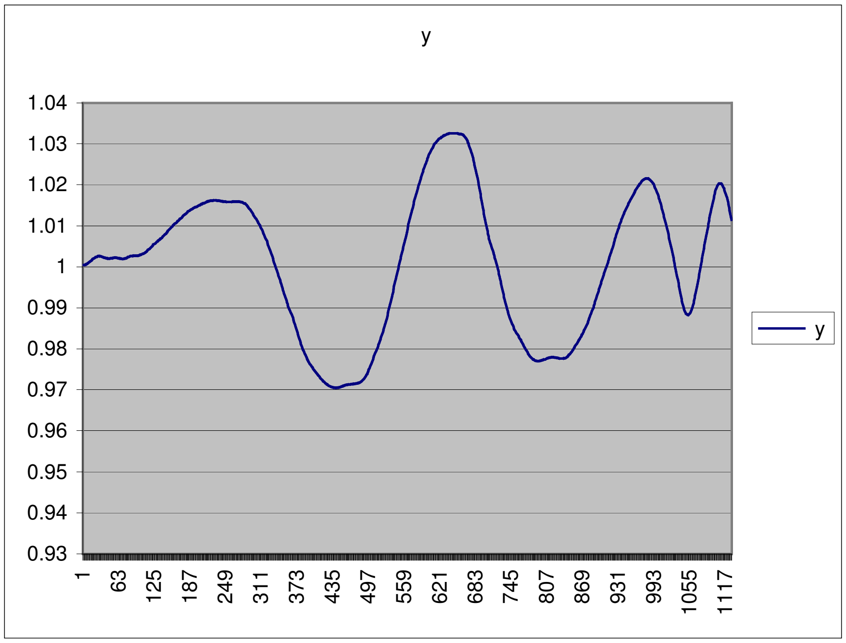


Figure 3.2-2 Centre of Gravity in the Y axis (static walk)

The method to calculate the centre of gravity is run every time the Humanoid class is told to update the simulated robot's joint positions, which happens every 0.02 seconds. This gives the control algorithm access to the robot's current centre of gravity.

3.3 Velocities and Accelerations of the Centre of Gravity

The velocities and accelerations of the centre of gravity were also calculated. This was done in the hope that they might be of some use in controlling the robot. Access to the velocities and accelerations of the robot's centre of gravity would allow the control algorithm to know which way the robot was moving, rather than simply its position at a single point in time.

The velocity of the centre of gravity was calculated as the change in displacement of the centre of gravity over time, and was calculated for each of the centre of gravity's axis.

```
cog_vel_X = (cog_x_2 - cog_x_1) / delta_t;
```

Equation 3.3-1 Velocity of the centre of gravity

Similarly, the acceleration of the centre of gravity was calculated as the change in velocity of the centre of gravity over time, and was calculated for each of the centre of gravity's axis.

```
cog_acc_X = (cog_vel_2 - cog_vel_1) / delta_t;
```

Equation 3.3-2 Acceleration of the centre of gravity

A number of attempts were made in trying to compute the velocities and accelerations, in an effort to remove the noise that occurs when sampling at such a fast rate.

The first attempt at calculating the velocities and accelerations resulted in meaningless graphs. Because the sampling interval is so small, even a slight change in the position of the centre of gravity would result in a large velocity. Similarly, even a slight change in the velocity of the centre of gravity would result in an even larger change in acceleration.

Additionally, because the centre of gravity did not change its velocity smoothly between samples and often oscillated around particular values, the resulting graph of accelerations would oscillate between large positive numbers (due to the velocity oscillating upwards) and large negative numbers (due to the velocity oscillating downwards). Such readings of the accelerations would have been useless, as the gait algorithm that relied on them would become extremely confused. An oscillating acceleration reading would not tell the difference between the robot falling, and the robot in a stable position.

The solution to this problem came in the form of a sliding window average over the centre of gravity's velocities. The window would cover the velocity readings that occur within a certain time frame, and average them out, giving a much more usable velocity reading from which accelerations could be calculated.

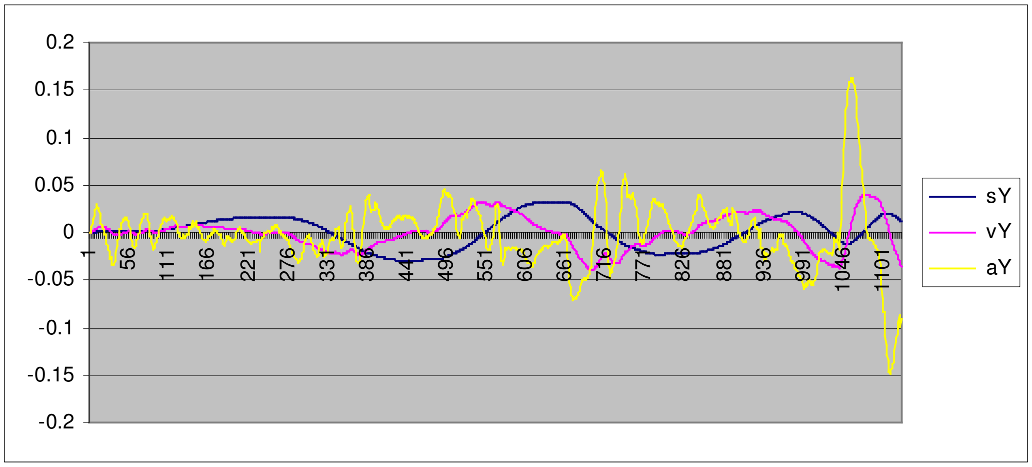


Figure 3.3-1 Graph of acceleration, velocity and displacement of the Centre of Gravity

The sliding window was also used over the accelerations to provide a smoother graph of accelerations. The sliding window's time frame was set at 0.2 seconds, however this is easily changed if required. A smaller time frame results in a noisier graph.

The calculation of velocities and accelerations occurred at the same time the centre of gravity was updated. On each update, the sliding window was also shifted forwards another 0.02 seconds. The above graph (Figure 3.3-1) shows an example of the accelerations and velocities calculated using the sliding window method. The graph shows the displacement of the centre of gravity in the side-to-side direction in blue, the velocity in purple, and acceleration in yellow. Note that when the displacement does not change, the velocity is zero, and when the velocity does not change, the acceleration is zero.

4 Simulated Feet

4.1 Analysis

The proposed dynamic walk solution relies heavily on keeping the centre of gravity within the foot support area. This can only be achieved if the control algorithm has access to appropriate foot coordinates, including the centre of the foot, and the corners of the foot. Despite the fact that the simulator is able to draw the feet in the correct manner, the desired coordinates are not readily accessible. The simulator does provide enough information about its links that make it possible to calculate the desired points. Therefore additions to the Humanoid class had to be made that would allow these calculations to occur.

Dynamechs provides the coordinates of the points that join links together. Therefore, it is possible obtain the coordinates of the point that joins the foot to the ankle; however, Dynamechs does not provide the coordinates of other necessary points that are required to calculate the centre of the foot, or the corners of the foot.

However, the simulator does provide an accessible three-dimensional rotation matrix, that when applied to a vector, rotates it to point along the same direction as the foot, using the ankle join position as the origin. Therefore, once the measurements of the foot from the ankle join position to the centre and outer corners are known and mapped into Cartesian coordinates, the rotation matrix can then be applied. The resulting coordinates map out a foot that is facing the same direction, in three dimensions, as the simulated foot.

The coordinates, however, map out a foot with its ankle join position at the origin of the global coordinates, rather than at the actual position where the foot should lie. It is therefore necessary to translate those coordinates to the desired position, by adding point that joins the foot to the ankle to each of the points.

The resulting coordinates, after performing the rotation and translation, map out the centre and outer corners of the foot.

4.2 Implementation

The distances from the ankle to the to the outer corners of the foot can be found by looking in the file that loads the GURoo robot into the simulator. The right foot's corners are specified below in Figure 4.2-1, as an illustration.

```
Contact_Locations       0.055 0.1   0.055
                        0.055 0.1   -0.145
                        0.055 -0.05 -0.145
                        0.055 -0.05 0.055
```

Figure 4.2-1 Coordinates of right foot in relation to ankle, taken from GURoo's load file

In order to find the distance from the ankle to the centre of the foot, it was necessary to consult a schematic of the foot, provided by UQ (Figure 4.2-2).

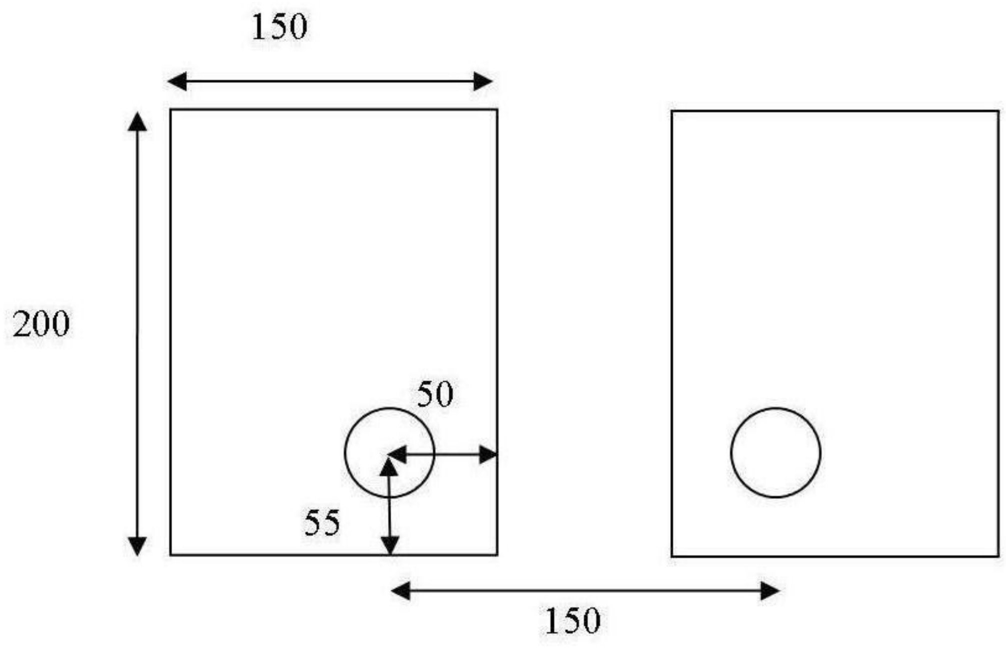


Figure 4.2-2 Schematic of GURoo's feet

From the schematic, it was calculated that the coordinates for the centre of the right foot in relation to the ankle were (0.055, 0.025, -0.045), specified in order of (z, y, x). It was necessary to specify the coordinates in this manner so they would match the order that they were given in the load file.

Accessing the rotation matrix and position of the ankle in global coordinates was performed through obtaining pointers to required objects within the Dynamechs simulator. Local copies were made of the rotation matrix and ankle coordinates. This occurred during the loop that was used to calculate the centre of gravity. When the feet were accessed in order to obtain their positions and masses, the rotation matrices and positions were copied.

In order to apply the rotation matrix to the coordinates of the distances from the ankle to the corners and centre of the foot, a simple method that performed matrix multiplication was constructed. The inputs were a rotation matrix and a Cartesian vector point, and the result of the rotation was copied back into the Cartesian vector point input. This method was run on all corner and centre coordinates.

Simple coordinate addition then occurred to add the global coordinates of the joint position to the centre and corner coordinates. The resulting coordinates mapped out the corners of the foot, as well as the foot's centre.

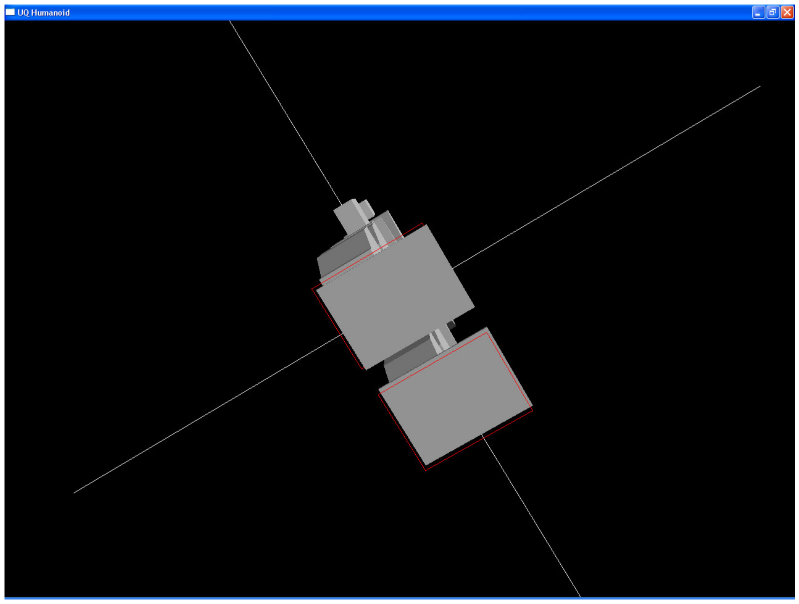


Figure 4.2-3 Looking at GURoo's feet from below

The simulator's graphic interface was again edited to show the outlines of the feet, produced from the coordinates just calculated, in order to view more clearly the placement of the centre of gravity within the foot support area (Figure 4.2-3). Additionally, the outlines provide a more accurate view of the simulated position of the foot, over the VRML modeled foot, as they represent actual coordinates of the foot. The outlines are drawn in red.

4.3 Centre of Foot and the Centre of Gravity

4.3.1 Analysis

The distance between the centre of foot, and the centre of gravity had to be calculated. The need for this is due to the nature in which the position of the centre of gravity is controlled in the dynamic walk algorithm.

The ideal position for the centre of gravity of the robot is over the centre of its support foot, if a stable state is required. If the centre of gravity is either to the left or to the right of the foot's centre, it can be corrected by rotating the ankle roll in the appropriate manner. Similarly, if the center of gravity is either in front of or behind the centre of the foot, then appropriate changes to ankle pitch can be used to correct the centre of gravity.

It was therefore desirable to break down the distance from the centre of the foot to the centre of gravity into left/right and forwards/backwards components, relative to the direction that the foot pointed. This is illustrated in Figure 4.3-1. The desired distance to calculate in the left/right direction is Y, and in the forwards/backwards direction is X. The method that performs this calculation should continue to work, even if the foot is rotated (as pictured).

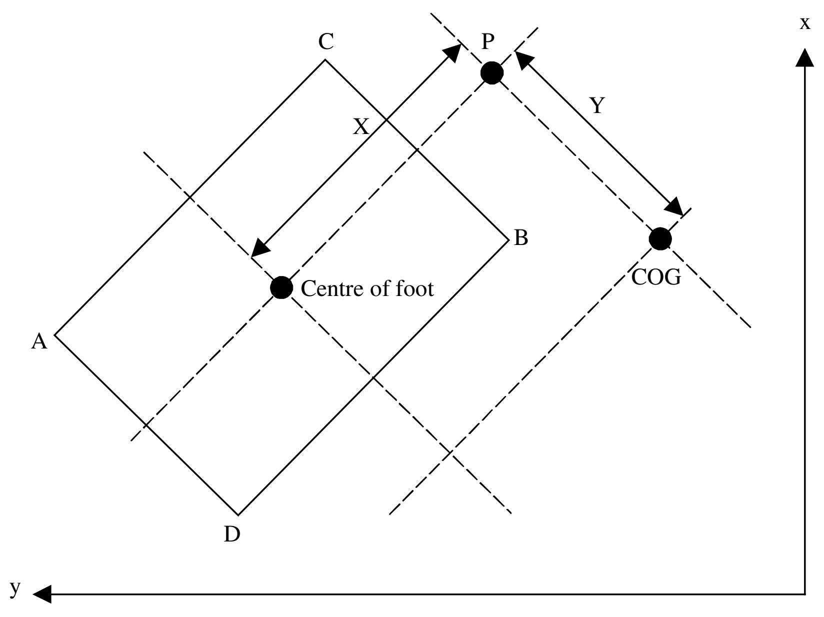


Figure 4.3-1 Centre of Foot and Centre of Gravity

4.3.2 Implementation

Because the distance between the centre of gravity and the centre of the foot would only be calculated when the foot is in a support phase (i.e., firmly on the ground), the Z component of the centre of the foot and centre of gravity could be disregarded.

The points A, B, C, D are the known global coordinates of the corners of the foot. The centre of the foot (COF) and the centre of gravity (COG) are also known, in world coordinates.

The point P represents the point whose global coordinates are required. Once P is known the distance X is then equal to the distance between P and X. Similarly for Y.

Point P is found by firstly establishing the equation of the line that passes through COF and is parallel to AC (Equation 4.3-1). The equation of the line that passes through COG and is parallel to CB is also required (Equation 4.3-2). The point in which these two lines intersect will yield the global coordinates of point P. The resultant equations for calculating point P are given in Equation 4.3-3.

```
m1 = (Cy - Ay) / (Cx - Ax)
y = (m1 * x) + COFy - (m1 * COFx)
```

Equation 4.3-1 Line intersecting COF parallel to AC

```
m2 = (Cy - By) / (Cx - Bx)
y = (m2 * x) + COGy - (m2 * COGx)
```

Equation 4.3-2 Line intersecting COG parallel to CB

```
Px = (COFy - (m1 * COFx) + (m2 * COGx) - COGy) / (m2 - m1);
Py = (m1 * Px) + COFy - (m1 * COFx)
```

Equation 4.3-3 Point P

The distances X and Y can then be calculated using the distance between two points formula (Equation 4.3-4). For X, the required distance is that between P and COG, while for Y, it is the distance between P and COG.

```
distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)
```

Equation 4.3-4 Distance between two points

The final step in calculating the distances X and Y is to adjust their sign. Simply, X is positive if it lies in front of the foot, and negative if it lies behind. Y is positive if it lies to the left of the foot, and negative if it lies to the right. Obviously, this would fail if the robot turned more than 180°, however, turning will not be implemented in the control algorithm.

The distances X and Y for both feet were calculated every time the Humanoid class was called to update the simulator.

4.4 Foot Contact with Ground

The control algorithm was also required to know if the foot was touching the ground. This was necessary to allow the walking finite state machine to enter the various foot strike states (5.4).

Additionally, the knowledge of whether the foot was on the ground could also provide important information regarding the robot's state. For example, if a foot is off the ground, there is no use using it as a support foot, and changes must be made to the robot's kinematics in order to move it back to the ground.

The four corner points of each foot are known (4.2), and these points represent three-dimensional Cartesian coordinates. Therefore, the Z coordinates of these points effectively returns the distance that point is from the ground. If the Z coordinate of a particular point is greater than zero, than that point of the foot is off the ground.

Various methods were then implemented to test whether the toe or heel of the foot was touching the ground, or if the entire foot was in contact with the ground. The methods simply returned true if the foot was in contact with the ground, or false otherwise. The implementation of these methods gave the simulated GURoo a form of primitive foot sensors.

5 Simulated Walk

5.1 Initiation and Balance

5.1.1 Analysis

The initial state of the robot (Figure 5.1-1) given by the simulator has the robot standing upright, steadily on two feet. All joints are in a state in which no control, as yet, has been applied to them.

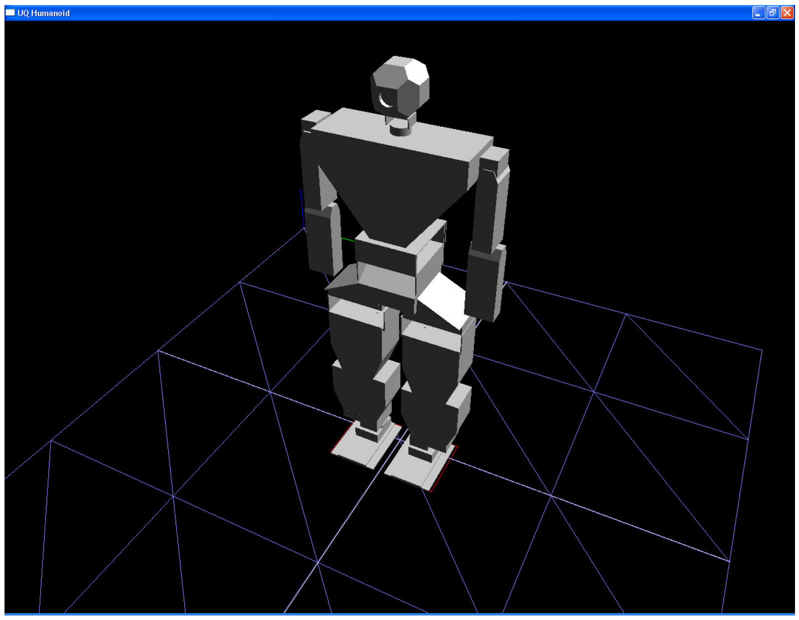


Figure 5.1-1 Initial state of robot in simulation

As a result, the robot balances with its centre of gravity placed above its ankles in the sagittal plane, and exactly halfway between its feet in the frontal plane. The white crosshair shown beneath the robot in Figure 5.1-1 gives the position of the centre of gravity.

In order to allow the robot to enter the walking finite state machine (5.2), the initial position of the robot has to be moved to one that matches a morphology that would allow it to enter a state within the finite state machine.

The knee straighten state (5.3) has an appropriate starting position to aim for, since out of all other states, this would require the least amount of movement to get to, and would also reflect the manner in which humans begin walking.

The knee straighten state is entered when the centre of gravity of the robot is above the support foot and the non-support leg needs to straightened out and readied for ground impact in order to support the robot as it steps forward. In order to enter this state, the knee of the non-support leg must be bent, and the foot of that leg cannot be behind the robot. Essentially, the position required to enter this state is that of a person standing on one leg.

In order to achieve such a state given the initial position of the robot, a two-phase approach is used (Figure 5.1-2), similar to that of Parseghian in [14]. The robot firstly has to shift its centre of mass onto its support foot in order to allow the non-support leg to be lifted without causing the robot to topple. Once this shift has been suitably achieved, it must then begin lifting its non-support foot from the ground, whilst keeping the centre of mass over its supporting foot.

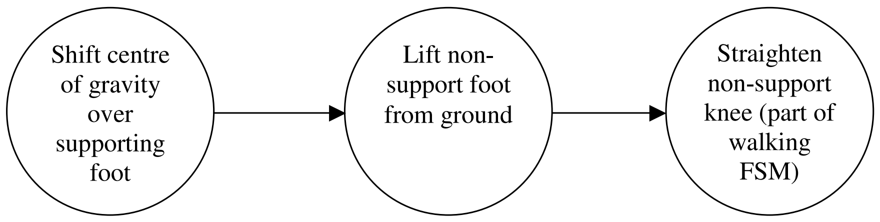

Shift centre of gravity over supporting foot
Lift non-support foot from ground
Straighten non-support knee (part of walking FSM)

Figure 5.1-2 Phases of walking initiation

5.1.2 Implementation

In order to shift the centre of mass of the robot over the supporting foot (the left foot in this case), the robot had to be leant to the left. The control for this movement stemmed from the left ankle.

By changing the roll angle of the left ankle, and then subsequently altering all other leg roll joints (left hip roll, right hip roll and right ankle roll) to equal the same angle as the left ankle, the robot could be leant to the left, whilst keeping both feet on the ground and its torso in an upright position (Figure 5.1-3).

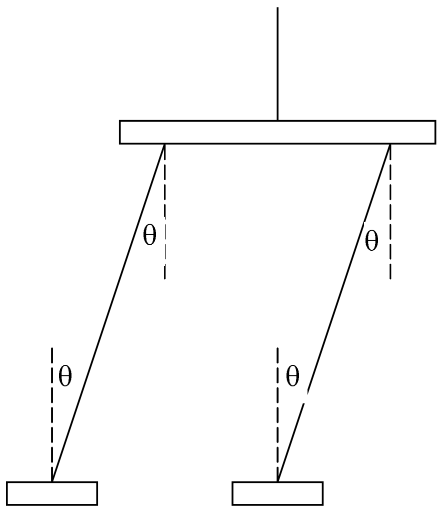


Figure 5.1-3 Walk initiation roll angles (front on)

In order to compute the desired left ankle roll angle required to smoothly and gradually lean the robot to the left, a proportional controller was used (Equation 5.1-1). The controller increases the left ankle roll angle proportionally to the distance that the centre of gravity is from the centre of the left foot. That distance is measured in the left foot's frontal plane, using the method described in 4.3. Essentially, if the distance between the centre of gravity and centre of foot is large, the controller increases the angle quickly. As the distance decreases and the robot nears its desired position, the increase in angle declines so as to reduce overshoot.

```
desiredLeftAnkleRoll = currentLeftAnkleRoll + pDist(k,
              desiredCogY, currentCogY);
```

Equation 5.1-1 Walk initiation ankle roll angle

Figure	Meaning / Value
`desiredLeftAnkleRoll`	The resultant left ankle roll angle to which the joint should be set.
`currentLeftAnkleRoll`	The current left ankle roll angle.
`PDist`	A method that represents a proportional controller (see 2.3).

`K`	= 2. A constant required for the pDist method. (see 2.3).
`desiredCogY`	= 0. Desired centre of gravity in the Y direction in relation to the centre of the left foot (see 2.3, 4.3). Set to 0 since we want the centre of gravity to be over the middle of the left foot.
`currentCogY`	Current centre of gravity in the Y direction in relation to the centre of the left foot (see 2.3, 4.3).

Table 5.1–1 Explanation of symbols in Equation 5.1-1

The robot's control architecture requires that joint velocities, rather than joint positions or torques be sent to the motor controllers. In order to calculate the required velocity to move the left ankle roll joint from its current state to its newly calculated desired state, another proportional control method was used (Equation 5.1-2). Similar to the control mentioned above, if the distance having to travel is great, the velocity is high. As the distance required traveling decreases, the velocity also decreases.

```
desiredLeftAnkleRollVel = (desiredLeftAnkleRoll -
        currentLeftAnkleRoll) * k;
```

Equation 5.1-2 Walk initiation left ankle roll velocity

Figure	**Meaning / Value**
`desiredLeftAnkleRollVel`	The resultant left ankle roll velocity.
`desiredLeftAnkleRoll`	The desired left ankle roll angle, calculated in Equation 5.1-1.
`currentLeftAnkleRoll`	The current left ankle roll angle.
`K`	= 0.02. A constant required for the proportional control method. Tuned in the same manner as for the pDist controller (see 2.3).

Table 5.1–2 Explanation of symbols in Equation 5.1-2

Right ankle and right hip roll joints are controlled to be equal to the same angle as left ankle roll, whereas left hip roll is controlled to be equal to negative left ankle roll (left hip roll is negative due to the nature of the setup of the robot – left hip roll is opposite to all other roll joints). The required velocity of these joints is set in the same fashion as the left ankle roll joint.

```
desiredRightAnkleRollVel = (desiredLeftAnkleRoll -
        currentRightAnkleRoll) * k;
```

Equation 5.1-3 Walk initiation right ankle roll velocity

```
desiredRightHipRollVel = (desiredLeftAnkleRoll -
        currentRightHipRoll) * k;
```

Equation 5.1-4 Walk initiation right hip roll velocity

```
desiredLeftHipAnkleRollVel = (-desiredLeftAnkleRoll -
        currentAnkleRoll) * k;
```

Equation 5.1-5 Walk initiation left hip roll velocity

The centre of mass in the fore-aft direction did not have to be changed, since the centre of gravity only had to be shifted sideways. However, the ankle pitch angle had to be maintained to prevent the robot leaning forward or backwards. If the robot were let to lean forwards, it would make it impossible to lift the right leg without the right foot becoming caught. If the robot were let to lean backwards, it might topple. As a result, the ankle pitch angle was set to 0 (ankle pitch is measured from vertical).

Hip pitch control did have to be implemented nevertheless, in case some ankle pitch movement did occur (exaggerated in Figure 5.1-4). Hip pitch was controlled to equal ankle pitch, in order to keep the torso upright. In the control algorithms, hip pitch was set to negative ankle pitch (again, due to the nature of the robot setup, and having the joint move in the desired direction).

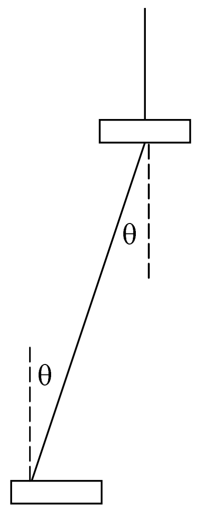


Figure 5.1-4 Walk initiation pitch angles (side on)

The required velocities to control the pitch joints were calculated in a similar fashion to those of roll (Equation 5.1-6).

```
desiredLeftAnklePitchVel = (desiredLeftAnklePitch -
        currentLeftAnklePitch) * k;
```

Equation 5.1-6 Walk initiation left ankle pitch velocity

Figure	Meaning / Value
`desiredLeftAnklePitchVel`	The resultant left ankle pitch velocity.
`desiredLeftAnklePitch`	= 0. The desired left ankle pitch angle. Set to 0 since we want the legs perpendicular to the feet.
`currentLeftAnklePitch`	The current left ankle pitch angle.
`k`	= 0.02. A constant required for the proportional control method. Tuned in the same manner as for the pDist controller (see 2.3).

Table 5.1–3 Explanation of symbols in Equation 5.1-6

```
desiredRightAnklePitchVel = (desiredLeftAnklePitch -
        currentRightAnklePitch) * k;
```

Equation 5.1-7 Walk initiation right ankle pitch velocity

```
desiredRightHipPitchVel = (-desiredLeftAnklePitch -
        currentRightHipPitch) * k;
```

Equation 5.1-8 Walk initiation right hip pitch velocity

```
desiredLeftHipPitchVel = (desiredLeftAnklePitch -
        currentLeftHipPitch) * k;
```

Equation 5.1-9 Walk initiation left hip pitch velocity

Left and right knees did not have to be controlled in this state. The simulator has the knees straight on start, and there has been no need to bend them as yet, hence no control is applied.

Once the robot has leant far enough to the left (Figure 5.1-5), and has moved its centre of gravity over the centre of the left foot (or at least, within a certain threshold), it then enters the foot pickup state, which requires that the robot move its right foot off the ground.

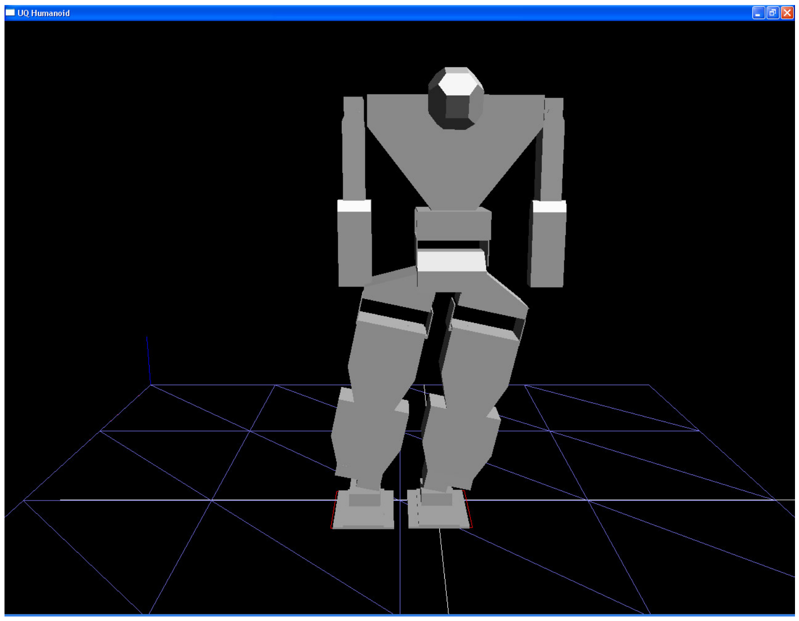


Figure 5.1-5 Leaning to the left

Due to the fact that the right foot is being moved off the ground in this state, the robot becomes subject to increased disturbance, since there is now only one support foot, and the non-support leg is moving. A more robust control was therefore required to keep the centre of gravity stable in the sagittal plane.

A desired position for the centre of gravity in relation to the left (support) foot had to be decided. Keeping the centre of gravity above the left ankle, only in the fore-aft direction proved an obvious choice, as it would allow the right foot to move smoothly from the ground. Initially, it was attempted to move the centre of gravity over the centre of the left foot in the fore-aft plane, however, this would not allow the right foot to move off the ground without forcing the robot upon its left heel (which was not stable).

Centre of gravity in the sagittal plane is controlled by the left ankle pitch angle. The calculations for the desired left ankle pitch angle are similar to those in Equation 5.1-1 for calculating the desired left ankle roll.

```
desiredLeftAnklePitch = currentLeftAnklePitch + pDist(k,
                  currentCogX, desiredCogX);
```

Equation 5.1-10 Foot pickup left ankle pitch angle

Figure	Meaning / Value
`desiredLeftAnklePitch`	The resultant left ankle pitch angle to which the joint should be set.
`currentLeftAnklePitch`	The current left ankle pitch angle.
`pDist`	A method that represents a proportional controller (see 2.3).
`k`	= 0.002. A constant required for the pDist method. (see 2.3). This number is small since we do not want to change the pitch greatly.
`desiredCogX`	= -0.045. Desired centre of gravity in the X direction in relation to the centre of the left foot (see 2.3, 4.3). Set to – 0.045 since we want the centre of gravity to be over the ankle, and the ankle is 4.5cm behind the centre of the foot.
`currentCogX`	Current centre of gravity in the X direction in relation to the centre of the left foot (see 2.3, 4.3).

Table 5.1–4 Explanation of symbols in Equation 5.1-10

Left ankle pitch velocity is controlled the same way as in the previous state (Equation 5.1-6), however this time, a proportional control convenience method is used (Equation 5.1-11). k is still set to 0.02.

```
desiredLeftAnklePitchVel = pDist(k, currentLeftAnklePitch,
                  desiredLeftAnklePitch);
```

Equation 5.1-11 Foot pickup left ankle pitch velocity

Left hip pitch is also controlled in the same manner as the previous state (Equation 5.1-9), to ensure that the torso is always upright. Again, the proportional control convenience method is used (Equation 5.1-12). k is still set to 0.02.

```
desiredLeftHipPitchVel = pDist(k, currentLeftHipPitch,
                desiredLeftAnklePitch);
```

Equation 5.1-12 Foot pickup left hip pitch velocity

The same method of controlling the centre of gravity in the frontal plane is used in the foot pickup state as in the previous lean state. However, since the centre of gravity is already in position, in order to create more stability, the constant used in calculating the desired left ankle roll angle (k) is decreased to 0.002 (therefore the ankle angle will fluctuate less, reducing wobble).

Although the right foot is not on the ground in this state, right ankle roll is still controlled to have the foot parallel to the ground, ready for when the right foot next makes contact.

Calculation of roll velocities also made use of the proportional control convenience method. k is still set to 0.02.

```
desiredLeftAnkleRollVel = pDist(k, currentLeftAnkleRoll,
                desiredLeftAnkleRoll);
```

Equation 5.1-13 Foot pickup left ankle roll velocity

```
desiredLeftHipRollVel = pDist(k, currentLeftHipRoll, -
                desiredLeftAnkleRoll);
```

Equation 5.1-14 Foot pickup left hip roll velocity

```
desiredRightHipRollVel = pDist(k, currentRightHipRoll,
                desiredLeftAnkleRoll);
```

Equation 5.1-15 Foot pickup right hip roll velocity

```
desiredRightAnkleRollVel = pDist(k, currentRightAnkleRoll,
                desiredLeftAnkleRoll);
```

Equation 5.1-16 Foot pickup right ankle roll velocity

It is now plausible to lift the right foot from the ground as the centre of gravity is entirely controlled to be over the left, and as a result, the right foot will no longer be required to support the robot.

In order to lift the right foot from the ground, the right hip must first rotate the thigh slightly upwards, the knee must bend, and the foot is required to stay perpendicular to the ground.

Rotating the hip joint out to a desired angle, and then controlling the right ankle pitch to be equal to the right hip pitch can easily achieve this. The knee, at the same time, has to be bent inwards at twice the angle of the right hip pitch (Figure 5.1-6).

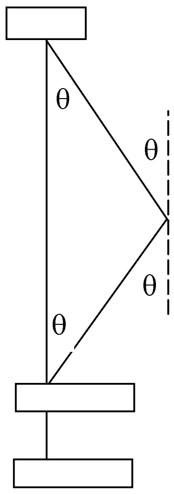


Figure 5.1-6 Foot pickup pitch angles (side on)

The angle to which the right hip would be rotated was set at 20°. This angle was chosen as it allows the foot to be moved clearly from the ground, and also results in a sufficient stride length, discussed further in 5.4.

Control of the right hip was similar to that of left ankle pitch in which the change in angle was calculated first (Equation 5.1-17), followed by the velocity required to cover that required change in angle (Equation 5.1-18).

```
desiredRightHipPitch = currentRightHipPitch + pDist(k,
  currentRightHipPitch, desiredRightHipPitchFinal);
```

Equation 5.1-17 Foot pickup right hip pitch angle

Figure	Meaning / Value
`desiredRightHipPitch`	The resultant right hip pitch angle to which the joint should be set.
`currentRightHipPitch`	The current right hip pitch angle.
`pDist`	A method that represents a proportional controller (see 2.3).
`k`	= 0.1. A constant required for the pDist method. (see 2.3).
`desiredRightHipPitchFinal`	= 20. Desired angle that we want to set rotate the thigh out to.

Table 5.1–5 Explanation of symbols in Equation 5.1-17

```
desiredRightHipPitchVel = pDist(k, currentRightHipPitch,
                desiredRightHipPitch);
```

Equation 5.1-18 Foot pickup right hip pitch velocity

Control of the right ankle pitch was performed in the same manner as that of the left, however the desired angle, which the velocity is moving towards, was equivalent to the right hip pitch angle instead (Equation 5.1-19).

```
desiredRightAnklePitchVel = pDist(k, currentRightAnklePitch,
                currentRightHipPitch);
```

Equation 5.1-19 Foot pickup right ankle roll velocity

The right knee angle had to move inwards at twice the angle that the right hip pitch was moving. Again, due to the setup of the robot, it was necessary to set the angle at twice the hip pitch, plus another 90° (see 2.3 for more details). The required velocity was calculated as follows:

```
desiredRightKneeVel = pDist(k, currentRightKnee,
      (currentRightHipPitch * 2) + 90);
```

Equation 5.1-20 Foot pickup right ankle roll velocity

In the all the control equations for right leg pitch velocities, the proportion constant (k) was still 0.02.

When the right hip pitch reaches within a certain threshold of the desired right hip pitch (Figure 5.1-7), the robot then enters the knee straighten phase of the walking finite state machine (5.3).

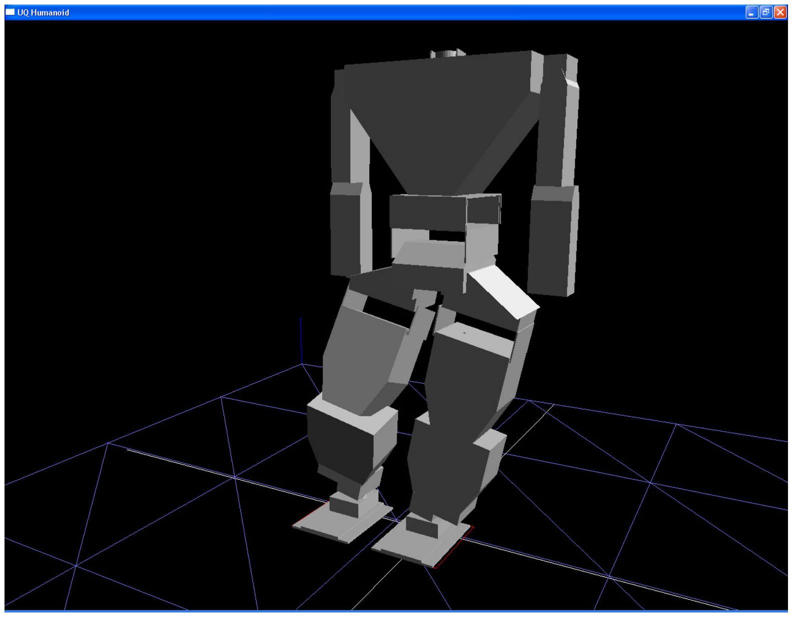


Figure 5.1-7 Right leg lifted from ground

It is worth noting that if the robot is not instructed to move into the knee straighten phase, then left in this state, the control is sufficient to allow the robot to continue balancing on one leg indefinitely.

5.2 Walking Finite State Machine

The walking cycle is broken down into eight states, which comprise the walking finite state machine (Figure 5.2-1).

One step is comprised of four states. A step starts at the point where the robot, standing on one leg, steps forward and transfers its weight onto its non-support leg (therefore making it the new support leg). A step is completed when all the weight is removed from the initial support leg, and that leg has been elevated, readied to step out again. The four states that comprise this movement are the knee straighten state, the foot strike state, the foot on ground state, and the toe off state.

By repeating those four states on the opposite leg, one cycle of the walk is completed, and the robot is ready to begin the cycle again.

The knee straighten state is responsible for straightening the non-support leg, and bringing the non-support foot into contact with the ground.

The foot strike state is responsible for shifting the centre of gravity in the frontal plane evenly between the two feet on the ground. It is also responsible for moving the centre of gravity in the sagittal plane over the front foot (which is to be the new support foot).

The foot on ground state is required to move the centre of gravity of the robot entirely over the new support foot, and place the rear leg in a position where it can be lifted from the ground.

The toe off brings the rear leg off the ground, and forwards next to the robot, to create a state where the robot is standing on one leg, ready to enter the knee straighten state on the opposite leg.

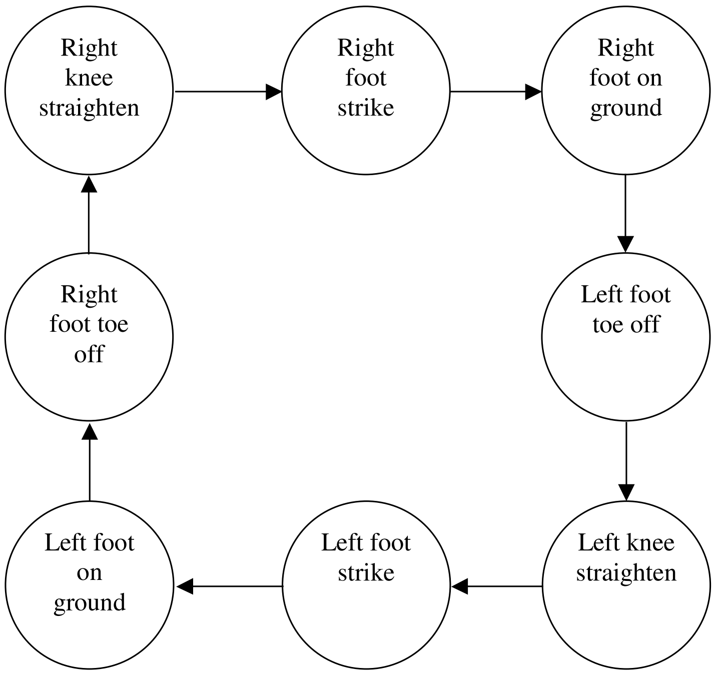


Figure 5.2-1 States of the walking finite state machine

5.3 Knee Straighten

5.3.1 Analyisis

The knee straighten state is entered when the non-support leg of the robot has been sufficiently lifted from the ground, and the foot of that leg is no longer behind the robot. The last condition is necessary to ensure that the non-support foot has been sufficiently moved forward since the toe off state, to prevent the foot from scuffing the ground as the leg is extended. On entry to this state, the centre of gravity is entirely over the supporting foot. Figure 5.1-7 illustrates appropriate conditions for the knee straighten state to begin.

The knee straighten state aims to reposition the non-support foot back on the ground, in a manner that would allow the robot to walk forwards. At the completion of the knee straighten state, the robot will have both feet on the ground, one in front of the other, aiming for the centre of gravity to be equally spread between the feet in the frontal and sagittal planes.

In order to move to the desired position, the non-support leg (the right leg in this case) has to be extended by straightening the knee. The right hip pitch needs to be maintained in order to ensure that the leg will remain out in front of the robot. The right ankle pitch needs to be parallel with the ground on impact, to prevent a premature foot strike (which would occur if the ankle was positioned in a non-parallel manner).

It was hoped that the robot's passive dynamics could be used in order to aid in the straightening of the knee. The initial position of the knee is such, that if it were allowed to swing freely it should automatically begin straightening, and only a small amount of control would have to be used to complete the straightening process. However, the robot's simulated knee-joint had too much friction and would not allow for the knee to swing downwards quickly enough, hence a constant control had to be used.

The centre of gravity in the side-to-side direction has to be shifted from a position that is entirely above the support (left) foot, to one that will be between both feet when the right foot makes contact with the ground. This is achieved by leaning the robot to the non-support (right) side.

The centre of gravity in the fore-aft direction also needs to be shifted forwards to allow the non-support leg to come in contact with the ground. This is achieved by leaning the robot forwards. As the robot leans forwards, the non-support leg will move closer towards the ground.

The rate of change in centre of gravity in the frontal plane should be equivalent to the change in the sagittal plane, to ensure that the robot does not fall forwards or to the side too quickly.

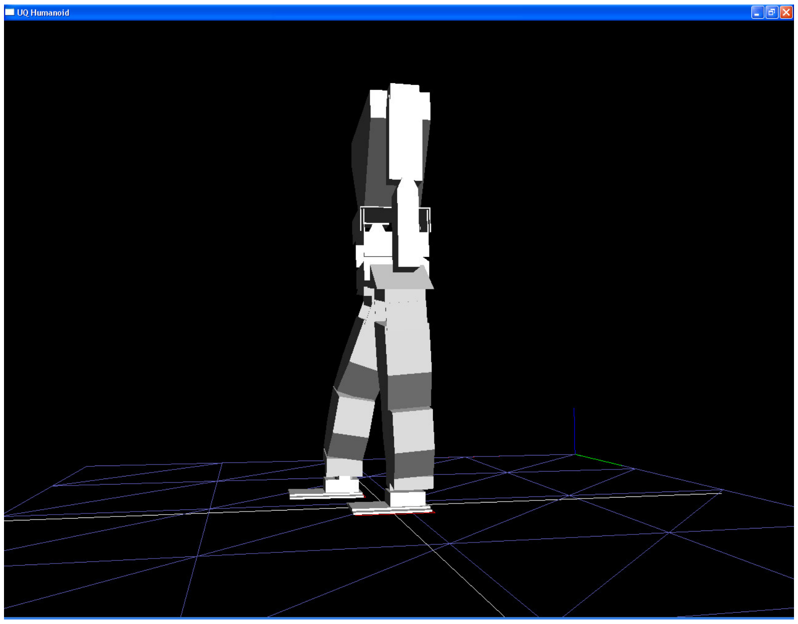


Figure 5.3-1 During the knee straighten phase

5.3.2 Implementation

In order to straighten the non-support (right) knee, a proportional control over the knee angle is used. The final desired angle for the right knee is 90° (knee angles are measured from the perpendicular to the thigh). k tuned to 0.08.

```
desiredRightKnee = currentRightKnee + pDist(k, currentRightKnee,
                        desiredRightKneeFinal);
```

Equation 5.3-1 Knee straighten state right knee angle

In order to calculate the desired velocity to move the right knee to keep up with the desired right knee angle, another proportional control was used. k = 0.08.

```
desiredRightKneeVel = pDist(k, currentRightKnee,
                    desiredRightKnee);
```

Equation 5.3-2 Knee straighten state right knee velocity

The right foot has to be kept parallel to the ground, and controlling right ankle pitch does this. The angle at which to set right ankle pitch is calculated on the premise that the four angles within the polygon that is enclosed by the legs and ground is equal to 360° (Figure 5.3-2). Angles a (left ankle pitch), b (combined hip pitches) and c (right knee) are known, hence d is controlled to complete the polygon's angles sum to equal 360 (Equation 5.3-3).

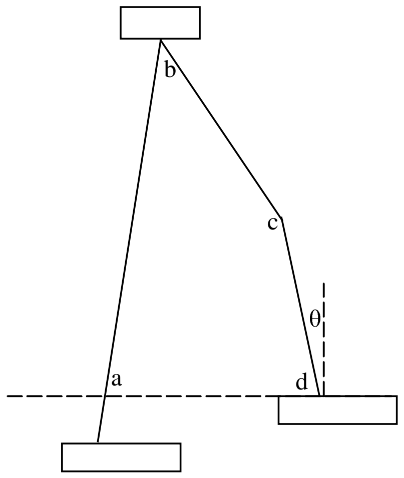


Figure 5.3-2 Knee straighten state pitch angles (side on)

```
desiredRightAnklePitch = -(90 - (360 - ((90 - leftAnklePitch) -
  leftHipPitch + rightHipPitch + (180 - (rightKnee - 90)))));
```

Equation 5.3-3 Knee straighten state desired right ankle pitch

A proportional controller with k = 0.08 is used to calculate the desired velocity of the right ankle pitch.

Centre of gravity in the sagittal plane is controlled to move forward, between the front and rear feet. This is controlled by left ankle pitch. The desired final angle of the left ankle is equal to the angle that the right hip pitch is controlled, in order to achieve the desired placement of the centre of gravity, and also to allow the torso to remain upright (Figure 5.3-3).

Figure 5.3-3 also explains how a suitable stride angle is chosen. The stride angle has to be large enough, to allow, in the fore-aft direction, the feet to be spread apart in the double support phase. The feet have to be spread out enough to allow the centre of gravity in the fore-aft direction to be moved totally over the front foot, so the rear foot can be easily lifted from the ground in the toe off state. The feet cannot overlap in the fore-aft direction; otherwise, a more complicated control method is needed in order to lift the rear foot from the ground.

At the same time, the stride angle cannot be too large, such that the centre of gravity cannot be wholly on the front foot. If the centre of gravity cannot be placed wholly on the front foot, it becomes impossible to lift the rear leg from the ground.

Therefore, a stride angle of 20° was chosen, which meets the above requirements.

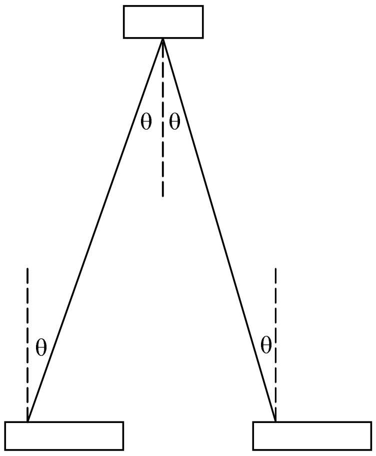


Figure 5.3-3 Knee straighten pitch angles in final state (side on)

The desired left ankle pitch angle is controlled proportionally, as in Equation 5.3-4, with k = 0.03. The velocity control for left ankle pitch is also a simple proportional controller, with k = 0.04. Relatively small values of k are used, since we want the robot to move slowly forwards, in order to minimize the impact forces experienced when the right foot makes contact.

```
desiredLeftAnklePitch = leftAnklePitch + pDist(k, leftAnklePitch,
                                rightHipPitch);
```

Equation 5.3-4 Knee straighten state left ankle pitch angle

Left hip pitch angle is controlled to be equal to the negative of the current left ankle pitch angle. The velocity for the joint is controlled by a simple proportional controller, with k = 0.02.

Centre of gravity in the frontal plane is controlled to move between the feet. This is achieved by having the robot lean towards the right, until the ankle roll value equals zero (roll angle is zero when the leg is upright). The robot is to lean to the side at the same rate as the robot leans forwards, to ensure that the robot does not tip either too far to the front, or too far to the side.

Therefore, left ankle roll velocity is set to be directly proportional to the left ankle pitch, depending on how far the left ankle pitch is from its final position, and how far the left ankle roll is from upright (Equation 5.3-5). In this case, the constant k is used to help tune the velocity controller, in case the robot moves either too slowly of quickly to the side. k = 1.4.

```
desiredLeftAnkleRollVel = (desiredLeftAnklePitchVel /
   (desiredLeftAnklePitchFinal - leftAnklePitch)) *
   (desiredLeftAnkleRollFinal - leftAnkleRoll) * k;
```

Equation 5.3-5 Knee straighten state left ankle roll velocity

As in other cases in which the robot is controlled to lean, right ankle roll and right hip roll were controlled to be at the same angle as left ankle roll. Simple proportional controllers were used to calculate the desired velocity, with k = 0.08 and 0.05 for the hip and ankle respectively. Left hip roll was controlled to be at negative left ankle roll, and a simple proportional controller was used with k = 0.05, to calculate the required velocity.

5.4 Foot Strike

5.4.1 Analyisis

The foot strike state represents the beginning of the double support phase, in which two feet are supporting the robot, one in front of the other. The state is entered when the foot of the non-support leg touches the ground due to movements in the previous knee straighten state (Figure 5.4-1).

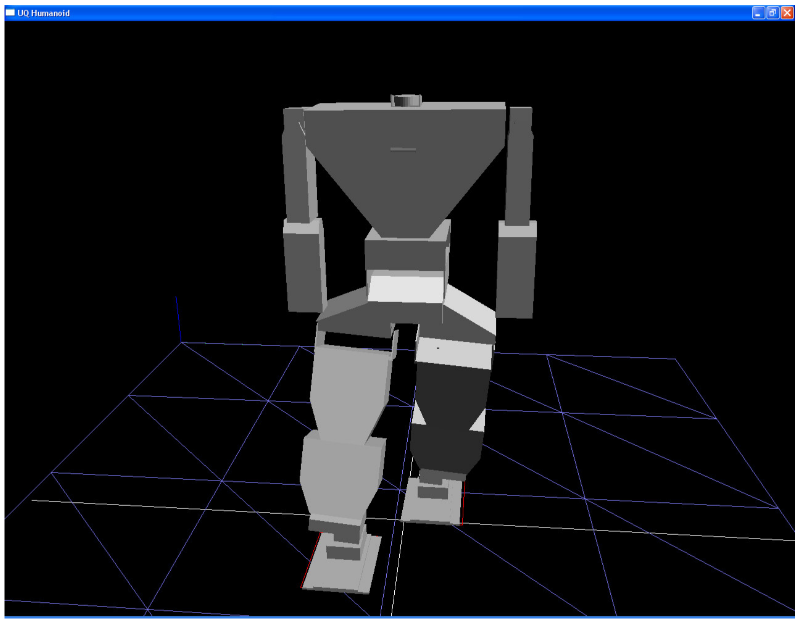


Figure 5.4-1 Entering right foot strike state

Once the foot touches the ground, it is the foot strike state's responsibility to position the centre of gravity over the front foot in the sagittal plane, to ensure that the foot is firmly on the ground. This is necessary so that the control of the robot can then stem from the front foot, to enable the robot to move its front leg into a stable, vertical support position, and also to allow its rear foot to be lifted from the ground.

The original plan for this state was to employ passive-dynamics to some degree, to allow the robot to move forwards on its own momentum. If a passive-dynamics approach could have been used, only a small amount of control in the front foot's pitch would have been required. A passive-dynamics approach would allow the front ankle to freely rotate forwards due to the natural momentum that pushes the robot forwards. The momentum would have been generated by the robot moving forwards in the previous knee straighten state. It was hoped that enough momentum could be generated in the previous state by hastening the movements, to allow the robot to naturally desire to continue moving forwards. However, a suitably large amount of momentum could not be generated to overcome the friction within the simulated front ankle. Additionally, the robot became increasingly unstable as the motions of the previous state were sped up.

The final solution used in order to move the centre of gravity forwards onto the front foot was inspired by motions of various bipeds that climb stairs, or uphill. By leaning their torsos forwards, such bipeds are able to shift their centres of gravity suitably forwards to realize a stable position.

Therefore, in order to shift the centre of gravity over the front foot in the fore-aft direction, the torso was leant forwards (Figure 5.4-2) until the desired centre of gravity was met. Since the stride length is not large, the torso lean angle is not that great.

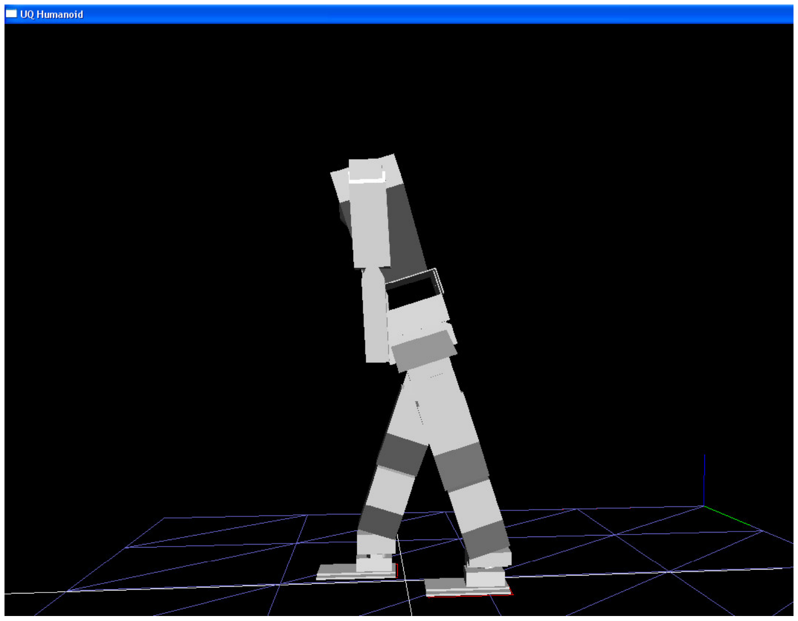


Figure 5.4-2 Leaning torso forwards

The centre of gravity in the frontal plane has to remain equally distributed between the feet in this state, so that in the subsequent foot on ground phase, the rear foot is able to push against the ground to move the robot forwards, without causing the robot to twist.

5.4.2 Implementation

The centre of gravity in the frontal plane is set to be equally between the feet. Since the front leg (right) is entering its support phase, it controls roll movements in order to lean the robot in desired direction to achieve the required centre of gravity.

The desired ankle roll angle is calculated using the exact same controller examined in Equation 5.1-1, however, modified for the right leg. k = 6, and the desired centre of gravity in the frontal plane to be 10cm to the left of the centre of the foot (desiredCogY = 0.1). The desired centre of gravity value of 0.1 is the position that is halfway between the feet (see Figure 4.2-2 for a schematic of GURoo's feet).

Right ankle roll velocity is calculated using a simple proportional controller to provide enough velocity to maintain the desired ankle roll angle. For this controller, k = 0.07.

Right hip and left ankle roll velocities are also calculated using a proportional controller, with k = 0.04 in both cases, and the desired angles equal to the current right ankle roll ankle. The left hip also uses a proportional controller with k = 0.04, with desired angle equal to negative current right ankle roll.

Since the robot has moved forwards enough to allow the front foot to also reach the ground, ankle pitch angles of both feet are no longer required to change, so both velocities of left and right ankle pitch joints are set to zero.

The front (right) knee, however, despite having landed, may not have reached a fully extended position and maybe still slightly bent. Therefore, a proportional control over the knee angle is set up, similar to that of the previous state (Equation 5.3-1), in order to fully extend the knee. Parameters for the controller are k = 0.02, and desired final angle = 90. Velocity of the right knee to meet the desired angle is also calculated using the same controller as in the previous state (Equation 5.3-2), with k = 0.02.

Left and right hip pitch are used to lean the robot's torso forward, in order to move the centre of gravity in the sagittal plane over the front foot. The desired angles are calculated using the proportional controllers in Equation 5.4-1, with the desired centre of gravity in the x direction = –0.05 (i.e. 5 cm behind the centre of the right foot). The value of k = 6. cogX is the distance of the centre of gravity in the x direction, from the centre of the right foot (see 4.3).

```
    desiredRightHipPitch = rightHipPitch - pDist(k, cogX,
                          desiredCogX);
desiredLeftHipPitch = leftHipPitch - pDist(k, cogX, desiredCogX);
```

Equation 5.4-1 Foot strike state hip pitch angle control

Hip pitch velocity controllers are proportional, and have k = 0.06.

The robot moves into the foot on ground state when the centre of gravity moves to be only 8cm behind the centre of the right foot, in the x direction.

5.5 Foot on Ground

5.5.1 Analyisis

The foot on ground phases is entered when the centre of gravity of the robot in the fore-aft direction has been moved suitably over the front foot (Figure 5.5-1), to allow the rear foot to push against the ground. Pushing the rear foot against the ground allows the robot to begin straightening the torso (which was bent forwards in the previous state), and also to allows the rear heel to come off the ground.

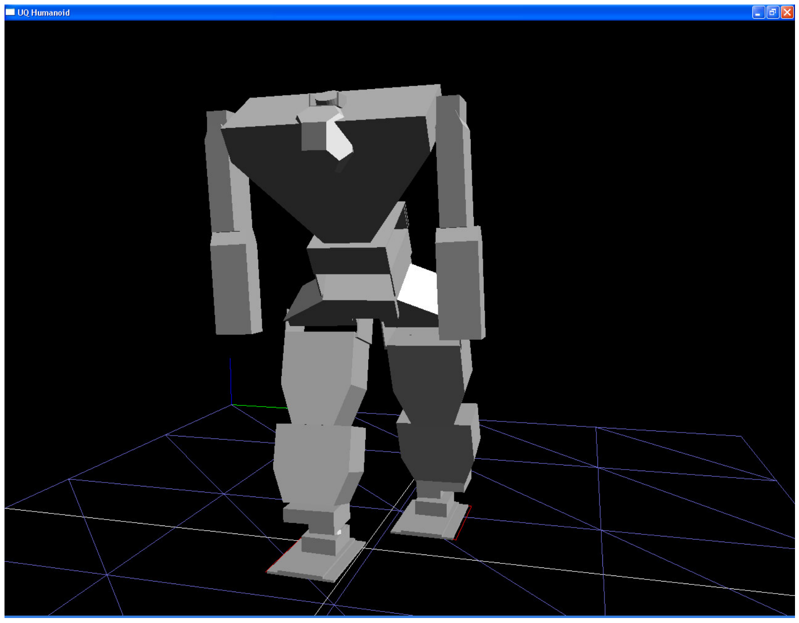


Figure 5.5-1 Entering the right foot on ground state

The desired final position for this state would have the front support leg nearly vertically upright, with the torso bent slightly forwards to counteract the mass of the rear leg. The centre of gravity would be totally above the front foot. This would then allow an easy transition into the toe off state, in which the rear foot is lifted from the ground.

In order to reach the desired position that allows the toe off state to occur, the torso and front leg must be straightened, and the centre of gravity moved completely over the front foot.

Straightening the torso and front leg, whilst still increasing the centre of gravity to be over the front foot, occurs with the aid of the rear foot pushing against the ground. By pushing with the rear foot's toes against the ground, a force is exerted through the rear leg that effectively pushes the robot forward at the hips. This force allows the front leg and torso to become more vertical without compromising the centre of gravity in the fore-aft direction (Figure 5.5-2).

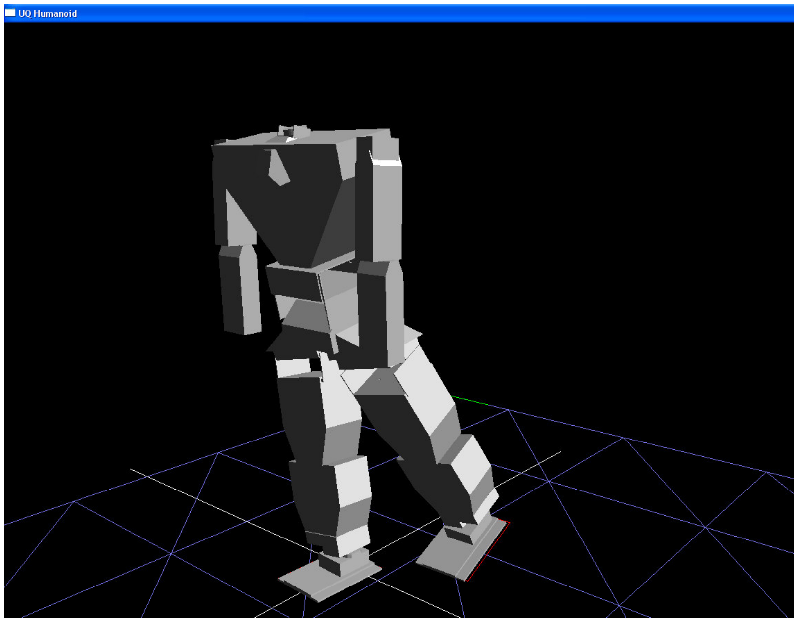


Figure 5.5-2 During the right foot on ground state

The use of the rear foot pushing against the ground also allows for the heel to rise from the ground, which is a desired movement heading towards initiation of toe off.

The centre of gravity in the side-to-side direction also has to be shifted completely over the front foot. As in other states, leaning the robot in the desired direction performs this.

5.5.2 Implementation

The centre of gravity of the robot in the side-to-side direction needs to be shifted over the centre of the front foot. This is performed by using the same proportional controller as in the previous section to control the desired ankle roll, however, this time the desired centre of gravity is set to be completely over the front foot (desiredCogY = 0). k = 6 for this controller.

Roll velocity calculations were performed in exactly the same way as in the previous state, using the same k values.

Left ankle pitch is controlled to push the toes against the ground, in order to nudge the robot forwards, and lift the heel. An extremely simple control method is used, such that the velocity controller creates a velocity to try and reach the desired joint position less 5° (Equation 5.5-1). However, if the front foot ever moves off the ground, the left ankle pitch velocity is set to zero. This ensures that the toes will not push against the ground too much, and cause the robot to fall.

```
desiredLeftAnklePitchVel = pDist(k, leftAnklePitch, leftAnklePitch
                                   - 5);
```

Equation 5.5-1 Foot on ground state left ankle velocity control

Right ankle pitch is controlled so that the right leg becomes vertically upright, to bring the centre of gravity above the right foot. It uses a proportional controller over the angle (Equation 5.5-2) (k = 6, desiredCogFromRightX = 0) and a simple proportional controller over velocity, with k = 0.04.

```
desiredRightAnklePitch = rightAnklePitch + pDist(k, cogFromRightX,
                         desiredCogFromRightX);
```

Equation 5.5-2 Foot on ground state right ankle angle control

Right hip pitch is controlled to move in such a way that it moves the torso into a more upright position. However, the velocity is only set when the centre of gravity in the fore-aft-direction is less than 7cm behind the centre of the right foot, to ensure that the robot will not fall over backwards. As illustrated in Figure 5.3-3, the right hip pitch needs to be set to equal the right ankle pitch, to allow the torso to become upright. A proportional control over the right hip pitch angle was used (Equation 5.5-3), k = 0.03, and a simple proportional controller with k = 0.04 was used to calculate the desired velocity.

```
desiredRightHipPitch = rightHipPitch + pDist(k, rightHipPitch,
                       rightAnklePitch);
```

Equation 5.5-3 Foot on ground state right hip pitch angle control

Right knee pitch is controlled in exactly the same manner in the previous state, to ensure that it is upright.

Left hip pitch and left knee joints are not controlled.

The toe off state is entered when the centre of gravity in the sagittal plane moves above the right ankle, and the torso has reached an angle that is less than 15° from vertical.

5.6 Toe Off

5.6.1 Analyisis

The toe off state begins when the centre of gravity in the sagittal plane has been moved sufficiently above the front foot, and when the torso lean angle becomes closer to a vertical position (similar to Figure 5.5-2).

The aim of the toe off state is to bring the robot back into a position the same as that of Figure 5.1-7, in which the non-support leg is lifted from the ground, the centre of gravity is fully over the supporting foot, and the non-support foot is not behind the robot. Once this state has been achieved, the robot can easily return back into the walking finite state machine, with the only difference being the states are set up for the opposite leg.

The toe off state must lift the rear foot's toes from the ground, bend the knee, and rotate the leg forward in order to swing the leg to a desired position. It is important that while the leg is swinging through, the foot does not scuff the ground. This also has to be achieved while keeping the centre of gravity in a stable position above the supporting foot (Figure 5.6-1).

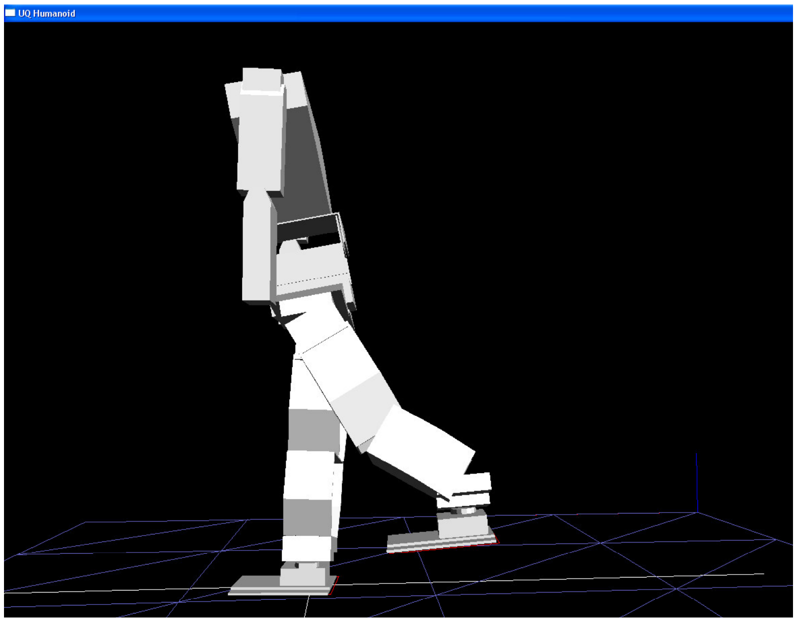


Figure 5.6-1 During the left foot toe off state

5.6.2 Implementation

The centre of gravity in the frontal plane needs to be kept directly above the right foot, and is achieved through the exact same control methods for roll joints as in the previous state.

Right ankle and right hip joints are also controlled in the same manner as the previous state, in order to keep the centre of gravity of the robot above the right foot, and also allow for the torso to become more upright.

Left ankle pitch is controlled in a manner similar to that illustrated in Figure 5.3-2, in order to keep the left foot parallel to the ground. However in this case, the foot being controlled lies behind the support leg rather than in front. Moving the left foot in such a manner will allow the toe to come off the ground.

Once the toe has lifted from the ground, the left knee is controlled to bend inwards, to an angle that is twice the desired stride angle plus 90°, to move the leg into a bent state similar to that of Figure 5.1-6. A proportional control is used to calculate the desired knee angle, with $k = 0.06$ and the final desired angle to be equal to twice the stride angle, plus 90° (Equation 5.6-1). The stride angle, as discussed earlier, is 20°.

```
desiredLeftKnee = leftKnee + pDist(k, leftKnee, 2 * strideAngle +
                              90);
```

Equation 5.6-1 Toe off state left knee angle control

A simple proportional control over knee velocity is used, with k = 0.04.

The left hip is also controlled to rotate in a manner that will swing the left leg through to arrive at the state shown in Figure 5.1-7, however with the legs reversed.

A proportional controller is used to calculate the desired angle of left hip pitch, with k = 0.06 (Equation 5.6-2). A simple proportional controller, with k = 0.04, is used to calculate the desired velocity.

```
desiredLeftHipPitch = leftHipPitch + pDist(k, leftHipPitch,
                              strideAngle);
```

Equation 5.6-2 Toe off state left hip pitch angle control

It is important to note, however, that both hip joints and left knee joint are not moved unless the left foot has come off the ground. Moving these joints whilst the left foot was still in contact with the ground would cause a ground reaction force to come up through the left leg, and push the robot over. Therefore, it is necessary to remove the left leg from the ground before the other joints are moved, to prevent it pushing against the ground caused by the other joints' movements.

Once the robot moves into a position such as that as is shown in Figure 5.1-7, it then continues to progress through the states of the finite state machine starting at the knee straighten phase. However, the left leg is straightened rather than the right.

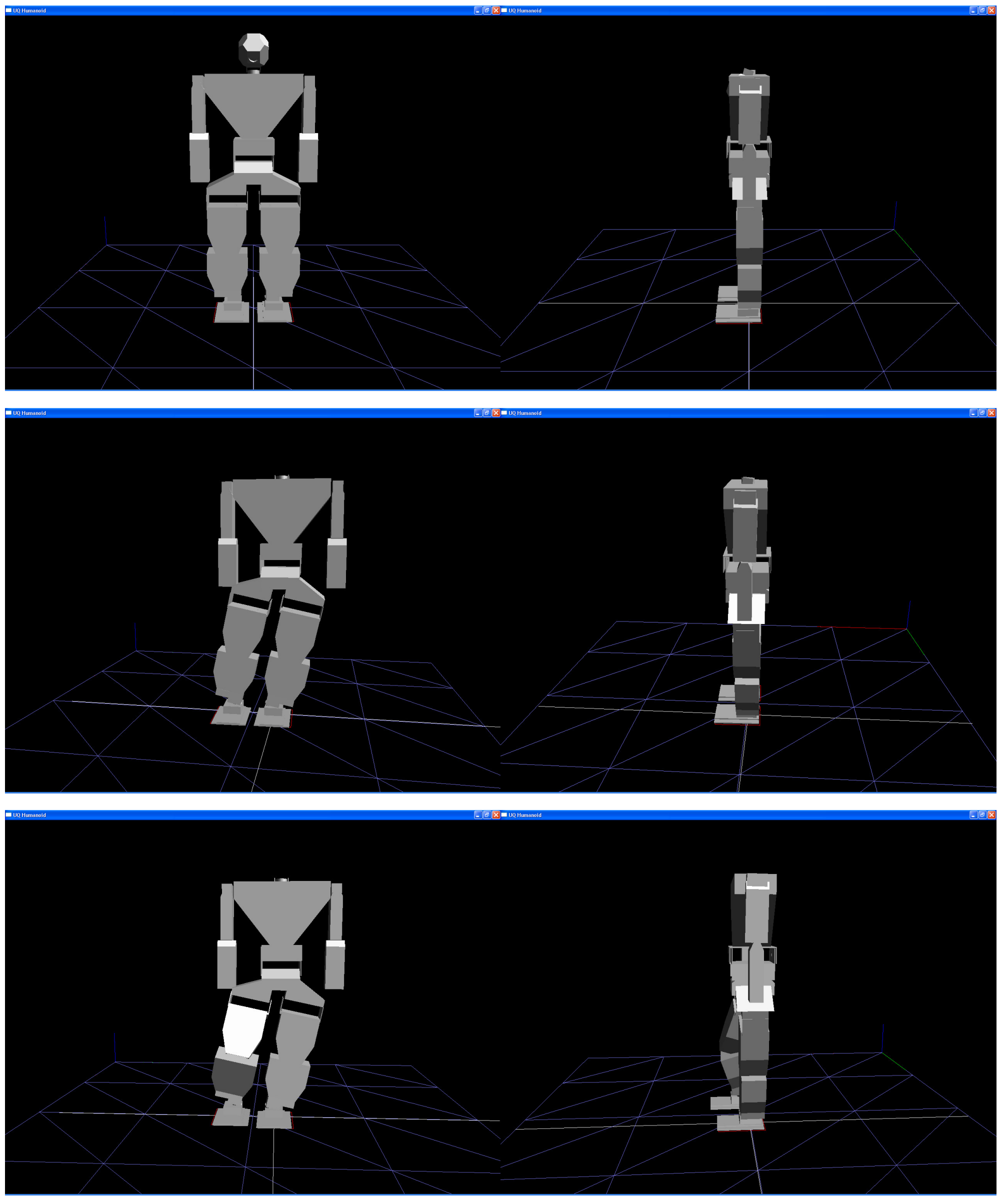
The control methods for the remaining states of the walking finite state machine are exactly the same as the opposite leg counterparts, apart from the obvious necessity to swap the roles of left and right legs.

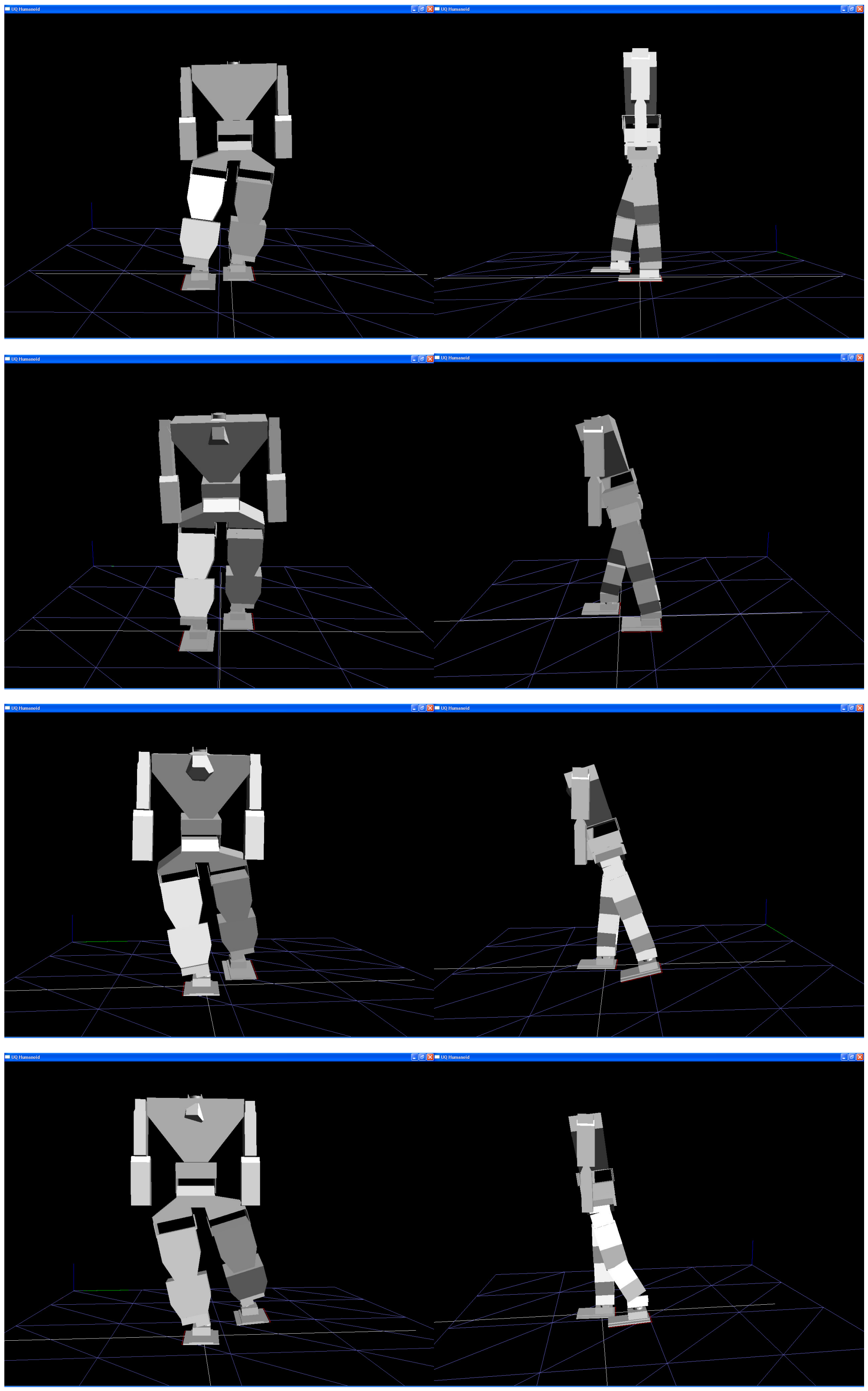
6 Results and Interpretation

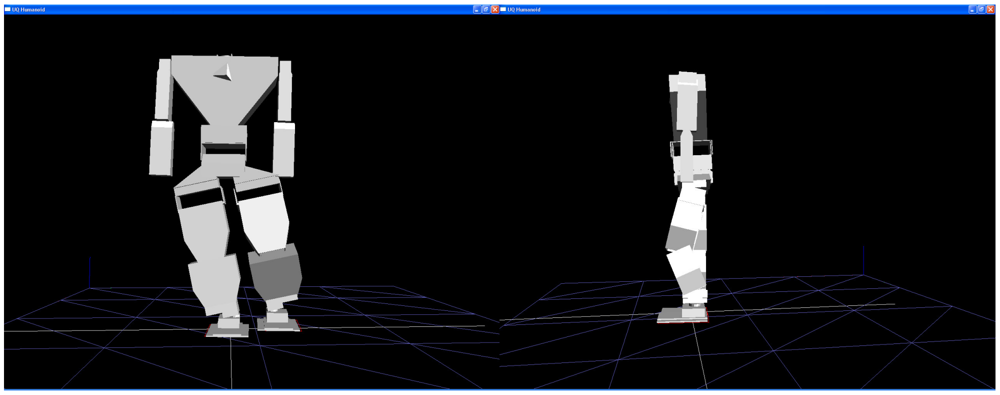
6.1 Simulated Walk

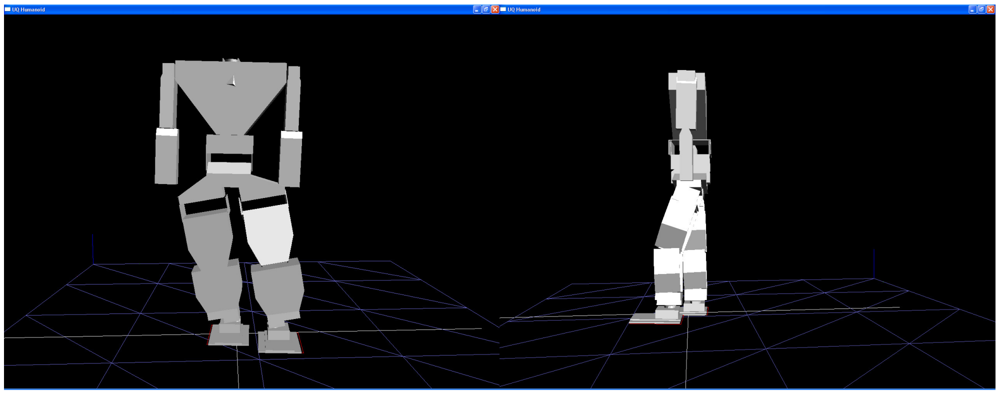
6.1.1 Complete Walking Cycle

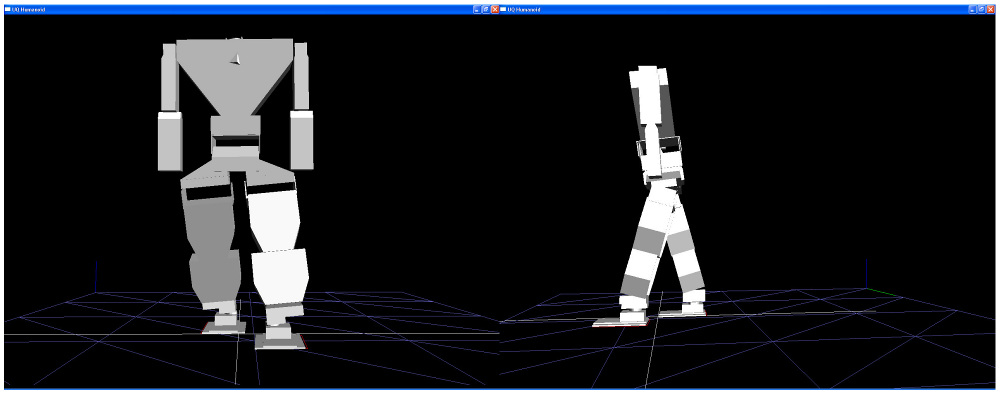
The pictures below illustrate the aforementioned walking cycle in action. In the snapshots of the simulator below, the robot completes two steps, one on each foot, using the control mentioned in 5.2, and is in the ready position to enter the finite state machine to start the third step.

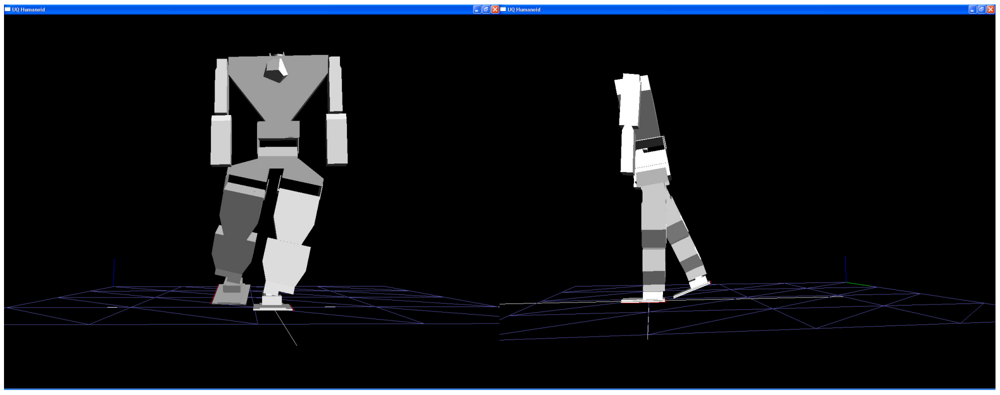



UQ Humanoid


UQ Humanoid
UQ Humanoid


UQ Humanoid
UQ Humanoid


UQ Humanoid
UQ Humanoid


UQ Humanoid
UQ Humanoid

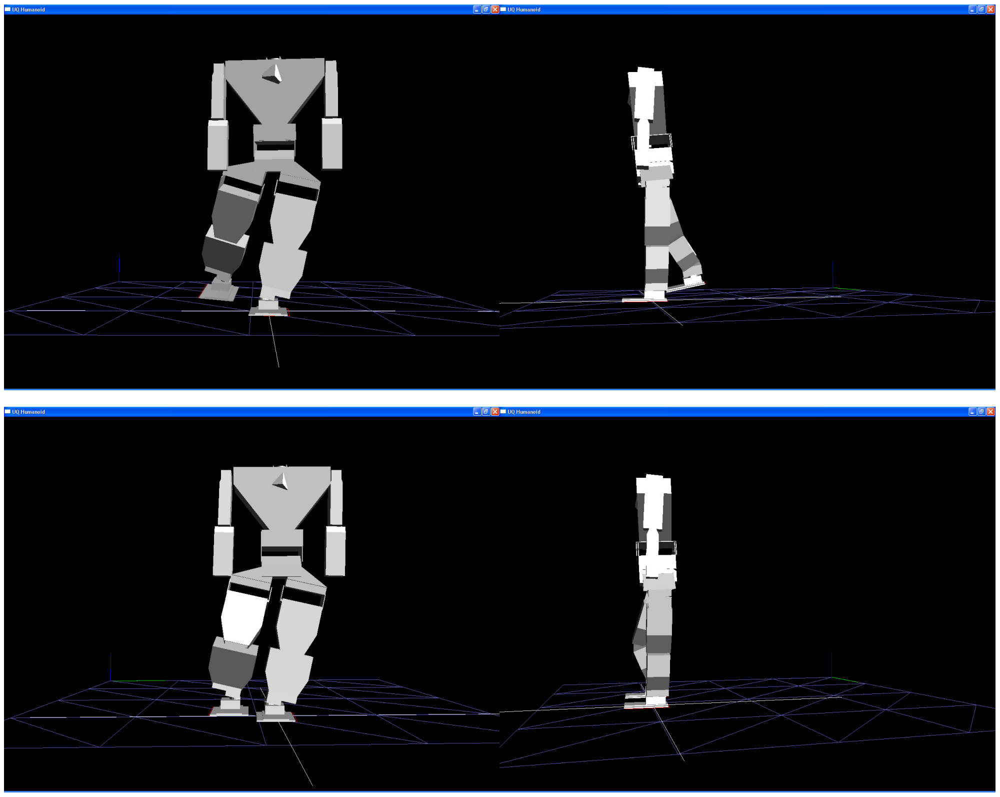


Figure 6.1-1 Walk in action

Walking initiation states, as well as the eight states (four for each leg) of the dynamic walk finite state machine were successfully implemented on the simulated GURoo robot.

The completion of the walking initiation states allows the simulated robot to come to a position in which it is ready to enter the walking finite state machine. In moving towards that position, the simulated robot illustrates the smoothness in the control techniques, as well as the ability of the robot to balance dynamically on one foot.

The completion of the eight walking states has resulted in the successful development of a dynamic gait for the simulated GURoo robot. By repetition of the eight states, the robot can successfully walk. The competence of this walk is discussed below, in the areas of stability, efficiency and speed. A final comparison to the original walk is also provided.

6.1.2 Stability

The robot demonstrates a dynamic ability to cope with various small disturbances while in the walking finite state machine. This is evident from visual observation of the simulated robot whilst in action.

For example, the step sizes that are taken by the robot are not always the same length. A bounce of the front leg, when taking a step, due to impact forces, sometimes causes the robot's step length to vary. However, this does not affect the balance of the robot. It is evident that the robot is quite able to cope with such a disturbance, due to the control mechanisms that aim in shifting the centre of gravity to a stable position.

The impact forces, which are felt hardest in the Z direction, when the robot's foot strikes the ground, are also insufficient to cause the robot to collapse (Figure 6.1-2).

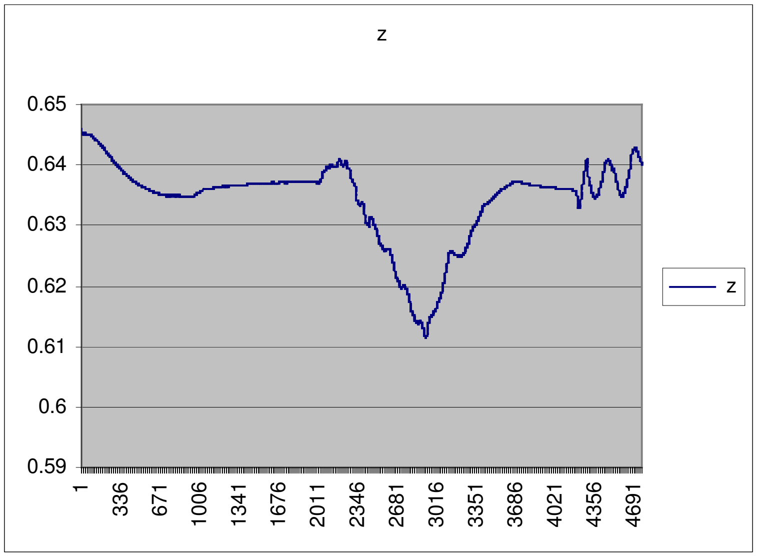


Figure 6.1-2 Dynamic walk Centre of Gravity in Z

It is evident, that despite the rapid decrease in centre of gravity in the z direction, the robot is able to counter the change in centre of gravity, shown by the steep incline back upwards.

Additionally, the dynamic balancing capabilities of the robot are shown during walking initiation, in which the robot leans to one side, lifts up one leg, and balances. Various small perturbations, such as those occurring due to uncontrollable head and arm sway movements are counteracted and controlled via the support leg's ankle and control mechanisms. The robot never falls when in the process of walking initiation.

However, the control methods of the robot are not sufficient enough to control large disturbances, such as those experienced when trying to make use of the robot's passive-dynamics. For example, when moving forwards from the single support to double support phase, this transition has to be taken quite slowly, to ensure that large amounts of momentum do not tip the robot in any particular direction. The robot is unable the effectively control the momentum which faster stepping speeds create.

Additionally, it is doubtful as to whether the robot would be able to achieve a stable control if it were pushed. The robot's control has no "emergency" mechanisms in place to counter a large disturbance, such as a push.

The inability to control the robot's joints above the legs also leads to great instability in the walk. When left to take a number of steps, the robot's spine joints begin to move in large amounts, particularly in the foot on ground state. This results in the robot bending in half, at its spine (Figure 6.1-3).

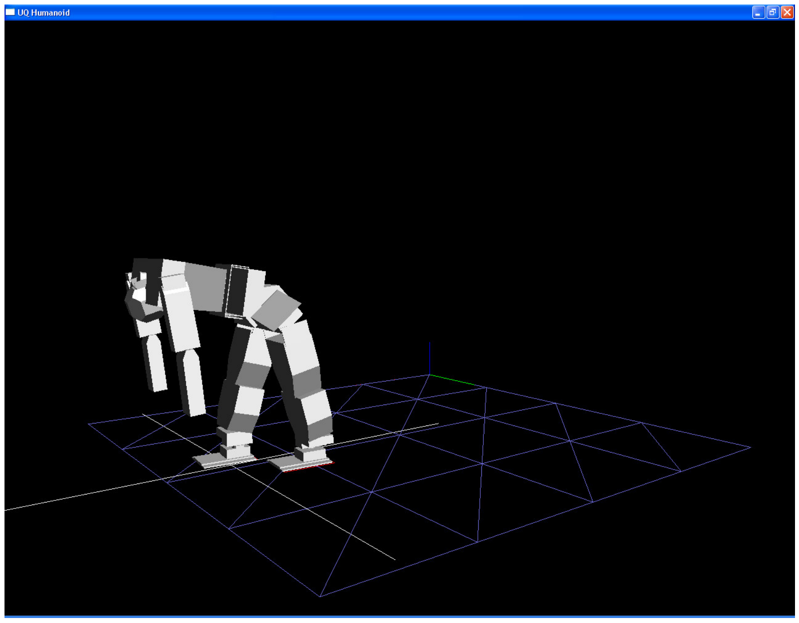


Figure 6.1-3 Instability due to spine bending

It should be noted that if all joints of the robot could be controlled, rather than just the ones within the legs, then this phenomena would not occur. Inability to control all joints stems from the fact that the simulator simply does not perform the instructions relayed to the non-leg joints, this yet to be implemented by UQ.

6.1.3 Efficiency

The walk is quite inefficient, in comparison to many other recently developed gaits, due mainly to the slow speed that it occurs and various movements that are required to keep the robot stable. However, the walk still does contain some efficient movements within it.

Firstly, the support legs are always controlled to be straight at the knee joints. Supporting the robot using a straight leg is much more efficient than supporting the robot with a bent leg, as less torque has to be applied to the motor in order to keep the posture.

Secondly, the robots feet hardly slip. This was a quite a large problem seen in the initial walk of the robot. In the initial walk, although the robot's feet move in a direction that suggests the robot should be stepping forwards, the feet often slip and the robot only manages an small increase in distance from where it had started. This is fairly uncommon in the step that is taken using the dynamic walking algorithm. This is probably due to the fact that the centre of gravity is placed more wholly over the support foot, creating ample friction between the ground and the foot base to prevent slippage.

Thirdly, each step the robot takes results in a clear movement forwards. Unlike shuffle type walks, in which a large number of steps are required to move the robot forwards a certain distance, a single step using this algorithm results in a clear increase in distance.

However, the robot is unable to make use of any passive-dynamics, due to the inability to control momentum, and high amounts of friction between its simulated leg joints. As a result, techniques such as leaning the torso forwards in order to shift the centre of gravity need to be used. Such techniques are quite inefficient, as the robot then has to expend energy in order to perform the movement, and then also has to use energy to undo the moment.

6.1.4 Speed

The speeds of the movements of the robot are quite slow, due to aiming towards increasing stability. As a result, the overall speed of the robot is quite slow. It takes somewhere in the vicinity of 20 seconds for the robot, starting from its initial state, to take one step. However, after taking that step, the centre of gravity of the robot only increased by 20cm. The average speed for a step is somewhere around 0.01 m/s, which is quite slow.

6.1.5 Comparison against Original Walk

The dynamic walk algorithm that has been implemented on the simulated robot, in a number of ways, illustrates various improvements from the initial walk.

The most obvious of these is the walk's awareness of the robot's centre of gravity. The walk is able to, to some extent, adjust the robot's positioning in order to move the centre of gravity into a stable position. The initial walk did not allow for this to occur, and if the centre of gravity strayed from its desired path, then there was no way in which it could be corrected.

Another advantage of the dynamic walk algorithm is its efficiency over the original walk. The original walk has to take five or six steps to reach the same position as the dynamic walk reaches in one step.

Speed is an area in which the original walk still provides better results than the dynamic walk. The speed of the original walk, although still quite slow, at about 0.02 m/s is twice as fast as the dynamic walk. However, this is because of the developmental state that the dynamic walk is in at present. Given ample time to tune the various controllers, it should be possible to increase the walking speed.

6.2 Practical Application

The dynamic walk relies largely on calculations of the centre of gravity dynamically, as well as on the knowledge of the position of its feet. At present, there are no onboard means for calculating the centre of gravity of the robot, unless the simulator itself is run on the robot. However, given the small amount of processing power onboard, running of the simulator is not a viable solution.

Additionally, the walk relies on foot sensors in order to know when its feet are touching the ground. At present, there are no sensors on the robot, at all.

Care was taken to prevent radical changes in desired joint velocities, via use of small velocities and proportional controllers, so as to allow the control method to be used on the actual robot, without ruining the motors. However, it is unknown as yet as to whether enough care was taken, and if the control method is safe to use.

7 Conclusions and Future Work

7.1 Conclusions

A method for the dynamic calculation of the centre of gravity of the simulated GURoo robot, using the Dynamechs package, was implemented successfully. Methods for calculating the velocity and acceleration of the centre of gravity, as well as a means to smooth the velocities and accelerations, were also implemented.

Methods for calculating the actual position of the robot's simulated feet, as well as methods for computing the distance from the centre of the foot to the centre of gravity, broken down into x and y components relative to the foot's direction, were also successfully implemented.

By using the dynamically calculated centre of gravity, as well as the position of the centre of gravity in relation to the foot centres, a simple method for control of a dynamic gait was developed, and implemented successfully. The gait was developed using basic geometric principles in order to calculate joint angles. Examining the centre of gravity, and attempting to place the robot in a position that was similar to that of a human's created the required morphologies of the robot throughout the different states.

It can be concluded that proportional controllers, in conjunction with the calculations of the dynamic centre of gravity, and the distance between the centre of gravity and foot can be used to make the simulated GURoo balance on one foot.

It can also be concluded that the centre of gravity of the simulated GURoo robot can be controlled to be over a point in relation to the supporting foot, using a simple proportional controller to control the desired angle, and another proportional controller to the calculate desired joint velocities of the robot.

It can also be concluded that simple control techniques, and simple feedback from a simulated robot can be used to create a dynamic gait for a humanoid robot.

7.2 Future Work

The current implementation of the walk does not control joints above the hips, simply because the simulator at present does not allow for this to occur. However, it would be quite beneficial to implement some sort of control on other joints, in order to prevent the build up of swaying momentum that is caused by uncontrolled joints. Additionally, it would also be beneficial to the current algorithm if the spine joints were not allowed to move.

As for the current algorithm, a smoother transition between states has to be created. At present, there are periods as large as 3 seconds in which nothing occurs to be happening to the robot, and the robot looks as if it is motionless. However, the robot is actually is in the process of making small changes to its joint angles in order to create the exact morphology required to enter the next state of the finite state machine. The algorithm should have some kind of "give" in this regard, and allow the robot to enter the next state when it gets close enough, rather than have to be exact. This would result in a more natural looking walking motion.

The robot's passive dynamics could also be more thoroughly explored, as to create a faster and more efficient gait. The current gait, in some respects looks natural, but the greater part of the walk resembles a slow plodding, this due to the fact that passive-dynamic concepts were not implemented.

8 Bibliography

1 An anthropomorphic biped robot: dynamic concepts and technological design
Sardain, P.; Rostami, M.; Bessonnet, G.
Systems, Man and Cybernetics, Part A, IEEE Transactions on , Volume: 28 Issue: 6 , Nov. 1998
Page(s): 823 –838

2 Powered flight, child's play, silly wheels and walking machines
McGeer, T.
Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on , 14-19 May 1989
Page(s): 1592 -1597 vol.3

3 Passive walking with knees
McGeer, T.
Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on , 1990
Page(s): 1640 -1645 vol.3

4 An Uncontrolled Toy That Can Walk But Cannot Stand Still.
Coleman, M, Ruina, A, Physical Review Letters April 1998, Vol 80, Issue 16 pp. 3658 – 3661

5 From passive to active dynamic walking
Ohta, H.; Yamakita, M.; Furuta, K.
Decision and Control, 1999. Proceedings of the 38th IEEE Conference on , Volume: 4 , 1999
Page(s): 3883 -3885 vol.4

6 A 3-D passive-dynamic walking robot with two legs and knees.
Collins, SH, Wisse, M, Ruina, A, IJRR In press April 2001

7 Yet Another Humanoid Walking - PDW with Torso under Simple PD Control
Masaki Haruna, Masaki Ogino, Koh Hosoda, and Minoru Asada
Proc. of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001) 2001.

8 On the stability of Biped Locomotion
M. Vukobratovic, A.A. Frank, and D. Juricic,
Biomedical Engineering, IEEE Transactions on , Volume: 17 No: 1, 1970
Page(s): 25-36

9 Balance control analysis of humanoid robot based on ZMP feedback control
Napoleon; Nakaura, S.; SaInpei, M.
Intelligent Robots and System, 2002. IEEE/RSJ International Conference on , Volume: 3 , 2002
Page(s): 2437 –2442

10 The development of Honda humanoid robot
Hirai, K.; Hirose, M.; Haikawa, Y.; Takenaka, T.
Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on , Volume: 2 , 1998
Page(s): 1321 -1326 vol.2

11 Online walking pattern generation for biped humanoid robot with trunk
Hun-ok Lim; Kaneshima, Y.; Takanishi, A.
Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on , Volume: 3 , 2002
Page(s): 3111 -3116 vol.3

12 The intelligent ASIMO: system overview and integration
Sakagami, Y.; Watanabe, R.; Aoyarna, C.; Matsunaga, S.; Higaki, N.; Fujimura, K.
Intelligent Robots and System, 2002. IEEE/RSJ International Conference on , Volume: 3 , 2002
Page(s): 2478 –2483

13 Design and construction of a series of compact humanoid robots and development of biped walk control strategies
Furuta, T.; Tawara, A.; Okumura, Y.; Shimizu, M.; Tomiyama, K.
Robotics and Autonomous Systems, Volume: 37, 2001
Page(s): 81 – 100

14 Control of a Simulated, Three-Dimensional Bipedal Robot to Initiate Walking, Continue Walking, Rock Side-to-Side, and Balance
Parseghian, A.S.
Masters Thesis, Science in Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2000.

15 Exploiting Natural Dynamics in the Control of a 3D Bipedal Walking Simulation
Pratt, J. E.; Pratt, G. A.
Climbing and Walking Robots, 1999. International Conference on, September 1999

16 http://www.ai.mit.edu/projects/leglab

17 Biped dynamic walking using reinforcement learning
Benbrahim, H.; Franklin, J.
Robots and Autonomous Systems, Volume: 22, 1997

18 Reinforcement learning with fuzzy evaluative feedback for a biped robot
Changjiu Zhou; Qingchun Meng
Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on , Volume: 4 , 2000
Page(s): 3829 -3834 vol.4

19 A method for co-evolving morphology and walking pattern of biped humanoid robot
Endo, K.; Yamasaki, F.; Maeno, T.; Kitano, H.
Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on , Volume: 3 , 2002
Page(s): 2775 -2780 vol.3

20 http://dynamechs.sourceforge.net/