# A Torso Driven Walking Algorithm for Dynamically Balanced Variable Speed Biped Robots

by

ALISTAIR JAMES SUTHERLAND

BSc (IT), University of Western Australia, 1991

BE (EE), University of Western Australia, 1993

DISSERTATION

This thesis is presented to the

School of Electrical, Electronic and Computer Engineering

for the degree of

Doctor of Philosophy of Engineering

at

The University of Western Australia

June 2006

# Abstract

As a contribution toward the objective of developing useful walking machines, this dissertation considers solutions to some of the problems involved with bipedal robot development. The main area of focus involves control system design and implementation for dynamically balanced walking robots.

A new algorithm "Torso Driven Walking" is presented, which reduces the complexity of the control problem to that of balancing the robot's torso. All other aspects of the system are indirectly controlled by the changing robot state resulting from direct control of the robot's torso. The result is literally a "top-down" approach to control, where the control system actively balances the top-most component of the robot's body, leaving the control of the lower limbs to a passive "state-driven" system designed to ensure the robot always keeps at least one leg between the torso and the ground.

A series of low-cost robots and simulation systems have been constructed as experimental platforms for testing the proposed new control system. The robots have been designed to balance on "point" feet, and so the control system must be able to dynamically maintain balance, while moving at a variable velocity.

The Torso Driven Walking control system achieves a fully dynamic, variable speed walking behaviour that does not rely on maintaining a stable supporting polygon for balance. In addition, the system exhibits a high degree of tolerance for low frequency "bias" or "drift" errors. These measurement errors are commonly encountered when using sensors for detecting torso inclination.

# Acknowledgements

In completion of this Thesis I would like to thank several people for their support and valuable contributions.

Direction – A/Prof Thomas Bräunl

Proof reader and constant support – Virginia Sutherland

Inspiration – Cormac Sutherland

Assistance with dynamics revision – Dr. Nathan Scott

Genetic algorithm introduction and advice – Adrian Boeing

Advice, encouragement, coffee conversation and many ideas –

Martin Wicke

Philippe Leclercq

Uli Mezger

**Table of Contents**

# 1 Introduction

## 1.1 Problem identification

Human beings, together with almost all other creatures on this planet, exhibit modes of locomotion that are far more adaptable, wide ranging and robust than any machine that has yet been developed. The goal of research into walking robots, and bipedal walking research in particular, is to work towards the development of robots that can walk, run and climb as well as a human being, or other animal. The fact that this has not yet been achieved is evident by the absence of walking machines in everyday life.

As a contribution toward the objective of developing useful walking machines, this dissertation will consider solutions to some of the problems involved with bipedal robot development. The main area of focus involves control system design and implementation. A new biped walking control system, "Torso Driven Walking" is introduced. Other areas of concern in biped robotics research, including robotic design and sensor selection and interpretation will also be considered. A series of low-cost robots and simulation systems have been constructed as experimental platforms for testing the hypotheses with respect to these areas of interest in the field of walking robots.

Legged robots have the potential for significant advantages over conventional, wheeled robots. The most often cited is the ability to navigate more difficult terrain, and negotiate obstacles where a wheeled vehicle would not be able to pass. Commonly identified applications for biped robots include use as automata in rough terrain, as vehicles to assist the disabled or as robotic explorers in space missions. These visions are far from realised. The problem currently is that walking robots are much more expensive, complicated to build, and difficult to control, than wheeled or even flying machines.

The difficulties in developing walking machines are the primary reason for the interest in investigating bipedal robots. More ambitious uses for a walking robot can be invented, but they often sound rather contrived. Many of the commonly identified applications require such a high level of robotic sophistication that it will be many years before they could hope to be seriously considered practical – if they ever are. Of course, if we never start looking into the problems, robust walking machines capable of performing the more ambitious goals of researchers in this field will continue to remain solely in the realm of science fiction.

The problem of making a robot that can walk well is worth investigating because it has been recognised as something that is harder than it looks, and requires integration of solutions to a range of problems that can be extrapolated to other fields of robotic research. Walking robots are a non-trivial test platform for developing and evaluating solutions to robotic challenges ranging from sensor interpretation to artificial intelligence and adaptive control. Areas of focus for this thesis are:

<u>Topics of interest:</u>

- Control system fabrication.
- Sensor selection and interpretation.
- Software modelling techniques.
- Issues associated with porting solutions from a simulated environment to a physical system.
- Appropriate application of adaptive control methodologies.

### 1.1.1 State of research today

There are several hundred research groups actively developing bipedal walking robots and related technologies. For example, one web catalogue has sixty three biped robots:

http://www.walking-machines.org/

Most are slow-moving statically balanced walkers. Many have enormous feet so they do not fall over. They tend to be heavy, rigid and underpowered, with a typical walking speed of 10 or more seconds per step. Many, including our own prototype robot "Johnny Walker", simply execute pre-determined joint trajectories and are thus not much different from expensive wind-up toys. In order to achieve more robust robot walking, more sophisticated control systems are required.

The following robots are commonly acknowledged as being among the "leading edge" in the field of humanoid walking robots, in terms of robot construction and control. All of these robots are the result of many years' research, and considerable financial investment.

#### 1.1.1.1 "Qrio" (Sony)

In December, 2003, Sony unveiled a new version of their Humanoid robot "Qrio", that is able to run. In humanoid research, running is commonly defined as a gait where both feet leave the ground during execution of the steps. The running ability currently being reported for "Qrio" carries the robot for a distance of 3-4m, at a speed of 14m/minute. Qrio's feet are airborne for a total of only 20ms during each step. Considering Qrio's size of approximately 60cm, this running performance is equivalent to a human taking 142 seconds to run 100m, but stopping after 12m. While – according to the commonly held definition – the gait described can technically be called running, there is still a lot of work to be done before a robot will be demonstrated to run like a human.

Image source: http://www.dottocomu.com/b/archives/cat_robots.html

**Figure 1-1 Sony's humanoid robot, "Qrio".**

Depending on which report you read, Qrio's price would range anywhere between the cost of a small car, to that of a luxury car. Since Sony have not yet made any models available for purchase, all we can take from this is the fact that a substantial investment must have been made by Sony to get Qrio to the state it has reached. For the time being, Qrio's roles are likely to be restricted to public relations work for Sony. Qrio's recent duties have ranged from conducting the Tokyo Philharmonic Orchestra (playing Beethoven's 5th Symphony), to performing dance routines at press conferences.

#### 1.1.1.2 "Asimo" (Honda)

The culmination of 18 years development, "Asimo" is the latest in a line of humanoid robots coming from the Honda research laboratories. Standing at 1.2m in height, and weighing 52kg, Asimo is the size of a human child. With twenty six degrees of freedom, including six in each leg, Asimo exhibits a wide range of possible movements. Pressure sensors in the feet, a gyroscope and acceleration sensors in the torso are inputs into a variation of the zero-moment-point control system (see Chapter 2.3.1), which is used to adjust a series of pre-generated joint trajectories. The resulting walking pattern can reach a top speed of 1.6 km/h. To continue the comparison made with Qrio's running speed, this performance is equivalent to walking 100m in 225 seconds – or 3¾ minutes.

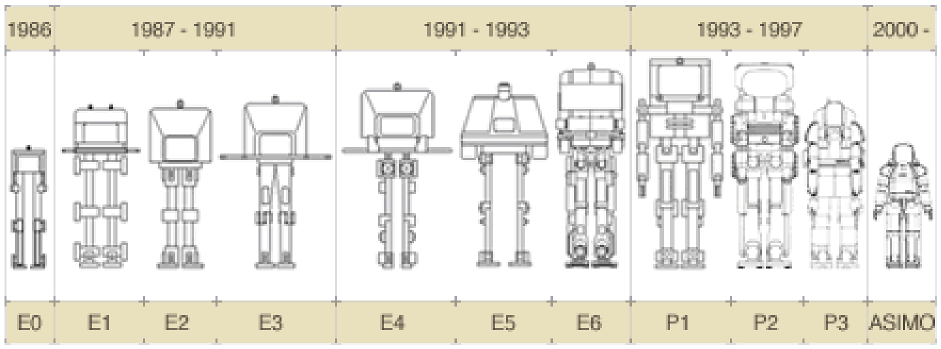


Image source: http://world.honda.com/ASIMO/history/

**Figure 1-2 Eighteen years of Honda bipeds**

As with Sony's robot, at present Honda has declined to suggest an estimated price to purchase an Asimo unit. However Asimo is finding limited commercial application, Honda is reported to be renting Asimo units to four companies, including IBM Japan and a science museum in Tokyo, for a reputed price tag of $163,000 a year.

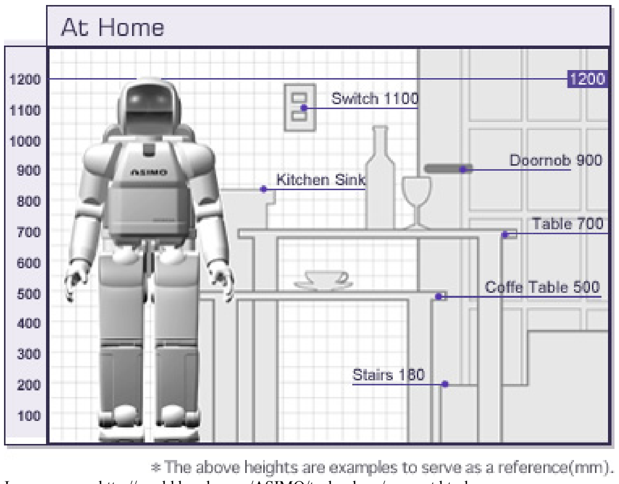


Image source: http://world.honda.com/ASIMO/technology/concept.html

**Figure 1-3 "Asimo" at home**

Like the Qrio robot for Sony, for the time being the robot Asimo's main role is to provide a demonstration of the company's technical expertise. Asimo has appeared in a number of advertisements for Honda, and one unit was recently making a public relations tour of high schools in North America [92].

### 1.1.1.3 MIT Leg Laboratory

MIT's legged laboratory has a long history of research into legged robots. Beginning with the demonstration of a planar hopping robot in 1982, the Leg Laboratory has developed a number of robots demonstrating walking and running gaits. Early robots such as the "3D-Biped" have simplified "telescoping" legs, and move by hopping to keep balance.

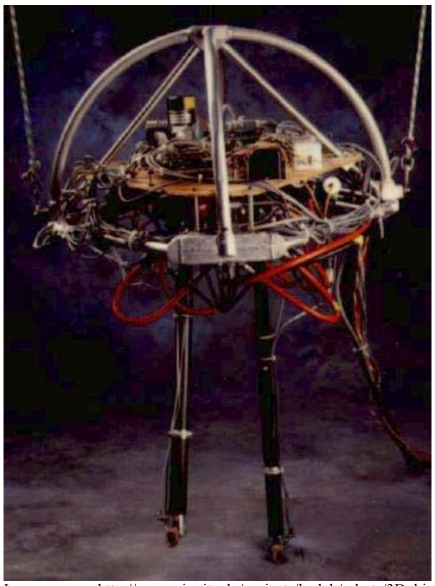


Image source: http://www.ai.mit.edu/projects/leglab/robots/3D_biped/3D_biped.html

**Figure 1-4 MIT Leg Laboratory – "3d-Biped"**

The laboratory's current robots exhibit fully articulated biologically inspired legs. Troody is a well known robot, built to resemble a Troodon (a species of dinosaur). This robot exhibits sixteen degrees of freedom, with seven in each leg. Each toe is independently actuated. In order to provide joint behaviour that appears to more closely mimic biological structures, Troody employs a technology called "series elastic actuators". This simply means that motors to drive the robot joints are attached to springs, providing more natural looking joint characteristics than traditionally actuated robots.

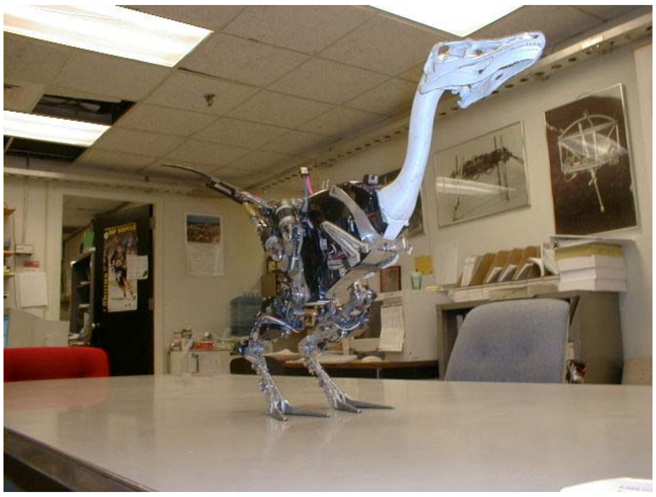


Image source: http http://www.ai.mit.edu/people/chunks/chunks.html
**Figure 1-5 MIT Leg Laboratory – "Troody"**

The researchers responsible for developing Troody have formed a spin-off company, "Dinosaur Robots Inc", through which they hope to sell future versions of dinosaur inspired robots four use in museums, the entertainment industry, and as expensive toys.

### 1.1.2 Torso Driven Walking control system

Torso Driven Walking is a new control system for balancing biped systems, introduced in this thesis. The Torso Driven Walking control system is a "top-down" inverted pendulum inspired control system. Inverted pendulum models are often used to describe biological walking systems, such as in [57], [59], and have been shown to generate very efficient walking gaits [58]. Inverted pendulums require truly dynamic balance, and yet the classical solution to the problem of inverted pendulum control is surprisingly simple. The equation at the heart of most inverted pendulum inspired control systems is expressed by (1), where the balancing horizontal force can be determined using a simple linear relationship between four state variables [26].

$$f_x = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x} \qquad (1)$$

In Equation (1), $\theta$ and $x$ are respectively the inclination angle and displacement of the pendulum, whereas $\{a, b, c, d\}$ are simple scalar values. The Torso Driven Walking control system uses this equation to balance the robot's torso through the application of a torque to the supporting leg's hip joint. This leaves control of the robot's feet to an ancillary algorithm, which has no need to directly consider the problem of balance, and is merely responsible for ensuring there is always a foot between the robot and the ground. In this way, a robot balancing using the Torso Driven Walking control system walks as a consequence of keeping balance, rather than attempting to maintain balance while walking.

Some of the advantages realised by the Torso Driven Walking control system include:

- The inverted pendulum based balancing algorithm (1) is the only component requiring an absolute inclination reading. Chapter 5.7 demonstrates this component can robustly handle sensor noise – especially gyroscope "drift" or "bias". This allows inexpensive solid state gyroscopes to be used as the inclination sensor.
- The TDW algorithm is relatively easy to explain, and tuning parameter behaviours can be described intuitively.
- The control system is modular in nature, allowing distinct components of the system to be developed and tested in isolation to the complete system.
- The control system does not require an accurate system model of the robot being controlled.

- Foot placement is arbitrarily determined, allowing for the possibility of selecting footholds according to terrain, rather than to maintain balance.
- All of the balancing torque, as well as a significant component of translational forces are generated by the hip joint actuators, which can be mounted within the robot's torso.

The Torso Driven Waking control system is described in greater detail in Chapter 3, and is the focus of many of the experiments presented in this thesis.

## 1.2 Research goals

The goal of this thesis was to produce a simple, robust, low-cost biped walking machine. With this strategic objective in mind, the "primary" goals were:

Primary goals:

- To design and build a robot that exhibits a natural looking gait.
- To implement a control strategy that does not require a precise kinematic/kinetic model.
- To implement a control strategy that allows operators to command the robot to move with a specific, variable target velocity.
- To accommodate moderate amounts of sensor noise without causing the robot to fall.
- Demonstrate that the Torso Driven Walking control system is a viable control system for biped walking.
- To keep power sources and computers on board.
- To restrict the developmental costs to the constraints of a limited budget.

Unfortunately, even the simplest biped robot is an extremely complex system. Usually there are a high number of degrees of freedom, each joint is driven by some kind of actuator, and state information is read by a number of sensors. Imperfections and noise creep into the system in all of these areas – especially in collecting and interpreting sensor data, and modelling actuator behaviour. In order to regulate this complex system, a control algorithm needs to use noisy state information observed by the sensors to impose order on the system, with the final result being the robot walks.

If you have just built a robot, installed some sensors, and written a control algorithm – the most likely resulting robotic behaviour is to consistently fall over. It is very difficult to troubleshoot such a complex system, with so many different potential modes of failure. If the robot is falling, is the control system at fault? Perhaps the sensors are not giving accurate information? Perhaps they are not even giving the right kind of information? The list of possible causes is very large. Usually more than one problem will be present at once – even if you fix one of the failure modes, you may not realise it as the robot is still falling due to another problem.

The strategy employed to cope with this complexity has been to develop a series of small, inexpensive robots. Each robot has been used to help isolate and attempt to find solutions to issues that have been identified from previous work as critical to the development of low-cost walking machines. Additionally, a software simulation has been developed for each new robot, allowing control systems to be designed and tested in a wholly controllable artificial environment before being exposed to the challenge of real world control. Successive robots carry forward solutions that were proven on their simpler predecessor, and introduce a limited number of new challenges. The idea is that each new robot is a step closer to realising our primary goal.

## 1.3 Robotic experimental platforms

### 1.3.1 Johnny Walker

Johnny Walker, and experiments conducted using Johnny Walker, will be more fully described in Chapter 4.

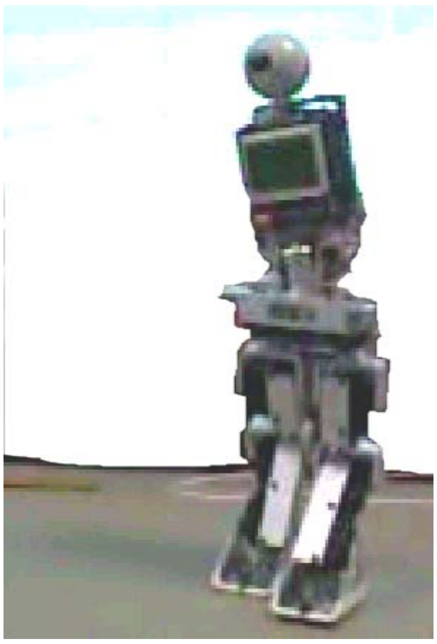


**Figure 1-6 Johnny Walker**

Johnny is a 9 degree-of-freedom, SERVO actuated, biped robot. Johnny was designed and built as a first pass at building a walking robot. While not able to achieve anything close to our stated goals for walking using this robot, it was a useful tool for identifying areas critical to the development and control of walking robots. Some key issues identified as a result of experiments with Johnny Walker were:

- Importance of appropriate sensor selection and interpretation.
- Desirability of force control of joints.

### 1.3.2 Ballybot

The Ballybot, and experiments conducted using the Ballybot, will be more fully described in Chapter 5.

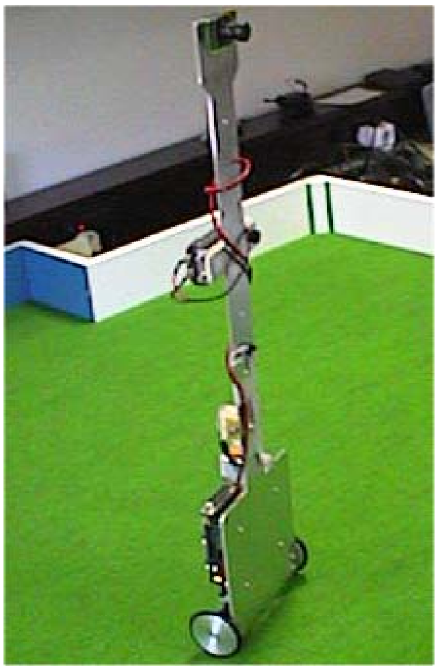


**Figure 1-7 Ballybot**

The Ballybot is an inverted pendulum inspired robot. It was designed primarily as a test platform for experiments with sensors for robot balance. The robot needs to measure its orientation, especially inclination angle, in order to determine a torque, which when applied to the robot's wheels keeps the Ballybot from falling over. The idea was that if a collection of sensors can be identified that allow this robot to balance, the same sensors will also be suitable for providing orientation information to a more complicated walking robot.

Topics investigated using the Ballybot included:

- Sensor selection and integration.
- Balancing algorithms
- Artificial neural networks as a tool for developing control systems from scratch

### 1.3.3 Legbot

The most recent experimental platform, this robot extends the Ballybot by replacing wheels with a pair of two degree of freedom legs. The feet have been designed to approximate "points" when viewed in the sagittal (x-y) plane, but to be very wide in the z direction. This design constrains the robot motion to the sagittal plane, and reduces the 3D system to a planar 2D robot. This simplifies the task of modelling the robot dynamics, and therefore makes the development of a control strategy simpler, while still being extensible to the more general 3D case.

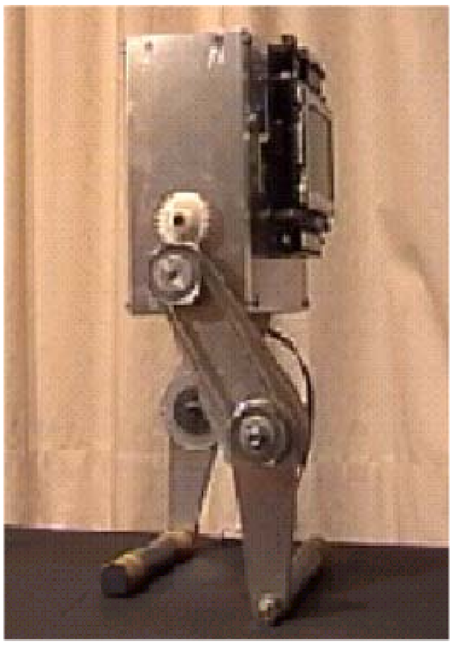


**Figure 1-8 Legbot**

Topics investigated using the Legbot included:

- Force control of joints
- Balancing control systems for legged robots
- Investigate genetic algorithms as a tool for tuning a generic control system

The Legbot will be described in more detail in Chapter 6.

## 1.4 Thesis organisation

Chapter 2 Control systems for walking robots. Presents a review of the development of control systems for biped walking, including a summary of recent research.

Chapter 3 Torso Driven Walking. Presents a detailed design of the Torso Driven Walking control system, which is the subject of the bulk of this thesis.

Chapter 4-6 Investigations and Experiments

Chapter 4 Experiments with Johnny Walker and Jack Daniels

- Preliminary experiments with an existing robot platform are used to highlight areas of concern in the development and application of biped robot systems. The interrelationships between control system, sensors, actuators and physical construction are highlighted.

Chapter 5 Ballybot experiment

- The Ballybot is used as a test system for the most critical component of the Torso Driven Walking control system (the Torso Attitude Controller subsystem)
- Issues relating to the implementation of Torso Driven Walking on a physical Ballybot robot are identified and addressed.
- An experimental comparison is made between Torso Driven Walking and a "first-principles" Artificial Neural Network control system for the Ballybot.

Chapter 6 Legbot experiments

- A simulation of the Legbot is used as a test system for the complete Torso Driven Walking control system.

- Genetic Algorithms are developed as a tool for automatic tuning of the Torso Driven Walking control system.
- Issues relating to the porting of a Torso Driven Walking controller to alternative environments are investigated.
- An experimental comparison is made between Torso Driven Walking, and an alternative biped control system known as Virtual Model Control.
- Issues relating to the implementation of Torso Driven Walking on a physical Legbot robot are identified.

| | |
|---|---|
| Chapter 7 | Conclusion. A summary of the process followed during the course of this thesis is presented, together with a review of the research goals and a number of ideas for future research. |

# 2 Control Systems for Walking Robots

"Motion is created by the destruction of balance, that is, of equality of weight, for nothing can move by itself, which does not leave its state of balance, and that thing moves most rapidly, which is furthest from its balance." - Leonardo da Vinci.

While there are many aspects of robotic research that must be integrated successfully to produce a walking biped robot, the area of most critical importance is the robot's control system. Developing a suitable control system for experimental robots has been the major focus of the work done towards preparing this thesis. As a result, it is appropriate to review some of the history of biped robot control, as well as introduce some of the terminology used to discuss biped robot control systems.

## 2.1 Some ancient history

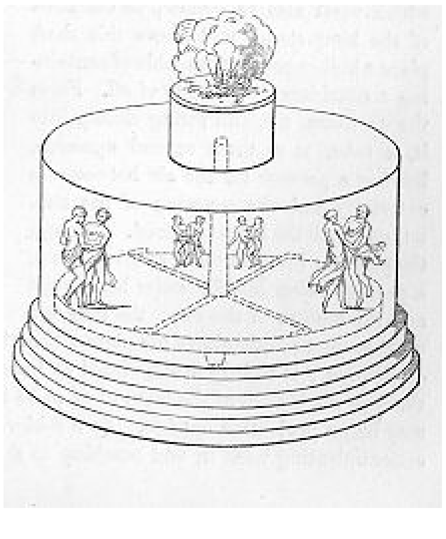


Image source: http://www.history.rochester.edu/steam/hero/section70.htm

**Figure 2-1 Figures made to dance by fire on an altar, an early example of steam powered automata attributed to the Greek mathematician Heron of Alexandria.**

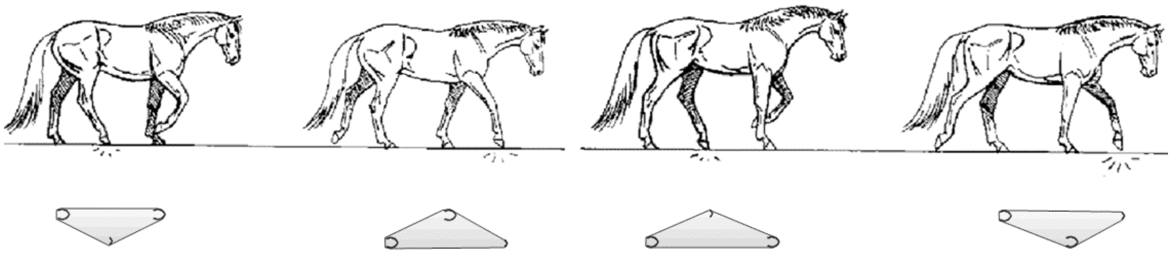
Throughout recorded history, people have always had a fascination with the idea of building artificial men. Legends of artificial people and animals are common to many cultures. As well as telling myths featuring artificial men, people have been building animated models of human beings, or automata, for thousands of years.

Despite this long interest, it is only recently that people have begun to have some success making biped robots walk, as recent advances in technologies and materials make the actuators and control systems required for robot walking possible. Of all these advances, it is the advent of powerful digital computers in the last century, which has been most critical in enabling researchers to find ways to generate the dynamic, adaptable and timely control signals required to keep a system as complex as a biped robot balanced while walking.

## 2.2 Humanoid walking classifications

### 2.2.1 Supporting polygon

Many of the approaches to controlling robot walking behaviour rely on the control system tracking the robot's orientation in space with respect to its "footprint" on the ground. This "footprint" is an imaginary area drawn on the ground, such that all parts of the robot in contact with the ground at a given time delimit the outside border of the area. If only one foot is in contact with the ground, this area is the contact area of the foot. If multiple feet are on the ground, this area includes the contact areas of the supporting feet as well as the area between the contact feet, even though there may not be any part of the robot contacting the ground at the intermediate points. The term used in literature to describe this shape or projection on the ground is the "supporting polygon" [62].



Horse walking gait images sourced from: http://www.angelfire.com/tx2/kidshorses/gaits.html

**Figure 2-2 – The supporting polygon associated with a horse's walking gait. The polygon's shape continually changes as different hooves contact the ground over the course of the horse's walk.**

A basic understanding of physics can demonstrate that if a structure is immobile, or *static*, and its centre of mass projected onto the ground (GCoM) lies within the structure's supporting polygon, the structure will remain standing. The structure is statically stable. When the structure's GCoM falls outside the bounds of its supporting polygon, then the structure will tip over. In the diagram shown by Figure 2-3, structure A is statically stable, while structure B is not.

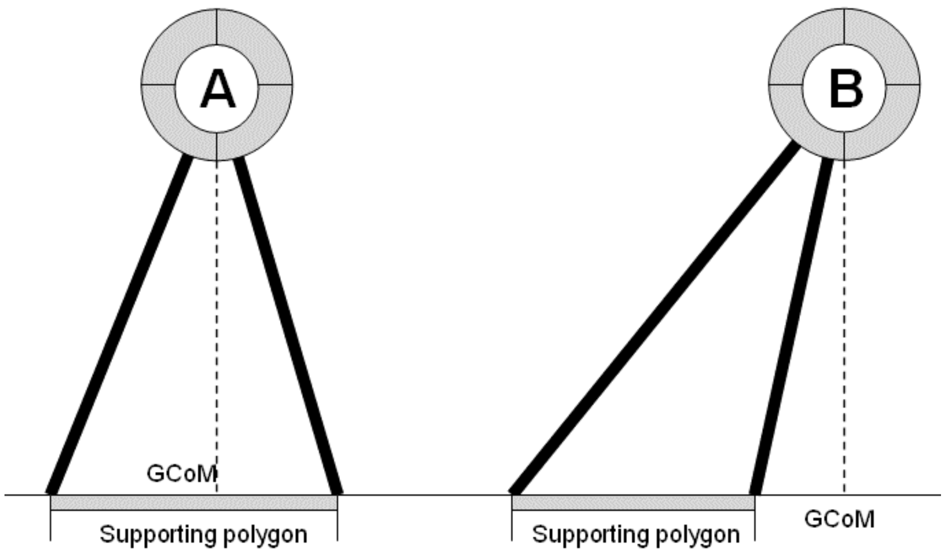


**Figure 2-3 Structure (A) is statically balanced, while (B) is not. In the absence of any external forces, structure (B) will fall.**

If the dynamic forces associated with a structure's motion are small enough, this rule can be extrapolated to allow an evaluation of the stability of moving structures. Robot control systems using this concept to maintain stable walking patterns have become generally known as static walking or static crawling control strategies.

### 2.2.2 Static walking

While the label is a modern invention, the relationship between static balance and centre of mass has been understood for a very long time. Leonardo da Vinci demonstrates an understanding of the concept as he describes how to draw human figures in motion, in his notebooks.

"The centre of gravity of a man who lifts one of his feet from the ground always rests on the centre of the sole of the foot [he stands on]" - Leonardo da Vinci [86]

It is a fairly simple matter to construct robots that automatically walk by keeping their GCoM within the supporting polygon. The objective is to generate a walking joint trajectory that satisfies the constraint of always keeping the GCoM within the robot's supporting polygon. The trajectory is then "played back" on the robot, at a slow enough speed that dynamic contributions to rotational forces are kept small enough that the robot maintains its balance. The joint trajectory can be determined offline by iteratively calculating the position of the robot centre of mass at various points along the walking cycle. Alternatively, instead of calculating the GCoM, the robot itself can be "posed" and its static balance experimentally verified in its various walking positions.

Simple static walkers do not require complex control systems, as all they need to do is play back a pre-recorded joint trajectory. Early experiments with walking robots used kinematic linkages to move limbs in such a way that the structure is statically balanced at all times, while executing stepping motions. Up to a point, errors introduced by the presence of dynamic forces, uneven ground or other unexpected disturbances could be compensated for by increasing the size of the supporting polygon – usually by giving the robot large, or many, feet. This is an approach that is still sometimes used today, even in more "sophisticated" dynamic walking experiments.

The advent of the digital computer allowed scientists to easily modify a robot's gait in response to unexpected disturbances. In 1968, Frank & McGhee conducted experiments with static walking robots that were controlled by microprocessor [60]. The microprocessor gives researchers far more flexibility in developing robot gaits, which have been implemented by a number of researchers on a range of static walking quadrupeds and hexapods. Research is continuing today on the descendants of these early static walking, multi-legged machines (for example, Long, et. al. [83]).

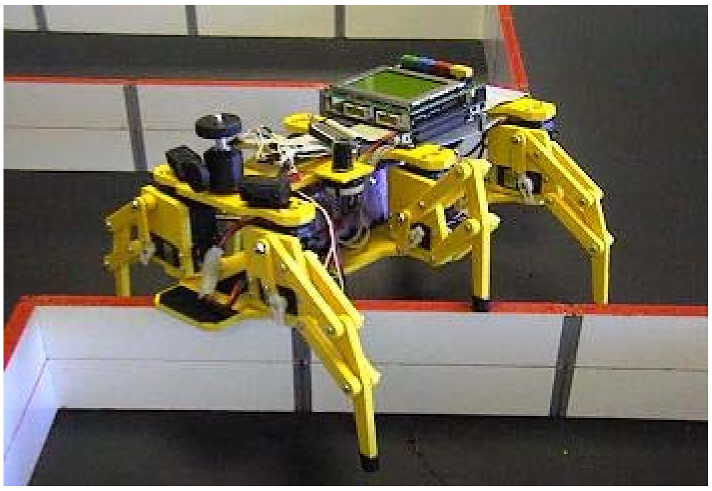


Image source: http://robotics.ee.uwa.edu.au/eyebot/doc/robots/walker.html

**Figure 2-4 – EyeBot Walker cheats by stepping over walls in a maze puzzle**

While static walking robots certainly have their applications, they are severely limited by the need to minimise dynamic forces in the system, and to constrain the centre of mass to within a supporting polygon. High speeds, natural looking movements, gait flexibility and the ability to take advantage of dynamic movements to recover equilibrium if the robot starts to fall, all remain out of reach of static walking machines. In 1980, Kato et al. [61] made some of the first steps away from the static walking model by developing a biped robot control system that executed a gait they described as "quasi-dynamic walking".

### 2.2.3 Quasi-dynamic walking.

Quasi-dynamic is a term given to a class of walking machines which are statically balanced for much of their gait, but deliberately lose their balance during some portion of each step. The term was introduced by Kato et al. [61], when they presented their research into a walking biped robot. Their machine was essentially a static walker that made use of some limited dynamic movements in order to quickly transfer its weight between feet as it walked. The free leg was first positioned to "catch" the robot, and then the robot caused itself to "fall" onto the prepared airborne leg. An inverted pendulum model was used to plan the dynamic weight transfer.

While quasi-dynamic walking is generally considered an improvement on static walking, in that a greater range of movements are possible, the control system driving these robots is not necessarily more sophisticated. Usually the proportion of the gait which is dynamic is short and fast enough that the resulting dynamic response is highly repeatable. Most quasi-dynamic robots still require large feet to ensure the transition from a dynamic to statically balanced state is safely achieved. This means that most robots exhibiting a "quasi-dynamic" gait can still achieve walking by playback of a predetermined joint trajectory. Active control is often not required, and not used. Many popular "walking" toy robots move in this way. A recent example is the "Robosapiens" toy, produced by Hong Kong based company WowWee.

Image source: http://www.wowwee.com

**Figure 2-5 – "Robosapiens" is a quasi-dynamic walking toy.**

Robosapiens moves by rocking its torso from side to side in order to transfer its weight from one super-sized foot to the other. Technically this toy robot fulfils the criteria for "quasi-dynamic" walking, but as far as its walking control system is concerned, it is no more complex than the wind-up clockwork toys.

The walking robot prototype, Johnny Walker (described in Chapter 4), was able to walk using a quasi-dynamic walking gait. Very little in the way of active control was performed by this robot.

Quasi-dynamic robot control systems are essentially a limited extension of static walking control systems. They gain some extra flexibility from the ability to break the rule about always being continuously statically balanced. This increases the control options available to gait designers. Quasi-dynamic robots often rely on a long statically balanced phase to recover from perturbations resulting from the short dynamic part of their gait. Since they are "enhanced" static walkers, usually they require large feet and are slow moving…even if, as in the case of "Robosapiens", the legs are taking many small fast steps.

Both static and quasi-dynamic biped walkers fail to take full advantage of the flexibility afforded to walking machines by their structure. This means many of the advantages that are commonly cited as justification for building walking machines cannot be realised by this category of robot. Currently much of the interest in walking robots is focused on the development of truly dynamic walking robots, which require far more sophisticated and active control systems to operate.

### 2.2.4 Dynamic walking.

A robot classified as a dynamic walker is not required to return to a statically balanced state at any point during its walking motion. These "dynamic walkers" are sometimes called "actively balanced" robots, as during "dynamic" motion the control system must constantly take actions to keep the robot from falling over. In effect, dynamic walking is achieved by shifting the robot into a state of a continuous, controlled fall. Of course, being a dynamic walker does not necessarily preclude a robot from using static walking strategies when it chooses, but this should no longer be a constraint. In order to focus more on the dynamic control of walking robots, many experimenters in dynamic balance have constructed their robots to preclude any possibility of static balance. Robots such as MIT's "Uni-hopper" [55] must continuously maintain dynamic balance, or they will fall – they can never be statically balanced.

Obviously, dynamic walking is considerably more difficult to achieve than "static" or "quasi-dynamic" walking. However, if researchers want to realise the potential of legged robot locomotion, robust dynamic balance is a fundamental requirement. The ability to maintain balance while dynamically walking and running is a trait possessed by almost all walking animals – although it is true that some creatures are better at it than others.

The advantages dynamic walkers possess over static and quasi-dynamic machines stem primarily from increased gait versatility and speed. A robot that is not bound by the constraints of maintaining static balance is free to adapt its gait to environmental conditions. For example, dynamic walkers should be free to pick out isolated footholds when crossing difficult terrain. Since dynamic walkers are constantly in a statically unstable state, it should be easier for these machines to recover easily from unexpected bumps and knocks. Since the robots are not required to minimise their dynamic forces, they can be developed to move far more rapidly than static and quasi-static walkers, increasing the range of movements that can be utilised. And finally, since they are not relying on a large supporting polygon to stay balanced, dynamic walkers are freed from the requirement to possess large, ungainly "clown" feet.

An extension to the idea of dynamic walking is dynamic running. Running is usually defined as a gait where contact is lost with the ground during a portion of the robot's step.

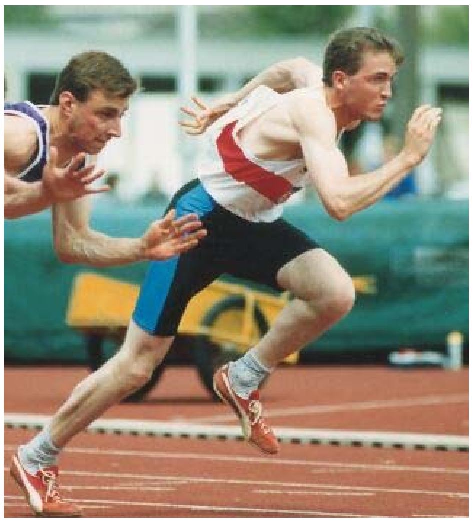


Image source: http://www.olympischeflamme.de/

**Figure 2-6 – Sprinters use a dynamic gait**

By observing the strict interpretation of this definition, it is possible to engineer a "quasi-dynamic" gait that can be called a run. If a quasi-dynamic gait involves "hopping" from one foot to the other, while maintaining static balance between hops it can technically be described as a run. This is not the action that comes to mind when most people imagine a running gait. To truly be capable of running, a robot should be dynamically balanced.

## 2.3 Control systems for dynamic gaits

Traditional approaches to the control of bipedal locomotion have focused primarily on two broad strategies. The first, "classical control", requires the development of precise mathematical models of the dynamic system. The model is then solved using a variety of techniques to generate control solutions for specific situations [30], [31], [32]. The alternative approach relies on developing adaptive control strategies, common examples being neural networks [42], [43], genetic algorithms [47], [69], [70], [74], [82] and reinforcement learning [76], [77], [78]. In this discussion, these techniques shall be referred to as "adaptive control" strategies. Ideally, in these adaptive approaches, the robot can learn suitable behaviour, without the need to accurately model the dynamic system. Recent work on developing control algorithms for biped robots has mixed elements from both the conventional and adaptive schools of control [43].

### 2.3.1 Zero Moment Point

Perhaps the most widely used classical approach to dynamic biped walking are those control systems based on the measurement of zero moment point (ZMP). The ZMP is a tool used to measure the dynamic stability of a walking system. In and of itself, the ZMP is not a complete control system. Rather, the ZMP measurement is used by a control system in the same way that the centre of mass projected onto the ground (GCoM) can be used for static walking systems. It is used in the offline generation of dynamically stable walking gaits, and online to predict if a system needs to take some kind of corrective action to prevent the loss of its dynamic stability.

Dynamic stability, as defined by researchers using ZMP walking strategies, has a specific meaning: In order for a biped walker to be dynamically stable at a particular moment in time, its feet must remain motionless with respect to the ground for the entire time that they support the robot's weight. If a robot begins to "tip over" the edge of its support foot, then it is no longer considered dynamically stable.

The ZMP is defined as the point within the supporting polygon of a dynamically stable structure at which a single ground reaction force acts in order to cause the sum of all moments of active forces on the robotic system to equal zero. The diagram below (Figure 2-7) shows a representation of a typical robot foot.

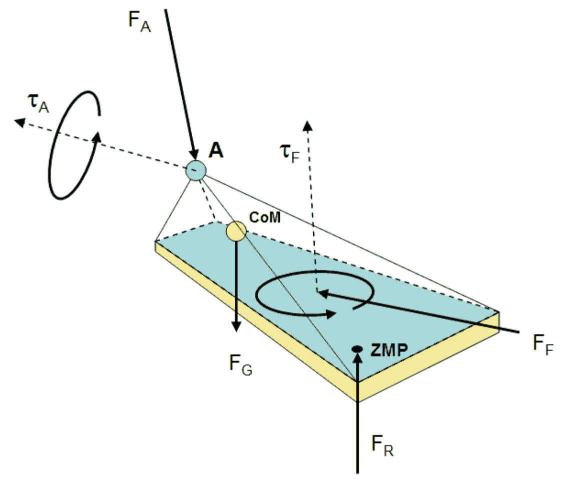


**Figure 2-7 – Forces and torque acting on a robot foot**

Interaction between the foot and the rest of the robot's structure can be represented by a single force $F_A$ and torque $\tau_A$ acting on the foot's ankle joint. Additional forces working on the system include:

- $F_G$, the gravitational force acting on the foot, through the foot's centre of mass
- $F_F$ and $\tau_F$ the frictional force and torque preventing the foot from sliding or twisting along the ground, and
- $F_R$, the ground reaction force preventing the foot from penetrating the ground surface.

The ZMP definition of a dynamically stable robot requires the foot does not roll or tilt with respect to the ground during gait execution. This means that to be stable, the resultant of all the pitch and roll torques acting on the foot must be zero. It is left to the ground reaction force $F_R$ to cancel out any horizontal components of torque resulting from $\tau_A$ and horizontal torque contributions of all other forces. Since the ground reaction force is fixed in magnitude, the only way it can compensate for varying horizontal torque is to be applied at different points on the robot's supporting polygon.

Note that it is possible for a robot's foot to slide while still satisfying the ZMP stability criteria.

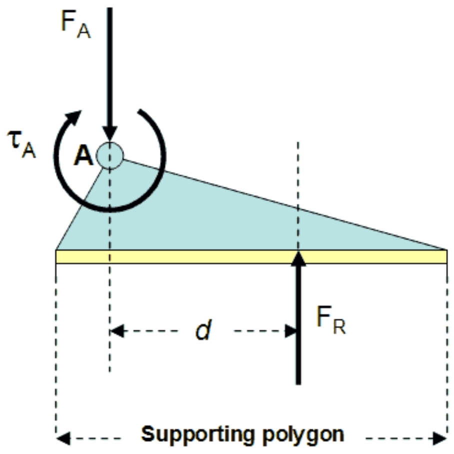


**Figure 2-8 – Simplified foot dynamics**

In the example depicted by Figure 2-8 above, as the distance *d* increases, the reactive torque produced by $F_R$ also increases, such that:

$$\tau_R = d\ F_R$$

If a point can be found within the supporting polygon such that the reactive torque $\tau_R$ applied at this point exactly cancels out all other torques acting on the foot, then this point is known as the zero moment point, and the robot is dynamically stable. In the example above, *w*hen *d* coincides with the ZMP, the reactive force $F_R$ will cancel out all other torques, so if all other forces are known, the location of the ZMP can be calculated:

$$\tau_A + \tau_R = 0$$

$$\tau_A + d_{ZMP}\ F_R = 0$$

$$d_{ZMP} = - \tau_A / F_R$$

In reality, the ground reaction force does not act through a single point. Instead it is distributed across the supporting polygon as a pressure applied to the sole of the robot's foot. In dynamic calculations, the pressure between the foot and the ground can be replaced by single force acting through the centre of pressure (CoP). If the robot is dynamically stable, then the effect of this reactive pressure is equivalent to the result of a ground reaction force acting through the ZMP. In other words, the ZMP and CoP coincide. This is good news for researchers intending to use the ZMP to help control a walking gait, because the CoP can be easily measured using pressure sensors in the robot's feet.

As torque transmitted to the foot by the rest of the robot system increases, the ZMP must move closer to the edge of the foot in order to generate enough reactive torque to maintain dynamic stability. Once the ZMP reaches the foot's edge, the system is only marginally stable and any further increase in applied torque will cause the system to tip over. So in dynamically balanced systems, the ZMP has the same relationship to the robot's supporting polygon as the GCoM does in statically balanced systems. As the ZMP moves towards the border of the robot's supporting polygon, the robot's dynamic stability becomes more marginal.

By definition, the ZMP can never exist outside of the robots supporting polygon. If the robot is not dynamically stable, then there is no zero moment point. Sometimes it is useful to be able to compare the relative stability of various unstable states. To this end, researchers have extended the ZMP concept to include an imaginary point outside of the robot's supporting polygon, where the ground reaction force would have to act if it were to maintain dynamic stability. This imaginary force is calculated in the same way as the

ZMP, with the constraint to remain within the supporting polygon removed. In order to highlight the difference between this imaginary point, and a real ZMP, the terms foot rotation index (FRI) or fictional zero moment point (FZMP) have been suggested [51]. Since it is impossible to apply a force to an object without contacting it, while it is outside the robot's supporting polygon, the FRI does not exist and so cannot be measured. While the FRI lies within the robot's supporting polygon, it is equivalent to ZMP and the robot is dynamically stable. Since the FRI is not always measurable, and mainly applies to systems that have already lost dynamic stability, it is not very useful for on-line ZMP based control systems. FRI/FZMP calculations are more usefully applied to off-line gait synthesis tasks.

There are two applications for which the ZMP/FRI measurements are typically used: gait synthesis and control feedback. Gait synthesis is usually determined off-line using a variety of algorithms [8], [31], [53], [87], [88]. These all attempt to constrain the ZMP to a region within the supporting polygon, giving a stability margin. If environmental conditions change, for example the robot needs to climb a slope, or negotiate a staircase, the gait calculations must be repeated to maintain suitable robot behaviour. For use in control feedback, the CoP can be directly measured using sensors in the robot's feet. While the robot is dynamically balanced, we know that the measured CoP is equivalent to the ZMP. The robot control system monitors this inferred ZMP, and takes some kind of corrective action to shift the CoP away from the borders of the supporting polygon if its stability margin is threatened. These corrective actions are usually pre-planned, often involving movements of the robot torso or arms to try and restore the ZMP stability margin.

In their humanoid robots "Asimo" and "Qrio", Honda and Sony have both developed sophisticated biped robots that use variations of ZMP based control. Both of these robots are impressive demonstrations of dynamic biped walking, however I believe both are limited by the reliance of their control systems on pre-planned joint trajectories. In each case, the control strategies involve creating a library of pre-recorded joint space trajectories, each of which executes a specific action, such as walking, turning left or right, starting and stopping. Joint trajectories resulting in more complicated actions such as climbing stairs are also stored in this library. These movements are designed to maintain a large “stability margin”, as measured by the distance between the ZMP and the edge of the supporting polygon. Specific actions can then be “played back” on the humanoid robot, as directed by a remote operator. Further robustness is gained by monitoring the CoP of the robot’s contact with the ground, and modifying the pre-recorded gait to compensate for unexpected deviations in the ZMP. Honda’s Asimo unit includes an additional level of sophistication that they call predictive control [75]. In the Honda system, planned future movements are anticipated and pre-emptive corrective action taken to ensure a large stability margin is maintained during transition between walking actions. For example, if the robot is about to turn to the left while it is currently walking straight, before it turns it can begin to shift its ZMP away from the right hand edge of the supporting polygon.

Despite the achievements of these systems, both share a number of disadvantages with other control systems based on zero moment point prediction and monitoring. Some of the problems with using a ZMP approach to control system design for walking robots include:

- Robot gait must be predominantly determined off-line, which restricts adaptability to terrain. ZMP based control systems do not allow a robot to deal with situations for which it has not been explicitly pre-programmed.

- Corrective movements are also pre-determined, often involving movements of the robot torso or arms to try and restore ZMP safety margins.
- ZMP systems are still bound by restrictive stability constraints. This means that a ZMP based control system inherits many of the disadvantages that were supposed to have been removed in the move from static to dynamic walking system.
- ZMP Calculations give a measure of dynamic stability - they do not provide active balance solutions. Alternative methods must be used to actually determine the control actions required to keep a robot balancing
- The entire premise of ZMP analysis insists that the robot must at all times maintain contact between the entire sole of the robot foot and the ground. The obvious result is a flat-footed, unnatural walking gait.

Zero moment point based control systems are only a small step forward from static walking machines. In ZMP systems, the concept of "static stability" is replaced by "dynamic stability", an artificial constraint promoted as a required characteristic of dynamically walking machines. Robot movements are constrained to a subset of actions that keep a calculated mapping of robot state within the pre-defined boundaries. While these boundaries are less restrictive than those of static walkers, they are still restrictive.

A system that is dynamically stable is not guaranteed to be safe from falling over. All that can be said for certain is that while the ZMP is within the supporting polygon, the robot's foot will remain flat on the ground. For example, in the system described in Figure 2-9, dynamic stability can be maintained by limiting the magnitude of torque that can be transmitted to the robot foot via the ankle joint.

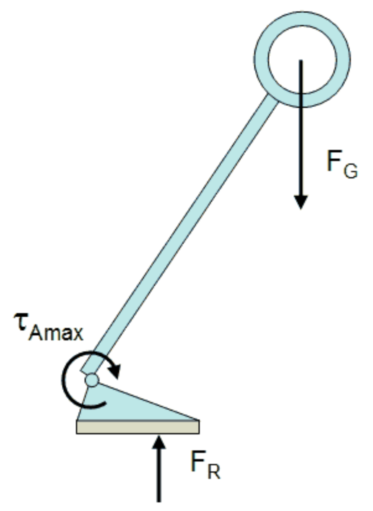


**Figure 2-9 – The system guarantees maintenance of the ZMP definition of dynamic balance by limiting $\tau_{Amax}$, but it still collapses. It is not balanced at all.**

Since the ZMP remains in the supporting polygon, the system is dynamically stable, and its foot remains in contact with the ground. But it still falls over, as a small ankle torque is insufficient to prevent the robot structure above the ankle from collapsing to the ground.

Human beings, the blue-print for humanoid walking systems, do not walk like this at all. Dynamic stability, if it exists at all during a person's walking gait, is an incidental state – not a requirement. If information about the CoP or ZMP is obscured, a person will generally have no problems walking. This can easily be tested by observing stilt walkers, infants tip-toeing or people walking in ski boots. For most of the walking stride only a portion of the foot is in contact with the ground at any time, and during a stride the supporting foot rolls from heel to toe. In fact, it is quite difficult and unnatural to try and walk while deliberately keeping your feet flat on the ground. In short, people do not maintain "dynamic stability" when they walk, and a robot shouldn't need to either. It is generally not considered a compliment to call someone flat footed.

McGeer’s passive walkers [58] use the foot as a lever to help ensure the raised/returning leg does not collide with the ground. Such a movement relies on the foot being to rotate as the robot transitions through its gait. The passive walker’s feet are designed to have a semi-circular profile, as shown by Figure 2-10. A ZMP balanced robot cannot move like this – it must be flat-footed.

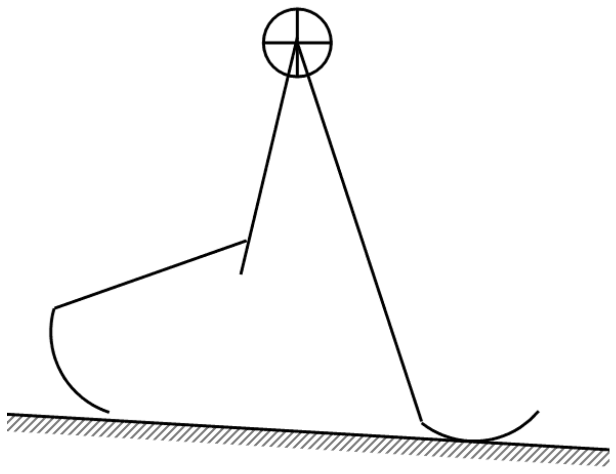


**Figure 2-10 A simplified system model diagram of McGeer's passive walking robot. This kind of walker cannot be controlled using ZMP approaches, as its feet must be free to rotate as the robot swings over the supporting foot.**

There is a class of walking control systems that does not rely on maintaining some artificial measure of “stability” in order to achieve walking. These systems are based on one of the earliest experiments for control of unstable/non-linear systems, the inverted pendulum.

### 2.3.2 Inverted Pendulum Inspired Systems

Inverted pendulums are the classic “balancing broomstick” problem, where a pole or rod (the “pendulum”), is balanced by the application of a horizontal force to a cart on which the pendulum stands.

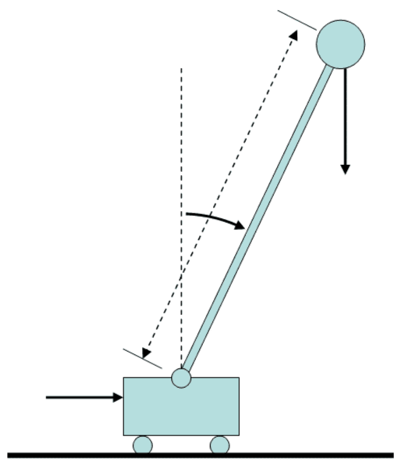
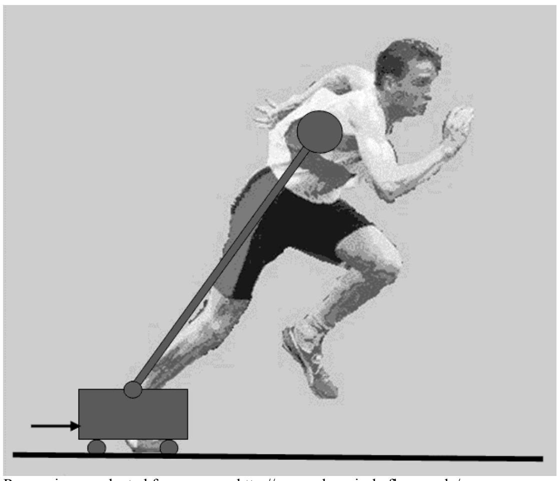


**Figure 2-11 – The classic inverted pendulum**

Experiments balancing inverted pendulums are not new, with the first known automated example built in 1951 by Claude Shannon. Despite the simplicity of the system, inverted pendulums are an example of a non-linear control problem, which is never completely stable. A controller must continuously act to correct any deviation in the pendulum angle from vertical or it will fall. The classical solution to balancing an inverted pendulum is well known, and is fairly easy to implement. As a result, inverted pendulums have often been selected as a test platform for evaluating the performance of new algorithms intended for controlling non-linear systems [4], [6], [11], [13], [18], [19], [22], [89], [90], [91].

Inverted pendulums are of particular interest in the study of biped walking because the dynamics of a human body walking share some obvious characteristics with that of a simple inverted pendulum system [57], [59]. In a simplification of the forces acting on a balancing human, a body's dynamics during the single support phase of a walking or running gait can be modelled as an inverted pendulum. The supporting foot's contact with the ground is the fulcrum, about which the body rotates. Instead of an external balancing force being applied to a pendulum cart, humans (and autonomous robots) can exert horizontal forces against the ground, in order to generate frictional reaction forces to help maintain balance.



Runner image adapted from source: http://www.olympischeflamme.de/

**Figure 2-12 – A simple inverted pendulum model for a running man**

Not surprisingly, inverted pendulum models have been used by a number of researchers to simplify the dynamics of humanoid walking during the development of robotic control systems [7], [21], [61].

Unlike the ZMP based control systems, which have a precise definition, and fairly consistent treatment by researchers, there are a great many ways inverted pendulum models can be incorporated into control system designs. As a result, researchers have used these models to varying extents and for varying purposes. Some of the ways various researchers have made use of the inverted pendulum model include:

- Used to predict or plan "optimal" foot placement as the robot is preparing to change supporting feet. [21], [61]
- Can be used to determine an appropriate joint torque required to assist or maintain balance. [21], [7]

Two specific examples of walking algorithms relying on an inverted pendulum model include the "linear inverted pendulum mode" and "gravity-compensated inverted pendulum mode" control systems.

#### 2.3.2.1 Linear Inverted Pendulum Mode (LIPM) [7]

This control system simplifies the dynamics of a biped system, using an inverted pendulum model, to the extent that the on-board controller is able to operate quickly enough to perform control decisions on-line. The control system first selects a series of footholds, based on prior knowledge or active sensing of the terrain profile. Next, a trajectory for the robot's Centre of Mass (CoM) is calculated, such that the CoM traverses at a constant height above the pre-selected footholds. This constraint is executed by the hip and knee joints, and is not directly controlled by the dynamic model. The robot legs are assumed to have negligible mass, and to be exchange instantaneously as the robot takes each step. The system is modelled by treating the single support phase of a step as a linearised inverted pendulum. Active control is achieved by applying torque as directed by the model to the supporting leg's ankle joint.

#### 2.3.2.2 Gravity-Compensated Inverted Pendulum Mode (GCIPM) [9], [53]

The GCIPM strategy is similar to the LIPM, on which it is based. The primary extension introduced consists of an analysis of ZMP and active control of the free leg while making a step. The free leg is no longer assumed to have no mass, although its mass is modelled as a point mass concentrated at the location of the free foot. The mass of the supporting leg is still assumed to be zero. The GCIPM controller generated walking trajectories were tested using a simulation and demonstrated improvements in stability over LIPM control.

### 2.3.3 Ballistic Control

A constraint common to all the previously discussed control system models is that the robot is required at all times to keep at least one foot on the ground. This means the robot is limited to walking, and cannot run or jump unless a different control strategy is selected.

Two of the most well known examples of recently created running robots, Sony's "Qrio" and Honda's "Asimo", overcome the difficulties introduced by the airborne component by minimising the time spent without ground contact. The image shown in Figure 2-13 is taken from a video, published by Honda, of "Asimo" running. In the image, the leading foot is about to contact the ground, while the trailing foot is yet to leave it.

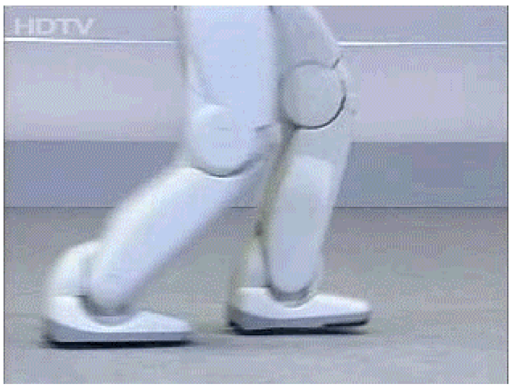


Image source: http://www.world.honda.com/HDTV/ASIMO/200412-run/index.htm
**Figure 2-13 –Asimo prepares for its short ballistic phase**

Using this approach, the researchers are able to claim their robots are running, without the need to substantially alter the ZMP based control systems used to make their robots walk. On the other hand, if a robot is truly running, the fully ballistic phase of its motion must be a significant component of the overall gait. In the example above, Asimo's "run" would not disqualify it from an Olympic walking race!

Control of a robot during the ballistic phase of its gait has received special attention from a number of researchers, perhaps most famously by Marc Raibert in his book, "Legged Robots that Balance" [55], one of the most frequently cited works in the field of bipedal locomotion. Raibert suggests that control of a running robot can be reduced to a series of jumps, or hops, where each leg can be modelled independently as a spring mechanism. To test this idea, he built a series of robots with pneumatic, telescoping legs that were designed to hop around on one, two or four legs.

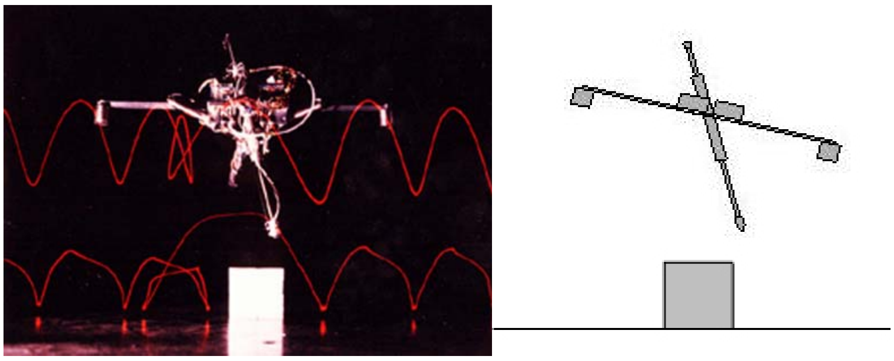


Image source: http://people.csail.mit.edu/russt/lrcc/robot.cgi?id=1

**Figure 2-14 –Raibert's one-legged hopping robot.**

Since Raibert saw control of a running robot as an extension of that of hopping robots, he first designed a controller for a single legged, hopping robot. The resulting controller was then successfully applied to robots with more than one leg, with each leg driven as one of a number of coordinated hopping legs. For the case of a single hopping leg, a running step consists of two major phases – the support phase, and the ballistic phase. During the support phase, while the foot is on the ground, a body posture correction torque is applied by the hip joint keeping the robot upright. During the ballistic phase, while the entire robot is airborne, the leg is moved into position for landing. The landing points are carefully selected by the robot in order to control the horizontal acceleration of the robot. Depending on the current horizontal velocity of the robot, landing points further ahead of the robot will slow the robot down, while landing points behind will speed it up. Maps of landing points and their corresponding effect on acceleration were experimentally determined for various velocities.

Raibert's hopping machines, and similar robots built by other researchers, are truly dynamic machines. They rely entirely on their constant motion to stay stable upright, and so are never able to stop and stand still. For real-world applications, depending on the circumstances a robot should be able to combine the dynamic capabilities of ballistic machines, with the careful stability which is the strength of ZMP based walkers. Tools such as the ZMP and LIPM can be used to help control the robots while they are in the support phase of a running gait, and the integration of these control systems should allow a robot to switch between walking and running gaits without the need to switch between dramatically different control systems.

### 2.3.4 Virtual Model Control

"Virtual Model Control" (VMC) is a methodology introduced by Pratt for developing robotic controllers [66], [80], [85]. Virtual Model Control employs an array of virtual springs, dampers and other imaginary components to disassociate the low level control of the robot from high level commands. These virtual components are attached at various points to a model of the robot being controlled, and are used to "pull" the robot in the desired directions. The components provide a high level representation of the commands being sent to the VM control system.

Once the virtual components have been attached to a robot model, the imaginary forces with which they act on the robot are determined. For example, the virtual spring shown in Figure 2-15 attached to a robot arm's end effector exerts an imaginary force on the end effector, pulling towards the "spring set point" position.

The resultant imaginary forces acting on the robot are then examined, and a set of joint torques that generates the same resultant forces is determined. Selecting the actual torque values to provide the virtual model behaviour usually requires complex inverse kinematic equations, as forces are converted from linear Cartesian forces to the equivalent joint torques.

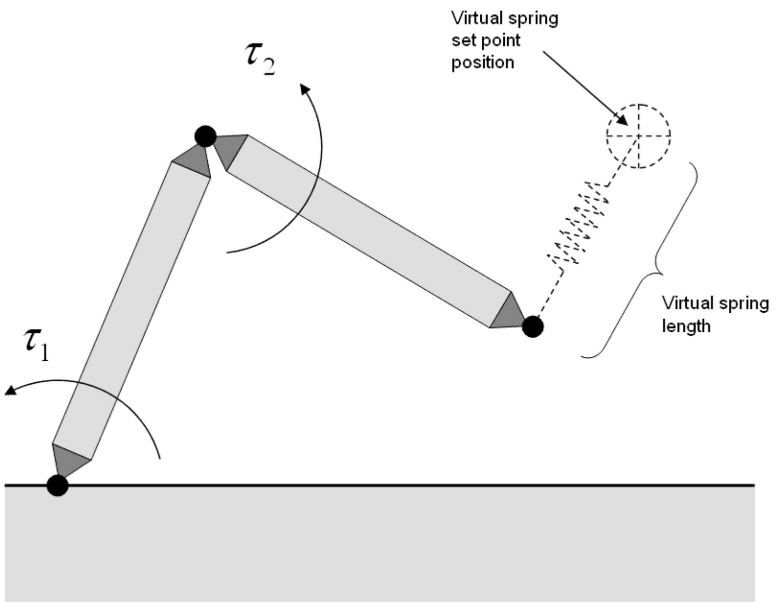


**Figure 2-15 Virtual model control of a 2-link planar robot arm could be achieved through the use of a single virtual "spring" component attaching the end effector to an imaginary "virtual spring set point position". Joint torques that will that generate the desired "virtual" force need to be determined analytically. Usually a virtual damper is used in association with any spring component to prevent oscillations in the resulting robot movement.**

Depending on the composition of the virtual controllers, the resulting robot actions can be compliant, natural looking, efficient and safe. Pratt states that the aim of VMC control is not to reproduce a specific joint trajectory, but rather to "encourage" the robot to follow the VM supplied instructions [80]. A significant advantage of using a VMC approach is that design of robotic controllers can be described in an intuitive manner. The robots appear to be drawn about like a puppet on a string.

The VM concept may well have been inspired by the idea of "series elastic" actuators, which was developed in the same laboratory [81]. Series elastic actuators introduce an organic looking elasticity into force-based joint actuation by using tendon-like structures to transmit torque to a robot's actuators. An advantage of these actuation systems, used by Pratt in his physical robots, is that the bulk of actuator's weight can be concentrated in a robot's body, reducing the weight of legs. VMC control lends itself to implementations using series-elastic actuation, as these actuators already behave with a "spring/damper" component to their response. Alternatively, VMC control can be used to mimic series-elastic behaviour using actuators capable of applying specific, variable torque, and can be used to drive systems which do not exhibit the natural elasticity of series-elastic actuators.

Despite the attractions of VMC approach to control system design, there are some important classes of problem for which the VMC methodology is not suitable. The most important of these from a biped walking control point of view is the classic inverted pendulum problem, since inverted pendulums are often used as the basis for system models of walking bipeds. Figure 2-16 illustrates two views of the same inverted pendulum system, together with the VMC components that would be used to try and control it.

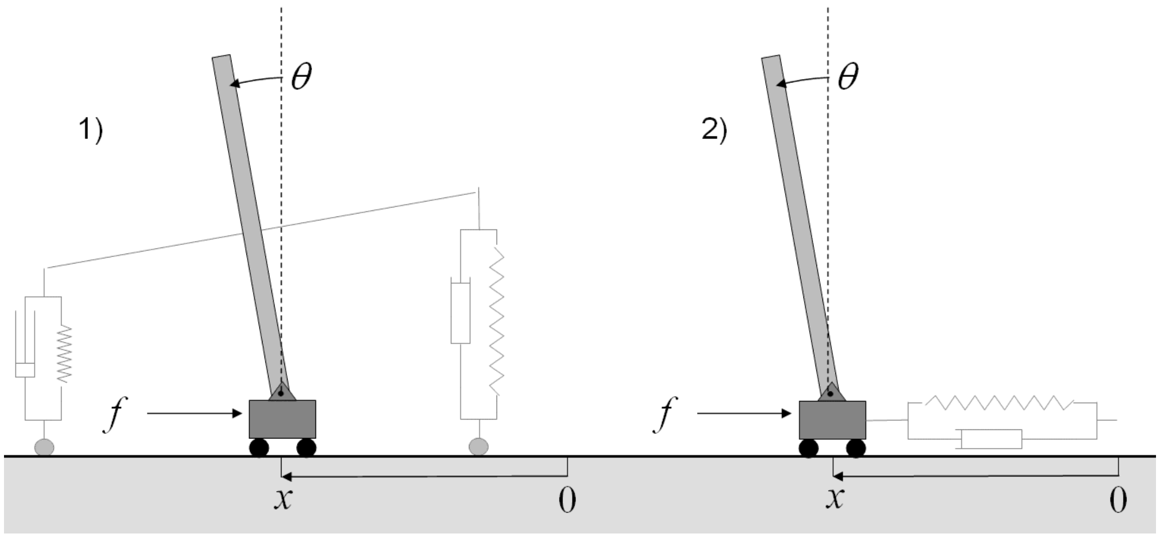


**Figure 2-16 Two virtual model components are attached to an inverted pendulum. (1) A virtual model "granny walker" is used to apply corrective torque to hold the pendulum vertical. (2) a spring/damper virtual component pulls the pendulum base to the centre of the workspace. Since the pendulum can only be controlled by a single horizontal force, it is not possible to satisfy the requirements of both components at the same time.**

In the diagram, the following VMC components are used:

1) Spring/damper virtual components are used to maintain the angle of the pendulum by generating a corrective torque if the pendulum leans to either side.

2) A second spring/damper component is used to control the displacement of the pendulum base, pulling it towards a target position (in this case the point $x = 0$).

Consider the situation shown in the diagram, where pendulum angle $\theta > \frac{\pi}{2}$ while the pendulum displacement $x < 0$. Component (1) would require a negative torque be applied to the pendulum, while component (2) would demand a positive force be applied to the pendulum base. The problem is that an inverted pendulum is controlled using a single input signal – the horizontal force to be applied to the pendulum's base. To satisfy the first component's requirement, a negative force would have to be applied, while the second component requires a positive force. It is not possible to satisfy both components at the same time. The classical solution to the inverted pendulum in this example is a simple relationship [26]:

$$u = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x}$$

But a design based on VMC control does not allow us to find this solution.

Despite these limitations, Pratt [66] has successfully applied VMC to the control of a planar biped 5-link walking robot which has a similar system model to the Legbot robot used in the experiments in Chapter 6. The robot walks, and maintains balance, by selecting joint torques that mimic the effects of a "virtual granny walker" to hold the torso vertical and off the ground, together with a "dog track bunny" to drag the robot after a moving target.

The "granny walker" virtual component maintaining torso inclination and altitude is active at all times (both single and double support phases) during the robot's walk. Positional control cannot be achieved via VMC models during the single support phase of a walk, for the same reasons VMC cannot be used to solve the inverted pendulum problem (see above). As a result, positional control is managed by the knee joints during the double support phase of walking gait only.

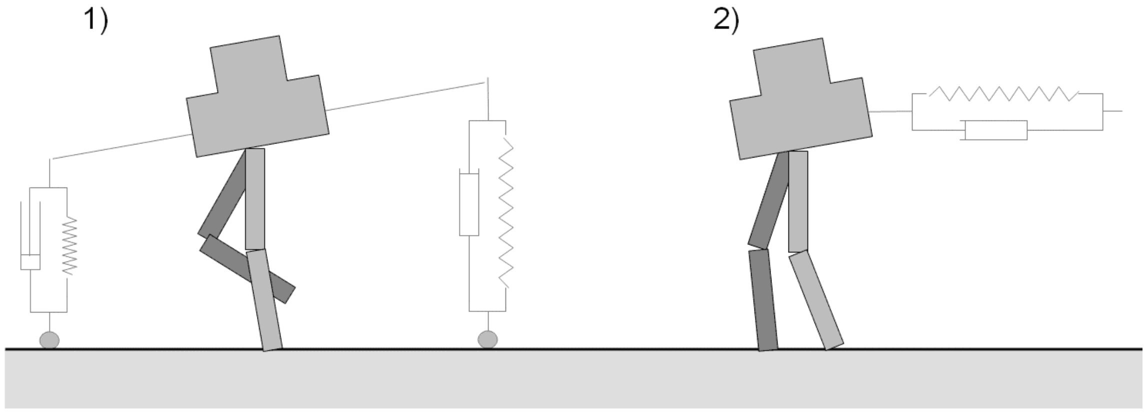


**Figure 2-17 Pratt's virtual model controlled biped robot. (1) A virtual "granny walker" holds the robot off the ground, and keeps the torso angle upright. (2) When both of the robot's feet are on the ground, a second VM component regulates the robot's horizontal velocity.**

Pratt's VMC implementation of a biped walker is only one possible approach using this design philosophy. Virtual Model Control could be combined with other methods (such as ZMP or reinforcement learning trajectory approaches to walking control). In this thesis, VMC methodologies are utilised in one component of the Torso Driven Walking control system (the "Collision Avoidance Controller", Chapter 3.2.7). A number of other components of the TDW control system share some characteristics with VMC. For example, raised leg joint trajectories are implemented using PD control. This configuration would be described in a VMC system using a virtual "spring/damper" component to connect each joint to a set point angle. On the other hand, any PD based controller fits this definition. To really be considered a Virtual Model system, all trajectory planning should be performed using real world coordinates (not joint space), then the required torques calculated in joint space.

Chapter 6.8 describes an experimental comparison between Pratt's Virtual Model Control system for Biped Walking, and the Torso Driven Walking control system.

### 2.3.5 Adaptive trajectory based controllers

Early implementations of trajectory based walking algorithms, required control system designers to plan the robot's movement off-line, and then experimentally modify the trajectory on the target platform, manually altering the "program" to try and compensate for observed failures. Tools such as static and ZMP stability criteria enabled designers to predict the stability of potential joint trajectory "programs", but the work of crafting a trajectory is still difficult. Recently a number of researchers have applied adaptive techniques to the problem of crafting joint trajectories for controlling walking robots. [74], [70], [76], [77], [78].

#### 2.3.5.1 Finding a stable "fixed" trajectory

Many adaptive systems such as Boeing [74], Zhang [70] learn to follow or find trajectories that do not cause the robot to fall over when replayed. Both of these researchers employed genetic algorithms to generate cubic polynomials which produced the periodic gait trajectories. Boeing's systems employed a fitness function evaluating a simulated robot's performance in playing a "walk forward" trajectory. Zhang's GA system evaluated a trajectory's fitness by examining its ZMP based stability criteria. Using the ZMP stability criteria for fitness evaluation will mean the resulting trajectories will be a sub-set of the solutions that may have been found using an experimental evaluation, and the results may not include the most dynamic or fastest solutions. On the other hand, they are likely to be safe.

#### 2.3.5.2 Modulating an arbitrary trajectory

One of the disadvantages of trajectory based control systems is that if the environmental conditions experienced by the robot change, then a new trajectory will need to be developed. For example, the robot needs to learn a new trajectory if it wants to walk down hill, as its speed will tend to increase with steeper gradients.

Recent work by Morimoto, et. al. [76], [77], introduces a technique they have named "Poincaré-map-based reinforcement learning". In this context, a "Poincaré map" is simply an alternative name for a "joint trajectory" without reference to time.

The Poincaré-map-based algorithm is used to show that by simply modulating the trajectory's step cycle time and step length, a trajectory can be modified "on-the-fly" to cope with unexpected changes in the robot's state. For example, if a robot begins to walk down hill, by lengthening the stride and quickening the pace, the same control system that successfully walked on a flat surface could be used to walk downhill. Furthermore, they have implied that, assuming the required step length and cycle time can be found, any reasonable periodic walking trajectory can be used to generate a balanced walking gait. The researchers suggest motion capture could be used to craft the joint trajectory, rather than the hand-designed pattern they have used.

Using a number of reinforcement-learning algorithms, the Poincaré algorithm learns to predict the desirability of the system state resulting from various foot placement and step period decisions, given a current system state. These models are used to decide on the foot placement and step period to use for subsequent steps. The control system makes these decisions only once each step cycle, during the "single support" phase of the robot's walk, when the robot's raised foot is above the support foot. Because most of the parameters describing the joint trajectory are fixed, with only two variables to generate, and because control decisions are made only once each step, the learning process can be very fast. The researchers report that stable controllers can be achieved within 100-200 trials. This low number of required trials means it is entirely feasible to perform all of the training required for reinforcement learning on a physical robot.

However, while the resulting control system is more robust than fixed trajectory systems, the behaviour exhibited by the control system is entirely reactive. This is not actually velocity control, but simply the controller adapting the robot's gait to overcome external perturbations (such as a slope, changing surface properties, externally applied forces, etc.). In order to deliberately alter the robot's velocity, another approach is required. The robot is still unable to stop or move backwards without changing its "program" to another trajectory.

The torso is kept balanced by a torque incidentally applied via the hip joint as the PD control implementing the trajectory playback goes through its paces. The control system makes no attempt to explicitly control the orientation of the robot's torso. A disadvantage of this approach is that if the robot falls, it will be hard! On the other hand, this approach eliminates the need for sensing torso inclination, which is one of the more difficult state measurements to accurately determine.

#### 2.3.5.3 Approximating trajectories

Sharon and van de Panne present an imitation-based reinforcement learning algorithm for finding a planar bipedal walking trajectory [78]. The algorithm attempts to develop a trajectory that follows a walking "style" introduced to the algorithm as an example or demonstration walking gait. Note that this gait does not have to be ideal – or even possible. The example trajectory is intended to be used as a suggestion, rather than a requirement. The demonstration trajectory is generated using key-frames of the robot joint positions at various times during a step, and interpolating to find the complete trajectory.

Unlike most other trajectory based controllers, the control system resulting from Sharon's algorithm is not simply a description of the joint trajectory (or Poincaré map) the robot should perform. Instead, the algorithm creates a function mapping the current system state to a set of controller output values (usually joint torques). For each robot state, the algorithm learns the control actions that will be most likely result in the robot following the example trajectory.

The function mapping 'input space' to output torque is encoded as a set of nodes, each of which is associated with a "point" in the robot's "input-space" and a vector of control output values. The "input-space" is the hyper-dimensional volume which encompasses the values obtainable by the robot's sensors. This configuration effectively quantises the controller's input space. At any instant, the robot's current state is identified using a weighed minimum distance function measuring the distance to the closest node, and that node's output vector is used to drive the robot's joint torques. Figure 2-18 displays an abstraction of the controller representation and operation, and has been taken from [78].

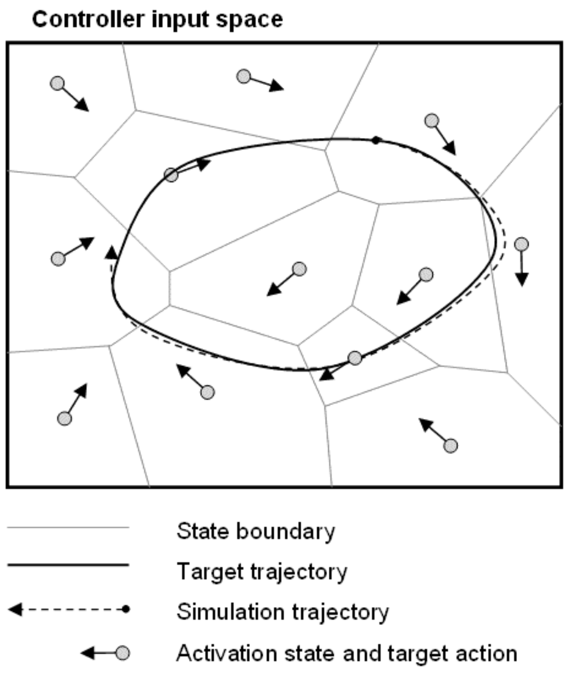


Image source: Sharon, D, van de Panne, M., "Synthesis of controllers for stylized planar bipedal walking", Reference [78].
**Figure 2-18 Abstracted representation of Sharon's imitation-based controller operation.**

Before the Reinforcement-Learning algorithm is applied, the input/output mapping function is initialised by distributing the set of controlling nodes evenly along the target trajectory. The output vector initially associated with each node is calculated to drive the robot towards the centre pose of the next node. Optimization involved determining quantization boundaries, as well as associated control signals.

The fitness function used in this example of an imitation-based reinforcement learning algorithm measures how closely the robot follows the specified target trajectory over time. Balance is implied by a high score, as the robot has to stay balanced to accumulate the score. The researchers demonstrate their control system successfully learns to balance a range of planar biped robots with point feet. In addition, by applying different example trajectories to the learning algorithm, they are able to generate walking controllers with significantly different styles and stride durations. As with the Poincaré-map controllers, this result demonstrates again that the range of possible trajectories a robot can use to execute its walk is surprisingly variable.

Because the controller is not learning a trajectory explicitly, but rather learns torques and forces that cause the robot to converge towards the trajectory, the system should be able to robustly handle unexpected disturbances. Experimental results published show the imitation controller successfully navigating unexpected terrain, including a "bumpy" surface and a slope [78]. Despite this robust behaviour, the control system is not controllable. The robot will always strive to follow the single "example" trajectory. The result is a single, constant velocity walking gait for each controller. As with most other trajectory based systems, if the robot is to do anything else, new trajectories must be learned. In addition, as separate control signals need to be learned for every possible input state, training time is expensive. The robot required around 23,000 trials to learn each new trajectory.

A significant problem common to all trajectory based walking control systems is variability. The control system designer needs to develop a new trajectory for every new speed, direction and action. As an added complication, it is not possible to arbitrarily shift from playing one trajectory to another, as the robot will need to enter a trajectory from approximately the same state each time. This means that if the robot is currently playing a "walking forward" trajectory, and receives a command to turn left, the robot must find a way to transition between the forward walking and turning left programs.

### 2.3.6 "First principles" adaptive control systems

One of the great attractions in using adaptive control systems is the potential to develop a control system without needing to fully understand the system you are controlling. Learning from "first principles" implies the control system will be able to learn from scratch everything to do with controlling the system.

In practice, what this means is that researchers do not give any hints or implied knowledge to these systems, and a very simple feedback mechanism is used to shape the adaptation. For example in walking systems, the training system's only feedback might be a failure signal if the robot falls over. Everything else needs to be learnt by the control system through some form of trial and error. In a robot balancing application, such as biped walking, the system even needs to learn that keeping the robot upright is safer than leaning far forwards or backwards.

#### 2.3.6.1 Artificial neural networks

Artificial neural networks have often been suggested as a potential tool for implementing "first principles" walking control systems for biped robots. Early attempts involved feeding the robot state into a single "monolithic" network, and training the robot to behave in a sensible way. In practice this proves to be an impractical approach, due to the large size of the required network, the correspondingly large training times, together with the necessity for a "nearly sensible" output at all times.

Since inverted pendulums are often used to describe biped walking system models, it makes sense to briefly consider "first principle" adaptive control systems that have been applied to solving the inverted pendulum problem.

Anderson [13] presented a method of training a neural network to balance an inverted pendulum using only a single failure signal if the pendulum falls or moves too far from the origin. Anderson's system consisted of two components:

- A predictive controller, which learns to predict the future state of a robotic system, based on a measurement of the current state, together with a potential control signal, and:

- An evaluator component which learns to judge comparative qualities of different robot states.

The evaluation function judges the likelihood that a given state will result in a failure. Control signals generated by the action network are limited to the set of three possible values: $\{-f, 0.0, +f\}$, where $f$ is the maximum horizontal force that can be applied to the pendulum base.

Anderson contrasts his technique favourably with a system presented by Michie and Chambers [23], called "BOXES". In this system, the entire state space that can be experienced by the system is quantised into regions, where an appropriate control action is learned for each region. For an inverted pendulum, the state space was divided into 162 regions. This approach can quickly become unwieldy for systems with a large number of state variables.

Hougen, Fischer & Johnam [18], have presented a system, based on a technique they call SONNET (Self Organising Neural Networks with Eligibility Traces), which they have used to control a real cart and pole. This system also limits its success/failure feedback to a single failure signal when the cart crashes. They limit the state information sampled to cart position and pole angle.

Chapter 5.8 discusses the implementation of a simple "evaluator/predictor" back-propagation neural network (BPN) based controller that can be trained to balance a simulated inverted pendulum system. The control system shares many characteristics with Anderson's approach, with the major difference being that my controller can apply a variable control force to balance the pendulum. The controller operates with no a priori knowledge of the system model of the inverted pendulum. In addition, the control system initially knows nothing about relative quality of various pendulum states.

All this knowledge is found by trial and error during training of the networks, while limiting all performance feedback to an error signal when the pendulum has crashed.

Despite these and other successes in controlling inverted pendulums with neural networks, there are significant problems which rule out attempting to control a biped robot with neural networks working from “first principles”.

For the relatively simple inverted pendulum problem, all of these ANN based controllers successfully learn to balance the pendulum. However even for this simple problem, ANN controllers require an immense amount of training data, as well as very careful selection and presentation of training data. Given the performance issues experienced with ANN control of inverted pendulum systems, simply extending these kinds of controllers to much more complex biped robots would prove to be prohibitively expensive.

#### 2.3.6.2 Genetic algorithms learning with no assistance

Recently, Wolff and Nordin have presented some ideas in applying genetic algorithm techniques to “first principles” biped walking [67], [69]. Unlike some of the ANN systems previously discussed, the researchers define a fitness function used to measure the quality of the resulting gaits. This feedback is richer than the single failure signal input to Anderson’s ANN system, and “evaluator/predictor” ANN system described in Chapter 5.8. A control system accumulates fitness score by increasing the time it keeps the robot’s balance, as well as by covering greater distances during the test.

The control program evolved by Wolff and Nordin’s algorithm encodes its behaviour as a sequence of simple mathematical operations. Each instruction consists of four elements, encoded as integers:

- Input register 1 (may be an input signal or an internal register).
- Input register 2 (may be an input signal or an internal register).
- An operation to perform on registers 1 and 2 (i.e. ADD, SUB, MUL, DIV, SINE)
- An output register, which may be an internal register, or an output signal.

The researchers report that at the conclusion of GA evolution, the best performing individuals outperformed a manually developed controller. The control systems successfully balanced the robots for 20 seconds (the trial duration). Information about the distance covered is unavailable.

However, as with the ANN approaches, this GA system's "first principles" approach is time consuming. To achieve successful walking durations of 20 seconds, 6000 tournaments must be conducted. Since each tournament requires separate trials of four individuals, the control system needed to perform 24,000 distinct trials lasting up to 20 seconds each in order to evolve a walking pattern. In addition, the pattern that resulted is not controllable, as it is only judged on its ability to walk forward. Finally, as with ANN approaches, the behaviour of the control system is encoded in a set of indecipherable numbers, and so cannot be understood by casual examination.

A review of these attempts at a "first-principles" adaptive approach to developing walking control systems leads to the conclusion that pre-knowledge of all or part of a system model will be an essential part of any successful strategy. The most appropriate place for adaptive approaches will be in hybrid classical-adaptive control systems.

### 2.3.7 Hybrid classical-adaptive control systems

If the researchers know how a system should behave, using that knowledge to help "bootstrap" an adaptive controller will greatly reduce the training times required to learn successful behaviour.

#### 2.3.7.1 Adaptive components in a larger system

Some of the most successful implementations of adaptive control to biped walking strategies have been where the walking problem is divided into smaller component parts, where each component of the problem is controlled by a separately trained adaptive system [42], [43], [44]. Identifying these components requires the use of a dynamic model such as those used in classically based control strategies.

#### 2.3.7.2 Adaptive "tuning" of classical solutions

Almost any classical control system can become adaptive if adaptive systems are used to tune the classical system. For example, the classical solution to an inverted pendulum problem can be described using a single equation whose behaviour can be controlled by modulating four numbers. The equation is repeated here for convenience:

$$u = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x}$$

Finding appropriate values for these numbers is a trivial task to determine experimentally, and can be found by a human after a handful of trials. The same task could just as easily be performed by an adaptive system such as genetic algorithms.

This approach of applying adaptive tuning to a classically based controller will be investigated during the course of this thesis, when a genetic algorithm tuning system is applied to the Torso Driven Walking control system.  These experiments are described in Chapter 6.6.

# 3 Torso Driven Walking

"The faster a man runs, the more he leans forward towards the point he runs to and throws more weight in front of his axis than behind.", Leonardo da Vinci [86].

This thesis is proposing a simple control system for bipedal locomotion based on the idea that our desired outcome of walking or running can be obtained as a side effect of actions taken to maintain balance. This is a different philosophy to that behind the majority of walking algorithms, especially those which create and replay a joint trajectory. Examples of these trajectory based control systems include most zero moment point (ZMP) strategies [31], [51], [53], [62], [75], [70] as well as many reinforcement learning [76], [77], [78] and genetic algorithm based adaptive control systems [70], [74]. Systems such as the ZMP based controller employed by Honda's "Asimo" robot [75] often utilise pre-planned "corrective actions" in an attempt to maintain balance while walking.

Rather than trying to keep balanced while we walk, the Torso Driven Walking control system walks to keep balanced. The result is a solution that provides a robust control system, capable of traversing uneven terrain, modulating forward velocity, and even of running. Running and walking gaits are treated the same way by our control system, eliminating the need to implement multiple control systems in order to achieve different gaits. The TDW control system is simple enough that adaptive training strategies can easily be implemented to improve the system performance in simulation, or online. The entire control system can be completely described by a simple floating point array containing 52 values.

These numbers specifying the behaviour of the TDW control system are all values of clearly defined parameters. Each number has an easily understood impact on the behaviour of the control system. For example, perhaps the most critical parameter value is the “torso offset angle”, the angle at which the control system attempts to hold the torso as it balances. By simply examining these numbers, a human observer can understand how the control system implementation behaves. This contrasts with some commonly used adaptive control system designs. For example, neural network based controllers, such as [11], [13], [79] store their configuration as a collection of neuron ‘synapse’ weights. Some genetic algorithm approaches, such as [69] store their behaviour as a collection of encoded operations mapping sensor readings to internal registers and output signals. For all of these systems, a human attempting to interpret the control system by observing its configuration has very little chance of success.

In this chapter, the “Torso Driven Walking” control system’s architecture and design philosophy is introduced. Experiments with various implementations of the control system have been conducted, and are described in Chapters 5 and 6. In Chapter 5, the “torso attitude control” subsystem of “Torso Driven Walking” is tested on a simple balancing robot, the wheeled “Ballybot”. In Chapter 6, the complete “Torso Driven Walking” control system is implemented on a simulated planar biped robot. Preliminary work in porting the control system to a physical robot is also described. Specific adaptations of the control system to address implementation issues with these environments are discussed in more detail in the relevant chapters.

## 3.1 Control system issues to consider

When planning the control system design, a number of desirable traits were identified that required consideration.

- Where possible, a walking algorithm should be able to utilise the robot’s natural dynamics.
- Constraints imposed by the robot structure must be considered.

- High level control of the robot should be simple.
- The robot gait should be variable, without requiring significant re-calculations.
- Running gaits should not be treated differently to walking gaits.
- The control system should emphasise force control over joint control.
- A gait should not simply replay recorded joint trajectories.
- The robot should not need to maintain static contact between its supporting feet and the ground.
- The control system must be suitable for experiments using adaptive techniques such as genetic algorithms.

Each of these requirements has influenced the design of the "Torso Driven Walking" control system to varying extents.

### 3.1.1 A general control solution

There are almost as many different walking control systems as there are experimental robots. This is partly because the physical dynamics and capabilities of each robot design are never the same, but is also due to the fact that control system reuse does not appear to be a significant consideration in the design of most walking systems. Often only very high level concepts behind a specific control system's design can be reused on different platforms.

Concessions towards providing a "generic" control system have been made in two ways:

1) The control system design is strongly hierarchical. Components of the walking algorithm which are more likely to be reusable (such as the Torso Attitude Controller) are separate from components that are more likely to be platform specific.

2) The design of the control system is such that as long as the target robot has a similar morphology, the control system can be adapted to the new environment (demonstrated in Chapter 6.7 and 6.9).

### 3.1.2 Utilise robot's natural dynamics

Where possible, a robot should try to make use of the natural dynamics of its structure, in order to assist its motion. For example, allowing the free leg to swing forward at its natural swing rate will use far less energy than if the leg is being constrained to follow a faster or slower trajectory. In the first case, no energy needs to be input into the system, while in the second the robot's motors will need to work continuously to maintain the planned velocity. It has been shown that it is possible to design bipedal robots that walk with very little energy input. McGeer introduced an entirely passive biped that used gravity, and the natural dynamics of the carefully designed robot system, to walk down a gentle slope [58]. No additional control, or energy input was required. Obviously, the ability to take advantage of these kinds of energy savings will be severely influenced by the target robot's physical construction.

A disadvantage with this approach to design is that a control system making strong use of a robot's physical characteristics is not likely be easily transferable to a different robotic platform. The more a robotic control system is designed to take advantage of robot specific dynamics, the less generic is the resulting control system.

The "Torso Driven Walking" control system has been designed to take a compromise approach to efficiency. Our control system is capable of operating at various rates, moving through the stages of a walking gait as a result of triggers, and adaptable timers. It is during commissioning of the control system on a specific robot that the system can be tuned to try to work with, rather than against, the robot's natural movements.

### 3.1.3 Constraints imposed by physical robot

A related issue to using the robot's natural dynamics to assist walking are constraints imposed by a physical robot design. These constraints could include factors such as limited motor torque, availability and quality of state sensors, processor speed and joint friction.

Consider the possible configurations of the knee of a bipedal robot's supporting leg during the "single support phase" of a walking step. Many current walking algorithms require the knee to be bent at a fairly large angle, which places a considerable strain on the knee actuators if they are to prevent the robot sinking to the ground.

The reasons for keeping a bend in the knee can vary depending on the control system: LIPM based control systems need to keep the knee bent during the support phase of a step in order to allow the torso to follow a flat trajectory over the ground [7]. Both Sony and Honda's humanoid robots exhibit a "bent knee" walking style, although the reason for this is not clear. A bent knee will lower the robot's centre of gravity, which would help satisfy the ZMP based stability criteria used by these robots. It is possible that the bent knee position was chosen for purely aesthetic reasons.

Regardless of any benefits that may result from bent kneed walking, the "Torso Driven Walking" control system that has been designed to severely limit the magnitude of the knee joint angle during the support phase. This is mainly due to the physical constraints of limited actuator torque, and significant frictional forces present in the experimental robot's knee.

### 3.1.4 Simple high level control

At the end of the day, walking is just a way of getting around. A robot should not be distracted from its primary function by figuring out how to walk – presumably it will already have some useful task to perform. With that in mind, the "Torso Driven Walking" control system is designed to be as simple to operate as possible. The control system takes as its input parameters either of two variables:

- Target horizontal velocity, or:
- Target horizontal displacement.

As much as possible, issues such as uneven ground, unexpected perturbations, and gait modifications should be taken care of implicitly by our control system design. At the very least, they should not be exposed to the high level control interface.

### 3.1.5 Variable gait speed.

Walking faster is not simply a matter of moving your legs faster. This is evident from an examination of static walking machines. If they move too quickly, they tend to fall over! A control system needs to make some fundamental modifications to its gait in order to change speeds. One of the objectives considered is to enable the control system to easily modulate its walking speed, without having to make changes to the control system itself. A robot should not need to "remember" different gait patterns for every selectable velocity, nor should it need to make complex gait re-calculations in order to change speed.

Because the control system has been designed to move in order to maintain balance, its most basic requirement is to be able to easily modulate its velocity. As a consequence, a variable gait speed is obtained at no additional computational cost.

### 3.1.6 Force control vs. positional control.

Many robotic control systems are primarily concerned with forcing the robot to follow prescribed joint trajectories. This is an inappropriate approach to controlling a balancing robot. Using positional control, unless the robot's movements are constrained to ZMP based stability criteria (see Chapter 2.3.1), there is no guarantee that the robot will remain balanced. Even with joints fixed to known positions, it is still possible for the robot to tip over. Some kind of force control is an essential part of dynamically controlled robot balance.

One problem with force control in robotics is the increased complexity. The kinematic equations required to accurately determine required joint torque for desired end effector forces are complicated and time consuming. Additionally, these dynamic equations are very specific to the hardware platform on which they have been calculated. Any changes to this platform, such as improved motors, or changing joint frictions will require the dynamic model to be redesigned. Positional control is much simpler.

In the "Torso Driven Walking" control system, the primary balancing task is performed by a robot hip joint using force control, while the job of keeping the legs underneath the robot is performed using positional PD control loops.

### 3.1.7 Avoid pre-recorded joint trajectories.

One superficially obvious way to make a robot walk is to generate a static set of joint trajectories. As the joint set points are played back, the robot will move following a prescribed path. If playing back this joint trajectory causes the robot to move its limbs in a walking motion, and if the robot remains upright during the joint trajectory playback, then we have a walking control system. This is the general approach that has been followed by a number of research groups (including [70], [74-78]), and in experiments described in Chapter 4. Some of the problems with this approach are:

- The resulting control system is not adaptable – dynamically changing speeds, stride length, etc is not possible.
- Complicated joint trajectories use up a lot of memory, still often in short supply on an autonomous robot.
- The control system is not portable – new trajectories will need to be found after any change to the robot.

While the Torso Driven Walking control system does not rely on pre-recorded joint trajectories, some elements of the robot gait are reminiscent of joint trajectory systems. In the Torso Driven Walking control system, the trajectory of the free foot during the single support phase of a walking gait is controlled by driving joint angle set points. These set points are driven in a response to changes in the supporting leg angle, rather than following a trajectory over time.

### 3.1.8 Allow rolling foot-ground contact.

Many walking algorithms require the supporting foot to remain in contact with the ground, and to not roll or tip over. The zero moment point class of walking control systems are a classic example. ZMP systems require the presence of a supporting polygon in order to satisfy their stability criteria, and need to ensure the robot's movements do not cause any rotation of the supporting polygon out of the ground plane. During the single support phase of a biped's walking gait, the supporting polygon is by definition the contact area between the robot foot and the ground. If the control system is doing its job then the supporting foot will not be allowed to roll or tip over. Note that

ZMP stability does allow the supporting foot to slide or twist – as long as solid contact is maintained with the ground surface. It is just "tipping" movements that are illegal.

For reasons that have been discussed in Chapter 2.3.1, consistent ground contact is an artificial constraint that is not an appropriate foundation on which to base a control system for dynamic biped robots. It is certainly not a restriction to human walking, where the supporting foot rolls from heel to toe over the course of every step.

To emphasise the fact that this kind of contact is unnecessary in a walking system, 'point' feet are used on all the robots used during development of the TDW control system. These robot's feet are constantly rotating with respect to the ground. While the robots have only a single foot on the ground, they have no (or a very small) supporting polygon.

### 3.1.9 Design with adaptive learning in mind.

Adaptive control systems promise the ability to learn how to control a system that is not fully understood. In robotic systems this situation is frequently the case. Friction forces are unpredictable, etc.

The "Torso Driven Walking" control system can be tuned to operate on various hardware platforms, or in different environments, by manipulating the tuning parameters of the control program. For example, two of the primary variables are the PD parameters controlling the response of the free leg to changes in set point positions.

Tuning of these parameters is a good task for applying adaptive strategies. The problem size is reasonable, being limited to 51 floating point variables. And the fitness is easily determined – the longer the robot system stays balanced, the fitter the solution.

Experiments using genetic algorithms and programming techniques for tuning our "Torso Driven Walking" control system are presented in Chapter 6.6.

## 3.2 Control system design

The following design describes in detail the structure and components making up the "Torso Driven Walking" control system. Experiments implementing components of the control system are the focus of Chapters 5 and 6.

### 3.2.1 Overview of control system

The "Torso Driven Walking" control system uses the observed system state of the robot, together with a control signal input by an operator, to determine a vector of output joint torque values to be applied to the robot's joints. The control system's objective is to maintain balance of the robot, while satisfying the input command as closely as possible. Figure 3-1 illustrates the control system's context diagram.

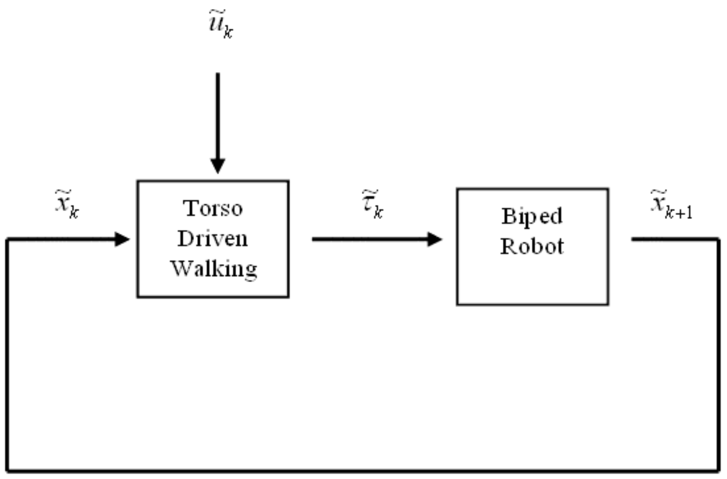


**Figure 3-1 Torso Driven Walking control flow context diagram.**

The basic hypothesis used by the "Torso Driven Walking" control system is that: if the robot torso remains upright, and the robot's feet are below the torso, then the robot has not fallen over. The sequence of images shown in Figure 3-2 illustrates the desired behaviour of the Torso Driven Walking control system.

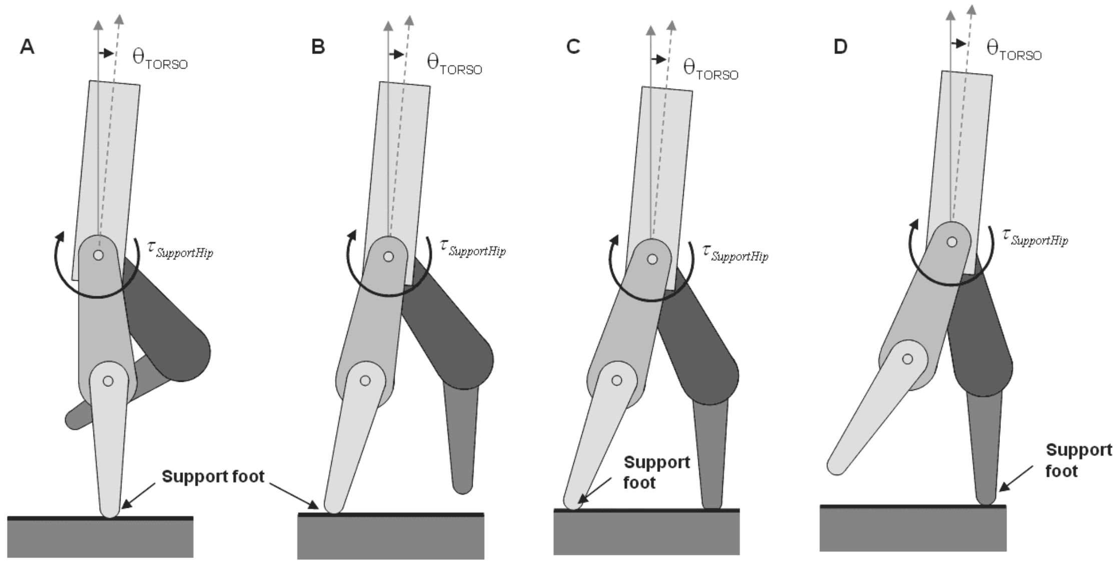


**Figure 3-2 Torso Driven Walking algorithm behaviour, for a planar robot. (A) From an initial standing position, the robot balances the torso by applying a torque to the supporting hip. (B) This causes the supporting leg to drift away from its central position. The raised leg needs to move into position to take over the supporting leg role. (C) Eventually the "raised" leg strikes the ground. Torso balance continues to be achieved by applying torque to the single hip designated as the support hip. (D) As the robot's weight shifts to the second leg, the control system needs to switch the "support leg" designation and lift the old supporting leg.**

Starting from an initial position of balancing on one leg, the control system balances the torso by applying a joint torque to the robot support leg's hip joint, while standing on one foot.

As the balancing torque is applied to the supporting leg's hip, the foot of the support leg will tend to move away from the centre line of the robot, just as our own feet move away from our torso as we walk. Whenever the supporting leg moves forwards or backwards, the raised leg needs to be moved in the opposite direction, so that it will be in position to take over the support leg's role.

At some point, the extended leg will contact the ground, and the robot will begin the "double support" phase of its gait. Even though both feet are in contact with the ground during this phase, only one of the legs is designated as the "supporting leg" by the control system. This leg's hip joint continues to be used to balance the robot torso.

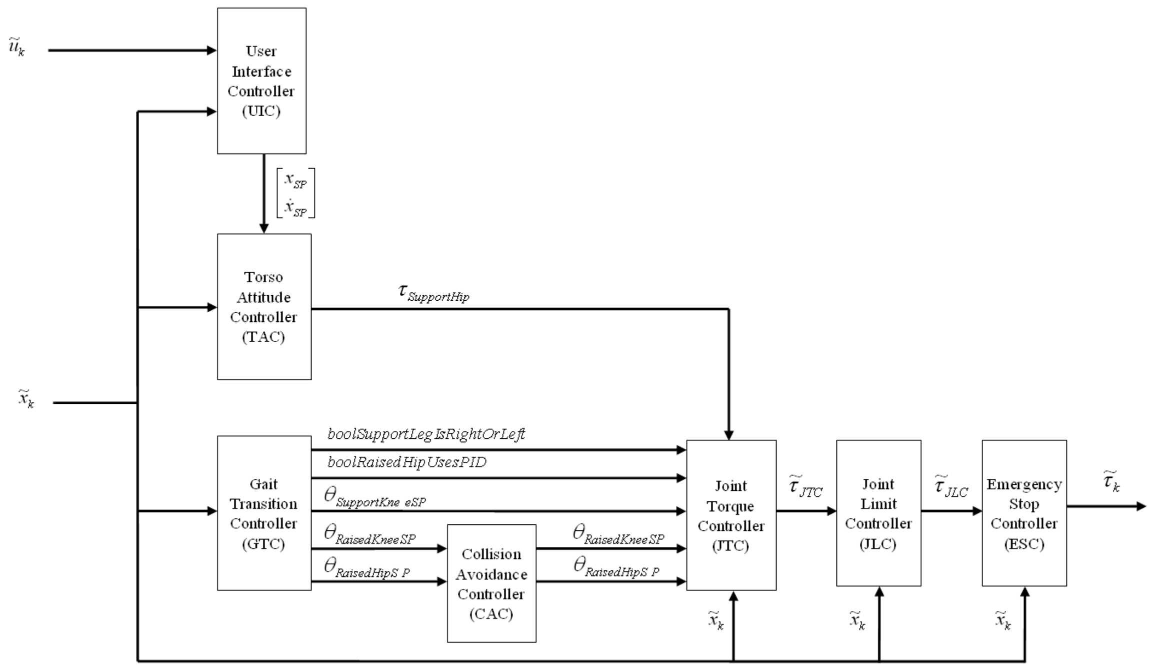
As the robot's weight transitions over the "non-support" leg, the control system must change the leg being designated as the support leg, and the old support leg is lifted off the ground.

At any time, this flow of robot stance transitions must be reversible, in case the balancing torque applied to the robot's supporting hip causes the robot to change direction.

If the robot's steps are small enough, and fast enough, then the entire system should remain balanced in much the same way as a simple inverted pendulum.

### 3.2.2 System model

In order to achieve this behaviour, the Torso Driven Walking control system has been developed a hierarchical system consisting of a number of distinct, simple components. Figure 3-3 shows the system model illustrating the relationship between components of the Torso Driven Walking control system.



$\tilde{u}_k$
User Interface Controller (UIC)
$\begin{bmatrix} x_{SP} \\ \dot{x}_{SP} \end{bmatrix}$
Torso Attitude Controller (TAC)
$\tau_{SupportHip}$
$\tilde{x}_k$
boolSupportLegIsRightOrLeft
boolRaisedHipUsesPID
$\theta_{SupportKneeSP}$
Gait Transition Controller (GTC)
$\theta_{RaisedKneeSP}$
$\theta_{RaisedHipSP}$
Collision Avoidance Controller (CAC)
$\theta_{RaisedKneeSP}$
$\theta_{RaisedHipSP}$
Joint Torque Controller (JTC)
$\tilde{\tau}_{JTC}$
Joint Limit Controller (JLC)
$\tilde{\tau}_{JLC}$
Emergency Stop Controller (ESC)
$\tilde{\tau}_k$
$\tilde{x}_k$
$\tilde{x}_k$
$\tilde{x}_k$

**Figure 3-3 Torso Driven Walking control system components.**

### 3.2.3 Breakdown of components:

The components making up the Torso Driven Walking control system include:

1) User interface controller (UIC). A control interface is provided to safely convert user supplied target velocities or displacements into set point values for desired system state variables.
2) Torso attitude controller (TAC). An inverse pendulum control model is used to determine hip torques required to keep the robot's torso upright.
3) Gait transition controller (GTC). A state machine, primarily driven by the position of the supporting leg in relation to the torso, is used to prepare for and execute transitions between supporting feet. This is the component that executes a walking gait.

4) Collision avoidance controller (CAC). A "virtual model" controller that works to prevent collisions between the robot links by maintaining an exclusion zone repelling links in danger of colliding.
5) Joint torque controller (JTC). Responsible for managing the torque being applied to each joint. Supporting hip torque is provided by the Torso Attitude Controller, while set point positions for the remaining joints are supplied by the Gait Transition Controller. PD control is used to drive joints to positions commanded by the Gait Transition Controller.
6) Joint limit controller (JLC). A second "virtual model" controller that operates on the robot joints, ensuring they continue operating within desired ranges.
7) Emergency stop controller (ESC). Extreme limit shutdown commands are used to help prevent the robot from damaging itself in the event of controller errors, or accidents.

This modular approach to control system design is not new. Raibert's hopping machines [55] also integrated a number of simple yet distinct controllers that working together result in robots that hop about to maintain dynamic balance.

### 3.2.4 User interface controller (UIC).

This is the interface between the operating software and the control system. It provides a simple interface, allowing for control of the robot's position or velocity using a single input variable.

#### 3.2.4.1 Objective

Take commands from the user, and pass them on to the rest of the control system in a measured way. The subsystem is responsible for preventing large jumps or discontinuities in the target set point values for position and velocity that are used as inputs to the inverted pendulum based Torso Attitude Controller.

### 3.2.4.2 Implementation

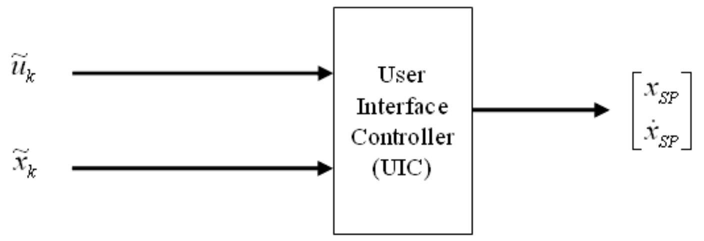

$\tilde{u}_k$
$\tilde{x}_k$
User Interface Controller (UIC)
$\begin{bmatrix} x_{SP} \\ \dot{x}_{SP} \end{bmatrix}$

**Figure 3-4 The user interface controller (UIC) uses the current system state, together with an input command signal to generate a set point trajectory for the robot's displacement.**

The User Interface Controller accepts an input control command, specifying a target position or speed that the user has sent to the robot. Each time step *k*, the User Interface Controller will update the internal control system set point variables for displacement ($x_{SP}$) and velocity ($\dot{x}_{SP}$). These parameters are then used as inputs to the Torso Attitude Controller subsystem.

Boundary values for the set point displacement $x_{SP}$ should be set to an area within the robot's workspace. It is also reasonable to set minimum and maximum limits to the velocity set point $\dot{x}_{SP}$. The User Interface Controller prevents discontinuities in the set point signals by enforcing a maximum acceleration that can be applied to the robot's set point state as it attempts to satisfy the command signal.

## 3.2.5 Torso attitude controller (TAC)

This component is the heart of the control system. Indirectly, the Torso Attitude Controller drives all other parts of the Torso Driven Walking control system.

### 3.2.5.1 Objective:

The single goal of the Torso attitude controller at each control interval is to determine a joint torque which – when applied to the supporting hip – will keep the torso upright. This component of the control system is not concerned with any other aspects of the robotic system. It is assumed that issues such as coordinating the exchange of supporting legs and holding the torso away from the ground will be managed by other elements of the control system.

This kind of torso attitude correction has been used by Raibert to correct torso inclinations in his hopping robots [55]. The difference with the "Torso Driven Walking" system is that Raibert's hopping machines remained balanced primarily through careful selection of foot placements. His torso attitude correction algorithm is of secondary importance. In Torso Driven Walking, the Torso Attitude Controller is the key component for maintaining balance.

### 3.2.5.2 Implementation

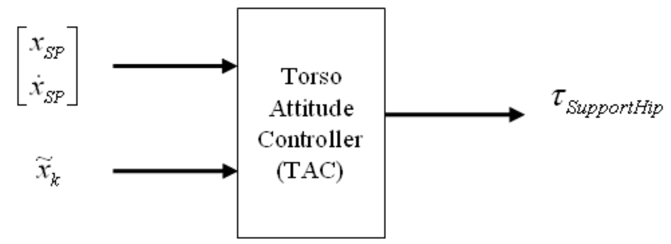

$\begin{bmatrix} x_{SP} \\ \dot{x}_{SP} \end{bmatrix}$
$\tilde{x}_k$
Torso Attitude Controller (TAC)
$\tau_{SupportHip}$

**Figure 3-5 The torso attitude controller (TAC) uses the current system state, together with an updating target position and velocity to determine a balancing torque to apply to the support leg's hip joint.**

i. Torso system model

Everything above the waist of the biped robot is treated as part of a single rigid link, called the torso. In the interests of increasing simplicity, the influence of arm, head and flexible torso movement on the robot has been disregarded in the development of a system model. The simplified 3-D system model is shown in Figure 3-6.

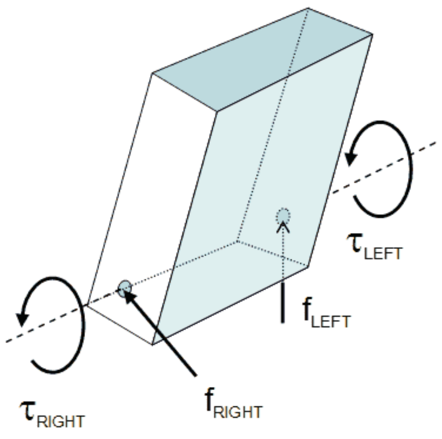


**Figure 3-6 Torso system model.**

For the purposes of this discussion, only the planar case is considered (which is an approach taken in the experimental investigations as well). As a result, the system model can be simplified further, as in Figure 3-7.

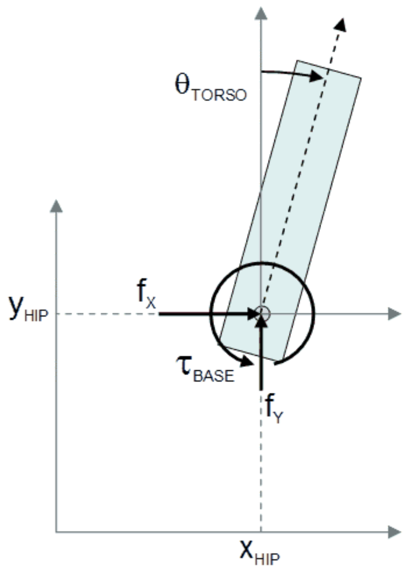


**Figure 3-7 Planar torso system model**

**Table 3-1 Torso system model state variables**

| | |
|---|---|
| $\tau_{BASE}$ | Torque acting on the base of the robot's torso. This is the sum of both the left and right hip joint torques. (Including any friction components). |
| $x_{HIP}$, $y_{HIP}$ | World coordinates of the hip joint in the sagittal (x-y) plane of the robot. |
| $f_x$, $f_y$ | Translational force components, transmitted to torso by the lower limbs. |
| $\theta_{TORSO}$ | Angle of inclination of the torso. |

The left and right motor torque is known, so if frictional torque is assumed to be negligible, $\tau_{BASE}$ can be calculated:

$$\tau_{BASE} = \tau_{LEFT} + \tau_{RIGHT} \tag{1}$$

The translational forces ($f_x$, $f_y$) transmitted to the torso by the robot's legs come from three primary sources:

1) Transmitted ground reaction force through the supporting leg
2) Dynamic forces induced by the free leg's swinging motion
3) Collision impulses as the free leg strikes the ground

All of these translational force components are difficult to calculate on-line, and require an accurate system model to give meaningful results. In many cases the system model includes a significant unknown component, making accurate estimation of transmitted ground reaction force and dynamically induced forces difficult. Including the influence of collision impulses is an even more difficult task – so much so that it is unreasonable to attempt to analytically account for these forces in an online control system.

As a result, the TAC subsystem disregards all components of $f_x$ and $f_y$. Instead the effect of these forces is treated as an external noise, and must be compensated for by the system's natural robustness. This means that the system diagram can be simplified even further, as shown in Figure 3-8.

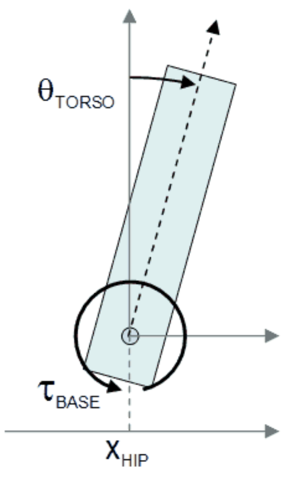


**Figure 3-8 Planar torso system model, horizontal translational forces are assumed to be small compared to $\tau_{BASE}$, and are omitted from the diagram. Vertical translational forces are assumed to simply hold the base of the torso at a constant height.**

ii. Control rule

The simplified torso system model shown in Figure 3-8 is in the form of an inverted pendulum system. Inverted pendulum systems have a well known classical solution that can be used to balance the pendulum at a fixed position. A balancing horizontal force can be determined using a simple linear relationship between four state variables [26]. The classical solution to the inverted pendulum problem takes the form:

$$u = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x} \qquad (2)$$

**Table 3-2 Equation (2) variable definitions**

| | |
|---|---|
| $u$ | Control variable. Usually this variable is the applied balancing horizontal force. In the case of the TAC, the variable specifies a balancing hip joint torque $\tau_{BASE}$. |
| $\theta$, $\dot{\theta}$ | Angle of inclination, and angular velocity, of the pendulum. |
| $x$, $\dot{x}$ | Pendulum base horizontal displacement and velocity. |
| $\{a,b,c,d\}$ | A vector of scalar values. |

Given desired performance characteristics in terms of settling time and responsiveness, appropriate scalar values for $\{a,b,c,d\}$ that result in successful balancing can be determined analytically in a process described by Ogata [26]. This approach requires an accurate system model to be effective. As previously discussed, an accurate system model is something we do not have. In practice, it proves to be far more practical to experimentally determine values for the scalar variables. In general terms:

- Increasing the value of '*a*' will cause the system to respond more vigorously to changes in angle.
- Increasing '*b*' will mitigate the effects of '*a*', and help prevent overshooting the desired torso angle.
- Increasing the value of '*c*' will cause the system to respond more vigorously to deviations in horizontal displacement.
- Increasing '*d*' will mitigate the effects of '*c*', and help prevent overshooting the desired torso angle.

This classical solution to the inverted pendulum control system is used to create a control rule for the "Torso attitude control" component. Since it is unsatisfactory to simply balance the robot in one place, a number of new variables to control the robot's position and velocity have been introduced.

The TAC component takes as input values the set point position ($x_{SP}$) and velocity ($\dot{x}_{SP}$) values output by the UIC component. Together with $x_{HIP}$ and $\dot{x}_{HIP}$ (the displacement and velocity of the torso's base), these values are used to calculate the $x$ and $\dot{x}$ terms from equation (2).

As a result, the equation used to balance the robot's torso becomes:

$$\tau_{BASE} = a\left(\theta_{TORSO} - \theta_{OFFSET}\right) + b\left(\dot{\theta}_{TORSO} - \dot{\theta}_{OFFSET}\right) + c\left(x_{HIP} - x_{SP}\right) + d\left(\dot{x}_{HIP} - \dot{x}_{SP}\right) \tag{3}$$

If $\tau_{SUPPORT}$ is the torque to be supplied to the support leg's hip joint, and $\tau_{FREE}$ is the torque being supplied to the free leg (by the JTC component), we can use Equation (1) to express $\tau_{SUPPORT}$ by (5):

$$\tau_{SUPPORT} = a\left(\theta_{TORSO} - \theta_{OFFSET}\right) + b\left(\dot{\theta}_{TORSO} - \dot{\theta}_{OFFSET}\right) + c\left(x_{HIP} - x_{SP}\right) + d\left(\dot{x}_{HIP} - \dot{x}_{SP}\right) - \tau_{FREE} \tag{4}$$

Equation (4) is the control signal output from the TAC component, and used to drive the supporting leg's hip joint.

3.2.5.3 Complications and assumptions:

i. Arms and heads or elastic torsos

The control system simplifies the entire robotic system above the waist, replacing any torso arms and head with a rigid single link. The controller relies on the natural robustness of its inverted pendulum based control system to compensate for any changes in the torso structure, by treating the dynamic impact of possible changes as noise.

ii. Leg mass

Translational forces acting on the hip joints of the torso will be generated by rotational motion of the raised leg, and to a lesser extent by the motion of the supporting leg. The controller assumes that these forces will have a small magnitude compared to other forces acting on the robot's torso. Their influence is treated as noise.

iii. Ground reaction force

Translational forces acting on the hip joints are also generated by ground reaction forces acting on the supporting foot being transmitted through the robot leg to the hip joints. In the simplified system considered by the TAC, the most significant components of ground reaction forces are a resultant of gravity working on the torso, and translational forces generated by the hip torque. As long as the robot's foot remains in contact with the ground, a significant component of horizontal force applied to the base of the torso will be proportional to the applied hip joint torque.

What this means is that it is a reasonable simplification to consider the robot torso an inverted pendulum, and to balance the pendulum using experimentally determined values for $\{a, b, c, d\}$. However, it also suggests that when the robot's foot leaves the ground, the experimentally determined parameters will no longer be valid, as hip torque no longer generates a reactive translational force. The symptoms observed experimentally are that the leg joints fly out of control after the supporting foot leaves the ground. As a result, torque is not applied to the balancing leg's hip if the foot is not in contact with the ground.

To illustrate this, the relationship between hip torque and translational ground reaction force is described in Figure 3-9, and the following equations. To simplify the derivation of the system equations the following initial assumptions have been made:

- The balancing robot is modelled as a 2-link robot.
- Contact point does not slide, and the foot remains in contact with the ground.
- Supporting knee is 'locked'.

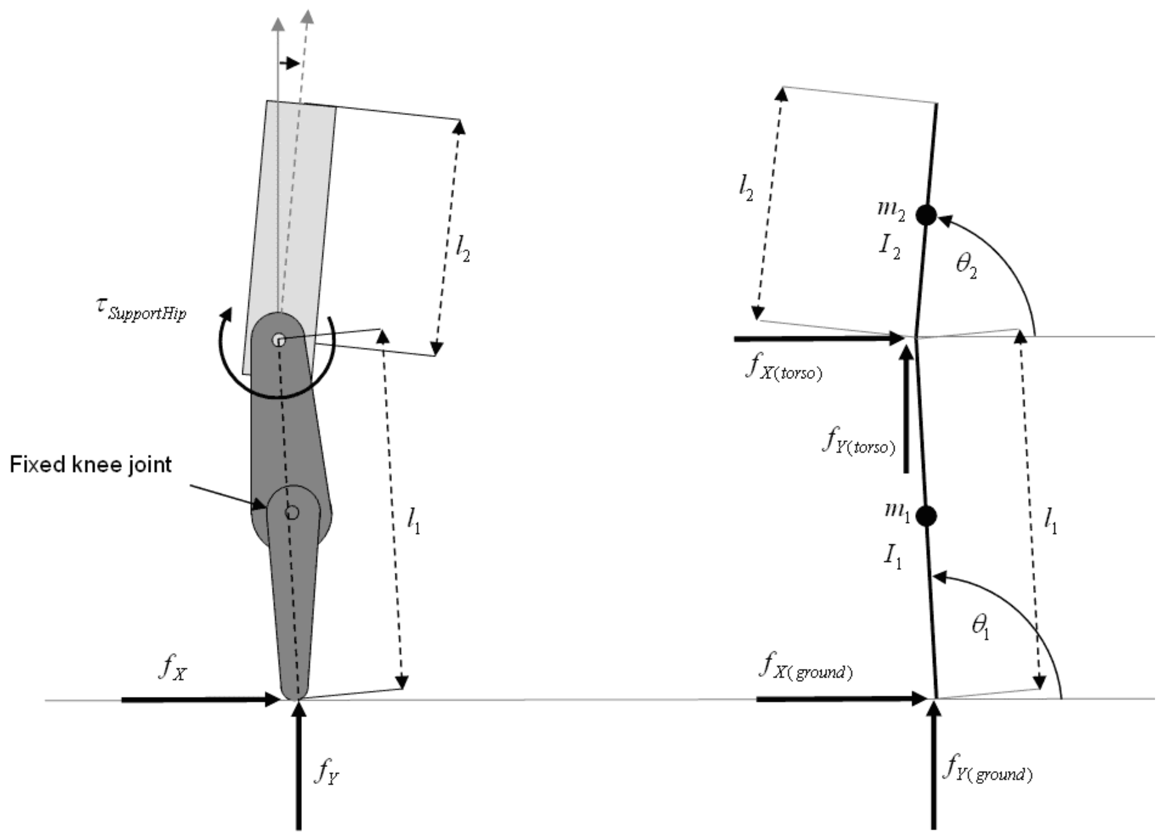


**Figure 3-9 A balancing robot is modelled as a planar 2-link robot with a fixed knee joint. Assuming the robot's foot remains fixed on the ground, the relationship between torso translational force, ground reaction force, and link angular accelerations is described by Equations (5) to (10).**

Equations (5) to (10) describe the behaviour of the two link planar robot in Figure 3-9.

The equations are derived through application of the Newton-Euler iterative algorithm [41]. The derivation of these equations is complex, and so has not been included here. Readers interested in the derivation of these equations should refer to Appendix C-4 , where a general system model for describing multi-link planar robotic systems is discussed.

$$f_{X(torso)} = -\frac{m_2 l_2 s_2}{2}\ddot{\theta}_2 - m_2 l_1 s_1 \ddot{\theta}_1 - m_2 l_1 \dot{\theta}_1^{\,2} c_1 - \frac{m_2 l_2 \dot{\theta}_2^{\,2} c_2}{2} \tag{5}$$

$$f_{Y(torso)} = \frac{m_2 l_2 c_2}{2}\ddot{\theta}_2 + m_2 l_1 c_1 \ddot{\theta}_1 + g\, m_2 - m_2 l_1 \dot{\theta}_1^{\,2} s_1 - \frac{m_2 l_2 \dot{\theta}_2^{\,2} s_2}{2} \tag{6}$$

$$f_{X(ground)} = f_{X(torso)} - \frac{m_1 l_1 s_1}{2}\ddot{\theta}_1 - \frac{m_1 l_1 \dot{\theta}_1^{\,2} c_1}{2} \tag{7}$$

$$f_{Y(ground)} = f_{Y(torso)} + \frac{m_1 l_1 c_1}{2}\ddot{\theta}_1 - \frac{m_1 l_1 \dot{\theta}_1^{\,2} s_1}{2} + g\, m_1 \tag{8}$$

$$-\tau_{SupportHip} - \frac{f_{Y(ground)} l_1 c_1}{2} + \frac{f_{X(ground)} l_1 s_1}{2} + \frac{f_{X(torso)} l_1 s_1}{2} - \frac{f_{Y(torso)} l_1 c_1}{2} = I_1\,\ddot{\theta}_1 \tag{9}$$

$$-\tau_{SupportHip} + \frac{f_{X(torso)} l_2 s_2}{2} - \frac{f_{Y(torso)} l_2 c_2}{2} = I_2\,\ddot{\theta}_2 \tag{10}$$

**Table 3-3 Planar 2-link robot system variables**

| | |
|---|---|
| $f_{X(torso)}$ | Horizontal force experienced by the base of the torso. |
| $f_{Y(torso)}$ | Vertical force experienced by the base of the torso. |
| $f_{X(ground)}$ | Horizontal ground reaction force. |
| $f_{Y(ground)}$ | Vertical ground reaction force. |
| $m_1$ | Mass of the robot leg. |
| $m_2$ | Mass of the robot torso. |
| $I_1$ | Moment of inertia of the robot leg. |
| $I_2$ | Moment of inertia of the robot torso. |
| $l_1$ | Length of the robot leg. |
| $l_2$ | Length of the robot torso. |
| $\theta_1, \dot{\theta}_1, \ddot{\theta}_1$ | Angle, angular velocity and angular acceleration of the robot leg. |
| $\theta_2, \dot{\theta}_2, \ddot{\theta}_2$ | Angle, angular velocity and angular acceleration of the robot torso. |
| $c_1, s_1$ | Cosine and sine of $\theta_1$. |
| $c_2, s_2$ | Cosine and sine of $\theta_2$. |

|  |  |
|---|---|
| $g$ | Gravitational acceleration (+ 9.81 ms$^{-2}$). |
| $\tau_{SupportHip}$ | Torque applied to the leg by the torso's hip joint actuator. |

Equations (5) to (10) can be rearranged to express $f_{X(torso)}$ as a function of the system variables ($m_1, m_2, I_1, I_2, l_1, l_2, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, g$ and $\tau_{SupportHip}$). However the resulting equation is extremely complex. Instead I consider the case here where:

- The robot torso is held upright, as it will be if the TAC does its job correctly ($\theta_2 \approx \frac{\pi}{2}$).
- The supporting leg's foot is close to the ground projection of the robot's torso ($\theta_1 \approx \frac{\pi}{2}$).
- The robot is not rotating quickly ($\dot{\theta}_1 \approx \dot{\theta}_2 \approx 0$).
- The mass of the leg link is very small, compared to the torso ($m_1 \approx 0, I_1 \approx 0$).

Equations (7) and (9) reduce to:

$$f_{X(ground)} = f_{X(torso)} \tag{11}$$

$$-2\tau_{SupportHip} + f_{X(ground)} l_1 + f_{X(torso)} l_1 = 0 \tag{12}$$

Substitute (11) into (12), we see that:

$$f_{X(torso)} \approx \frac{\tau_{SupportHip}}{l_1} \tag{13}$$

This horizontal reaction force, proportional to the applied torque, works with the torque to assist in balancing the torso "pendulum".

When the simplifying conditions resulting in Equation (13) no longer exist, the relationship between torque and horizontal ground reaction force will not be so strong. This variability in the relationship between horizontal ground reaction force and hip torque is treated as noise by the TAC component of the Torso Driven Walking control system.

### iv. Collision impulses

Each time the robot finishes a step there is a collision between its raised foot and the ground. During a collision, translational impulses will propagate throughout the robot system resulting in almost instantaneous changes in joint angular velocities. It is not reasonable to try and compensate or predict these impulses, as it is an intractable problem to solve. The effects of collisions will vary depending on collision speed, attitude and on the characteristics of the materials colliding. Instead the TAC component treats the effect of collision impulses as noise, and once again the robustness of the inverted pendulum controller is trusted to recover any loss in equilibrium.

### v. Hip joint friction forces

Joint friction can have a serious impact on the performance of any control system, including the "torso attitude controller". The impact of friction is to apply a reasonably constant force, working against the direction of the current joint angular velocity. Even small amounts of friction can adversely impact an inverted pendulum controller, as the effect of friction is to cancel out the small balancing forces generated when the pendulum is nearly vertical. The result is a balancing control system that appears sluggish, as the pendulum often has to fall a significant distance before the effects of friction on the controller are overcome. Where possible a control system should compensate for frictional forces in some way.

The approach to friction compensation is described in Chapter 5. Essentially, the task requires predicting, at each time step, the level of friction acting against the motion of the joint. The control system should then apply an equal and opposite compensation torque to the joint's actuators. This process is not managed by the Torso Attitude Controller, and will not be discussed further here.

### 3.2.6 Gait transition controller (GTC)

The primary responsibility of the gait transition controller is to ensure there is always at least one leg preventing the robot torso from sinking to the ground.

#### 3.2.6.1 Objective

As the robot's torso is balanced by the Torso Attitude Controller, torque applied to the supporting leg's hip will cause the leg to move from its position beneath the torso. The Gait Transition Controller's primary objective is to make sure the raised leg is moved into position to take over the supporting leg role when necessary. An additional responsibility of the Gait Transition Controller is to ensure that the supporting leg holds the robot's weight without buckling.

### 3.2.6.2 Implementation

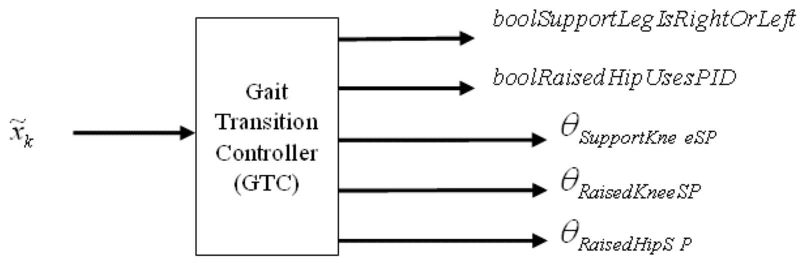


**Figure 3-10 The gait transition controller (TAC) uses the current system state, to generate a joint space target position to which all joints but the supporting hip are attracted.**

The Gait Transition Controller requirements for managing the position of the free leg and supporting leg's knee joint are achieved through the use of a state machine, driven by the position of the supporting leg relative to the robot's torso. Figure 3-11 shows the full Gait Transition Controller state transition diagram, consisting of 14 states, each of which describes a distinct part of the robot's gait.

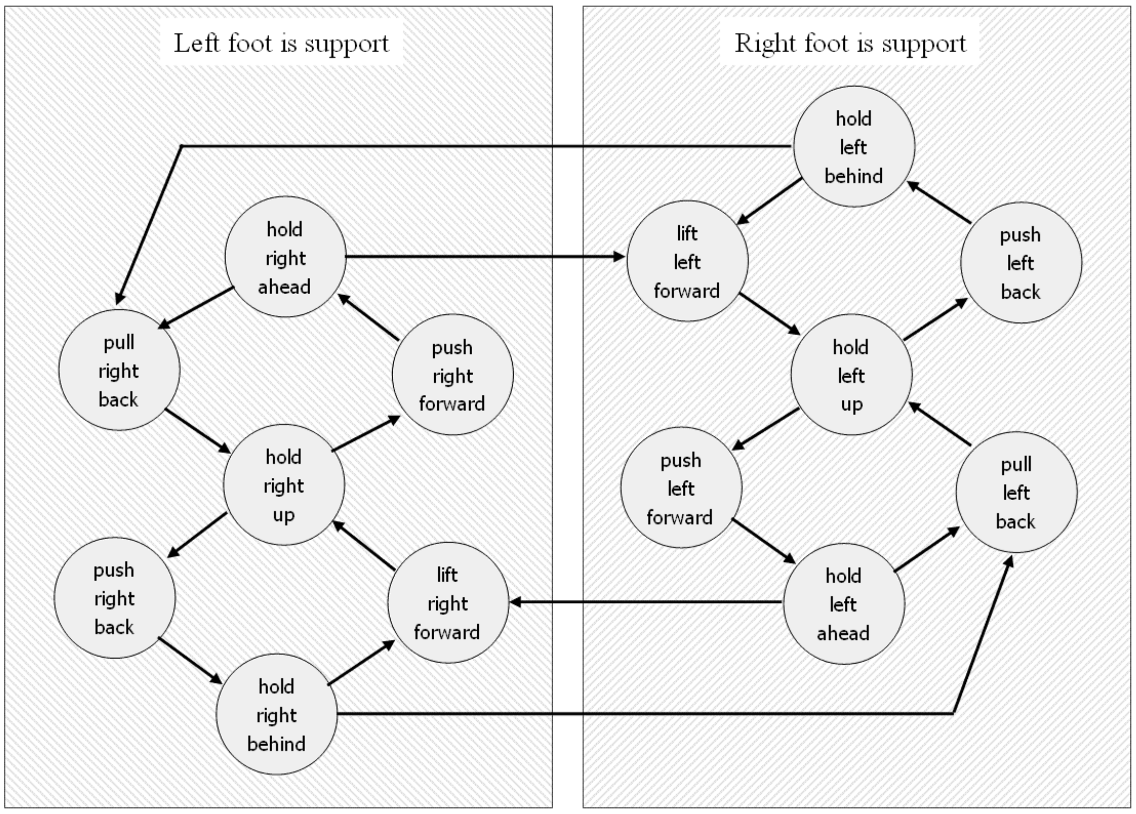


**Figure 3-11 Gait transition controller state transition diagram.**

In each state, the controller has a specific objective, and a number of triggers to initiate transition between states. As the robot walks, the state transition controller follows a clockwise or anticlockwise path through the state machine depicted in Figure 3-11. If the robot is walking forward, the path is clockwise, following these states:

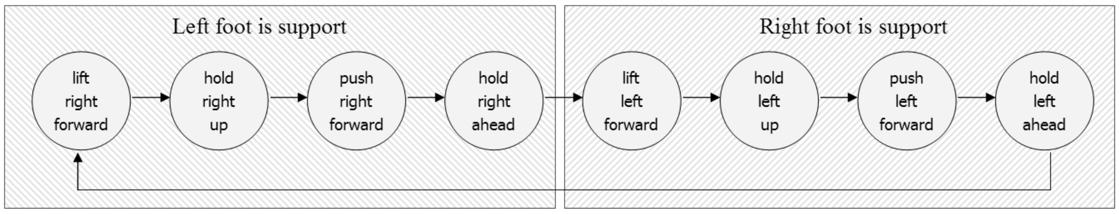


**Figure 3-12 State transitions when walking forward.**

At any time, the state machine may be required to change direction. If the robot is reversing, the path is anticlockwise, following these states:

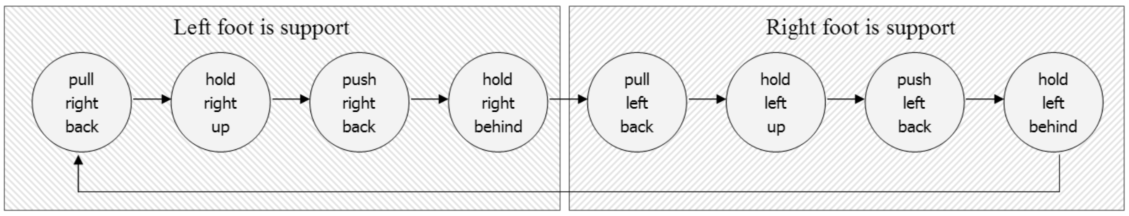


**Figure 3-13 State transitions when walking backward.**

Each of the gait transition controller's states is treated as a distinct module, further decomposing the "Torso Driven Walking" control system. A Gait Transition Controller state is made up of two types of rules; transition triggers and set point trajectories. Together these two components describe a state program that is executed for as long as the state is maintained.

i. Transition triggers

Transition triggers are most commonly fired if a system variable exceeds a tuneable value. For example, the system shifts from the state "*hold left up*" to "*push left* *forward*" when the angle between the supporting (right) foot and the torso moves below a threshold value. Some other kinds of transition trigger may include events such as the raised leg's foot striking the ground, or a joint trajectory transit timer expiring.

ii. Set point trajectories

Set point trajectories are defined by the Gait Transition Controller as a target joint angle, together with a trajectory transit time (ms). At the start of a trajectory, the joint set points are initialised to the corresponding joint angles currently being measured by the robot's system variables. As the trajectory is then executed, the set point values are varied from the initial value to the trajectory target value, over an interval specified by the trajectories' transit time.

The effect of a changing set point trajectory can be visualised as pulling the end of a virtual spring/damper integrated with the robot's joints. As the set points move away from the robot's current position, the joints follow as if they are being dragged by a spring. This is equivalent to a joint-space representation of Pratt's "virtual model control" paradigm [66].

iii. State program

A state program may include multiple joint set point trajectories, which can work in parallel (if they apply to different joints) or in a sequence (if the trajectories apply to the same joint). Together, the transition trigger limit values, joint trajectory set point targets and transit times make up the tuneable parameters of the gait transition controller module.

Whereas the state transition diagram (Figure 3-11) shows 14 possible system states, the only difference between the left and right diagram is which leg has been designated as the "support leg". By specifying that the robot gait should be symmetrical, the number of state programs that must be developed is reduced to seven. A summary of the seven GTC state programs, and the transition triggers and trajectories associated with each program, is provided here. For an example of a specific implementation, please refer to Chapter 6.5.

1) Hold leg up

During this state, the raised leg is moved into a position above the support foot, ready to move in either direction. *boolRaisedHipUsesPID* is **true** for the duration of this state.

The robot transitions to the "*push leg forward*" state if the supporting leg drifts too far back and to the "*push leg back*" state if it drifts too far forward.

2) Push leg forward

During this state, the raised leg is pushed ahead, ready to catch the robot's weight. *boolRaisedHipUsesPID* is **true** for the duration of this state.

The robot transitions to the "*hold leg ahead*" state once the raised foot has struck the ground.

3) Hold leg ahead

During this state, both legs are on the ground. The robot is preparing to shift its weight onto the forward leg. *boolRaisedHipUsesPID* is **false** for the duration of this state.

The robot transitions to the "*pull leg back*" state if the supporting (rear) foot moves back under the robot's torso. The robot transitions to the "*lift leg forward*" state once the supporting leg is judged to have moved far enough away from the torso, of if the supporting leg has become airborne.

4) Lift leg forward

During this state, the rear foot is lifted off the ground and swung forward. *boolRaisedHipUsesPID* is **true** for the duration of this state.

The robot transitions to the "*hold leg up*" state once the foot has been given sufficient time to clear the ground.

5) Push leg back

During this state, the raised leg is pushed back – ready to catch the robot's weight as the robot moves backwards. *boolRaisedHipUsesPID* is **true** for the duration of this state.

The robot transitions to the "*hold leg behind"* state once the raised foot strikes the ground.

6) Hold leg behind

During this state, both legs are on the ground. The robot is preparing to shift its weight onto the back leg. *boolRaisedHipUsesPID* is **false** for the duration of this state.

The robot transitions to the "*list leg forward*" state if the supporting (front) foot moves back under the robot's torso. The robot transitions to the "*pull leg back*" state once the supporting leg is judged to have moved far enough away from the torso, of if the supporting leg has become airborne.

7) Pull leg back

During this state, the front foot is lifted off the ground and swung back. *boolRaisedHipUsesPID* is **true** for the duration of this state.

The robot transitions to the "*hold leg up*" state once the foot has been given sufficient time to clear the ground.

### 3.2.6.3 Complications and assumptions

i. Supporting leg knee joint

Obviously, if the supporting leg is to hold the robot's torso off the ground, some attention must be given to the control of the supporting knee. This role is handled by the Gait Transition Controller by assigning a joint trajectory to the supporting knee that holds the knee to a predefined angle.

1) Initial position

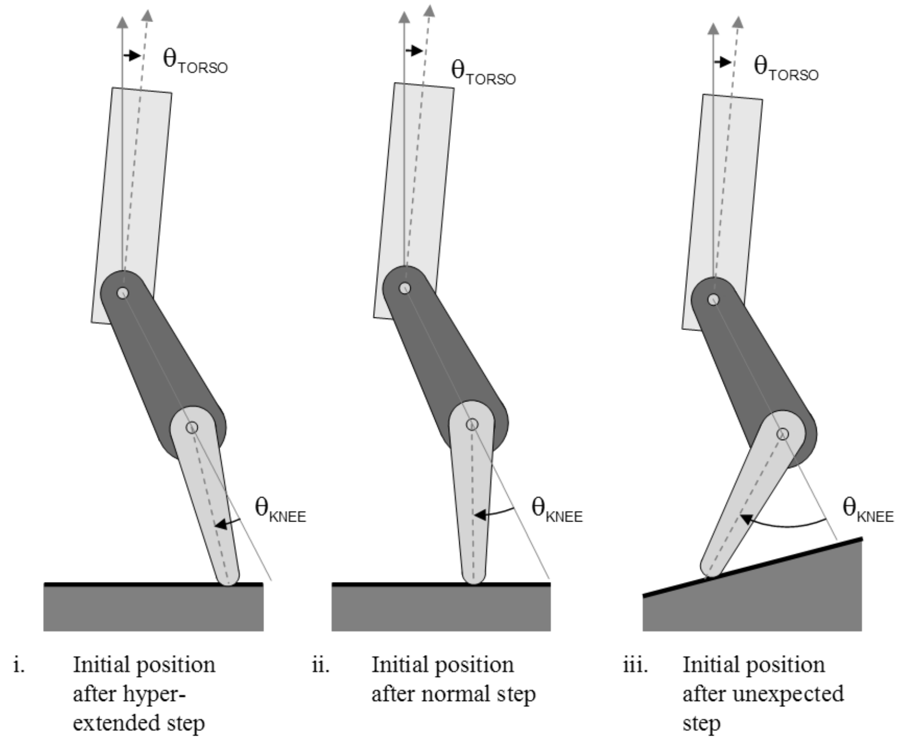

$\theta_{TORSO}$
$\theta_{KNEE}$
$\theta_{TORSO}$
$\theta_{KNEE}$
$\theta_{TORSO}$
$\theta_{KNEE}$
i. Initial position after hyper-extended step
ii. Initial position after normal step
iii. Initial position after unexpected step

**Figure 3-14 Common initial positions for support leg**

When a leg becomes the supporting leg, its initial leg position will be fairly arbitrary – it cannot be predicted by the control system. The robot may have stepped on an obstacle, or stubbed its toe, it could be moving forwards or backwards, etc. Some examples of common initial orientations of the support leg are shown in Figure 3-14.

Regardless of its initial position, the Gait Transition Controller needs to move the support knee joint into an optimal "operating position", and hold it there. It is not required to do any balancing, as this is the job of the Torso Attitude Controller subsystem.

2) Operating position of supporting knee

Determining an optimal operating position for the supporting knee joint is a tuning task that needs to be performed when configuring the control system for a particular robotic platform. The set point value chosen will depend strongly on the capabilities of the physical robot on which the control system is being implemented. Possible choices range from a completely straight knee joint, to a significantly bent knee. Selecting a single "ideal" position will require a compromise in order to balance the advantages and disadvantages of the various configuration options.

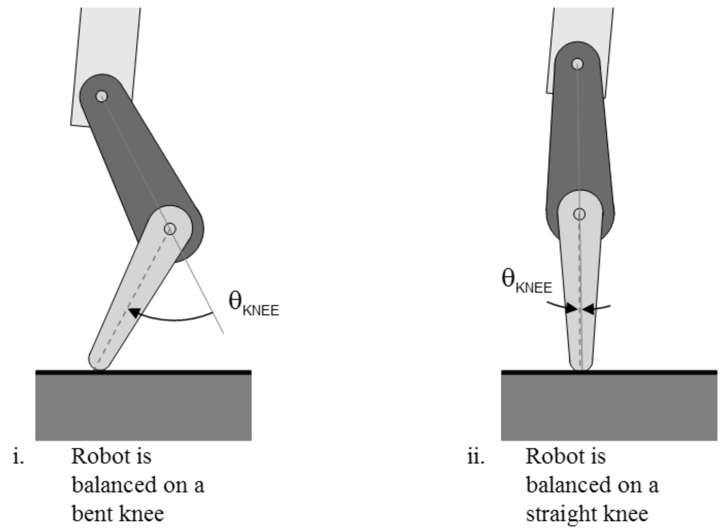

$\theta_{KNEE}$
i. Robot is balanced on a bent knee
$\theta_{KNEE}$
ii. Robot is balanced on a straight knee

**Figure 3-15 Two extreme values for the balancing leg's knee operating position**

There are a number of considerations that will influence the choice of angle when selecting an operating set point position for the balancing leg's knee joint:

3) Minimise supporting knee torque

Minimising the knee torque required will result in reduced power consumption, and wear and tear on actuators. More importantly, a robot will always have a maximum output torque that can be supplied by the joint actuators. If this maximum torque is low, large supporting knee joint angles will not be possible.

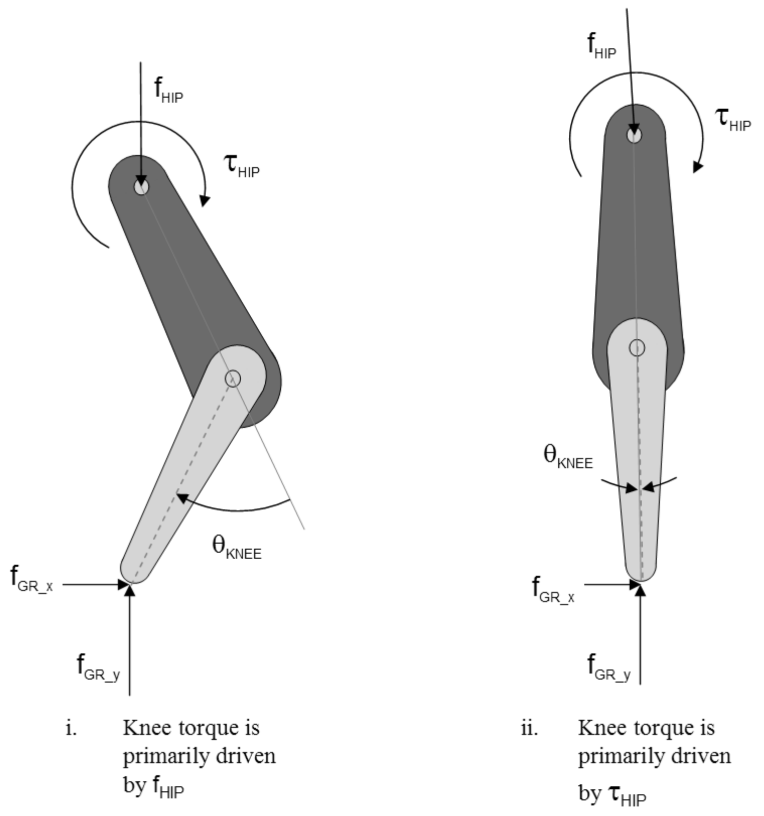


**Figure 3-16 System model showing some of the forces and torques acting on a robot's thigh and shin links.**

Knee torque output by the "support leg controller" component needs to work against deviations between the measured knee position and the desired operating position. These deviations are the result of torques and forces acting on the shin and thigh joints of the robot leg. Figure 3-16 illustrates a simple robot leg, and some of the factors that result in the need for a corrective knee torque.

**Table 3-4 System parameters resulting in the need for stabilising knee torque.**

| | |
|---|---|
| $\tau_{HIP}$ | A torso balancing hip torque applied by the TAC component. |
| $f_{HIP}$ | Translational forces generated by the torso, acting on the hip joint. |
| $f_{GR}$ | The ground reaction force acting on the robot's foot. |

The system model diagrams in Figure 3-16 illustrate two possible configurations for the operating position of the supporting leg. The bent configuration (i.) works best for countering deviations driven by a clockwise $\tau_{HIP}$, while the straight legged configuration (ii.) minimises deviations driven by $f_{HIP}$ and $f_{GR}$. The target robot's construction (especially weight), and the strength of its actuators will determine a suitable operating angle that minimises power requirements made on the knee joint. Usually, a straighter knee joint will be more efficient.

4) Implications of supporting knee angle for maintaining ground contact

Although the Torso Driven Walking control system is designed to cope with loss of ground contact, the Torso Attitude Controller component will suspend application of a balancing torque for the duration time the supporting foot is above the ground surface. As a result, it is prudent to consider the implications of the supporting knee angle on ground contact. A glance at Figure 3-16 will show that if the torso were to rise for any reason, the robot is in danger of losing contact with the ground. In the first configuration, the ground contact will be maintained as the supporting leg extends with a rise in robot torso height. However when the leg is held straight, as in configuration (ii), any rise at all in the torso position will result in loss of contact with the ground.

### 3.2.7 Collision avoidance controller (CAC)

The Collision Avoidance Controller is a "virtual model" controller that works to prevent collisions between the robot links by maintaining an exclusion zone repelling links in danger of colliding.

#### 3.2.7.1 Objective

As a result of some experimental investigations (Chapter 6.6) the need for some additional system to prevent the robot's feet from colliding together was required. The Collision Avoidance Controller was developed as a way to prevent links of the robot that might collide from coming too close together.

#### 3.2.7.2 Implementation

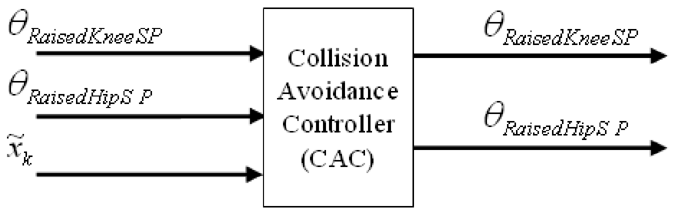


**Figure 3-17 The collision avoidance controller (CAC) uses the current system state, to modify the raised leg joint set point positions, keeping the set point positions outside an "exclusion zone" about components of the robot that might be a danger.**

The Collision Avoidance Controller shown in Figure 3-17 monitors the current robot joint angle state measurements for the supporting leg. The state variables are compared with the raised leg set point positions being output by the Gait Transition Controller subsystem. The Collision Avoidance Controller ensures that the set point angles for the raised leg will keep the leg at a safe distance from collisions with the supporting leg. It does this by adjusting the raised leg set point positions if a safety threshold is breached.

For the case of the simple planar "Legbot" robot discussed in Chapter 6, the robot's rotational motion is constrained to the sagittal plane due to the structure of the robot's feet. The feet have been designed as rods that are fixed to the robot's ankles and aligned perpendicular to the ground, as shown in Figure 3-18.

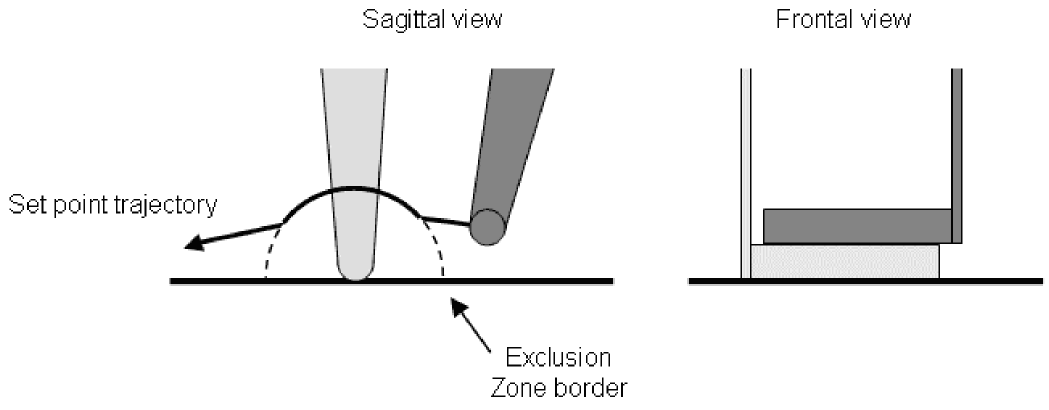


**Figure 3-18 The Legbot robot restricts rotational motion to the sagittal plane through the design of its feet. As a consequence of the robot's structure, the only internal collision possible between the robot's links is if the feet occupy the same space in the sagittal plane. To avoid any collisions, the raised foot's set point position is kept out of an exclusion zone maintained by the Collision Avoidance Controller.**

As a consequence, the only possible collision that can occur between the robot's links occurs if the feet occupy the same space in the sagittal plane. To prevent a collision, the Collision Avoidance Controller will calculate the Cartesian coordinates of the support foot based on sensor measurements, and then calculate the coordinates of the raised foot, as if it were at the set point positions output by the Gait Transition Controller. If the distance between these two feet is less than a reasonable safety margin, the set point foot position will be raised until it is at the safety margin. Figure 3-18 illustrates the resulting set point trajectory for the robot's raised foot. For this example, the tuneable Collision Avoidance Controller parameter is the radius of the set point exclusion zone.

### 3.2.8 Joint torque controller (JTC)

The Joint Torque Controller is responsible for determining the torque that should be applied to each of the robot's joints.

### 3.2.8.1 Objective

The Joint Torque Controller is responsible for coordinating the application of joint torques to the robot's actuators. In the case of the supporting leg's hip joint, the Joint Torque Controller simply passes on the torque specified by the Torso Attitude Controller to the joint. For all other joints, the Joint Torque Controller needs to determine an appropriate torque to use when driving the joints from their current positions towards the set point positions provided by the gait transition controller subsystem.

### 3.2.8.2 Implementation

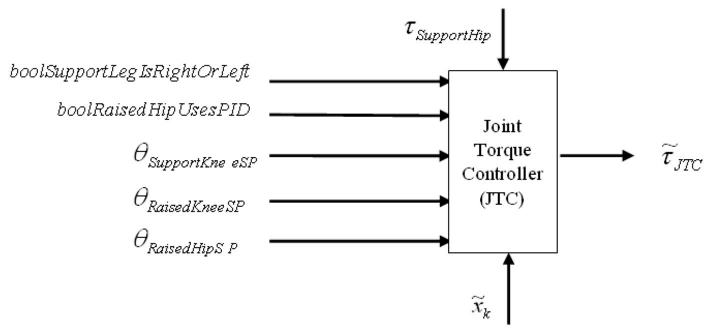


**Figure 3-19 Each time interval, the joint torque controller (JTC) uses the current system state, together with set point commands and balancing torques input from the rest of the control system's components to determine an array of output joint torques to apply to the robot..**

The support leg's hip torque is set to the balancing torque provided by the Torso Attitude Controller subsystem, unless the support foot is not in contact with the ground, in which case the supporting leg's hip torque is set to zero.

The remaining robot's joints are controlled using a simple PD control algorithm, which tracks the changing joint set point positions output by the Gait Transition Controller state programs. Joints controlled via PD include:

- Support leg's knee
- Raised leg's knee

- Raised leg's hip (if the variable *boolRaisedHipUsesPID* is **true**, otherwise torque applied to the hip joint is zero.)

The Joint Torque Controller tuning parameters are the proportional and differential constants of the PD control loop used to drive the robot joints towards the set point positions

### 3.2.9 Joint limit controller (JLC)

The Joint Limit Controller is a "virtual model" controller that operates on the robot joints, ensuring they continue operating within desired ranges.

#### 3.2.9.1 Objective

The Joint Limit Controller is responsible for keeping the robot's limbs operating in the correct configurations by applying a corrective torque if the joint limits are exceeded. The need for a Joint Limit Controller was identified during the course of preliminary experiments conducted using the Torso Driven Walking control system. It was observed that due to the double-jointed nature of our robot's knee joint, as the robot pushed against the ground to step forwards, the knee joint would tend to buckle backwards. The simple PD control supplied by the Joint Torque Controller was insufficient to keep the robot's leg straight.

In nature, it can be observed that the knee joints of birds and animals either bend forwards, or backwards, but not in both directions.

### 3.2.9.2 Implementation

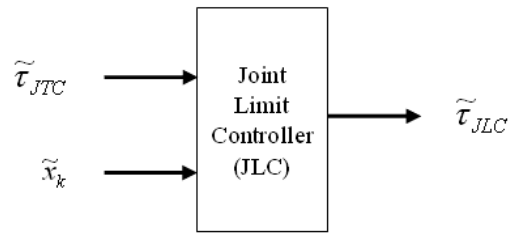


**Figure 3-20 Each time interval, the joint limit controller (JTC) monitors the robot's system state variables. If any joint angles are determined to be outside the limit boundary for that joint, then the torque being applied to the offending joint is updated to correct the transgression.**

Many robots are designed with physical range limits effectively prevent the knee joints from hyper-extending, and locking the leg straight during the support phase of a walking gait. A locking knee joint is one of the primary requirements for passive walking with knees [58]. Another advantage that would be realized from physical limits to joint ranges is a simplification in the control of the robot through the removal of multiple solutions in the robot kinematic equations.

Unfortunately, it is not always the case that robots will posses physical joint limits where they are needed. Additionally, a robot's joints would need to be very well designed in order to withstand the jarring impulses that would be experienced by the joint if the links are allowed to effectively crash into the joint limit as they reach the end of their range of motion.

The Torso Driven Walking control system avoids this problem by implementing the joint limit controller subsystem. The approach I have taken is inspired by the "spring/damper" collision model used in the software simulation dynamic model for the Legbot (see Appendix C-4.2). However, the algorithm is equivalent to an implementation of Pratt's virtual model control [66]. The Legbot simulator's collision model calculates a ground reaction force based on the penetration of the robot into the ground. In contrast, the Joint Limit Controller uses a similar calculation to determine an overriding control torque to apply to joints that have moved beyond the predetermined joint limit positions. The form of the collision model is an asymmetrical PD control system, where:

As well as assisting control, this kind of joint limit enforcement can be applied to all of the robot's joints in order to ensure they remain within the desired or safe operating range. Thus the Joint Limit Controller can be used to prevent damage to the robot, as well as provide active assistance to the control system.

### 3.2.10 Emergency Stop Controller (ESC)

The Emergency Stop Controller is used in a physical robot to mitigate damage that might occur to the robot due to incorrect control system outputs, or unexpected situations. The robot monitors the system's joint angles, and simply shuts off all power to the robot's actuators if any of the joint angles exceed a predetermined safety limit.

#### 3.2.10.1 Objective

Unexpected external forces, hardware failures and software errors are all potential causes of damage to the robot. The objective of the "Extreme limit controller" is to prevent or minimise damage to the robot, in the event of a failure of other components of the control system. Safety is not such a concern in our experiments, as the robots we have built are all small, lightweight and weak. For larger robots, the purpose of this subsystem should be extended to take into account safety considerations.

Together with impacts with the environment, internal collisions have the potential to cause significant damage to a physical robot. Internal collisions occur as joints reach their physical limit values while still travelling at large angular velocities. The potential for damage can be compounded if the joint controllers continue to exert torque after a collision in an attempt to drive the joint through its limit position. Control system errors such as this are particularly common during the implementation and tuning stages of control system development.

While environmental collisions are difficult to predict, and may be unavoidable, internal collisions are much easier to manage. The "extreme limit controller" monitors the robot's joint measurements, and shutdowns all power to the actuators if any joint is outside predetermined limits. While this action may not prevent the internal collision from occurring, the likelihood of damage to the joint or actuators is reduced if the robot is inert.

The Emergency Stop Controller operates as a last resort failsafe mechanism; generally the Joint Limit Controller component will already have had a chance to politely prevent intrusion of the robot's joint state into illegal states.

3.2.10.2 Implementation

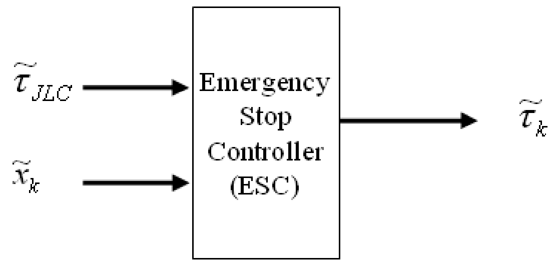


**Figure 3-21 The emergency stop controller monitors the current system state, and prevents any torque being sent to the robot joints if a failsafe condition has been tripped.**

The Emergency Stop Controller simply monitors the current system state variables. If it observes any joint position readings which transgress a set of pre-determined extreme limit values, the Emergency Stop Controller will abort all control of the robot joints. The robot will "crash stop".

Selecting these joint extreme limits is relatively simple matter of selecting joint positions that are well outside the expected operating range of the joint, and yet (usually) within the maximum range of motion for a joint.

Because the emergency shutdown of robot actuators is unsupervised, there is some risk in conducting a "crash stop" procedure. If the robot is busy walking, or running, and "crash stops" it will fall to the ground, and may sustain damage as a result of the fall. We don't want to risk an emergency shutdown in the normal course of operation, so the selected limits should not be close to the robot's normal operating range. Similarly, if the limits are too close to the extreme range of joint motion, then internal collisions may still occur with some force.

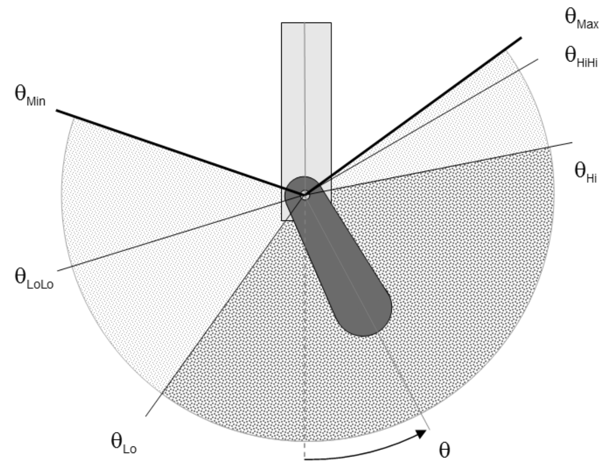


**Figure 3-22 Example hip joint angle limits.**

The diagram Figure 3-22 illustrates a possible configuration of hip joint limit positions.

**Table 3-5 Example joint limits used by the JLC and ESC subsystems.**

| | |
|---|---|
| $\theta_{HIP}$ | Hip joint angle measurement. |
| $\theta_{Min}$ , $\theta_{Max}$ | Physical joint limits. |
| $\theta_{HiHi}$ , $\theta_{LoLo}$ | Limits used by the ESC to initiate a crash stop. |
| $\theta_{Hi}$ , $\theta_{Lo}$ | Limits used by the JTC to initiate a corrective joint torque. |

Sometimes it may be desirable to use the hardware joint limit to assist the robot motion – for example the knee joint may be physically designed not to bend forwards, so that no joint torque needs to be used to hold the knee still while stepping forwards. This configuration allows for efficient walking gaits Pratt [66]. In this situation, it may still desirable to implement an extreme limit, in case the knee joint breaks. Obviously, in this example the shutdown will be too late to save the knee joint. Hopefully the Emergency Stop Controller will prevent further damage to other parts of the system. Figure 3-23 depicts a knee joint space diagram showing such a configuration.

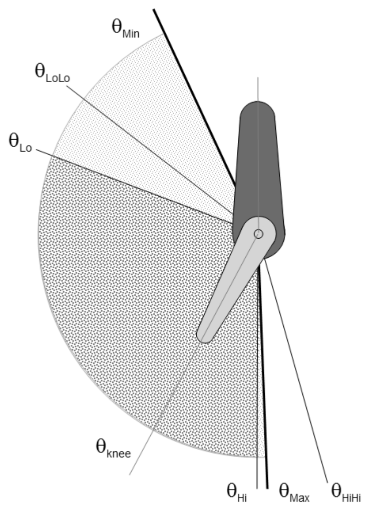


**Figure 3-23 Example knee joint angle limits. The ESC limits are outside the physical range of motion of the knee, and will only be tripped if the knee joint has broken. This may still result in reduced damage to other components of the system.**

# 4 Experiments with Johnny Walker & Jack Daniels

This chapter describes a preliminary experiment conducted using the pre-existing robotic platform “Johnny Walker”. The aim of the experiment was to review work previously done by researchers in the robotics lab, to evaluate a number of sensors for inclusion in later robotic designs and to gain some practical experience with the challenges confronting developers of biped robots. Since the robots employ many of the hardware and software components that would be utilised when developing subsequent robots, these experiments also provided valuable experience in working with the Eyebot robotic platform[1].

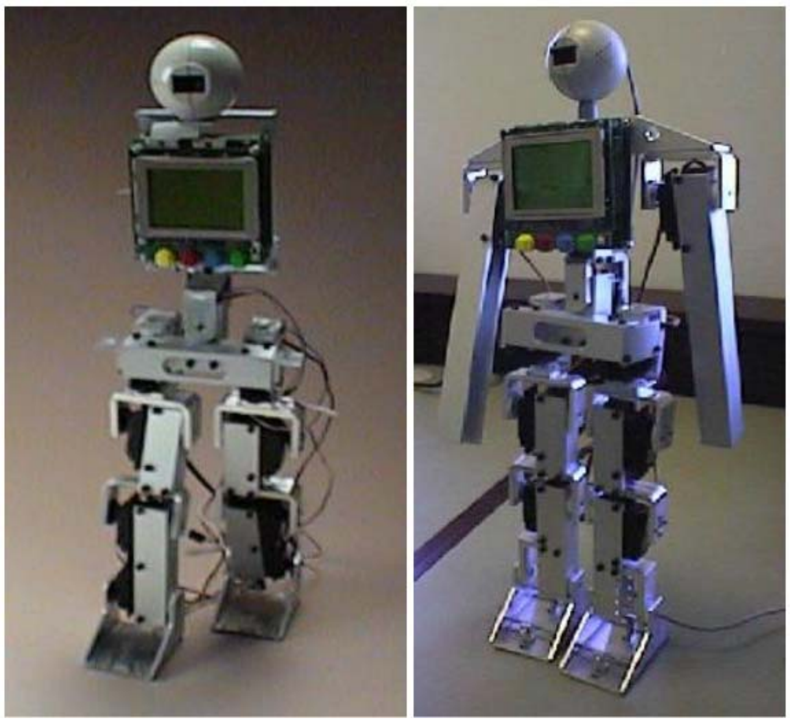


**Figure 4-1 Johnny Walker (left) and Jack Daniels, a pair of rigid link servo actuated experimental robotic platforms, designed for investigations into bipedal robotic control.**

[1] The Eyebot robotic platform is described in Appendix A-1.

## 4.1 Introducing Johnny and Jack

"Johnny Walker" and "Jack Daniels" are two prototype biped robots, developed by honours students Pepper, Nicholls and Ng for the CIIPS research group at the University of Western Australia. While these projects [1-3] were not particularly successful in terms of producing a walking biped robot, they demonstrate a number of valuable lessons. Some of the successes realised by these projects include:

- Demonstrated that is possible to create low-cost biped robots, and control them using the Eyebot controller.
- A significant amount of work was done preparing software libraries to control the robot hardware, and read sensors. While some of this work was specific to the Johnny and Jack[2] platforms, I have been able to reuse some of this software in later robots.
- Some limited walking was achieved; Nicholls [2] reported a maximum successful walking duration of three steps. A short video of his three-step walking gait is available at http://robotics.ee.uwa.edu.au/eyebot/mpg/walk-2leg/johnny.mpg. A qualitative evaluation of walking performance was not provided by either Pepper or Ng [1], [3]. From their thesis discussions, it appears as though the final performance achieved by all three researchers was comparable.

More importantly, all three researchers have drawn similar conclusions regarding the reasons for the failure of the Johnny Walker and Jack Daniels robots to successfully execute a robust walking algorithm:

[2] Robotic platforms "Johnny Walker" and "Jack Daniels" will be referred to as "Johnny" and "Jack" during this discussion.

- An inability to apply a controllable acceleration or velocity to servo driven joints results in “jerky” motion and vibrations that can be detrimental to the stability of any control strategy.
- The inability to detect the positions of servo joints prevents the application of a sophisticated control system. Servo joints are commanded to move to a certain position by the controller, but there is no guarantee that they reach the commanded position.
- Actuators need to be powerful enough to allow the robot to recover from falling – it is not enough to provide actuators capable of simply holding the robot while it is in an equilibrium state.
- When possible, actuator motors should use a different power source to sensor equipment. The power drained by the robotic actuators can have an adverse effect on the accuracy of sensor equipment.

## 4.2 Previous work with Johnny and Jack

### 4.2.1 Walking control systems

Control systems implemented by Nicholls and Ng [2], [3] on Johnny and Jack both involved an off-line gait generation, played back as a time varying trajectory of set point signals sent to the robot’s servos. Pepper developed a robot state display application [1], which showed a graphical representation of the robot as its joints were set to user specified angles. The program was used as a tool for designing a set point trajectory that resulted in a “natural looking” walking gait. The resulting set point “program” was then downloaded onto the robot for evaluation. This system was not a dynamic simulation, so the effectiveness of candidate gait trajectories could not be evaluated in the simulation. Initially the gait trajectory was determined “intuitively” – later a set of trigonometric functions was used to generate a gait that could be modulated by varying frequency of various gait components. Ng [2] based his robot gait on a study of human gait trajectories resulting in a set of sinusoidal joint trajectories. Pepper did not attempt walking, concentrating on sensor interpretation and static balance instead.

### 4.2.2 Sensor interpretation

Nicholls and Pepper concentrated much of their effort attempting to extract useful inclination measurements from the analogue accelerometers mounted on Johnny Walker. Various software and hardware filtering techniques were used to try and reduce noise from the angle estimates. Part of the difficulty these researchers experienced with the analogue accelerometers could be traced to problems with the sensor interaction with the Eyebot's RoBIOS software[3]. In order to read the digital accelerometers mounted on Jack Daniels, Ng wrote a very useful assembly language function for reading pulse-width modulated (PWM) signals into Eyebot registers.

### 4.2.3 PID control

The previous researchers working on Johnny and Jack attempted to implement a form of positional feedback control on the robot to assist its balance. In all three cases, the only useful state feedback available was a noisy hip or torso inclination angle measurement. All three researchers tried to maintain balance by modifying the set point angles of either hip or ankle joints, by an amount determined by a PID controller. The error signal used was the difference between the sensor output during the execution of a trial, and an "expected" sensor output trajectory. Rather than calculating the ideal or "expected" sensor output, this measure was defined experimentally as the "average" output trajectory of a large number of trial runs. Since all of the trials used to determine a reference signal failed, it is not clear how this approach could be expected to result in

[3] A serious error with the AD converter firmware has only recently been discovered and corrected.

a useful reference trajectory. Given the lack of success in producing walking gaits, it follows that none of these attempts were particularly successful.

Even if the researchers could have had complete confidence in the reference sensor signal, attempting to maintain balance by simply manipulating the body's posture is only an effective strategy in static balance scenarios. Force control will be required in order to provide robust balance in a dynamic environment.

## 4.3 What do I want from these robots?

A practical reason for my interest in performing preliminary experiments on the robots Johnny Walker and Jack Daniels was the desire to be able to present some kind of working demo of a biped walking robot. Even though the best walking behaviour I could achieve was very primitive (the behaviour achieved is described in Chapter 4.6), being forced to work within the limitations of the existing robot design highlighted the need to consider the totality of sensors, actuators, physical design and control system design when developing a biped robot.

While a sophisticated walking algorithm is obviously difficult[4] to implement on Johnny Walker and Jack Daniels, these robots do have some advantages in terms of simplicity and low cost. Experimenting on these existing robots was a good way for me to see what components I might be able to take from them to reuse in the design of my own robots.

One thing all researchers using Johnny and Jack have agreed on is the importance of sensor data for control feedback during operation. An important lesson is that appropriate sensors should be identified and evaluated before resources are committed to building a new walking robot. The existing platforms Johnny and Jack allowed me to

[4] I believe such an algorithm is in fact impossible to implement on these platforms.

conduct some preliminary sensor evaluation before I began construction of my new robots. One sense I was particularly interested in using was vision. While Johnny and Jack were both already equipped with a camera, none of the researchers that had experimented with them tried using the camera to assist with balance.

Finally, since much of the software to perform basic control actions on the existing robots had already been written [1-3], working with these existing systems was fast way to gain experience working with the Eyebot/RoBIOS platforms. This experience proved to be valuable when I was developing my own experimental robots, discussed in Chapters 5 and 6.

## 4.4 Construction and materials

### 4.4.1 Physical construction

Johnny and Jack are both rigid link robots, constructed predominantly out of folded aluminium sheets. The robots are of equivalent height and weight, standing around 50 cm in height and weigh around 1.5 kg. The robots differ only slightly in the choice of actuated joints and number of "degrees-of-freedom". Construction details for Johnny Walker are provided by Nicholls [2], while details for Jack Daniels are given by Ng [3].

**Table 4-1 Construction summary for the robots Johnny Walker and Jack Daniels**

| **Robot** | Johnny | Jack |
|---|---|---|
| **Weight** | 1.78 kg | 1.8 kg |
| **Height** | 50 cm | 48 cm |
| **Degrees of freedom** | 9 | 11 |
| **Actuated joints** | Left and right ankle<br><br>Left and right knee | Left and right knee<br><br>Left and right hip (bend) |

| | | |
|---|---|---|
| | Left and right hip (bend)<br><br>Left and right hip (twist)<br><br>Torso | Left and right hip (twist)<br><br>Torso<br><br>Left and right shoulder |
| **Passive joints** | None | Left and right ankle |

### 4.4.2 Controller

Both robots use the Eyebot controller for mobile robots, developed by Joker Robotics. For information about the Eyebot controller, see Appendix A-1.

### 4.4.3 Actuators

Both Johnny Walker and Jack Daniels utilise servo motors for actuation. These motors provide a form of positional control, where the servo attempts to move the current position to a target angle specified by the controller. No feedback is provided by the servo to the controller to allow a high level control system to verify whether the set point position has been successfully reached.

Positional control is achieved by the servos through the use of an internal PD controller that moves the servo to a position indicated by a pulse width modulated (PWM) input signal. The proportional and differential parameters of the servo's PD controller are inaccessible to an external controller, which means the response of the servo cannot be modified by the robots. Since most servos are designed to operate light loads, in a robotics environment the loading experienced by the servo often means its performance is sub-optimal.

Additionally, servos are usually designed to operate with a "dead band" zone around the set point position. When the servo position is within the "dead band" zone, no torque will be applied to the servo shaft. This is to prevent unnecessary servo actuation, conserving power usage. Unfortunately it also means the servo cannot achieve accuracy of more than about 1 degree.

**Table 4-2 Joint actuators used on Johnny Walker and Jack Daniels**

| **Robot** | Johnny | Jack |
|---|---|---|
| **Servo** | Futaba S9402 Servo | Hitech Servo HS-300 |

### 4.4.4 Sensors

The robots Johnny and Jack are equipped with a similar array of sensors, consisting of linear accelerometers, optical digital switches and a camera.

**Table 4-3 Sensors installed on Johnny Walker and Jack Daniels**

| **Robot** | Johnny | Jack |
|---|---|---|
| **Accelerometer** | 2 * ADXL05 1-axis analogue sensor | 1 * ADXL202 2-axis digital sensor |
| **Digital switches** | Honeywell optoelectronic switches (HOA2498) | Honeywell optoelectronic switches (HOA2498) |
| **Camera** | Connectix QuickCAM colour camera | Connectix QuickCAM B/W camera |

#### 4.4.4.1 Accelerometers

The accelerometers installed on the robots Johnny and Jack were originally intended to measure the orientation (pitch and roll) of the robot link to which the sensor has been attached. The analogue sensor on Johnny Walker is attached to the torso, while the digital sensor on Jack Daniels is attached to the robot's hip. Each robot's accelerometers are installed in such a way as to give readings of acceleration vectors in the horizontal plane of the robot, measuring acceleration in the x-z plane (also known as the transverse plane)[5]. Figure 4-2 illustrates the orientation of the x, y and z axis with respect to the robot links. The x-y plane is aligned with the robot's sagittal plane, while the z-y plane is the frontal plane.

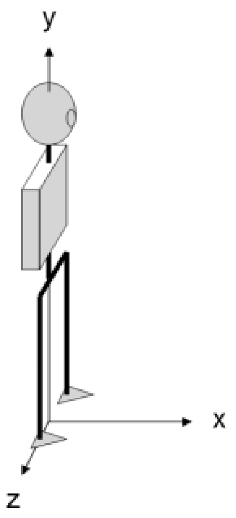


**Figure 4-2 Axis orientation for Johnny Walker and Jack Daniels. Accelerometers installed on each robot measure an acceleration vector in the x-z plane of the robot, also known as the "transverse" plane.**

The analogue and digital accelerometers used are very noisy, a problem compounded by the unavoidable "servo jitter" experienced by the robot as the servos continuously work to maintain their positions. Much of the early work with these robots was devoted to finding appropriate filters to make the accelerometers usable [1], [2].

As an additional complication, my preliminary experiments with the analogue sensors installed on Johnny confirmed Pepper's observation that there were some problems relating to the sensor interaction with the analogue input ports on the Eyebot platform. Namely that:

[5] In the frames of reference used in [1-3], the accelerometers are aligned along the x and y directions. The coordinate axes have been relabelled in this discussion to maintain consistency with the remainder of the thesis.

- Occasional AD converter readings for particular channels would in fact read the incorrect channel's value.
- If the AD converter was not continuously being read, then the channel appears to sometimes "hold" the last value.

These behaviours were unpredictable, and considerably reduced my confidence in using the analogue accelerometers in experiments. The problems did not appear to occur if only a single analogue sensor is being read, as was the case during my experiments with an inclinometer sensor in Chapter 5.7. I understand that since these experiments were conducted a fault with the Eyebot's RoBIOS operating system has been identified, which may resolve these problems.

Finally, experiments reading the accelerometer while the servos were under high load showed highly variable results. I believe this is partly due to power drained by the actuators adversely changing the reference voltage used by the accelerometer sensor.

#### 4.4.4.2 Cameras

The robots are both equipped with a QuickCAM Connectix digital camera. These cameras are able to take colour or black and white images at a resolution of 80 x 60 pixels. My preliminary investigations into using the camera as a sensor for assisting balancing robots were not particularly encouraging, and have not been included in this thesis.

#### 4.4.4.3 Optical Switches

The robots were originally designed with four push-button switches mounted at the corners of each foot. By the time I became involved in the project, the push-button switches had been replaced by infra-red optical switches. These are calibrated to detect contact between the robot's feet and the ground. Unfortunately, depending on the reflective properties of the surface the robot is walking on, the switches can give false positive readings when they are close to the ground. Occasionally during preliminary experiments, the robot foot could be as much as 1-2 cm above the ground while the sensor signals a positive ground contact.

## 4.5 Experimental overview

A single preliminary experiment conducted using the Johnny Walker robotic platform is described in this chapter:

i. Walking demo using Johnny Walker

Given the robot's design limitations, what is the best kind of walking demo that can be obtained? The aim of this experiment was to try and develop a demo walking gait for one of the existing robots. The demo was to be completely autonomous, and not rely on any external power sources. The most critical lesson learned from this exercise was the importance of selecting appropriate sensors and actuators during the design of a robot and its control system.

## 4.6 Walking demo using Johnny Walker

### 4.6.1 Introduction

This development of a walking demo for Johnny Walker was conducted in collaboration with Martin Wicke, another student interested in bipedal robotics. The objective of this experiment was to implement the best walking algorithm possible on the robot Johnny Walker, given its design limitations. By determining what walking behaviour is possible with a robot like Johnny Walker, this experiment identifies challenges that need to be considered when developing a walking robot. A more immediately obvious benefit was the ability to demonstrate a walking gait on a real robot. Even if the resulting walking pattern is not sophisticated, it is always nice to be able to show something concrete to visitors to the laboratory!

### 4.6.2 Materials

Johnny Walker was selected as the robot used to attempt a walking demo. This is because unlike his "brother" robot Jack Daniels, Johnny has fully actuated ankles, and was able to stand without needing active balance. Johnny Walker is a nine degree-of-freedom biped robot, powered by servo actuators and equipped with a camera, a pair of analogue accelerometers and an array of optical switches in its feet. The construction of this robot is summarised in 4.4, and described in more detail by Nicholls [2].

The ankle, knee, hip and torso actuators installed on Johnny Walker are servo motors. At any time, the servos can be commanded to a new set point angle, using the RoBIOS function `SERVOSet(Position)`. How the servo attempts to reach this set point is not controllable by the Eyebot control system. Additionally, no information about the servo position can be determined by the Eyebot control system. If the servo is unable to reach the commanded position for any reason, the controller has no way of determining this.

It is worthwhile reviewing the sensors installed on Johnny Walker, as no information about the robot state can be used by the control system unless it has first been recorded through the use of the installed sensors. Sensors installed on Johnny Walker include:

- Analogue accelerometers – read by Johnny as a pair of analogue input, using RoBIOS function `OSGetAD(channel)`. Data obtained through these sensors was not reliable enough for the sensor to be used as an inclination sensor.

Instead I used the accelerometers as vibration sensors, which is the purpose for which they are best suited[6].

- Optical switches – due to their lack of sensitivity, I did not use these sensors in the demo walking control system.
- Camera – the camera installed on Johnny Walker was not used during this experiment.

As a consequence of the sensors installed, the following state information is available to a control system implemented on Johnny Walker:

- Measure of vibration intensity.
- Bitmap image of the view from the robot's camera.

Since no work has been done on utilising the camera, the only information Johnny can detect is how badly he is shaking!

### 4.6.3 Methods

The sensors and actuators installed on a robot directly influence the kind of control system that can be implemented. In the case of Johnny Walker, none of the more sophisticated control systems discussed in Chapter 2 can be attempted. For example:

- Can't do ZMP based walking – need force measurements of ground contact. Contact switches are insufficient for calculating the centre of pressure.
- Can't do Inverted pendulum control – even if the accelerometers used were able to give timely and accurate measures of robot inclination, this approach requires the ability to apply variable force to base of the robot "pendulum" – with the

[6] One of the primary applications for the ADXL05 accelerometers used on Johnny Walker is to detect vibrations induced by car collisions, for the deployment of protective airbag devices.

servo driven actuators installed on Johnny Walker we have no control over this force.

- Not even static walking is possible, as we cannot read the servo positions – and hence calculate the position of the robot's centre of mass.

This means the only viable control system is a kind of experimentally developed "quasi-dynamic" walking, where a joint trajectory is created using trial and error. The robot has to trust that playback of the pre-determined trajectory will result in a repeatable walking behaviour. The resulting control system is "quasi-dynamic" only by virtue of the fact that at some stage during the joint trajectory playback, the robot's centre of mass is likely to be outside the supporting polygon[7]. This is more by accident than conscious design, as there is nothing sophisticated about this kind of experimentally determined gait. It is certainly not any "better" than a static walking approach.

Previously attempted control systems implemented on Johnny Walker were also examples of this kind of heuristic technique [2]. Nicholls' basic walking algorithm consisted of a cyclical trajectory of joint set point positions, which would move the robot's limbs in a continuous and smooth walking pattern. Unfortunately the control system developed by Nicholls was only able to take a maximum of three steps before falling. Part of the explanation for this failure is that as the robot prepares to take each step, the commands sent to the servos remain the same, regardless of the robot's actual position. Since we cannot read the positions of the robot's joints, an assumption has to be made that the joints positions at the start of each step will be identical. This is almost never the case, and when the same control signals are repeated, errors in joint positions are magnified each step until the robot falls.

[7] See Chapter 2; all parts of the robot in contact with the ground delimit the outside border of the supporting polygon.

In an effort to mitigate this problem of increasing uncertainty regarding the position of the robot's limbs, my approach was to have the robot take a single step, wait for the robot joints to settle into their new positions, and only then attempt to take another step. The approach is quite a bit simpler than attempting to make smooth, continuous steps.

An overview of the walking algorithm used in this experiment:

1. Initial position, with right leg back. Hips are angled with the leading leg's hip ahead of the trailing leg's hip.
2. Quickly lift the rear leg forward:
    a. Lean the torso forward by bending forward at both of the hip joints.
    b. Swing the torso away from the rear leg, so the robot weight shifts dynamically onto the forward leg.
    c. Lift rear leg, bringing it forward and straighten support leg.
    d. Begin to swivel hips, helping to bring the rear leg forward – continue to straighten support leg as it takes the strain of the robot's full weight.
    e. Continue to swivel hips, as the raised leg now moves ahead of the support leg.
    f. Finish swivelling hips.
    g. Lower raised leg, the robot joints now are a mirror image of the positions at the beginning of step 2.
3. Wait for balance, by monitoring the digital accelerometer until large vibrations cease.
4. Repeat steps 2 & 3, swinging forward alternate legs.

The gait was experimentally synthesised, following the outline above, until a trajectory of joint angles was found that resulted in a satisfactory walking performance. A number of considerations influencing the robot's posture and movement arose during the experimental process of developing a repeatable walking pattern.

The walking demo was required to rely solely on on-board power sources – namely a backpack of six AA sized rechargeable batteries. It proved to be frustratingly difficult to balance the robot while it was wearing its battery pack in the designed configuration. To improve the robot's balance the backpack was broken into two groups of three batteries, and redistributed about the robot. Three batteries remained on its back, while the remaining three were slung beneath the torso. This redistribution significantly improved the robots stability.

Previous attempts at creating walking behaviours with Johnny Walker attempted to develop an anthropomorphic gait. Apart from aesthetics, there is no reason to constrain the joint trajectory to make it walk like a human – in fact Johnny was more stable if it bent its knees backwards, like a bird or dinosaur. This is the configuration used in the walking demo.

The robot can stand quite firmly when both feet are down. It is not able to stand on one leg for very long, before the weight of the robot causes joint angles to shift in unpredictable ways. Since the walking algorithm relies on the robot joints being close to a known position before each step is taken, we cannot afford to allow unpredictable changes to the robot state. As a consequence, the robot needed to limit the amount of time it spends in the "single support" phase of the gait. When walking forwards, it needs to pick up and replace the rear foot as quickly as possible.

The torso can play an important role in assisting bipedal walking. In Johnny Walker, the role of torso is to shift weight off the rear leg so that it can be brought forward without causing the robot to fall. While previous researchers [1-3] relied on the static weight of the torso to achieve this end, this experiment showed it can be far more effective to use dynamic properties of the torso movement. Vigorously swinging the torso in the direction of the supporting leg causes the weight of the robot to shift off the leg being lifted when the torso reaches the limit of its movement. This effect is far more pronounced than torque contributed by the robot's weight.

### 4.6.4 Results

Using this method, a walking gait for Johnny Walker was developed that could successfully take a series of steps. Figure 4-3 shows a sequence of images taken as Johnny Walker executes a step during a walking demo. The full video can be downloaded from: http://www.ee.uwa.edu.au/~suthe-aj/thesis/johnny/johnny.mpg.

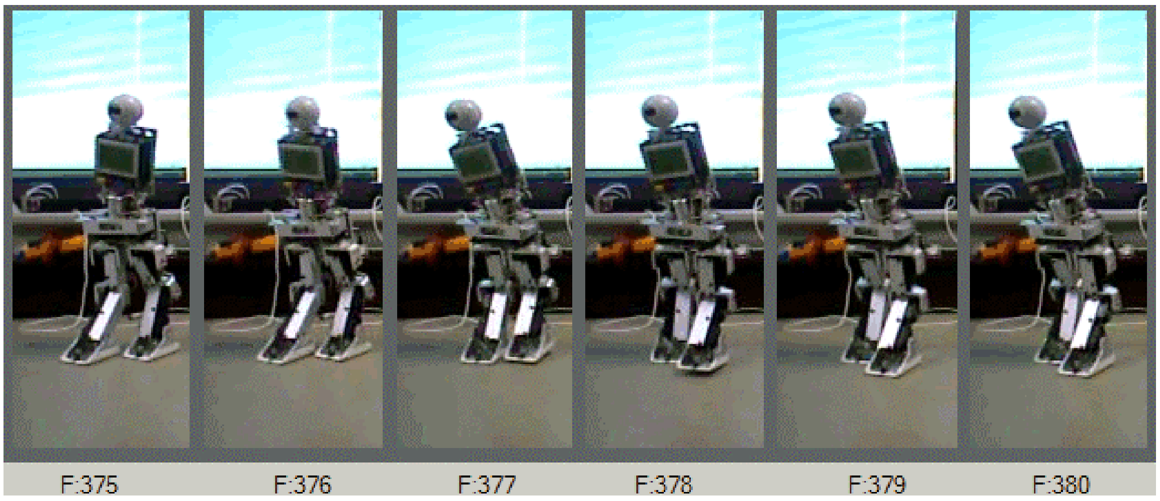


**Figure 4-3 Johnny Walker steps out. The step begins with the torso and hips swinging the weight of the robot over the leading foot, and ends when the rear foot is replaced ahead of the robot. The robot must then wait until it is steady before taking the next step.**

Despite the care taken to create a repeatable walk, eventually the robot always falls. In the video referenced above, the robot fell once after only two steps, then (once restarted) walks for twenty steps before falling again. In a typical trial, the robot walked around 15-20 steps before falling. The most common reason for a fall is that the robot state has drifted too far from the assumed "at start" positions before the robot attempts the step on which it falls.

Another cause for walking failures is the changing behaviour of the robot servos as its on-board power supply is drained through use. Servo actuation causes a significant power drain, and their performance becomes noticeably degraded after only a few minutes of operation. Since the robot gait relies on repeatability of actuator motion, any the change to the servo response causes the robot gait to fail.

### 4.6.5 Discussion

We succeeded in the stated goal of this experiment, which was to produce a walking demo using the existing robot, Johnny Walker. Through this experience, I gained some important experience about some of the challenges that need to be considered when developing a walking robot. The most important lesson gained is that a robot should be designed and built with a specific control system already in mind. The choice of control system is fundamental to the selection of actuation and sensing instruments, and these things should all be designed in parallel.

The experiment has shown that an appropriate selection of sensor devices installed on a robot is critical for supporting any proposed control strategy. An autonomous robot can infer detail about its orientation within the environment from no other source than the sensors it has been equipped with. The lack of suitable sensors on Johnny Walker forced us to utilize a very primitive control system that was not able to adapt to any unexpected situations. The minimum state variables that would need to be measured for some of the control systems discussed in Chapter 2 include:

**Table 4-4 Minimum state variables required for a selection of control system categories.**

| **Static walking** | **ZMP based** | **Inverted pendulum based** |
|---|---|---|
| Joint angles(*) | Joint angles(*) | Joint angles |
| | Foot pressure sensors | Inclination sensors |

(*) Joint angle measurements may not be required if it can be safely assumed that the actuator's positional control will always be satisfied.

- Joint angles. Perhaps the biggest drawback of the servo driven systems is the inability to know if the servo has actually moved to the position to which it was commanded. Potentiometers or optical encoders would help overcome this problem.
- Foot pressure sensors can be used to measure centre of pressure for ZMP based walking systems.
- Robot inclination. The accelerometers installed on Johnny Walker were intended to be used to detect inclination. The particular arrangement of sensors used on Johnny Walker has proven to be inadequate for this purpose.

Another lesson learned is the importance to build confidence in the components being used to operate the biped robot. If a complex robot falls, it is always difficult to isolate the cause of the problem. This is especially true if the reason for failure could be as basic as unsuitable system state measurements. Part of the design process should be an evaluation of sensors in a controlled environment.

Sensors installed on the robot which require a constant reference voltage level may need to be provided with a separate power source to high energy consuming components such as actuator motors. The susceptibility of sensor measurements to interference from other component's power consumption is something that should be experimentally determined before committing to applying a particular sensor/actuation plan.

Finally, it has become clear through the process of conducting this experiment that actuator choice is another important issue. The servo based actuators used in Johnny

Walker and Jack Daniels, while inexpensive and simple to use, exhibit two critical deficiencies when used for robot actuation:

- Servos lack positional feedback.
- Servo lacks force control.

While the lack of positional feedback could be compensated for through the use of additional sensors, such as potentiometers or optical encoders, the inability to apply variable force control to the robot joints rules out the use of inverted pendulum based control systems. Since such systems are the basis of the Torso Driven Walking control system, servo actuation was not employed on the robots discussed in the remainder of this thesis.

Note that by physically modifying the servo motors, it would be possible to implement some kind of force control. However in this case, the actuators would no longer be servos.

## 4.7 Johnny and Jack's legacy

### 4.7.1 Lessons learned from Johnny and Jack

Through the course of conducting the preliminary experiments with the robots Johnny Walker and Jack Daniels, a number of lessons were learned about the challenges confronting researchers into bipedal walking robots. In addition, valuable experience was gained in working with the Eyebot robotic platform, on which the balancing robots described in Chapters 5 and 6 are built.

Some of the specific objectives achieved, and lessons learned were:

- The development of a successful (if primitive) walking demonstration with Johnny Walker.
- Demonstrated the importance of selecting robotic sensors to suit planned control systems.
- Demonstrated the importance of selecting appropriate actuators to suit planned control systems.
- Discounted using the analogue acceleration sensor for inclination detection in future robots.

At this stage I believe I have gained all that I can from the robots Johnny and Jack, and I don't plan to conduct any more work with them.

### 4.7.2 Related experiments

Some experiments continue to be conducted by researchers with Johnny and Jack-like robots [27], [73], [74]. The robot "Andy Droid", while much lighter, has an almost identical construction to Johnny and Jack – including the use of servos for joint actuation. An important difference is that the robot "Andy" has been constructed with the addition of strain gauges in the feet. These sensors can be used to detect the centre of pressure in each foot, which means the robot may well be suited to ZMP[8] based walking algorithms. As ZMP based walking systems do not necessarily require joint force control, the positional control supplied by the servo motors may be satisfactory.

This class of ZMP "dynamic" walking is not the focus of this thesis. The robots described in Chapters 5 and 6 are designed to be fully dynamic balancers, utilising the

[8] ZMP – Zero Moment Point control systems are discussed in Chapter 2.3.1, "Zero Moment Point". For stability, these systems require that the measured "centre of pressure" does not approach the edge of the robot's feet.

inverted pendulum based “Torso Driven Walking” control system introduced in Chapter 3.

## 4.8 Experimental review

When developing a biped robot, it is clear that all components of the system must be designed to work together. Design of a control system is just as important as planning the physical construction of the robot and the selection of sensors and actuators. In fact the type of control system to be implemented will directly impact the kinds of sensors and actuators installed on the robot. The two key messages taken away from the experiments in this chapter are:

- Robot state sensors must be chosen to suit the planned control system.
- Robot joint actuators must be chosen to suit the planned control system.

The “Torso Driven Walking” control system, introduced in Chapter 3, requires measurements of the following system state information:

1. Torso inclination
2. Joint angles
3. Ground contact

Of these measurements, torso inclination is the most critical, and the most difficult to obtain. There are a number of sensors that might provide usable inclination measurements, including the accelerometers used in this chapter. Choosing a sensor involves a trade off between performance measures of accuracy, sampling rate, resource requirements, cost and ease of use.

In Chapter 5, a simple balancing robot designed to experimentally evaluate a variety of sensors for the detection of torso inclination is introduced.

# 5 Experiments with the Ballybot

This chapter describes a series of experiments that were conducted while developing a new experimental robot, the Ballybot. The robot was designed as an experimental platform for investigating issues with the control of balancing robots, free from the additional complexity required in order to control a legged robot. The two primary issues for which the robot was intended were investigations into control systems for dynamically balancing robots (in particular the Torso Driven Walking control system), and the determination of an appropriate sensor configuration for determining inclination angle that could be used in a more complex, balancing legged robot.

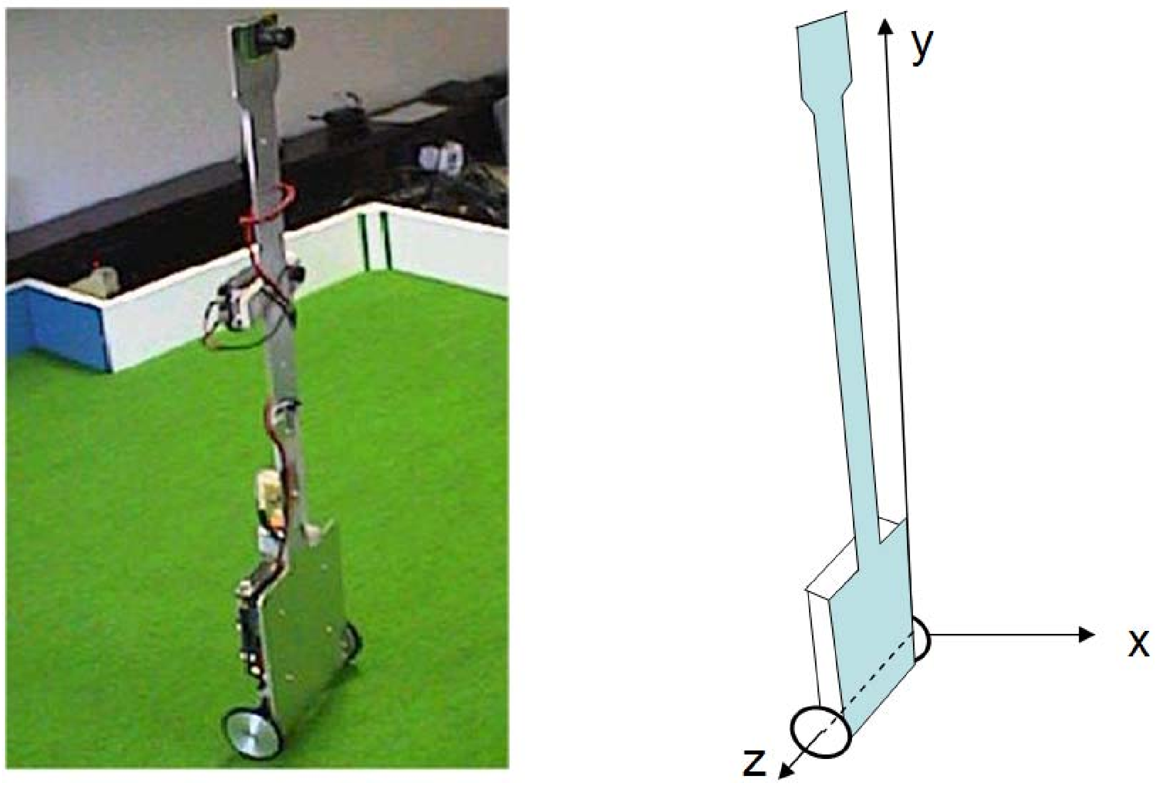


**Figure 5-1 Ballybot, a simple autonomous inverted pendulum robot. The robot has been designed as a tool for experimentally evaluating control systems and sensor configurations for balancing robots.**

## 5.1 Introducing Ballybot

The Ballybot is an autonomous, single degree of freedom mobile robot, based on the model of an "inverted pendulum". It consists of a rigid link, at the base of which are mounted a pair of wheels. The two wheels are mounted in parallel, sharing a common axis. As a result, the robot can rotate freely in the sagittal (x-y) plane, but is constrained from rotating in the frontal (y-z) plane. This design effectively makes the Ballybot a planar robot, simplifying the control system, sensor configuration and dynamic simulation. Lessons learned from experiments with planar robots can easily be extended to three dimensional systems.

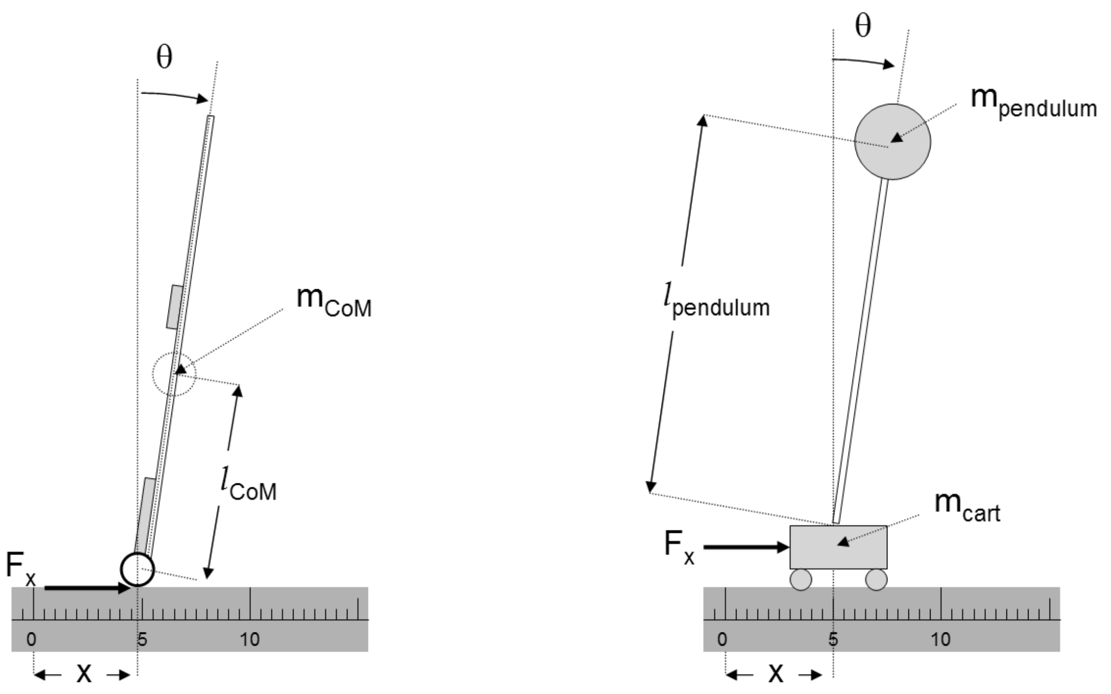


**Figure 5-2 The Ballybot's system dynamics are very similar to that of the classic "Inverted Pendulum" problem. The Ballybot generates its own balancing force $F_x$ by applying torque to the wheel motors.**

The Ballybot is an example of a "purely dynamic" balancing robot, meaning that at no time will the robot be statically balanced. In order to maintain balance, the robot needs to apply a variable horizontal force to its base. In traditional "inverted pendulum" architectures, an external balancing force is applied directly to a cart on which the pendulum is fastened. The Ballybot generates its own balancing horizontal force by inducing a ground reaction force through the application of motor torque to the robot's wheels.

The robot has been designed to allow easy access to the controller's I/O ports, to make changing sensor configurations as simple as possible. Each wheel is independently driven, to allow the robot to be driven straight, as well as change direction while balancing. The resulting experimental robot is robust, simple to build and maintain, and inexpensive.

## 5.2 Why build a Ballybot?

### 5.2.1 Ballybot for sensor evaluation

When a complex robotic system, such as a biped robot, fails to operate correctly it is usually difficult to isolate the cause (or causes) of the failure. One of the theories suggested for the failure to walk of the servo driven biped robots, Johnny Walker and Jack Daniels (Chapter 4), was the lack of a suitable sensor capable of timely and accurate inclination detection [1-3]. If we are to accept this analysis, the question that has to be asked is "how fast/accurate do these sensors need to be?" The Ballybot robot was developed to help answer this question experimentally.

The Ballybot robot design allows researchers to focus on the problem of sensor selection and interpretation for balancing robots, without having to worry about any additional issues encountered when working with more complex robots. The robot has a single degree of freedom and is controlled by a single input variable, the DC motor applied voltage. This simple design results in behaviour that can be described by a relatively simple dynamic model, which has been used to create a software simulation. Both the dynamic model and software simulation are described in Appendix B.

While the design has been kept as simple as possible, the robot still requires a fully dynamic, balancing control system, to prevent it from falling over. The Ballybot needs accurate and timely estimates of angular orientation in order to maintain balance. Since the robot is essentially balancing on a "point", it will need more accurate and faster angular estimates than most legged balancing robots, which can use the foot and ankle to contribute to balance in meaningful ways.

In addition to testing inclination sensors, experiments using the Ballybot allowed the interaction between candidate sensors and the EyeCon controller to be tested in a "real-world" operating environment. In some cases problems with a sensor may be exacerbated, or even caused, by the interaction between the sensor and controller. Examples of this were the problems experienced by the robot Johnny Walker, using an analogue accelerometer with the EyeCon controller[9].

The combination of a group of sensors, and the software algorithm that interprets their signals, is considered a sensor "module" that can be experimentally evaluated on the Ballybot platform. If a sensor module can be found that is able to maintain balance on the Ballybot, then the same module should be suitable for more complicated robots, such as the bipedal Legbot discussed in Chapter 6. If this more complicated robot falls while using a sensor module proven on the Ballybot platform, the problem causing the robot failure is unlikely to be with the sensors used, and other causes should first be investigated.

[9] Experiments discussed in Chapter 4.6, "Walking demo using Johnny Walker".

### 5.2.2 Ballybot for control system investigations

As well as providing a test bed for evaluating sensors, the Ballybot is an ideal platform for experimenting with balancing control systems. Inverted pendulum models have already been used as the basis of a number of bipedal walking strategies, including [7]-[10]. In the Torso Driven Walking control system[10] the balancing problem experienced by the Ballybot robot is very similar to that encountered by the "Torso Attitude Controller" sub-system. While the TAC controller tries to keep the robot torso upright by applying torque to the robot's hip joint, the Ballybot stays upright by applying torque to the robot's wheels.

One of the big divisions in classification of control system architectures is the distinction between adaptive and conventional control systems[11]. Essentially, an "adaptive" control system learns for itself how to perform its function, while a "conventional" system needs to be told what to do. Adaptive control systems are attractive, because they can be applied to systems that may not be completely understood.

In this thesis, the Ballybot robot has been used to investigate the possibility of applying an adaptive approach to a simple balancing control system. The hope was if a simple robot like the Ballybot could be successfully balanced using a purely adaptive control system, then it may be feasible to attempt the same thing with a more complicated legged robot.

[10] The Torso Driven Walking control system is discussed in Chapter 3.
[11] The adaptive and classical control system classifications are discussed in Chapter 2 "Control Systems for Walking Robots".

## 5.3 Materials

### 5.3.1 Physical construction

The Ballybot's frame consists of a single sheet of aluminium, cut to shape with strategically drilled holes used to mount the EyeCon controller, DC motors, batteries and various sensors. The design includes a mount for a digital camera, placed in the head of the robot. This sensor was not used during the experiments in this chapter, as my preliminary investigations in computer vision discouraged me from using the camera as a sensor for active balance. The design of the Ballybot frame is shown in Figure 5-3:

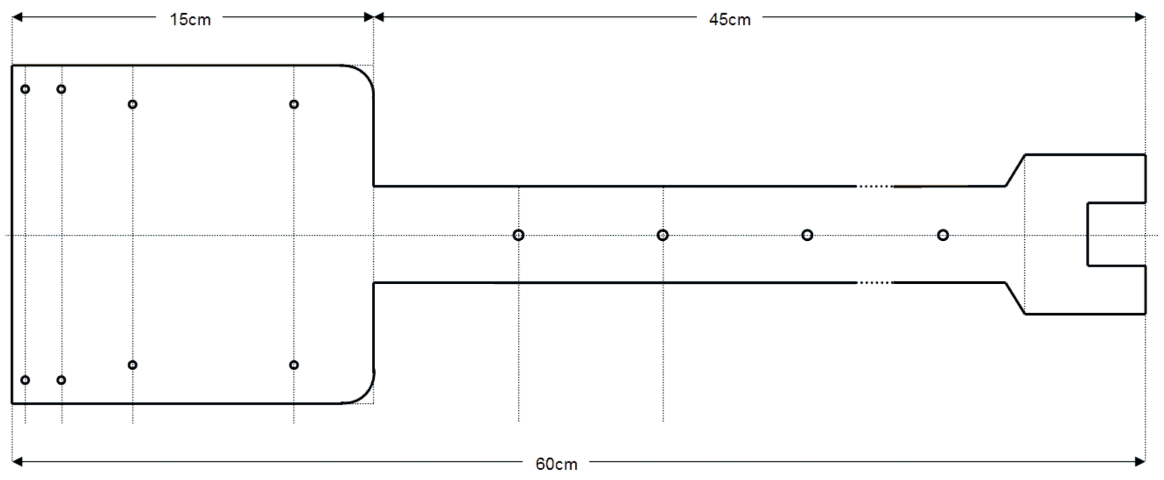


**Figure 5-3 Ballybot frame is a single sheet of aluminium, with holes drilled to mount the controller, batteries and various sensors and actuators.**

During the course of these experiments it was found that while the motors were operating, large vibrations propagated through the Ballybot's aluminium frame. This is because the metal frame was too flexible in the sagittal (x-y) plane. The addition of a balsa wood "backbone" stiffened the frame sufficiently to prevent this oscillation without adding significant weight.

### 5.3.2 Controller

The Ballybot robot uses the EyeCon controller for mobile robots, developed by Joker Robotics [37]. For information about the EyeCon controller, see Appendix A-1, “Eyebot robotic platform”.

The robot has been designed to allow easy access to the controller’s IO ports, to simplify the swapping of sensor devices. As a consequence, the controller itself is not well protected against collisions with the ground, in case the robot falls. Care must be taken when conducting experiments to prevent damage to the controller.

### 5.3.3 Actuators

The robot’s wheels are driven by a pair of Faulhaber series 2230/006S DC Motors, with a 9:1 gearing ratio. Each motor shaft is directly connected to one of the Ballybot’s wheels. These motors have encapsulated encoders, and can be used to measure the displacement and velocity of the robot base. Most importantly, by using a DC motor it is possible to generate a variable torque. This is a requirement for most inverted pendulum based control systems. Further information about DC motors is provided in Appendix A-2.2, “DC Motors”.

### 5.3.4 Sensors

One of the primary Ballybot design features was the ability to quickly change the robot’s sensor configuration. Some of the sensors used in experiments with the Ballybot included:

i. Gyroscope

The solid state gyroscope used in these experiments measures angular velocity. This measurement can be integrated over time to provide a measure of the angular displacement experienced by the robot. If the robot's initial position is known, the angular displacement can be used to determine the robot's inclination. Experiment 5.7 "Sensors for balance" describes the process by which the robot tilt angle is estimated using the gyroscope sensor.

ii. Angular inclinometer

An inclinometer is used to measure tilt angles. Although similar in design to a linear accelerometer, the output signal is a single angle estimate, rather than acceleration vector components.

iii. Shaft encoders

Optical shaft encoders encapsulated in the DC motor housing provide easily accessible measurements of motor shaft angle. In the Ballybot robot, assuming the robot wheels do not slide or bounce over the ground, the shaft's angular displacement is proportional to the distance the robot has travelled. If the robot's initial position is known, this displacement can be used to determine the robot's position within the workspace. This technique for robot navigation is often referred to as "dead reckoning" [27], [38].

### 5.3.5 Software simulation

A software simulation can be an invaluable tool in the development of control systems for robotic systems. Simulation allows us to design and test control systems in a wholly controllable artificial environment, before we expose them to the challenge of real world control. Some specific examples of advantages afforded through the use of software simulations are:

- Control systems can be investigated without risking damage to the physical robot.
- Complications such as sensor noise and friction can be controlled, allowing the system designer to concentrate on one problem at a time.
- Trials can be performed many times faster than may be possible on a physical system. This is especially important when investigating adaptive approaches to control.
- It is easy to distribute demonstrations of control system designs.
- Developing a simulation requires a good understanding of the dynamics of the system you are modelling. Knowledge which is also valuable when working with the physical robot.

I have developed a software simulation of the Ballybot robot system, implemented as an ActiveX control, which can be viewed on a web page using Microsoft's web browser, Internet Explorer. In an attempt to clarify this discussion, details about the development of the Ballybot simulation system, have been omitted from the main body of this document. These details include the software design and dynamic model derivation, and have been provided in Appendix B.

## 5.4 Experimental Overview

A series of experiments were conducted while developing this robot, including:

Chapter 5.5 - Torso Attitude Controller for the Ballybot

The Torso Attitude Controller is the most critical component of the Torso Driven Walking control system (Chapter 3), simultaneously responsible for maintaining balance of the robot's torso, and the horizontal displacement of the robot. In this experiment, the Ballybot simulation is used to test the proposed "Torso Attitude Controller" sub-system.

## Chapter 5.6 - Force from voltage

Regardless of whether an inverted pendulum control system is adaptive or conventional, it still needs to be able to regulate the application of force to the base of the Ballybot robot. The Ballybot can control the output torque of its motors by adjusting the motor input voltage. Unfortunately the relationship between applied voltage and resultant force is not linear. In this experiment a mathematical model is developed that the Ballybot's control system uses to determine the required input voltage needed to produce a specific ground reaction force requested by the control system.

## Chapter 5.7 - Sensors for balance

One of the more difficult problems facing developers of bipedal walking robots is the selection of appropriate sensors for maintaining balance. In this experiment, the Ballybot robot is used to evaluate proposed robotic tilt sensors and sensor data processing algorithms. Two inexpensive sensors in particular are examined: a piezo-electric gyroscope designed for remote control helicopters, and an inclinometer. Results demonstrate that the Torso Driven Walking control system is able to maintain balance using the gyroscope, even though the sensor experiences a significant amount of bias error. While balance is maintained, it is not possible to simultaneously control the robot's horizontal position unless both inclinometer and gyroscope are used co-operatively to control the robot.

## Chapter 5.8 - Comparison with a “first-principles” adaptive control system

Using the Ballybot software simulation[12], an artificial neural network based adaptive control system was developed to balance the Ballybot robot. The resulting control system is an example of a “first-principles” adaptive control system, such as those described in Chapter 2.3.6. The development of this alternative control system allowed the relative advantages of such an approach to be compared with that of the Torso Attitude Controller sub-system of the Torso Driven Walking control system. While the adaptive approach successfully learns to balance the robot, with no preconceived or pre-programmed notion of how this should be achieved, significant disadvantages were encountered. Most significantly, the control system requires a very large number of training cycles and does not always converge to a satisfactory solution.

[12] Described in Appendix B.

## 5.5 Torso Attitude Controller for the Ballybot

**Abstract:** The Torso Attitude Controller is the most critical component of the Torso Driven Walking control system (Chapter 3), simultaneously responsible for maintaining balance of the robot's torso, and the horizontal displacement of the robot. In this experiment, the Ballybot simulation is used to test the proposed "Torso Attitude Controller" sub-system.

### 5.5.1 Introduction

The "Torso Attitude Controller" is the central component of the "Torso Driven Walking" control system for biped robots, introduced in Chapter 3. The Torso Attitude Controller attempts to balance the robot's torso by applying a variable torque to the hip joint of a biped robot's supporting leg. All other components of the Torso Driven Walking system are indirectly controlled by the Torso Attitude Controller, so the performance of the Torso Attitude Controller is critical to the success of the walking algorithm.

The Ballybot robot was designed to be an experimental platform for developing and testing the Torso Attitude Controller in isolation from the rest of the biped control system. This is an example of the modular approach to control system synthesis that was followed during these investigations. The Ballybot and a biped robot's torso share very similar system models. Consequently, a control system that can balance the Ballybot should also be able to fulfil the function of "Torso Attitude Controller" in the much more complicated Torso Driven Walking biped robot control system.

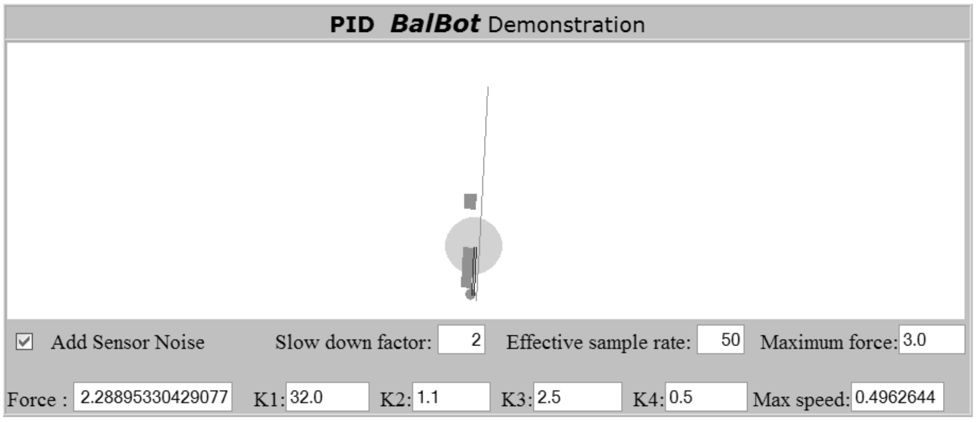
### 5.5.2 Materials

#### i. Ballybot simulator

This experiment makes use of the Ballybot software simulator, described in Appendix B. The URL of the demo used during this experiment is:

http://www.ee.uwa.edu.au/~suthe-aj/thesis/ballybot/balbotdemo.htm.

The simulated robot's profile has been configured to represent that of the physical Ballybot robot. A screenshot taken of the simulation test environment can be seen in Figure 5-4.


PID ***BalBot*** Demonstration

☑ Add Sensor Noise  Slow down factor: 2  Effective sample rate: 50  Maximum force: 3.0

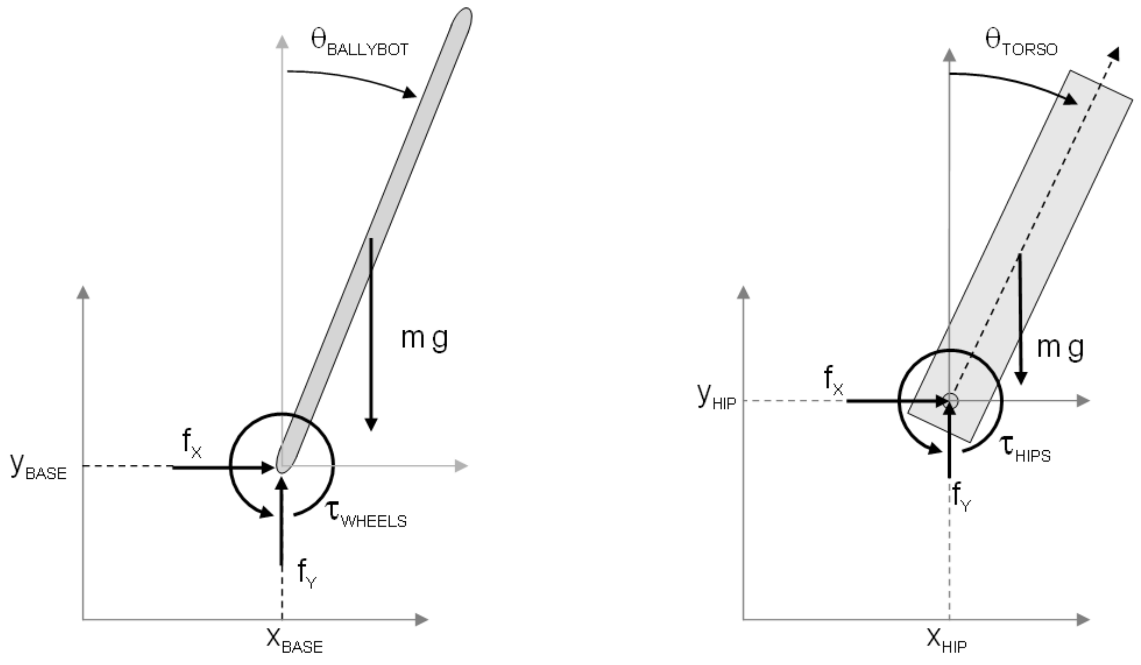
Force : 2.28895330429077  K1: 32.0  K2: 1.1  K3: 2.5  K4: 0.5  Max speed: 0.4962644

**Figure 5-4 Screenshot of Ballybot software simulation in action. The profile of the robot is shown, with the controller, batteries, motors and frame represented by a collection of rectangles and circles. The robot's centre of mass is indicated by a light red circle in the background of the image.**

### 5.5.3 Methods

#### i. Implementing the Torso Attitude Controller on a Ballybot robot

The Torso Attitude Controller's task is to determine, at each sample interval, a torque to apply to a biped robot's hip joints in order to maintain balance of the robot's torso. In the "Torso Driven Walking" control system, the Torso Attitude Controller applies the balancing torque to the hip joint of the supporting leg only. The Ballybot robot has no legs; rather the hip joints are replaced by a pair of motor driven wheels. Torque applied evenly to the robot's wheels translates directly into a balancing ground reaction force. A comparison of the two systems shows that the system models are very similar (Figure 5-5):



**(a) Ballybot system model** **(b) Biped torso system model**

**Figure 5-5 The Ballybot and biped torso system models are almost identical.**

In the Ballybot system, the base of the robot is constrained to remain in contact with the ground. The vertical reaction force ($f_Y$) is a passive force, acting to prevent the base of the robot from penetrating the ground. The horizontal reaction force ($f_X$) is entirely generated by the robot wheel torque ($\tau_{WHEELS}$).

The vertical position of the hip joints of the biped torso, for which the TAC has been developed, is not fixed. As the robot walks, it is very likely that the hip joint will move up and down, in response to actions of the robot's legs. The robot legs subject the system to translational impulses and forces acting on the hip joints. The horizontal hip joint force ($f_X$) will not be solely dependent on hip torque ($\tau_{HIPS}$), and the vertical reaction force ($f_Y$) is not entirely passive. However, as long as the rest of the "Torso Driven Walking" control system components are doing their job correctly, these active forces will be only a small component of the total force acting on the robot's hip joints. They are treated as noise, and ignored by the Torso Attitude Controller.

ii. Control System

From Chapter 3.2.5, "Torso attitude controller (TAC)": whenever the supporting foot is in contact with the ground, the TAC takes the form:

$$\tau_{SUPPORT}\ (k) = a\,\theta_{TORSO} + b\,\dot{\theta}_{TORSO} + c\,(x_{HIP} - x_{SP}) + d\,(\dot{x}_{HIP} - \dot{x}_{SP}) - \tau_{FREE}(k) \quad (1)$$

On a Ballybot robot, both wheels are always in contact with the ground, and the support torque is evenly distributed between the two wheels. There is no "free" leg, so:

$$\tau_{SUPPORT} = \tau_{WHEELS}$$

$$\tau_{FREE} = 0$$

In this experiment, the robot is being held steady in the centre of the workspace; the displacement and velocity set points are both zero:

$$x_{SP} = 0$$

$$\dot{x}_{SP} = 0$$

The Torso Attitude Controller control system is being applied to the Ballybot simulator, which is controlled by applying a horizontal balancing force using the ActiveX control's

`ApplyForce()` method[13]. Since the horizontal ground reaction force is proportional to motor torque:

$$f\,(k) \propto \tau_{SUPPORT}$$

The TAC controller equation can be reduced to the following equation:

$$f(k) = \tilde{w}^T \cdot \tilde{x}(k) \qquad (2)$$

Where:

$$\tilde{x}\,(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \theta(k) \\ \dot{\theta}(k) \end{bmatrix} \qquad \tilde{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Equation (2) is in the form of a commonly cited classical solution for the inverted pendulum problem, from Ogata [26].

**Table 5-1 Inverted pendulum control system variables.**

| | | | |
|---|---|---|---|
| $\tilde{x}\,(k)$ | State vector of Ballybot system. | $\tilde{w}$ | Constant weight vector applied to measured robot state. |
| $x\,(k)$ | Horizontal displacement of the base of the Ballybot. | $\dot{x}\,(k)$ | Horizontal velocity of the base of the Ballybot. |
| $\theta\,(k)$ | Angle of inclination. | $\dot{\theta}\,(k)$ | Angular velocity. |
| $f\,(k)$ | Horizontal force applied to the base of the Ballybot. | | |

Figure 5-6 shows the control flow diagram for the Torso Attitude Controller for Ballybot control system:

[13] The BalanceBot interface is described in Appendix B-3.

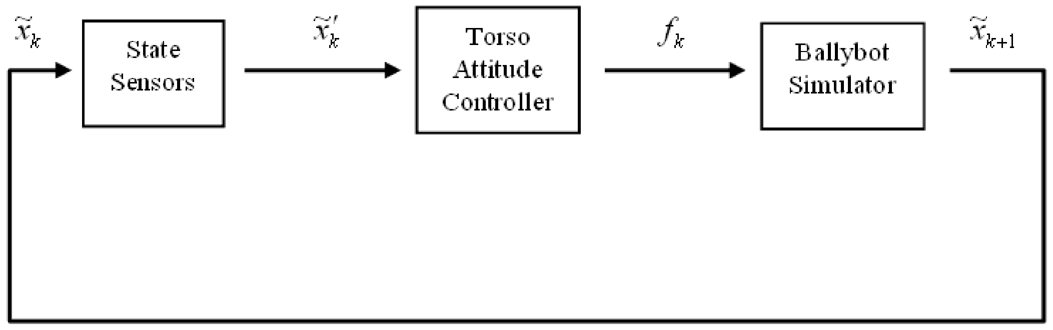


**Figure 5-6 Control flow diagram for "Torso Attitude Controller", as used to control the Ballybot robot simulation.**

An advantage of the conventional Torso Attitude Controller over the adaptive system presented in Chapter 5.8 is its simplicity. In this experiment, the entire control system is implemented by three lines of VBScript, as shown by the example code in Figure 5-7. The script is called every 50 ms to read the system state, and calculate and apply the balancing force.

```
' State sample function, called every 50ms to
' determine an appropriate balancing force...
Sub ControlTimer_OnTimer
      ' Get the system state
      call BalanceBot.GetFullStateVariant( _
             fX, fXVelocity, fAngle, fAngleVelocity )

      ' Calculate balancing force
      fValue = _
            K1 * fAngle + K2 * fAngleVelocity + _
            K3 * fX + K4 * fXVelocity

      ' Apply balancing force
      BalanceBot.ApplyForce( fValue )
End Sub
```

**Figure 5-7 VBScript used to implement the Torso Attitude Controller. The object "BalanceBot" is an instance of the "BalanceBot.ocx" ActiveX control, which encapsulates the Ballybot dynamic system.**

iii. Tuning the control system

The performance of the system can be modified by adjusting the weight vector elements:

$$\tilde{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Values for these weights can be calculated analytically, for example by using the "pole placement" technique described by Ogata [26]. However the calculations are complex, and need to be repeated if any part of the system is changed – such as moving the position of the battery pack on the robot or changing the rate at which samples are taken and controls performed. Instead the appropriate weights are determined experimentally, using the Ballybot simulator hosted in an HTML web page.

### 5.5.4 Results

Tuning of the weight vector was performed manually, using the software simulation to observe the effect of modifying loop parameters. Using this approach, a set of control system weights that successfully balanced the pendulum system can be determined in minutes.

The tuning process is very forgiving, as there are many possible solutions that succeed in balancing the simulated Ballybot. Each solution varies slightly in its behaviour, in terms of settling time and damping in response to disturbances.

Due to the robust nature of the control system, minor changes to the system model do not cause the system to fail. Major changes can quickly be accommodated by experimentally readjusting the control parameters.

### 5.5.5 Discussion

The small number of trials needed to tune the Torso Attitude Controller on a simulated Ballybot robot is a particularly promising result. This means that when the control system is ported to a real robot, it should be possible to develop a satisfactory implementation in a short amount of time, minimising unnecessary wear and tear on the physical robot.

Obviously, controlling a simulated robot is very different to controlling a real robot in the physical world. This experiment proved the Torso Driven Walking's Torso Attitude Controller subsystem can be used to balance the Ballybot. The main objective of the remaining experiments in this chapter will be porting the system to a physical robot, and using the resulting system to evaluate sensors for balancing robots.

## 5.6 Force from voltage

**Abstract:** Regardless of the type of control system to be implemented, successful balancing of the Ballybot requires the ability to deliver an appropriate variable torque to the robot's wheels. This experiment investigates the relationship between applied motor voltage, motor shaft velocity and resultant output torque on the physical Ballybot robot.

### 5.6.1 Introduction

Any control system that attempts to balance the Ballybot robot must do so by varying wheel torque, in order to induce a balancing horizontal ground reaction force. The Ballybot's wheel torque is supplied by a pair of DC motors that are controlled by applying an input voltage to the motors. In order to control the balancing force induced by torque supplied to the robot's wheels, a control system needs to be able to model the relationship between applied voltage and resulting horizontal ground reaction force. If a reliable model mapping desired motor torque to applied voltage can be found, then future experiments with the Ballybot can concentrate on the problem of sensor interpretation with confidence.

An overview of DC Motors is been presented in Appendix A-2.2. Using an equivalent circuit model for a DC motor, it is shown that motor output torque is proportional to applied voltage, and inversely proportional to angular velocity and the rate of change in coil current:

$$\tau_{motor} = K_{motor} (V_{applied} - K_e \omega - L_{motor} ({}^{dI}/_{dt})) / R_{motor} \qquad (1)$$

Since a convenient tool for sensing ($^{dI}/_{dt}$) using the Ballybot's EyeCon controller[14] is unavailable, a simplified motor model, suggested by Bräunl [27] is used. This model assumes that $L_{motor}$= 0 H, resulting in the relationship:

$$\tau_{motor} = K_{motor} (V_{applied} - K_e \omega) / R_{motor} \qquad (2)$$

The question that needs to be answered by this is experiment is: "is this a reasonable approximation?" If so, an experimentally determined relationship should be planar, and a graph would look something like Figure 5-8. If this experiment determines that it does not, a more accurate system model must be formulated.

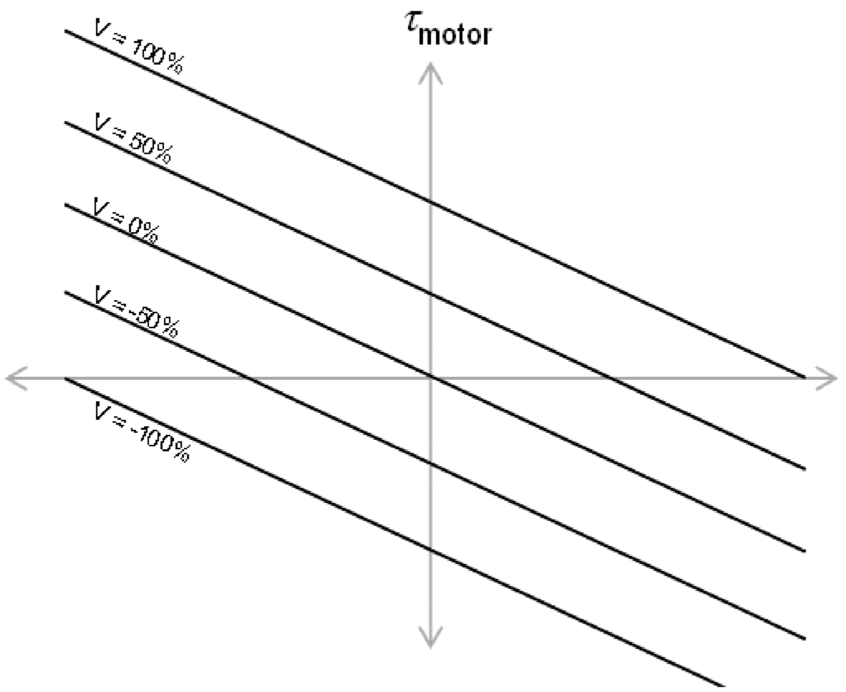


**Figure 5-8 If the simplified motor model (equation 2) is valid, then a graph of motor torque vs. shaft angular velocity for varying input voltages would look something like this.**

There are many possible reasons why this simple, planar relationship may not be valid. Some possibilities include:

[14] A possibility might be to add a resistor in series with the motor power supply, and use the Ballybot's analogue inputs to measure the voltage drop across the resistor, allowing current to be inferred. Due to some issues encountered using multiple analogue inputs on the EyeCon controller, this was not attempted.

- Friction between the motor shaft and the motor casing or wheel mounts will have a major impact on the torque/voltage/velocity relationship.
- Since the system is a real-world component, powered by batteries, there is a good chance that the voltage cannot be accurately controlled. Large power drains from other parts of the robotic system, degrading battery strength over time, capacitances and other energy losses in the controller could all influence the quality of voltage regulation.
- It is possible that the influence of the motor coil inductance cannot be ignored.

Of these effects, friction definitely will have an effect on the effective wheel torque generated by an applied voltage. The effect of this force can be modelled as a torque working against the direction of angular velocity.

A number of models for calculating the magnitude of frictional forces have been used by many researchers. A common approach is to use a static friction at zero velocity, and a dynamic component while the system is moving (Liu [28], Dupont [29]). A simple "kinetic/static" model for friction is shown in Figure 5-9. When the robot is stationary, friction works against motor torque with a magnitude $\tau_{static}$. When the robot is moving, friction works against angular velocity with a magnitude $\tau_{kinetic}$.

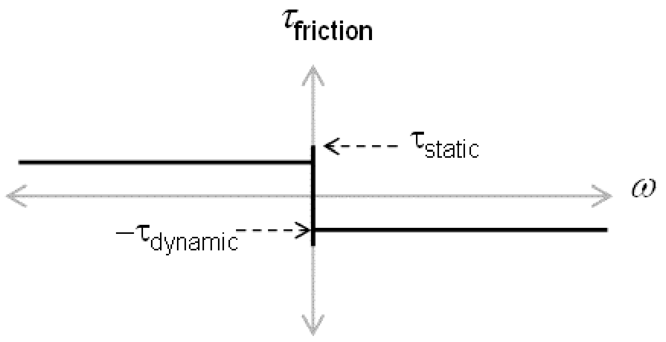


**Figure 5-9 A simple kinetic/static friction model**

If this friction model is simplified further, by assuming $\tau_{\text{static}} = \tau_{\text{kinetic}}$, the influence of friction on the ideal motor model can be illustrated by Figure 5-10:

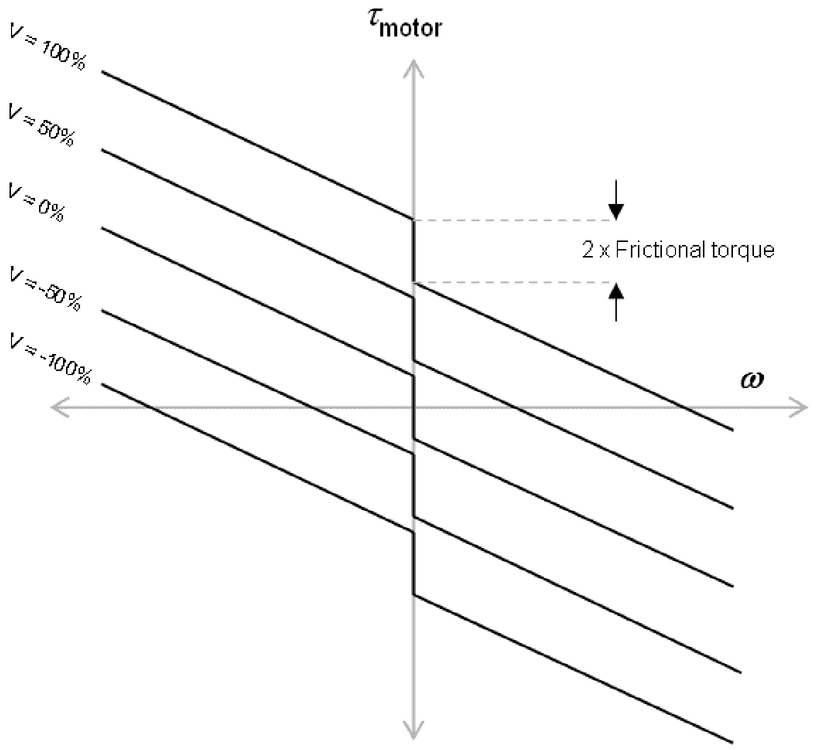


**Figure 5-10 Motor model incorporating simplified effects of friction.**

Luckily, the Ballybot robot is a simple enough system that the relationship between applied voltage, wheel torque and wheel angular velocity can be experimentally determined. Since the experiment is using the motors "in-situ", it is also testing other implementation specific influences, such as effects of changing voltage due to power drain or current limits, etc.

### 5.6.2 Materials

i. Ballybot robot

The physical Ballybot robot is used to conduct this experiment.

ii. Toy truck

A toy truck was used to support the Ballybot's "head" as it lay on the ground, effectively turning the robot into a simple wheeled vehicle. The truck was not powered in any way. Its wheels could be spun with very little frictional resistance[15].

### 5.6.3 Methods

i. Software

A simple software program was written to record the robot's translational displacement, using the DC motor's inbuilt optical encoders. At the end of a trial, the recorded displacement measurements, together with corresponding timestamps were then transmitted to a computer for processing.

ii. Trial preparation

Before each trial, the Ballybot robot was rested on the toy truck, with the combined system effectively becoming a single vehicle, as shown in Figure 5-11:

[15] Nevertheless, friction introduced by the toy car will be a component of the experimentally determined friction in this experiment.

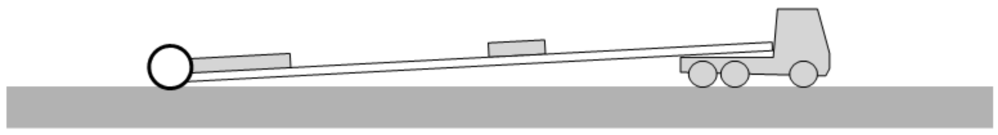


**Figure 5-11 Ballybot robot lying on the tray of a toy truck. This system was used to experimentally determine thee relationship between applied actuator voltage and resultant force.**

iii. Trial execution

When a voltage is applied by the Ballybot controller to the DC motors, the system accelerates across the bench top. A number of trial runs were taken to explore the effect of changing motor voltage and robot velocity to the acceleration experienced by the system.

During each trial, displacement of the robot was measured using the motor's inbuilt optical encoders. Velocity and acceleration of the robot were calculated from the first and second derivatives of the measured displacement with respect to time. This approach can be expected to introduce some measurement errors, since differentiating a noisy measurement has the effect of amplifying the signal noise. Fortunately, for the most part, the resulting trends do not exhibit an unreasonable level of noise.

Specific trial run scenarios included:

- Decelerations with zero voltage applied (to determine friction effects and inductive influences at 0V).
- Accelerate robot from a standing start, with each trial applying a different constant voltage.
- Accelerate robot from an initial negative velocity, with each trial applying a different constant voltage.

During this experiment:

- Applied voltage level is expressed as a % value, ranging from -100% to +100%. This is the representation used by the RoBIOS operating system. The corresponding voltage level is not important to the Ballybot implementation.
- Acceleration measurements have not been converted to torque, since these terms are linearly related.
- Robot velocity measurements have not been converted to wheel angular velocity, since the terms are linearly related.

The goal of the experiment is to determine a mathematical model describing the relationship between velocity, voltage and acceleration of the experimental system. Hopefully the resulting measurements will either confirm the simple model illustrated by Figure 5-10, or else a new model will be evident.

### 5.6.4 Results

The results presented here demonstrate the process by which the acceleration/voltage/velocity relationship was investigated, finally culminating in a proposed motor model to be used in subsequent experiments.

#### i. Accelerate from rest

Starting from rest, various steady voltages are applied. Changing accelerations are recorded as velocity increases due to the applied voltage:

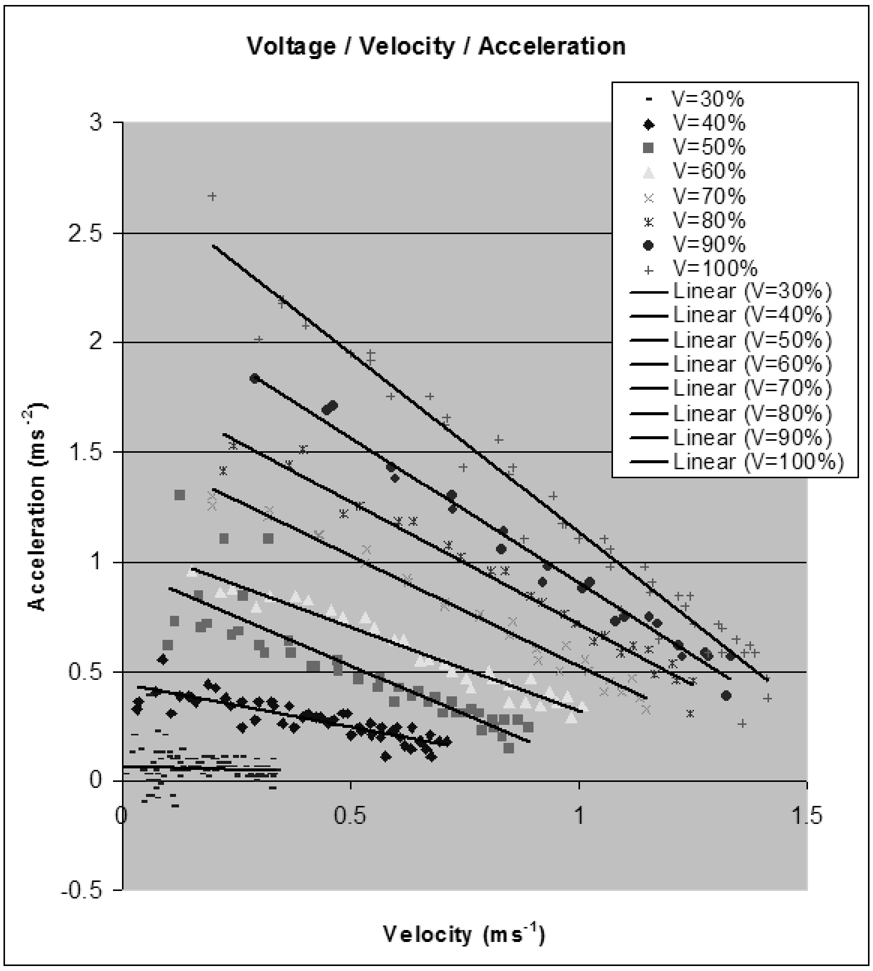


**Figure 5-12 Accelerating from rest, with varying applied voltage levels.**

Observations:

- With an applied voltage of not much under 30%, the robot was unable to overcome the static friction and remained motionless.
- Trends of acceleration vs. velocity are close to parallel (as predicted by equation 2). However, as voltage level drops trend gradients appear to become less steep.
- At lower velocities, the occurrence of noisy acceleration measurements increases. This biases the linear regression lines shown on the chart for low voltage trials.

ii. Deceleration with zero voltage

The robot is accelerated to -1.5 ms$^{-1}$, then switches off power to the motors and records the deceleration as the system rolls to a stop:

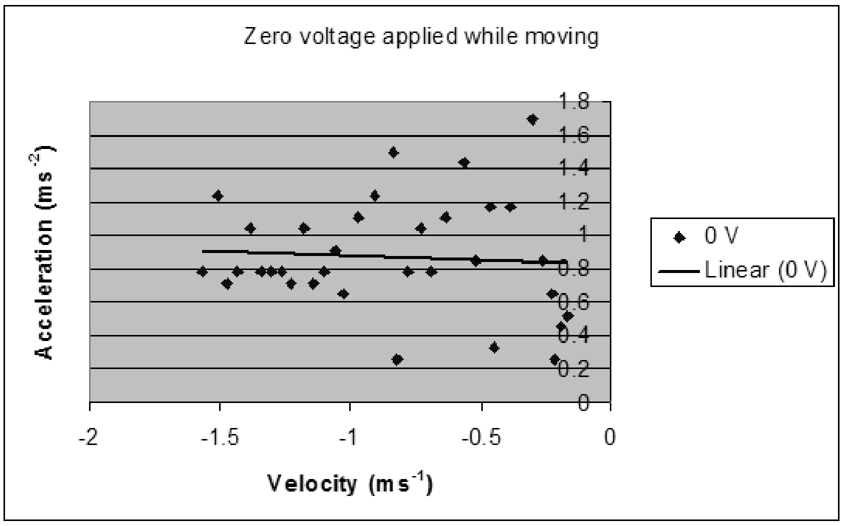


**Figure 5-13 Deceleration acting on moving robot with zero applied voltage**

Observations:

- The measured acceleration appears to be linear, and constant (although very noisy).
- Signal noise measurements become greater at low velocities.
- The acceleration due to friction effects is approximately 0.85 ms$^{-2}$.

iii. Accelerate from initial negative velocity

Record acceleration as velocity increases due to applied voltage. Initial velocity is negative, so the effects of friction should reverse as velocity moves from < 0 ms$^{-1}$ to > 0 ms$^{-1}$.

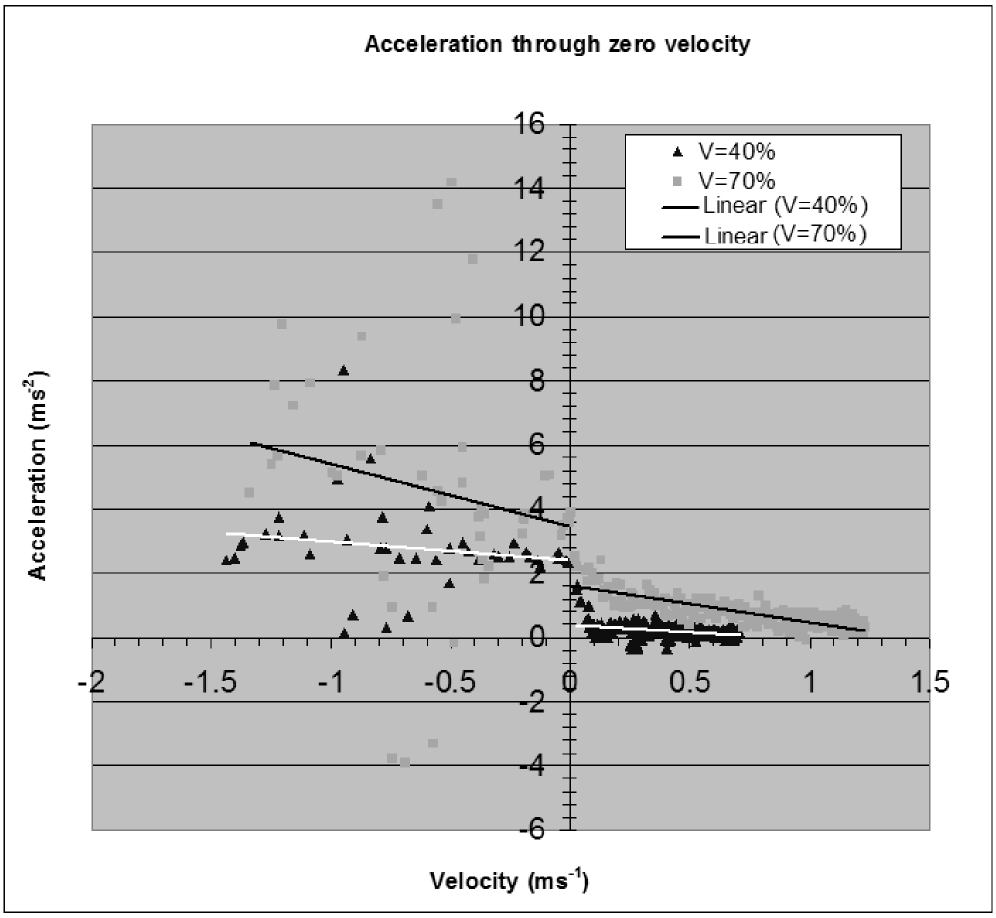


**Figure 5-14 Two trials examine the transition from negative to positive velocity, with a constant applied voltage.**

Observations:

- There is a large level of noise in acceleration measurements taken while the applied voltage acts against the velocity. (In Figure 5-14, this is when velocity is < 0 ms$^{-1}$).
- The step change in acceleration as velocity shifts from negative to positive is due to friction effects. The magnitude of the step change is in the order of 1.8 ms$^{-2}$. I expected the magnitude of the step change to be twice the "friction" value. Consequently, this experiment suggests a "friction" deceleration of $\approx 0.9$ ms$^{-2}$, which is consistent with the "friction" estimate of 0.85 ms$^{-2}$ made in trial (ii).

- The slope of the linear regression lines on either side of the acceleration discontinuity are close to parallel for the same applied voltage. Differences are probably due to noisy acceleration measurements.
- The regression lines of trials taken using the two voltages (40% and 70%) are significantly different, with the slope of the lower voltage trend less steep than the higher voltage trend.

### 5.6.5 Motor model synthesis

Examining the experimental results above, it is obvious that velocity retarding factors, collectively termed “friction”[16], have a large impact on the Ballybot motor model. Other observations have included:

- The slope of regression line at constant voltage appears to be constant.
- The slope of regression line is less as the applied voltage magnitude is reduced.
- There is a friction related discontinuity in acceleration as the robot velocity moves through 0 $ms^{-1}$.

Disregarding outlier measurements and extrapolating remaining samples from Figure 5-12, results in the graph shown by Figure 5-15. This graph relates acceleration to velocity, for various applied voltages, ranging from +30 to +100%.

[16] Factors making up this “velocity retarding force” include frictional forces, but may also include factors such as motor coil inductance, fluctuating power supply levels, etc.

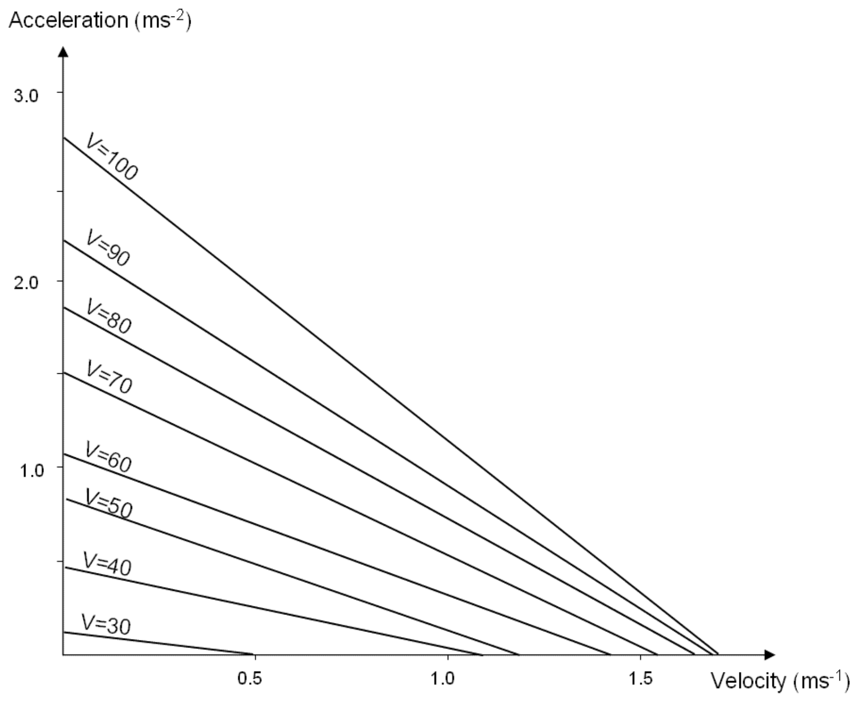


**Figure 5-15 Approximate regression lines for acceleration vs. velocity for various voltages.**

Since the robot velocity is always positive in this chart, the graph shows experimentally measured acceleration levels that have been reduced by friction acting on the robot system. The objective was to develop a motor model for the Ballybot that disregards the effects of friction. Fiction compensation could then be explicitly added to this "frictionless" model, in such a way as to keep the effects of friction a distinct component of the final motor model.

Figure 5-16 redraws the experimentally determined acceleration/velocity relationships, with the acceleration retarding effects of friction removed. This adds 0.85 $ms^{-2}$ to the experimental measurements:

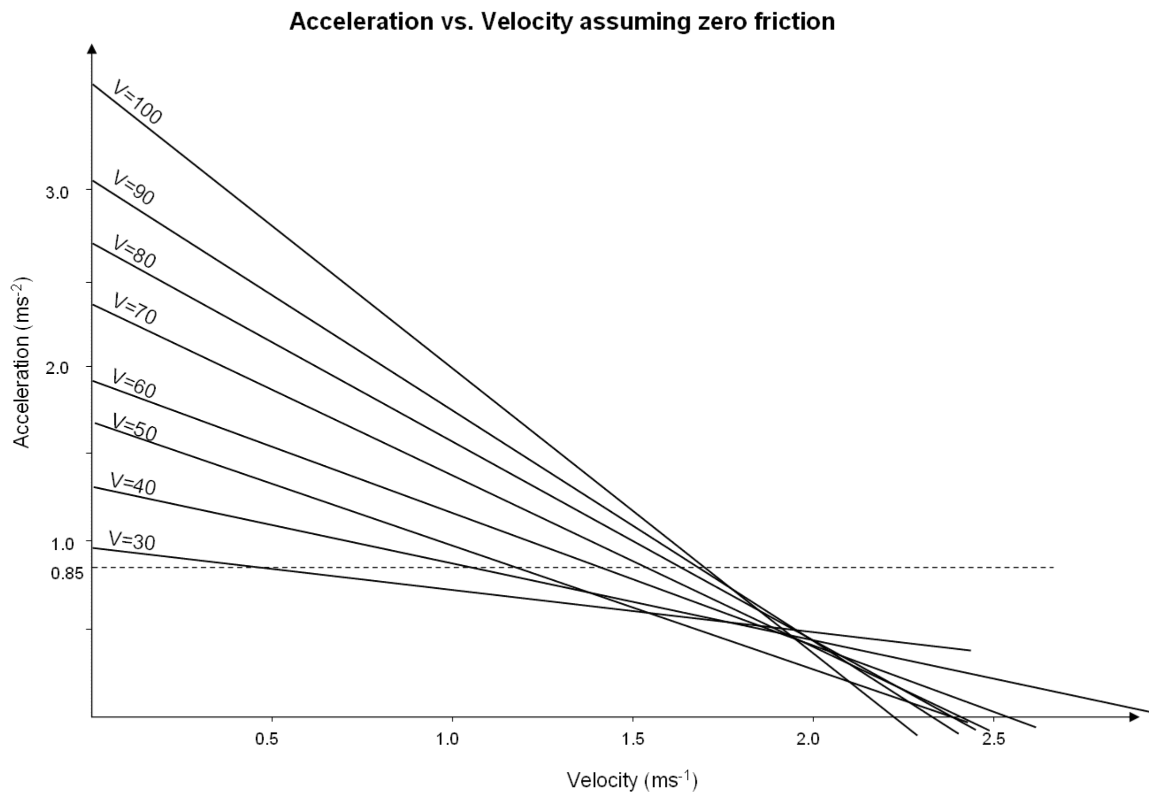


**Figure 5-16 This graph shows acceleration vs. velocity with a range of applied voltages. The graph has been adjusted to remove the effects of friction, assuming that friction reduces acceleration by 0.85 ms$^{-2}$, while velocity is positive.**

The relationship suggested by Figure 5-16 is used to develop a "frictionless" Ballybot motor model for applied voltages > 0. Since the behaviour of the motors is symmetrical, once a model for positive applied voltage is determined, a corresponding model for negative applied voltage could be generated.

**Assumption #1: The relationship between acceleration and velocity is linear for a constant applied voltage.**

Applying assumption #1, the model takes the form:

$$acc_{V+}(V_{applied}, v) = a_{initial}(V_{applied}) - \frac{a_{initial}(V_{applied})}{v_{\max}(V_{applied})} v \qquad (3)$$

Where:

| | |
|---|---|
| $acc_{V+}(V_{applied}, v)$ | Acceleration as a function of applied voltage and current velocity, valid for $V_{applied} \geq 0$. |
| $V_{applied}$ | Applied voltage |
| $v$ | Current velocity |
| $a_{initial}(V_{applied})$ | Acceleration due to applied voltage, when velocity is zero. This is the y-axis intercept. |
| $v_{\max}(V_{applied})$ | Projected maximum velocity at a steady applied voltage (assuming no friction). This is the x-axis intercept. |

To calculate the relationship, first the functions $a_{initial}(V_{applied})$ and $v_{\max}(V_{applied})$ must be determined. Plotting the y-axis intercept values read from figure 30 suggests that $a_{initial}(V_{applied})$ can be represented by a linear function.

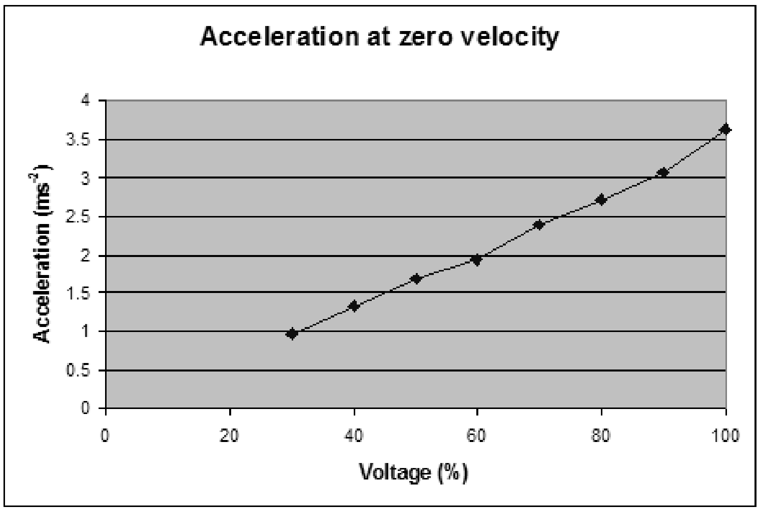


**Figure 5-17 The relationship between initial acceleration and voltage can be represented by a linear function.**

**Assumption #2: The relationship between applied voltage and initial acceleration is linear.**

From Figure 5-17, and applying assumption #2:

$$a_{initial}(V_{applied}) \approx 0.034\,V_{applied} \tag{4}$$

Determining an equation to represent $v_{\max}(V_{applied})$ is more difficult. This function represents the x-axis intercepts of trends in Figure 5-16. An examination of the graph shows that all but the two lowest voltage trend lines cross the x-axis within 0.25 ms$^{-1}$ of each other. I have assumed that variation in the position of the x-axis intercept is caused by experimental errors. The noise experienced recording the lowest voltage accelerations is most extreme, which might explain the high variation in x-axis intercept positions for $V_{applied} = 30V$ and $40V$.

**Assumption #3: The terminal velocity of the robot in the absence of friction is a constant value, independent of applied voltage.**

As a result of assumption #3, a constant, scalar value was used to represent the maximum robot velocity:

$$v_{\max}(V_{applied}) \approx 2.3 \tag{5}$$

Substituting equations (4) and (5) $\rightarrow$ (3):

$$acc_{V+}(V_{applied}, v) = 0.034\,V_{applied} - 0.0148\,V_{applied}\,v \tag{6}$$

The corresponding model for negative applied voltage is:

$$acc_{V-}(V_{applied}, v) = 0.034\,V_{applied} + 0.0148\,V_{applied}\,v \qquad (7)$$

Since the Ballybot requires the ability to specify a resultant ground reaction force, and this force must compensate for any frictional forces, the motor force required to compensate for friction must be calculated:

$$f_{motor} = f_{balance} + f_{friction}\text{, for } v > 0\text{ ms}^{-1} \qquad (8)$$

$$f_{motor} = f_{balance} - f_{friction}\text{, for } v < 0\text{ ms}^{-1} \qquad (9)$$

Where:

| | |
|---|---|
| $f_{balance}$ | Force that the control system wants to apply to the base of the robot to prevent it falling. |
| $f_{friction}$ | Magnitude of the frictional force acting against the current velocity of the robot. |
| $f_{motor}$ | Force that must be generated by the motor (using the "frictionless" motor model) |

Using Equations (8) and (9), the motor model determines a "frictionless" force ($f_{motor}$) that must be supplied by the motors to achieve the desired balancing force ($f_{balance}$).

Using the relationship: $f = Ma$, and the fact that in the frictionless model, a positive force can only be supplied by a positive voltage (and vice versa for negative force):

$$f_{motor} = M\,acc(V_{applied}, v) \qquad (10)$$

$$f_{motor} = M\left(0.034\,V_{applied} - 0.0148 V_{applied}\,v\right),\ \text{if}\ \ f_{motor} \geq 0 \qquad (11)$$

$$f_{motor} = M\left(0.034\,V_{applied} + 0.0148 V_{applied}\,v\right),\ \text{if}\ \ f_{motor} \geq 0 \qquad (12)$$

$$M = \text{(Mass of Ballybot + mass of the toy truck) = 0.890 kg} \qquad (13)$$

Substitute (13) $\rightarrow$ (11) and (12), and rearranging the equations:

$$V_{applied} = \frac{f_{motor}}{0.0303 - 0.013v},\ \text{if}\ \ f_{motor} \geq 0 \qquad (14)$$

$$V_{applied} = \frac{f_{motor}}{0.0303 + 0.013v},\ \text{if}\ \ f_{motor} < 0 \qquad (15)$$

So the final system model requires the use of equations (8), (9), (14) and (15). These equations were used by the Ballybot control system to operate the robot's motors during experiment 5.7.

### 5.6.6 Discussion

The process of using the Ballybot motor model to calculate voltage required to provide a desired force, given a known current velocity is summarised here:

1. Calculate required motor force, using:

$$f_{motor} = f_{balance} + f_{friction},\ \text{for}\ \ v > 0\ \text{ms}^{-1} \qquad (8)$$

$$f_{motor} = f_{balance} - f_{friction},\ \text{for}\ \ v < 0\ \text{ms}^{-1} \qquad (9)$$

2. Calculate required input voltage, using:

$$V_{applied} = \frac{f_{motor}}{0.0303 - 0.013v},\ \text{if}\ \ f_{motor} \geq 0 \qquad (14)$$

$$V_{applied} = \frac{f_{motor}}{0.0303 + 0.013v}, \text{ if } f_{motor} < 0 \qquad (15)$$

This motor model is presented graphically in Figure 5-18. Voltages applied in the direction of the robot's motion have a diminishing impact on robot acceleration as the magnitude of the robot's velocity increases. Conversely, voltage acting against the direction of the robot's motion has increasing impact on deceleration with larger velocities.

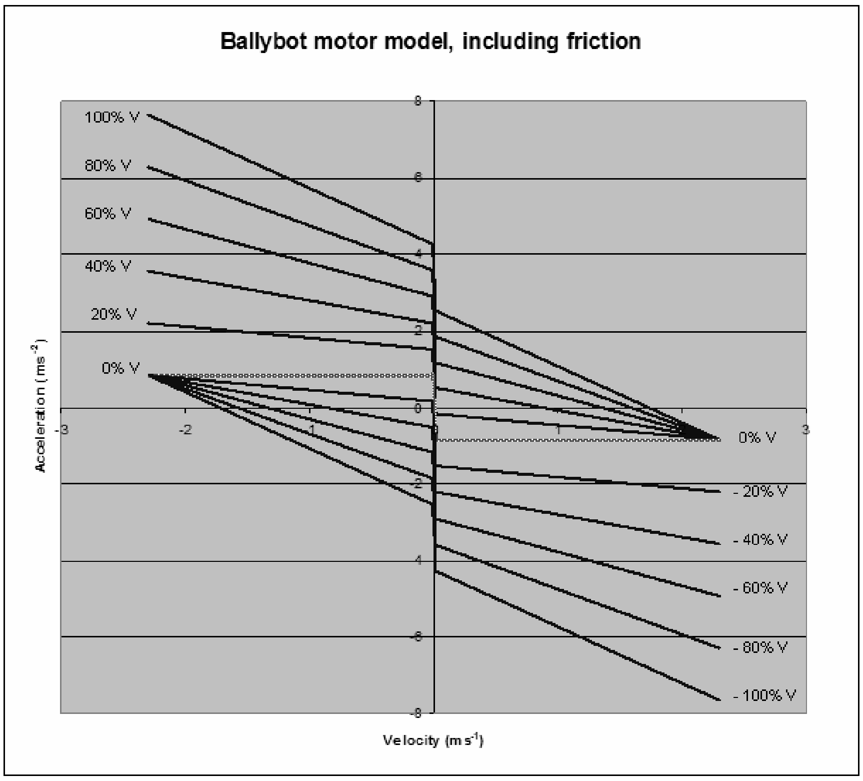


**Figure 5-18 A graphical representation of the full Ballybot motor model, used in subsequent experiments.**

If the Ballybot control system is doing its job correctly, in operation the Ballybot velocity will be held close to 0 ms$^{-1}$. The maximum velocity achieved by the robot during the simulations conducted during experiment 5.5 was less than 1 ms$^{-1}$, even in the presence of simulated sensor noise. This means that the operating Ballybot should remain in the velocity envelope for which the system model has been experimentally validated. As a result, the system model described here is suitable for use on the

Ballybot robot, and has been applied to the final Ballybot implementation, described in Chapter 5.7.

## 5.7 Sensors for balance

**Abstract**: One of the more difficult problems facing developers of bipedal walking robots is the selection of appropriate sensors for maintaining balance. In this experiment, the Ballybot robot is used to evaluate proposed robotic tilt sensors and sensor data processing algorithms. Two inexpensive sensors in particular are examined: a piezo-electric gyroscope designed for remote control helicopters, and an inclinometer. The results demonstrate that while it is not possible to balance the robot with only one of these sensors, it is possible to use the sensors co-operatively to control the robot.

### 5.7.1 Introduction

In order to effectively maintain balance in biped machines, the selection of an appropriate set of sensors for providing system state estimations is critical [1-3]. Additionally, if an evaluation of sensor performance is left until the completion of the construction of a complex biped robot, it will be difficult to determine if a failure to achieve the desired results is due to deficiencies in control algorithms, robot construction, or poor sensor performance.

The primary motivation for building the Ballybot robot was to provide a test platform for investigating the suitability of proposed sensor configuration and data processing techniques to balancing robotic systems. The Ballybot is an example of an autonomous inverted pendulum robot. This class of robot has already been used as the basis of a number of bipedal walking strategies [7-10], the dynamics can be constrained to two dimensions and the cost of producing the robot is low.

The Ballybot robot system consists of the physical Ballybot robot, the Ballybot "Torso Attitude Controller" control system, the Ballybot "DC motor model" and state estimation sensors. Through a series of previously conducted experiments (detailed in 5.5 & 5.6), a high degree of confidence has been established regarding the performance of all components of the Ballybot robotic system, with the exception of sensors for tilt angle estimation – the subject of this experiment.

Investigations into candidate sensors for detecting the robot's tilt angle have concentrated on evaluating two sensors for measuring the robot tilt angle: a piezo-electric gyroscope and an inclinometer. While it was possible to balance the robot using the gyroscope only, it was not possible to simultaneously control the horizontal displacement of the robot. This is due to the way the control system automatically compensates for the gyroscope's bias error. When both the gyroscope and inclinometer sensors were used cooperatively, it was possible to successfully balance the robot while controlling its displacement.

Note that a number of other sensor types, such as pressure/force sensors and encoders/potentiometers may also be useful in the control of humanoid robots, especially for control strategies relying on the detection of the zero moment point [30-32]. The Ballybot inverted pendulum based test platform is not suitable for evaluating these kinds of sensors.

### 5.7.2 Materials

i. Ballybot robot

The physical Ballybot robot is used to conduct this experiment.

ii. Tilt angle sensors being used

During the course of this experiment, the following tilt sensors were tested on the Ballybot:

- **Gyroscope (Hitec GY-130 piezo rate gyro).** This is a piezo-electric gyroscope designed for use in remote controlled vehicles, such as model helicopters. The gyroscope modifies a servo control signal by an amount proportional to its estimate of angular velocity. Instead of using the gyro to control a servo, we read back the modified servo signal to obtain a measurement of angular velocity. An estimate of angular displacement is obtained by integrating the velocity signal over time.
- **Inclinometer (SEIKA Inclinometer N3).** This sensor outputs an analogue signal, proportional to the angular displacement of the sensor. The sensor is accurate within the range -30 to +30 degrees.

Neither sensor on its own completely solves the problem of balancing the physical robot. The gyroscope performs best of the two sensors, consistently balancing the robot for intervals longer than 2 minutes, after which the robot usually moves out of the workspace and the experiment must be halted. The robot is unable to balance using feedback from the inclinometer only.

Experiments using both sensors in a collaborative manner have provided the best results. The most successful algorithm utilises data sampled from the inclinometer to recalibrate the gyroscope "on-line".

### 5.7.3 Methods

A control system is implemented to balance the pendulum by regulating the voltage supplied to the robot's motors. Then various combinations of state sensors and data processing algorithms were trialled, until a combination was found which successfully balanced the Ballybot robot.

### 5.7.3.1 Control system

During this experiment, the Ballybot utilises the Torso Driven Walking subsystem "Torso Attitude Controller". This component has been tested using the software simulation in Chapter 5.5, and is used in conjunction with the Ballybot's "DC Motor Model" developed in Chapter 5.6. The resulting system is represented by the control flow diagram, in Figure 5-19:

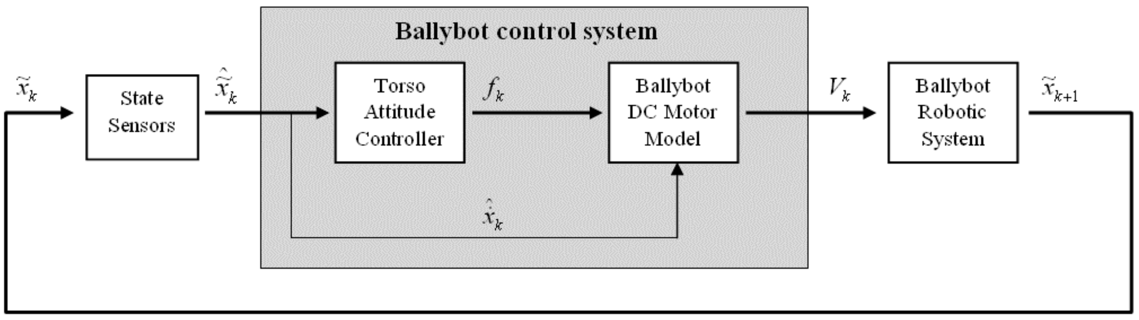


**Figure 5-19 Control flow diagram for the complete Ballybot robotic system**

The system variables referenced by the control flow diagram, and all other components of the control system are summarised in Table 5-2:

**Table 5-2 Ballybot control system variables.**

| | | | |
|---|---|---|---|
| $\tilde{x}_k$ | State vector of the Ballybot system: $$\tilde{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \\ \theta_k \\ \dot{\theta}_k \end{bmatrix}$$ | $\tilde{w}$ | Constant weight vector applied to measured robot state: $$\tilde{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$ |
| $\hat{\tilde{x}}_k$ | Estimate of the state vector of the Ballybot system. | $\hat{\dot{x}}_k$ | Estimated horizontal velocity of the base of the Ballybot. |
| $x_k$ | Horizontal displacement of the base of the Ballybot. | $\theta_k$ | Angle of inclination. |

| | | | |
|---|---|---|---|
| $\dot{x}_k$ | Horizontal velocity of the base of the Ballybot. | $\dot{\theta}_k$ | Angular velocity. |
| $f_k$ | Horizontal force applied to the base of the Ballybot. | $V_k$ | Voltage applied to Ballybot motors (-100% to +100%). |
| $f_{motor}$ | Force that must be generated by the motor torque. | $f_{friction}$ | Magnitude of the frictional force acting against the current velocity of the robot. ($\approx 0.85N$) |

i. Torso Attitude Controller (for Ballybot)

This system was developed during experiment 5.5, "Torso Attitude Controller for the Ballybot". The controller takes an estimate of the four state variables ($x_k$, $\dot{x}_k$, $\theta_k$, $\dot{\theta}_k$) as inputs, and determines an appropriate balancing force ($f_k$) to keep the robot upright and centred. The required balancing force is calculated using equation (1):

$$f_k = \tilde{w}^T \cdot \hat{\tilde{x}}_k \tag{1}$$

The weight vector components $(w_1, w_2, w_3, w_4)$ were determined experimentally, using initial settings obtained using the software simulation in Chapter 5.5.

ii. Ballybot DC motor model

The Ballybot utilises the "DC motor model", developed in Chapter 5.6. Using the experimentally determined relationship, the robot determines an applied voltage level ($V_k$) that will result in the desired balancing force ($f_k$), given the current estimate of robot velocity ($\hat{\dot{x}}_k$). This voltage is determined by:

1. Calculate required motor force, using:

$$f_{motor} = f_k + f_{friction}, \text{ for } \hat{\dot{x}}_k > 0 \text{ ms}^{-1} \tag{2}$$

$$f_{motor} = f_k - f_{friction}, \text{ for } \hat{\dot{x}}_k < 0 \text{ ms}^{-1} \tag{3}$$

2. Calculate required input voltage, using:

$$V_k = \frac{f_{motor}}{0.0303 - 0.013\hat{\dot{x}}}, \text{ if } f_{motor} \geq 0 \tag{4}$$

$$V_k = \frac{f_{motor}}{0.0303 + 0.013\hat{\dot{x}}}, \text{ if } f_{motor} < 0 \tag{5}$$

Since the voltage is expressed as a percentage of maximum applicable voltage, it is restricted to the range (-100%, +100%).

5.7.3.2 Obtaining system state measurements:

The proportional control strategy selected for implementation on the physical robot, and described by (1), requires a measurement of four state variables: $(x, \dot{x}, \theta, \dot{\theta})$.

**Table 5-3 Ballybot control system input.**

| Variable | Description | Sensor(s) |
|---|---|---|
| $x$ | Position | Shaft encoders built in to motors. |
| $\dot{x}$ | Velocity | Estimated by differentiating the encoder's $x$ reading. |
| $\theta$ | Angle | i. Estimated by integrating the gyroscope's $\dot{\theta}$ reading.<br>ii. Measured directly from inclinometer. |
| $\dot{\theta}$ | Angular velocity | i. Measured directly from gyroscope.<br>ii. Estimated by differentiating inclinometer's $\theta$ reading. |

Angle and angular velocity are the state variables of interest to biped robot applications. The two sensors for reading angular state variables that are investigated in this experiment are the **SEIKA Inclinometer N3 (Analogue output model)** and **Hitec GY-130 piezo rate gyro**.

i. Reading the gyroscope

The Hitec GY-130 Piezo Gyroscope is a single rate, single axis gyroscope, designed for use in remote control helicopters and fixed wing aircraft applications [34]. The gyroscope is intended to be placed between a remote controlled aircraft's receiver and its rudder servo, as shown in Figure 5-20.

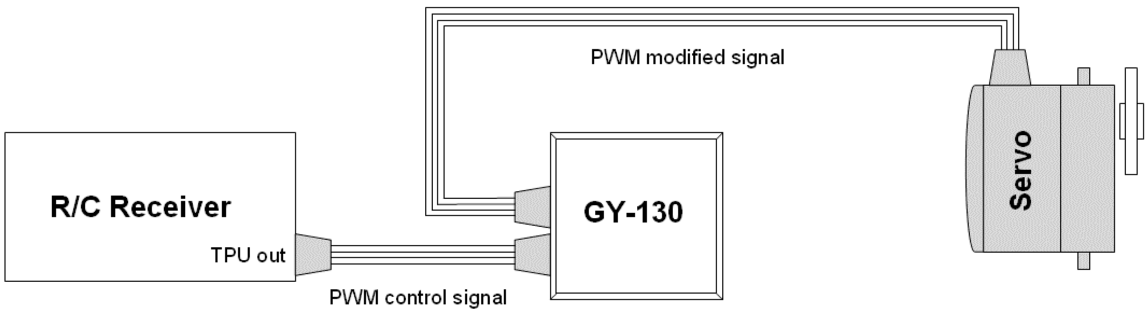


**Figure 5-20 Intended sensor configuration for GY-130 Rate Gyro. The gyroscope is used to counteract rotation of the aircraft by adjusting the signals being sent to the rudder servo.**

The gyro is used to adjust the Pulse Width Modulated (PWM) control signal being sent to the rudder servo by an amount proportional to the angular velocity measured by the sensor. In this way, the gyroscope makes the aircraft easier for a pilot to control, as it works to automatically dampen rotational velocity experienced by the aircraft.

In order to use the gyroscope to provide measurements of angular velocity to a controller, the physical configuration must be changed. Instead of using the gyroscope to adjust a servo control signal, the gyroscope output can be fed back into the EyeCon controller, as shown in Figure 5-21:

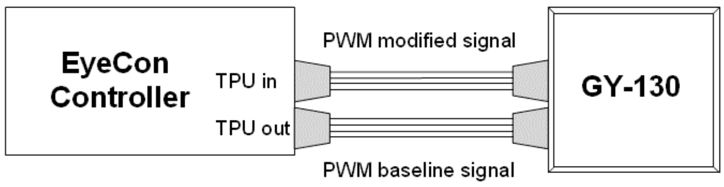


**Figure 5-21 The EyeCon controller can obtain an estimate of angular velocity by comparing the modified PWM signal output by the gyroscope to a baseline signal being sent to the gyroscope's input.**

During operation the EyeCon's output TPU channel PWM signal is held to a steady, baseline value. The gyroscope modifies the PWM signal by an amount proportional to the angular velocity experienced by the sensor. Since the baseline signal is being held steady, any changes in the PWM modified signal will be proportional to its estimate of angular velocity.

Assuming the velocity measurements are conducted frequently enough, an estimate of angular displacement can then be obtained by integrating the velocity signal.

In order for the velocity and displacement estimates to be useful, the gyroscope signals need to be calibrated. For the initial calibration of the gyroscope:

- The gyro signal for zero angular velocity ($\dot{\theta} = 0$) is obtained by holding the sensor steady for a large number of samples, and averaging the sensor output during calibration.
- The gyroscope is then rotated through 90 degrees, and the angular velocity ($\dot{\theta}$) signal integrated as the gyroscope moves, giving the scale of the gyroscope's angular ($\theta$) measurement.
- Finally, the gyroscope needs to be moved to a known position before the trial begins, so the angular displacement determined by the sensor can be used to provide an angle estimate.

### ii. Reading the inclinometer

The Seika/NTT N3 Inclinometer is a capacitive liquid based sensor with integrated sensor electronics [33]. The model being investigated here is an analogue instrument; however the manufacturer does supply a digital version.

The sensor output is a linear analogue signal, providing an estimate for tilt angle while the sensor is operating between (-30, +30) degrees. Outside this range the sensor output is no longer linear. The inclination estimate of the sensor is based on a reading of the linear acceleration vector acting on the sensor. The sensor assumes this acceleration is caused by the gravitational effect of the earth[17]. If additional linear acceleration components are acting on the sensor (if for example, the robot is moving), the sensor accuracy will be reduced.

The inclinometer's analogue output can be easily read using the RoBIOS function **`OSGetAD()`**. Since the inclinometer only requires a single analogue input, potential EyeCon problems with switching analogue input channels will be avoided[18].

Angular velocity can be derived from the angle estimates by differentiating the sensor signal with respect to time. Care must be taken, since differentiating a noisy signal will result in magnified noise in the derived signal [26].

One major advantage the inclinometer sensor has over the gyroscope is that there is no need to calibrate the sensor. The angular measurement is an absolute angle, and the range of sensor output values is invariant.

[17] Although the inclinometer's specification sheet does make the point that the sensor will also work on the surface of the moon!

[18] In Chapter 4.4.4, an issue with accessing sensor data when switching analogue input channels was encountered. At the time this experiment was conducted, this fault could still be reproduced. It is my understanding that the issue has been recently identified and corrected.

iii. Raw sensor data

Raw angular displacement readings obtained from both of these sensors, while the robot moves between +/- 30.0 degrees can be seen below, in Figure 5-22.

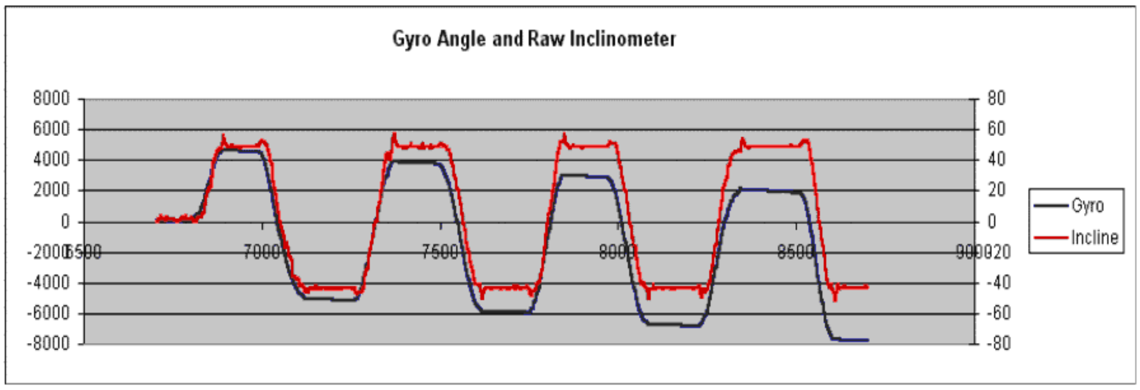


**Figure 5-22 Raw sensor output, read from the gyroscope and inclinometer. The inclinometer signal is noisy and lags behind the gyroscope output, while the gyro signal "drifts" over time.**

There are some obvious problems exhibited by the raw sensor output, which may impact the usefulness of the sensors. Sensor specific problems encountered include:

- Inclinometer noise: The inclinometer readings can display a significant amount of noise, especially during and shortly after changes to angular velocity, as can be seen in Figure 5-22.
- Inclinometer "lag": The inclinometer readings lag behind the gyroscope by about 50 ms. Figure 5-23 compares trends of both sensor measurements as the Ballybot robot begins to fall forward. The portion of the trend examined follows the inclinometer output as it moves through its linear range of (-30, +30) degrees.

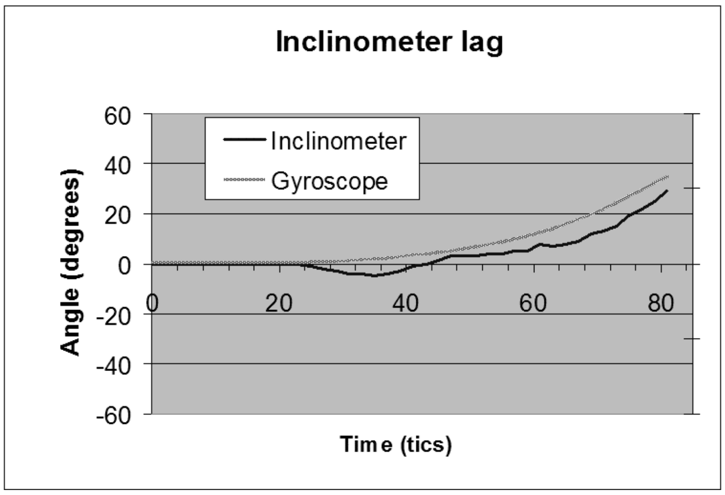


**Figure 5-23 The inclinometer signal appears to lag behind the gyroscope by about 5 tics (approximately 50 ms).**

- Gyroscope "velocity drift": Figure 5-24 shows a trend of the raw gyroscope output data, while the sensor is being held still. Over time, the "zero velocity" signal received from the gyroscope drifts away from its initial “zero” reading. This means that the sensor’s estimate of angular velocity, and therefore angle becomes increasingly inaccurate after a short time.

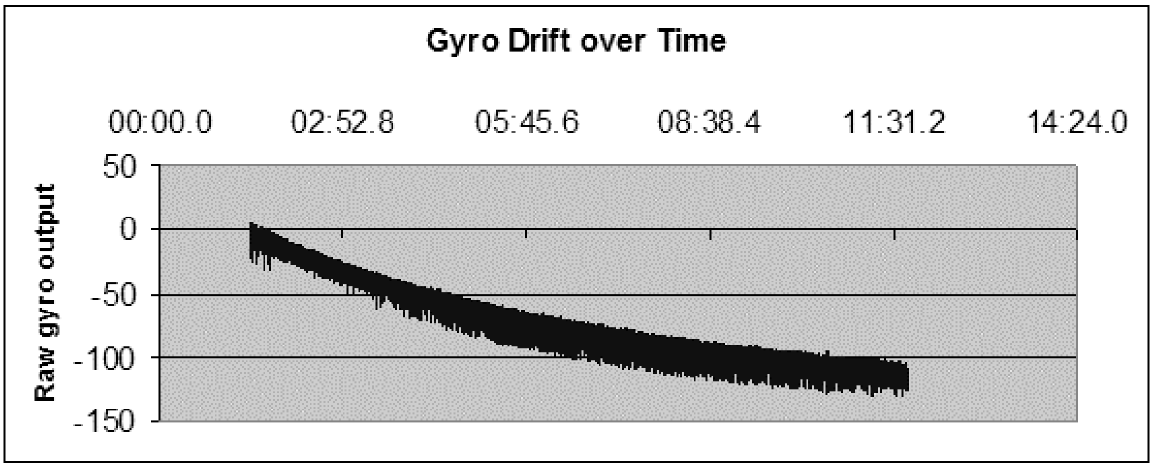


**Figure 5-24 The gyroscope output signal "drifts" over time, even though the sensor itself is being held still. This gyroscope “drift” results in an increasing error in angular velocity estimation. Since the angle estimate is generated by integrating the velocity signal, a small velocity error quickly becomes a large angular error.**

#### 5.7.3.3 Experimental procedure

The procedure followed to investigate various sensors was to iteratively perform the following steps:

1. Propose candidate sensor and data processing algorithm for obtaining state estimates. Each combination of sensor(s) and algorithm is referred to as a sensor "module".
2. Install sensor module on Ballybot robot.
3. Operate the robot, manually tuning the control loop parameters $(w_1, w_2, w_3, w_4)$, trying to balance robot.
4. If no success, then alter sensor or data processing algorithm and repeat.

Tuning of the parameters is a quick process, and should take no longer than 20 minutes or so for a trial. If it is taking longer, and the robot does not look like it is about to balance, then the proposed sensor and/or data processing algorithm are inadequate.

### 5.7.4 Results

The results presented here demonstrate the process by which inclination sensor modules for the Ballybot robot were investigated, finally culminating in a proposed sensor configuration to be used in subsequent biped robots.

#### 5.7.4.1 Balance using gyroscope only

The first experiment makes use of the Gyroscope reading of angular velocity, and the derived angle estimate, to control the robot. Samples of angular velocity are taken from the gyroscope every 10 ms, and this value is integrated to provide an estimate of the current angle. The controller samples the measured state variables every 30 ms to determine the sequence of control signals to apply to the robot motors.

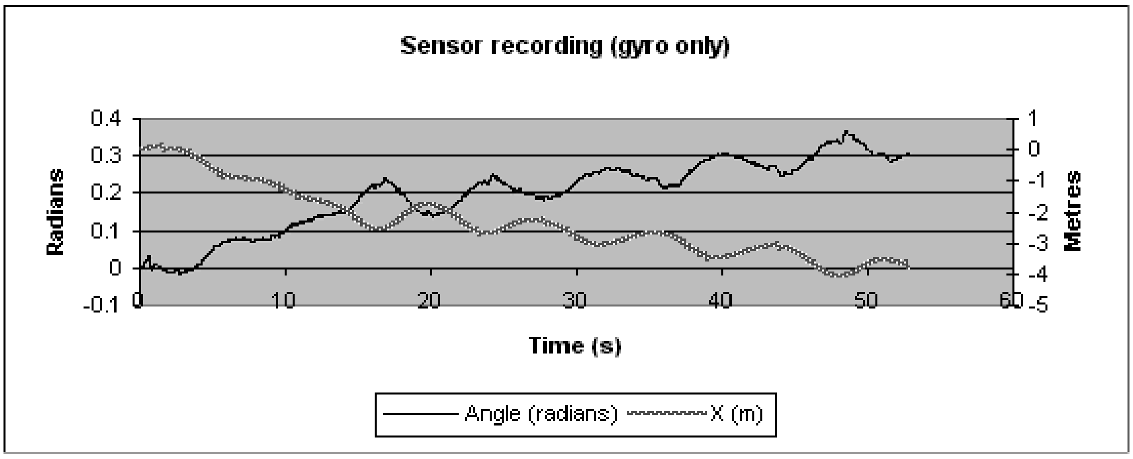


**Figure 5-25 Sensor recording of control by gyroscope.**

The resulting behaviour is a robot that maintains its balance well, remaining upright indefinitely. As time progresses, the robot begins to wander from the origin point, until it moves out of the workspace or collides with an obstacle. Figure 39 shows a recording of measured angle and displacement state variables during the course of an experiment.

Observations:

An examination of the sensor readings in Figure 5-25, suggests that the cause of the robot's increasing displacement is the gyroscope "velocity drift" observable in Figure 5-24. This velocity drift introduces an increasing error to the gyroscope's estimate of angle. Importantly, the robot does not fall, because as this error increases, the robot's displacement increases to compensate:

From equation (1), we see that the balancing force calculated by the control system is:

$$f_k = w_1\, x_k + w_2\, \dot{x}_k + w_3\, \theta_k + w_4\, \dot{\theta}_k \qquad (6)$$

And so the displacement observed during the experimental trial is proportional to the error in gyroscope angle estimate:

$$\theta_{error} \approx -\frac{w_1 x}{w_3} \tag{7}$$

It may be possible to correct for this error using the robot displacement to modify angle. This kind of solution would not be portable to other kinds of balancing robot[19], as it relies on the use an inverted pendulum balancing model, as well as the ability to accurately measure displacement.

#### 5.7.4.2 Balance using inclinometer only

In the second experiment, the inclinometer is used to provide a measure of robot angle. Using this approach the robot is unable to balance, as corrective actions in response to changing angles were applied too slowly. The response of the system appeared to be sluggish. This is likely due to the $\approx 50$ ms lag that can be observed in the inclinometer output (Figure 5-23).

Note that it is possible that the lag and noise observed in the inclinometer output signal may not have been caused by the inclinometer sensor itself. Instead these problems may have been introduced by the EyeCon controller's analogue input hardware. Another possibility is that the robot power source providing a reference voltage to the inclinometer may not be constant, as the same battery used to drive the motors also provides power to the robot's sensors. Experiments to investigate these scenarios were not undertaken.

#### 5.7.4.3 Balance using gyroscope and inclinometer cooperatively

The third experiment in this series uses the gyroscope control system described in 5.7.4.2 and the inclinometer co-operatively. The one real advantage the inclinometer

[19] It is, however, suitable for use in biped robots controlled using the “Torso Driven Walking” control system.

sensor has over the gyroscope is that its output signal for a given angle does not drift over time. This property is used to implement an algorithm that continuously recalibrates the gyroscope while the gyroscope is used by the controller to actively balance the robot.

The online recalibration algorithm is summarised below:

1. Use the gyroscope to control the robot motion, exactly as in experiment 5.7.4.2.
2. Monitor the variance of the inclinometer signal, while the robot is operating.
3. When a low variance is observed, the algorithm assumes that a reliable angular measurement has been taken from the inclinometer, and that the robot angle is not changing currently.
4. Whenever receiving a good signal, the inclinometer measure of angle is used to recalibrate the gyroscope angle and angular velocity.
5. The algorithm includes a minimum interval between recalibrations, to allow the control system time to adjust to large changes in estimated angle when errors are corrected.

Results of this experiment can be seen in Figure 5-26:

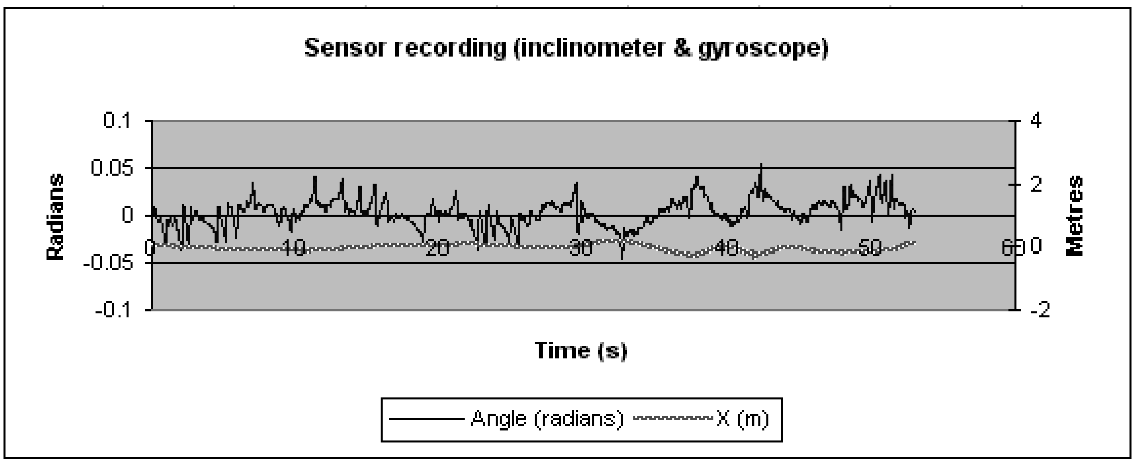


**Figure 5-26 Cooperative sensor control of the Ballybot, results in both a balanced and centred robot.**

The robot balances as well or better than the system described in 5.7.4.2. However, due to the online recalibration of the robot angle the drift in displacement is also minimised. A video of the robot balancing with this control strategy can be seen at this URL:

http://robotics.ee.uwa.edu.au/eyebot/mpg/balancing/ballybot1b.mpg

This combined sensor approach should prove portable to other varieties of balancing robot, including biped robots, since its estimate of robot angle and angular velocity relies solely on measurements taken from the gyroscope and inclinometer sensors.

### 5.7.5 Discussion

In this series of experiments, the Ballybot experimental platform was used to investigate sensors and sensor data processing algorithms for tilt angle detection in balancing robotic systems.

Of the sensors that were investigated, the Hitec GY-130 piezo rate gyro proved to be most effective in providing accurate and timely estimates of robot angle and angular velocity. This was the only sensor which the Ballybot control system was able to use on its own to successfully balance the robot.

The gyroscope does exhibit its own idiosyncratic problems. The most obvious of these is a drift in its measurement of "zero" angular velocity. Despite this problem of "gyro drift", the BallyBot's "Torso Attitude Controller" control system is able to maintain balance due to the fact that changing robot displacement can counteract angle measurement errors – as long as the error bias does not change too quickly. Instead of falling over, the robot simply moves away from the centre of the workspace.

Gyroscope drift, or bias, is a well known problem. Most solutions require a second sensor to be used to provide an absolute reference signal for the gyroscope. For example, Roumeliotis [36] describes a system which tracks the position of the sun to recalibrate a gyro signal. The SEIKA N3 inclinometer is used to provide an absolute reference signal, allowing on-line recalibration of the gyroscope. When using the inclinometer to recalibrate the gyroscope, the robot is able to both remain balanced and keep to the centre of the workspace.

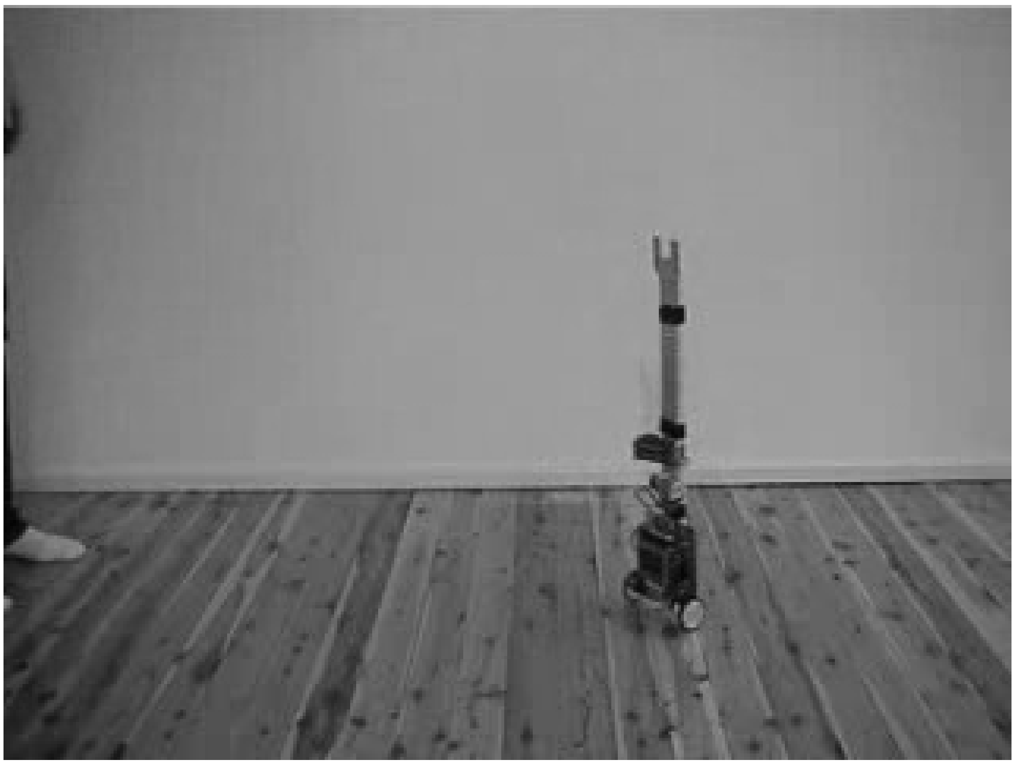


**Figure 5-27 Ballybot robot balances in the dining room, using the gyroscope for active balance and the inclinometer to perform on-line recalibration of the gyro.**

Due to the success of the gyroscope/inclinometer sensor package in balancing the Ballybot robot, the same configuration of sensors should be suitable for more general Torso Driven Walking applications.

## 5.8 Comparison with a "first-principles" adaptive control system

**Abstract:** Using the Ballybot software simulation[20], an artificial neural network based adaptive control system was developed to balance the Ballybot robot. The resulting control system is an example of a "first-principles" adaptive control system, such as those described in Chapter 2.3.6. Initially, the control system knows nothing about relative quality of various pendulum states. All this knowledge must be found by trial and error during training of the networks, while all performance feedback is limited to a single error signal when the pendulum has crashed.

The development of this alternative control system allowed the relative advantages of such an approach to be compared with that of the Torso Attitude Controller sub-system of the Torso Driven Walking control system. While the adaptive approach successfully learns to balance the robot, with no preconceived or pre-programmed notion of how this should be achieved, significant disadvantages were encountered. Most significantly, the control system requires a very large number of training cycles and does not always converge to a satisfactory solution

### 5.8.1 Introduction

In this experiment, the Ballybot simulation described in 7.4 is used to investigate the feasibility of applying a fully adaptive control system to the problem of biped robot control. If an adaptive approach can successfully control the Ballybot simulation, then perhaps the technique could be extended to a more complex biped robot system.

The resulting "first-principles" adaptive control system is used as a comparison to the more conventional solution used by the Torso Driven Walking control system in its Torso Attitude Controller subsystem. If the adaptive approach were to prove

[20] Described in Appendix B.

advantageous in the simple "Ballybot" environment, the experiment could be extended to the more complicated legged robot platform.

The inverted pendulum has become a classic subject for studies into adaptive control of non-linear systems. There are a large number of published approaches to adaptive control of inverted pendulums, including [11-22]. A common goal of many of these systems is to learn to balance the inverted pendulum without being given any external assistance. For instance, the system should learn for itself that keeping the pole as upright as possible is a good way to avoid falling over.

### i. Problem specification

An adaptive control system is desired that, beginning from a state of complete ignorance, can learn to balance the simulated Ballybot robot by applying a variable horizontal force to the base of the robot.

The control system must learn the task of balancing the robot even when all feedback regarding the quality of its control decisions is limited to a single failure signal, raised when the pendulum falls or moves too far from the centre of the robot's workspace.

In addition, the system must be able to operate with a large enough interval between state samples to permit some rudimentary sensor data processing, as final our objective would be to transfer the application to a physical robot. As a result, the control system is constrained to operate satisfactorily with a minimum sample rate of 0.25 seconds, an arbitrarily selected time interval that should allow time for some simple sensor interpretation or other processor tasks.

Finally, the control system must be robust enough to deal with some degree of sensor noise. One of the objectives of this investigation was to determine the level of accuracy with which sensors mounted on a real robot will need to determine the robot system state.

The control system described here is an example of the evaluator/predictor class of pendulum controllers, composed of two co-operative BPN networks. One network learns to evaluate the "quality" of an observed system state, and the other learns to predict the likely resulting state after a given control action is applied to the observed state. The experiment demonstrates that given the selection of appropriate training data this kind of simple BPN controller can learn to balance an inverted pendulum.

## ii. Prior work

There has been a significant amount of research into adaptive systems for balancing pendulums, since Donaldson [12] in 1960. Initial solutions required considerable a priori knowledge both of the dynamics of an inverted pendulum system, as well as knowledge of what constitutes the 'ideal' system state. The success of these approaches depends heavily on the accuracy with which the dynamic equations, used to determine a solution, model the real system. Obviously in the case of a simulation we know the exact equations driving the inverted pendulum system; however it is unlikely that these models will describe a real world system with the accuracy required to satisfactorily control the robot. This problem becomes compounded when the robot to control is more complex than an inverted pendulum, as is the case in even the simplest bipedal robot.

As described by Anderson [13], early applications of neural networks to the problem of balancing inverted pendulums (such as Widrow [14], [15]) first built a traditional control system to balance the pendulum, and then trained a neural network to mimic the results obtained from the controller. A drawback of this approach is that the network can never surpass the performance of the traditional controller used to train it. If a "conventional" control system is available, and the "adaptive" system's performance can only approach that of the conventional system, then there is no point using an adaptive system! All that we are doing is introducing unnecessary complication into the system.

An alternative method of training a network is to have it learn to mimic a human controller, who is capable of balancing the pendulum. In systems where an accurate model is unknown, or unavailable, this technique can result in an automated performance that approaches that of the human trainer. This approach has been investigated by a number of researchers, including Guez et. al [16]. Unfortunately, it is not always the case that a human is able to demonstrate the correct actions, especially in more complicated systems. In these situations, an automated method of evaluating the performance of a network must be implemented.

Many approaches rely on providing rules for measuring the quality of a given system state. For example, Connell [17] measures the system error as the distance between the current system state, and the robot's ideal state. The drawback here is that we need to know beforehand what the ideal state for the system is. This is obvious in the case of an inverted pendulum, where the robot should be upright, stationary and in the centre of the workspace. For more complex systems, the specification of an "ideal state" may not be so easy.

Anderson [13] presented a method of training a network using only a failure signal if the pendulum falls or moves too far from the origin. In this system two networks are trained, one is an evaluation network, which learns to judge the likelihood that a given state will result in a failure, and the other is an action network that determines the control signal to use in response to an observed state. Control signals are limited to the set of three possible values: {-f, 0.0, +f}, where f is the maximum horizontal force that can be applied.

Anderson contrasts his technique favourably with a system presented by Michie and Chambers [23], called BOXES. In this system, the entire state space that can be experienced by the system is quantised into regions, where an appropriate control action is learned for each region. For an inverted pendulum, the state space was divided into 162 regions. This approach can quickly become unwieldy for systems with a large number of state variables.

Hougen, Fischer & Johnam [18], have presented a system, based on a technique they call SONNET (Self Organising Neural Networks with Eligibility Traces), which they have used to control a real cart and pole. This system also limits its success/failure feedback to a single failure signal when the cart crashes. They limit the state information sampled to cart position and pole angle.

### 5.8.2 Materials

This experiment makes use of the Ballybot software simulator, described in Appendix B.

### 5.8.3 Methods

#### i. Control system

The adaptive Ballybot control system presented here is similar to Anderson's temporal difference approach [13], in that a single failure signal is used as feedback together with two neural networks. An interval of 0.025 seconds between samples was chosen in order to allow time for processing any (theoretical) sensor data. Additionally, the system is to determine the most appropriate controlling force, rather than limiting its controlling action to the “full left”, “full right” or “no force” choices used by Anderson in [13].

The adaptive Ballybot control system is composed of a “Control Manager”, responsible for coordinating the interaction between other control system components, and two simple back-propagation neural networks. These two networks are an “evaluation network” and a “prediction network”:

- The evaluation network is trained to assign a quality value to any observed or predicted system state, thus allowing the controller to judge one state to be “better” than another. “Better”, means that the more favoured state is less likely to result in the control system failing, or at least will take longer to fail.
- The prediction network learns to predict the state variable values that will be observed on the next sample, given the current system state and a proposed control input force. The final component of the system is a controller, which uses the prediction and control networks to select an appropriate control signal input force, given an observed system state.

The networks were implemented using a back propagation network algorithm described by Freeman and Skapura [24].

Figure 5-28 shows the control flow diagram for the adaptive Ballybot control system:

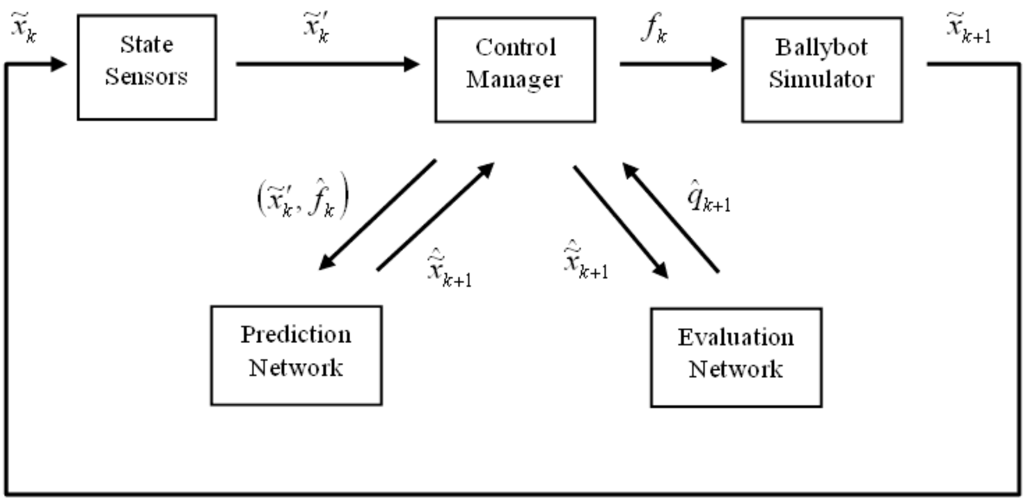


**Figure 5-28 Control flow diagram for the adaptive Ballybot control system. The control manager coordinates the use of prediction and evaluation neural networks to determine the most appropriate control force to apply in response to an observed system state.**

**Table 5-4 Control flow diagrams system variables.**

| | | | |
|---|---|---|---|
| $\tilde{x}$ | State vector of system: $\{x, v, \theta, w\}$ | $\hat{\tilde{x}}$ | Proposed control force to apply to pendulum. |
| $\tilde{x}'$ | Sampled system state (includes noise) | $\hat{q}$ | Evaluation of quality of a predicted state, $\hat{\tilde{x}}$. |
| $\hat{f}$ | Proposed control force to apply to pendulum. | $f$ | Control force applied to the pendulum |

## Prediction network:

The purpose of the prediction network is to learn the behaviour of the inverted pendulum system. It does this by continuously observing the relationship between the current system state, $\tilde{x}'_k$, the previously observed state, $\tilde{x}'_{k-1}$, and the previously applied control signal $f_{k-1}$.

The prediction network is a simple 4-layer BPN network, with 5 input nodes, 4 output nodes and 2 hidden layers with 20 neurons each.

- Each input neuron is associated with one of the four variables making up the current Ballybot system state, plus a proposed control force.

- Each output neuron is associated with one of the four variables making up the predicted Ballybot system state.

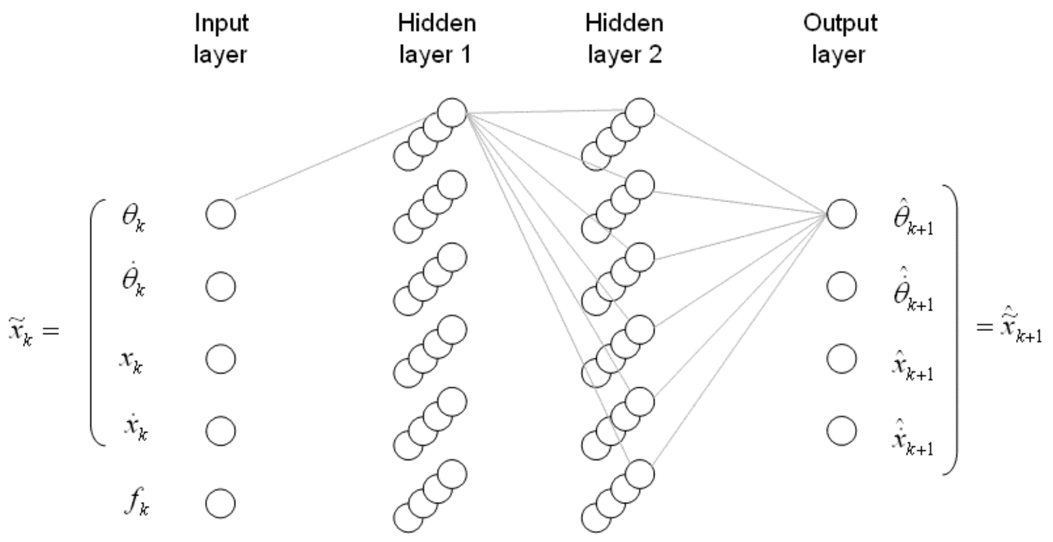


**Figure 5-29 Network topology of the Prediction Network. Each neuron in a preceding layer is connected to every neuron in the next layer by a modelled synapse. Most of these connections have been left off the diagram for clarity.**

### Evaluation network:

The purpose of the evaluation network is to provide a way to compare a collection of predicted system states. This evaluation ability allows the control manager to select a controlling force that causes the system to move to the most desirable of the predicted system states. The evaluation network will rate a state as more desirable if it decides the system is less likely to receive a failure signal soon after experiencing that state.

The evaluation network topology is a 4-layer BPN network, with 4 input nodes, a single output node and 2 hidden layers with 15 neurons each.

- Each input neuron is associated with one of the four Ballybot system state variables: $(\theta, \dot{\theta}, x, \dot{x})$.
- The output neuron is trained to provide a scalar value representing the quality of the input state: (0 = poor quality, 1.0 = high quality).

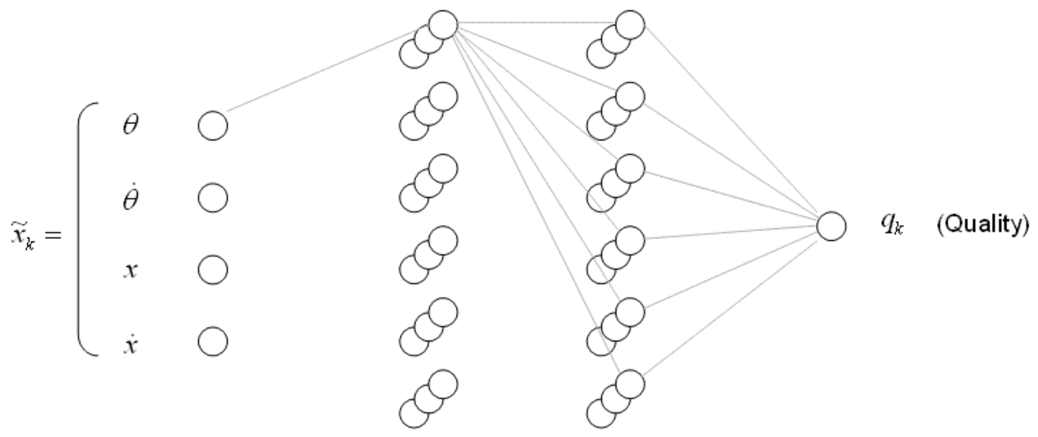


**Figure 5-30 Network topology of the Evaluation Network. Each neuron in a preceding layer is connected to every neuron in the next layer by a modelled synapse. Most of these connections have been left off the diagram for clarity.**

As each trial begins, the control manager creates a state evaluation queue, see Figure 5-31. The queue size is initially fixed to a small value; I used an initial size of *N*=12 samples. As each state observation is passed to the control manager, the new sample is added to the head of the state evaluation queue.

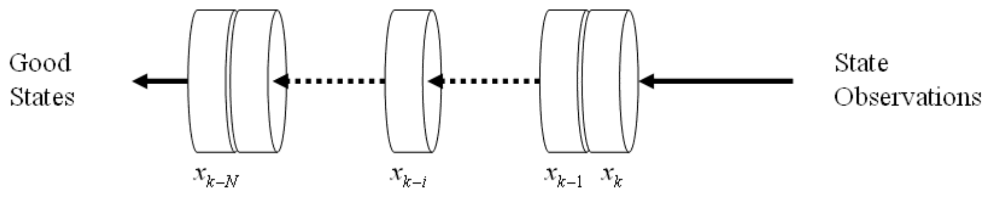


**Figure 5-31 The state evaluation queue. New state observations are fed into the head of the queue. As states grow older, they are assigned higher quality scores. If the trial lasts long enough for samples to fall out the back of the queue, they are given the maximum quality score of 1.0. If the system crashes, all states currently in the queue will be assigned a varying quality.**

As the system operates, training data is collected and stored in a database of 20,000 input/output relationships:

1. Any state that survives long enough to fall out the tail of the queue is at least *N* samples old. The control manager assumes this is an example of an 'ideal' state, which does not result in a failure, and stores a training input/output relationship for the network that matches the state $\tilde{x}'_{k-N}$ with a quality of 1.0.

2. If a failure is detected, all states currently being held in the queue are stored as training samples resulting in output values of varying levels of quality. The quality assigned to a particular state varies linearly with its queue position. State $\tilde{x}'_k$ is matched with a quality of 0.0, while state $\tilde{x}'_{k-i}$ is stored with a quality of $(i/N)$.

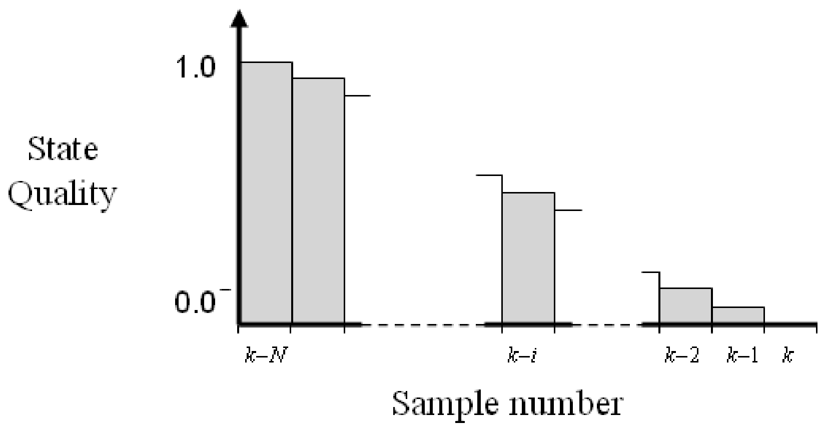


**Figure 5-32 State samples in the state evaluation queue are assigned a quality value, which increases linearly from zero to one, as the state moves further down the queue.**

As the controller becomes better at balancing the pendulum, the average number of state samples taken before a failure should increase. If the state queue length is held constant, the evaluation network cannot differentiate between states that always fail after $N$ samples, and the performance of the system stagnates. To continue to improve the performance of the network, the length of the state queue must be gradually increased as the controller performance improves.

**Control manager:**

The control manager is a simple algorithm that co-ordinates the interaction between the various components in the system controller. Tasks performed by the control manager include:

1. Accept samples of the system state and store them in the state evaluation queue

2. Add any states that have passed entirely through the queue to the evaluation network's training data set, with a quality value of 1.0.
3. If a failure signal has been received, add all states currently in the state queue to the evaluation network's training data set, with appropriate quality values.
4. Collect observations of the change in system state, as a result of the previously selected control force, and store them in the prediction network's training data set.
5. Determine the control force to use in response to an observed state by:
    a. Iterate through the range of possible control forces (I used the range -3N to +3N, considering forces with a granularity of 0.6N).
    b. For each possible control force, use the prediction network to predict the state that would result from application of the force.
    c. Use the evaluation network to determine the quality of the predicted state.
    d. Select the control force that results in a predicted state with the maximum evaluated quality as the control force to apply.

ii. Training the networks

Perhaps the greatest difficulty in applying BPN networks to problem solving is the selection of what training points to present the networks. Through trial and error, I have found the proposed control system to be very sensitive to the selection and quality of training data. Some of the lessons learned include:

- The system states that do not result in a failure signal should be limited to far less than +/- 90 degrees. If the pendulum passes through 30 degrees, the situation is usually irrecoverable. Rather than cause the prediction networks to needlessly learn the system behaviour in these extreme regions, the learning rate can be improved by limiting the state space size.

- An initial training period must be performed where the control action selected in response to an observation is a random force. If this is not done, I found the system usually selects a maximum magnitude force to apply, and never chooses another force. As a result, the prediction network can never learn how the system behaves when experiencing other control forces, so the system does not learn to balance a pendulum.
- Training data points should not be presented to the networks as they are observed. Rather, the observations should be collected in a set of training points, from which a random sample is selected to train the network. This is necessary due to the fact that consecutive observations are very similar, and if they are all applied in order, this can cause the network to "forget" earlier training by "over-learning" the more recent samples.
- The initial conditions for the state variables during training must be randomised. If not, the system can incorrectly place emphasis on the wrong state variables. For example, if we do not randomise the $x$ parameter, we will find that the system places too much emphasis on the value of $x$. This is because the system always starts with $x=0$, and by the time it fails, $x$ has always changed.
- As the performance of the system improves, we need to increase the size of the sample queue. This is because any samples that come all the way through the queue are treated as having an equivalent quality. As the system performance increases we need to be able to differentiate between these states or our system performance plateaus.
- Finally, during one of the training runs, the system will manage to balance the pendulum indefinitely. This can be dangerous, as if the trial is not halted, the

networks will continue to train with data limited to a very small subset of possible system states, which can result in the networks "forgetting" how to cope with a more general set of possible states. This problem is mitigated through the use of large training sets.

### 5.8.4 Results

The adaptive Ballybot control system is a relatively simple adaptive algorithm that learns how to control an unknown, non-linear, dynamic system while restricting training feedback to a binary error signal when a trial fails. After approximately three simulated days of continuous training, the system learned to balance a pendulum satisfactorily.

In addition, the controller can continue to balance the pendulum while experiencing sensor noise (simulating vibrations or sensor errors) to the system state. Randomly distributed sensor noise of up to +/- 2.3 degrees could be added to the robot angle measurement without dramatically reducing the reliability of the control system. The system's reliability reduces if the maximum noise level increases past this level.

One of the advantages of using a simple system such as the Ballybot to develop the adaptive control system is that the resulting behaviour can be easily analysed. Figure 5-33 shows the control profile of the system as the most critical state variable, robot inclination angle ($\theta$), is varied from -30 to +30 degrees. All other state variables have been held at zero units and are omitted from the diagram.

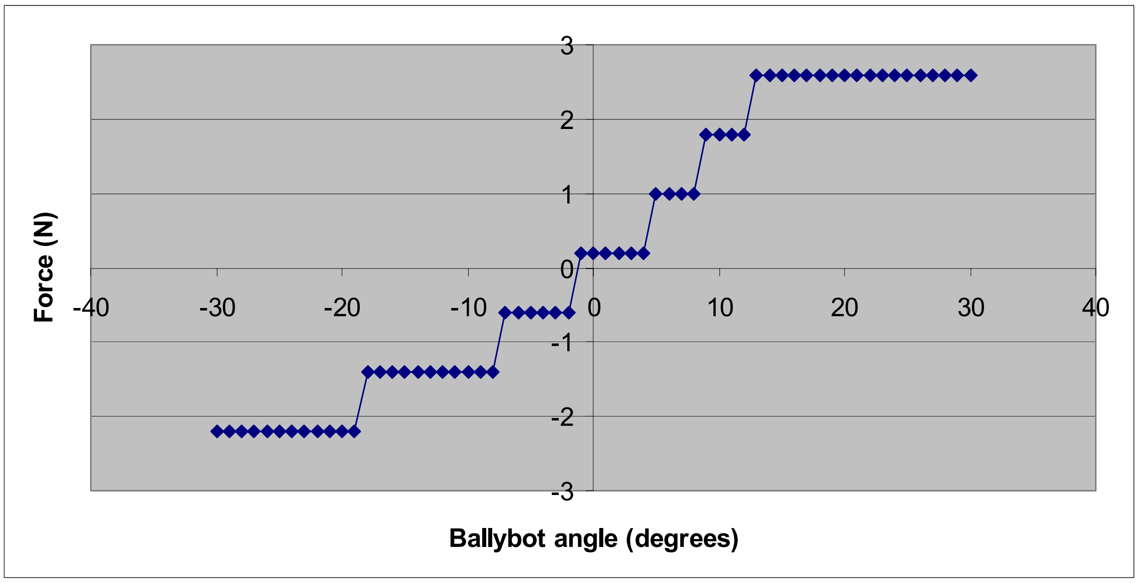


**Figure 5-33 Balancing force applied by the adaptive Ballybot control system, as inclination angle varies. The inclination angle is varied from -30 to +30 degrees, while all other state variables are held at zero units.**

Interestingly, the profile exhibited in Figure 5-33 is very similar to that of the classical solution for an inverted pendulum controller investigated in Experiment 5.5. In the classical solution, applied force is linearly related to each of the state variables. In the scenario depicted by the chart, where one state measurement is varied while the others are held steady, the classical solution would be a sloped line passing through the chart origin.

The step changes in force are an artefact of the way the state controller iteratively examines possible force values, using the prediction and evaluation networks. They could be eliminated by reducing the granularity with which potential control forces are examined. This would have unnecessarily increased the number of calculations needed to determine each control interval's balancing force, reducing processing time that would otherwise be available for the CPU to perform other tasks.

### 5.8.5 Discussion

In this experiment, a simple back propagation neural network based control system that learns to balance a simulated inverted pendulum is introduced. The controller has no a priori knowledge about the system model of a pendulum, and limits all feedback of its performance to a single failure signal when the system crashes.

This approach has a number of advantages over many other techniques, which make it appealing, including:

- Simple to understand and implement.
- Requires no knowledge of the robot model.
- Does not require resource intensive state space segmentation, making it a suitable control algorithm for staging on often-constrained robotic hardware.
- Over time, the controller can adapt to changes in the system dynamics.

However we have also encountered some serious disadvantages, including:

- System requires a long period of training, which will be difficult to endure when training on a real inverted pendulum system.
- Training points applied to the neural networks must be carefully managed in order for the controller to learn the task.
- System training "trials" need to be initialised with a random system state. This could prove difficult to achieve when training on a real robot.
- The system does not reliably converge to a solution. In fact most trials fail to result in a solution. If the approach cannot reliably balance a simple single degree-of-freedom robot, it is unlikely to give a solution for a much more complex biped robot.

The long training times will be a significant disadvantage when staging the system on a real robot. Starting from a completely random state, our simulation took three days of “training time”, to learn to balance a pendulum. During this time system state samples were taken every 0.025 seconds, resulting in approximately 10 million pairs of training vectors being applied to the neural networks. The large number of trials conducted would have worn out a real robot (and its researcher) long before it learned to balance!

This disadvantage can be overcome by training the controller on a computer simulation before transferring it to the real system. However, to build an accurate simulation we need a good understanding of the behaviour of the system we are attempting to control…which defeats one of the major motivations for using an adaptive control system. Another possibility is to perform initial training off-line using an approximated system model, and complete the training on-line with the physical robot. In Sutherland [25], it has been shown that a neural network based controller trained using a simplified system model can continue to operate when applied to a more complex system. Over time, the adaptive nature of the system allows the controller to optimise its performance in the new environment.

### 5.8.6 Comparison with Torso Driven Walking

The “Torso Attitude Controller” is a conventional control system, which can be used to balance the Ballybot robot. In Chapter 5.5, the TAC is implemented on the same software simulation system that was used to develop the “first principles” adaptive system described in this experiment. Since the systems are implemented on the same platform, their relative advantages can be easily compared.

Both control systems are capable of balancing the Ballybot robot, and handle a moderate level of noise. In fact, as was shown by Figure 5-11, the adaptive system’s learned behaviour is very similar to the TAC control outputs for the same input data. The major advantage of the conventional Torso Attitude Controller over the adaptive system presented here is its simplicity.

The Torso Attitude Controller subsystem can be manually configured to balance a simulated inverted pendulum within a couple of minutes. This contrasts very favourably with the minimum of three days continuous training required by the "first principles" adaptive control system. It its defence, the adaptive system has a much more difficult problem to solve – it needs to learn to predict the response of the system, as well to distinguish good system states from bad ones. The conventional Torso Attitude Controller has been built leveraging existing knowledge about the system behaviour and what a "good" state is.

Another advantage the Torso Attitude Controller subsystem has over the ANN approach is that its behaviour is immediately apparent from analysing the control system's mathematical model. The behaviour of the ANN system is encoded by the many varying network weights, obscuring the details from an observer. On balance, the experiments have demonstrated that a conventional system is usually a more desirable approach, as long as a model of the system being controlled is available.

## 5.9 Experimental review

The Ballybot has been developed in a modular fashion, with each component of the system being fully tested before moving on the next step. The process followed began with the development of a software simulation for investigating the system behaviour, and was completed with the development of a control system and sensor package for maintaining the balance of the robot.

1. Developed a dynamic software simulation of the Ballybot system.

2. Implemented the "Torso Attitude Controller" subsystem of Torso Driven Walking using the simulation created in step 1.
3. Experimentally developed a DC motor model, describing the relationship between applied voltage, resultant balancing force and the robot base velocity.
4. Using the Torso Attitude Controller subsystem (step 2) and DC motor model (step 3), a number of sensor configurations were evaluated for detection of robot tilt angle and angular velocity.
5. Finished with a balancing robot, and a sensor package easily transferable to more complicated robotic platforms.
6. For comparison with the Torso Attitude Controller, a "first-principles" back propagation neural network based adaptive control system was developed using the simulation created in step 1.

All of these experiments were performed using the same controller, sensors and motors that were then used to develop a biped balancing robot, the Legbot discussed in Chapter 6.

Dividing a complex problem, like a biped robot, into modules that can be independently tested in controlled environments is an effective design process. The Ballybot robot achieved two important goals towards being able to implement a biped robot:

- A reliable sensor module has been developed and tested in a real-world environment.
- The central component of my proposed "Torso Driven Walking" control system has been tested and validated in a real-world environment.

In both cases, the system components that have been developed were implemented and tested using the same hardware and software components as will be used in the walking biped robot. All that remains is to add legs, which is the subject of Chapter 6.

The Ballybot robot provided an ideal platform for experimenting with two of the major categories of control system design: adaptive and conventional control systems. Despite the popularity of adaptive systems for the control system designs, the experiments implementing an ANN based controller for the Ballybot[21] demonstrate how inefficient working with such systems can often be.

When available, classical solutions to control problems are likely to be far more efficient than adaptive approaches. Considering that the conventional "Torso Attitude Controller" control system implemented on the Ballybot could be completely described by a single equation in four variables, a simpler adaptive system cannot be found.

In the next chapter, Chapter 6 "Experiments with the Legbot", an alternative use for adaptive systems in biped robot control system design is investigated. Rather than attempt to implement an entirely adaptive control system, the adaptive design approaches are instead used to tune a conventional control system.

The Ballybot robot was used to develop and test a sensor "module" that is fast and accurate enough to keep an inverted pendulum balanced. The sensor module is composed of a rate gyroscope working together with an inclinometer. In operation with the Ballybot, the gyroscope output is used to actively balance the robot, while the inclinometer output is used to periodically recalibrate the gyroscope in order to correct gyro drift, or bias.

[21] Chapter 5.8, "Comparison with a "first-principles" adaptive control system".

There are some issues with using the inclinometer, namely that the robot must be stable and still for long enough for the inclinometer signal to become steady (and trustworthy). However, the "Torso Attitude Controller" (TAC) control system only needs the gyroscope bias corrected in order to maintain its set point position in the workspace. Gyro bias does not cause the robot to fall.

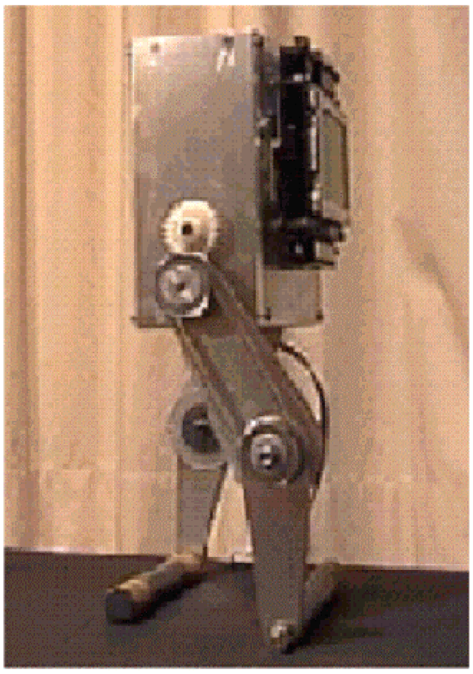
The "Torso Driven Walking" control system[22], of which the TAC is the central component, inherits this robust behaviour, as the only component of the biped control system that relies on an estimate of an inclination angle is the TAC.

This means that, even if the inclinometer is unable to provide regular re-calibration in the potentially high vibration environment of a biped robot, the symptoms observed by the robot will simply be to move away from the centre of the workspace to compensate for the gyroscope bias. Even in the worst case scenario, the proposed sensor module should not be a cause of the robot failing.

[22] The "Torso Driven Walking" control system is described in detail in Chapter 3, "Torso Driven Walking".

# 6 Experiments with the Legbot

This chapter details the design, construction and commissioning of a new legged robot, which has been given the creative name "Legbot". Based on the successful "Ballybot" robot[23], the Legbot was developed to continue the investigations into walking robots.



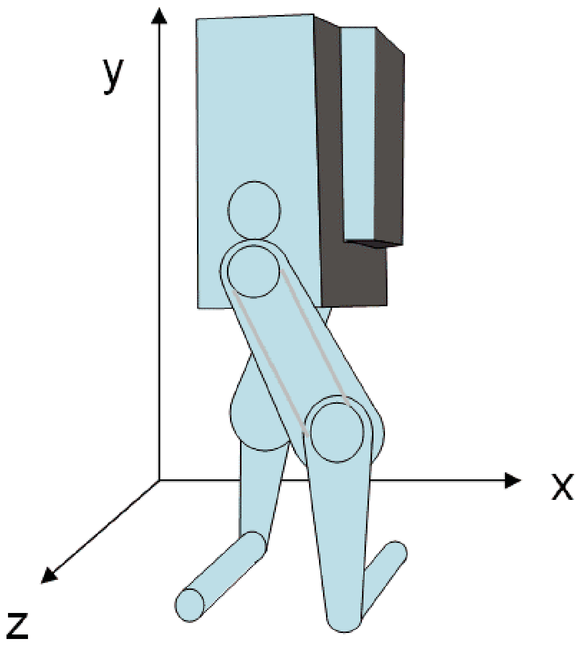


**Figure 6-1 Legbot robot, extends the Ballybot robot system by replacing wheels with a pair of simple legs. While the robot movement is constrained to the x-y plane through the use of very wide feet, in that plane the robot can fall freely, and must maintain dynamic balance in order to move.**

The primary purpose for the development of the Legbot was to investigate issues relating to the control of the robot's legs in dynamic walking control systems. Specifically, the robot was used to develop and test the "Torso Driven Walking" control system, introduced in Chapter 3. An additional goal was to prove that the sensor package developed using the Ballybot[24] robot can be used to successfully balance more complex robots.

[23] My experiments with the Ballybot robot are described in Chapter 5.
[24] In Chapter 5.7, "Sensors for balance".

## 6.1 Introducing Legbot

The Legbot is an autonomous mobile robot, with four actuated degrees of freedom. The philosophy used to develop the Legbot design was to extend the Ballybot robot by adding legs, while keeping the rest of the system as similar to the Ballybot as possible.

The Ballybot's wheels have been replaced by a pair of simple two degree of freedom legs. The legs are designed to have a wide profile in the frontal (y-z) plane, but to appear as "point" feet in the sagittal (x-y) plane.

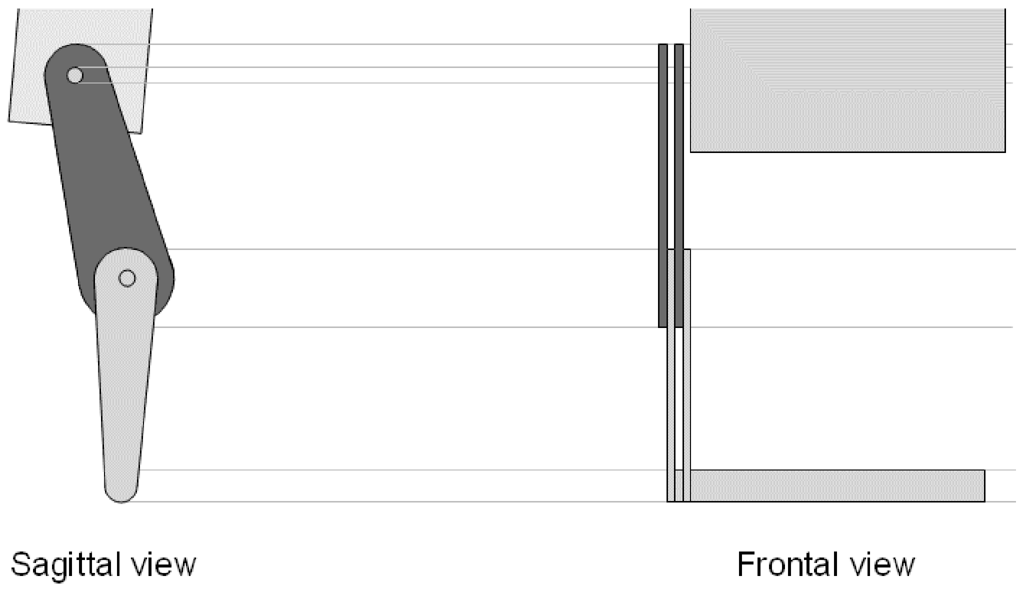


**Figure 6-2 The Legbot's two degree of freedom legs are designed to prevent rotation in the frontal plane, while allowing the robot to rotate freely in the sagittal plane.**

The leg shape constrains the robot motion to the sagittal (x-y) plane, effectively turning the Legbot into a "planar" robot. This was done to keep the system as simple as possible, given the already large increase in complexity introduced by the need to control the robot's legs.

Static, quasi-dynamic and zero moment point "dynamically stable" control systems[25] will not be able to balance the Legbot while it is walking, as there is no "supporting polygon[26] present when the robot stands on one leg. If the robot is going to walk successfully, it will need a fully dynamic walking control system, such as those based on inverted pendulum controllers. The "Torso Driven Walking" control system, introduced in Chapter 3, has been developed to provide the kind of fully dynamic control the Legbot requires.

Despite the obvious changes introduced by the robot's legs, the Legbot reuses many components that were first tested on the Ballybot robot, including:

- Controller hardware and software is identical on the two robot platforms.
- Robot actuators are DC motors on both platforms, although the motor specifications are different.
- The combination Gyroscope/Inclinometer sensor module developed using the Ballybot is installed on the Legbot robot.
- The Torso Attitude Control system, used to balance the Ballybot, is reused as a component of the "Torso Driven Walking" control system.

## 6.2 Motivation

### 6.2.1 Controlling robotic legs

Controlling a robot using legs is far more complicated than driving using wheels. The Ballybot robot was able to generate the ground reaction force it uses to maintain balance by simply changing the voltage being applied to the robot's wheels. In contrast, the Legbot robot not only needs to generate the same kind of balancing ground reaction

[25] These control system classifications are described in Chapters 2.2.2, 2.2.3 and 2.3.1.
[26] When a robot is balancing on one foot, the supporting polygon is the contact area between the foot and the ground (Chapter 2.2.1).

force, but must also make sure there is always a leg in position to support the robot's weight, and that the supporting leg exerts sufficient force to keep the torso away from the ground.

Compared to many legged robots, the Legbot's kinematic equations are very simple. Each leg is comprised of only two rigid links, connected to the torso, and each other, by a pair of revolute joints. Each leg joint is controlled by the coordinated application of input voltage to a pair of actuators, which supply the joints with variable torque.

In spite of the relatively simple design of the Legbot's legs, the robot still needs to solve many of the problems that would be encountered by more complex legged robots. For example:

- Regardless of the control system implemented, the Legbot robot control system will need to be able to accurately determine each leg's joint angles.
- In order to achieve positional or force control of the legs, the robot control system will need to be able to reliably supply an accurate control torque to the joints.
- This joint torque control will need to compensate for any friction operating in the leg joints.

The Legbot robot makes a good platform for investigating solutions to these problems, due to the relative simplicity of the robot design. Once solutions have been proven on the Legbot platform, they should be transferable to more complex legged robotic systems, in much the same manner that sensor configurations tested on the Ballybot were reused on the Legbot.

### 6.2.2 Balancing control systems for legged robots

The Legbot was designed to continue investigations into dynamically balancing control systems, which were begun using the Ballybot robot. Like the Ballybot, the Legbot is effectively a planar robot. As a result, development of the control system will be far simpler than would be the case for a robot that needs to balance in three dimensions. The development of a suitable software simulation is also simplified by being able to consider the two dimensional motion only.

In the sagittal (x-y) plane, the robot is essentially balancing on “point” feet. There is no effective supporting polygon[27], which means the robot is never “statically stable” in the sagittal plane. Like the Ballybot, the Legbot robot requires a fully dynamic control system to maintain balance.

The Legbot robot will be used to experimentally validate the “Torso Driven Walking” control system[28], which is an inverted pendulum inspired robotic control system. Some components of the “Torso Driven Walking” control system have already been experimentally validated; specifically, the “Torso Attitude Controller” has been tested using the Ballybot robot. The Legbot platform will provide the opportunity to develop and test an implementation of the full TDW control system.

Lessons learned from experiments with the planar Legbot robot should be applicable to more complex three dimensional robotic systems. Such a robot would be the next logical step in the progression of robotic experiments presented in this thesis.

[27] The “supporting polygon” is defined in Chapter 2.2.1.
[28] The “Torso Driven Walking” control system (TAC) is described in Chapter 3.2.5.

### 6.2.3 Genetic algorithms for tuning control systems

While the Ballybot was used to investigate one kind of adaptive control system (Artificial Neural Networks, in Chapter 5.8), the Legbot provides an ideal platform for experiments in another field of adaptive control, Genetic Algorithms. GA approaches lend themselves particularly well to the problem of tuning complex systems, where individual solutions can be described using a well defined data structure.

The "Torso Driven Walking" (TDW) control system being developed for the Legbot robot fits these criteria. While the TDW algorithm is fairly strictly defined, it requires a significant amount of tuning before it can successfully balance a walking robot. Tuning the control system involves the selection of over 50 floating point values relating to various control system parameters such as trajectory durations, transition triggers and joint set point positions.

The Legbot robot, and its associated software simulation[29], allows the appropriateness of GA tuning of the TDW control system to be evaluated experimentally. The approach taken is to first use GA to tune the control system offline, using the software simulation. Then, once a population of reasonable solutions has been evolved, issues relating to the transfer of the solutions to the physical Legbot robot are examined.

## 6.3 Materials

### 6.3.1 Draft robot design

Figure 6-3 displays an overview of the draft design developed for the Legbot robot:

[29] The Legbot software simulation is described in Appendix C.

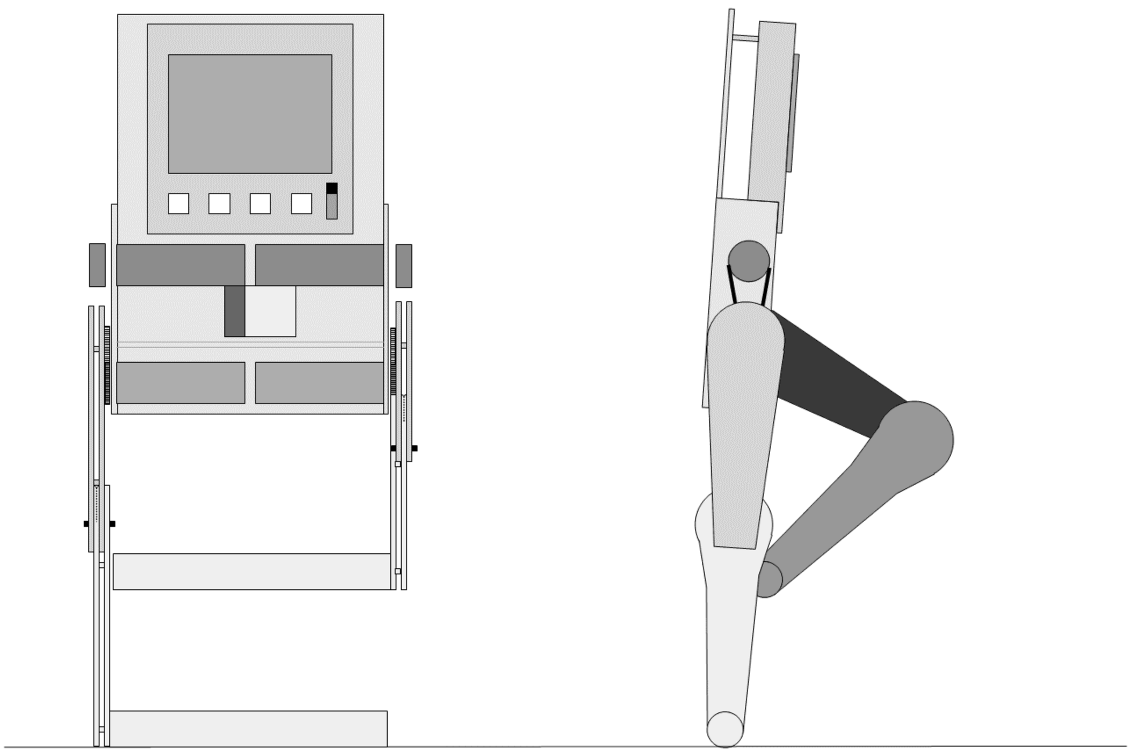


**Figure 6-3 Draft design for the Legbot robot. In an effort to keep the legs as light as possible, all controllers, batteries, motors and sensors are mounted in the robot torso.**

The draft Legbot design was produced with the following objectives in mind:

- Minimize mass of legs. In an ideal world, it would be possible to construct zero mass legs. This would reduce impact of moving legs on the balance of the torso.
- Mount all motors and batteries in the robot torso.
- Minimise total robot mass, by limiting the amount of aluminium used in the construction.
- Utilise "tendons" for knee joint actuation.
- Provide similar mountings for the EyeCon controller as were used on the Ballybot robot.
- Disassociate motors from the load bearing axels, so the motor shaft does not need to bear the weight of the robot.
- Design the legs and feet to constrain robot motion to the robot's sagittal plane.
- Minimise the number of degrees of freedom associated with the robot legs.

It has been suggested by some researchers in the area of biped walking (Koechling [48]), that in order to achieve smooth walking or running, a robot requires a large number of degrees of freedom. In contrast, the Legbot robot has been designed with only two actuated degrees of freedom in each leg. The contact point between the robot's foot and the ground can be considered an additional, passive degree of freedom. Despite the low number of degrees of freedom, the Legbot will still be capable of smooth motion. This is due to the fact that the robot is constrained to operating in only two dimensions. Chapters 6.5 – 6.8 demonstrate that a four degree of freedom, planar robot like the Legbot, is capable of smooth, human-like, walking trajectories.

Figure 6-4 illustrates the proposed relationship between the motor shafts and the robot joints. In the draft design, the hip joint is mounted on a freely rotating axis, and is driven by a motor mounted below the joint. The knee joint is actuated by a pair of tendons driven by the knee motor mounted in the robot torso.

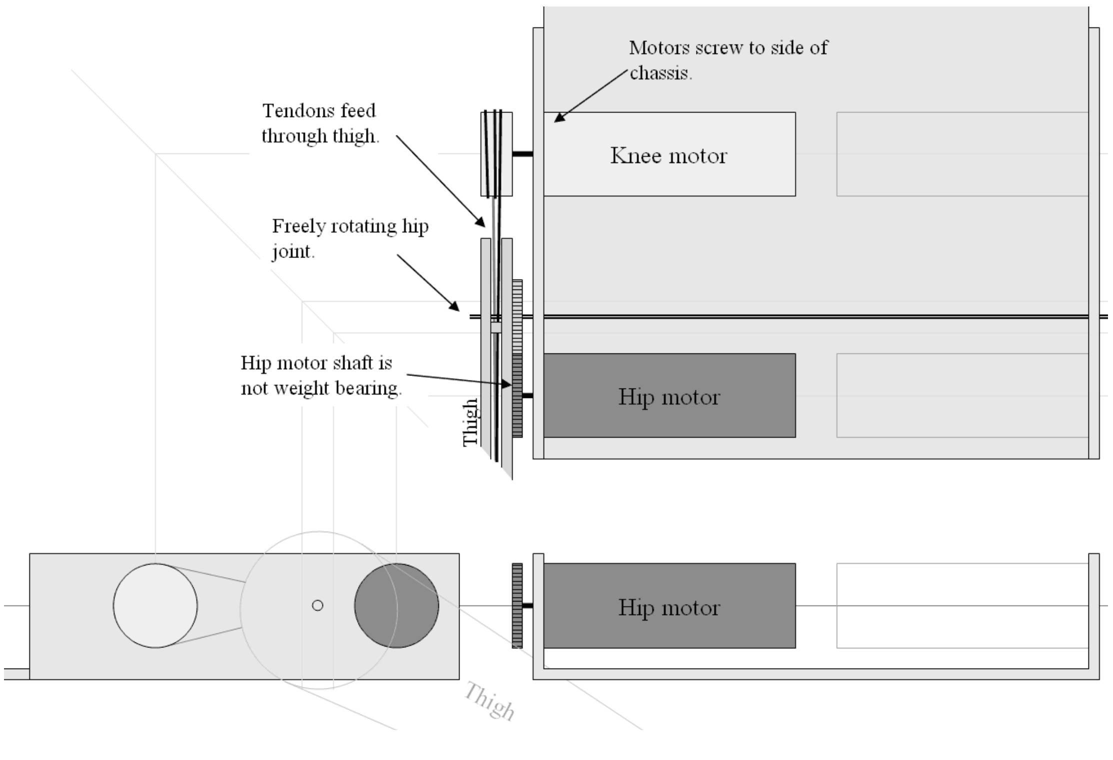

Motors screw to side of chassis.
Tendons feed through thigh.
Knee motor
Freely rotating hip joint.
Hip motor shaft is not weight bearing.
Hip motor
Thigh
Hip motor
Thigh

**Figure 6-4 Draft design motor mountings, and connections between drive shafts and robot hip joint. The knee joint is actuated by "tendons" running through the thigh, connected to the knee motor shaft.**

### 6.3.2 Physical construction

Using the draft robot design as a starting template, the real robot was constructed by Mr Kenneth Fogden, of the UWA engineering workshop [56]. Mr Fogden did a fantastic job turning the draft design into a real robot, and was able to build a robot that satisfied many of the original design goals listed in 6.3.1.

However, the constraints imposed by working with real world components meant that inevitably some compromises were required. Some of the most significant of these included:

- The robot is relatively heavy, weighing in at approximately 1.4 kg. A considerable amount of aluminium and lexan has been used to make the torso chassis.
- The knee joints experience a significant and unpredictable amount of friction, varying between 10-40% of the maximum torque that can be applied by the motors.
- The knee motor shafts bear the entire weight of the robot, as the knee motor shafts double as the hip joint mountings.
- Tendons proved to be unfeasible design decision, and have been replaced by simple belts drive connecting a cog mounted on the knee motor actuator to the actual knee joint.

Not all of the compromises listed above were detrimental to the robot performance. In particular, the belt drive replacing my proposed tendon system greatly simplifies construction and control of the final robot. The belt drive system is made feasible due to the innovative way Mr Fogden has designed and constructed the robot's hip joint.

Since the hip joint is mounted over the knee motor shaft, the distance between the knee actuator and the knee does not change as the hip angle changes. Figure 6-5 shows all components used to construct the hip joint of the Legbot robot. The hip joint freely rotates over a copper sheath (a) and cog (b), which are clamped to the knee motor shaft using a grub screw (d).

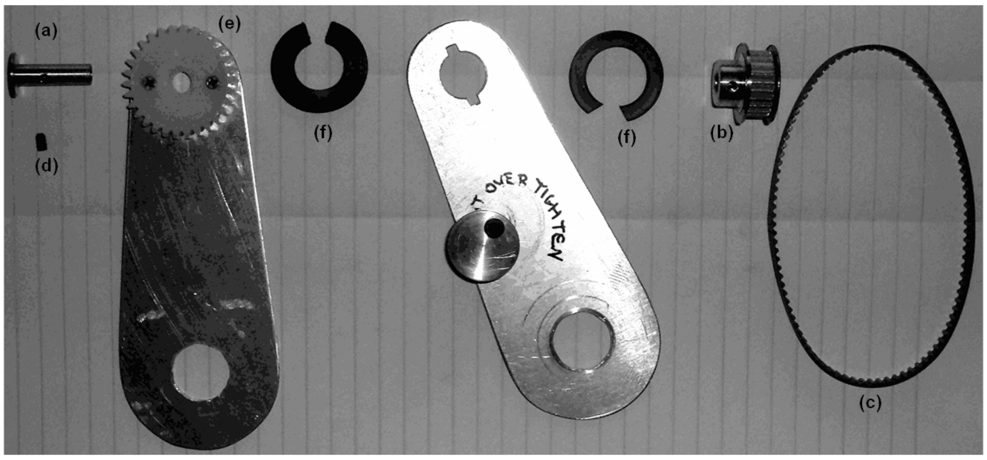

(a)
(e)
(d)
(f)
(f)
(b)
(c)
(a) Copper sheath
(b) Knee joint actuator cog
(c) Knee belt drive
(d) Grub screw
(e) Hip joint passive cog
(f) Low friction nylon spacers

**Figure 6-5 Components used to construct the Legbot's Hip joint and thigh link.**

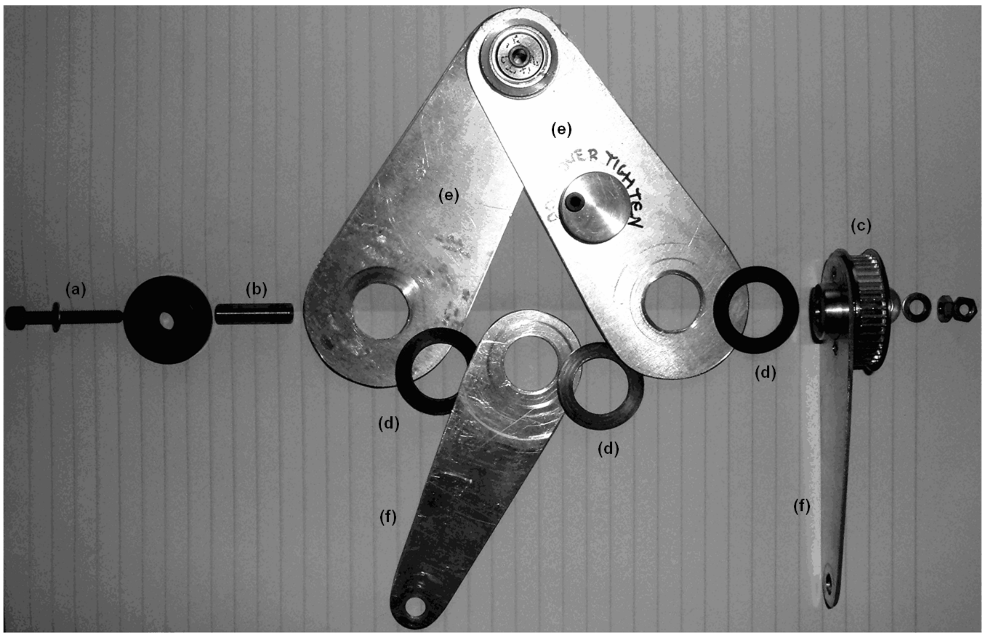


**Figure 6-6 Knee joint components, shown together with the assembled hip joint, thigh and shin links.**

### 6.3.3 Controller

The Legbot robot uses the EyeCon controller for mobile robots, developed by Joker Robotics [37]. For information about the EyeCon controller, and its RoBIOS operating system, see Appendix A-1, “Eyebot robotic platform”.

### 6.3.4 Actuators

The robot’s legs are each driven by a pair of Faulhaber DC Motors. These motors are similar to those used to drive the Ballybot robot, and enable the controller to apply a variable torque to the robot joints. Additionally, the motors have encapsulated encoders, which can be used to estimate the joint angles of the robot links.

### 6.3.5 Sensors

#### i. Inclination sensor package

Legbot uses the sensor package developed and tested using the Ballybot robot system, in 5.7, “Sensors for balance”. This sensor package uses a gyroscope to provide real-time control estimates of robot inclination and angular velocity, together with an inclinometer that is used to recalibrate the gyroscope.

#### ii. Gyroscope

The sensor package uses the Hitec GY-130 piezo rate gyro.

#### iii. Inclinometer

The sensor package uses the SEIKA Inclinometer N3 (Analogue output model).

iv. Shaft encoders

Optical shaft encoders encapsulated in the DC motor housing provide easily accessible measurements of motor shaft angle. In the Legbot robot, the shaft angular displacements can be used to determine the robot joint angles.

### 6.3.6 Software simulation

As with the Ballybot series of experiments (Chapter 5), a software simulation of the Legbot robotic system has been developed. This simulation allows researchers to experiment with various control systems designs, without risking damage to the physical robot. Ideally, the simulator would also have allowed preliminary investigations into possible robot designs to be conducted before beginning construction of the physical robot. Unfortunately, due to time constraints this was not possible, and the Legbot robot was designed and built in parallel with the development of the software simulation.

In an attempt to clarify this discussion, details about the development of the Legbot simulation system have been omitted from the main body of this document. These details include the software design and dynamic model derivation, and have been provided in Appendix C.

## 6.4 Experimental Overview

A series of experiments were conducted while developing this robot, including:

Chapter 6.5 - Torso Driven Walking control system

The feasibility of controlling a biped walking robot using the Torso Driven Walking control system (detailed in Chapter 3) is investigated using a software simulation of the Legbot robot. After a reasonably lengthy interval of manual control system parameter tuning, the various components of the control system work together to successfully balance the simulated robot as it walks. The Torso Driven Walking control system is proved to be a promising control system for biped robots, but was very difficult to optimise. This was due to the large number of tuneable parameters making up each implementation of the control system.

Chapter 6.6 - Genetic algorithm tuning of TDW

Genetic algorithms are evaluated as a tool for automatically tuning the Torso Driven Walking control system developed in 6.5. Such an approach enables the optimisation of the TDW parameters to be completely automated. This effectively eliminates one of the major drawbacks associated with the TDW controller. It is demonstrated that GA tuning of the Torso Driven Walking control system results in a robust algorithm capable of driving a biped robot with a variable velocity, while maintaining balance.

Chapter 6.7 - Porting the TDW control system to new environments

The adaptability of the Torso Driven Walking control system is investigated by implementing the control system on a second, independently developed and publicly available simulation system (Newton Game Dynamics [71]). This time the robot is simulated in three dimensions, and frictional forces as well as associated slippage of the robots feet are considered. The experiment demonstrates that a TDW control system developed on one system can be ported to a second environment and still perform moderately well. To get a good performance, the GA evolution of control systems must be repeated in the new environment. A process for taking a pre-evolved population of tuning parameters sets from one environment, and continuing evolution in a second environment is investigated, but is ultimately found to give unsatisfactory results.

Chapter 6.8 - TDW comparison with Virtual Model Control

In order to understand better advantages that may be realised through using the Torso Driven Walking control system to drive a biped robot, a comparison is conducted between the TDW control system and an implementation of Pratt's "Virtual Model Controller", patterned after Pratt's description in [85]. Each system is tested within the same simulated environment, and is asked to execute the same set of test procedures. It is demonstrated that TDW is more controllable and requires less joint torque than a VMC algorithm implemented on the same platform. In addition, the TDW control system can robustly handle gyroscope "bias" errors, while the VMC controller fails in the presence of a bias error in inclination measurement.

Chapter 6.9 - Physical legbot

The physical Legbot robot is evaluated to determine the accuracy of joint angle measurements, and the relationship between applied actuator voltage and joint torque. These characteristics are then compared with the requirements predicted by the simulation systems for a robot sharing the same characteristics as the physical Legbot robot. Unfortunately, the experiments described demonstrate that while the joint positions can estimated with a reasonable degree of accuracy, the joint torque that can be delivered by the actuators is low and difficult to control. This low and imprecise joint torque is well below the minimum torque required by the Torso Driven Walking control system.

## 6.5 Torso Driven Walking control system

**Abstract:** In this experiment, the feasibility of controlling a biped walking robot using the Torso Driven Walking control system is investigated. The TDW control system is applied to a software simulation of the Legbot robot. After a reasonably lengthy interval of manual control system parameter tuning, the various components of the control system work together to successfully balance the simulated robot as it walks.

### 6.5.1 Introduction

The goal of this experiment was to conduct preliminary investigations into the feasibility of controlling a biped robot using the Torso Driven Walking control system presented in Chapter 3. At this initial stage, the objective was not a perfect walking performance, simply to prove that walking is possible, and that the Torso Driven Walking control system is worth investigating further. Subsequent experiments examine techniques for improving the basic walking performance obtained in this experiment.

The Torso Driven Walking control system attempts to tackle the complex problem of biped robot control by decomposing the problem into a collection of simple constituent modules. If each module's behaviour is tuned correctly, working together the components can be shown to successfully balance a walking robot. Each module can be designed and tested independently, and then integrated to form the entire system. For example, the Torso Attitude Controller, an important component of the TDW system, has already been tested in a real-world environment on the Ballybot robot in Experiment 5.7.

Since the objective of this experiment is to validate the TDW design concept, the experimental system has been implemented on a simulated robot system. This made the initial development and tuning of the control system as simple as possible. Not only can the robotic environment be completely controlled, but the physical structure of the robot could also be adjusted to further simplify the control system development.

### 6.5.2 Materials

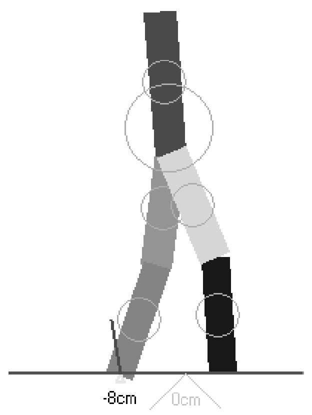
#### 6.5.2.1 Software simulation

This experiment makes use of the Legbot software simulator, described in Appendix C.

#### 6.5.2.2 Simulated robot architecture

The simulated robot architecture is summarised in Table 6-1. The link parameters have been changed slightly from those of the physical robot. This was done to simplify the process of manually tuning the prototype control system. The changes included lengthening and lightening the thigh and shin links of the robot, as well as representing each of the robot links by simple rectangles.

**Table 6-1 Simulated Legbot specifications.**

| | **Link** | **Specifications** |
|---|---|---|
|  | 1: Torso | 1kg, 186mm x 40mm, red. |
| | 2: Thigh (L) | 100g, 150mm x 40mm, green. |
| | 3: Shin (L) | 80g, 150mm x 35mm, light blue. |
| | 4: Thigh (R) | 100g, 150mm x 40mm, yellow. |
| | 5: Thigh (R) | 80g, 150mm x 35mm, blue. |

### 6.5.3 Control system overview

A full description of the Torso Driven Walking control system has been provided in Chapter 3. A short overview of the control system is presented here, together with tailoring required to adapt it to the simulated Legbot robot. A number of TDW components have been simplified, or removed from the control system for this experiment. Figure 6-7 illustrates the control flow context diagram for the Torso Driven Walking control system.

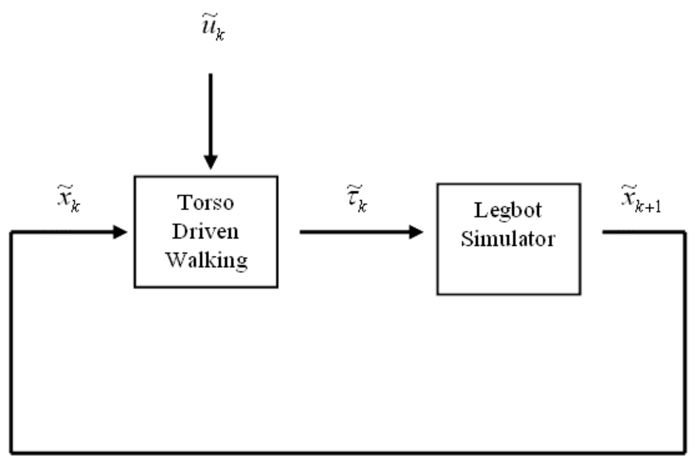


**Figure 6-7 Torso Driven Walking control flow context diagram.**

A description of the system variables interacting with the Torso Driven Walking control system is provided by Table 6-2.

**Table 6-2 System variables for Torso Driven Walking context diagram**

| | |
|---|---|
| $\tilde{x}_k$ | State vector of the Legbot system. This is an array of all joint angles, torso inclination and angular velocity, and the horizontal displacement of the base of the robot's torso. |
| $\tilde{\tau}_k$ | An array of output joint torques, which the robot applies to the four actuated joints. |
| $\tilde{u}_k$ | Control signal input by the user. In this experiment, there is no direct input from the user, so $\tilde{u}_k = 0$. |

### 6.5.3.1 Torso Driven Walking Overview

As shown by Figure 6-7, The Torso Driven Walking control system uses the observed system state of the robot, together with a control signal input by an operator, to determine an array of output joint torque values to be applied to the robot joints. The control system's objective is to maintain balance of the robot, while satisfying the input command as closely as possible.

Table 6-3 details the system state variables making up the state vector $\tilde{x}_k$, used by the Torso Driven Walking control system in this experiment:

**Table 6-3 State variables used by the Legbot's Torso Driven Walking control system.**

| | |
|---|---|
| $\theta_{Torso}$ | Inclination angle of the robot's torso. On the physical robot, this would be read by the combination gyroscope/inclinometer sensor module. |
| $\theta_{RHip}$ | Right hip joint angle, measuring the angle between the torso and the right thigh links. |
| $\theta_{LHip}$ | Left hip joint angle, measuring the angle between the torso and the left thigh links. |
| $\theta_{RKnee}$ | Right knee joint angle, measuring the angle between the right thigh and the right shin links. |
| $\theta_{LKnee}$ | Left knee joint angle, measuring the angle between the left thigh and the left shin links. |
| $x_{Hip}$ | Horizontal displacement (m) of the base of the torso (the hip joint). |
| $c_{RightFoot}$ | Flag indicating the right foot is in contact with the ground. |
| $c_{LeftFoot}$ | Flag indicating the right foot is in contact with the ground. |

Figure 6-8 decomposes the Torso Driven Walking control system into its constituent modules. Only those modules implemented in this experiment are shown in the diagram. For a detailed description of the full TDW control system, please refer to Chapter 3.

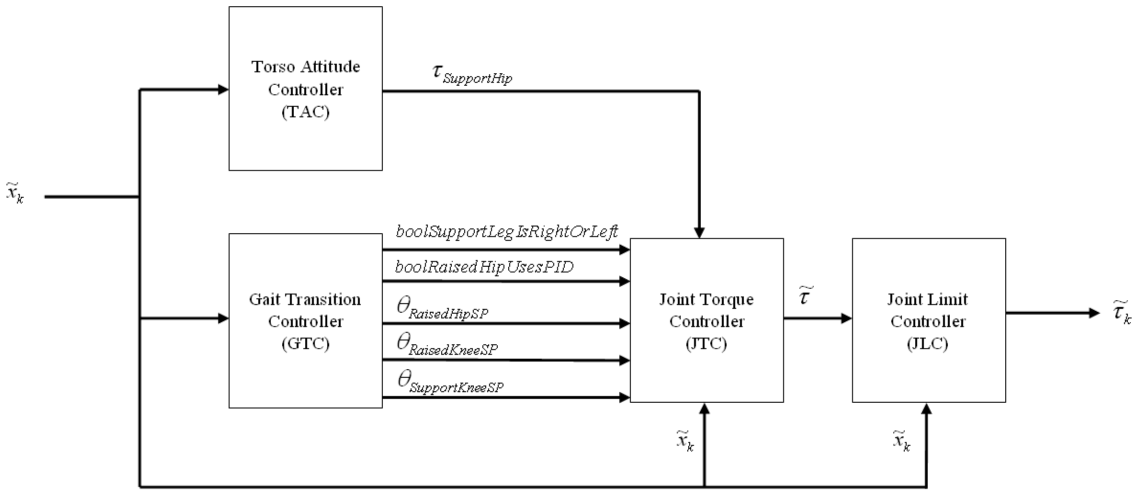


**Figure 6-8 Torso Driven Walking control system components.**

### 6.5.3.2 Torso Attitude Controller (TAC)

The TAC determines a balancing torque ($\tau_{SupportHip}$) to be applied to the hip of the robot's supporting leg. The controller uses the torso state variables $\theta_{Torso}$ and $x_{Hip}$ as input parameters to the Torso Attitude Controller equation, derived in Chapter 3.2.5.2:

$$\tau_{SupHip} = a\,(\theta_{Torso} - \theta_{Offset}) + b\,\dot{\theta}_{Torso} + c\,(x_{Hip} - x_{SP}) + d\,(\dot{x}_{Hip} - \dot{x}_{SP}) \qquad (1)$$

Normally the values for $x_{SP}$ and $\dot{x}_{SP}$ are supplied by the User Interface Controller[30] subsystem, however for this experiment, these values are held to zero and the User Interface Controller is not used.

Values for the velocity parameters $\dot{\theta}_{Torso}$ and $\dot{x}_{Hip}$ are estimated by observing the rate of change of the torso's angle and displacement over each control system sample interval ($T$):

$$\dot{\theta}_{Torso} = (\theta_k - \theta_{k-1})/T \qquad (2)$$

[30] The User Interface Controller is described in Chapter 3.UIC.

$$\dot{x}_{Hip} = (x_k - x_{k-1}) / T \qquad (3)$$

The Torso Attitude Controller tuning parameters that must be experimentally determined are listed in Table 6-4:

**Table 6-4 Tuning parameters for the TAC.**

| Parameter | Description | Example |
|---|---|---|
| $\theta_{Offset}$ | Target inclination angle of the robot's torso. | -2.1 |
| $a$ | Relates torso deviation to balancing torque. | 9.0 |
| $b$ | Relates torso angular velocity to balancing torque. | 0.3 |
| $c$ | Relates hip horizontal displacement to balancing torque. | -1.6 |
| $d$ | Relates hip horizontal velocity to balancing torque. | -0.6 |

### 6.5.3.3 Gait Transition Controller (GTC)

As the robot's torso is being balanced by the Torso Attitude Controller, torque applied to the supporting leg's hip will cause the leg to move away from its position beneath the torso. The Gait Transition Controller's primary role is to make sure the raised leg is moved into position to take over the supporting leg role when necessary. An additional responsibility of the Gait Transition Controller is to ensure that the supporting leg holds the robot's weight without buckling. These requirements are achieved through the use of a state machine, driven by the position of the supporting leg relative to the robot's torso. The output variables of the Gait Transition Controller subsystem include:

**Table 6-5 Output variables of the GTC subsystem.**

| | |
|---|---|
| *boolSupportLegIsRightOrLeft* | Indicates which leg is currently the support leg. |
| *boolRaisedHipUsesPID* | Usually the raised hip joint is controlled by a PID controller, however in a few states it is left unactuated. |

| | |
|---|---|
| $\theta_{RaisedHipSP}$ | Set point angle for the raised hip joint. |
| $\theta_{RaisedKneeSP}$ | Set point angle for the raised knee joint. |
| $\theta_{SupportKneeSP}$ | Set point angle for the support knee joint. |

While details of the Gait Transition Controller state machine are provided in Chapter 3.2.6, the state transition diagram has been reprinted in Figure 6-9 for reference.

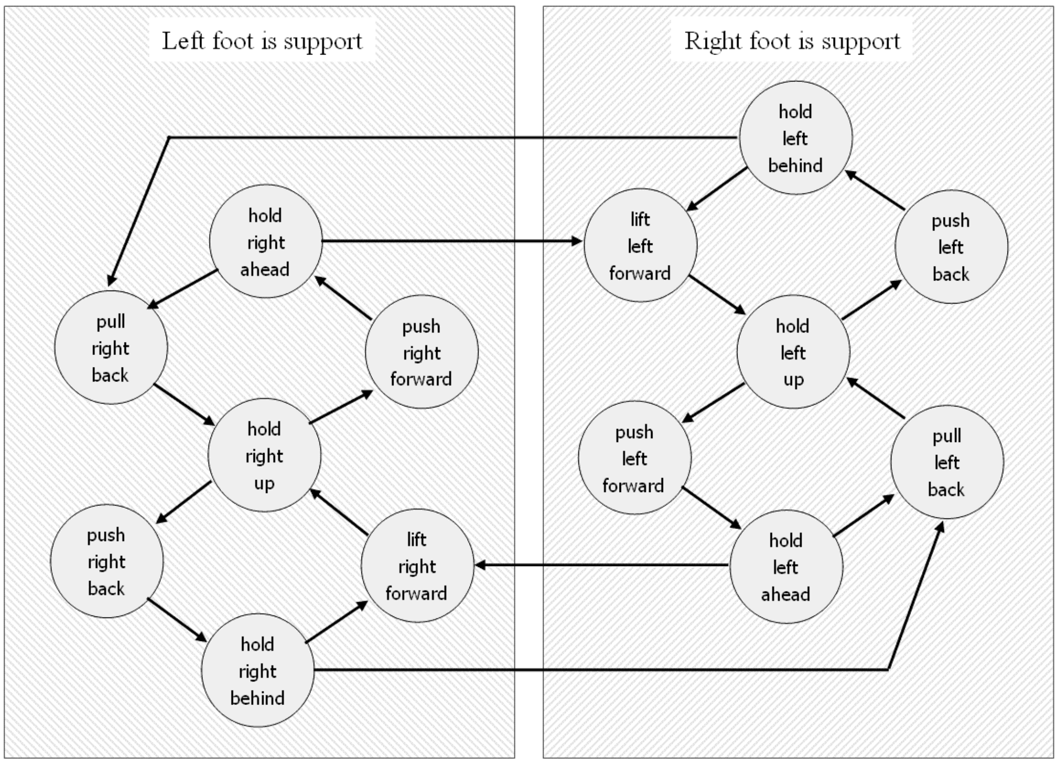


**Figure 6-9 Gait Transition Controller state diagram.**

Since the robot gait is symmetrical, only seven of the Gait Transition Controller states need to be defined. Each state program can be used for situations where either the left or right leg is acting as the support leg. The label of each state program describes the action being performed on the non-support leg during that state. A summary of the seven Gait Transition Controller state programs, and the tuning parameters associated with each state, is provided by Table 6-6. For a more detailed description of the state programs, transition triggers and trajectories, please refer to Chapter 3.2.6.

**Table 6-6 Tuning parameters for the seven GTC state programs.**

| | | |
|---|---|---|
| **Hold leg up**<br>During this state, the raised leg is held above the support foot, ready to move in either direction.<br>*boolRaisedHipUsesPID* is **true** for the duration of this state. | | |
| *Transition triggers* | | |
| # | Parameter description | Example |
| 1 | Transition to **Push leg forward** when angle between support foot and torso is less than trigger value. | -10.4 |
| 2 | Transition to **Push leg back** when angle between support foot and torso is greater than trigger value. | 7.7 |
| *Trajectories* | | |
| # | Parameter description | Example |
| 1 | Time taken to move joint set points to target values (ms) | 43.2 |
| | Support knee target value (degrees) | -7.23 |
| | Raised knee target value (degrees) | -63.3 |
| | Raised hip target value (degrees) | 20.4 |

| | | |
|---|---|---|
| **Push leg forward**<br>During this state, the raised leg is pushed ahead, ready to catch the robot's weight.<br>*boolRaisedHipUsesPID* is **true** for the duration of this state. | | |
| *Transition triggers* | | |
| None. State changes to **Hold leg ahead** when the raised foot hits the ground. | | |
| *Trajectories* | | |
| # | Parameter description | Example |
| | Time taken to move all joint set points to target values (ms) | 13.9 |
| | Support knee target value (degrees) | -5.6 |

| | | |
|---|---|---|
| 1 | Raised knee target value (degrees) | -30.5 |
| | Raised hip target value (degrees) | 20.5 |
| 2 | Time taken to move all set points to target values (ms) | 10.5 |
| | Support knee target value (degrees) | -6.7 |
| | Raised knee target value (degrees) | -16.5 |
| | Raised hip target value (degrees) | 11.4 |

**Hold leg ahead**

During this state, both legs are on the ground. The robot is preparing to shift its weight onto the forward leg.

*boolRaisedHipUsesPID* is **false** for the duration of this state.

*Transition triggers*

| # | Parameter description | Example |
|---|---|---|
| 1 | Transition to **Pull leg back** when angle between support foot and torso is greater than trigger value. | -2.6 |
| 2 | Transition to **Lift leg forward** when angle between support foot and torso is less than trigger value. *boolSupportLegIsRightOrLeft* is **changed** during the state transition. | 7.7 |
| 3 | Transition to **Lift leg forward** if the support foot is airborne and elapsed time in this state is greater than the trigger value. *boolSupportLegIsRightOrLeft* is **changed** during the state transition. | 23.9 |

*Trajectories*

| # | Parameter description | Example |
|---|---|---|
| 1 | Time taken to move support knee set point to target value (ms) | 27.0 |
| | Support knee target value (degrees) | -3.1 |
| | Time taken to move raised knee set point to target value (ms) | 591.7 |
| | Raised knee target value (degrees) | -7.9 |

**Lift leg forward**

During this state, the rear foot is lifted off the ground and swung forward.

| | | |
|---|---|---|
| *boolRaisedHipUsesPID* is **true** for the duration of this state. | | |
| *Transition triggers* | | |
| # | Parameter description | Example |
| 1 | Transition to **Hold leg up** when time expired exceeds target value. | 32.3 |
| *Trajectories* | | |
| # | Parameter description | Example |
| 1 | Time taken to move joint set points to target values (ms) | 32.8 |
| | Support knee target value (degrees) | -10.8 |
| | Raised knee target value (degrees) | -71.7 |
| | Raised hip target value (degrees) | 18.6 |

| | | |
|---|---|---|
| **Push leg back**<br>During this state, the raised leg is pushed back, ready to catch the robot's weight as the robot moves backwards.<br>*boolRaisedHipUsesPID* is **true** for the duration of this state. | | |
| *Transition triggers* | | |
| None. State changes to **Hold leg behind** when the raised foot hits the ground. | | |
| *Trajectories* | | |
| # | Parameter description | Example |
| 1 | Time taken to move joint set points to target values (ms) | 21.3 |
| | Support knee target value (degrees) | -2.5 |
| | Raised knee target value (degrees) | -46.4 |
| | Raised hip target value (degrees) | 6.6 |
| 2 | Time taken to move set points to target values (ms) | 17.6 |
| | Support knee target value (degrees) | -0.3 |
| | Raised knee target value (degrees) | -12.5 |
| | Raised hip target value (degrees) | 7.9 |

**Hold leg behind**

During this state, both legs are on the ground. The robot is preparing to shift its weight onto the back leg.

*boolRaisedHipUsesPID* is **false** for the duration of this state.

*Transition triggers*

| # | Parameter description | Example |
|---|---|---|
| 1 | Transition to **Lift leg forward** when angle between support foot and torso is less than trigger value. | 5.5 |
| 2 | Transition to **Pull leg back** when angle between support foot and torso is greater than trigger value.<br>*boolSupportLegIsRightOrLeft* is **changed** during the state transition. | 10.4 |
| 3 | Transition to **Pull leg back** if the support foot is airborne and elapsed time in this state is greater than the trigger value.<br>*boolSupportLegIsRightOrLeft* is **changed** during the state transition. | 18.5 |

*Trajectories*

| # | Parameter description | Example |
|---|---|---|
| 1 | Time taken to move support knee set point to target value (ms) | 18.1 |
|  | Support knee target value (degrees) | -0.2 |
|  | Time taken to move raised knee set point to target value (ms) | 352 |
|  | Raised knee target value (degrees) | -8.8 |

**Pull leg back**

During this state, the front foot is lifted off the ground and swung back.

*boolRaisedHipUsesPID* is **true** for the duration of this state.

*Transition triggers*

| # | Parameter description | Example |
|---|---|---|
| 1 | Transition to **Hold leg up** when time expired (ms) exceeds target value. | 42.0 |

*Trajectories*

| # | Parameter description | Example |
|---|---|---|
|  | Time taken to move joint set points to target values (ms) | 44.3 |

|   |   |   |
|---|---|---|
| 1 | Support knee target value (degrees) | -4.3 |
|   | Raised knee target value (degrees) | -85.7 |
|   | Raised hip target value (degrees) | 25.2 |

### 6.5.3.4 Joint Torque Controller (JTC)

The Joint Torque Controller is responsible for determining the torque that should be applied to each of the robot's joints:

The support leg's hip torque is set to the balancing torque provided by the Torso Attitude Controller subsystem, unless the support foot is not in contact with the ground, in which case the supporting leg's hip torque is set to zero.

Most of the remaining robot's joints are controlled using a simple PD control algorithm, which tracks the changing joint set point positions output by the Gait Transition Controller state programs. Joints controlled via PD include:

- Support leg's knee
- Raised leg's knee
- Raised leg's hip (if the variable *boolRaisedHipUsesPID* is **true**, otherwise torque applied to the hip joint is *zero*.)

**Table 6-7 PID Tuning parameters used by the JTC.**

| Parameter | Description | Example |
|---|---|---|
| *P* | Proportional constant | 0.05 |
| *D* | Differential constant | 0.00074 |

#### 6.5.3.5 Joint Limit Controller

The Joint Limit Controller monitors the value of the robot's knee joints, and applies a corrective torque if the joint moves outside a predetermined limit. This effectively prevents the Legbot's double-jointed knees from hyper-extending.

The motivation for, and theory behind, the Joint Limit Controller subsystem is described in detail in Chapter 3.2.9. In summary, restricting a joint's range of motion can be useful for a number of reasons, including:

- Simulating a "locking" knee joint, a requirement for passive walking.
- Preventing damage to the robot by reducing the chance of internal collisions.
- Simplifying the robot dynamics by eliminating the possibility of multiple solutions in the robot's kinematic equations.

The Joint Limit Controller's implementation has been inspired by the "spring/damper" collision model used in the simulation's dynamic model[31]. Whereas the collision model calculates a ground reaction force based on the penetration of the robot into the ground, the Joint Limit Controller uses a similar calculation to determine an overriding control torque to apply to joints that have moved beyond the predetermined joint limit positions. The form of the collision model is identical to an asymmetrical PD control system, and this is how the limit control has been implemented by the Joint Limit Controller.

[31] The simulation's dynamic model "Collision model" is described in Appendix C-4.2

**Table 6-8 Tuning parameters for the JLC.**

| *Parameter* | *Description* | *Example* |
|---|---|---|
| *P* | Proportional constant | 21 |
| *D* | Differential constant | 0.0002 |
| Hip limits | Hi and lo limits of hip operating range | (-90.0, +90.0) |
| Knee limits | Hi and lo limits of knee operating range | (-90.0, -1.0) |

## 6.5.4 Methods

In this experiment, the Torso Driven Walking control system has been implemented using a sample interval of 0.001 seconds. This fast sample rate was chosen to simplify the manual development of the control system, making the PD controllers responsible for adjusting joint angles easier to tune.

The experiment involved an iterative process of algorithm adjustment, followed by experimental trials as tuning parameters were manually adjusted. The process was repeated until a configuration was found that maintained the robot's balance for a reasonable length of time. For this experiment, a walking duration of 60 simulated seconds was judged to be a successful trial.

Experimental procedure:

1. Run the simulation
2. Observe how it fails
3. Tune the control system parameters, and repeat (1-3) until either:
4. Common failure modes suggest a change be made to the program, or satisfactory walking performance is obtained.

Figure 6-10 illustrates the experimental procedure as a flow chart:

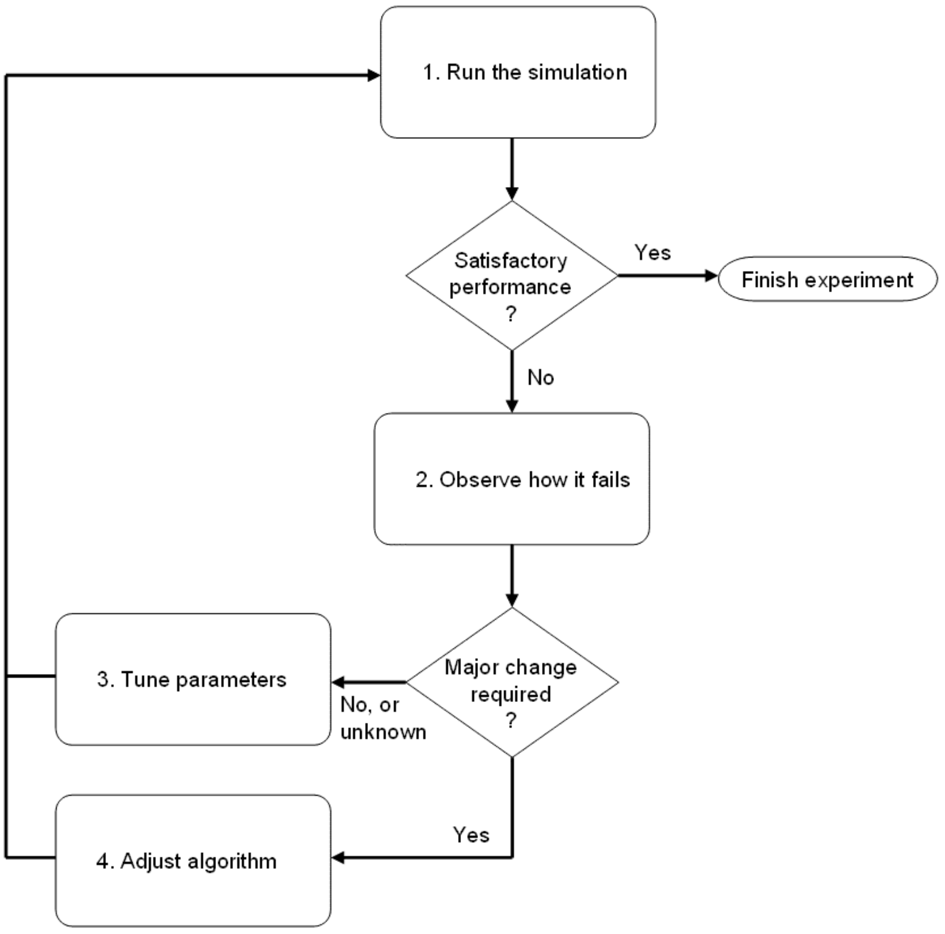


**Figure 6-10 Experimental procedure flow chart.**

As most of the tuning parameters belong to the Gait Transition Controller subsystem, adjusting this component was the most time consuming part of the process. The problem of tuning the Gait Transition Controller is made easier due to the state machine nature of the controller. Changes to the control system can largely be limited to one state at a time, making sure the robot behaviour "looks" right at the moment of transition to the next state.

### 6.5.5 Results

This manual, iterative approach to control system design resulted in a basic walking pattern that was reasonably stable. Due to the large number of tuning parameters, and the interrelated nature of the parameter changes, tuning the system was a time consuming process. It took in the order of 8 hours to manually tune the system sufficiently to achieve the 60 second balancing target of this experiment.

The graph shown in Figure 6-11 illustrates how the performance of the TDW control system varies as one of the tuning parameters is changed. In this example, the parameter being varied is the torso offset angle. The fitness of a control system trial is defined as the proportion of time the robot remains balanced, given the maximum trial duration of 60 seconds. Similar changes in performance could be seen when varying other parameters.

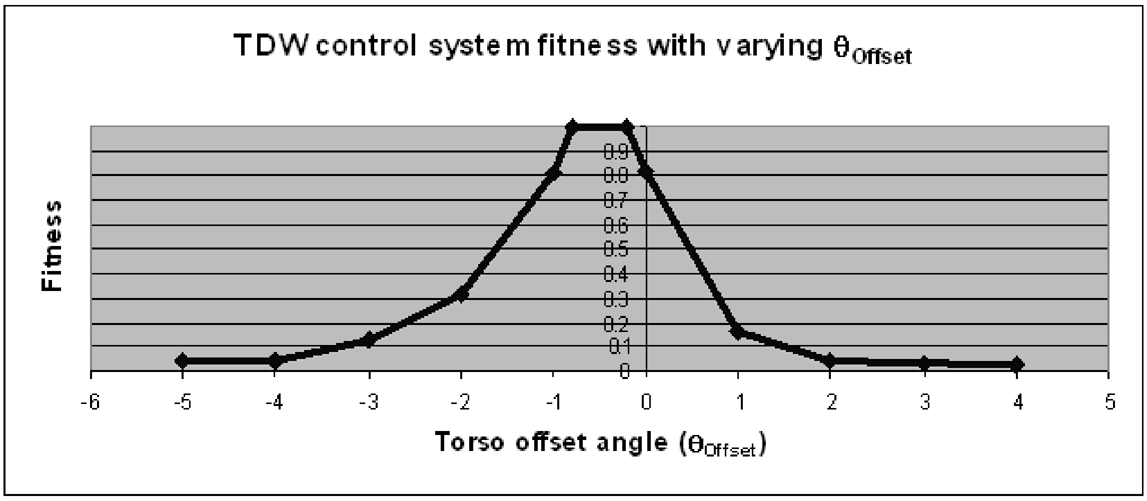


**Figure 6-11 Trend shows varying control system fitness, as the torso offset angle parameter is tuned.**

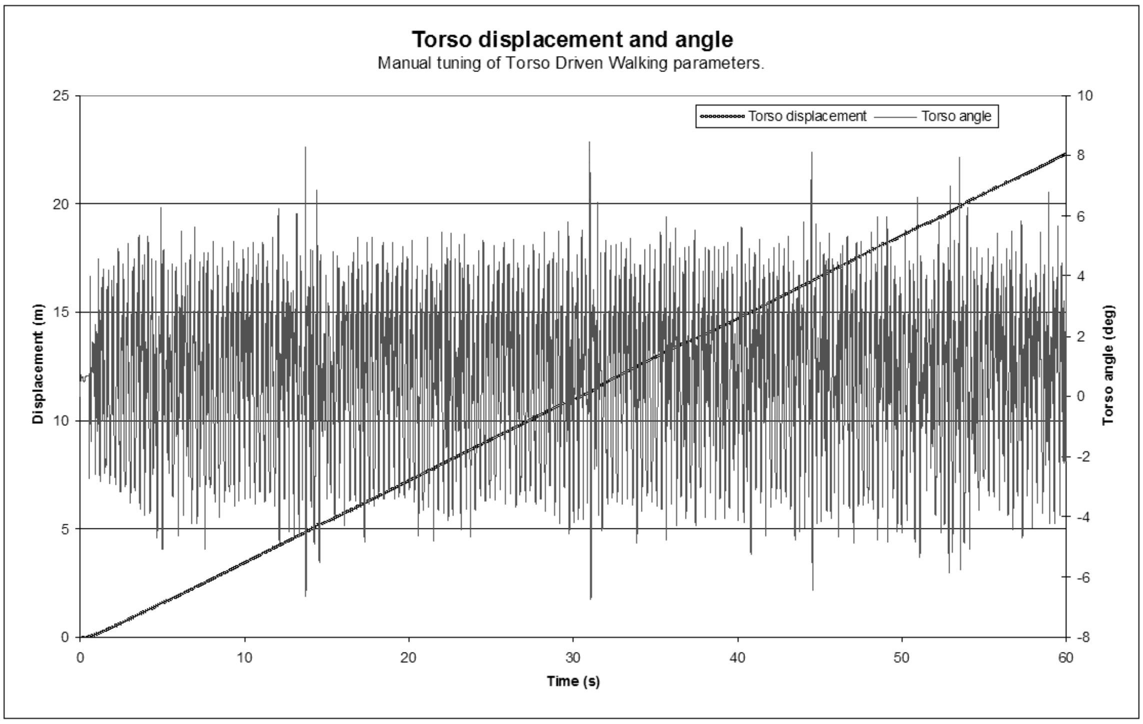


**Figure 6-12 the manually tuned Torso Driven Walking control system was able to maintain balance of the simulated robot for the 60 second trial duration.**

The control system regularly maintains balance for quite a long time; however it is not a perfect solution:

- Small little changes to the control system or initial state can cause the robot to fail.
- The robot velocity was not controllable, nor was the robot position.
- The simulated robot walked forward pretty well, but was not good at going backwards.

A demonstration of the manually tuned Torso Driven Walking control system can be seen at the URL:

http://www.ee.uwa.edu.au/~suthe-aj/thesis/legbot/manualLegbot.htm

The demo and simulation system at this URL are implemented as unsigned ActiveX controls and may require users to adjust their InternetExplorer security settings in order to view the controls.

### 6.5.6 Discussion

The manually tuned Torso Driven Walking experiment discussed here achieved its primary goal, which was to prove that the TDW control system is a feasible approach to the control of walking robots. The walking algorithm behaviour is dynamic, appears organic, and looks sophisticated. Even when the robot falls, the behaviour of the prototype control system is encouraging. The Legbot looks like it is trying to keep balance in a reasonable way – it appears to react sensibly to unexpected conditions. If more time had been spent tuning the control system, the performance of the control system would have continued to improve. What is not clear is what the maximum performance of the control system might be.

Selecting appropriate values for the control system's tuning parameters is critical to the system's performance. In this experiment, a human operator was responsible for selecting appropriate values for parameters, while observing the impact of changes to the behaviour of the system. This approach allowed the development of the walking algorithm to be directed in an intelligent way.

Due to the modular nature of the control system, it was natural to focus on one component at a time, when making changes to the control system tuning parameters during the experimental process. For example, changes made to the control system between trials might be limited to a single state of the Gait Transition Controller component, such as the "Hold Leg Up" state. This state's parameters could be iteratively modified until the robot movement during the course of the state "looks" right, at which time attention can be focused on the next Gait Transition Controller state.

While there are some advantages to having a human undertake the tuning of the control system parameters, there are also significant penalties in taking this approach:

- Perhaps the most significant disadvantage is that the process of manually tuning the Torso Driven Walking control system's more than 50 parameters is labour intensive, time consuming, and very tedious.
- Determination of what "looks right" is a subjective measure, and may not actually result in control decisions that helps the robot to walk. After all, the Legbot is not a human being, so trying to make it move like a human may not result in the most efficient walking gait.
- The tuning parameters are not independent variables – changing one variable value may have an impact on the optimal value for an entire collection of related parameters. The relationships between groups of parameters may not be obvious, and as a result, it is very unlikely that an optimal solution will be found using a manual tuning strategy.

Ideally it would be possible to delegate the tedious job of tuning the control system parameters to a computer, leaving the experimenter free to oversee the control system development and consider more fundamental algorithm changes that may be required.

Genetic Algorithms are a class of adaptive algorithm ideally suited to this purpose, and are the subject of the next experiment, "Genetic algorithm tuning of TDW", described in Chapter 6.6. These tuning approaches, applied to the Torso Driven Walking control system, will result in a robust, controllable, variable speed walking algorithm for the simulated Legbot robot.

## 6.6 Genetic algorithm tuning of TDW

**Abstract:** In this experiment genetic algorithms are evaluated as a technique for automatically tuning the Torso Driven Walking control system introduced in 6.5. The Torso Driven Walking control system is an effective control system for biped robots, but proved to be very difficult to optimise. This was due to the large number of tuneable parameters making up each implementation of the control system. Genetic algorithm approaches enable the optimisation of the TDW parameters to be completely automated. This effectively eliminates one of the major drawbacks associated with the TDW controller.

### 6.6.1 Introduction

In Experiment 6.5, an implementation of the Torso Driven Walking control system was developed which successfully balanced a simulated biped robot as it walks. The robot balances dynamically, with a natural looking gait, and without the need for a supporting polygon. While the control system proved to be feasible, a significant cost in terms of time and effort spent optimising the control system's performance was encountered. The control system requires the tuning of over 50 interrelated parameters, which work together to determine the behaviour of the control system. In the previous experiment, each of these parameters needed to be manually adjusted, and the impact of parameter changes subjectively determined by observing the control system's behaviour. This is a process which does not lend itself to achieving an optimal solution. After a number of days of experimentally adjusting the control system parameters, a walking gait was developed, which kept the robot upright as it moved forward at a steady speed. However, the control system was not able to move the robot with a variable speed, nor could it move the robot backwards or hold it at a steady position. Achieving these goals would have required further tuning of the control system parameters. Furthermore, if any details of the structure of the robot being controlled were to be changed, the entire tuning process would need to be repeated from scratch. In order to be a viable control system, an automated approach to determining the control system parameters is a significant requirement.

Genetic algorithms (GA) are a class of adaptive systems that are ideally suited to the kind of optimisation problem posed by tuning of the Torso Driven Walking control system. Genetic algorithms, and the terminology used to describe them, take their inspiration from the study of biological evolution. The objects being evolved are termed “genomes” or “individuals”, with each genome being a potential solution to the problem being investigated. A pool of potential solutions (genomes) is maintained by the algorithm, and termed a “population”. Individuals within the population are evaluated by an objective “fitness” function, with poorly performing genomes eliminated over a sequence of “generations” as the system evolves. This is survival of the fittest. Most GA algorithms ensure new genomes introduced to the population in subsequent generations inherit some of their characteristics (or “genes”) from one or more high performing individuals within the population. Some genetic algorithms allow for random “mutations” to be introduced, to ensure the population doesn’t stagnate. This serves the same purpose as simulated “annealing” sometimes used to train neural network systems.

The result is an automated system that iteratively adjusts and evaluates potential solutions to a problem, in a directed way, with minimal human intervention. The basic premise of iteratively adjusting and testing potential solutions is similar to the process that was used to manually tune the Torso Driven Walking control system in Experiment 6.5, with the advantage that the GA system will exhibit infinite patience as it “evolves” a solution. In addition, since the GA system is an automated solution, it can be expected to be largely free of having its results prejudiced by the influence of any pre-conceived notions held by the experimenter of what might constitute the "right" solution.

They have been applied to walking systems before, including [67], [69], [70], [74]. Some of these are discussed in Chapter 2.3. In this experiment, GA tuning will be restricted to control system parameters only. Other elements making up the control system structure, such as the number of states in the "State Transition Controller", will not be adjusted using genetic algorithms.

### 6.6.2 Materials

#### 6.6.2.1 Torso Driven Walking control system

The Torso Driven Walking control system is used as the basis for this investigation into GA tuning. The control system, and its constituent components, is described fully in Chapter 3.

#### 6.6.2.2 Software simulation (Legbot simulator)

As with the manually tuned TDW control system, this experiment makes use of the Legbot software simulator, described in Appendix C, to evaluate the performance of the Torso Driven Walking control system.

#### 6.6.2.3 Genetic algorithms class library

MIT's "GALib" genetic algorithm library [68] was used as the basis for the software classes developed during the course of this experiment.

### 6.6.3 Methods

In the experimental procedure described here, genetic algorithm techniques are applied to the problem of determining optimal values for the "Torso Driven Walking" control system tuning parameters listed in 6.5.3. Much of the work involved in developing the solution involved an iterative process of evaluating potential designs for components of the system.

Some of the major components of a GA system that must be determined include:

- Genome definition
- Fitness function
- Crossover function
- Mutation
- Speciation

These components and some of the issues surrounding them are discussed in the following sections. In addition to these items, standard GA parameters must also be determined. In these experiments I have settled on the following values for standard GA parameters:

- Population size ~ 30 individuals.
- Number of generations ~ depends on the rate of convergence (up to approx. 150).
- Replacement rates ~ 20% of the population is replaced after each generation.

#### 6.6.3.1 Genome definition

The key to a successful implementation of a GA approach to solving a problem is the ability to express every possible solution to a problem using the data stored in a single structure known as a "genome". This structure is often described as being analogous to a living organism's chromosomes. The information stored within the genome is the data which the genetic algorithm will manipulate as it evolves a solution. A set of specific values stored in a genome describes an "individual" possible solution to the problem being investigated.

Since the genetic algorithm can only work with data stored within the genome, the algorithm will never be able to evolve solutions that cannot be described the information stored in the genome's data structure. As a result, it is important that a genome being designed to solve a specific problem contains enough detail to allow researchers to be confident that it will be possible to use the genome to describe a satisfactory solution to the problem at hand. Offsetting this requirement to be as expansive as possible is the increased training time the algorithm requires exploring the solution space resulting from the use of a larger genome.

For the Torso Driven Walking control system, the GA genome needs to describe all the parameters which the algorithm is required to tune. The more parameters that are inserted into the genome, the more likely it is that the algorithm will be able to converge to a good solution. On the other hand, as the genome size increases, the algorithm will take a much longer time to find that solution.

The structure chosen to describe a genome for the TDW control system is a simple array of floating point values. In initial experiments, the number of parameters added to the genome was very small – initially the only parameter included was the "torso offset angle". As the experiments progressed, more and more parameters were added, until in the final experiments 41 parameter values were being evolved simultaneously.

#### 6.6.3.2 Fitness function

Almost as important as the structure of the genome, is the fitness function used to judge the quality of individual solutions. A good quality fitness function is essential, as the algorithm will be automated, and must rely entirely on the fitness function to compare the relative merits of two different individuals. If the fitness function is poorly defined then the algorithm can easily produce strange results that might satisfy the fitness function, but not solve the real world problem.

The input to a fitness function is a genome containing data describing an individual solution to be evaluated. The output is a single, scalar value indicating the relative quality of the individual. The goal of the fitness function is to somehow map the input data to a quality signal, in such a way that high scoring individuals exhibit the characteristics the researcher wants to see. In the case of a biped walking robot, the fitness function is designed to reward the following behaviours:

1. Stay balanced – the robot should be rewarded for not falling over.
2. Keep walking – the robot should be penalised if it gets stuck.
3. Stay close to a target position – the robot should be rewarded for being able to follow a moving target position.

The only way to tell how well an individual is able to meet these objectives is to run an experiment. As a result, the fitness function involves running a trial of the control system described by the input genome data on the Legbot simulation system. In order to keep the training times to a reasonable limit, each trial is limited to a maximum duration of 60 seconds. A trial may be prematurely halted by the system if a critical error is encountered – for example, if the robot falls over the trial will be halted.

In order to ensure the control system resulting from the genetic algorithm are able to drive the robot to follow a variable speed trajectory, the robot's set point position is adjusted during the course of each trial to follow the predetermined profile shown in Figure 6-13. This ensures that a robot that is able to stand still, as well as move forwards and backwards at variable speed, will score a higher fitness result than a control system that cannot change the robot's horizontal velocity.

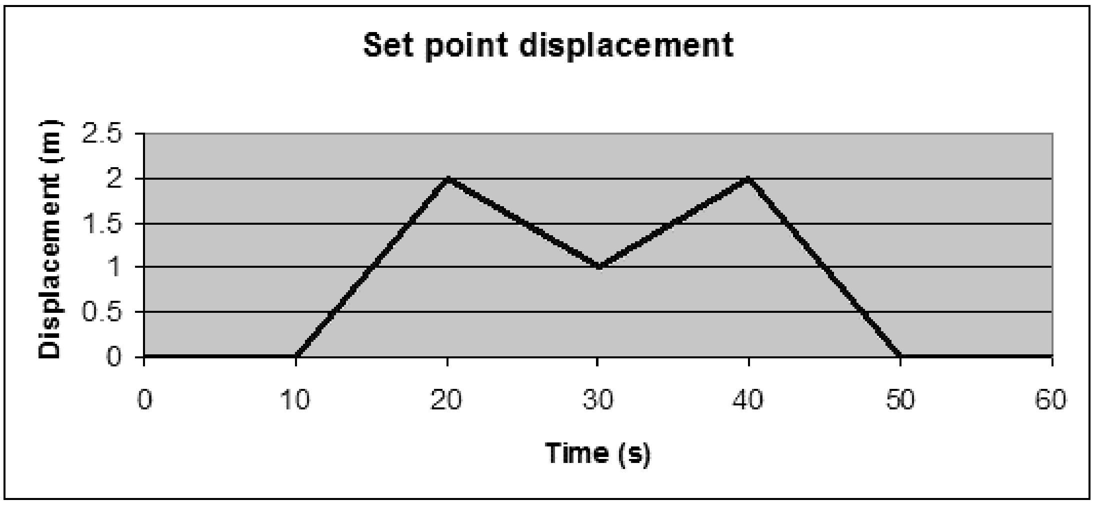


**Figure 6-13 The target set point position is varied in a consistent way in every trial conducted to determine the fitness of an individual.**

The algorithm used by the Torso Driven Walking fitness function is described below:

- Each second, a partial fitness score is calculated, based on the distance between the robot's torso and the target location, together with a small bonus score for staying upright:

```
// This function is called each second that the trial
// runs, accumulating the individual's fitness score.
void UpdateFitnessScore()
{
    float fError_x = abs( EstimateHipDisplacement() );

    // Score increment due to proximity to target
    if (fError_x < SCORING_LIMIT) {
        g_fFitness += 1 - (fError_x / SCORING_LIMIT);
    }

    // Bonus score for simply staying upright
    g_fFitness += 0.25;
}
```

With a `SCORING_LIMIT` of 0.5m, the partial fitness score exhibits the profile described by Figure 6-14.

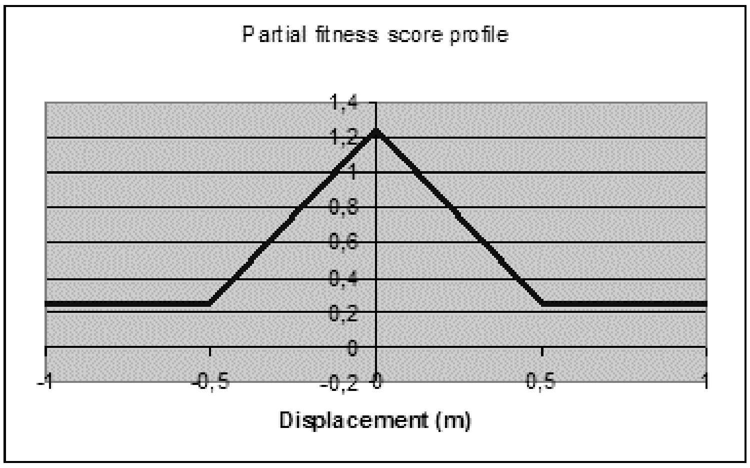


**Figure 6-14 The profile of a partial fitness score varies with the displacement of the robot from a target position.**

- The trial runs for a maximum simulated time of 60 seconds, at the end of which time the accumulated fitness score is scaled to a floating point value ranging between zero and one:

  ```
  // Scale final fitness to a range of {0 -> 1}
  g_fFitness *= 0.8;
  ```

- If the robot falls over, the trial is immediately terminated and the robot no longer accumulates a fitness score.
- If the robot's feet come too close together (< 2cm), the trial is terminated and the robot no longer accumulates a fitness score.
- If both of the robot's feet remain continuously on the ground for a reasonably long duration (2 seconds), then the robot is judged to be standing still and the trial is terminated. The robot no longer accumulates a fitness score.

#### 6.6.3.3 Crossover function

In the terminology used to discuss genetic algorithms, a crossover function is a euphemism for the process of reproduction within the population being evolved. The idea is that new individuals being added to the population inherit most of their characteristics from parent individuals selected from the previous generation. Assuming that high performing individuals are selected as parents for the subsequent generations, the goal of a crossover function is to ensure that desirable traits are passed on to future generations. A crossover function may be sexual – where two (or more) parents combine to produce the new individuals – or asexual – where only a single parent is required, and the child shares most of the characteristics of its parent. In the experiments discussed in this chapter, both types of crossover function are utilised.

There are an infinite variety of crossover functions that can be applied to a GA problem, but all of them share a few characteristics. The input to the function is one or more parent individuals. The output of the function is one (or more) "child" individuals, which share the same genome as the parents. The data contained by the output child's genome is influenced in some way by the data making up of each of its parents. Continuing the biologically inspired theme for naming elements of a genetic algorithm, this data is often referred to as the individual's genes.

Since the crossover function's job is to create a new individual by manipulating the genes of its parents, and filling out a new copy if the genome, the structure of the genome will have a strong influence on the kinds of crossover operations that should be performed.

Some commonly used techniques employed by crossover functions working with arrays include swapping array elements, or sub-arrays, between the parent genomes, rearranging segments of sub-arrays within the same genome and rearranging segments of bits making up the numbers in a floating point array. None of these techniques particularly suit the genome structure used in these experiments. Since each gene has a very specific meaning, it does not make sense to swap the positions of these genes (for example switching the value for torso inclination angle with the trajectory duration of the "lift leg up" action).

To implement the "sexual" crossover function, a fairly simple algorithm is applied: every gene making up the child's genome is set to a weighted average of the value of each of the parent's corresponding gene. The weight used to generate the new value is randomly re-determined for each gene in the genome, before the parent values are combined, as illustrated in Figure 6-15:

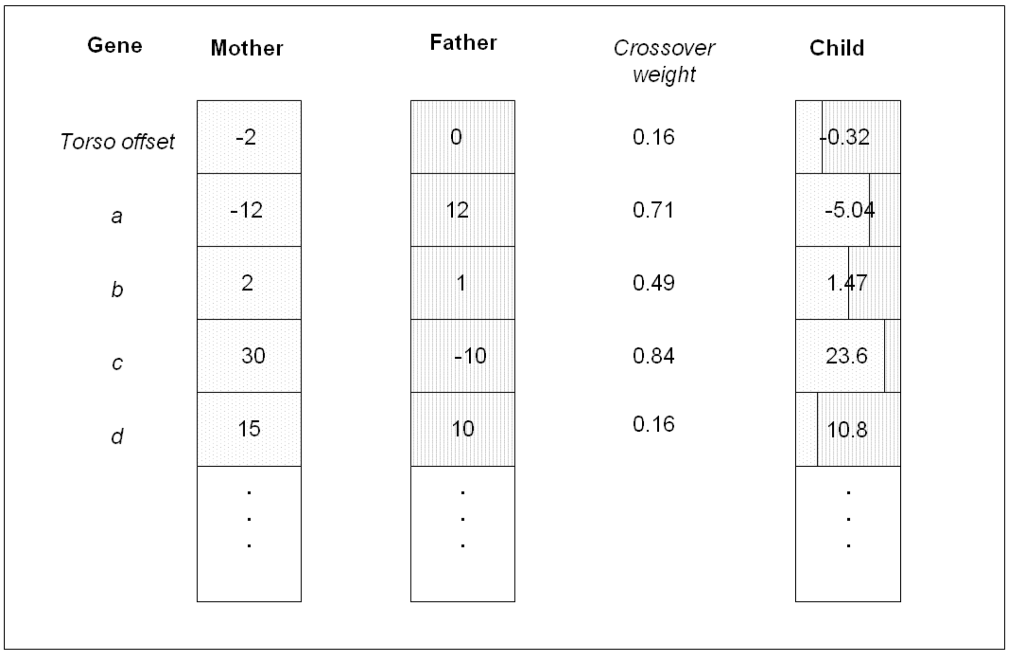


**Figure 6-15 The sexual crossover function used in these experiments calculates a weighted average of each of the parent's genes to fill in the child's genome. In this example, the five genes describing the "Torso Attitude Controller" tuning parameters are shown. The data values shown are for illustrative purposes only.**

### 6.6.3.4 Mutation

Obviously, creating children by simply "averaging" the parent's characteristics is not good for promoting a system's "biodiversity". As the population evolves, the genetic makeup of individuals within the population will converge to a single set of genetic values. This is fine if the behaviour of those individuals happens to be the optimal solution, but this result is very unlikely. Such an event would require randomly generated members of the original population to span the optimal solution. Something needs to be done to ensure some children can exhibit exciting new behaviours as the system evolves. In genetic algorithms, this is often achieved through the use of simulated "mutation".

In the genetic algorithm developed for tuning the "Torso Driven Walking" control system, mutations can be introduced in one of two ways:

- Standard GA mutation rate is 10% chance per individual per generation, using a mutation rate of 0.1.
- Asexual reproduction will force a mutation of the child's genome, applying a mutation rate of 0.8.

If an individual is selected as the subject of a mutation, the following algorithm is performed:

```
For each gene in the individual's genome:
    There is a probability of (mutation rate) that the gene will
    mutate.  If so:
        The gene value += GAGaussianFloat( gene range / 20)
```

This means that if an individual is cloned from an existing member of the population (asexual reproduction), then each gene in the child's genome has a high chance (80%) of changing by the value of a random variable selected with a Gaussian probability distributed about zero, with a standard deviation proportional to the nominal range of possible values for the gene.

If an individual is generated using a sexual crossover operation, there is only a 10% chance that an individual will be spontaneously exposed to the mutation function. In this case however, the chance for each gene to mutate is much smaller (also only 10%). If a mutation occurs however, the gene value will vary by a similar amount as genes mutating after an asexual reproduction (a Gaussian random variable, distributed about zero, with a standard deviation proportional to the gene value range).

#### 6.6.3.5 Speciation

An interesting behaviour was noticed during initial experiments where the genetic algorithm was applied to tuning a single parameter of the Torso Driven Walking control system, the "torso offset angle". Each time the experiment was run, the individuals in the experiments population converged to a single value for the target "torso offset", which successfully balanced the robot. Not surprisingly, the value the system most often converged to was close to 0 degrees – the value used when manually tuning the system in Chapter 6.5.

Occasionally however, the system would converge to a second value, a torso offset angle close to -4 degrees. This second possible solution was totally unexpected, and illustrates nicely one of the major benefits that can be achieved through the use of an automated tuning system: The algorithm has a good chance to discover possible solutions that might be missed by a human experimenter.

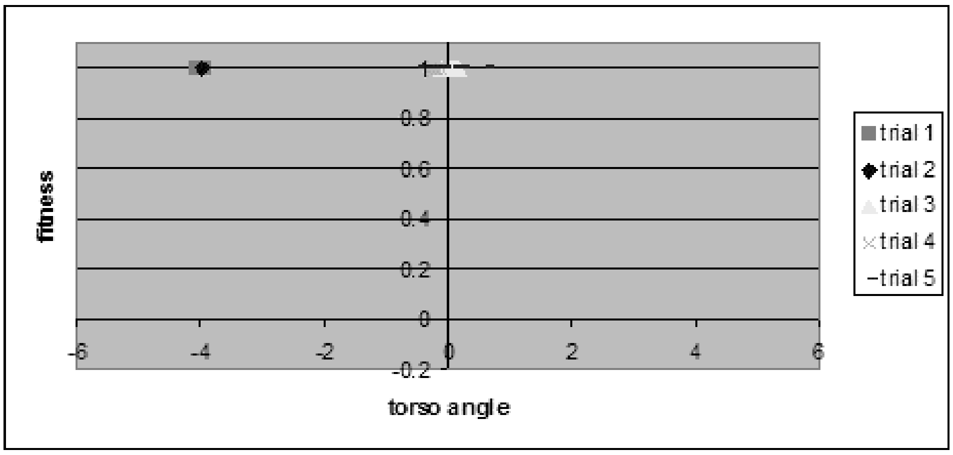


**Figure 6-16 With no "speciation", each trial evolution resulted in a population of individuals which exhibited one of two possible "ideal" torso angles. In none of the preliminary trials did the algorithm generate a population including both possible solutions.**

The problem this experiment illustrates is that while the algorithm could determine two possible solutions, in none of the trials did the final population include individuals who had converged to both solutions. The trial would find one solution or the other, but never both.

There are a number of reasons why such a result is undesirable – the main one being that it is unwise to place all the evolutionary eggs in one basket! If the environment in which a population evolves were to change, the risk is that the population will be unable to adapt to the new circumstances. There are plenty examples in biology of species over-specialising and being unable to adapt to changing environmental conditions. One way environmental conditions could be expected to change is if parameters such as robot link length, or controller update rate, were to be adjusted during the execution of the evolution. In such a situation it is undesirable to risk the entire population becoming extinct. This is especially true if several days' worth of training have already been invested in the experiment.

Ideally, the final population should include examples of all possible solutions. Such a condition would imply the coexistence of more than one distinct group of individuals within the GA population. This situation is analogous to the evolution of distinct solution "species" within the population. In the experiment above, it would be desirable to see one species evolved to exhibit a "torso offset" angle of 0 degrees, and another to -4 – with both species co-existing in the same population. This result was achieved by imposing a "closeness" condition on parents selected as input to a "sexual" crossover operation. By testing the Euclidean distance between genome instances, a sexual crossover is only allowed to occur if the parents are similar.

The result upon repeating the experiment of evolving a single "torso offset" gene is shown in Figure 6-17. As the system trains, two sub-groups soon become evident in the population, congregating about each of the two possible solutions.

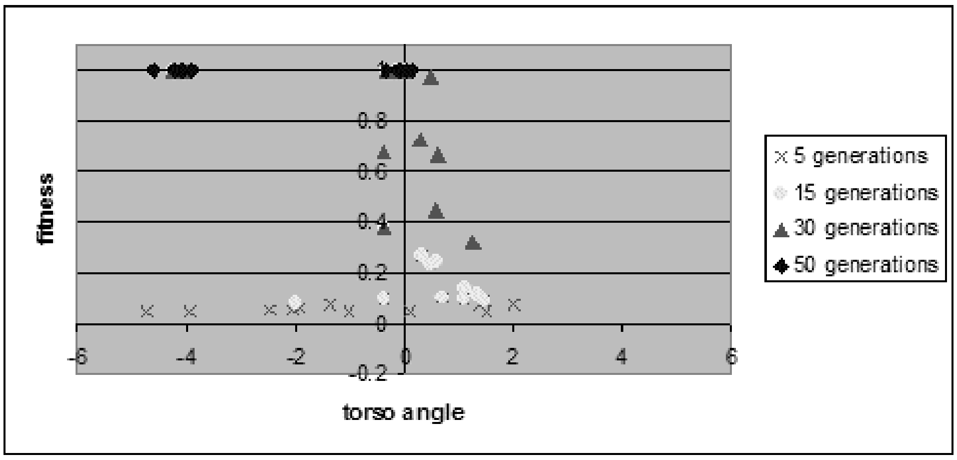


**Figure 6-17 Forcing parents of a sexual cross-over function to be similar allows the evolution of distinct sub-groups (or species) within the population. As a result, it is possible for the final population to include both possible solutions for ideal "torso offset" angle.**

The same procedure described here for training a genome consisting of a single gene was then simply extended to include many more training parameters.

### 6.6.3.6 Intelligent design

Freedom from the drudgery of manually adjusting weights allows researchers to focus on the higher level characteristics of the control system. Especially if it is possible to observe the evolving behaviour of the control systems as they evolve.

Two specific examples of changes that were made in response to observations of the evolving control system performance were: (i) changes to gene value limits, and (ii) collision avoidance controller.

i. Determining gene value limits

As the control systems in a GA's population evolve, the profile of the fitness of individuals making up the population forms a curve that tends to improve rapidly at the beginning of the trial, but only slowly as the "best" individuals are determined, and become the dominant genomes within the population. Typical fitness profiles are shown in Figure 6-18 and Figure 6-19.

During this process of evolution, the gene values describing the population's individuals are varied within a set of pre-determined limit values. The limit values are different for each gene in the genome. Determining appropriate values for these limits is a fairly arbitrary process. Limiting the range of possible values for each gene will increase the speed with which the populations converge on an "ideal" set of species, but may preclude the best possible solutions. The approach taken to counter this possibility is to examine the values of the genes of the most successful individuals in a trial's final population. If any of these values are close to the predetermined limits for the corresponding gene, then the limit is increased and the trial re-run.

In this way, it was found that the PID values chosen for the manually tuned solution, as well as the limits for the GA algorithm, were far too conservative. Table 6-9 lists the value ranges that were used during initial experiments to constrain the possible values of the tuning parameters used in the "Torso Attitude Controller" component of the Torso Driven Walking control system.

**Table 6-9 Gene value ranges used during preliminary experiments, for the Torso Attitude Controller tuning parameters.**

| *Gene* | *Min* | *Max* |
|---|---:|---:|
| *Offset* | -5 | 5 |
| *a* | 6 | 18 |
| *b* | 0 | 0.6 |
| *c* | -4 | 0 |
| *d* | -2 | 0 |

Figure 6-18 depicts the changing quality of the individuals within the population during a preliminary experiment conducted using the gene value limits listed in Table 6-9. Within 50 generations the fitness score stabilises at what is an admittedly healthy level.

Note that for this early trial, and for that depicted in Figure 6-19, the evaluation function used is not that described in 6.6.3.2. Instead the fitness score is simply: ("duration of trial"/60). In addition, the foot proximity clause for terminating a trial is not being enforced during these preliminary trials. Together, these changes to the evaluation function result in unusually high scores for population fitness.

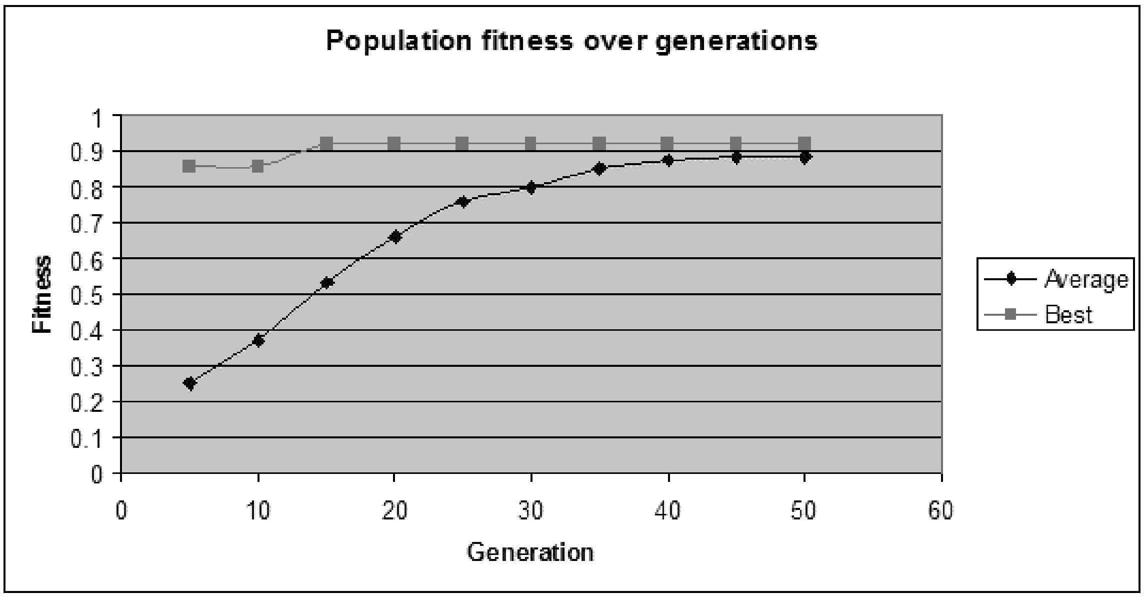


**Figure 6-18 Preliminary GA results, using sub-optimal gene range limits. At the conclusion of the evolution it was discovered that the best performing individuals in the population had a number of gene values at, or very close to, the corresponding limits selected for the gene. This suggests the best possible solution could not have been found by the algorithm, as the gene values required lie outside the envelope of possible individual configuration.**

After examining the individuals making up the final population, it was found that the values for the "b" and "d" genes were at the extreme limits of the range of possible values. As a consequence, the range of these parameters was increased, and the trial repeated until the resulting population's gene values fell well within the limits. The resulting values for gene limits are shown in Table 6-10.

**Table 6-10 Gene value ranges used during subsequent experiments, after the initial population evolved values for the "*b*" and "*d*" genes that were at the limit of the gene's original range.**

| *Gene* | *Min* | *Max* |
|---|---|---|
| *Offset* | -5 | 5 |
| *a* | 6 | 18 |
| *b* | 0 | 3.6 |
| *c* | -4 | 0 |
| *d* | -6 | 0 |

Figure 6-18 shows how increasing the range of genetic values for the Torso Attitude Controller resulted in improving population fitness. The best performing individual maintains balance for approximately 95% of the trial duration (~57 seconds). Also evident is the fact that the population has not yet stabilised and further improvements could be expected with increased training generations.

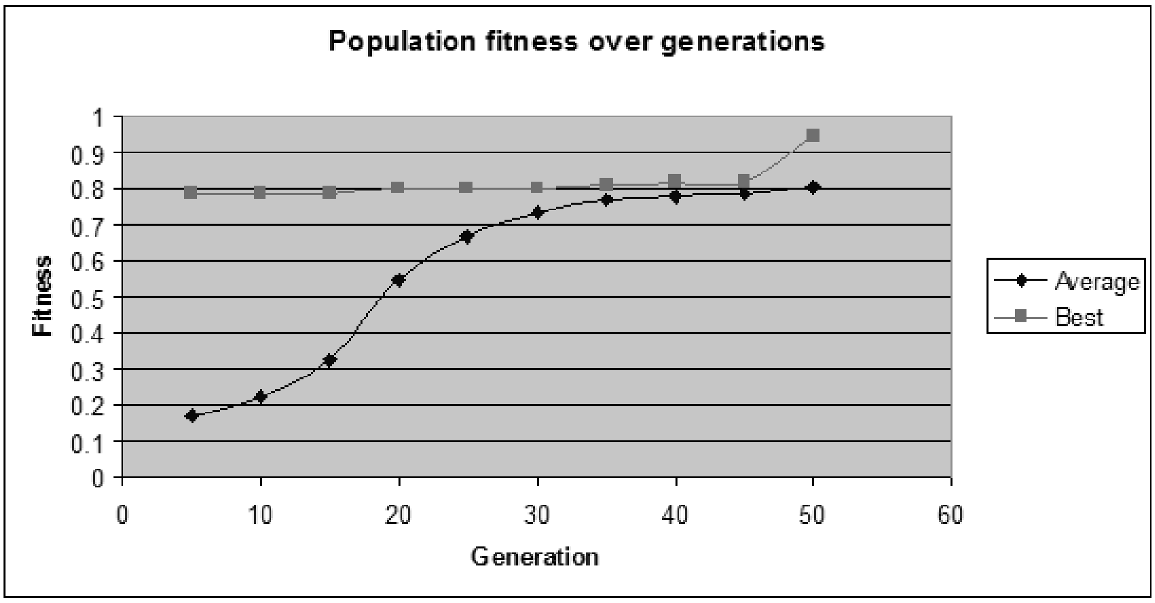


**Figure 6-19 The experiment is repeated, using an expanded range of possible gene values. After 50 generations the performance of the individuals in the population is already improved - even though the system has not yet converged to a stable population.**

ii. Collision avoidance controller

During a number of trial evolutions, it was noticed that even though the population's variation was stabilising – indicating that the "optimal" solution has been found – there was still a fairly poor performance. The most commonly observed mode of failure was as a result of the feet coming too close together. If the feet move closer than 2cm to each other, the fitness function correctly terminates the evolution, as such a condition would risk the physical "Legbot" robot tripping over its own feet. Figure 6-20 shows the evolving fitness function of the robot systems exhibiting this problem.

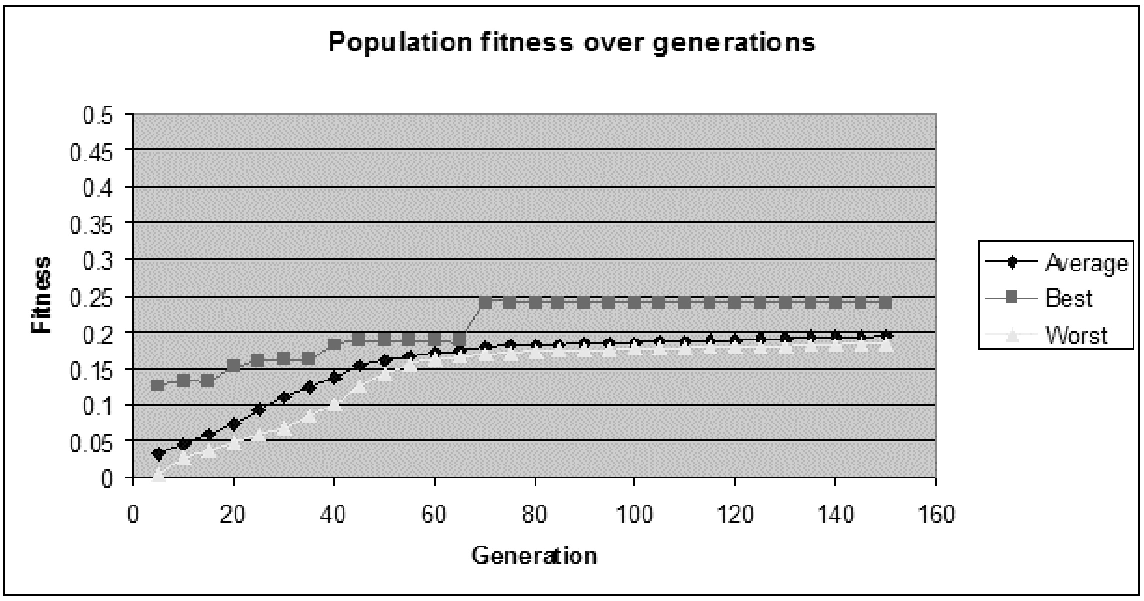


**Figure 6-20 Without a "Collision Avoidance Controller", the quality of the resulting control system is limited to a fairly low value (~0.25). In practice, most individuals in the population appear to fail by tripping over their feet less than 15 seconds into the simulated trial.**

After 150 generations, the best performing individual still has a very low fitness score (less than 0.25). By the end of the evolution, most individuals being tested fail after only 10-20 seconds has elapsed in the trial. After observing the behaviour of the best performing individuals, it was determined that a change to the control system itself would be required in order to improve the system's performance. Simply fiddling with the tuning parameters would be unlikely to solve the problem.

The solution implemented was to add a new component to the Torso Driven Walking control system in order to prevent the feet from coming too close together. This component was the "Collision Avoidance Controller", and is described in 3.2.7. After implementing the Collision Avoidance Controller and re-running the evolution experiments, the failure rate due to foot proximity was greatly reduced, resulting in the improved fitness scores that can be seen in Figure 6-21.

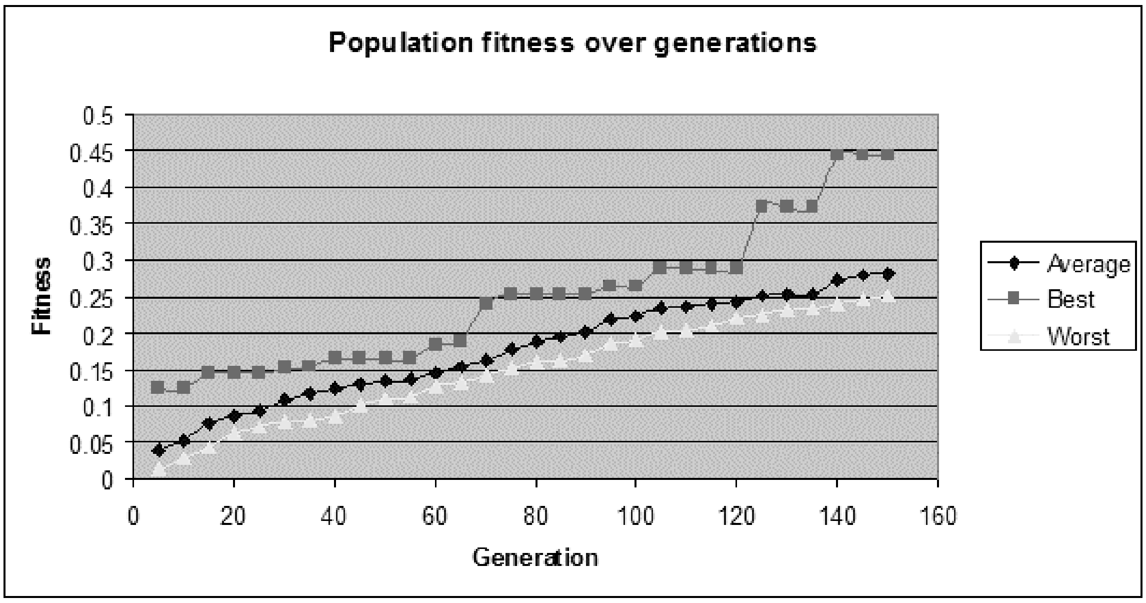


**Figure 6-21 When the Torso Driven Walking control system includes the "Collision Avoidance Controller" subsystem, the performance steadily improves with advancing generations. With a high final quality score (~ 0.45), the robot maintains its balance for the entire duration of the 60 second trial.**

## 6.6.4 Results

The genetic algorithm resulting from the process described above achieved all of the goals for which it was designed. The algorithm is able to consistently evolve sets of tuning parameters, which when applied to the Torso Driven Walking control system, successfully drive a simulated Legbot robot as it follows a target position moving with a variable velocity. The algorithm tends to converge to a stable population of solutions after about 150 generations. Figure 6-21 shows the typical profile of a trial as the fitness of the population improves with advancing generations.

The resulting set of tuning parameters, when applied to the Torso Driven Walking control system provides a stable, yet dynamic walking gait. The robot was able to track a target set point position moving with a variable velocity, while maintaining balance. Figure 6-22 charts the robot's horizontal displacement, and torso inclination angle, as the robot tracks a target position that moves with a variable velocity.

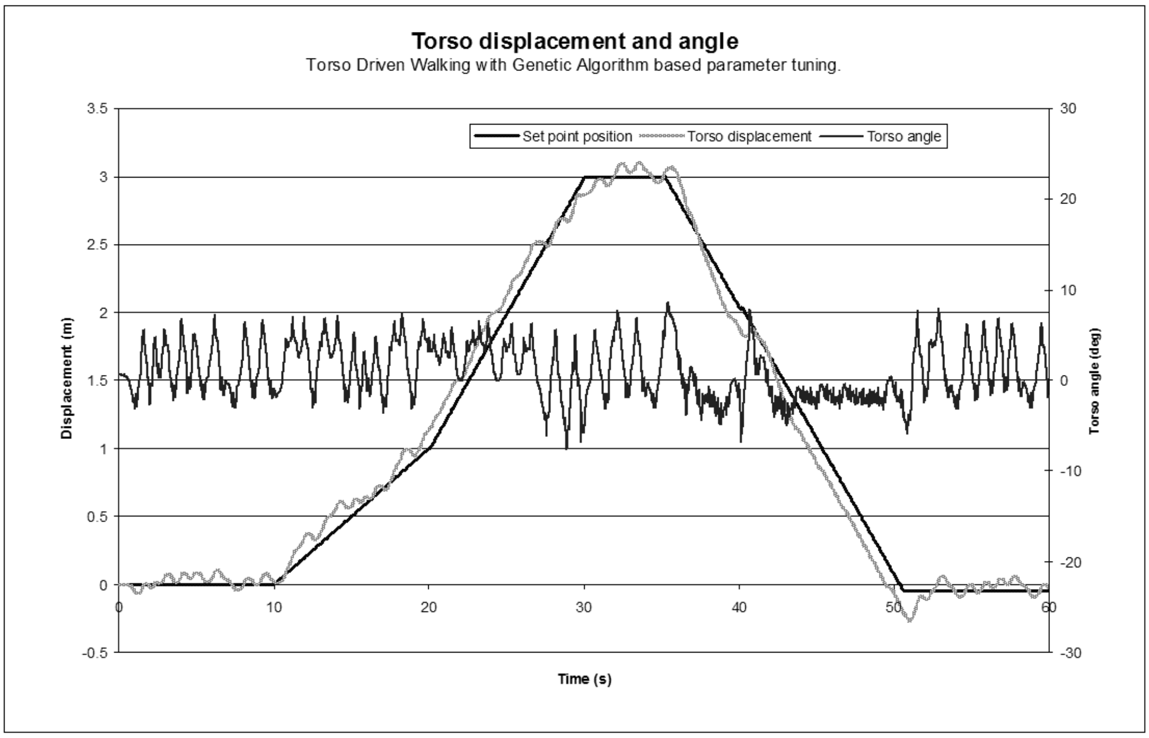


**Figure 6-22 The GA-tuned Torso Driven Walking control system can balance the simulated robot indefinitely, while following a target position moving with variable velocity.**

Once a satisfactory performance was achieved by the control system, the effect of noise was introduced to the state measurements used by the control system. The simulated noise parameters included:

- Joint angle quantisation of 0.1 degrees.
- Torso inclination bias error drifts every 0.002 seconds by an amount varying between +/- 0.1 degrees.
- Torso transient error varied from +/- 0.1 degrees at each measurement.

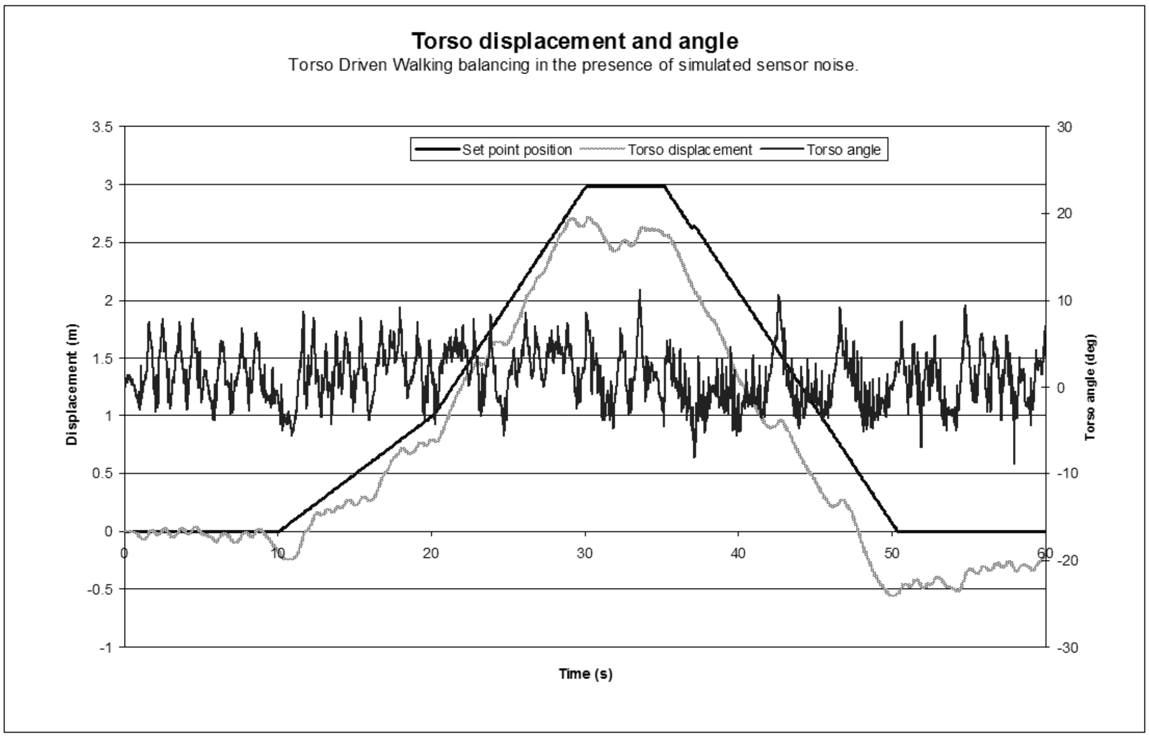


**Figure 6-23 The GA-tuned Torso Driven Walking control system is able to maintain balance in the presence of simulated sensor noise. In this trial, joint angle measurements are quantized to a precision of 0.1 degrees. The torso angle measurement is modified by a simulated signal bias that drifts by +/- 0.1 degrees every 0.002 seconds, and is also impacted by a transient noise signal varying between +/- 0.1 degrees. In this diagram, the simulated noise components have been omitted from the state signals in order to improve clarity.**

Despite the fact that the system was not trained in the presence of the modelled sensor noise, the control system was able to maintain balance during the course of a simulated 60 second trial. Figure 6-23 shows the results of a typical trial in the presence of simulated sensor noise.

A demonstration of the automatically tuned Torso Driven Walking control system can be seen at the URL: http://www.ee.uwa.edu.au/~suthe-aj/thesis/legbot/autoLegbot.htm.

### 6.6.5 Discussion

In this experiment, a set of GA rules have been developed, which when applied to the task of tuning the Torso Driven Walking control system, result in a robust controller for biped locomotion. The availability of an automated tuning algorithm for the control system means that the TDW system becomes a feasible option for implementing a robot's control system.  Without such a process, the time and effort required to manually determine an appropriate set of tuning parameters is prohibitively expensive.

Also, the algorithm has discovered optimal solutions that were entirely outside the boundary of solutions examined in the manual tuning experiment (Chapter 6.5).  For example, upon examining the parameters of the individual used to balance the robot in Figure 6-22, it was observed that the differential displacement component of the controller maintaining the torso attitude has a numerical value of "-4.1".  This value is more than two times lower than the minimum value considered when developing a manual solution.

Compared to many other adaptive systems, the GA tuning system for Torso Driven Walking requires a reasonably low number of trials to produce a satisfactory solution. Less than 1000 distinct trials are needed to test 150 generations during the evolution of a set of tuning parameters.  This number of trials is fairly low when compared to other adaptive control systems.  For example, the "first-principles" ANN algorithm for balancing an inverted pendulum (Chapter 5.8) required well over 10,000 trials to complete training.

Despite the relatively short training time, 1000 required trials are still a large number to have to perform by hand on a physical robot.  In this experiment all training was performed on a software simulator due to the convenience in stopping and starting evolutions, as well as the ability to run an evolution with no active input from a human operator.  Obviously, for optimal results, the simulation needs to model the physical system as closely as possible.  Since this experiment used the same simplified simulation system that was used for the "manually tuned" Torso Driven Walking experiment (Chapter 6.5), a required future step will be to alter the simulation to mimic as closely as possible the behaviour of the targeted "Legbot" physical platform.  Some major changes that will need to be considered include reduced sample rates and joint torque limits, as well as some minor changes to the robot morphology.

One possible advantage of using a GA based approach is that the simulator can be adapted slowly, during the execution of an evolution. The GA population should be able to adapt to incremental changes in the simulation behaviour in much the same way environmental change drives biological evolutionary processes. Extending this idea further would imply that initial GA training can be performed online, using a simulator that mimics as closely as possible the real robot. Once the population of solutions learns this system, the evolutionary process can be continued on the physical robot. Hopefully this second "on-line" stage will require far fewer generations than would otherwise be thee case to generate satisfactory results, which would mean reduced wear on the physical robot – as well as reduced fatigue for the researcher!

In Chapter 6.7, the feasibility of this approach was investigated by moving the control system from the simulation system used for this experiment to a new environment. In this case, a freely available, third party physics simulation system, "Newton Game Dynamics" was used.

## 6.7 Porting the TDW control system to new environments

**Abstract:** In this experiment, the adaptability of the Torso Driven Walking control system is investigated by implementing the control system on a second, independently developed, simulation system. This time the robot is simulated in three dimensions, and frictional forces as well as associated slippage of the robots feet are considered. The experiment demonstrates that the TDW control system developed on one system can be ported to a second environment and still perform very well. A process for taking a pre-evolved population of tuning parameters sets from one environment, and continuing evolution in a second environment is investigated, but is ultimately found to give unsatisfactory results.

### 6.7.1 Introduction

The previous experiment (Chapter 6.6) introduces a Genetic Algorithm, which was able to successfully tune an implementation of the Torso Driven Walking control system. The resulting control system was able to balance a planar 5-link robot operating in a simulation environment, the "Legbot" simulation system[32]. The robot was able to walk forwards or backwards, with a variable and controllable torso velocity, as well as being able to maintain a stationary position balancing on one foot.

If the TDW control system is to be a feasible system for balancing and controlling bipedal walking robots, the successful results I have obtained with the control system working within the "Legbot" simulator must be repeatable in other environments. In addition, while developing the Legbot simulation system allowed ultimate control over how the dynamic model behaved, there is the risk that errors may have introduced in the implementation of the simulator. With this possibility in mind, it is appropriate to

[32] The "Legbot" dynamic simulator for multi-link planar robots is described in detail in Appendix C.

validate the results by applying the same control system to a mock-up of the Legbot robot in a second, independent environment.

To achieve these goals, the Torso Driven Walking control system was ported to an environment that uses a third party dynamic simulator as its physics engine. This simulator is "Newton Game Dynamics" (NGD), a free library designed to provide real-time simulation of physics environments. The Newton Game Dynamics API is available from [71].

Applying the TDW control system to a robot simulated within the NGD environment attempts to prove three things:

i. The Legbot simulation environment is reasonably accurate. This is because the behaviour of complex objects modelled in each of the two environments, are reasonably consistent.
ii. The GA tuned TDW controller is reasonably robust. The TDW approach is able to cope with the inevitable differences in behaviour of bodies modelled in two fundamentally different dynamic simulators.
iii. Continued application of GA tuning in a new environment quickly results in a control system that exhibits satisfactory performance. If true, this would mean a minimum amount of GA training will be required in the new environment.

The experiment is performed in three stages. First the control system and associated tuning parameters evolved in Chapter 6.6 are directly ported to an NGD environment. The algorithm is applied to a simulated robot that mimics as closely as possible the planar robot used to evolve these tuning parameters. The behaviour of the resulting control system is then analysed, and will show that as long as there is sufficient friction between the ground surface and the robot's feet to prevent "slippage" of the robot's balancing feet, the control system can prevent the robot from falling.

Next, the genetic algorithm evolution process described in Chapter 6.6 is repeated using the NGD simulation environment to implement the GA objective function. In this way, an "optimal" set of Torso Driven Walking tuning parameters is generated for the new environment. The performance of the resulting control systems are compared with the results obtained in Chapter 6.6. Despite the potential complications introduced through the use of a three dimensional physics engine that incorporates frictional forces in its model, it is found that the TDW controlled robots running in the NGD environment have outperformed the robots operating in the "Legbot" simulation environment.

Finally, the strategy of continuing to evolve a population of individual solutions in a new environment is investigated. Using this process, it was hoped that a set of tuning parameters that balance the robot in the second environment could be found in a significantly smaller number of trials than would be the case if the GA training was started from scratch. A system exhibiting this behaviour would be particularly advantageous when the time comes to port the control system to a physical robot. A shorter training cycle would imply reduced wear, tear, and possibly damage, to the physical robot. Unfortunately, the results suggest that this is not the case for populations evolved using the genetic algorithm described in this thesis.

### 6.7.2 Materials

#### 6.7.2.1 Simulation environment

i. Simple DirectMedia Layer (SDL) graphics library, [72].

This is a cross-platform multimedia library designed to provide support for integrating UI input devices with an OpenGL graphics implementation. This library is used to capture mouse inputs, and to output a graphical representation of the scene generated by the physics engine.

ii. Physics Abstraction Layer (PAL) software library, [84].

This software library integrates a number of physics libraries, including the NGD dynamics engine, with the SDL graphics library. In addition the PAL allows software developers to describe the scene and objects within the scene via a simple XML notation. In this way experimental parameters, such as the mass of robot links and the coefficient of restitution between material types can be altered without requiring the entire simulator to be re-compiled.

iii. Newton Game Dynamics (NGD) simulation environment, [71].

Newton Game Dynamics is a free[33] library designed to provide real-time simulation of physical environments. The simulator uses a proprietary deterministic solver, which is not based on traditional LCP or iterative methods. The developers claim the engine possesses the speed and reliability of both methods. Details on the algorithm used to implement the engine are not available, but some broad characteristics can be inferred by the classes and interfaces used by the library.

Some of the main differences between NGD and my own simulation engine (described in Appendix C) include:

- Three dimensions. The NGD simulator is a three-dimensional physics engine.
- Friction and slippage between feet and the ground. The Legbot simulator assumed there would be no slipping of the contact points between the robot and the ground. NGD does not make this assumption.

[33] Downloading and use of NGD requires acceptance of a license agreement requiring acknowledgement of any use of NGD in a commercial or shareware product built using the library.

- Minimum update rate. The shortest update rate between "frames" that can be used with NGD is 1/600 seconds. The Legbot simulator allows for frame rates as fast as 1/1000 seconds. Generally, a shorter interval between frames results in improved accuracy at the expense of calculation speed.
- Minimum joint masses. The NGD system is designed to model objects significantly larger and heavier than the links making up the Legbot robot. As a result some inconsistent behaviour was encountered when trying to simulate small and light objects using NGD. This problem is exacerbated when using a short frame interval.
- In the Legbot simulation system, the behaviour of a dynamic system of interconnected links is predicted by generating and solving a system of linear equations relating joint angular acceleration to force applied at contact points between the body and external objects. This approach ensures the integrity of link positions relative to each other is maintained.
- In the NGD environment, each link within a body is independently animated. Joint integrity appears to be maintained through the application of an "elastic" force holding links together through the joint. It is possible to separate links if enough force is applied causing the links to pull apart.

In summary, the NGD simulator is much faster than the Legbot simulator (Appendix C), and incorporates some new features (for example, a third dimension, and the effects of friction). On the other hand, the Legbot simulator allows for shorter time intervals between simulation "frames", and is able to continue operating with lighter and smaller objects. In addition, the Legbot simulator guarantees that the spatial relationship between the robot links is maintained.

#### 6.7.2.2 Control system

The experiment utilises the "Torso Driven Walking control system, with Genetic Algorithm parameter tuning". This configuration is described in detail in Chapter 6.6.

### 6.7.3 Methods

This experiment has been conducted in three stages:

- Direct port of control system and tuning parameters to the NGD model.
- Evolution of new solutions within the NGD environment.
- Investigate evolution shared between the two environments.

#### 6.7.3.1 Direct port of control system and tuning parameters to NGD model

The control system and tuning parameters determined in Chapter 6.6 are taken and directly implemented on a three dimensional robot modelled using the NGD physics engine. In the port of the simulated robot to the NGD environment it was necessary to make some minor changes to the robot's structure:

- The robot's feet are no longer "point" feet, but have instead been modelled using boxes. Boxes were chosen to represent the feet because the PAL system does not allow for the placement of horizontally aligned cylinders. Each foot has a cross-section of (0.01m x 0.01m), and is 0.17m wide.
- In order to prevent the robot's "shin" links from colliding with the ground, or otherwise interfering with the feet during the simulation, the shins have been made very thin. The box making up each shin has dimensions of 0.15m high by 0.0035m across.
- The NDG simulator does not allow animation of objects with less than 0.01kg mass, and assumes that any such objects are immobile parts of the scene. As a

result, the robot's feet have been implemented with a mass of 10g, the minimum mass the model will accept.

Figure 6-24 provides an illustration of the simulated robot, as implemented within the NGD environment.

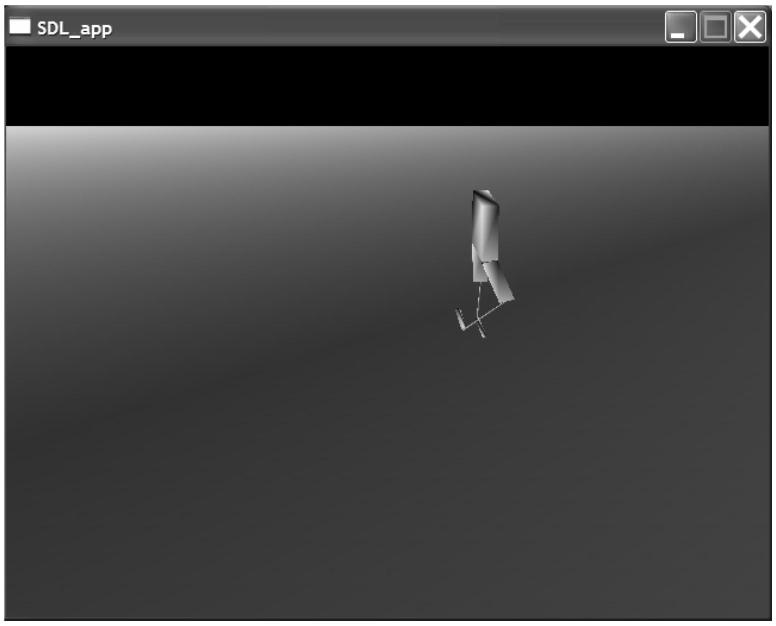


**Figure 6-24 A screenshot of the simulated Legbot, balancing within the "Newton Game Dynamics" environment. Besides the third dimension supported by the physics model, the most significant difference between this robot and the simulation used in Chapter 6.6 is the shape of the robot's shin and foot links. Here these links have been made as narrow as possible to try and approximate the behaviour of the "point" contacts assumed by the original dynamic model.**

#### 6.7.3.2 Evolution of new solutions within the NGD environment

In the second stage of this experiment, the genetic algorithm tuning process is restarted, running entirely within the new NGD environment. The starting population for this second evolution is randomly generated, in the same way as the trials described in Chapter 6.6.

Apart from the changes to the simulated robot's structure identified in 6.7.3.1, use of the Newton Game Dynamics simulation environment by the GA/TDW tuning process required some minor modifications to the objective function implemented by genetic algorithm itself. In the Legbot simulator version of the GA trainer, a set of tuning parameters was evaluated using an objective function that accumulated a fitness score over the duration of a trial. Solutions which accumulate larger scores are judged to be "better" solutions than those with a lower fitness score. The rate at which the objective score accumulates during the course of a trial is governed by the distance of the robot's torso from a set point position that moves following a predetermined trajectory over the course of the trial. A trial is halted after 60 seconds have elapsed, or if a predetermined termination condition has been met.

Obviously, the termination conditions will have a large impact on an individual's fitness score by preventing further accumulation of fitness score for the individual solution being evaluated. The objective function is described in detail in 6.6.3.2, and apart from some modifications to the trial termination conditions, has been largely re-used in this experiment. Changes to the trial termination conditions included:

i. Removal of the foot proximity termination condition

The "Foot proximity" termination condition has been removed for this experiment. In the Legbot simulation system a trial was terminated if the robot's feet were judged to be within an arbitrary "safe" distance (~2 cm) of each other. This was an attempt to avoid complications that might be caused by collisions between the robot's feet when moving the control system to a three dimensional robot. The planar Legbot simulator was not able to model the effects of a possible collision, and so the GA training system simply failed any trials which may have resulted in a collision. In many cases, solutions which probably would not have caused the robot to trip or fall were rejected because the robot's feet came too close together during the course of the trial.

Since the NGD dynamic simulator is a three dimensional physics engine, designed to predict the results of collisions between components of articulated objects, there is no need to implement an arbitrary foot proximity termination condition. If the feet come too close together, and cause the robot to fall, then the simulation will be terminated as a result of the robot falling. If there is only incidental contact between the robot's feet, and the robot does not fall, then there is no need to punish the set of tuning parameters being evaluated. This means that many possible solutions that were not possible in the "Legbot simulator" environment can score quite highly in the NGD system.

ii. Fall detection

In both Legbot and NGD simulation systems a trial is terminated when the robot is judged to have fallen. The Legbot simulator decides a trial robot has fallen if any part of the robot but its feet touches the ground. The NGD system determines a trial has fallen if any corner of its torso comes within 20mm of the ground. In each system the choice of fall detection used was made as a result of the information conveniently available from the respective simulation systems. Both approaches are reasonable, and no obvious differences in resulting fitness scores were noticed.

iii. Frozen individuals

As has been previously mentioned, when working with small objects and time intervals, the NGD system sometimes fails to operate correctly. When this happens, the simulator no longer updates the positions of the objects it is modelling, effectively freezing the scene. This behaviour was experienced during the GA training for a small number of potential solutions. Simply applying a particular set of Torso Driven Walking tuning parameters to a trial would occasionally result in such a frozen scene. In these cases, the objective function has no choice but to terminate the trial, awarding the individual a low objective score. It may be the case that many of these low scoring solutions would otherwise have performed very well, so some potentially high quality solutions may have been discarded.

#### 6.7.3.3 Investigate evolution shared between the two environments

The final stage of this experiment investigates the possibility of partially training a population of solutions in one environment, and then completing the training in a second environment. To do this, a number of experiments are conducted applying the GA training system running in the new NGD environment, as described in 6.7.3.2 above. However, instead of beginning each trial with a randomly generated set of individuals making up the initial population, a number of individuals evolved by the genetic algorithm within the Legbot environment are included. The composition of individuals in the initial population was varied to see if there were any advantages to be gained by transferring individuals evolved in one system to the other.

### 6.7.4 Results

#### 6.7.4.1 Direct port of control system and tuning parameters to NGD model

Upon moving the best solutions evolved on the Legbot simulation system to an NGD environment, a few observations are immediately apparent. While the robot moves in a similar manner to the planar simulator, its behaviour is not identical. The most obvious difference is that the robot’s feet tend to slip as they push against the ground to balance the torso, which would mean any resulting balance correction is less effective than in the Legbot environment. Figure 6-25 depicts the result of a typical trial as the robot attempts to maintain balance while keeping the torso close to a set point position.

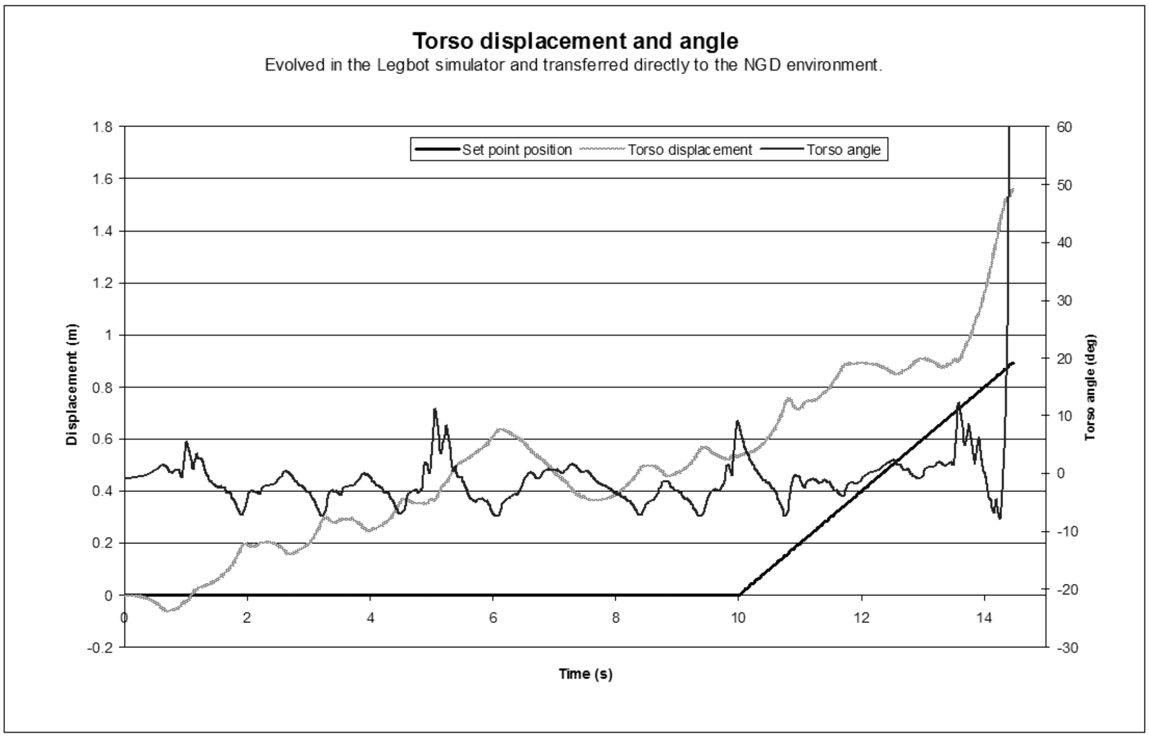


**Figure 6-25 A Torso Driven Walking implementation is transferred without any modification from the Legbot simulator where it was evolved to the NGD test environment. The robot balances for approximately 30 steps, before falling over.**

In this example, the robot balanced for approximately 30 steps, after which the robot falls over. Other individual solutions ported directly to the NGD environment became “stuck” after balancing for about 10 seconds, standing in one place with both feet on the ground, as illustrated by Figure 6-26.

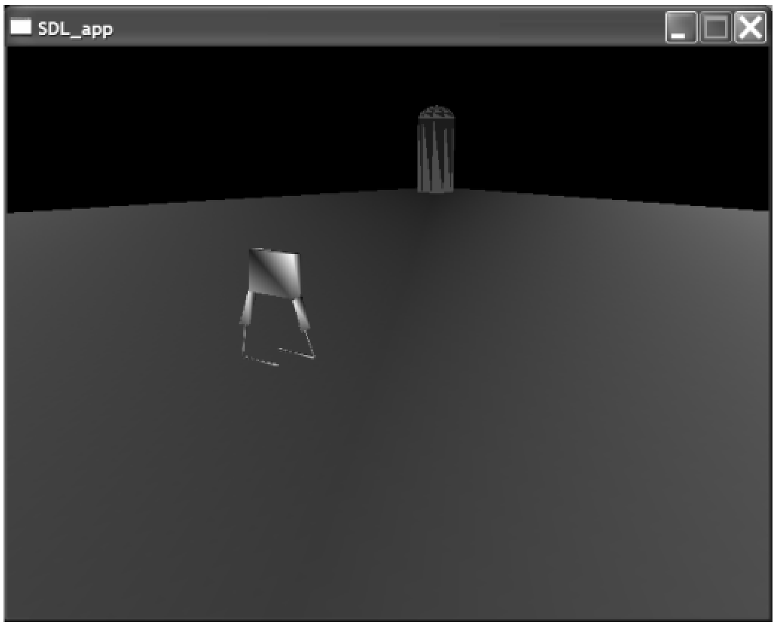


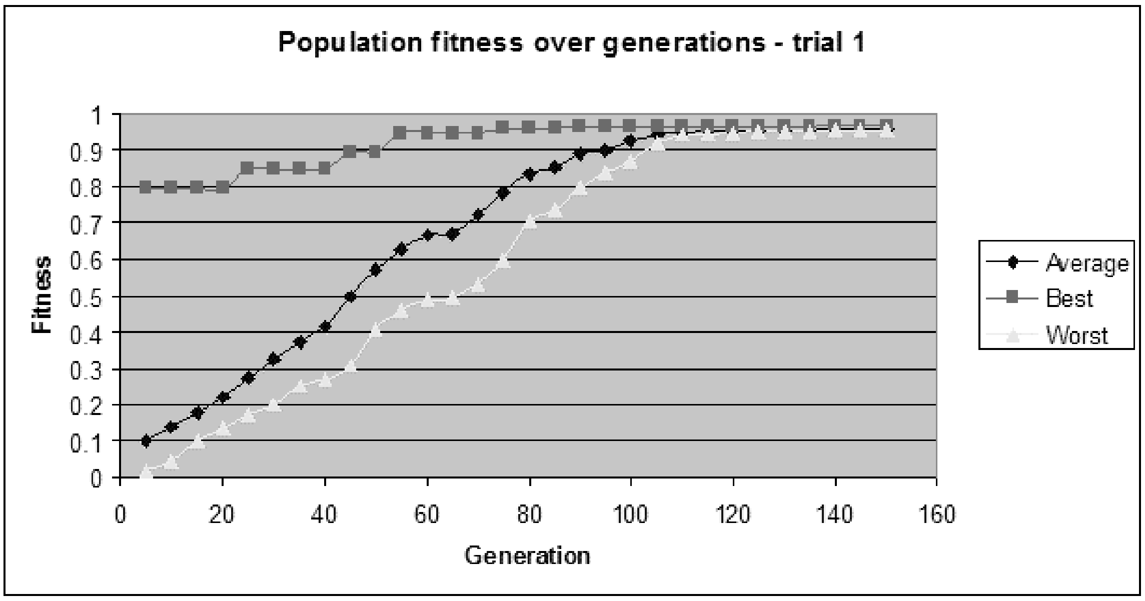
**Figure 6-26 Although some solutions ported directly to the NGD environment allow the robot to maintain balance, the resulting control systems are ultimately judged to fail as the robot becomes stuck in position with both feet on the ground.**

During the initial stages of the trial, the robot behaved in a similar fashion to the same control system implemented in the Legbot environment where it was evolved. This suggests that while the Legbot and NGD dynamic models are not identical, their outputs are quite similar. Over time the system states of the two trials diverge as small variations in the dynamic models are compounded.

While the trial shown in Figure 6-25, and others like it, demonstrated a reasonable performance, none of the best performing solutions evolved on the Legbot simulator can really be considered successful when transferred directly to the NGD environment. While many maintained balance for a number of steps, all failed within a short time. The most common failure modes occurred when the robot's feet slipped, either causing the robot to sink to the ground doing the "splits", leaving the robot stuck standing with its feet too far apart to continue moving, or resulting in the robot falling. It was fairly clear that at least some training will be required in the new environment before a good set of training parameters could be found.

### 6.7.4.2 Evolution of new solutions within the NGD environment

The evolution of Torso Driven Walking tuning parameters within the NGD environment proved to be very successful. Figure 6-27 depicts the training profile of two typical evolution trials.



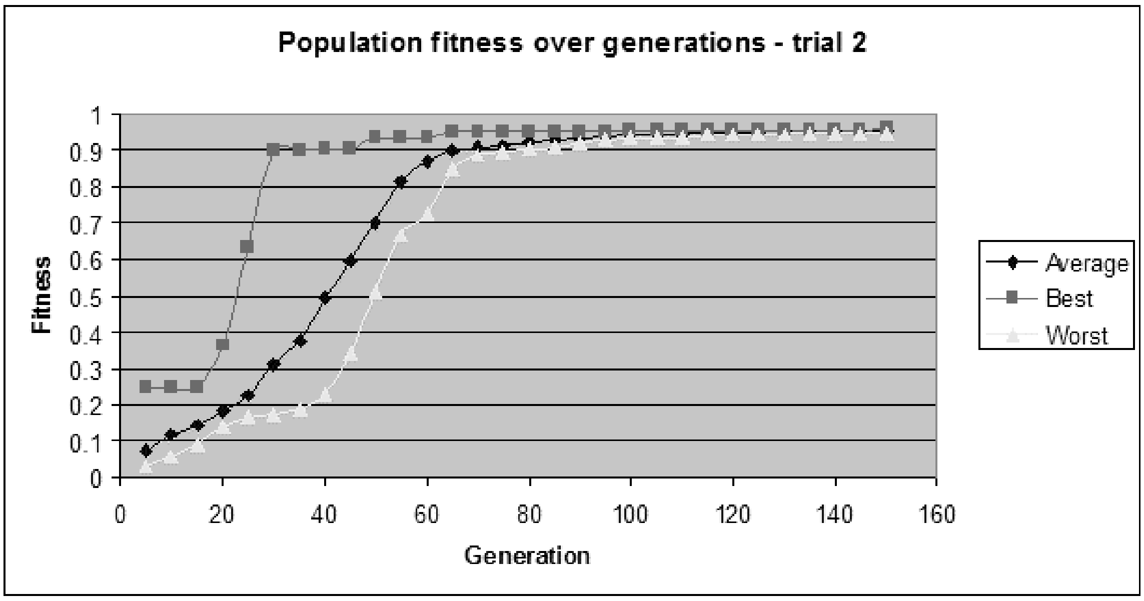


**Figure 6-27 When running in the NGD environment, the genetic algorithm tuning system quickly determines a set of parameters allowing the Torso Driven Walking control system to track a moving target set point position while maintaining balance.**

In both cases the systems quickly converge to a set of solutions scoring very highly on the algorithm's objective function. A fitness score of ~0.95 implies that the robot torso maintains an average distance from the moving target set point position of about 30mm.

These high performing solutions are found reasonably quickly. In both trials, the quality of the best performing individuals in the population is close to 0.9 after only 40 generations – requiring less than 240 individual experiments to be conducted.

Figure 6-28 demonstrates the performance of one of the final TDW implementations evolved using this process, as the robot follows a moving target position.

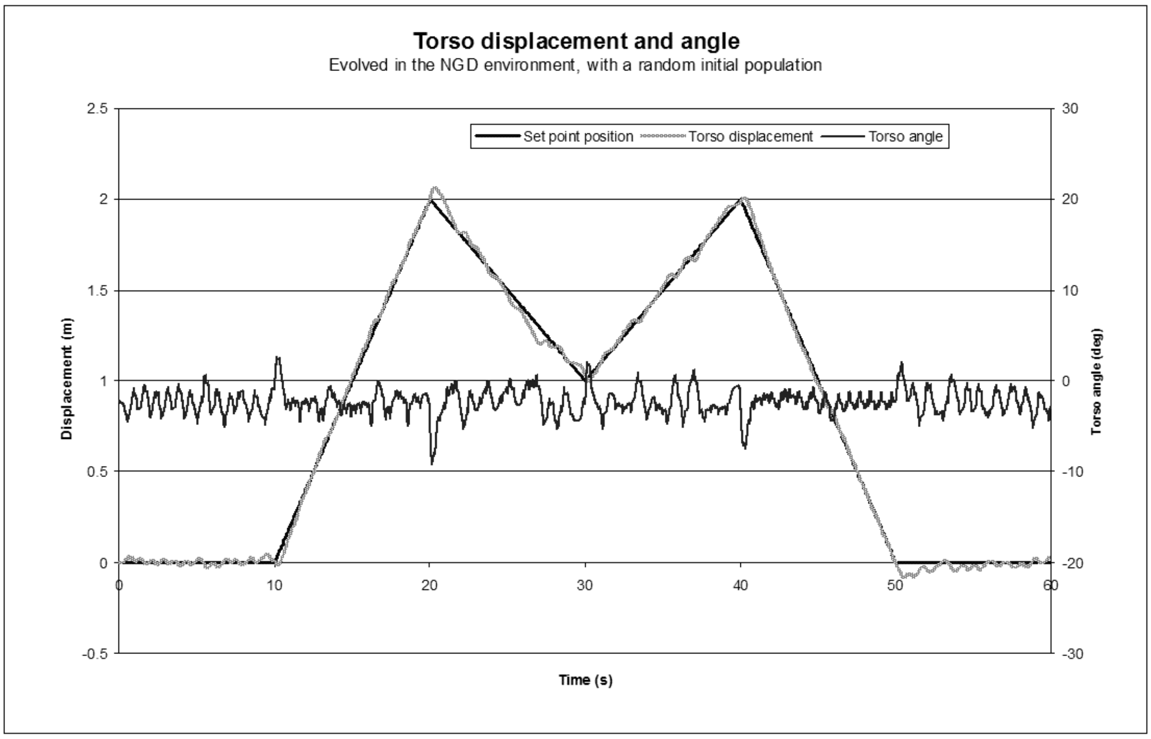


**Figure 6-28 After using the GA algorithm to train a Torso Driven Walking control system in the NGD environment, a simulated robot is able to closely track a set point position moving with variable velocity, while maintaining balance.**

A video of this trial, together with a demonstration is available from the URL: http://www.ee.uwa.edu.au/~suthe-aj/thesis/legbot/ngdwalking.avi

### 6.7.4.3 Investigate evolution shared between the two environments

As the results from 6.7.4.2 illustrate, evolving a set of tuning parameters for the Torso Driven Walking control system gave high quality results when the initial population is randomly generated. Within a reasonably short number of trials, the individual solutions within the population were able to achieve very high scores on the genetic algorithm's objective function. Despite this good result, a number of trials were conducted to see if seeding the initial population with individuals evolved in another environment might speed up the training process.

The following tables present the results obtained when investigating this possibility:

i. The entire population moves to the new environment.

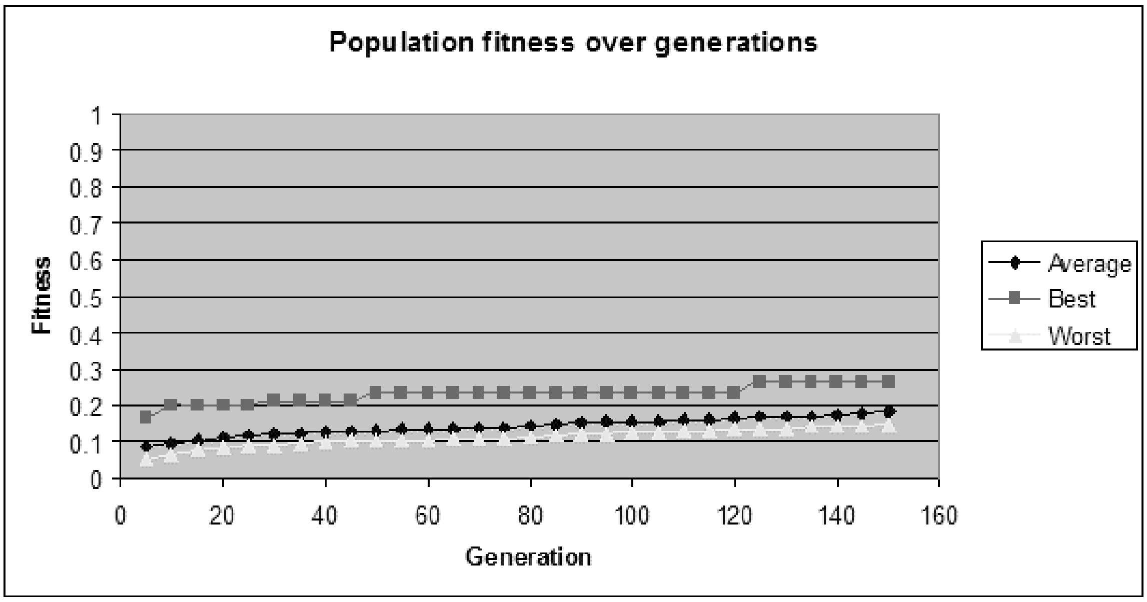


**Figure 6-29 When the entire initial population is made up of individuals resulting from a long period of evolution in another environment, the population is not able to adapt well, or quickly to the new environment.**

In this first trial, the entire initial population is made up of individuals that were the culmination of 150 generations of evolution within the Legbot simulation environment. In their original "Legbot simulator", the best individuals scored close to 0.45 on the genetic algorithm's fitness score, a result which saw the robot maintain balance for the entire duration of the test, while following the moving target position reasonably closely (see 6.6.4). When moved to the new environment, the same individuals scored between 0.05 and 0.15. Most of the trials failed as a result of the robot's feet slipping out from under it (as has been described in 6.7.4.1).

As Figure 6-29 demonstrates, this population continued to perform extremely poorly as evolution was continued in the new NGD environment. The most likely reason for this is that the individuals had over-specialised to conditions in the Legbot environment, in particular the solutions may be taking advantage of their ability to apply any horizontal force to the ground without causing the robot's feet to slip. When the individuals were transferred to the new environment, there was not enough variation in the solutions making up the population to allow them to quickly adapt to the new conditions. Only a slow improvement in the quality of the population can be seen – this slow improvement is the result of random, beneficial mutations generated by the genetic algorithm.

ii. Half of the population is transferred to the new environment.

In the second trial, illustrated by Figure 6-30, half of the initial population is made up of individuals that evolved within the Legbot simulation environment. The remainder of the initial individuals are randomly generated.

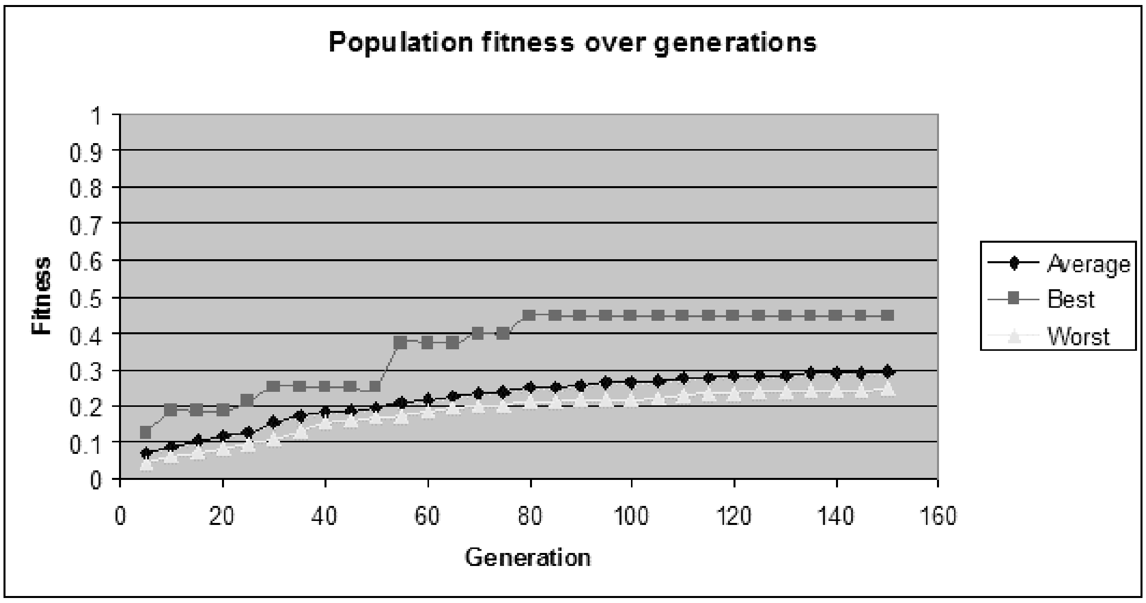


**Figure 6-30 When 50% of the initial population is made up of individuals evolved in another environment, the population is still not able to adapt well, or quickly to the new environment. The results are roughly twice as good as the case when 100% of the population is transported.**

While the population's performance has improved, when compared to the previous case, such a composition of initial population members still give far inferior results to the situation where the entire population is randomly generated. Despite the improved quality scores, even the best of these individuals fail to complete the entire 60 seconds of the trial duration, before they either fall or get stuck.

iii. A partially evolved population is moved to the new environment.

In the third trial, the entire population was transferred from an intermediate stage in their evolution within the Legbot environment. In this case, the population members moved had experienced 60 generations of evolution before transferring to the NGD based system. It was hoped that this population might not yet have become "over-specialised", and that the solutions would quickly adapt to changed conditions. As Figure 6-31 shows, this was not the case.

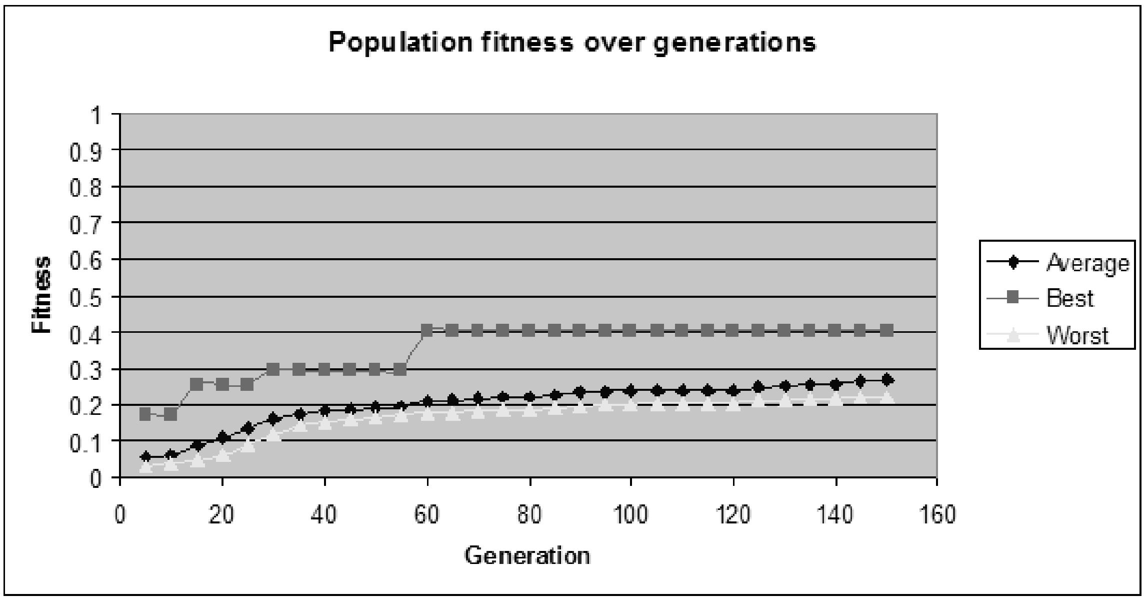


**Figure 6-31 Here the initial population is made up of individuals resulting from an incomplete evolution in the Legbot environment (60 generations). Once again, the population is still not able to adapt quickly to the new environment.**

The results obtained from seeding the initial population with individuals resulting from a partial evolution in the Legbot environment were also disappointing. The performance of the individuals after 150 further generations of evolution is roughly equivalent to the results when half of the population is randomly generated. This is true in terms of fitness score, as well as a subjective judgement of performance.

It appears from these results that initial training of the control system parameters in one simulated environment, with a view to completing the evolution in a second physical or simulated environment is not going to result in a shorter number of training cycles in the new environment. This does not mean that good solutions cannot be found this way, as in all the experiments, the population fitness continues to improve slowly as a result of mutations introduced by the algorithm. While improved training performance may be obtained by modifying the genetic algorithm itself (for example, by increasing the mutation rate), this approach is unlikely to ever prove to be faster than simply restarting the evolution from scratch.

### 6.7.5 Discussion

In this experiment, the adaptability of the Torso Driven Walking control system was investigated by implementing it within a second, independently developed, simulation system. The new simulator extended the dynamic model to a full three dimensions, as well as including the effects of friction and resulting complications caused when the robot's feet slip along the ground.

The initial trials in the new simulator were conducted with individual solutions that had been evolved to operate within an environment provided by the Legbot simulation system. While these individuals were not very successful, the best of them were able to cause the robot to take a reasonably large number of steps before it fell over or became stuck. This demonstrates that while the simulation environments are obviously different, over short time periods the system state predictions output from each model are similar. This implies that my own Legbot simulation environment is reasonably accurate, since the behaviour of complex objects modelled in each of the two environments is consistent.

Further investigations showed that when the same genetic algorithm tuning techniques are used to tune the Torso Driven Walking control system in the new environment, control system implementations are quickly found that successfully balance the robot while closely tracking a moving target position. In fact, the resulting solutions outperform the best of the Torso Driven Walking implementations hosted within the original Legbot simulation system. This result proves that the GA tuned Torso Driven Walking control system is reasonably robust. As long as a new set of tuning parameters can be evolved, the TDW approach is able to cope well with the inevitable differences in behaviour of bodies modelled in two fundamentally different dynamic simulators. This is good news for plans to port the control system to a physical robot, as it is fair to say that a similar disparity will exist between each of the two models predictions and a real world system's behaviour.

One of the major issues encountered by adaptive learning systems is the large number of trials needed to train the controller. This disadvantage can be seen in these experiments by the approximately 200 trials that needed to be conducted during evolution of the Torso Driven Walking tuning parameters. While this number of trials is fairly modest in comparison to many other adaptive learning systems, it still represents a significant cost in terms of man-hours and equipment wear and tear if the training is to be performed on a physical robot.

In the final part of this experiment a process for taking a pre-evolved population of tuning parameters sets from one environment, and continuing evolution in a second environment was investigated. It was hoped that this process could result in a reduced number of trials required in the target environment in order to generate an appropriate set of tuning parameters for the control system. Unfortunately, this did not prove to be the case, with the pre-evolved individuals actually having a detrimental impact on the learning performance of the algorithm. This is because the transported individuals have two impacts on the GA population into which they are introduced:

i. The new individuals were overspecialised.

As an example of this problem: in the experiments conducted here, individuals that had evolved in the Legbot environment were able to apply a reasonably high torque to their supporting leg in order to maintain balance. This is because it was impossible for the robot's feet to slide along the ground in the Legbot simulator. When the individuals moved into the NGD environment, the same level of torque caused the robot's feet to slip – eventually causing the robot to fall.

ii. The new individuals reduced "genetic" diversity.

This influence occurred in two ways. First, since most of the introduced solutions were very similar, the effect of introducing a number of them into a population was equivalent to cloning a single individual a number of times. As populations are a fixed size in this experiment, this directly reduces diversity. Second, in the initial stages of an evolution, most of the randomly generated solutions will perform very badly. Since the transported Legbot solutions do work for a short time, their fitness outscores the random solutions. As a result, the transported solutions are selected to be parents of the new generation, and the randomly generated individuals tend to be eliminated.

The results from the trials conducted in this series of experiments operate in their original environments lead to the conclusion that the Torso Driven Walking control system is a feasible candidate for implementation on a physical robot, with the proviso that GA tuning of the control system parameters will need to be conducted from scratch.

## 6.8 TDW comparison with Virtual Model Control

**Abstract:** In order to understand better advantages that may be realised through using the Torso Driven Walking control system to drive a biped robot, a comparison was conducted between the TDW control system and an implementation of Pratt's "Virtual Model Controller", following Pratt's description in [85]. Each system is tested within the same simulated environment, and is asked to execute the same set of simple test procedures. It is shown that TDW is more controllable and requires less joint torque than a VMC algorithm implemented on the same platform. In addition, the TDW control system can robustly handle gyroscope "bias" errors, while the VMC controller fails in the presence of a bias error in inclination measurement.

### 6.8.1 Introduction

Virtual Model Control is an attractive solution for controlling robotic systems because it explains the control system in an easily visualised set of rules that are designed to make intuitive sense. The control system shares some of its more general characteristics with the Torso Driven Walking algorithm, including the fact that the robotic platforms they are developed to control share an identical morphology. Both robotic platforms can be described by a 5-link planar system model. Both robots walk on "point" feet, and are controlled through the application of a variable torque to the robot's joints. The algorithms use the same set of system variables to make their control decisions. This symmetry in system model and sensor requirements makes the VMC algorithm a good choice of control system with which to compare the Torso Driven Walking control system.

As well as sharing common target platform morphology, there are similarities in the control systems themselves, including:

- In both algorithms, foot placement is arbitrarily determined, with the control of the raised leg managed by a state machine driven by the position of the support leg with respect to the robot's torso.
- Both systems decompose the problem of control system design into distinct components.
- Neither algorithm requires (or can use) the concept of "supporting polygon" common to many traditional biped control systems.
- Both algorithms require extensive tuning before a satisfactory implementation for a specific target system can be found. However in each case the tuning parameters are meaningful and the effects of modifying a parameter can be easily visualised.

Despite the similarities, there are fundamental differences in the design of each control system, which manifest in very different characteristics in the resulting robot behaviour. Some of these differences include:

- The VMC system is designed to maintain a constant hip altitude. This necessitates a "bent kneed" walking style.
- The hip joints in the VMC system are only used to control the inclination of the robot's torso. There is no positional component used in the control of the torso's inclination.
- With the hip joints dedicated to torso inclination control, the knee joints therefore are responsible for all other aspects of control – maintaining torso altitude, as well as controlling the robot's horizontal velocity.
- In an attempt to avoid presenting the control system with virtual forces that cannot be achieved by the robot, in a VMC system velocity control is only attempted during the double support phase of the robot's gait.

In order to appreciate the differences in the control robot behaviour resulting from each of these two approaches (VMC and TDW), both control systems are implemented on the same simulated robotic system.  The performance of each system will be evaluated with respect to a number of key areas, including:

- The ability of the control system to maintain balance, while moving at a controllable (and variable) velocity.
- The maximum torque the control system demands of its joint actuators.
- The implementation/commissioning complexity.
- Noise tolerance of the control system, with regard to accuracy of joint angle estimates.
- Noise tolerance of the control system, with regard to gyroscope bias or “drift” errors in inclination estimates
- The smoothness of torso motion during operation of the control system.

In almost all of these areas, the Torso Driven Walking control system compares very favourably with Virtual Model Control.

### 6.8.2 Materials

#### 6.8.2.1 Simulated robot platform and environment

This experiment is conducted on the three dimensional physics engine, and uses the simulated Legbot robot, both of which are described in Chapter 6.7.

#### 6.8.2.2 Torso Driven Walking control system

To contrast with the VMC control system, this experiment utilises the “Torso Driven Walking control system, with Genetic Algorithm parameter tuning”.  This system is described in detail in Chapters 6.6 and 6.7.

### 6.8.3 Methods

#### 6.8.3.1 Implementing Virtual Model Control (VMC)

Pratt has published his VMC algorithm for controlling a walking biped robot [85], together with information about the size and shape of his target robot systems. It so happens that each of Pratt's target robots shares its basic morphology with the Legbot planar robot. Pratt's robots are planar bipeds, with a pair of two-link legs. They can both be considered to have either "point" feet, or (in the case of the "Spring Flamingo" robot) "limp" ankles which can be modelled as point feet. These circumstances, together with the fact that Pratt's VMC controller is widely recognised as a successful walking algorithm make the VMC control system an attractive option with which to compare with the Torso Driven Walking controller.

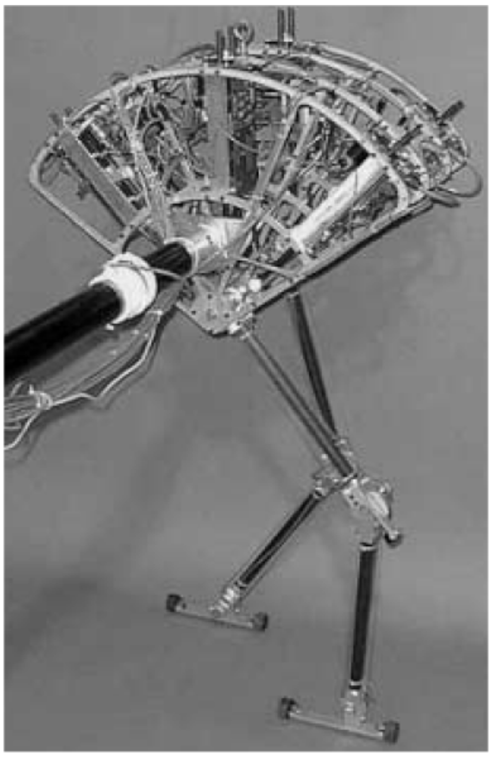


Image source: Pratt, et. al. "Virtual Model Control: An intuitive approach for bipedal locomotion", [85].

**Figure 6-32 An image of the "Spring Flamingo" robot, used by Pratt as one of the target systems for his Virtual Model Control biped walking algorithm. This robot shares many characteristics with the Legbot robot, including: Its motion is constrained, making it a planar robot. It is composed of five actuated links, a torso, two thighs and two shins. Although the robot has feet, they are unactuated and limp, allowing the robot's feet to be modelled as point feet.**

Figure 6-32, an image copied from Pratt's paper [85], illustrates the basic structure of his "Spring Flamingo" robot.

### i. Review: What is Virtual Model Control?

Since the VMC controller has already been introduced in Chapter 2.3.4, only a brief summary of some of the implementation details will be made here. When designing a VMC control system, a researcher adds a number of imaginary components that exert forces to the system being controlled in an intuitive way. These virtual components (usually spring/damper combinations) are arranged such that if they existed, the target robot would tend to behave in a desirable way. In Pratt's VMC biped algorithm, the major virtual components include a "granny walker", a "dog track bunny" and a "virtual linkage". The granny walker is responsible for holding the robot's torso off the ground and upright, but does not constrain its horizontal movement. The dog track bunny is used to regulate the robot's horizontal velocity, "pulling" the robot after it. The virtual linkage is used to raise the swing leg off the ground, and cause it to mirror the stance leg before setting down at the nominal stride length during the transition to double support phase. These three components will be described more fully in the following sections.

### ii. Torque from Force

Once a desired virtual force has been calculated from an analysis of the current state of the virtual components, an appropriate set of robot joint torques must be determined that will have the same effect as the virtual forces acting on the robot. In his paper [85], Pratt supplies the equations he has used to calculate the joint torques required to mimic the effect of the virtual force applied by the VMC components. To generate these equations, he uses the Jacobian matrix relating joint angular velocity to the velocity of the robot's end effector in order to associate desired "virtual" force with required joint torques. This use of the Jacobian is not necessarily accurate in a dynamic system as link masses and moments of inertia are not taken into account. However, if the leg masses are very light, and angular velocities small, it should be an acceptable approximation. In this experiment, Pratt's approach has been followed when calculating joint torque in the implementation of VMC.

iii. Virtual model control of the supporting leg in the single support phase

Whenever the robot is standing with one foot on the ground, the supporting leg is controlled through the application of a virtual "granny walker" component. This structure is composed of two spring/damper systems that work together to push the robot away from the ground while keeping the torso upright. Figure 6-33 shows the granny walker component applied to the Legbot robot.

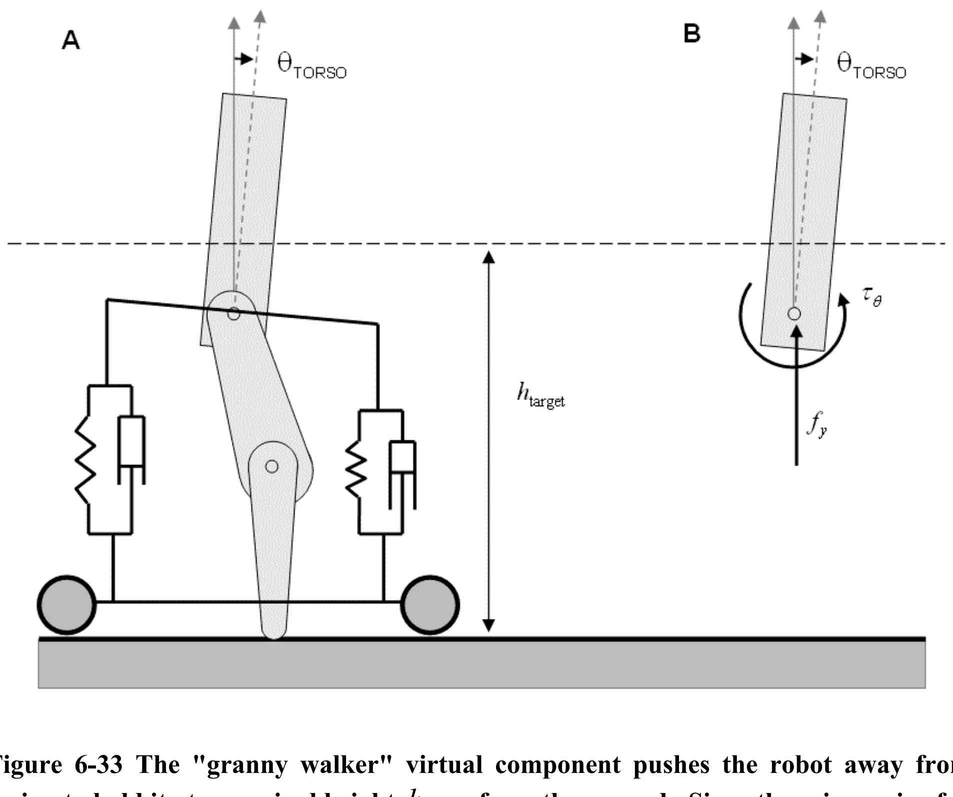


**Figure 6-33 The "granny walker" virtual component pushes the robot away from the ground, trying to hold it at a nominal height $h_{\text{target}}$ from the ground. Since there is a pair of spring/damper components, they work together to generate torque acting on the robot's torso. This torque holds the torso upright. Image (A) shows the VMC component, while (B) the corresponding virtual forces acting on the robot's hip.**

As the robot's hip joint drops below the target hip altitude $h_{\text{target}}$, the granny walker component exerts an imaginary vertical force $f_{\text{y}}$, and an imaginary torque $\tau_\theta$ to the hip of the robot, such that:

$$f_{\text{y}} = k_{\text{y}} \cdot \left(h_{\text{target}} - y_{\text{hip}}\right) + b_y \frac{\Delta\left(h_{\text{target}} - y_{\text{hip}}\right)}{T} \tag{1}$$

$$\tau_\theta = k_\theta \cdot \theta_{\text{torso}} + b_\theta \frac{\Delta\theta_{\text{torso}}}{T} \tag{2}$$

Where:

| | | | |
|---|---|---|---|
| $f_{\text{y}}$ | Vertical force acting on the hip. | $\tau_\theta$ | Anti-clockwise torque acting on the hip. |
| $k_{\text{y}}$ | Vertical spring stiffness. | $k_\theta$ | Rotational spring stiffness. |
| $b_y$ | Vertical damping coefficient. | $b_\theta$ | Rotational damping coefficient. |
| $y_{\text{hip}}$ | Height of the hip above the ground. | $\theta_{\text{torso}}$ | Torso inclination angle. |
| $h_{\text{target}}$ | Spring set point altitude. | $T$ | Control system update interval. |

For level-ground walking, the spring set point altitude ($h_{\text{target}}$) is a constant value. The choice of set point altitude will influence a number of gait factors, including the length of the double support phase, the energy efficiency of the gait and the maximum step length. For a robot with an extended leg length of 60 cm, Pratt used a set point altitude of 54 cm. Given that the Legbot's legs are 30 cm long, in this experiment a corresponding set point altitude of 27 cm is used.

$$h_{\text{target}} = 0.27 \text{ m} \tag{3}$$

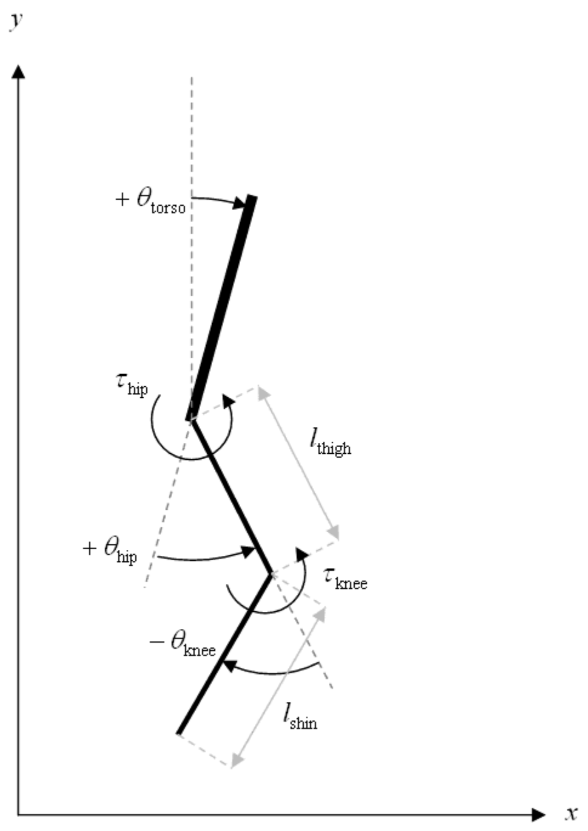


**Figure 6-34 Single leg state measurements available to the Legbot robot. These angles can be substituted into Pratt's equations for determining required joint torques given virtual forces. Note that the direction of positive torso angle measurement is opposite in sign to that of the knee and ankle joints.**

Once the virtual forces and torques are known, Pratt uses the Jacobian matrix to relate the virtual forces $f_{\text{y}}$ and $\tau_{\theta}$ to a set of "equivalent" hip and knee joint torques, $\tau_{\text{hip}}$ and $\tau_{\text{knee}}$. Following Pratt's procedure using the available Legbot system state measurements shown in Figure 6-34, the equations relating virtual force to joint torque are described by (4):

$$\begin{bmatrix} \tau_{\text{knee}} \\ \tau_{\text{hip}} \end{bmatrix} = \begin{bmatrix} \frac{-l_{\text{shin}}\, l_{\text{thigh}} \sin(\theta_{\text{knee}})}{l_{\text{shin}} \cos(\theta_{\text{hip}}+\theta_{\text{knee}}-\theta_{\text{torso}})+l_{\text{thigh}} \cos(\theta_{\text{hip}}-\theta_{\text{torso}})} & \frac{-l_{\text{shin}} \cos(\theta_{\text{hip}}+\theta_{\text{knee}}-\theta_{\text{torso}})}{l_{\text{shin}} \cos(\theta_{\text{hip}}+\theta_{\text{knee}}-\theta_{\text{torso}})+l_{\text{thigh}} \cos(\theta_{\text{hip}}-\theta_{\text{torso}})} \\ 0 & -1 \end{bmatrix} \begin{bmatrix} f_y \\ \tau_{\theta} \end{bmatrix} \qquad (4)$$

Using Equations (1-4), hip and knee torques are calculated for the supporting leg during the single support stage of the robot's walk.

iv. Virtual model control of both legs in the double support phase

The same granny walker component described for the single support phase also applies while the robot operates in the double support phase, resulting in a virtual vertical force and counter-clockwise torque being applied to the robot torso, as described by Equations (1-3). In addition to the “granny walker”, during the double support phase of the robot’s walk a “dog track bunny” component is added.

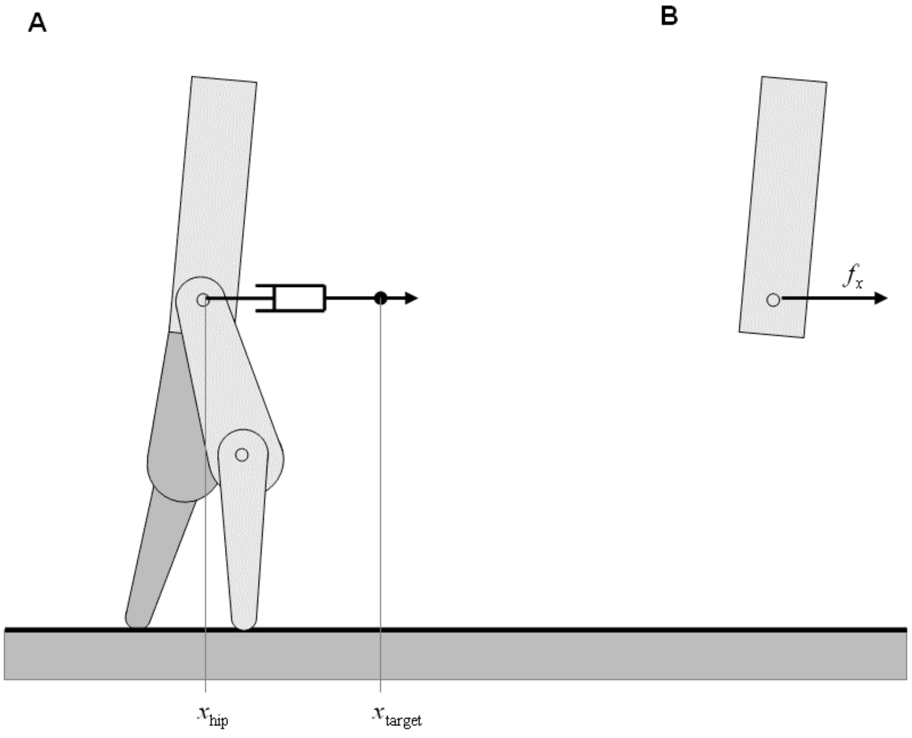


**Figure 6-35 The "dog-track bunny" virtual component exerts a horizontal force to the robot's torso, with a magnitude proportional to the difference in velocity between the robot and a moving target set-point. Image (A) shows the VMC component, while (B) the corresponding virtual force acting on the robot’s hip.**

This “dog-track bunny” component consists of a damper mechanism connecting the hip joint to a target position moving horizontally at the desired velocity. Figure 6-35 shows the virtual dog-track bunny component attached to the Legbot robot.

As the target set point moves away from the robot, a horizontal virtual force $f_x$ is generated, working to match the robot’s velocity to that of the target, such that:

$$f_x = b_x \frac{\Delta\left(x_{\text{target}} - x_{\text{hip}}\right)}{T} \qquad (5)$$

Where:

| | | | |
|---|---|---|---|
| $f_x$ | Horizontal (virtual) force acting on the hip. | $b_x$ | Horizontal damping coefficient. |
| $x_{\text{target}}$ | Target torso position. | $T$ | Control system update interval. |
| $x_{\text{hip}}$ | Horizontal torso displacement. | | |

The horizontal force generated by the dog-track bunny is applied simultaneously to the vertical force and counter-clockwise torque output by the granny walker to specify the full set of virtual forces being applied to the robot's torso during the double support phase.

Figure 6-36 illustrates the state joint angle measurements and torques of the Legbot robot during the double support phase of walk.

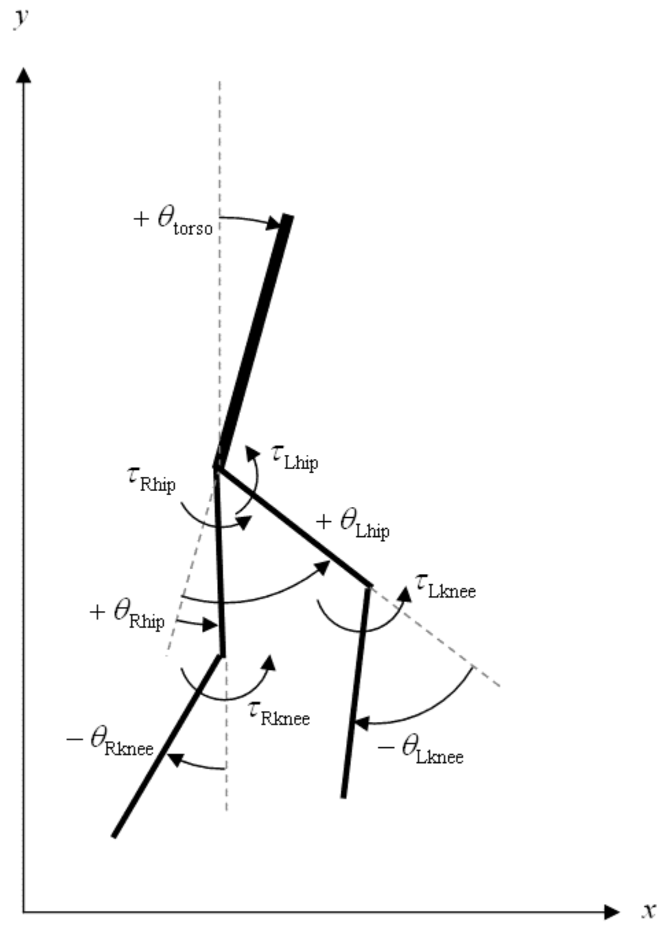


**Figure 6-36 Double support phase leg state measurements available to the Legbot robot. These angles can be substituted into Pratt's equations for determining required joint torques given virtual forces.**

Once again, following Pratt's Jacobian based method, a set of equations relating the virtual forces to required Legbot joint torques can be determined. The resulting equations are listed here, updated to account for differences in the direction of angular measurements used in Pratt's system model, and the Legbot model shown in Figure 6-36.

$$\begin{bmatrix} \tau_{\text{Lknee}} \\ \tau_{\text{Lhip}} \\ \tau_{\text{Rknee}} \\ \tau_{\text{Rhip}} \end{bmatrix} = \begin{bmatrix} -\frac{CV}{E} & -\frac{DV}{E} & \left(\frac{RC-V-QD}{2E}-\frac{1}{2}\right) \\ 0 & 0 & -\frac{1}{2} \\ \frac{AW}{E} & \frac{BW}{E} & \left(\frac{W+SB-TA}{2E}-\frac{1}{2}\right) \\ 0 & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} f_x \\ f_y \\ \tau_\theta \end{bmatrix} \qquad (6)$$

Where:

$$A = -l_{\text{shin}} \cos(\theta_{\text{Lhip}} + \theta_{\text{Lknee}} - \theta_{\text{torso}}) - l_{\text{thigh}} \cos(\theta_{\text{Lhip}} - \theta_{\text{torso}}) \tag{7}$$

$$B = -l_{\text{shin}} \sin(\theta_{\text{Lhip}} + \theta_{\text{Lknee}} - \theta_{\text{torso}}) - l_{\text{thigh}} \sin(\theta_{\text{Lhip}} - \theta_{\text{torso}}) \tag{8}$$

$$C = -l_{\text{shin}} \cos(\theta_{\text{Rhip}} + \theta_{\text{Rknee}} - \theta_{\text{torso}}) - l_{\text{thigh}} \cos(\theta_{\text{Rhip}} - \theta_{\text{torso}}) \tag{9}$$

$$D = -l_{\text{shin}} \sin(\theta_{\text{Rhip}} + \theta_{\text{Rknee}} - \theta_{\text{torso}}) - l_{\text{thigh}} \sin(\theta_{\text{Rhip}} - \theta_{\text{torso}}) \tag{10}$$

$$E = CB - DA \tag{11}$$

$$Q = -l_{\text{thigh}} \cos(\theta_{\text{Lhip}} - \theta_{\text{torso}}) \tag{12}$$

$$R = -l_{\text{thigh}} \sin(\theta_{\text{Lhip}} - \theta_{\text{torso}}) \tag{13}$$

$$S = -l_{\text{thigh}} \cos(\theta_{\text{Rhip}} - \theta_{\text{torso}}) \tag{14}$$

$$T = -l_{\text{thigh}} \sin(\theta_{\text{Rhip}} - \theta_{\text{torso}}) \tag{15}$$

$$V = l_{\text{shin}} l_{\text{thigh}} \sin(\theta_{\text{Lknee}}) \tag{16}$$

$$W = l_{\text{shin}} l_{\text{thigh}} \sin(\theta_{\text{Rknee}}) \tag{17}$$

v. Virtual model control of the swing leg in the single support phase

Control of the swing leg during the single support phase of the walk is managed using a spring/damper virtual linkage component. Pratt provides only superficial details about this component of his VMC control system, so there may be some differences between his implementation and my own. This experiment uses a system of angular spring/damper components attached to each joint of the raised leg. This is another way of saying the raised leg is controlled using a simple PD positional control algorithm.

The joint set-points are manipulated so that the raised foot is moved towards a target "swing" position moving at a fixed height $h_{\mathrm{swing}}$ above the support foot and mirroring the position of the support foot with respect to the robot's torso. Figure 6-37 shows the trajectory of the raised foot's set-point position, as it moves towards the "swing" target during the single support phase of the walk. To maintain the controllability of the raised leg, the velocity of the foot's set-point position is limited to a maximum of $v_{\mathrm{SP}}$ m/s.

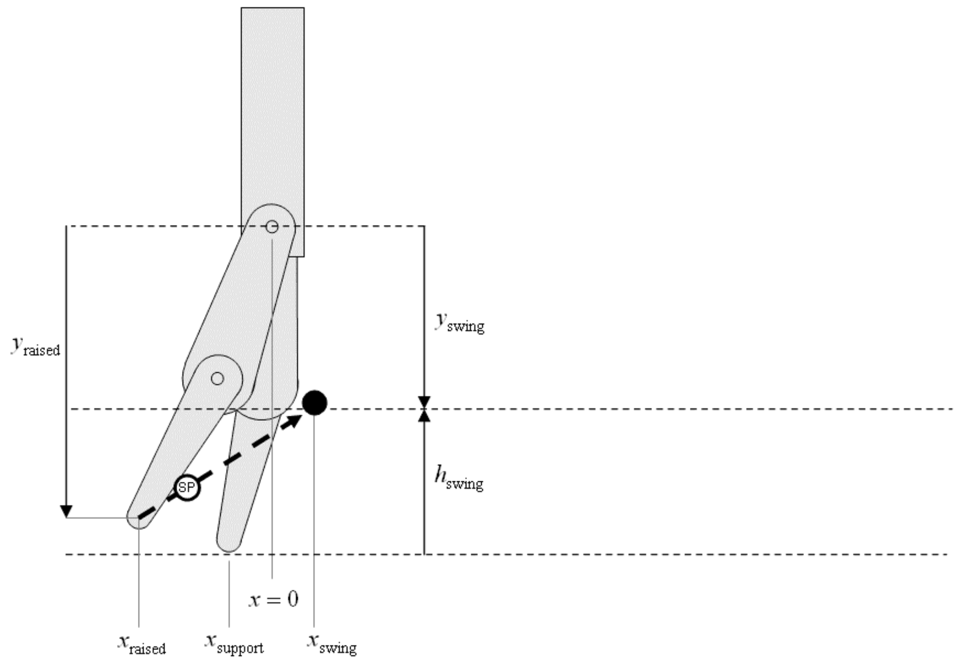


**Figure 6-37 The raised leg's virtual linkage moves the robot's raised foot to a point moving at a fixed height above the support foot, and at a horizontal displacement mirroring the support foot's position.**

All measurements used in implementing the virtual linkage controlling the swing leg are made relative to the robot's torso. Considering that the Legbot's knee and hip joint angle measurements are positive in a counter-clockwise direction, the following relationships can be determined:

Raised foot position, relative to the hip joint:

$$\begin{bmatrix} x_{\mathrm{raised}} \\ y_{\mathrm{raised}} \end{bmatrix} = \begin{bmatrix} l_{\mathrm{thigh}} \sin(\theta_{\mathrm{raisedHip}}) + l_{\mathrm{shin}} \sin(\theta_{\mathrm{raisedHip}} + \theta_{\mathrm{raisedKnee}}) \\ -l_{\mathrm{thigh}} \cos(\theta_{\mathrm{raisedHip}}) - l_{\mathrm{shin}} \cos(\theta_{\mathrm{raisedHip}} + \theta_{\mathrm{raisedKnee}}) \end{bmatrix} \tag{18}$$

Support foot position, relative to the hip joint:

$$\begin{bmatrix} x_{\text{support}} \\ y_{\text{support}} \end{bmatrix} = \begin{bmatrix} l_{\text{thigh}} \sin(\theta_{\text{supportHip}}) + l_{\text{shin}} \sin(\theta_{\text{supportHip}} + \theta_{\text{supportKne e}}) \\ -l_{\text{thigh}} \cos(\theta_{\text{supportHip}}) - l_{\text{shin}} \cos(\theta_{\text{supportHip}} + \theta_{\text{supportKne e}}) \end{bmatrix} \tag{19}$$

Swing virtual linkage target position, relative to the hip joint:

$$\begin{bmatrix} x_{\text{swing}} \\ y_{\text{swing}} \end{bmatrix} = \begin{bmatrix} -x_{\text{support}} \\ y_{\text{support}} + h_{\text{swing}} \end{bmatrix} \tag{20}$$

At the start of the single support phase, the foot's set-point position is set to mirror the current position of the raised foot. Each time the control system updates, the set-point is moved closer to the latest position of the virtual linkage target, at the maximum velocity of $v_{\text{SP}}$. Hip and knee set-point positions ($\theta_{\text{hipSP}}$ and $\theta_{\text{kneeSP}}$) are calculated to place the raised foot at the set-point location, and joint torques are determined using PD control to match the set-point angles:

$$\tau_{\text{hip}} = k_{\text{swing}} \cdot (\theta_{\text{hipSP}} - \theta_{\text{hip}}) + b_{\text{swing}} \frac{\Delta(\theta_{\text{hipSP}} - \theta_{\text{hip}})}{T} \tag{21}$$

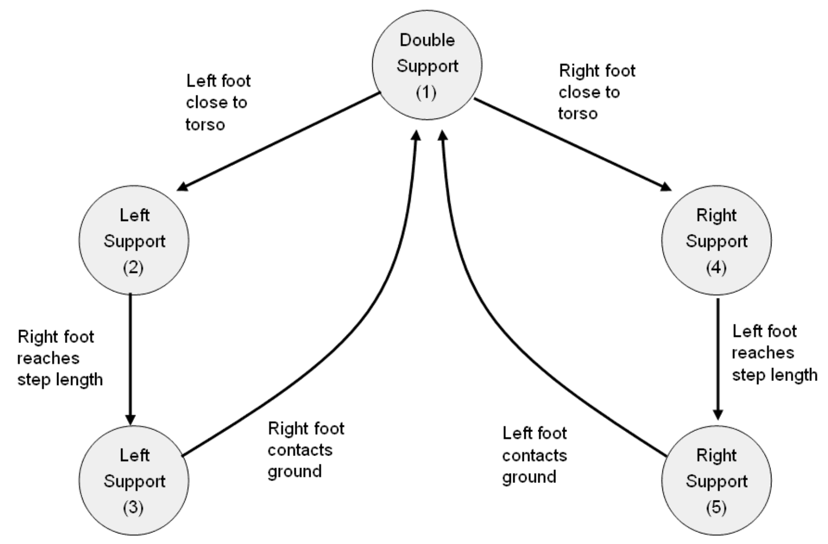
$$\tau_{\text{knee}} = k_{\text{swing}} \cdot (\theta_{\text{kneeSP}} - \theta_{\text{knee}}) + b_{\text{swing}} \frac{\Delta(\theta_{\text{kneeSP}} - \theta_{\text{knee}})}{T} \tag{22}$$

### vi. Transitioning between states

In order to coordinate the transition between double and single support stages of the walking gait, Pratt's VMC algorithm uses a state machine to activate and deactivate the various virtual components. The state machine uses the following triggers for transitioning between states:

- Transition from double to single support phase when the torso moves within a nominal distance $d_{\text{triggerup}}$ of the leading foot.
- Transition from single to double when the raised leg reaches a nominal step length $d_{\text{step}}$ ahead of (or behind) the support leg.

The transition from single to double support is achieved using an intermediate state, during which the raised leg is left unactuated to drop to the ground. In order to prevent issues with the foot bouncing on the ground, this experiment's implementation of the control system requires ground contact to be maintained for a small time ($t_{\text{touchdown}}$) before switching to the double support phase.



Double Support (1)
Left foot close to torso
Right foot close to torso
Left Support (2)
Right Support (4)
Right foot reaches step length
Left foot reaches step length
Left Support (3)
Right foot contacts ground
Left foot contacts ground
Right Support (5)

Image source: Pratt, et. al., "Virtual Model Control: An intuitive approach for bipedal locomotion", [85].
**Figure 6-38 State machine used for Virtual Model Control walking.**

Figure 6-41 provides a graphical representation of the state machine, and has been taken directly from [85]. A summary of the states, trigger events and associated virtual components is provided in Table 6-11.

**Table 6-11 State machine trigger events and virtual components.**

| State | Trigger event | Virtual components |
|---|---|---|
| Double support (1) | Raised foot contacts ground (for $t_{\text{touchdown}}$ seconds) | Granny walker<br>Dog-track bunny |
| Left support (2) | Torso nearly over left foot | Granny walker<br>Swing leg linkage |
| Left support (3) | Right foot at step length from left foot | Granny walker |
| Right support (4) | Torso nearly over right foot | Granny walker<br>Swing leg linkage |
| Right support (5) | Left foot at step length from left foot | Granny walker |

vii. Tuning parameters

The VMC control system for biped walking requires a relatively small number of tuning parameters. Typically the tuning parameter values are determined through trial and error in an experimental process. Pratt published the parameters he used to control a 12 kg robot with a leg length of 84 cm in [85]. The VMC tuning parameters are summarised in Table 6-12.

**Table 6-12 Tuning parameters used by the VMC control system.**

| Parameter | Pratt's value | Legbot value |
|---|---|---|
| Vertical stiffness ($k_{\text{y}}$) | 500 N/m | 650 N/m |
| Vertical damping coefficient ($b_y$) | 200 Ns/m | 3 Ns/m |
| Hip height spring set point ($h_{\text{target}}$) | 0.54 m | 0.27 m |
| Rotational spring stiffness ($k_\theta$) | 50 Nm/rad | 45 Nm/rad |
| Rotational damping | 20 Nms/rad | 0.3 Nms/rad |

| | | |
|---|---|---|
| coefficient ($b_\theta$) | | |
| Update interval ($T$) | unspecified | 2 ms |
| Step length ($d_{\text{step}}$) | 0.28 m | 0.07 m |
| Foot distance from torso causing transition to single support state ($d_{\text{triggerup}}$) | 0.03 m | 0.011 m |
| Step height ($h_{\text{swing}}$) | 0.07 m | 0.025 m |
| Target horizontal velocity ($\dot{x}_{\text{swing}}$) | 0.4 m/s | Can only balance walking at 0.2 m/s |
| Horizontal damping coefficient ($b_x$) | 200 Ns/m | 32.3 Ns/m |
| Swing linkage spring stiffness ($k_{\text{swing}}$) | unspecified | 250 N/m |
| Swing linkage damping coefficient ($b_{\text{swing}}$) | unspecified | 0.4 Ns/m |
| Swing set-point maximum velocity ($v_{\text{SP}}$) | unspecified | 0.7 m/s |
| Length of time the raised foot must touch the ground before state shifts to double support ($t_{\text{touchdown}}$) | unspecified | 30 ms |
| Maximum joint torque ($\tau_{\text{max}}$) | unspecified | 1.4 Nm |

#### 6.8.3.2 Conducting experimental comparisons

There are four major areas of control system performance that were evaluated for comparison with the Torso Driven Walking control system. These are:

- Target tracking ability
- Smoothness of torso movement
- Joint torque requirements
- Robustness to sensor noise

Experimental trials will be conducted on each control system to generate data evaluating the control systems performance in these areas.

i. Target tracking ability (positional control)

Apart from the obvious criteria of maintaining balance, the next most important attribute of a control system is its ability to manage the position of the robot. In the case of the Legbot robot, this means being able to specify the speed at which the robot should travel, and the position to which it needs to move.

In order to evaluate a control system's ability to control the position and velocity of the robot, the Torso Driven Walking objective function described in 6.6.3.2 is reused as a benchmark for comparing target tracking performance. To summarise, the objective function samples the error between the displacement of the Legbot's torso and a moving set-point target position. Error measurements are taken every second during a 60 second trial, with the performance of the control system accumulating a score inversely proportional to the distance of the robot from the target. During the trial the target position includes periods where it is stationary, as well as intervals during which it is moving forwards and backwards at varying speeds. If the control system being evaluated is able to maintain balance during the test, the resulting “fitness” score can be used to compare its target tracking performance with the TDW algorithm operating in the same environment with the same test.

In situations where the control system under investigation is not able to operate with a varying velocity, data will be collected for comparison with the Torso Driven walking control system operating at an equivalent velocity. This data will be needed when evaluating other aspects of the control system performance.

ii. Smoothness of torso movement

Apart from aesthetics, the ability to maintain a smooth torso motion during execution of a robot’s gait would be a useful contribution to reducing the magnitude of sensor noise experienced by the control system.

By comparing recordings of the robot’s system state variables as each control system operates at equivalent velocities, the performance of the control systems in terms of smoothness of torso movement can be compared. Ideally, variation in torso inclination will be small during execution of the robot’s walk.

iii. Torque profile

The level of joint torque a control system demands during execution of a walking gait has a large impact on the physical design of a robot. Control systems that are able to minimise the levels of torque they demand would require less powerful actuators, and operate more efficiently.

To investigate the torque requirements of each system, the control system’s output torque for all four actuated joints during the course of the experimental trials is recorded. Intervals where the robots are operating at equivalent velocities will be analysed for the characteristics of torque output from each control system. Measurements of interest are the maximum torque required, the average level of torque each joint requires and the "noisiness" of the joint's torque profile during a representative sample of a gait.

iv. Sensor noise robustness

How robustly a control system handles sensor noise will be an important factor in both the choice of control system, and sensors to install on a physical robot. Both the TDW and VMC control systems require the same set of system state measurements as input to their control algorithms. These state variables include an estimate of torso inclination, as well as joint angle estimates for each of the robot's four joints.

When investigation the impact of sensor noise on the control systems, the kinds of errors experienced by the sensors installed on the physical Legbot robot when measuring these two different types of angles are modelled:

- To simulate errors in torso inclination, a low-frequency error, or bias, is applied to the angle estimate utilised by the control systems. This means that the error in angle measurement steadily increases during the course of an experimental trial. From an examination of the gyroscope intended for the Legbot (see Chapter 5.7) an angle estimate bias of 2.0 degrees/second is a realistic level of gyroscope drift.
- To simulate errors in joint angle estimate, the angle measurement is quantised to a precision fixed for the duration of a trial. This simulates the actual quantisation occurring through the use of optical encoders, and also is a reasonable representation of the errors due to dead-bands in the joint angle measurements.

To compare the behaviour of each control system in the presence of sensor noise, the target tracking experiment is repeated in the presence of each type of simulated sensor noise. The level of noise will be gradually increased until a level is found at which each control system fails to maintain balance.

### 6.8.4 Results

#### 6.8.4.1 Target tracking (positional control)

Figure 6-39 shows the changing state variables of a Legbot robot controlled by the Torso Driven Walking control system, as the controller follows the objective function used to evaluate the accuracy with which the control system is able to follow a moving target, while maintaining balance.

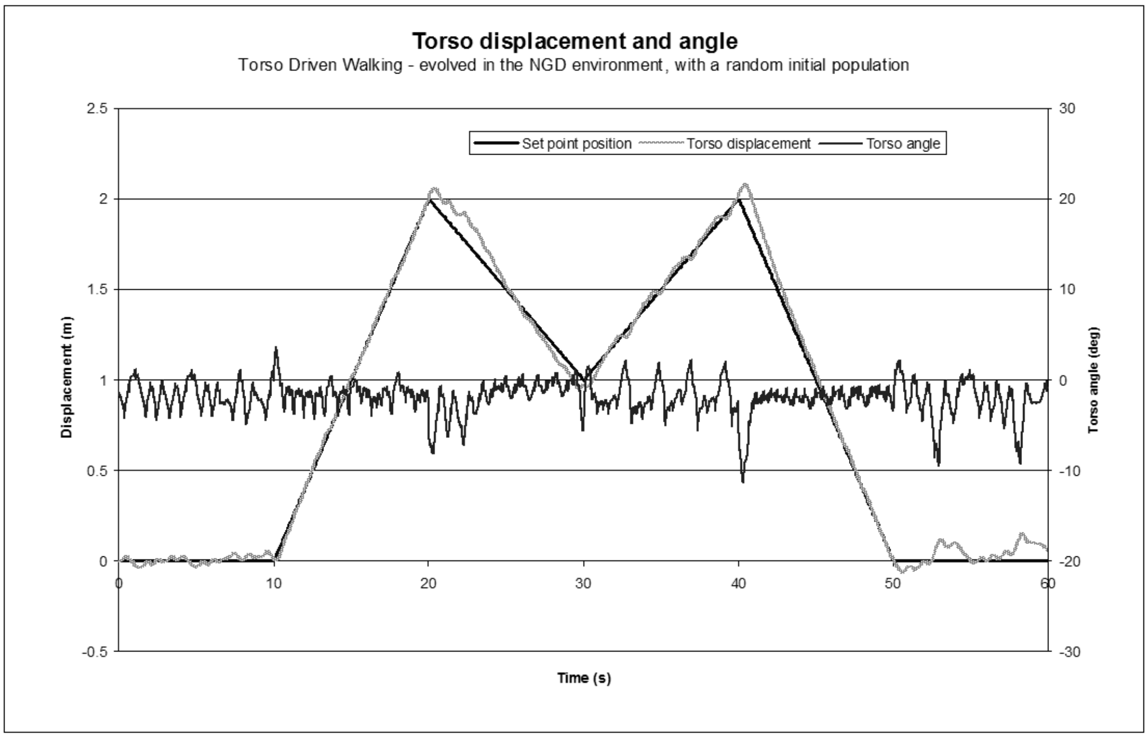


**Figure 6-39 Torso Driven Walking control system successfully follows a set-point target moving with varying velocity, both forwards and backwards.**

As the chart shows, the Torso Driven Walking control system is able to follow a set-point position moving with variable velocity over the course of a 60 second trial. The resulting robot movement closely matches the target velocity during the portions of the test where the target's velocity is constant, and adapts to instantaneous changes in the target velocity with some transient overshoot of the target position.

Unfortunately, the Virtual Model Control algorithm was not able to complete the same objective function. When starting the VMC controlled robot in the single support phase, the robot was unable to maintain a stationary (or close to stationary) position near the initial position of the target without falling over. Figure 6-40 shows the result observed when attempting to evaluate the VMC system's ability to track the objective function's set-point trajectory.

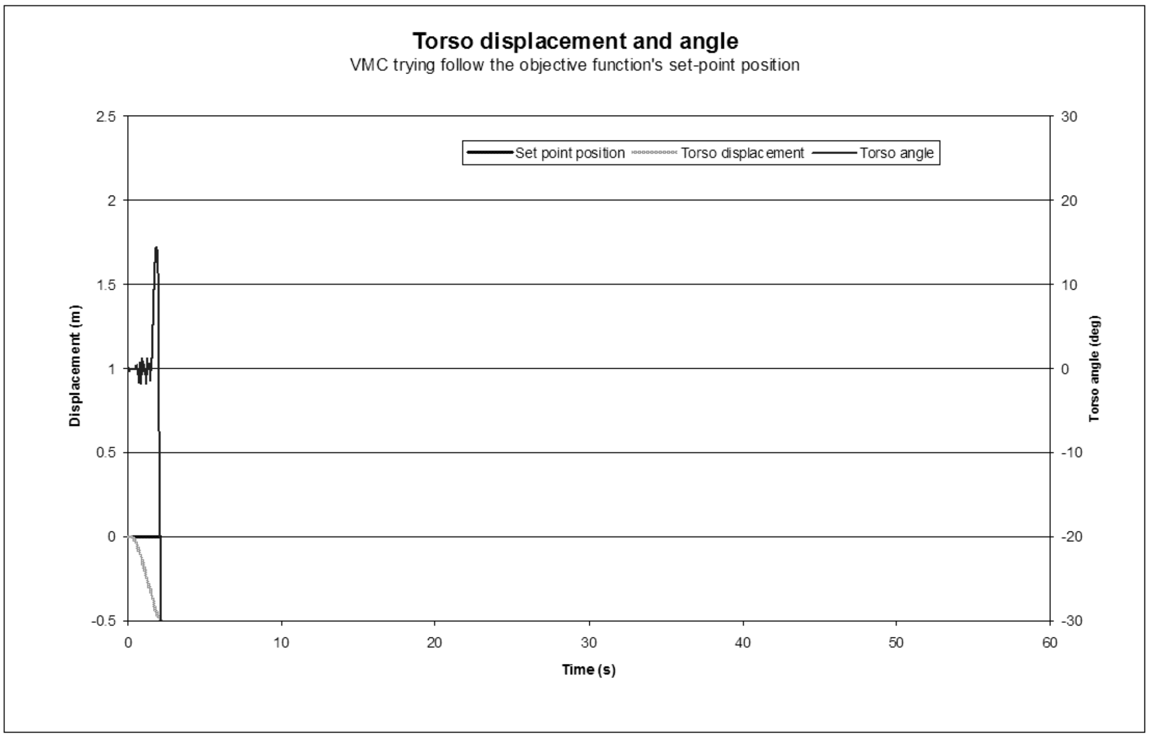


**Figure 6-40 The Virtual Model Controller is unable to follow the target set-point position, as it fails to hold the robot close to a stationary position.**

This inability to start from a stationary in initial position is consistent with Pratt's description of his control system, where his experiments are initiated by starting the robot with a non-zero horizontal velocity. Pratt describes starting the walking action by giving the robot a push [85]. While it is not easily possible to initialise the simulator with a non-zero velocity, a stable walking pattern can be induced using the VMC controller if the simulation is started with the robot leaning very slightly forward (1 degree). This initial inclination, together with a target trajectory that constantly moves with a steady velocity of 0.2 m/s resulted in a walking trajectory shown in Figure 6-41.

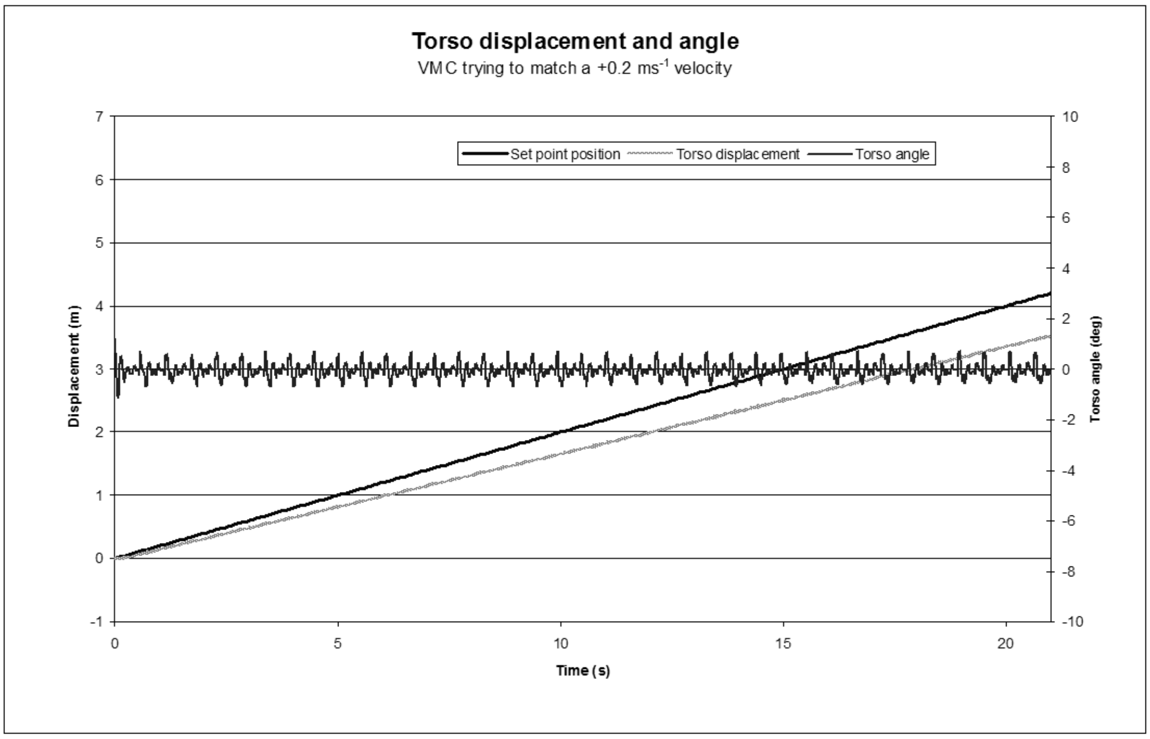


**Figure 6-41 The VMC system is able to control the simulated Legbot to walk, but only at a constant velocity. In addition, the control system needed a kick-start to get moving at the start of the trial. This was accomplished by leaning the robot in the forward direction before starting the simulator.**

Using the VMC control system, it was not possible to significantly vary the robot's velocity. Instead, the control system seems quite good at controlling the robot to move with a fixed velocity of approximately 0.17 m/s. The control system cannot stand still, move backwards, change directions, or even vary its speed significantly. Some small variation in speed could be seen by varying the tuning parameter $b_x$, the horizontal damping coefficient of the dog-track bunny. However, changing this factor only resulted in a variation of ~5% in the resulting horizontal velocity. The magnitude of velocity the robot follows is consistent with the velocity Pratt published in his results, after "scaling down" Pratt's robot velocity by the relative difference in size between the two robots.

In contrast, the TDW control system is able to modulate its velocity, following a set-point position moving with a velocity varying between -0.2 m/s and +0.2 m/s, with a maximum instantaneous change in velocity of 0.3 m/s, as was shown by Figure 6-39. In order to provide a comparison with the limited scope of movement achieved by the VMC implementation, a recording of the Torso Driven Walking controller marching forward at a constant velocity of 0.2 m/s is shown by Figure 6-42.

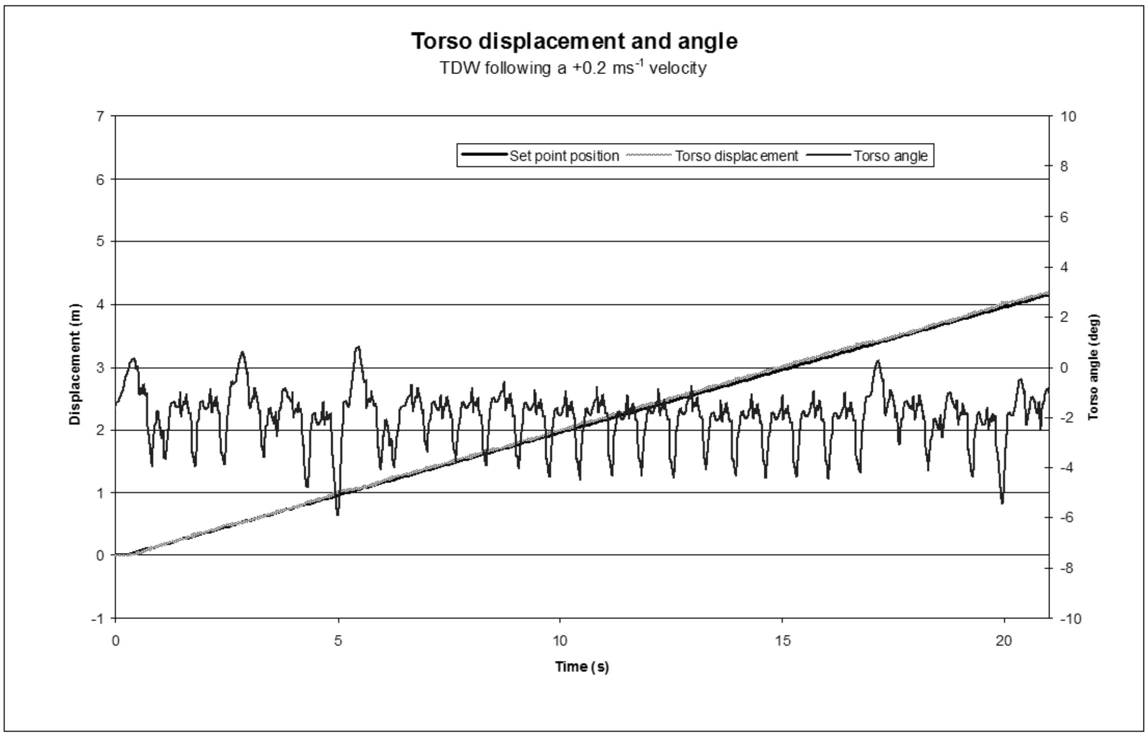


**Figure 6-42 The Torso Driven Walking control system driving the Legbot at a velocity of 0.2 m/s. The TDW control system does not require the robot to be “kick-started”, with the acceleration at the start of the trial being induced by a fairly large torso swing.**

### 6.8.4.2 Smoothness of torso movement

Figure 6-41 and Figure 6-42 each include a recording of the torso inclination values that result when the VMC and TDW control systems drive the robot at a constant velocity of 0.2 m/s.

It is clear from a glance at the two charts that the TDW system experiences a significantly larger variation in torso inclination than the VMC system. This is to be expected as the TDW system uses the imbalance caused by a change in torso angle to alter its horizontal acceleration, keeping it tracking the target. The VMC control system would be expected to result in a smoother ride, since regulating the torso inclination is one of the primary objectives of the control system, using the "Granny Walker" virtual component.

Statistics comparing the torso inclination signals during this experiment have been summarised in Table 6-13.

**Table 6-13 A comparison of the torso inclination experienced while walking at 0.2 m/s under the control of a Virtual Model Control and a Torso Driven Walking control system.**

| **Characteristic** | **Virtual Model Control** | **Torso Driven Walking** |
|---|---|---|
| Average inclination | -0.06 degrees | -2.00 degrees |
| Standard deviation | 0.28 | 1.01 |
| Max inclination | 1.20 degrees | 0.80 degrees |
| Min inclination | -1.14 degrees | -5.92 degrees |

#### 6.8.4.3 Torque profiles

Figure 6-43 and Figure 6-44 present trends of the joint torque acting on each of the robot's right hand hip and knee joints during a five second window. These measurements were recorded during time periods where both the Torso Driven Walking and Virtual Model Control systems are attempting to drive the Legbot robot at a velocity of 0.2 m/s.

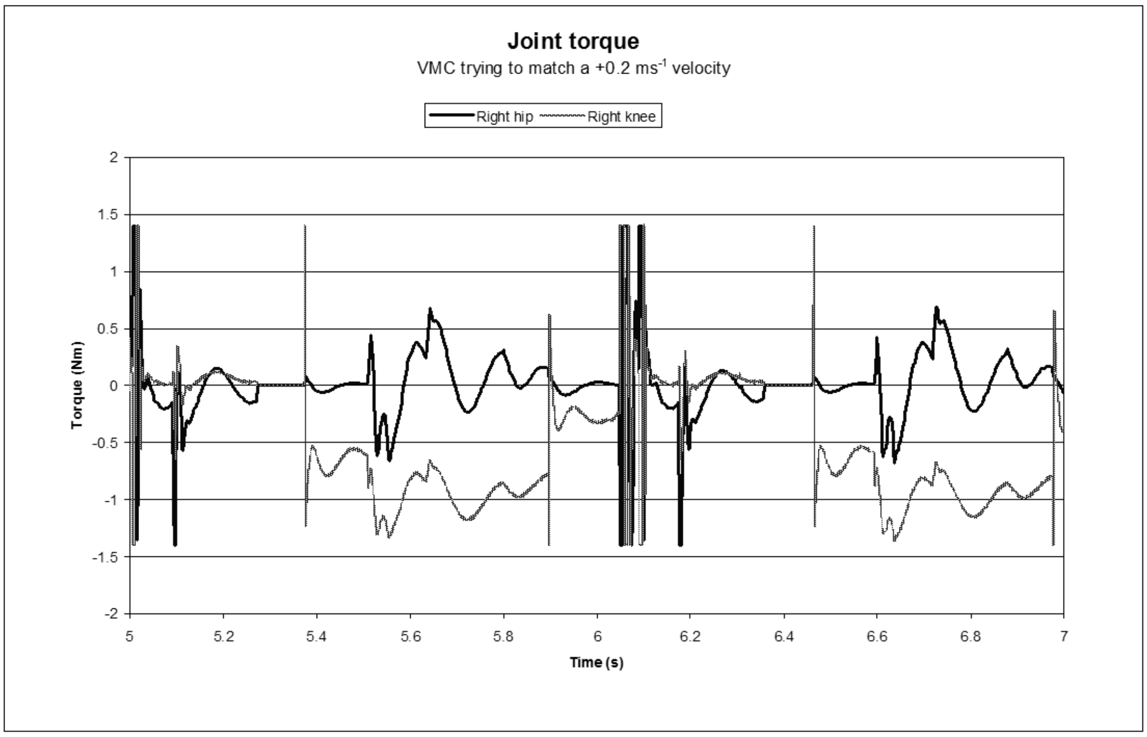


**Figure 6-43 Virtual Mode Control torque output as the control system tries to drive the robot to move at 0.2 m/s. (In fact the robot only manages ~0.17 m/s during this trial).**

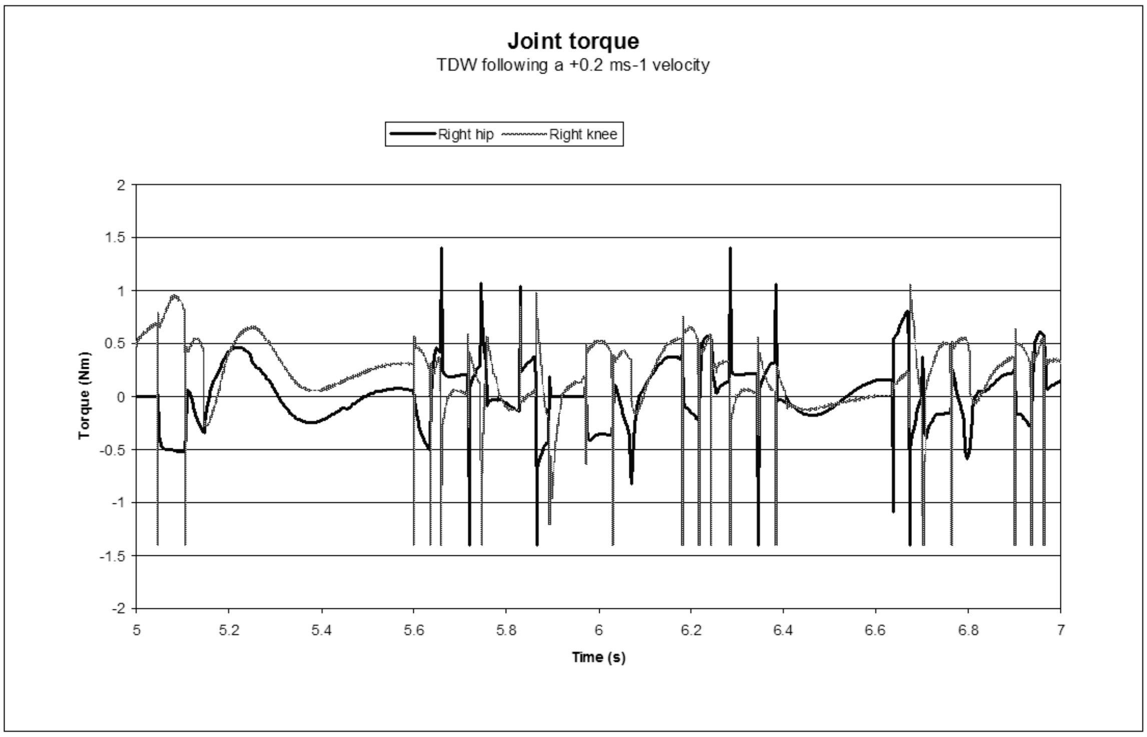


**Figure 6-44 Torso Driven Walking torque output as the control system drives the Legbot at a velocity of 0.2 m/s.**

While the torque profile of both control systems includes instances of "spikes" in the output torque, the spikes are distributed more widely through the TDW profile. On the

VMC chart, most of the extreme torque values occur within a short time interval immediately following the landing of the leg's foot.

An examination of the two charts reveals that the required knee torque is significantly larger in the Virtual Model Control experiment for about 50% of the trial duration. The high torque requirements coincide with the times the knee is supporting the robot's weight, and is a result of the "bent kneed" walking style necessitated by the control system design.

In order to more easily compare the torque requirements of each control system, the following two charts each combine torque recordings from both control systems for a single joint. Figure 6-45 compares the torque applied to the right hip joints during a five second window where both robots are trying to match a velocity of 0.2 m/s. The trend lines are calculated using a moving average with a window size of five samples. Figure 6-46 presents similar data for the right knee joint of the robot during the same test.

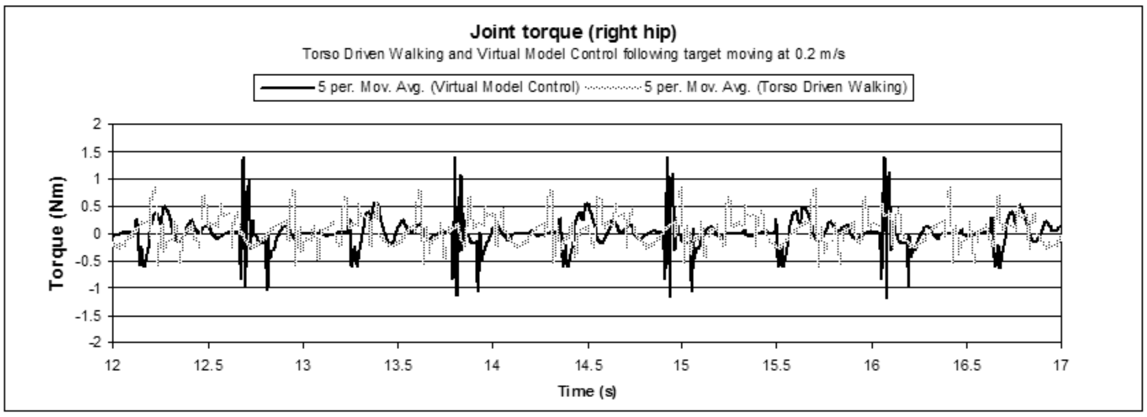


**Figure 6-45 Comparison of hip torque between TDW and VMC control systems, while driving the robot at approximately 0.2 m/s.**

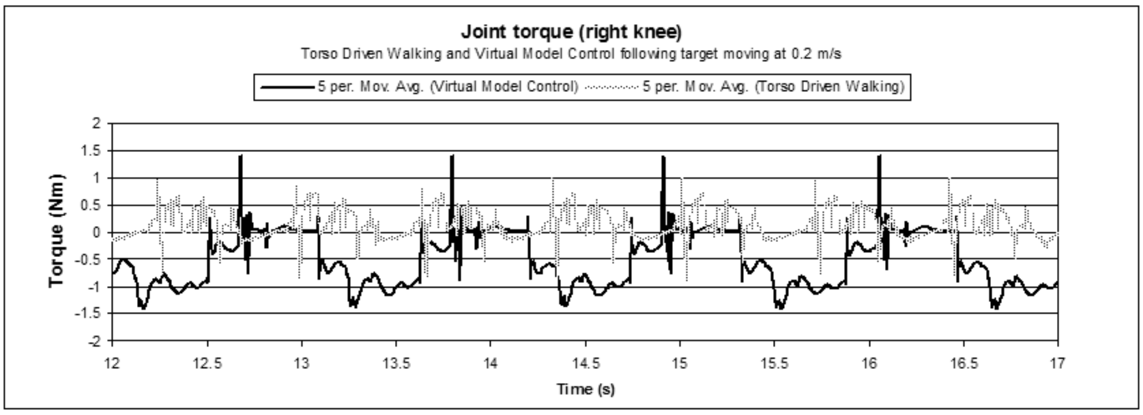


**Figure 6-46 Comparison of knee torque between TDW and VMC control systems, while driving the robot at approximately 0.2 m/s.**

For both joints, but especially the knee joint, the graphs clearly demonstrate that the Torso Driven Walking control system requires less average torque as well as a lower maximum torque to operate the robot than the Virtual Model Control system.

#### 6.8.4.4 Simulated sensor noise

Since the VMC control system is not able to control the Legbot with a variable speed, the robustness of the two control systems in the presence of sensor noise is investigated while they drive the robot forward at 0.2 m/s.

i. Joint angle quantisation

For each control system, the magnitude of joint angle quantisation is gradually increased until the robot is unable to maintain balance for 20 seconds. The performance of the Torso Driven Walking system appears to be more robust than the Virtual Model Control, with the TDW system failing with a joint quantisation over ~2.5 degrees, and VMC failing at a quantisation over ~0.1 degrees.

ii. Torso angle bias/drift

For each control system, the rate at which torso angle measurement bias accumulates is gradually increased until the robot is unable to maintain balance for 20 seconds. The

VMC control system proves particularly vulnerable to a torso bias error, and is unable to complete a 20 second trial with the bias set to a minimum level of 0.1 degrees/second. When the accumulated inclination error reaches about 1.8 degrees (after 18 seconds), the robot falls.

Torso Driven Walking on the other hand easily maintains balance for the duration of the trial, even in the presence of a large bias signal increasing at a rate between -2.0 and 7.0 degrees/second. Instead of falling, the bias error manifests in the control system's output as an error in the horizontal displacement of the robot from its target position. In the presence of a constantly increasing bias, the robot's response is to move away from the target position at a constant velocity. Because in this test the robot is moving with a constant velocity of +0.2 m/s, the effect of an increasing magnitude of sensor bias is asymmetrical. When the bias drift is positive, the corresponding compensating velocity is negative. In this experiment, the robot slows down to compensate, which is why the system can handle a greater magnitude of positive signal drift.

### iii. Combined torso angle bias/drift and joint quantisation

Since the VMC controller was unable to operate in the presence of a torso angle bias error, only the Torso Driven Walking control system has been investigated here. Figure 6-47 shows the TDW control system completing the full 60 second variable velocity objective function, while experienced the combined effects of:

- Torso angle measurement bias increasing at a rate of 1.0 degrees/second.
- Joint angle quantisation at a precision of 0.5 degrees.

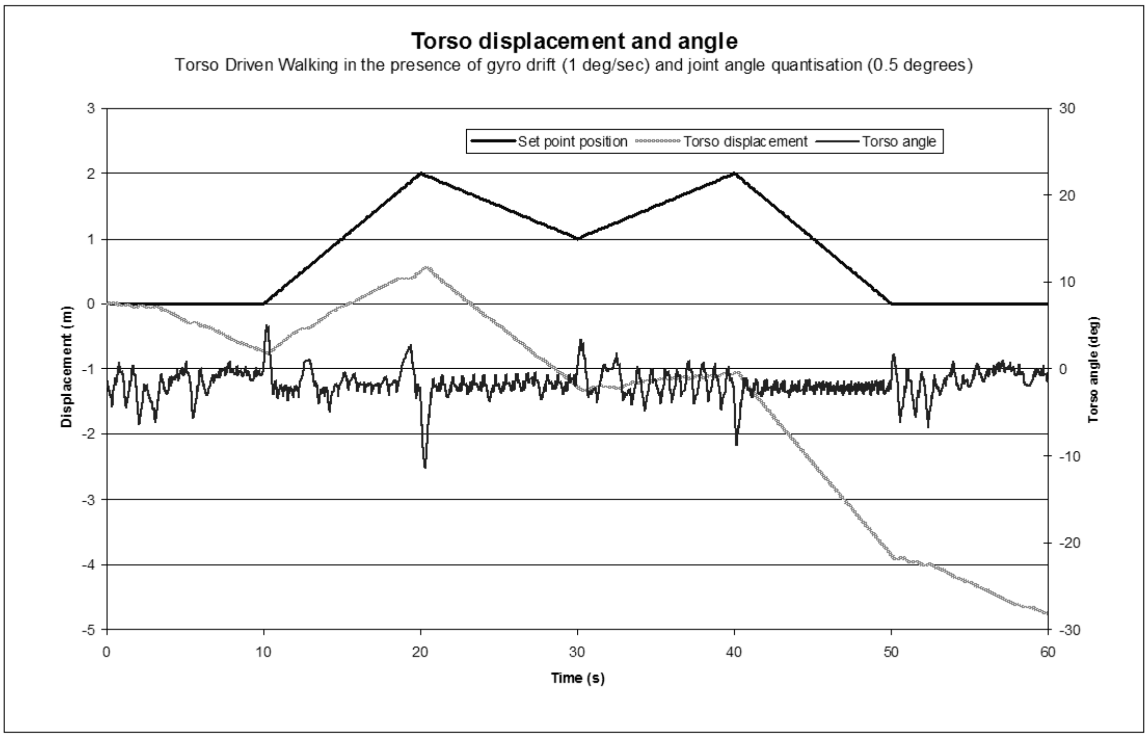


**Figure 6-47 The Torso Driven Walking control system is able to maintain balance of the Legbot in the presence of both gyroscope drift (or "bias"), as well as joint quantisation. The system automatically compensates for the accumulating error in torso angle estimate by moving away from the target set-point position. The torso angle data displayed in this graph is the actual torso angle , and does not include the effects of sensor bias.**

### 6.8.5 Discussion

In this experiment, Pratt's Virtual Model Control system for biped walking was implemented in order to compare it to the Torso Driven Walking control system. The VMC implementation achieves a walking performance that is consistent with Pratt's published results in [66], [80] and [85]. If given sufficient impetus at the start of a trial, the robot is able to maintain balance while smoothly walking forwards at a constant velocity. Using the VMC control system, the simulated robot walked at a constant velocity of approximately 0.17 m/s. Pratt's experimental robot walked at 0.4 m/s, however taking into account the difference in leg length between the two robot platforms, the velocity achieved by each VMC system is similar.

It is important to note that despite the similarity in performance between the VMC implementation investigated here and Pratt's published results, it is not possible to prove that the performance of my VMC control systems is identical to Pratt's. The VMC system requires a significant amount of system tuning, and it is possible that the final parameters used here are not an optimal set. This should be kept in mind when considering the differences found in the performance of the Torso Driven Walking and Virtual Model Control systems.

In comparing the Virtual Model Control and Torso Driven Walking control systems, the most obvious difference is that a VMC control system that allows a robot to move with a variable velocity or in the reverse direction could not be found. This is a serious deficiency in a control system whose ultimate purpose is to move a vehicle from one place to another. In the VMC implementation, the "dog-track" bunny, which is supposed to control the robot's horizontal velocity, had only a minimal influence on the robot's speed. With a target velocity fixed at 0.2 m/s, adjusting the horizontal damping of the "dog-track bunny" virtual component influenced the resulting velocity of the robot by a factor of approximately 5%. This virtual component appears to be more suited to regulating the robot's movement to a constant velocity rather than allowing the robot to move with variable velocity.

It is possible that further tuning of the VMC control system might improve the controllability of the robot's velocity, but more substantial modification to the algorithm would be required. Increasing the number of states in the VMC State Transition diagram would allow the robot to move with different characteristics depending on whether it is walking forwards or backwards. Similarly, increasing the number of tuning parameters may also allow the robot gait to be less constrained, and may increase the range of velocities with which the robot can walk without losing balance. This is the approach taken by the Torso Driven Walking control system in designing the path taken by the legs during execution of the waking gait. Of course the disadvantage is increased complexity in terms of tuning or commissioning the robot.

Another serious problem with VMC control systems became evident during attempts to tune a system capable of a variable velocity. It is not possible to guarantee that the robot being controlled will be able to satisfy the "virtual force" commands the control system generates using its virtual components. This problem has already been briefly discussed in a general way by considering application of VMC to the inverted pendulum problem in Chapter 2.3.4. The problem also exists in Pratt's VMC biped walking algorithm, even though Pratt limits the application of the horizontal virtual force to the double support phase of the walk. The following example (illustrated by Figure 6-48) highlights one situation experienced while tuning the VMC system.

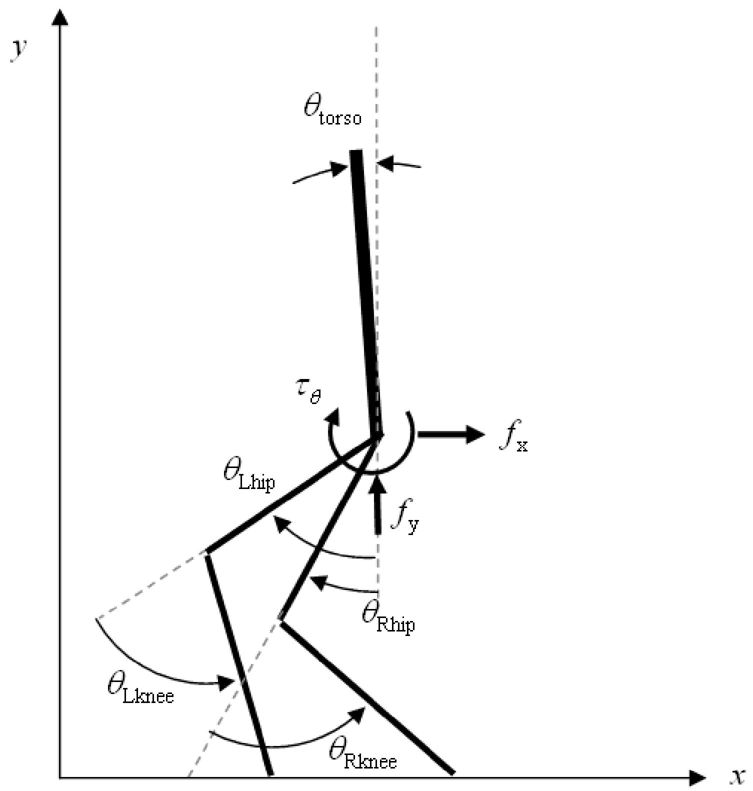


**Figure 6-48 A fairly typical system model configuration for a VMC controlled biped moving in the $+x$ direction. The virtual forces $f_x$, $f_y$ and $\tau_\theta$ are all determined by virtual components based on the robot torso's height, inclination and horizontal displacement.**

In the example, the robot is standing in the double support phase, and is being lifted up by a vertical virtual force ($f_y$) generated by the "granny walker" component, and pulled ahead by a horizontal virtual force ($f_x$) generated by the "dog-track bunny". A virtual torque ($\tau_\theta$) works to keep the torso upright.

**Table 6-14 Known system variable values for the example demonstrating a failure mode of Virtual Model Control.**

| **State variable** | **Value** |
|---|---|
| Torso inclination ($\theta_{\text{torso}}$) | 0.00 degrees |
| Right hip angle ($\theta_{\text{Rhip}}$) | -20.17 degrees |
| Right knee angle ($\theta_{\text{Rknee}}$) | 68.64 degrees |
| Left hip angle ($\theta_{\text{Lhip}}$) | -48.64 degrees |
| Left knee angle ($\theta_{\text{Lknee}}$) | 61.88 degrees |

The virtual forces {$f_y$, $f_x$, $\tau_\theta$} are also known, however their value depends on the tuning parameters that have been chosen for the VMC implementation. Pratt's algorithm attempts to find a set of joint torques {$\tau_{\text{Rhip}}$, $\tau_{\text{Rknee}}$, $\tau_{\text{Lhip}}$, $\tau_{\text{Lknee}}$} that will result in the specified virtual forces being applied to the robot's hip. In order to simplify the VMC equations, both hip and shin lengths are set to 1.0 m. Substituting the angles from Table 6-16 into the VMC equations (described in 6.8.3.1), generates the following equations for torque applied to the right leg:

$$\tau_{\text{Rhip}} = -\frac{\tau_\theta}{2} \tag{23}$$

In the example, the torso angle is zero, so both $\tau_\theta$ and $\tau_{\text{Rhip}}$ will also be zero. The equation to determine $\tau_{\text{Rknee}}$ is:

$$\tau_{\mathrm{Rknee}} = 1.02 f_x - 0.322 f_y \tag{24}$$

As equation (24) shows, if $f_x$ becomes larger, the anti-clockwise torque $\tau_{\mathrm{Rknee}}$ also increases. If $f_x > 0.32 f_y$, then $\tau_{\mathrm{Rknee}}$ is positive. In this example, a positive $\tau_{\mathrm{Rknee}}$ means the robot's right foot lifts off the ground, and the robot fails to generate the virtual forces required. In addition, the robot is no longer in a double-support phase, even though the control system still thinks it is. As a result, the robot falls over.

The ability to handle sensor noise is another area where Torso Driven Walking appears to outperform the VMC control system. The TDW control system was able to cope with a much higher level of joint angle quantisation (2.5 degrees compared with 0.1 degrees). The VMC performance against this criterion could probably be improved with further tuning of the control system, so these numbers may not be significant.

More importantly, the Torso Driven Walking robustly handles inclination bias, while the VMC system is unable to handle even a moderate amount of inclination measurement bias. This is an important finding, as gyroscope bias is normally a very difficult error to compensate for. Experiments with inclination sensors that do not experience a bias error (such as inclinometers and accelerometers) suggests these alternatives to gyroscopes may be too inaccurate, slow or susceptible to noise to balance fast moving robots (see Chapter 5.7). Pratt's robots did not experience problems measuring torso inclination because the robot was connected to a "boom", to which absolute angle sensors (such as optical encoders or potentiometers) could be attached. Any unconstrained robot will need to determine its inclination using on-board sensors.

The Virtual Model Control algorithm fails in the presence of a steady state inclination error because almost all the control system's calculations include torso inclination as a fundamental input. The Torso Driven Walking control system utilises torso inclination in only one of its components – the "Torso Attitude Controller", detailed in 3.2.5. This component is based on the simple control rule for an inverted pendulum:

$$f_x = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x} \qquad (25)$$

As long as the rate at which a torso inclination estimation error accumulates is not too fast, the error in angle measurement is automatically compensated for by an error in position. As is graphically demonstrated by Figure 6-47, the actual torso angle remains constant, even though the error in angle estimation is increasing.

The robust manner in which the Torso Driven Walking control system handles torso inclination estimate bias suggests a method by which the TDW control system could be self calibrating. From Equation (25), whenever the robot is close to stationary ($\dot{\theta} \approx 0$ and $\dot{x} \approx 0$), the error in angle estimate will be related to the known error in displacement, such that:

$$\theta_{\text{error}} \approx -\frac{c\,x_{\text{error}}}{a} \qquad (26)$$

Using equation (26), a robot implementing Torso Driven Walking would be able to continuously recalibrate its inclination sensor on-line. This self calibration has not been attempted in these experiments.

The maximum joint torque required by each control system is another area where the TDW control system appears to have an advantage over Virtual Model Control. This is primarily a consequence of the "bent-kneed" walking style the VMC controller uses to maximise time spent in the double support phase of its walk. Increased torque requirements mean stronger actuators are needed to control the robot, and the robot's power consumption will be higher.

To summarise, a number of key performance indicators where the Torso Driven Walking control system out-scores my implementation of Virtual Model Control have been identified. Some of the disadvantages of VMC when compared to Torso Driven Walking:

- Knees alone provide horizontal velocity regulation, but can only do this during the double support phase of the robot's walk.
- The robot's design requirement to move maintaining a constant torso height, together with the need to maximise the time spent in the double support phase while walking, results in a "bent knee" walking style, and consequently high torque requirements.
- The algorithm is susceptible to angle measurement "drift" errors, as there is no positional component to the control of the torso's inclination.
- Requires a fairly accurate system model, since joint torque is calculated using global (Jacobian based) matrix operations.
- There is no guarantee that the robot will be able to satisfy the requirements output by the system's virtual components, leading to unpredictable behaviour and gait state transitions.

Advantages of VMC compared to Torso Driven Walking:

- The Virtual Model Control system requires fewer tuning parameters to be determined, resulting in faster commissioning on new platforms.
- The VMC system results in a smoother ride for the robot's torso. This should result in lower noise experienced by sensors mounted on the torso, and potentially a reduced probability of physical damage to robot components.

Table 6-15 summarises some of the key areas of difference in the performance of the two control systems:

**Table 6-15 A summary of key performance indicators comparing the Torso Driven Walking control system to my implementation of Virtual Model Control.**

| **Indicator** | **Virtual Model Control** | **Torso Driven Walking** |
|---|---|---|
| Joint torque requirement | High<br>*Average total joint torque requirement while walking forward was 1.5 Nm* | Moderate<br>*Average total joint torque requirement while walking forward was 0.96 Nm* |
| Tuning parameter complexity | Moderate<br>*Increasing complexity might improve performance.* | High<br>*Mitigated by using GA based parameter tuning* |
| Velocity control | Poor<br>*Fixed at ~0.17 m/s.* | High<br>*Variable between +/- 0.2 m/s* |
| Noise tolerance – joint angle quantisation | Poor<br>*Handled ~0.1 degrees of quantisation.* | Moderate<br>*Handled ~2.5 degrees of quantisation.* |
| Noise tolerance – torso inclination bias/drift | Poor<br>*Fell once error reached ~1.8 degrees.* | High<br>*Maintains balance when rate of inclination drift is between -2 and +7 degrees/second.* |
| Noise tolerance – combined joint quantisation and inclination bias. | Not tested | Moderate.<br>*Completed test pattern with 0.5 degrees of joint quantisation and 1 degree/second inclination drift.* |
| Torso smoothness | High<br>*Torso deviated by 2.34 degrees while walking forwards.* | Moderate<br>*Torso deviated by 7.92 degrees while walking forward.* |

## 6.9 Physical legbot

**Abstract:** In this experiment, the physical Legbot robot is evaluated to determine the accuracy of joint angle measurements, and the relationship between applied actuator voltage and joint torque. These characteristics can then be compared with the requirements predicted by the simulation systems for a robot sharing the basic configuration of the physical Legbot robot. Unfortunately, the experiments described in this chapter demonstrate that while the joint positions can estimated with a reasonable degree of accuracy, the joint torque that can be delivered by the actuators is low and difficult to control. These attributes will make implementing a successful control system on the existing robot very difficult, if not impossible.

### 6.9.1 Introduction

In order to prove the viability of the Torso Driven Walking control system, it would be useful to demonstrate the system successfully controlling a physical robotic system. The intended target system for this demonstration was the Legbot robot (described in 6.3.2), which was built in parallel with the simulation systems used to develop the Torso Driven Walking control system.

This chapter describes a number of preliminary experiments that were conducted with the physical Legbot robot to determine the robot's suitability as a platform for implementing the Torso Driven Walking control system. These experiments focus on evaluating components of the control system that have not been independently verified by physical implementation on the Ballybot robot in Chapter 5. The two primary areas of concern that have not yet been experimentally verified are:

- The relationship between applied actuator voltage and resultant joint torque.

- The accuracy of joint angle measurements.

The problems with controlling joint torque are a result of two issues:

- The control system requires the actuators to deliver a precise, variable, torque to the robot's joints. While this torque varies during the course of the robot's operation, it must be able to reach a maximum level of ~0.5 Nm. This torque requirement was identified by experiments on the simulated robotic systems.
- Joint friction present in both hip and knee joints of the robot must be compensated for in some way. Generally this involves detecting the magnitude and direction in which the frictional torque is acting, and adjusting the actuator output torque to cancel out the effects of friction. This topic is quite an active area of robotics research (for some examples see Liu [28], Dupont [29]). Since this topic is not the focus of this thesis, in order to simplify the friction compensation, the magnitude of the frictional torque should be as small as possible and consistent.

In addition to torque, experiments on the simulation systems indicate that joint angles need to be estimated with an accuracy of ~3.0 degrees, before the control system performance begins to degrade.

### 6.9.2 Materials

#### i. Legbot experimental platform

This experiment makes use of the physical Legbot robot, described in Chapter 6.3.

### 6.9.3 Methods

#### 6.9.3.1 Determining joint angles

Using the optical encoders integrated with the Legbot's DC motors, a simple algorithm was developed to calculate the joint angles of both the knee and hip joints of each of the robot's legs. Figure 6-49 shows the configuration of the hip and knee motors, and the leg joints on the Legbot robot.

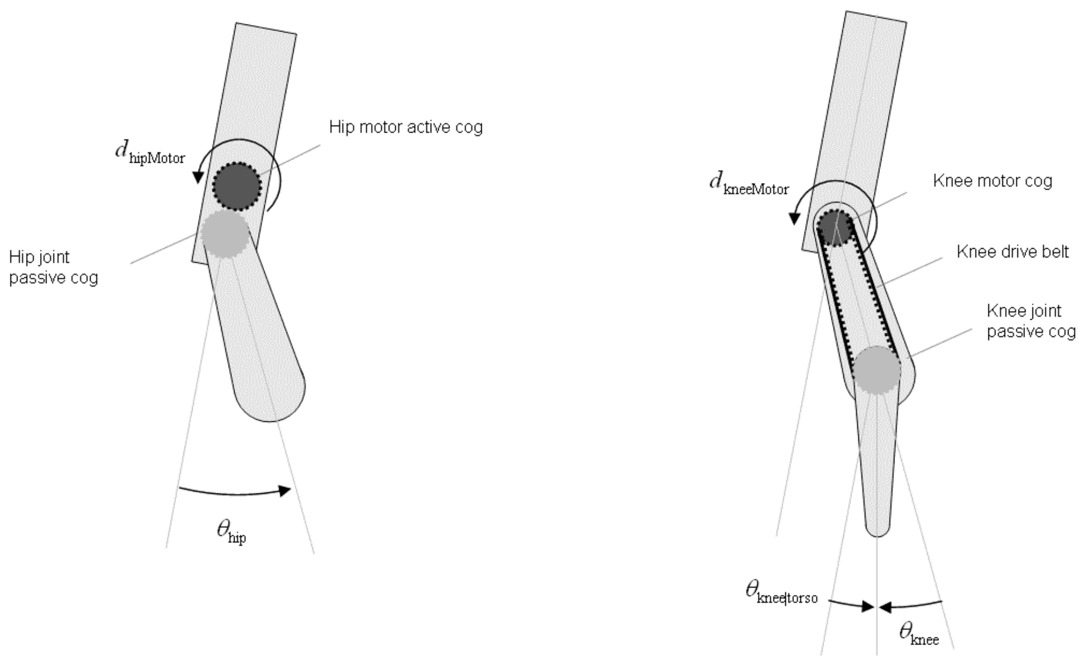


**Figure 6-49 The hip motor is mounted above the hip, and drives the passive hip joint directly. The knee motor shares its axis with the hip joint, and drives the knee joint via a drive belt. Note that there is minimal friction between the knee motor and the hip joint.**

The optical encoders built into each motor operate as odometers, outputting a signal proportional to the distance travelled since the encoder was initialised.

For the hip joint, the hip angle ($\theta_{\text{hip}}$) is proportional to the displacement output by the motor from its zero position ($d_{\text{hipMotor}}$). By measuring the change in displacement value after a known change in hip angle, the following relationship between hip angle and odometer displacement is determined:

$$\theta_{\text{hip}} = -2.59 \cdot d_{\text{hipMotor}} \tag{1}$$

The knee joint angle is slightly more complicated to determine. Since the knee motor is mounted on the robot's torso, the orientation of the shin with respect to the robot's torso is independent of the hip angle, as described by Equation (2):

$$\theta_{\text{knee|torso}} = 2.85 \cdot d_{\text{kneeMotor}} \tag{2}$$

However, most control systems are developed to work with a knee angle relating the change in orientation between the thigh and shin links, such that:

$$\theta_{\text{knee}} = \theta_{\text{knee|torso}} - \theta_{\text{hip}} \tag{3}$$

$$\theta_{\text{knee}} = 2.85 \cdot d_{\text{kneeMotor}} + 2.59 \cdot d_{\text{hipMotor}} \tag{4}$$

Once the robot joints have been calibrated to determine the "zero" positions of the knee and hip encoders, equations (1) and (4) are used to generate angular measurements for each of the robot's joints. The granularity of the angle measurements, as well as any latency in angle measurements will be used to determine the suitability of the encoders for use in the Torso Driven Walking control system.

#### 6.9.3.2 Determining Voltage / Torque relationship

Two simple experimental trials were conducted to investigate the torque response of the Legbot's actuators to an applied voltage. The first trial determines torque generated by an applied voltage when the motor is operating under "normal" conditions. "Normal" means that the motor's torque acts to induce a movement in the robot's joint. The second trial determines torque generated by an applied voltage when the motor is being "back-driven". In this configuration, the motor's torque acts to retard or prevent a movement in the robot's joint.

i. Normal operating mode

Figure 6-50 illustrates the experimental rig used to determine maximum torque that can be generated by the knee joint actuators used by the physical Legbot robot. In this trial the robot's actuators are working in their "normal" operating mode – they are inducing a movement in the robot's knee joint. As the figure shows, the robot is laid upon a flat surface with a known weight suspended from the robot's leg. For each applied voltage being examined, the maximum mass that the motor can lift is experimentally determined, and thus the torque generated by the applied voltage can be found. The resulting torque is sometimes referred to as an actuator's "stall-torque".

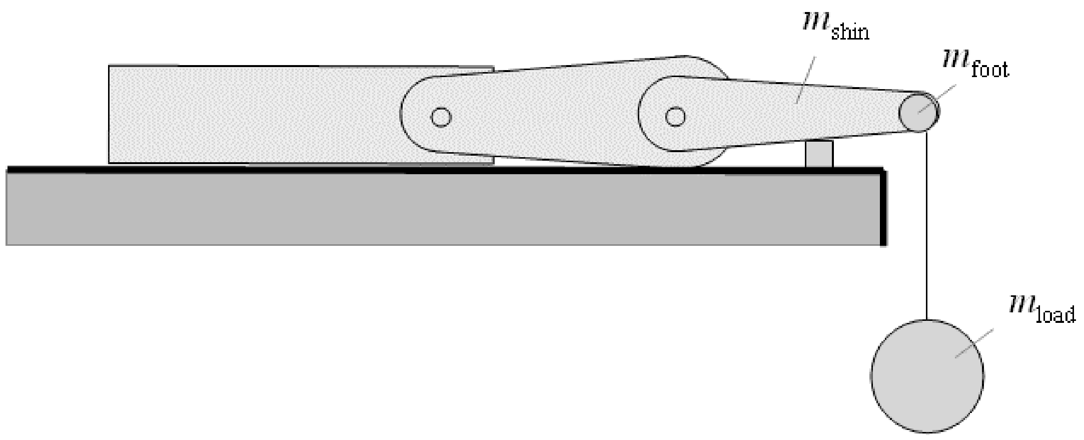


**Figure 6-50 Experimental rig for determining torque/voltage relationship on the Legbot robot, under "normal" operating conditions. A static load is suspended from the robot's foot, and the voltage required for the knee actuator to lift the robot's foot is experimentally determined.**

Figure 6-51 shows the forces acting on the knee joint in this initial trial.

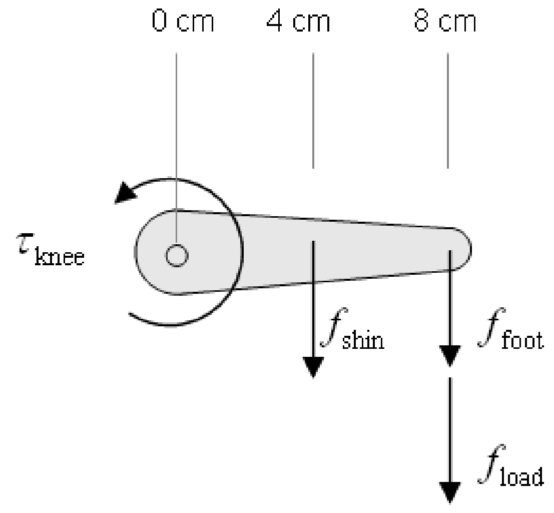


**Figure 6-51 The torque generated by an applied voltage can be estimated by considering the static forces that were overcome in order to raise the robot's foot.**

ii. Back-driven operating mode

In the second trial, the robot's actuator is being "back-driven". The weight applied to the robot's foot is acting against the applied torque to force the knee joint to move in the opposite direction to the induced motor torque. The experimental configuration is similar to the first trial, however this time the robot lies on its side, with a string pulling on the robot's foot with a known horizontal force. For each applied voltage being examined, the minimum mass that will cause the robot's leg to move against the direction of induced motor torque is experimentally determined. From this mass, the torque generated by the motor when it is being back-driven can be found.

#### 6.9.3.3 Determine required torque and noise tolerance

The Genetic Algorithm based Torso Driven Walking evolver described in Chapter 6.6 was used to develop candidate control systems for various robot configurations, including the current configuration of the Legbot robot. For each configuration examined, the maximum torque required and the maximum joint angle measurement errors that can be tolerated are experimentally determined.

In order to find the maximum torque required by each configuration, the control systems will be used to drive the simulated robot as it moves through its standard test pattern. Before each trial the maximum torque the control system is allowed to apply is reduced until the robot fails. In this way, the torque requirements for the robot configuration can be determined.

A similar process is used to determine the magnitude of potential error in joint angle measurements that the control system can cope with.

### 6.9.4 Results

#### 6.9.4.1 Joint angle estimation

Equations (1) and (4), copied from above, relate the motor shaft "displacement" output by the optical encoders to the Legbot's joint angles:

$$\theta_{\text{hip}} = -2.59 \cdot d_{\text{hipMotor}} \tag{1}$$

$$\theta_{\text{knee}} = 2.85 \cdot d_{\text{kneeMotor}} + 2.59 \cdot d_{\text{hipMotor}} \tag{4}$$

The optical encoders are capable of measuring 27,475 "clicks" per metre, suggesting that the sensors measure motor shaft "displacement" to a resolution of approximately 0.036 mm. Substituting $d = 0.036$ into (1) and (4) allows us to determine that the hip joint resolution is in the order of +/- 0.1 degrees, while knee joint measurement is within +/- 0.2 degrees.

However, by physically manipulating the robot's joints it was observed that a significant amount of latency is present in the linkages connecting the robot joints to the motor shaft. It is possible to manipulate each joint by as much as +/- 2 degrees without the sensor picking up the joint position change. These errors are introduced due to gaps between the teeth meshing the hip motor and joint cogs, and especially in the slack present in the belt drive connecting the knee motor to the knee joint.

### 6.9.4.2 Torque vs. Voltage relationship

i. Normal operating mode

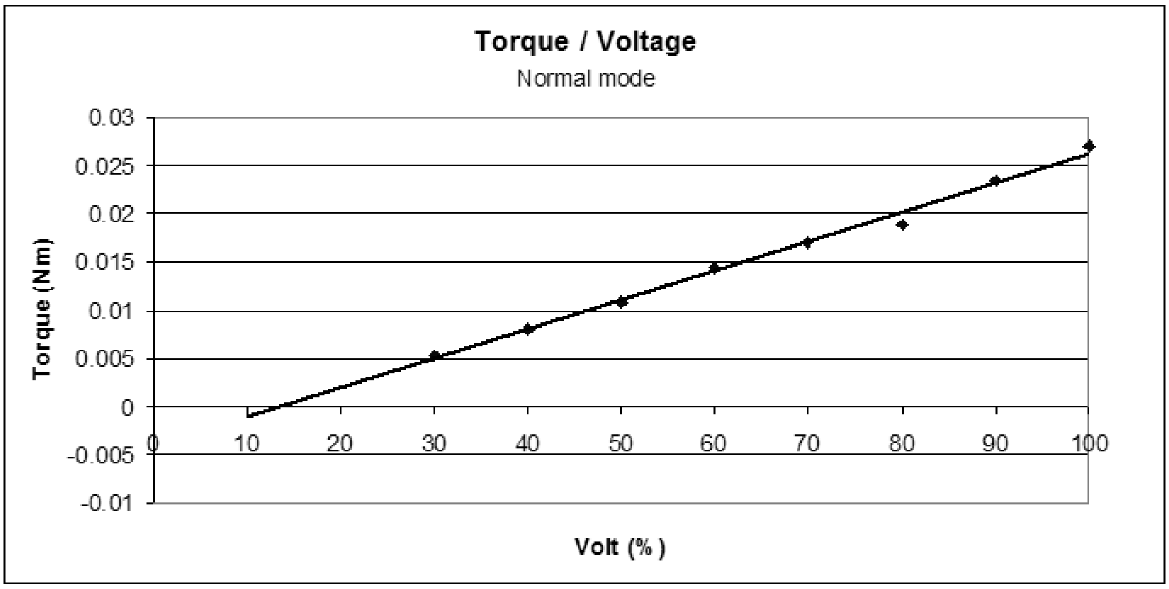


**Figure 6-52 Torque resulting from applied motor voltage under normal operating conditions, with zero angular velocity. The maximum torque that can be generated by the motor in order to induce motion in the joint is approximately 0.028 Nm.**

Figure 6-52 shows the relationship between applied voltage and torque of the knee joint of the Legbot robot performing the static load test. As can be quickly determined from an examination of the chart, the maximum torque applicable by the knee actuators under normal operating conditions is less than 0.03 Nm.

ii. Back-driven operating mode

When the motor is being back-driven, the voltage torque profile is significantly improved, as shown by Figure 6-53. Under these conditions the motor is capable of supplying a maximum retarding torque of around 0.15 Nm.

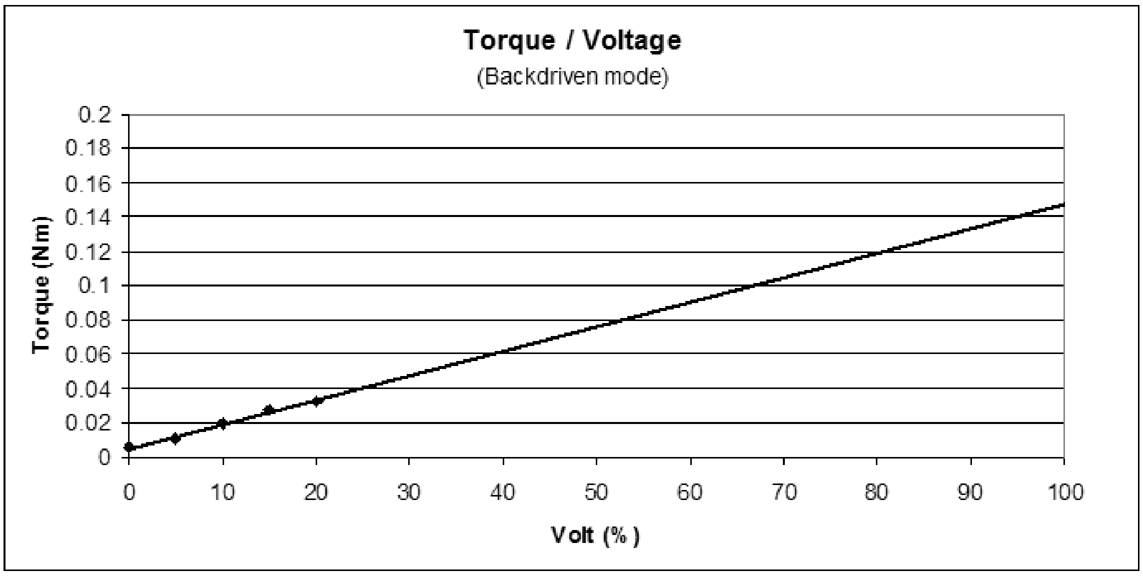


**Figure 6-53 Torque induced by applied motor voltage, when the motor is being "backdriven". The maximum torque that can be generated by the motor to work against an externally applied torque is approximately 0.15 Nm.**

#### 6.9.4.3 Required voltage and noise tolerance

A number of experiments were conducted applying the TDW control system to controlling a simulated Legbot under varying conditions of maximum torque and joint angle quantisation.

The following robot configurations have been examined:

- Simulated Legbot (from preceding chapters)
- Experimental Legbot (using physical robot's weight and dimensions)
- Lighter torso (half the torso weight)
- Lighter torso, legs and feet (half the torso, leg and feet weights)

For each configuration, experiments were conducted to determine the torque requirements, and level of joint angle quantisation that the control system could handle. The results are summarised in Table 6-16.

**Table 6-16 Torso Driven Walking joint torque and angle error requirements for various Legbot configurations.**

| **Robot configuration** | **Maximum torque** | **Maximum joint angle quantisation** |
|---|---|---|
| Simulated Legbot.<br>(30 cm legs)<br>Torso: 1000 g.<br>Thigh: 100 g<br>Shin: 80 g<br>Foot: 10 g | 0.8 Nm | 1 degree |
| Experimental Legbot.<br>(16 cm legs)<br>Torso: 1000g.<br>Thigh: 62g<br>Shin: 57g<br>Foot: 51g | 0.5 Nm | 2 degrees |
| Lighter torso Legbot.<br>(16 cm legs)<br>Torso: 500 g.<br>Thigh: 62 g<br>Shin: 57 g<br>Foot: 51 g | 0.4 Nm | 2 degrees |
| Lighter torso & legs.<br>(16 cm legs)<br>Torso: 500g.<br>Thigh: 31g<br>Shin: 28g<br>Foot: 25g | 0.3 Nm | 3 degrees |

### 6.9.5 Discussion

The first issue considered here is the accuracy of the joint angle measurements. On the current version of the Legbot robot, it was shown that the best accuracy in joint angle measurement that can be expected is in the order of +/- 2.0 degrees. This error is due almost entirely to the latency caused by the design of the linkages between the robot joints and motor shaft.

Experiments on the simulated Legbot robot have shown that this level of joint measurement error has a negative impact on the performance of the control system, however if the control system has been exposed to this level of noise while being trained, the robot is able to maintain balance. If the latency between the joint and motor shaft positions can be improved, then the level of error can be expected to reduce. The best possible granularity of joint angle measurement is approximately 0.1 degrees, assuming all latency can be removed. Angle measurements of this magnitude are well within the performance envelope of the Torso Driven Walking control system.

The second, more critical area involves the maximum torque that can be delivered by the actuators to the robot joints. As can be seen from an examination of the Torque/Voltage characteristics of the Legbot robot's actuators, the maximum torque that can be applied to the robot's legs is very low. When the motor is working to induce a motion in the robot's joint, the maximum torque that can be applied is < 0.028 Nm. The performance is a little better when the motor is being used to retard joint motion (i.e. it is being back-driven), with a maximum retarding torque of ~ 0.15 Nm.

Both of these torque levels are considerably lower than the maximum torque required by the Torso Driven Walking control system. The control system requires the ability to apply torque within the range of +/- 0.5 Nm, with improved performance gained with the ability to apply higher torque. It is clear that no further work can be undertaken with the physical robot until the structure has been redesigned to allow the motors to deliver the torque required by the candidate control systems. This redesign may involve increasing the gearing level of the robot's motors, reducing the weight of the robot, or both.

A related issue to the maximum torque that can be delivered to the robot's joints is the precision with which the torque can be applied. A significant complication related to the delivery of motor torque is the level of friction between the robot's joints. In the Legbot robot this friction is both relatively high and unpredictable, especially in the robot's knee joint. Due to the construction of the robot's leg, the knee joint friction impacts on the torque acting on both the knee and hip joints, exacerbating the problem. Any torque the actuator has to spend to compensate for friction will reduce the maximum torque that can be delivered to the robot's joints.

A final point worth mentioning is that the Torso Driven Walking control system needs to be able to detect when the robot's feet have come in contact with the ground. In the current version of the Legbot, no sensors are installed to detect this ground contact. To facilitate the implementation of the TDW control system on the Legbot, any future version should include some kind of digital switch to indicate ground contact for each foot.

To summarise; in order to find a suitable platform for implementing the Torso Driven Walking control system, a new physical robot will need to be constructed. The following changes should be made to the prototype robot discussed here:

- Reduce latency (as much as possible) between joint and motor shaft positions. An accuracy level of +/- 2 degrees is suggested by simulation trials.
- Increase the level of torque the motors can deliver to the robot's joints. For the Legbot robot here, a maximum torque of 0.5 Nm is suggested by simulation trials.
- Reducing the weight of the robot would reduce the maximum level of torque required. With the same robot dimensions, but halved weight, the torque requirements drop to ~0.3 Nm.

- Reduce the level of friction present in the robot’s joints (or at least make the friction more consistent/predictable).
- Addition of ground contact sensors is suggested.

# 7 Conclusion

The topic of robotic research into biped walking touches on a broad range of potential research areas in the field of robotics, including control system design, sensor noise reduction, friction compensation, mechanical design, dynamic simulation, and many more. Most of these areas are challenging problems in their own right, and worthy of a greater depth of focus than is possible in a single PhD thesis. In the broader field of research in biped locomotion, the focus of this thesis has been on the design of the robot's control system.

A new algorithm "Torso Driven Walking" is presented, which attempts to reduce the complexity of the control problem to that of balancing the robot's torso. All other aspects of the system are indirectly controlled by the changing robot state resulting from direct control of the robot's torso. The result is literally a "top-down" approach to control, where the control system actively balances the top-most component of the robot's body, leaving the control of the lower limbs to a passive "state-driven" system designed to ensure the robot always keeps at least one leg between the torso and the ground.

## 7.1 Project milestones

### 7.1.1 Sensor, actuator & control system interrelationship

Preliminary experiments using the Johnny Walker [2] prototype robot (Chapter 4) demonstrated how important it is that the robot's sensors, actuators and control system are all designed to complement each other. The choice of control system design, sensors and actuators are all tightly coupled. These three components need to be considered together when designing a biped robot. Design of a control system is at least as important as planning the physical construction of the robot and the selection of sensors and actuators. The choice of control system will determine the types of sensors the robot will require, as well as the accuracy required of the sensor's state estimates. Similarly, the control system design will determine the type, and strength of actuators required to power to robot's joints. In the case of the prototype robot, Johnny Walker, state measurements were restricted to a noisy estimate of torso inclination, and foot ground contact. Joint actuators were simple servo motors that could be controlled to move each joint to specified set-point position. Given the constraints imposed by this combination of sensors and actuators, it was not possible to develop a sophisticated control system for the prototype robot.

Common approaches to building biped control systems based on the supporting polygon, or other artificial measures of stability such as the "zero moment point", limit the scope of a walking algorithm. In addition, these approaches seem to result in relatively slow moving and ungainly gaits. In many of these control systems, a robot's control system is essentially a pre-determined joint trajectory. Usually, the trajectory is calculated off-line to ensure the pre-determined stability criteria are met, and modulated on-line to maintain the stability conditions as the system experiences low levels of disturbance.

Dynamic systems, such as those based on inverted pendulum systems, provide a promising solution to the control of walking robots. These systems are always un-balanced, and must work continuously to avoid falling over. Perversely, it is this requirement to constantly react to an unbalanced state that gives these control systems their robust character. The primary sensor for systems that do not rely on a supporting polygon for balance is detection of torso inclination.

### 7.1.2 Torso inclination

Torso inclination was the fundamental state variable used to control an actively balancing robot in this thesis. One of the problems that results from this approach is that detection of torso inclination is difficult. Many researchers working on similar systems avoid the problems associated with self-detection of torso inclination by measuring torso inclination externally to the robot itself. Some researchers take advantage of the fact that their robots are physically connected to the workspace by a boom, from which relative angle to the robot's torso gives a reliable inclination estimate [85]. Since one of the goals of this thesis was to keep all sensors on-board any experimental robot, an alternative solution is required.

Some sensors commonly used for the detection of inclination include gyroscopes, inclinometers and accelerometers. Experiences with the accelerometers installed on the prototype robot Johnny Walker led to the decision to discount these devices as being too prone to noise and vibration and too slow. Preliminary investigations with an inclinometer suggested a similar issue (which is not particularly surprising, since an inclinometer is constructed using accelerometers). After experimentally investigating a number of inclination sensors, a solid-state gyroscope was the most noise resistant and fastest of the sensor type within our research budget.

One of the disadvantages of a gyroscope sensor that the control system would need to overcome is a steady state error, known as "drift" or "bias". In the gyroscope used in this thesis, the "bias" error was observed as an error in angular velocity estimation that changed slowly over time. Both the rate and direction at which the velocity bias would change was unpredictable. Because the robot's torso inclination estimate was calculated by integrating the gyroscope's angular velocity estimate, a small bias in velocity estimate quickly grows to a large, increasing, error in angle estimate.

Because of the difficulty in predicting gyroscope bias, compensating for the drifting angle estimate is also very difficult. Common approaches involve recalibrating the gyroscope signal using some kind of absolute reference, such as landmarks external to the robot. The approach used was to utilise the output from the inclinometer sensor to periodically recalibrate the gyroscope. While the inclinometer was noisier and slower than the gyroscope, its output does give an absolute inclination estimate. Importantly however, the inclinometer output cannot be used continuously. In order to trust the sensor's readings, the robot has to be steady so sensor does not experience acceleration other than those due to gravity, and the sensor output must be heavily filtered. Also no reading can be taken while the robot is experiencing severe vibrations, which are a common event in a walking robot. Any control system utilising these sensors for detecting torso inclination will need to be able to operate in the presence of some gyro drift, with the bias error only periodically being recalibrated.

### 7.1.3 Inverted pendulum to Torso Driven Walking

Inverted pendulum inspired control systems offer an attractive starting point to the investigation of biped walking systems. An inverted pendulum shares many dynamic characteristics with a walking biped body. Such systems are perpetually unbalanced, and yet through the application of a well known and simple control rule, a stable control system that balances the inverted pendulum can be found. The equation relating the pendulum's inclination, displacement from a reference position and resulting horizontal balancing force is reprinted below:

$$f_x = a\,\theta + b\,\dot{\theta} + c\,x + d\,\dot{x}$$

By varying the reference points used to measure the pendulum's horizontal displacement, both the position at which the pendulum balances and the velocity with which it moves can be easily controlled. In this way, a control system based on an inverted pendulum model should result in a robot that not only balances, but is capable of moving at a specified, variable velocity while balancing. Unlike ZMP based control systems, an inverted pendulum balances *by* moving, rather than trying to balance *while* moving.

A significant attribute of inverted pendulum control systems was highlighted though the experiments of this thesis. Inverted pendulum based control systems can automatically maintain balance in the presence of a moderate rate of sensor bias error. The control system compensates for the gyroscope "drift" error by balancing about a point offset from its target horizontal position. As the drift error increases, instead of falling the robot simply balances at a point further away. This behaviour was first observed in experiments evaluating sensors for detecting inclination, using the Ballybot robot in Chapter 5.7.

The ability of an inverted pendulum controller to robustly handle gyroscope bias errors makes the controller a particularly attractive model on which to base the design of a walking control system. This is the motivation behind the design of the Torso Driven Walking control system. The control system balances the robot's torso through the application of a hip joint torque, using the "Torso Attitude Controller", a variation of the standard inverted pendulum solution.

The remaining elements of the Torso Driven Walking control system are concerned with coordination the movement of the robot's feet, ensuring there will always be a foot between the robot's torso and the ground. These systems move the robot's legs relative to its torso, so they do not require a measure of the robot's inclination, instead relying on a measurement of joint angles and ground contact to control the movement of the robot's legs.

This means that the only component of the Torso Driven Walking control system that relies on an absolute estimate of inclination is the inverted pendulum based Torso Attitude Controller. Consequently, the Torso Driven Walking control system will inherit the immunity to inclination bias errors, as well as the variable velocity control, from its inverted pendulum ancestor. Experiments, in Chapters 6.6.4 and 6.8.4.4, have shown that this hypothesis was correct.

### 7.1.4 Ballybot experimental platform

The first physical robot developed was the “Ballybot”, a simple inverted pendulum based robot that balanced by applying a variable torque to a pair of wheels mounted on at a single axis at the base of the robot. This robot was used to experimentally validate candidate sensors for detecting a robot’s inclination for use in an inverted pendulum based control system. The robot also tested the operation of the central component of the Torso Driven Walking control system (the Torso Attitude Controller) in a real-world environment.

Using the Ballybot, a sensor “module” that is fast and accurate enough to keep an inverted pendulum balanced was developed and tested. The sensor module was composed of a rate gyroscope providing angle estimates for the real-time operation of the control system, in conjunction with an inclinometer. The inclinometer was used to perform periodic on-line re-calibration of the gyroscope, to correct for the inevitable bias error. As predicted, even without recalibration of the gyroscope the robot controlled by Torso Driven Walking maintains balance in the presence of the gyroscope bias.

### 7.1.5 Legbot experimental platform

While balancing a legged robot's torso is the central component of the Torso Driven Walking control system, this is only part of the picture. Much of the algorithm complexity is tied up with the requirement to make sure there is always a leg between the robot's torso and the ground. To achieve this result in the Torso Driven Walking control system, this thesis proposes using a simple state machine that coordinates the movement of the robot's legs. The state machine is driven by the position of the supporting foot, with respect to the robot's torso. As the supporting foot moves away from beneath the torso, the raised leg is moved in the opposite direction, so that it is ready to take over the supporting leg's role when required.

In order to test this theory, a second robot platform was designed, the "Legbot" robot. This machine replaces the wheels of the Ballybot robot with a pair of two-link legs, each ending in a foot designed to appear as a "point" in the sagittal plane, but being very wide in the frontal plane. This arrangement effectively constrained the robot to planar movements, simplifying the resulting control system design.

A simulation of the Legbot robot, was used to experimentally develop the Gait Transition Controller component of the Torso Driven Walking control system. The Gait Transition Controller is responsible for coordinating the movement of the robot's legs in response to changes in position of the stance leg. This was an interesting exercise, in that it demonstrated how important control of the robot's legs is to the behaviour of the resulting control system. The leg's motion cannot be too simple, or else disturbances to the torso inclination could be too large for the controller to correct.

The result was that a fairly complicated arrangement of state programs and associated tuning parameters was needed to encode the relative motion of the robot's legs. This complexity meant tuning the Torso Driven Walking control system was a difficult and time consuming process. However, the task is an ideal candidate for the application of adaptive learning techniques, such as genetic algorithms.

### 7.1.6 Genetic algorithm tuning for Torso Driven Walking

Genetic Algorithm programming techniques were applied to the problem of tuning the Torso Driven Walking control system. This effectively changes Torso Driven Walking into a hybrid classical/adaptive control system. The result was the generation of a variable speed controller capable of dynamically balancing a biped robot. This system inherits the gyroscope "bias" immunity of the simpler inverted pendulum systems.

Implementing the Torso Driven Walking control system in a second Legbot simulation environment demonstrated that a Torso Driven Walking control system evolved in one environment can be ported to new (but similar) environment, and that the while the robot may fall, it will make reasonable decisions and may maintain balance for some time. The same experiments showed that in order to operate well in a new environment, the genetic algorithm tuning process must be restarted from scratch each time the system is moved to a new environment. This result is disappointing, since it implies that a large number of experimental trials (in the order of two to three hundred) will be required when porting the control system to a physical robot.

### 7.1.7 The physical Legbot robot

Perhaps the most disappointing outcome from this series of experiments was the inability to apply the Torso Driven Walking control system to a physical robot. Unfortunately, the torque requirements of the Torso Driven Walking control system, while comparable with other walking control systems (see Chapter 6.8), could not be met by the robot intended as the test platform for the algorithm. The design and building the physical robot preceded the understanding of the torque requirements of the control system. A more appropriate design for the legged robot is suggested in Chapter 6.9. A third robot designed from these learning is beyond the scope of the current thesis.

### 7.1.8 Comparisons with alternative control systems

Two alternative control systems have been contrasted with the Torso Driven Walking control system.

For the first comparison the Ballybot robot was utilised as a target system for an artificial neural network based "first-principles" balancing control system [79]. The results of this control system were compared to the implementation of the Torso Driven Walking's "Torso Attitude Controller" subsystem on the Ballybot robot. While the adaptive control system exhibited satisfactory performance, the long training times, complications involved in presenting training data to the neural networks and inconsistency of results were all significant disadvantages. Obviously the Torso Attitude Controller is a far simpler problem than the full-blown TDW control system. It can be expected that the required network training times would dramatically increase as the dimension of the problem increases. Consequently, a "first principles" ANN approach is not an ideal option for implementing a biped walking control system. A simpler alternative for adaptive solutions is to optimise a control system grounded in a conventional framework.

The second alternative control system I investigated was Pratt's "Virtual Model Control" [85]. The VMC control system was implemented on the full Legbot simulation system, and was able to achieve a walking performance comparable with

Pratt's published results. In most of the key areas, the TDW system outperforms VMC on the same benchmark tests, including:

- Variable velocity control of target robot
- Robustness of control system in response to sensor noise.
- Joint torque requirement

Of the indicators considered, the VMC system outperforms TDW in only two areas:

- Tuning parameter complexity
- Minimising unwanted torso movements

The first disadvantage is mitigated by the use of genetic algorithm based parameter tuning by the Torso Driven Walking control system, while the second item was not a requirement when developing the biped walking control system.

### 7.1.9 Summary

To summarise, the specific milestones achieved during the course of this thesis included:

1) Torso inclination identified as the primary system variable of interest.
2) Type of gyroscope sensors required to accurately detect torso inclination identified.
3) Developed "Torso Driven Walking" control system to robustly handle noise expected through the use of these sensors.
4) Experimentally validated sensor selection and torso attitude control subsystem of TDW on a physical robot platform.
5) Experimentally validated full Torso Driven Walking algorithm on a simulated legged robot.

6) Developed a genetic algorithm solution to TDW's major weakness (large number of tuning parameters).
7) Compared the Torso Driven Walking performance with two other distinct control systems:
    a. First principles ANN controller for the Ballybot robot
    b. Virtual Model Control for biped walking.

## 7.2 Review of research goals

While in the conduct of this thesis the ultimate objective of producing a simple, robust, low-cost biped walking machine was not achieved, a significant amount of progress has been made. Most of the primary objectives set at the start of this investigation have been achieved. In particular, a new algorithm for the dynamic control of biped walking robots was developed that demonstrated the ability to control an actively balancing robot, moving with a variable and controllable velocity, while robustly handling a significant level of gyroscope "bias".

Following is a discussion on the achievement against each of the primary goals stated in Chapter 1.2.

### 7.2.1 To design and build a robot that exhibits a natural looking gait.

Due to the physical limitations of the Legbot the Torso Driven Walking system was not able to be implemented on the physical robot. However, biped walking was demonstrated on two separate simulation environments, including one which was written as part of this thesis, and a second that is publicly available. The simulation system has been used to identify areas in which the physical robot must be improved before it can be expected to walk successfully. The most significant change required is an increase in the level of torque that can be supplied to the robot's actuators.

Given that the control system has been tested on a software simulation, it is possible to discuss the second part of this objective – exhibiting a natural looking gait. The following characteristics of the Torso Driven Walking implementation correspond with a "natural looking" gait:

- When moving at constant velocity, the robot maintains an upright torso posture.
- When changing directions, the robot induces an initial "lean" in the direction to which it wants to move.
- The movement of the robot's joints is smooth, and appears "elastic".
- The robot moves with a "straight" supporting leg, not the "bent-kneed" approach common to many biped walking algorithms.

### 7.2.2 To implement a control strategy that does not require a precise kinematic/kinetic model.

Achievement of this research objective was demonstrated in Chapter 6.9. One of the advantages of the Torso Driven Walking control system is the ease with which the control system can be adapted to changes in the target robot's configuration. In many cases the control system continued to work after radically changing the robot's configuration i.e. weight and limb lengths, without requiring the system to be re-tuned. In addition, the use of an adaptive process (genetic algorithms) to determining the control system's tuning parameter values means that the researcher only needs an approximate understanding of the robot's kinematics to successfully implement the control system.

7.2.3 To implement a control strategy that allows operators to command the robot to move with a specific, variable target velocity.

As the results described in Chapters 6.6, 6.7 and 6.8 demonstrate, the Torso Driven Walking control system is able to drive the robot with a variable velocity, backwards and forwards, as well as commanding the robot to maintain position at a specified location.

7.2.4 To accommodate moderate amounts of sensor noise without causing the robot to fall.

The experimental results in Chapters 6.6, 6.7 and 6.8 demonstrate the ability of the Torso Driven Walking control system to operate in the presence of a moderate level of sensor noise. The control system is particularly robust in the presence of Gyroscope bias errors, compensating for the bias error by moving away from the robot's set-point position while maintaining balance. The actual levels of noise the control system can cope with depend on a number of factors, including:

- The magnitude of torque that can be delivered by the joint actuators. When TDW was implemented on systems capable of delivering higher torque, TDW could cope with a higher level of sensor noise.
- The environment in which the control system tuning parameters were evolved. Evolving the control system in an environment experiencing sensor noise resulted in implementations that were better able to cope with sensor noise. As a trade-off, these systems tended to be less precise in controlling the robot's motion.

### 7.2.5 Demonstrate that the Torso Driven Walking control system is a viable control system for biped walking.

The preliminary experiments conducted on the Ballybot robot, together with extensive simulation system experiments have shown that the TDW control system is a viable option for control of a dynamically balancing biped robot. For example:

- The Ballybot has shown that the gyroscope and inclinometer sensor package is capable of balancing an inverted pendulum based robot.
- While the parameter tuning is best done through a genetic algorithm process, the number of trials required to converge to a reasonable walking performance is not prohibitive. (Something in the order of 200 trials can be expected).
- The update rates with which the control system must sample the robot's sensors and determine a new set of control signals is reasonable for most modern controllers. Most of the experiments are conducted using an update rate of 2 ms.
- The joint torque required by the control system, while too large for the intended target robot, are not unreasonable. In 6.9.4.3, Table 6-16 indicates required joint torque for a number of possible robot configurations. For a robot the size of the Legbot, but weighing around 500g, the maximum torque requirement is approximately 0.3 Nm.

### 7.2.6 To keep power sources and computers on board.

Since Torso Driven Walking was not implemented on the target physical robot, this requirement was not satisfied. However, the Ballybot robot, which was developed during investigations conducted for this thesis, did include all power sources and computers "on board".

The major limitation of the Legbot was the level of torque that can be supplied to the robot's actuators. Given that the simulator was able to demonstrate the TWD was effective when the robot configuration was changed, it is reasonable to be expect that if a robot was built that could deliver sufficient torque the power sources and computer could be on board.

### 7.2.7 To restrict the developmental costs to the constraints of a limited budget.

This item is more of an environmental constraint than a requirement – you can't spend what you don't have. However, in order to limit the costs of the development, a number of strategies were adopted to keep the total equipment costs to approximately $3000 for all robots developed as part of this investigation:

- Low cost sensors were used to obtain robot state measurements. This increased requirements on any control system to robustly handle sensor noise.
- Use of the low-cost Eyebot robot platform.
- Where appropriate physical platforms were unavailable, extensive use was made of software simulation techniques.

## 7.3 Future work

The following items illustrate a few potential opportunities for further experiments with the Torso Driven Walking control system. Ideas include working with the current control system implementation, extensions to the control system and the potential to investigate alternative implementations of Torso Driven Walking.

### 7.3.1 A new experimental robot

The most obvious next step in the development of the Torso Driven Walking control system is to construct a new physical robot to demonstrate the control system working in a physical environment. For this to be possible, some significant changes to the physical robot design will need to be undertaken, as outlined in Chapter 6.9.

### 7.3.2 Three dimensional walking

The Ballybot and Legbot robots and simulation systems considered in this thesis are all examples of planar robots, and only need to maintain balance in a single plane (the sagittal plane). The Torso Driven Walking control system has been designed to be extensible to the three dimensional case, and this is a natural course for future work on the control system. Implementing a three dimensional robot controlled by Torso Driven Walking will introduce some interesting new challenges in the robot's physical design. Most significantly, the robot's hip joints will need to support an additional degree of freedom.

### 7.3.3 Variable step length

A number of researchers (for example, Morimoto [76], [77]) have demonstrated that the ability to vary a walking robot's stride length promotes stability in a walking pattern. It may be possible to further improve the performance of the Torso Driven Walking control system by incorporating a velocity based "stride length" variable in the Gait Transition Controller's program for controlling the robot's raised foot.

### 7.3.4 Alternatives to the Gait Transition Controller

Since the Torso Driven Walking control system is designed to be modular, it would be quite possible to replace the existing Gait Transition Controller with alternative modules for coordinating the movement of the robot's feet. While the existing module has been shown to provide strong walking performance, it does exhibit the disadvantage of requiring a large number of tuning parameters. Alternative approaches to describing motion of the robot’s feet, such as Morimoto’s Poincaré maps [76-77], or Boeing’s interpolated splines [74], [82] might provide similar performance but be simpler to implement.

## Appendix A Materials

### A-1 Eyebot robotic platform

The Eyebot family of experimental robots includes all the mobile robots utilising a combination of the EyeCon embedded controller, and the RoBIOS operating system. Eyebot robots have been used in applications as diverse as wheeled, flying, walking, crawling and swimming mobile robots. A team of small, wheeled Eyebot robots regularly participates in international robot soccer competitions.

#### A-1.1 EyeCon controller

The EyeCon controller is the heart of the Eyebot robotic platform. The device is a small embedded controller intended for robotic research, produced by Joker Robotics [Joker]. Originally developed to provide an interface to digital cameras for mobile robotic applications, the EyeCon controller has been designed to allow easy interfacing to a wide range of robotic sensor and actuator devices.

The key to the EyeCon's versatility is the combination of a 32-bit Motorola microprocessor, a wide range of analogue and digital input and output ports and an extensive collection of device drivers and library functions provided by the RoBIOS operating system.

- allowing the robot to be programmed using standard programming languages
- moderately fast performance (The Motorola controller includes a 25MHz CPU)
- Ease of configuration – servos and sensors are simply plugged into the array of I/O ports on the back of the controller.

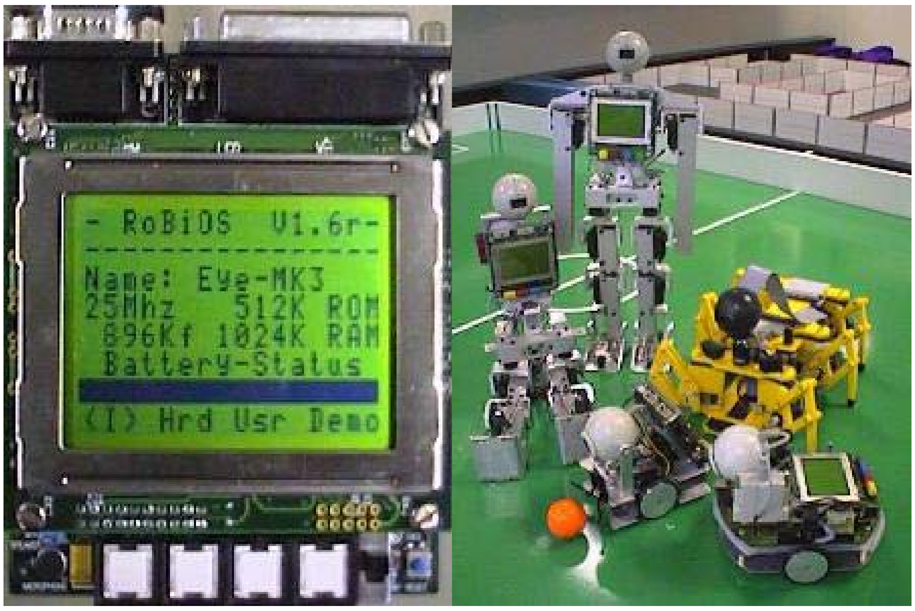


**Figure A-1 The EyeCon controller, together with a collection of some of the mobile "Eyebot" robots using the controller.**

Further details about the EyeCon controller can be obtained online from http://robotics.ee.uwa.edu.au/, or from the book *Embedded Robotics*, Bräunl [27].

## A-2 Actuators

### A-2.1 Servos

Robots may utilise servo motors for joint actuation. These motors provide a form of positional control, where the servo attempts to move the current position to a target angle specified by the controller. No feedback is provided by the servo to the controller to allow a high level control system to verify whether the set point position has been successfully reached.

Positional control is achieved by the servos through the use of an integrated PD controller that moves the servo to a position indicated by a pulse width modulated (PWM) input signal. The proportional and differential parameters of the servo's PD controller are inaccessible to an external controller, which means the response of the servo cannot be modified by the robots. Since most servos are designed to operate light loads, in a robotics environment the loading experienced by the servo often means its performance is sub-optimal.

Additionally, most servos are usually designed to operate with a “dead band” zone around the set point position. When the servo position is within the “dead band” zone, no torque will be applied to the servo shaft. This is to prevent unnecessary servo actuation, conserving power usage. Unfortunately it also means the servo cannot achieve accuracy of more than about 1 degree.

### A-2.2 DC Motors

DC motors are an alternative choice of actuator for robotic applications, which afford greater flexibility to servo systems. Their most important advantage is the ability to control the torque output by the motor. DC motors are relatively inexpensive, compact and powerful for their size.

In the experiments conducted using the Ballybot and Legbot robots (Chapters 5 and 6), I have chosen to work with the range of DC Micromotors developed by Faulhaber (see www.faulhaber.com). These DC Micromotors incorporate both a high precision gearbox and a shaft position encoder in the motor housing.

The inclusion of encapsulated shaft encoders is a significant advantage for motors used in robotics environments, providing easily accessible measures of shaft angular displacement. These measurements can be used by the robot’s controller to calculate the distance a wheeled robot has travelled, or the joint angles of a jointed robot. Since an encoder measures displacement in motor shaft position, some method of calibrating the joint angle or distance measurements must be implemented. This usually involves moving the robot to a known “reference” position before beginning operation.

Unlike servo actuators, a DC motor's shaft revolves freely. This is an advantage when driving wheeled robots, such as the Ballybot, but can also be a disadvantage. If a DC motor is being used as an actuator for a robot joint, as in the Legbot, it is possible for a DC motor to attempt to drive a joint beyond its limits. Care must be taken to prevent the possibility of damaging a robot joint powered by a DC motor actuator.

The extra versatility provided by DC motor control also comes at some cost in terms of ease of use. While servo actuators have an encapsulated positional control system, a robot using DC motors for actuation must implement its own positional, velocity or torque control systems. Regardless of the control scheme desired, implementation of control using DC motors is performed by regulating the input voltage to generate an output torque.

The relationship between applied voltage and motor torque in a DC Motor can be described using the equivalent circuit model, shown in Figure A-2. [27]

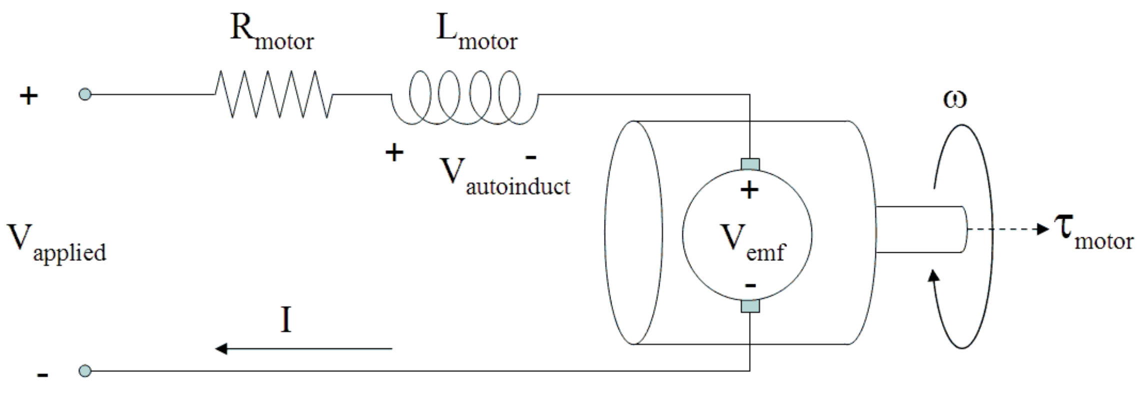


**Figure A-2 Equivalent circuit model for DC motor.**

Motor torque is proportional to current:

$$\tau_{motor} = K_{motor}\, I$$

Back electromotive force is generated by shaft velocity:

$$V_{emf} = K_e \, \omega$$

Voltage induced by changing current:

$$V_{autoinduct} = L_{motor} \left( {}^{dI}/_{dt} \right)$$

From the motor equivalent circuit model, we can say:

$$V_{applied} = I \, R_{motor} + V_{autoinduct} + V_{emf}$$

$$I = (V_{applied} - V_{emf} - V_{autoinduct}) / R_{motor}$$

$$I = (V_{applied} - K_e \, \omega - L_{motor} \, ({}^{dI}/_{dt})) / R_{motor}$$

In a simplified model, $L_{motor} = 0$ H:

$$\tau_{motor} = K_{motor} \, (V_{applied} - K_e \, \omega \,) / R_{motor}$$

$$\tau_{motor} = f \, (V_{applied}, \omega)$$

Motor torque is a function of applied voltage and shaft velocity.

# Appendix B Ballybot dynamic simulation

## B-1 Introduction

I have developed a software simulation of the Ballybot robot system, implemented as an ActiveX control, which can be viewed on a web page using Microsoft's web browser, Internet Explorer. Figure B-3 shows a screenshot of the simulation in action.

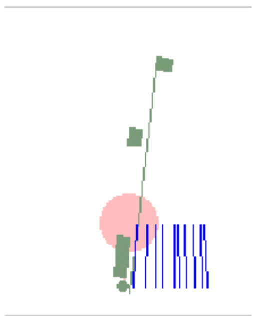


**Figure B-3 Screenshot of the Ballybot simulator. The robot colour, shape and mass distribution can be specified by the user, and will be drawn as specified. The robot's centre of mass is drawn as a light red circle in the background of the image. The robot's position during the last few control system samples is indicated by a series of blue lines.**

A working example of the software simulation is available from:

http://ciips.ee.uwa.edu.au/~suthe-aj/FIRAdemo/balbotdemo.htm.

The simulation takes an "analytical" approach to the dynamic model design, where forces acting between the ground and the robot are calculated such that interpenetration of the bodies being modelled cannot occur. This is a reasonable approach, as I do not have to consider collisions in the system design. An alternative would have been to develop a "penalty" approach to correct for penetration of the robot with the ground, a technique I have used when developing the Legbot simulation in chapter 8. A good discussion of some of the differences between analytical and penalty approaches, as well as other issues to consider before designing a dynamic simulation is given by Baraff [39].

## B-2 System design

The simulation discussed in this chapter has been implemented as an ActiveX control, a binary reusable software component often used in web-based applications. Implementing the simulation in this way allows the software simulation to be easily reused in environments ranging from Visual Basic applications to HTML pages hosted by Internet Explorer.

An ActiveX control is a COM object[34] that supports a standard set of interfaces[35], which allow it to be hosted by applications designed to be control "containers". In this experiment, I have hosted my "BalanceBot" ActiveX control in Visual Basic applications as well as Internet Explorer HTML pages. A detailed discussion of COM and ActiveX controls is outside the scope of this document. Interested parties may refer to Don Box's book "Essential COM" [35] for a thorough discussion of these technologies.

[34] COM Stands for "Component Object Model", and refers to a set of technologies developed by Microsoft to enable software developers to create modular applications supporting binary reuse of components.
[35] A COM Interface is a published set of properties, methods and events through which client applications are able to access a COM object's implementation. The interface is the mechanism used by a COM component to interact with other parts of the system.

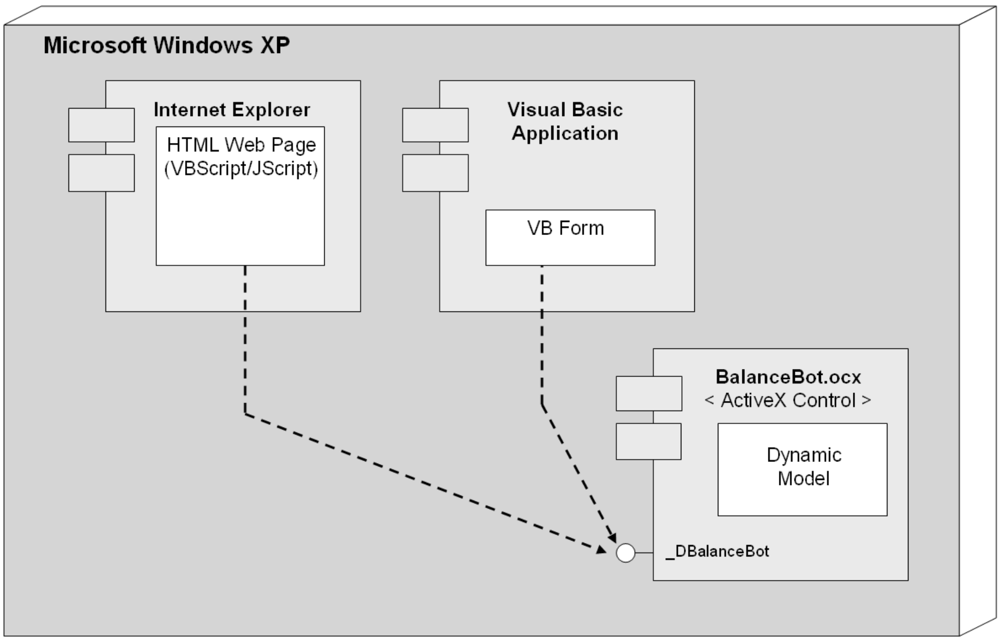


**Figure B-4 Component Diagram showing the relationship between the BalanceBot ActiveX control, and two possible control containers, Internet Explorer and a Visual Basic application. A container could instantiate more than one copy of the BalanceBot ActiveX control.**

As well as the standard interfaces which make the COM object an ActiveX control, a user extensible interface *IDispatch* allows controls to make custom methods, properties and events available to their hosting application. This is how the Visual Basic and HTML applications are able to control the Ballybot software simulation. In the BalanceBot ActiveX control, this dispatch interface is *_DBalanceBot*.

## B-3 Software design

A simplified class diagram (Figure B-5) illustrates the primary design structure of the BalanceBot ActiveX control.

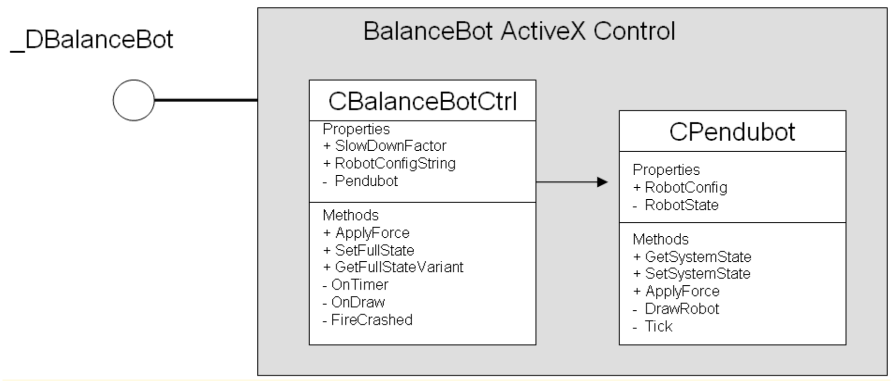


**Figure B-5 Simplified Class Diagram showing the three major components of the BalanceBot ActiveX control software design.**

## B-3.1 _DBalanceBot interface

This interface, inheriting from *IDispatch* is the mechanism used by external objects to access the ActiveX control's implementation. The interface is made up of properties, methods and an event:

**Table 7-1 _DBalanceBot Interface**

| **_DBalanceBot Interface** | |
|---|---|
| *Methods:* | |
| ApplyForce | Changes the force used by the dynamic model when calculating changes to the system state. |
| GetFullStateVarian t | Forces an update of the system state, and returns the resulting state variables. |
| SetFullState | Resets the system state variables. |
| *Properties:* | |
| RobotConfigString | Used to specify the shape, mass and colour of the Ballybot's cross-section. |
| *Events:* | |
| Crashed | Fired up to the control's container if the robot hits the ground, or leaves the workspace. |

### B-3.2 CBalanceBotCtrl class

The primary function of this class is to provide an implementation for the _DBalanceBot interface.

### B-3.3 CPendubot class

This class is responsible for implementing the dynamic model of the robotic system. The derivation of the dynamic model is described in detail in 7.4.4. The CPendubot class maintains the robot's system state and using the system model, calculates changes to the system state as a result of applied force and elapsed time.

## B-4 Dynamic model

The heart of a software simulation is the dynamic model used to predict the behaviour of the system. The objective of the dynamic model is to determine the robot state changes experienced by the system, given a known initial system state and input signal. Due to the simple design of the Ballybot, the robot state can be fully described using four system variables, describing linear and angular displacement and velocity. The input signal is a combination of the linear acceleration experienced by the base of the robot, and the robot's angular acceleration:

$$\text{Robot state:}\quad \tilde{x}(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \theta(k) \\ \dot{\theta}(k) \end{bmatrix} \quad \text{Acceleration:}\quad \tilde{u}(k) = \begin{bmatrix} \ddot{x}(k) \\ \ddot{\theta}(k) \end{bmatrix}$$

**Assumption 1: The behaviour of a continuous system can be modelled by a discrete system, where acceleration values are held constant between samples. As the value for *T* approaches zero, then the behaviour of the discrete dynamic model approaches that of the continuous system being modelled. In this simulation, I have used a value *T* = 0.01 seconds.**

If both robot state $\tilde{x}$ and acceleration vector $\tilde{u}$ are known for a time (*k*), then the robot state at time $\tilde{x}(k+1)$ can be calculated as follows:

$\tilde{x}(k+1) = \mathrm{A}\ \tilde{x}(k) + \mathrm{B}\ \tilde{u}(k)$, where:

$$\mathrm{A} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} T^2/2 & 0 \\ T & 0 \\ 0 & T^2/2 \\ 0 & T \end{bmatrix}$$

*T* = the simulation update interval (0.01 sec).

In order to predict the motion of the Ballybot simulator, the value of the acceleration vector $\tilde{u}(k)$ must be calculated for each time interval. As with inverted pendulum systems [ref], the linear and angular acceleration values making up $\tilde{u}(k)$ are a function of the robot's current system state, and applied horizontal force:

$$\tilde{u}(k) = f(\tilde{x}(k), \mathrm{F}_x(k)) \qquad (1)$$

To determine the relationship between system state, applied force and the resulting accelerations, the Ballybot system model must be analysed. The system model diagram of a Ballybot robot can be redrawn using a simple equivalent system model, as shown in Figure B-6.

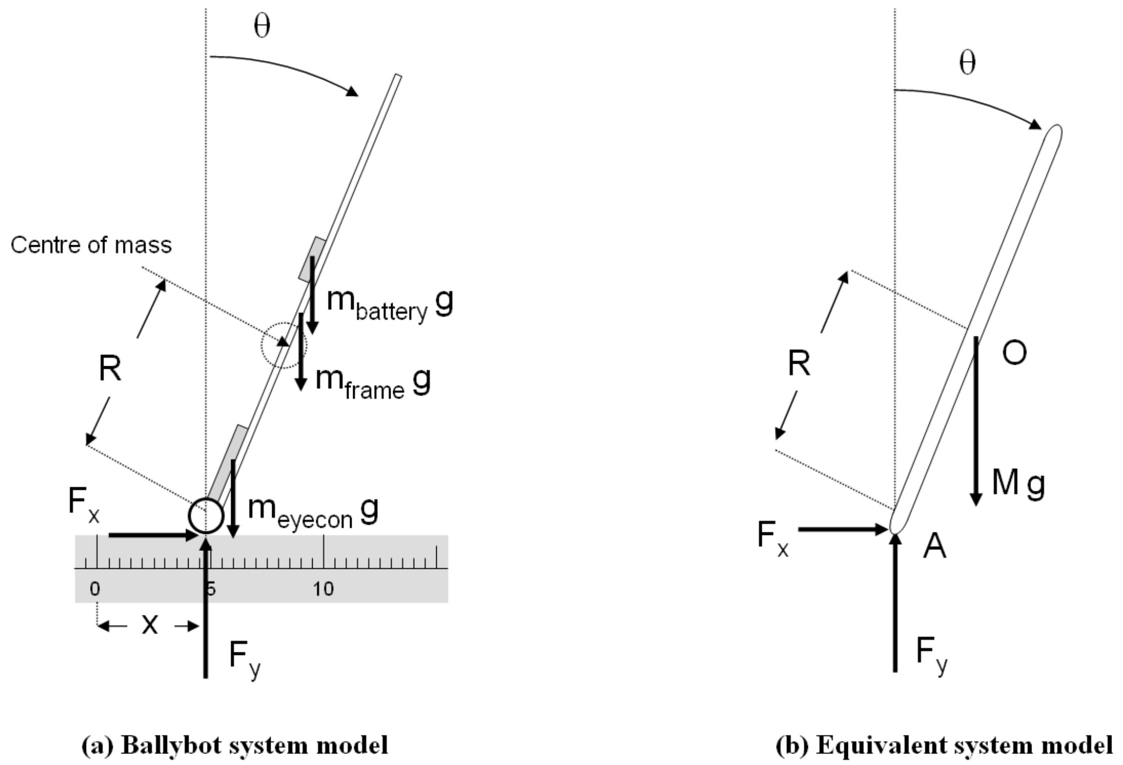


**Figure B-6 Ballybot equivalent system model. The robot can be modelled by a single rigid link, sharing the same total mass, centre of mass and moment of inertia as the Ballybot.**

**Table B-2 Ballybot dynamic model system variables**

| | | | |
|---|---|---|---|
| O | Centre of mass (centroid) of robot | M | Total mass of robot |
| A | Ground contact point of robot | g | Gravitational acceleration (approx. 9.81 ms$^{-2}$ towards the ground) |
| R | Distance from A to O | $F_x$ | Horizontal ground reaction force (proportional to motor torque) |
| θ | Angle of inclination of robot | $F_y$ | Vertical component of ground reaction force |
| $\bar{I}$ | Moment of inertia, about the centre of mass (point O) | x | Displacement of the base of the robot |

Balancing forces in the x and y directions:

$$\sum_x Force = M\ \ddot{x}_O$$

$$F_x = M\ \ddot{x}_o \tag{2}$$

$$\sum_y Force = M\ \ddot{y}_O$$

$$F_y - M\ g = M\ \ddot{y}_o \tag{3}$$

Balancing moments about the centre of mass of the system:

$$\sum_{clockwise} \overline{Moment} = \bar{I}\ddot{\theta}$$

$$F_y\ R \sin\theta - F_x\ R \cos\theta = \bar{I}\ddot{\theta} \tag{4}$$

**Assumption #2: The base of the robot will never leave the ground, therefore the velocity and acceleration of point A will always be horizontal.**

$$\dot{y}_A = 0\ ms^{-1} \tag{5}$$

$$\ddot{y}_A = 0\ ms^{-2} \tag{6}$$

The velocity diagram in Figure B-7 illustrates the relationship between the velocity of the base of the robot ($v_A$), and the velocity of the centre of mass of the robot ($v_O$):

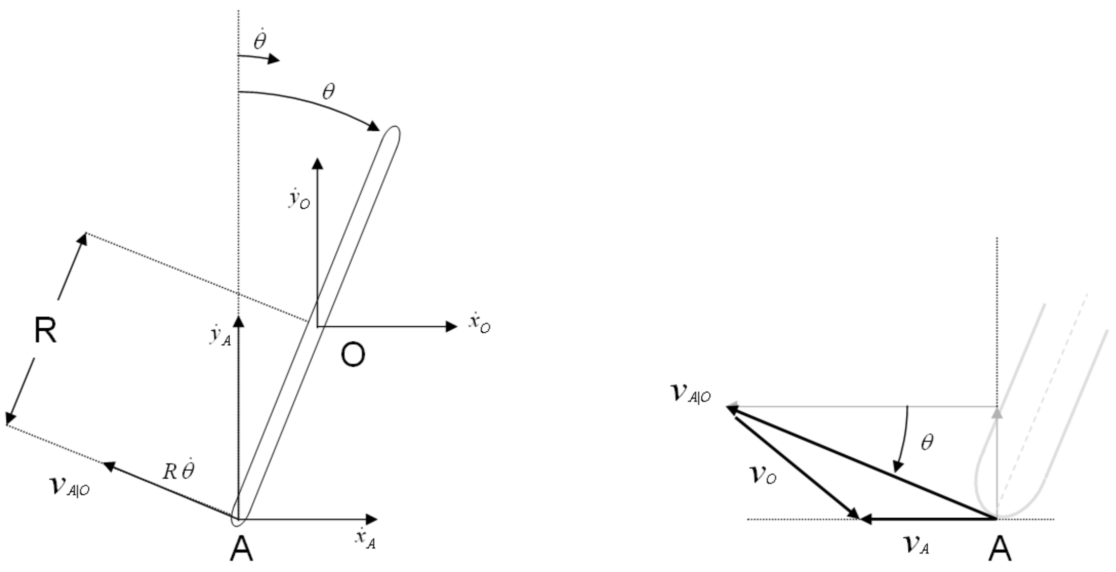


**Figure B-7 Velocity diagram for Ballybot system model. The relative velocities of A and O are a function of angular velocity. Due to assumption 2, the resultant velocity of A must be horizontal.**

$$v_A = v_O + v_{A|O}$$

$$v_A = \begin{bmatrix} \dot{x}_A \\ 0 \end{bmatrix}$$

$$v_O = \begin{bmatrix} \dot{x}_O \\ \dot{y}_O \end{bmatrix}$$

$v_{A|O} = R\dot{\theta}$, perpendicular to $\overline{AO}$

$$v_{A|O} = \begin{bmatrix} -R\dot{\theta}\cos(\theta) \\ R\dot{\theta}\sin(\theta) \end{bmatrix}$$

Therefore:

$$\dot{x}_A = \dot{x}_O - R\dot{\theta}\cos(\theta) \qquad (7)$$

$$\dot{y}_A = \dot{y}_O - R\dot{\theta}\sin(\theta)$$

We know from equation (5) that $\dot{y}_A = 0$, so:

$$\dot{y}_O = R\dot{\theta}\sin(\theta) \tag{8}$$

The acceleration diagram in Figure B-8 illustrates the relationship between the acceleration of the base of the robot ($a_A$) and the acceleration of the centre of mass of the robot ($a_O$):

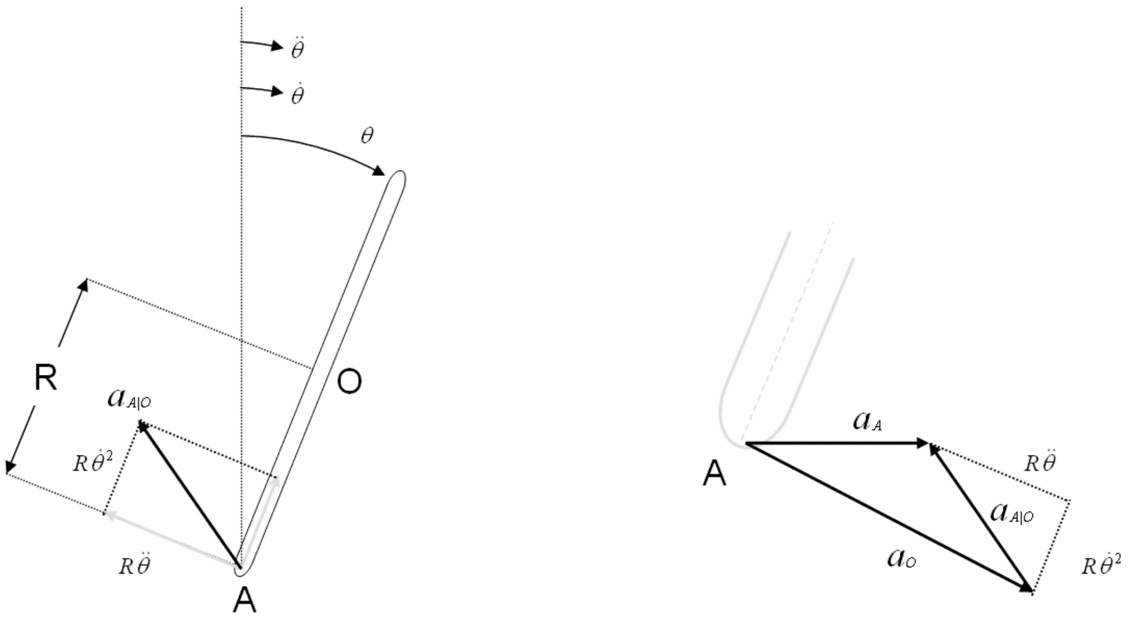


**Figure B-8 Acceleration diagram for Ballybot system model. The relative accelerations of A and O are a function of angular velocity and acceleration. As stated by assumption 2, the resultant acceleration of A must be horizontal.**

$$a_A = a_O + a_{A|O}$$

$$a_{A|O} = R\ddot{\theta}\ \text{(Perpendicular to }\overline{AO}\text{)} + R\dot{\theta}^2\ \text{(In the direction of }\overline{AO}\text{)}$$

$$a_{A|O} = \begin{bmatrix} R\dot{\theta}^2\sin(\theta) - R\ddot{\theta}\cos(\theta) \\ R\dot{\theta}^2\cos(\theta) + R\ddot{\theta}\sin(\theta) \end{bmatrix}$$

Therefore:

$$\ddot{x}_A = \ddot{x}_O + R\dot{\theta}^2 \sin(\theta) - R\ddot{\theta}\cos(\theta) \tag{9}$$

$$\ddot{y}_A = \ddot{y}_O + R\dot{\theta}^2 \cos(\theta) + R\ddot{\theta}\sin(\theta)$$

We know from equation (6) that $\ddot{y}_A = 0$, so:

$$\ddot{y}_O + R\dot{\theta}^2 \cos(\theta) + R\ddot{\theta}\sin(\theta) = 0 \tag{10}$$

Substitute (2) $\rightarrow$ (9):

$$\ddot{x}_A = \frac{F_x}{M} + R\dot{\theta}^2 \sin(\theta) - R\ddot{\theta}\cos(\theta)$$

$$\ddot{x}_A = \frac{F_x + MR(\dot{\theta}^2 \sin(\theta) - \ddot{\theta}\cos(\theta))}{M} \tag{11}$$

Substitute (10) $\rightarrow$ (3):

$$Mg - F_y = MR(\dot{\theta}^2 \cos(\theta) + \ddot{\theta}\sin(\theta))$$

$$F_y = Mg - MR\dot{\theta}^2 \cos(\theta) - MR\ddot{\theta}\sin(\theta) \tag{12}$$

Substituting (12) $\rightarrow$ (4):

$$R\sin(\theta)(Mg - MR\dot{\theta}^2 \cos(\theta) - MR\ddot{\theta}\sin(\theta)) - F_x R\cos(\theta) = \bar{I}\ddot{\theta}$$

$$\ddot{\theta} = \frac{g\sin(\theta) - \cos(\theta)\left(\dfrac{F_x}{M} + R\dot{\theta}^2 \sin(\theta)\right)}{\dfrac{\bar{I}}{MR} + R\sin^2(\theta)} \tag{13}$$

To summarise, the set of equations on the following page are used by the Ballybot simulator to iteratively calculate the system state of the simulated robotic system:

**Ballybot system model equations:**

$$\tilde{x}(k+1) = \mathrm{A}\ \tilde{x}(k) + \mathrm{B}\ \tilde{u}(k)$$

$$\tilde{x}(k) \quad = \begin{bmatrix} x_A(k) \\ \dot{x}_A(k) \\ \theta(k) \\ \dot{\theta}(k) \end{bmatrix}$$

$$\mathrm{A} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} T^2/2 & 0 \\ T & 0 \\ 0 & T^2/2 \\ 0 & T \end{bmatrix}$$

$$T = 0.01\,\mathrm{sec}$$

$$\tilde{u}(k) \quad = \begin{bmatrix} \ddot{x}_A(k) \\ \ddot{\theta}(k) \end{bmatrix}$$

$$\ddot{\theta} = \frac{g\sin\theta - \cos\theta\left(\dfrac{F_x}{M} + R\dot{\theta}^2\sin\theta\right)}{\dfrac{\bar{I}}{MR} + R\sin^2\theta}$$

$$\ddot{x}_A = \frac{F_x + MR\,(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{M}$$

## Appendix C Legbot dynamic model and software simulation

### C-1 Introduction

As with the Ballybot series of experiments (Chapter 5), a software simulation of the Legbot robotic system has been developed. This simulation allowed me to experiment with various control systems designs, without risking damage to the physical robot. Ideally, the simulator would also have allowed me to conduct preliminary investigations into possible robot designs, before beginning construction of the physical robot. Unfortunately, due to time constraints this was not possible, and the Legbot robot was designed and built in parallel with the development of the software simulation.

The simulation has been implemented as an ActiveX control, which can be downloaded and viewed on a web page using Microsoft's browser, Internet Explorer. Since an ActiveX control is a type of COM object, the resulting simulation is a binary re-usable component that can be easily included in applications developed using any environment that supports COM interoperability. For example, some compatible environments include: .NET, C++, Visual Basic, Internet Explorer/HTML, etc.

This simulation is far more complex than the Ballybot simulator discussed in chapter 7.4, with additional features including:

- Multiple robot links acting on each other.
- Interaction between the robot and the ground.
- Possibility for ballistic motion (i.e. jumping).

Due to the additional computational complexity introduced by the interaction of multiple robot links, the simulation is no longer able to update in "real time". As a result, rather than being driven by an internal timer, the client application is responsible for advancing the simulation's clock.

The simulation is designed to model a generic multi-linked planar robot, where links are arranged in a hierarchical tree-like manner. The *head* of each link may be connected to a single "parent" link, and the *tail* to multiple "child" links. Connections between links are the robot's joints, and the simulation assumes the control system will be able to apply variable torque acting at each of the joints. The shape and mass of each link can be specified by the simulation's client application, with each link made up of a collection of squares, rectangles, circles and rods of varying mass. To help users readily identify each link, a distinct colour can be assigned to each link. A series of example robot configurations is shown in the screenshots (Figure C-9, Figure C-10 and Figure C-11).

Features of the graphical output of the Legbot simulator ActiveX control include:

- A graphical representation of the robot's current state shows the robot position in relation to its environment.
- The centres of mass of each link, and of the total robot, are represented by a series of grey circles.
- Foot contact displacement labels are printed beneath each contact point.
- Ground contact force vectors are drawn at the contact points, the length of each vector is proportional to the simulator dynamic model's estimate of ground reaction force.

Figure C-9, Figure C-10 and Figure C-11 show screenshots taken of the simulator operating with various examples of *n*-link planar robots:

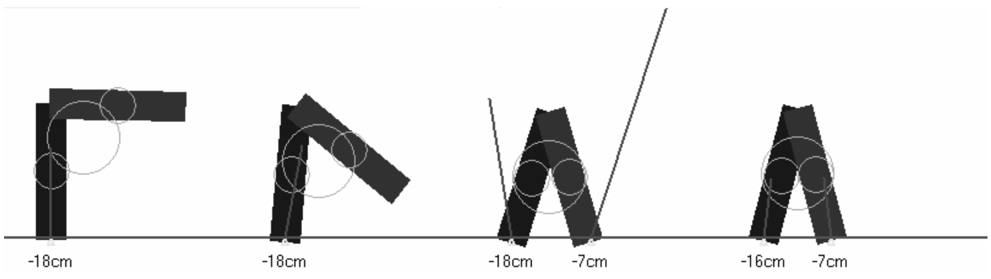


**Figure C-9 A sequence of screenshots of a passive 2-link chain falling. The chains start in a known configuration, and then collapse due to the effects of gravity. No torque is applied to the robot joints as the structure collapses.**

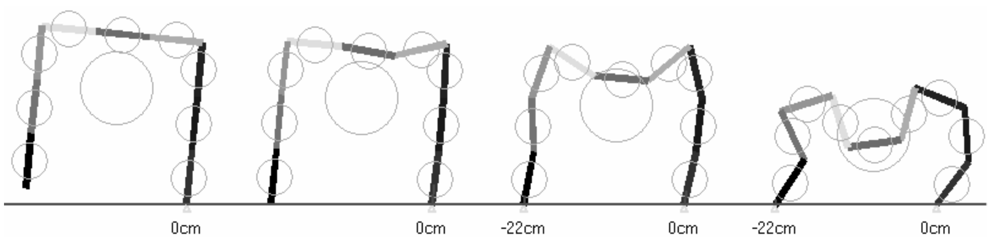


**Figure C-10 A sequence of screenshots of a passive 9-link chain falling.**

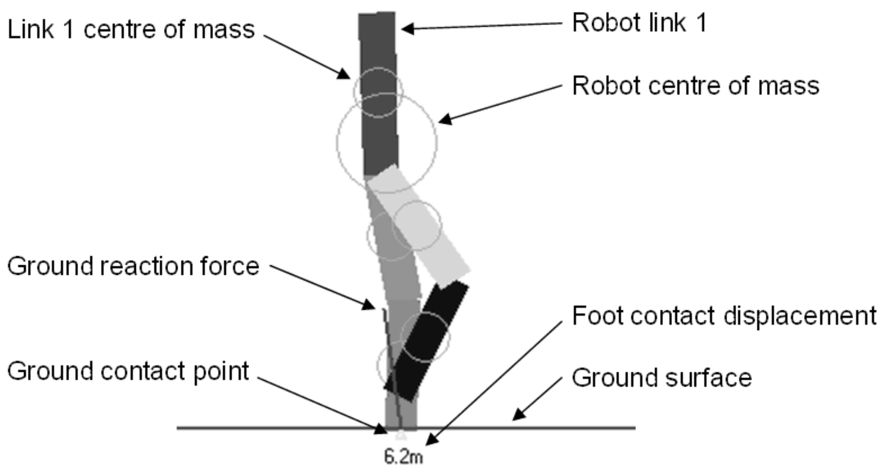


**Figure C-11 Screenshot of 5-link biped walking. Examples of the various display elements are labelled.**

## C-2 System design

The system design of the Legbot simulator is identical to that of the Ballybot simulator described in Appendix B. The simulation has been implemented as an ActiveX control[36]; a binary component that can be easily reused in environments ranging from Visual Basic applications to HTML pages viewed using Internet Explorer. A component diagram illustrating the relationship between the simulation, and two possible control containers is shown in Figure C-12:

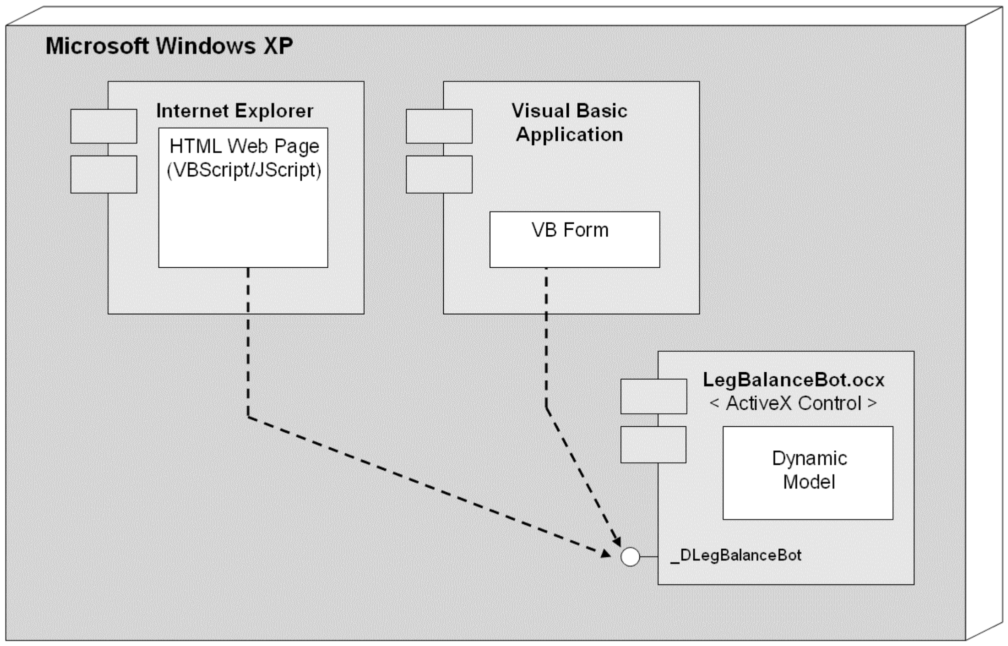


**Figure C-12 Component Diagram showing the relationship between the LegBalanceBot ActiveX control, and two possible control containers, Internet Explorer and a Visual Basic application. A container could instantiate more than one copy of the LegBalanceBot ActiveX control.**

## C-3 Software design

A simplified class diagram (Figure C-13) illustrates the primary design structure of the Legbot ActiveX control.

[36] For a thorough discussion of ActiveX, COM and related technologies, refer to "Essential COM", by Don Box [35].

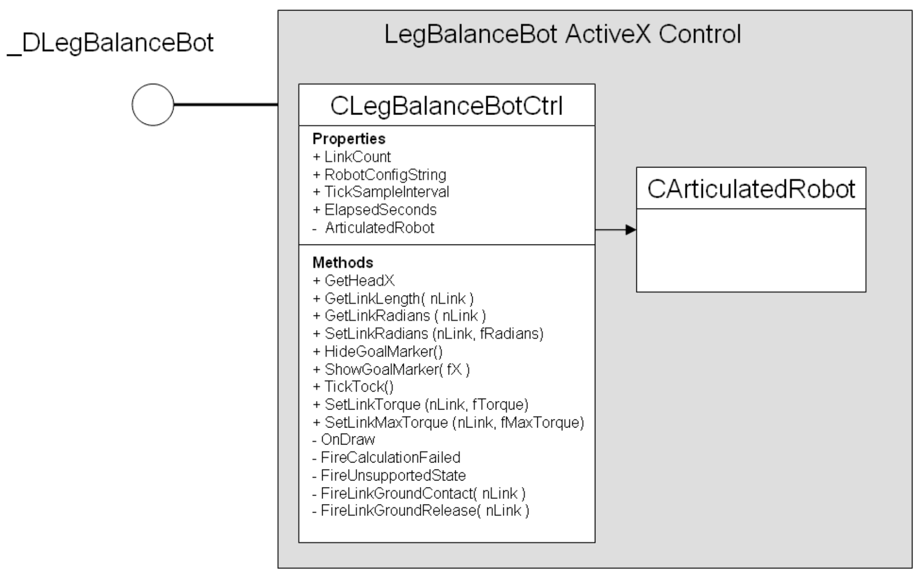


**Figure C-13 Simplified Class Diagram, showing the three major components of the LegBalanceBot ActiveX control software design.**

### C-3.1 _DLegBalanceBot interface

This interface, inheriting from *IDispatch* is the mechanism used by external objects to access the ActiveX control's implementation. The interface is made up of properties, methods and events:

**Table C-3 _DLegBalanceBot Interface**

| **_DLegBalanceBot Interface** | |
|---|---|
| *Methods:* | |
| TickTock | Used to increment the simulation's clock time. |
| SetLinkTorque | Apply a torque to the joint at the *head* of a specified link. |
| SetLinkMaxTorque | Set the maximum torque magnitude that can be applied to the *head* of the specified link. |
| GetLinkLength | Get the length of the specified link. |
| GetLinkRadians | Gets the joint angle of the *head* of the specified link in radians. |
| SetLinkRadians | Sets the joint angle of the *head* of the specified link in radians. (Used during initialisation). |
| GetHeadX | Gets the horizontal displacement of the *head* of the first |

| | |
|---|---|
| | robot link. |
| ShowGoalMarker | Draws a goal marker on the ground at the specified horizontal displacement. |
| HideGoalMarker | Removes the goal marker from the simulation display. |
| *Properties:* | |
| RobotConfigString | Used to specify the structure of the robot, as well as the shape, mass and colour of each robot link. |
| LinkCount | Returns the number of links making up the robot. |
| TickSampleInterval | Number of simulated seconds that elapse with each "TickTock" method call. |
| ElapsedSeconds | The number of simulated seconds that has elapsed since the start of a simulation run. |
| *Events:* | |
| CalculationFailed | Fired up to the control's container if the dynamic model encounters an unrecoverable error. |
| UnsupportedState | Fired up to the control's container if the robot is in an invalid state. Usually this means the robot has fallen over. |
| LinkGroundContact | Fired up to the control's container whenever a robot foot hits the ground. |
| LinkGroundRelease | Fired up to the control's container whenever a robot foot leaves the ground. |

### C-3.2 CBalanceBotCtrl class

The primary function of this class is to provide an implementation for the _DLegBalanceBot interface.

### C-3.3 CArticulatedRobot class

This class is responsible for implementing the dynamic model of the robotic system. The derivation of the dynamic model is described in detail in C-4. The CArticulatedRobot class maintains the robot's system state and using the system model, iteratively calculates changes to the system state as a result of applied joint torques and elapsed time.

## C-4 Dynamic model

The dynamic model of a general, multi-linked planar robot, such as the "Legbot" is far more complicated than that of the simple Ballybot system described in Appendix B. However, the goal of the model is the same: Given a current known robot state and applied input torques, the model needs to determine the robot state changes that will be experienced by the system.

### C-4.1 System state

The system state of any multi-linked, planar robot can be described by specifying:

- The angle and angular velocity of each link making up the robot
- The position and velocity of the robot's centre of mass, with respect to a global coordinate system.

For a planar robot with *n* links, the system state can be represented by equation (1):

*n*-link robot state vector:
$$\tilde{x}(k) = \begin{bmatrix} \theta_1(k) \\ \dot{\theta}_1(k) \\ \vdots \\ \theta_n(k) \\ \dot{\theta}_n(k) \\ x_{com}(k) \\ \dot{x}_{com}(k) \\ y_{com}(k) \\ \dot{y}_{com}(k) \end{bmatrix} \qquad (1)$$

As in the Ballybot simulator, the task of this dynamic model is to determine accelerations experienced by the joint variables in response to applied torque, and environmental interactions. These accelerations can be described by an acceleration vector:

State acceleration vector: $$\tilde{u}(k) = \begin{bmatrix} \ddot{\theta}_1(k) \\ \vdots \\ \ddot{\theta}_n(k) \\ \ddot{x}_{com}(k) \\ \ddot{y}_{com}(k) \end{bmatrix} \qquad (2)$$

Once the state vector $\tilde{x}(k)$ and acceleration vector $\tilde{u}(k)$ are known for a sample time, then calculating the next state is straightforward:

$$\tilde{x}(k+1) = f_n\left(\tilde{x}(k), \tilde{u}(k)\right) = \begin{bmatrix} \theta_1(k) + T\,\dot{\theta}_1(k) + \frac{T^2}{2}\ddot{\theta}_1(k) \\ \dot{\theta}_1(k) + T\,\ddot{\theta}_1(k) \\ \vdots \\ \theta_n(k) + T\,\dot{\theta}_n(k) + \frac{T^2}{2}\ddot{\theta}_n(k) \\ \dot{\theta}_n(k) + T\,\ddot{\theta}_n(k) \\ x_{com}(k) + T\,\dot{x}_{com}(k) + \frac{T^2}{2}\ddot{x}_{com}(k) \\ \dot{x}_{com}(k) + T\,\ddot{x}_{com}(k) \\ y_{com}(k) + T\,\dot{y}_{com}(k) + \frac{T^2}{2}\ddot{y}_{com}(k) \\ \dot{y}_{com}(k) + T\,\ddot{y}_{com}(k) \end{bmatrix} \qquad (3)$$

$T$ Represents the simulator's iteration interval, and must be a sufficiently small time to ensure the system accurately models a real robot. I used a time interval of 0.0001 seconds.

The iteration time interval used in this model needed to be far smaller than the 0.01 seconds used in the Ballybot simulator. This is because in a multi-linked robot, small errors in state measurements at the base of the robot are quickly magnified into large errors as their influence propagates through the chain of robot links. An unfortunate side effect of using a small value for $T$ is that the simulation can no longer update in "real-time". In a six link planar robot simulating the Legbot, the simulation updates roughly ten times slower than a "real-time" simulation would. I was not too concerned by this, as having a slow simulation is better than an inaccurate one!

Determining the acceleration vector $\tilde{u}(k)$ requires the coordinated use of three different models, as shown in Table C-4:

**Table C-4 Calculating state acceleration vector requires the use of three different dynamic models.**

| | |
|---|---|
| Collision model | Determines the forces resulting from interaction with the environment. In particular, robot feet striking the ground. |
| Ballistic model | Tracks the robot's centre of mass, and calculates linear accelerations of the centre of mass. |
| Link/Torque model | Iteratively calculates angular accelerations based on externally applied forces, internally applied torques and the current system state. |

### C-4.2 Collision model

Collisions between rigid objects are notoriously difficult to model accurately, and are a topic that continues to receive extensive treatment from researchers.

The approach I have used in the Legbot simulator is a "penalty" technique, where a reactive "penalty" force is calculated whenever two bodies collide. The penalty force model I have used is a variation of the "spring-damper" system, often used in dynamic simulations. An imaginary spring is inserted between the two points of the system which have collided, and as the objects penetrate each other, this imaginary spring stretches. The force exerted by the spring is fed back into the dynamic model, and works to restore a state of no penetration.

The algorithm used to implement this system is to:

1. Check for new contacts (collisions) and invalid states
2. Check for lost contacts

3. Determine current penetration of each contact point
4. Calculate penalty force acting at each contact point

The output of the collision model is an array of current contact points, together with the linear penalty force acting on the robot link at that point. These details are passed on to subsequent components of the dynamic model.

i. Check for new contacts and invalid states

Using the system state $\tilde{x}(k)$, and knowledge about the structure of the robot's environment, each link in the robot is tested for a new contact with the environment.

Even though each link may be comprised of a number of geometric elements, as a simplification in the collision model each link is treated as a simple line connecting the head and tail joints of the link. Contact is judged to have occurred if the y-axis position of either end of a link is less than ground level.

Since the state variables are iteratively updated, it is unlikely that the collision model will be run at the exact moment a collision occurs – some interpenetration will already have occurred. Rather than attempt to extrapolate back in time to determine the exact collision point, I have used the simplification:

New collision point = ground surface at the x-position of the link end effector.

This approximation is reasonable due to the small update interval used by the simulation ($T = 0.0001$s).

If the link end contacting the ground is a joint connecting to another link, then the system will halt due to the robot being in an invalid state. This makes subsequent simulation implementation much simpler, and is valid because the Legbot robot should always be walking on its feet, not its knees!

New contacts are maintained in an array of ground contact points, with each contact recording the contact position, link number and the link end which caused the contact.

Figure C-14 shows the system determining a new contact point for a five link "Legbot" robot. The tail of link 3 has just penetrated the ground:

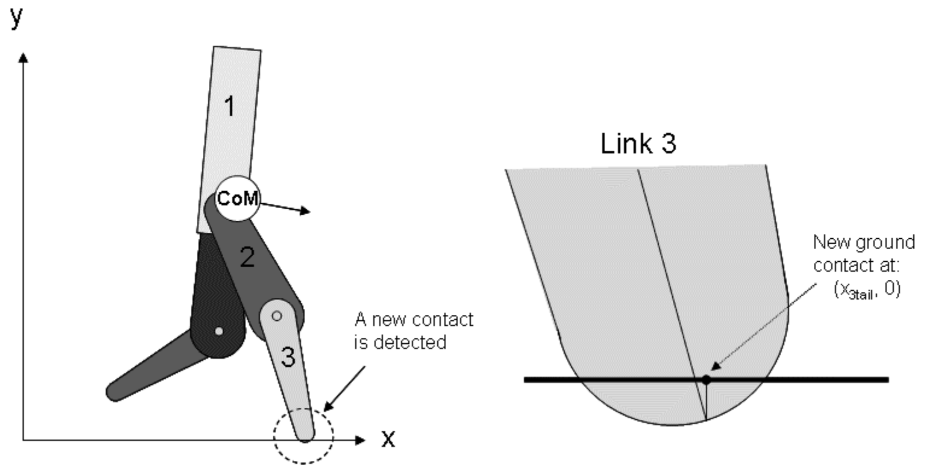


**Figure C-14 A new ground contact is detected between the "tail" of link 3 and the ground.**

ii. Check for lost contacts

If any link end effector was previously penetrating the ground, but is not any longer, then the corresponding ground contact point is removed.

iii. Determine current penetration of each contact point

For the remaining ground contacts, the penetration is a vector from the ground contact point, and the current location of the contact link's end effector:

$$\tilde{p} = \begin{bmatrix} x_{link} - x_{contact} \\ y_{link} - y_{contact} \end{bmatrix} \tag{4}$$

iv. Calculate penalty force acting at each contact point

The penalty force acting at each ground contact point is calculated using a "spring/damper" model, implemented by the following function:

$$\tilde{f}_P(k) = -\left( K_{elastic}\, \tilde{p}(k) + K_{stiff} \frac{\Delta\tilde{p}(k)}{T} \right) \tag{5}$$

Values for the elastic and stiffness coefficients can be adjusted to model the conditions of the surface material.

**Table C-5 Collision model variables**

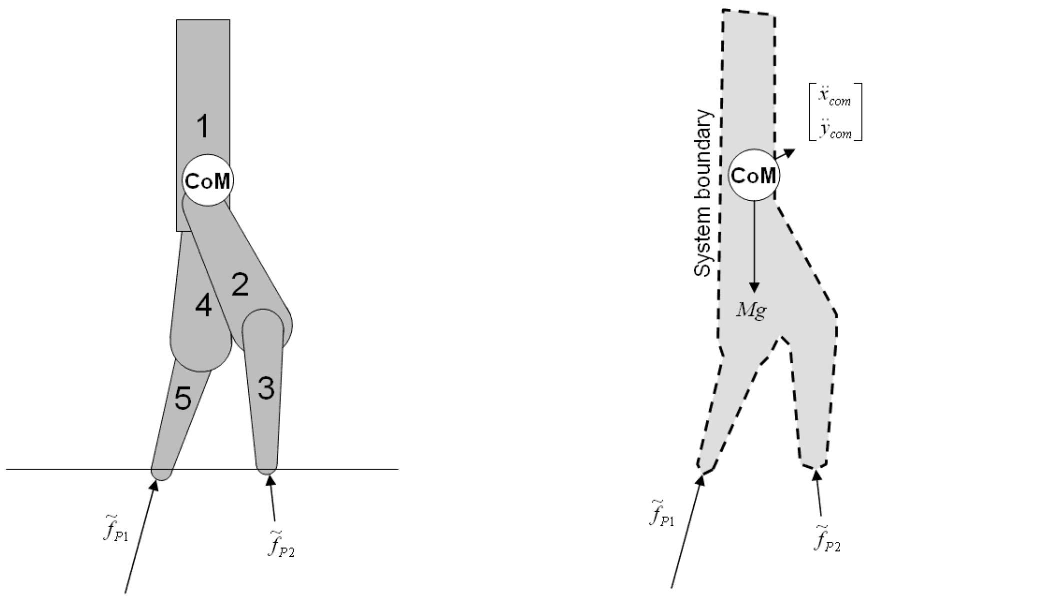
| | | | |
|---|---|---|---|
| $\tilde{p}(k)$ | Penetration vector of a contact link's end effector with the ground. | $\Delta\tilde{p}(k)$ | Change in the penetration vector since the last model update. |
| $x_{link}$ | 'x' position of the end effector of the link in contact with the ground. | $x_{contact}$ | 'x' position of the ground contact point. |
| $y_{link}$ | 'y' position of the end effector of the link in contact with the ground. | $y_{contact}$ | 'y' position of the ground contact point (= ground level). |
| $\tilde{f}_P(k)$ | Force acting on the robot link, as a result of ground contact. | $T$ | Model's iteration interval: (0.0001s) |
| $K_{elastic}$ | Elastic coefficient of the spring/damper system model. | $K_{stiff}$ | Stiffness coefficient of the spring/damper system model. |

## C-4.3 Ballistic model

Since the general *n*-link planar robots being modelled by this simulator are not constrained to slide along the ground as the Ballybot does, the system uses a ballistic model to track the robot's position in the environment.

The objective of the Ballistic model is to determine the accelerations acting on the robot's centre of mass, so the centre of mass position can be updated by the dynamic model using equation (3).

This acceleration can be easily determined by considering the entire robot as a single system [40], and examining all external forces acting on the robot, as shown by Figure C-15:



**(a) Legbot with two ground contacts** **(b) System boundary identifies external forces**

**Figure C-15 The acceleration experienced by a system's centre of mass is influenced only by external forces acting on the system. Interactions between components within the system boundary do not need to be considered.**

Balancing forces in the x and y directions:

$$\sum_x Force = M\,\ddot{x}_{com}$$

$$\ddot{x}_{com} = \frac{\sum_x Forces}{M} \tag{6}$$

$$\sum_y Force = M\,\ddot{y}_{com}$$

$$\ddot{y}_{com} = \frac{\sum_{y} Forces}{M} \quad (7)$$

The only external forces acting on the simulated robot are gravitational force and the ground contact penalty forces. Using ground contact forces supplied by the collision model (0), equations (6) and (7) can be rewritten:

$$\begin{bmatrix} \ddot{x}_{com}(k) \\ \ddot{y}_{com}(k) \end{bmatrix} = \frac{1}{M}\sum_{i=1,m} \tilde{f}_{Pi}(k) - \begin{bmatrix} 0 \\ g \end{bmatrix} \quad (8)$$

**Table C-6 Ballistic model variables**

| | | | |
|---|---|---|---|
| $\ddot{x}_{com}(k)$ | 'x' component of acceleration of the centre of mass. | $g$ | Gravitational acceleration acting on the robot. |
| $\ddot{y}_{com}(k)$ | 'y' component of acceleration of the centre of mass. | $M$ | Total mass of the robot. |
| $\tilde{f}_{Pi}(k)$ | $i^{th}$ ground contact penalty force. | | |

## C-4.4 Link/Torque model

The objective of the "Link/Torque" model is to determine angular accelerations of each of the robot links, given a known set of applied joint torques and end effector forces. A simplified system model of the five-link Legbot robot is shown in Figure C-16. Note that the algorithm developed here is applicable to more general *n*-link planar robots.

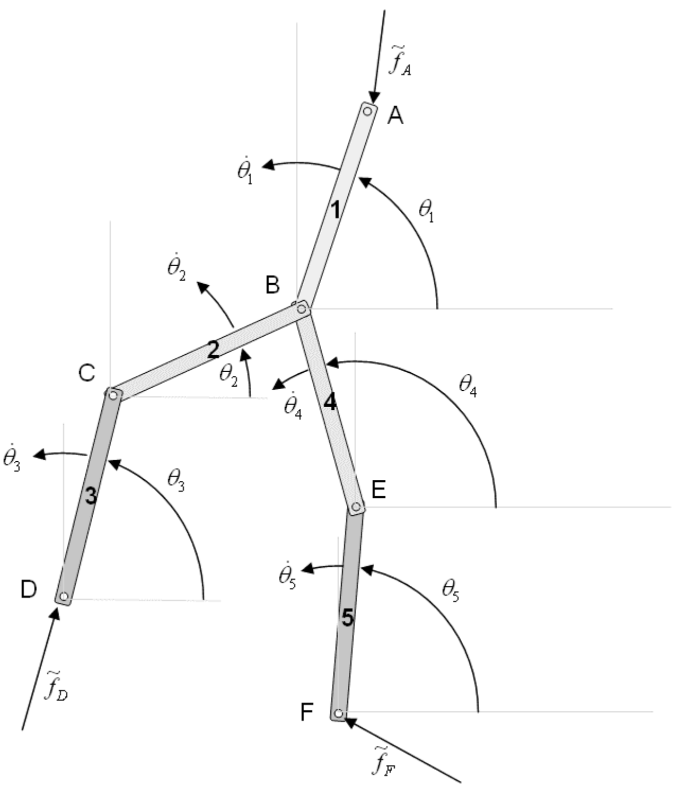


**Figure C-16 System model for a five-link planar robot. Links are numbered 1-5, joints are lettered A-F. Link angles are shown with reference to the world coordinate system. Link mass, centre of mass and joint torque details have been omitted for clarity.**

The link/torque dynamic model uses a variation of the Newton-Euler iterative equations to determine a set of linear equations relating link angular accelerations to joint torque. For a robot consisting of *n* links, the algorithm will determine *n*+2 linear equations with *n*+2 unknown variables. Since all joint torques are known, the equations can be solved to find angular accelerations for all *n* links. A good introduction to the Newton-Euler equations is given by Craig [41].

Some specific situations the model needs to support include:

- No fixed base frame – the robot has no fixed base link position.
- Branching link chains – the robot has to handle situations where more than two links are joined at one point. For example, at the hip joint of a biped.

- Determine angular acceleration from known applied torque – most Newton-Euler applications are used to determine required torque from desired acceleration.

The link/torque algorithm I have implemented performs the following steps at each simulation update:

1. Select a "base" link end effector.
2. Iteratively calculate linear accelerations, moving out from the robot base:
    a. Linear acceleration of each link's centre of mass.
    b. Linear acceleration of the far end of each link.
3. Iteratively propagate forces and torque back down the links.
4. Solve the resulting system of equations for link angular accelerations.

The output of the Link/Torque model is an array of link angular accelerations that, taken together with the centre of mass accelerations calculated by the ballistic model (C-4.3), completely describe the system state acceleration vector: $\tilde{u}(k)$.

i. Select a "base" link end effector.

One advantage of utilising a ballistic model to take into account gravitational acceleration on the robot's centre of mass, and a "penalty" based collision model to determine ground reaction forces, is that the link/torque model can treat the robot system as a collection of links hanging in zero gravity. Any external ground reaction forces induced by gravity have already been accounted for, as have externally imposed constraints on robot positions. The link/torque model only needs to consider the robot link's influence on each other when determining the relative motion of each link.

Since the model only needs to consider the robot link's relative motions, selecting a "*base*" link end effector is an arbitrary decision. Any of the robot's end effectors can be selected by the algorithm with the same result. For consistency, the simulation always chooses the head of the first link of the robot tree to be the "*base*" link. For the purposes of the Newton-Euler analysis, the end effector chosen to be the "*base*" is considered to be the zero point of the frame of reference used to conduct the Newton-Euler analysis.

This arbitrary selection of a base frame of reference does not cause complications for the *n*-link planar robot model, since all that the algorithm needs to determine is the link angular accelerations. Since the orientation of the Newton-Euler frame of reference is the same as the world coordinate system used by the rest of the dynamic model, the values of angular acceleration are identical in both coordinate systems.

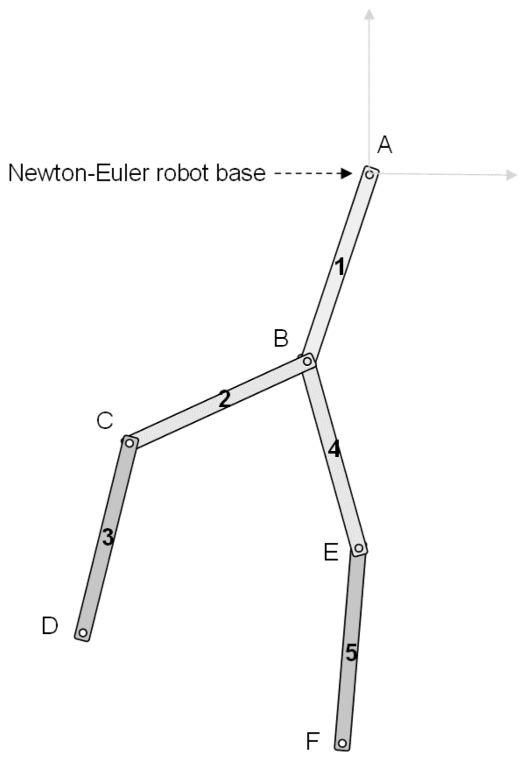


**Figure C-17 The Link/Torque model selects point 'A' as the Newton-Euler robot base, simply because the head of the first link is a convenient place to begin. Points 'D' and 'F' would have been equally valid choices, and selecting them would not have changed the algorithm result.**

In Figure C-17, point “A”, the head of link 1, is chosen to be the “*base*” of the 5-link planar robot. Points “D” or “F” would have been equally valid choices.

For the remainder of this analysis, link orientations may be described in terms of link “*base*” and link “*end*”:

- The “*base*” of an individual link is the joint on the link attached to the robot base, sometimes by a chain of intermediate links.
- The “*end*” of a link is the opposite end of the link to its “*base*”.

For example, in the robot depicted in Figure C-17, the *base* of link 2 is point “B”, while the *end* is point “C”.

ii. Iteratively calculate linear accelerations (relative to the robot centre of mass):

The next stage of the Link/Torque model is to iteratively calculate the accelerations of each link’s centre of mass and *end* point, with respect to the robot centre of mass.

Consider the first link of a general *n*-link planar robot, shown in Figure C-18:

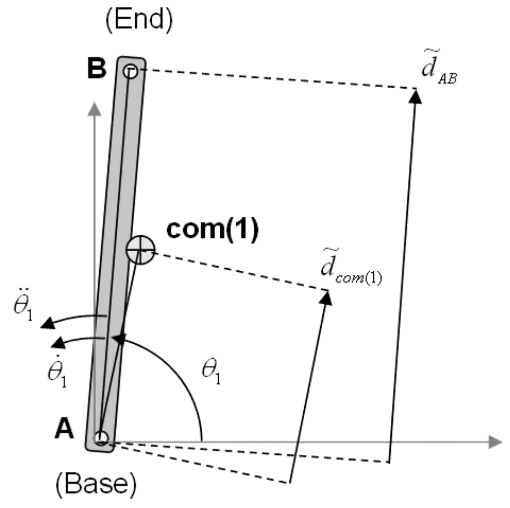


**Figure C-18 The "base" link of the robot. Note that the centre of mass of a robot link does not necessarily like on the line AB.**

**Table C-7 System variables for link 1:**

| | | | |
|---|---|---|---|
| $\tilde{a}_{com(1)}$ | Translational acceleration of the first link's centre of mass. | $\theta_1$ | Real-world angle of the link 1. |
| $\tilde{d}_{com(1)}$ | Displacement vector from A to the centre of mass of link 1. **Note that the centre of mass may not lie on the chord AB.** | $\dot{\theta}_1$ | Angular velocity of the link 1. |
| $\tilde{d}_1$ | Displacement vector from A to B. | $\ddot{\theta}_1$ | Angular acceleration of the link 1. |

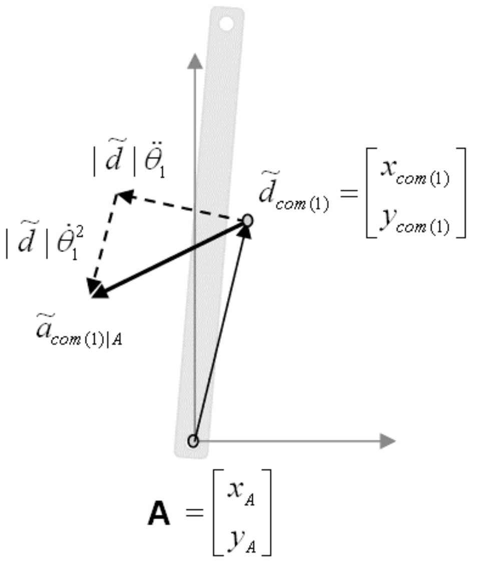
The linear acceleration experienced by the link's centre of mass can be described by the relationship:

$$\tilde{a}_{com(1)} = \tilde{a}_A + \tilde{a}_{com(1)\,|\,A} \tag{9}$$

Point "A" is the *base* of the robot, and it is accelerating at an unknown rate, with respect to the robot's centre of mass:

$$\tilde{a}_A = \begin{bmatrix} \ddot{x}_A \\ \ddot{y}_A \end{bmatrix} \text{ ms}^{-2} \tag{10}$$

$\tilde{a}_{com(1)|A}$ is the acceleration experienced by the link's centre of mass, relative to the robot "*base*". It can be determined by analysing the acceleration diagram in Figure C-19.



$|\tilde{d}|\ddot{\theta}_1$
$|\tilde{d}|\dot{\theta}_1^2$
$\tilde{d}_{com(1)} = \begin{bmatrix} x_{com(1)} \\ y_{com(1)} \end{bmatrix}$
$\tilde{a}_{com(1)|A}$
$\mathbf{A} = \begin{bmatrix} x_A \\ y_A \end{bmatrix}$

**Figure C-19 Acceleration diagram, showing relative acceleration of the centre of mass, with respect to the link base point "A".**

The components of link 1 centre of mass acceleration making up the vector $\tilde{a}_{com(1)|A}$ are:

$\left|\tilde{d}\right|\dot{\theta}_1^2$: Acting towards "A".

$\left|\tilde{d}\right|\ddot{\theta}_1$: Acting perpendicular to $\tilde{d}_{comAB}$, in the direction of $\ddot{\theta}_1$.

A geometric analysis of the diagram shown by Figure C-19 results in the following relationship describing $\tilde{a}_{com(1)|A}$:

$$\tilde{a}_{com(1)|A} = \begin{bmatrix} (y_A - y_{com(1)}) & \dot{\theta}_1^2(x_A - x_{com(1)}) \\ (x_{com(1)} - x_A) & \dot{\theta}_1^2(y_A - y_{com(1)}) \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ 1 \end{bmatrix}$$

From equations (9) and (10):

$$\tilde{a}_{com(1)} = \begin{bmatrix} \ddot{x}_A \\ \ddot{y}_A \end{bmatrix} + \begin{bmatrix} (y_A - y_{com(1)}) & \dot{\theta}_1^2(x_A - x_{com(1)}) \\ (x_{com(1)} - x_A) & \dot{\theta}_1^2(y_A - y_{com(1)}) \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ 1 \end{bmatrix} \qquad (11)$$

Since all the variables in Equation (11) are known, with the exception of $\ddot{\theta}_1$, $\ddot{x}_A$ and $\ddot{y}_A$, the equation can be rewritten in the form:

$$\tilde{a}_{com(1)} = C_1 \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \vdots \\ \ddot{\theta}_n \\ \ddot{x}_A \\ \ddot{y}_A \\ 1 \end{bmatrix} \tag{12}$$

Or, more compactly:

$$\tilde{a}_{com(1)} = C_1 \, \tilde{\ddot{\theta}} \tag{13}$$

Where $C_1$ is a $2 \times (n+3)$ matrix of scalar values. For a five link robot like the Legbot:

$$C_1 = \begin{bmatrix} (y_A - y_{com(1)}) & 0 & 0 & 0 & 0 & 1 & 0 & \dot{\theta}_1^2 (x_A - x_{com(1)}) \\ (x_{com(1)} - x_A) & 0 & 0 & 0 & 0 & 0 & 1 & \dot{\theta}_1^2 (y_A - y_{com(1)}) \end{bmatrix}$$

Using the same process, the relative acceleration of the end of the first link (point "B") can be expressed in the form:

$$\tilde{a}_B = A_1 \, \tilde{\ddot{\theta}} \tag{14}$$

Where $A_1$ is another $2 \times (n+3)$ matrix. For the first link of a five link robot:

$$A_1 = \begin{bmatrix} (y_A - y_B) & 0 & 0 & 0 & 0 & 1 & 0 & \dot{\theta}_1^2 (x_A - x_B) \\ (x_B - x_A) & 0 & 0 & 0 & 0 & 0 & 1 & \dot{\theta}_1^2 (y_A - y_B) \end{bmatrix} \tag{15}$$

All variables making up the elements of the acceleration matrices $A_1$ and $C_1$ are completely known by the software simulation. The only unknown variables in the two equations (13) and (14) are the acceleration vectors: $\tilde{a}_{com(1)}$ and $\tilde{a}_B$, and the vector $\tilde{\ddot{\theta}}$.

This process of determining acceleration matrices for the link centre of mass, and link end points, can be iteratively repeated through the entire robot structure. For example, consider the second link in the robot "chain", depicted in Figure C-20:

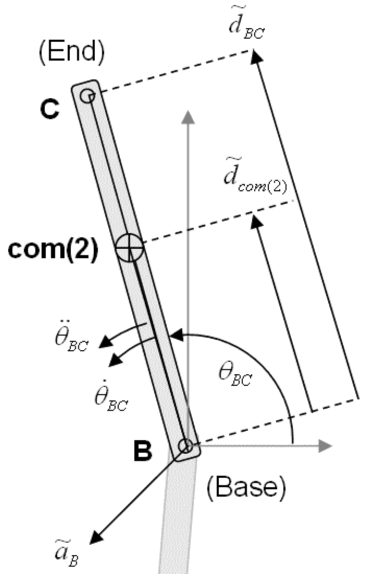


**Figure C-20 The second link in the robot chain. The acceleration of point B has already been calculated.**

$$\tilde{a}_{com(2)} = \tilde{a}_B + \tilde{a}_{com(2)|B} \tag{16}$$

$$\tilde{a}_{com(2)} = A_1\,\tilde{\ddot{\theta}} + \tilde{a}_{com(2)|B}$$

The acceleration of the centre of mass of link 2, with respect to the *base* of link 2 (point "B") is calculated in the same manner as in link 1, resulting in the following equation for a five link robot:

$$\tilde{a}_{com(2)\,|\,B} = \begin{bmatrix} 0 & (y_B - y_{com(2)}) & 0 & 0 & 0 & 0 & 0 & \dot{\theta}_2^2(x_B - x_{com(2)}) \\ 0 & (x_{com(2)} - x_B) & 0 & 0 & 0 & 0 & 0 & \dot{\theta}_2^2(y_B - y_{com(2)}) \end{bmatrix} \tilde{\ddot{\theta}} \qquad (17)$$

Substituting (17) and (15)$\rightarrow$ (16):

$$\tilde{a}_{com(2)} = \begin{bmatrix} (y_A - y_B) & (y_B - y_{com(2)}) & 0 & 0 & 0 & 1 & 0 & \dot{\theta}_1^2(x_A - x_B) + \dot{\theta}_2^2(x_B - x_{com(2)}) \\ (x_B - x_A) & (x_{com(2)} - x_B) & 0 & 0 & 0 & 0 & 1 & \dot{\theta}_1^2(y_A - y_B) + \dot{\theta}_2^2(y_B - y_{com(2)}) \end{bmatrix} \tilde{\ddot{\theta}}$$

Therefore the acceleration of the centre of mass of link 2 can be expressed in the form:

$$\tilde{a}_{com(2)} = C_2\, \tilde{\ddot{\theta}} \qquad (18)$$

Repeating the process to find the acceleration of the *end* of link 2:

$$\tilde{a}_C = \tilde{a}_B + \tilde{a}_{C\,|\,B} \qquad (19)$$

Results in an equation of the form:

$$\tilde{a}_C = A_2\, \tilde{\ddot{\theta}} \qquad (20)$$

This process of iteratively propagating joint accelerations along the robot link chain results in a collection of *n* matrix equations, one for each robot link. Each equation relates a vector of link angular accelerations (and *base* joint acceleration), with the accelerations of the centre of mass, and the far *end* of each link of the robot. The equations are:

Acceleration of the *i* th link's centre of mass:

$$\tilde{a}_{com(i)} = C_i\, \tilde{\ddot{\theta}} \qquad (21)$$

Acceleration of the "*end*" of the *i*th link:

$$\tilde{a}_i = A_i \, \tilde{\ddot{\theta}} \tag{22}$$

Where:

| | |
|---|---|
| $\tilde{a}_i$ | The acceleration vector of the *end* of the $i$th link in the robot chain, with respect to the robot's centre of mass. |
| $\tilde{a}_{com(i)}$ | The acceleration vector of the $i$th link's centre of mass, with respect to the robot's centre of mass. |
| $C_i$ | ($2 \times n+3$) matrix of known values, relating the $n+2$ unknown elements of $\tilde{\ddot{\theta}}$ to the linear acceleration experienced by the $i$th link's centre of mass. |
| $A_i$ | ($2 \times n+3$) matrix of known values, relating the $n+2$ unknown elements of $\tilde{\ddot{\theta}}$ to the linear acceleration experienced by the far *end* of the $i$th link. |
| $\tilde{\ddot{\theta}} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \vdots \\ \ddot{\theta}_n \\ \ddot{x}_A \\ \ddot{y}_A \\ 1 \end{bmatrix}$ | ($n$+3) element vector, including all ($n$) link angular accelerations as well as the x and y components of the acceleration of the robot *base* (with respect to the robot's centre of mass). |

iii. Iteratively propagate forces and torque back down the links

In step (ii) the linear acceleration of each link's centre of mass was expressed as a function of a vector of link angular accelerations. The next stage of the Link/Torque model is to iteratively propagate externally applied forces and internally applied torques back down the robot link chains.

The process begins with one of the end effectors of the robot link "tree". In Figure C-21, the link end chosen to be the *base* of the robot was point "A". That means the force propagation can begin with either points "D" or "F". In this example, I have chosen to begin by examining point "D", on link 3:

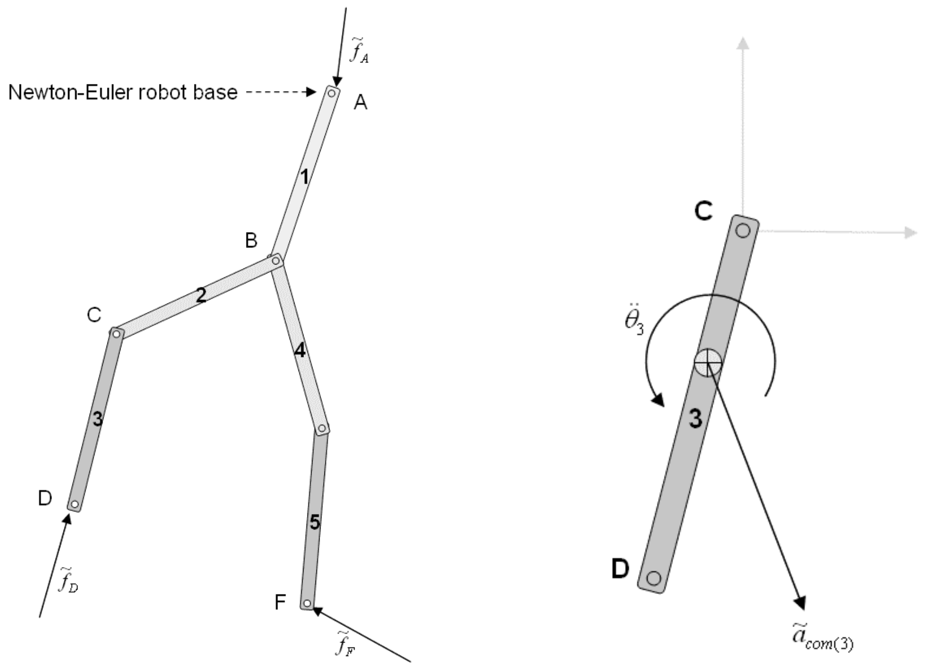


**Figure C-21 Since point "A" was selected as the base of the robot, either point "D" or "F" could be chosen to be the start of force propagation. After propagating accelerations through all robot links, we can express the link's acceleration as an angular and linear acceleration through the link centre of mass.**

Figure C-21 shows the linear and angular accelerations experienced by the centre of mass of link 3. These can be expressed in terms of link angular acceleration by the matrix equations determined in step (ii):

$$\tilde{a}_{com(3)} = C_3\, \tilde{\ddot{\theta}} \tag{23}$$

$$\ddot{\theta}_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tilde{\ddot{\theta}} \tag{24}$$

From the "observed" angular and translational accelerations experienced by the robot link, the resultant forces and torques acting on the link can be inferred:

$$\tilde{f}_{com(3)} = m_3 \tilde{a}_{com(3)}$$

$$\tilde{f}_{com(3)} = m_3 C_3\, \tilde{\ddot{\theta}} \tag{25}$$

$$\tau_{com(3)} = I_3\ddot{\theta}_3 \qquad (26)$$

These forces and torques must be the resultant of force and torque being applied to each end of the link, since no other forces can act on the link. Figure C-22 shows the relationship between the resultant force and torque acting through the centre of mass, and components acting at the link ends:

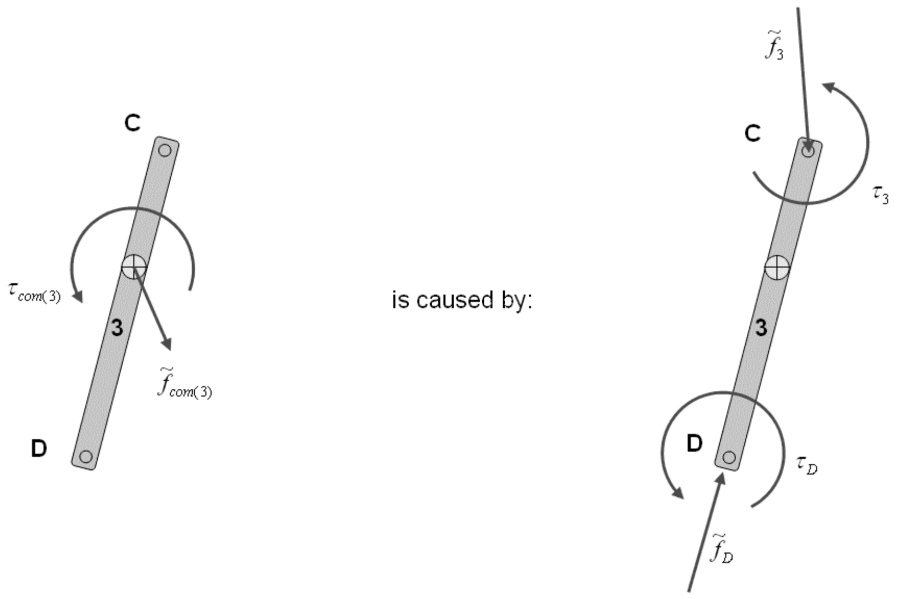


**Figure C-22 The resultant force and torque appearing to act on the centre of mass of a link are a produced by linear force and torque acting on each end of the link.**

Where:

| | | | |
|---|---|---|---|
| $\tilde{f}_{com(3)}$ | Resultant force appearing to act through the centre of mass of link 3. | $\tau_{com(3)}$ | Resultant torque appearing to act about the centre of mass of link 3. |
| $\tilde{f}_D$ | External force being applied to the *end* of link 3 at point "D". | $\tau_D$ | External torque being applied to the *end* of link 3 at point "D". |
| $\tilde{f}_3$ | Force being applied to the *base* of link 3. | $\tau_3$ | External torque being applied to the *base* of link 3 at point "C". |

Considering first the translational forces:

$$\tilde{f}_{com(3)} = \tilde{f}_D + \tilde{f}_3 \tag{27}$$

If the point "D" is in ground contact, the force $\tilde{f}_D$ will have been determined by the collision model, 0. Otherwise the point is airborne, and $\tilde{f}_D = 0$.

From (27) and (25):

$$\tilde{f}_3 = \tilde{f}_D - m_3 C_3 \ddot{\tilde{\theta}}$$

$$\tilde{f}_3 = \begin{bmatrix} 0 & \cdots & 0 & \tilde{f}_{Dx} \\ 0 & \cdots & 0 & \tilde{f}_{Dy} \end{bmatrix} \ddot{\tilde{\theta}} - m_3 C_3 \ddot{\tilde{\theta}} \tag{28}$$

Equation (28) can be rearranged to express $\tilde{f}_3$ as a matrix (of known element values), multiplied by $\ddot{\tilde{\theta}}$ :

$$\tilde{f}_3 = F_3 \ddot{\tilde{\theta}} \tag{29}$$

Next, the free body diagram of link 3, shown in Figure C-23, is used to determine the effects of the linear forces on the resultant torque acting through the link centre of mass:

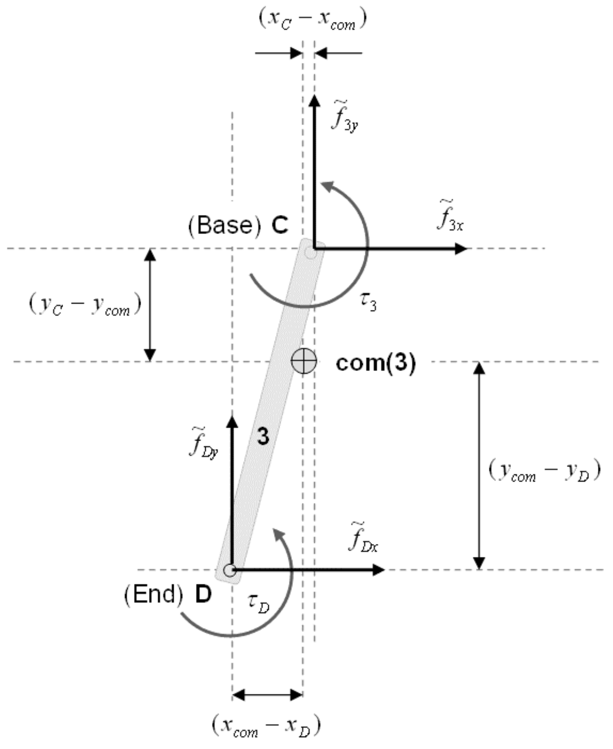


**Figure C-23 Free body diagram used to calculate resultant torque generated by translational forces and joint torque acting on the base and end of link 3.**

$$\tau_{com(3)} = \sum \tau$$

$$\tau_{com(3)} = \tau_D + \tau_3 + \tilde{f}_{Dy}(x_{com} - x_D) + \tilde{f}_{3x}(y_C - y_{com}) - \tilde{f}_{Dx}(y_{com} - y_D) - \tilde{f}_{3y}(x_C - x_{com}) \tag{30}$$

Substituting (26) $\rightarrow$ (30):

$$I_3\ddot{\theta}_3 = \tau_D + \tau_3 + \tilde{f}_{Dy}(x_{com} - x_D) + \tilde{f}_{3x}(y_C - y_{com}) - \tilde{f}_{Dx}(y_{com} - y_D) - \tilde{f}_{3y}(x_C - x_{com}) \tag{31}$$

Using (29), equation (31) can be rearranged to the form:

$$\tau_3 = \tilde{c}_3\ \tilde{\ddot{\theta}} \tag{32}$$

Where:

| | | | |
|---|---|---|---|
| $\tau_{com(3)}$ | Resultant force appearing to act through the centre of mass of link 3. | $\tau_D$ | External torque being applied to the *end* of link 3 (at point "D"). **Note: for link 3, this torque is zero.** |
| $\tau_3$ | Torque being applied to the *base* of link 3. | $\tilde{c}_3$ | Vector of known ($n$+3) elements, relating $\tau_3$ to $\tilde{\ddot{\theta}}$. |
| $\tilde{f}_3$ | Force acting on the *base* of link 3. | $\tilde{f}_D$ | Force acting on the *end* of link 3. |

This process can now be repeated for the next link, moving inwards towards the robot *base*, chosen in step (i). The free body diagram for link 2 is shown in Figure C-24:

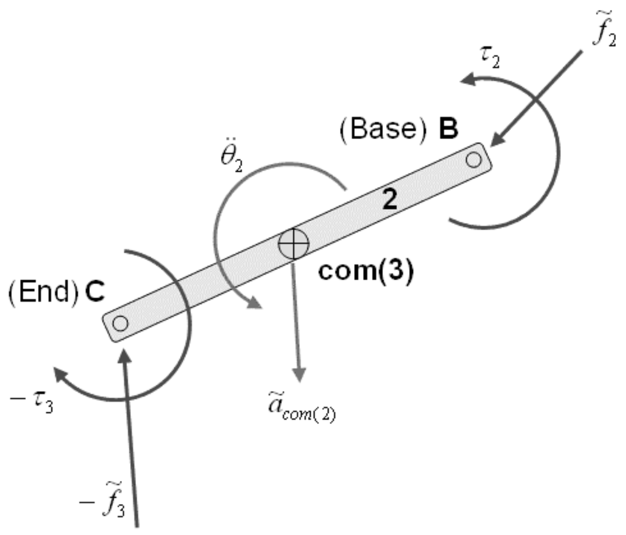


**Figure C-24 Free body diagram for link "2". Torque and force acting at point "C" appear to work in the opposite direction to that experienced by link 3.**

Using the same process outlined for link 3, force and torque acting on the base of link 2 can be expressed by equations (33) and (34).

Force acting on the *base* of link 2:

$$\tilde{f}_2 = F_2\,\tilde{\ddot{\theta}} \tag{33}$$

Torque applied to base of link 2:

$$\tau_2 = \tilde{c}_2 \, \tilde{\ddot{\theta}} \qquad (34)$$

This process of "inward iteration" can be continued all the way back to the base of the robot system, with one small variation for the case where multiple branches of the robot link tree converge. In this situation, the iterative inward calculations of force and torque must be performed on each branch of the tree of robot links. When the branches converge, the force and torque acting on the "end" of the parent link is the sum of the force and torque propagated back down each branch. Figure C-25 shows the forces and torque acting on the hip joint of a five link robot:

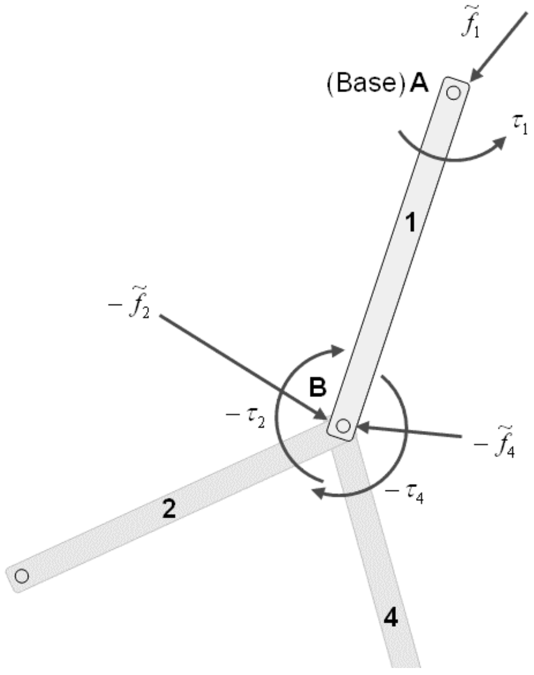


**Figure C-25 Force and torque acting on an inward link shared by two branches is the sum of the forces and torques propagated back from all "children" branches.**

After completing the inward iterations of force and torque calculations, forces and torques acting on the "*base*" of each link in the robot can be represented by the set of matrix calculations:

$$\tilde{f}_i = F_i \, \tilde{\ddot{\theta}} \tag{35}$$

$$\tau_i = \tilde{c}_i \, \tilde{\ddot{\theta}} \tag{36}$$

iv. Solve resulting equations for link angular accelerations

In step (iii), the system model calculated a set of linear and matrix equations for each link, expressing torque at the *base* of the link, and force applied to the *base* of the link, as a function of a vector of unknown variables: $\tilde{\ddot{\theta}}$.

For an *n*-link robot, the system has determined *n* equations relating joint torque to $\tilde{\ddot{\theta}}$:

$$\tau_i = \tilde{c}_i \, \tilde{\ddot{\theta}}$$

An additional two equations can be determined by examining equations describing the force being applied to the robot base:

$$\tilde{f}_1 = F_1 \, \tilde{\ddot{\theta}}$$

The force $\tilde{f}_1$ is known to the system model: If the *base* is in the air, the force is zero, otherwise the force is supplied by the system's collision model (0).

Since the system knows *n+2* linear equations, with *n+2* unknown variables, the equations can be solved to determine all the elements of the vector $\tilde{\ddot{\theta}}$.

I used matrix row reduction to solve the system of equations, and determine joint angular accelerations for each joint.

v. Equation summary

Outwards iterations to find accelerations:

- Acceleration of the $i$ th link's centre of mass:

$$\tilde{a}_{com(i)} = C_i \, \ddot{\tilde{\theta}} \tag{21}$$

- Acceleration of the *end* of the $i$th link:

$$\tilde{a}_i = A_i \, \ddot{\tilde{\theta}} \tag{22}$$

Inward iterations to find force and torque:

- Torque applied to *base* of link $i$:

$$\tau_i = \tilde{c}_i \, \ddot{\tilde{\theta}} \tag{36}$$

- Force acting on the *base* of link $i$:

$$\tilde{f}_i = F_i \, \ddot{\tilde{\theta}} \tag{29}$$

Using the two force equations for x and y components of external force applied to the base of the robot, and the $n$ torque equations, solve the system of $n$+2 linear equations to determine the values of the $n$+2 variables making up the vector $\ddot{\tilde{\theta}}$.

The variables used in this equation summary are summarised in Table C-8:

**Table C-8 Variables used in Newton-Euler algorithm.**

| | |
|---|---|
| $\tilde{\ddot{\theta}} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \vdots \\ \ddot{\theta}_n \\ \ddot{x}_A \\ \ddot{y}_A \\ 1 \end{bmatrix}$ | ($n$+3) element vector, including all ($n$) link angular accelerations as well as the x and y components of the acceleration of the robot *base* (with respect to the robot's centre of mass). |
| $\tilde{a}_i$ | The acceleration vector of the *end* of the *i*th link in the robot chain, with respect to the robot's centre of mass. |
| $\tilde{a}_{com(i)}$ | The acceleration vector of the *i*th link's centre of mass, with respect to the robot's centre of mass. |
| $C_i$ | (2 × $n$+3) matrix of known values, relating the $n$+2 unknown elements of $\tilde{\ddot{\theta}}$ to the linear acceleration experienced by the *i*th link's centre of mass. |
| $A_i$ | (2 × $n$+3) matrix of known values, relating the $n$+2 unknown elements of $\tilde{\ddot{\theta}}$ to the linear acceleration experienced by the far *end* of the *i*th link. |
| $\tau_i$ | Torque being applied to the *base* of link *i*. |
| $\tilde{c}_i$ | Vector of ($n$+3) elements, relating $\tau_i$ to $\tilde{\ddot{\theta}}$. |
| $\tilde{f}_3$ | Force acting on the *base* of link *i*. |
| $F_i$ | (2 × $n$+3) matrix of known values, relating the $n$+2 unknown elements of $\tilde{\ddot{\theta}}$ to force acting on the *base* of the *i*th link. |

## Bibliography

[1] Pepper, J., “Walking Algorithms for Biped Robot”, *EE Honours Dissertation*, 1999, University of Western Australia.

[2] Nicholls, E., “Bipedal Dynamic Walking in Robotics”, *EE Honours Dissertation*, 1998, University of Western Australia.

[3] Ng, Joon., “An Anthropomorphic Bipedal Robot”, *EE Honours Dissertation*, 1998, University of Western Australia.

[4] Wenzel, L., Vazquez, N., Jamal, ND. N. “Computer vision based inverted pendulum”, *Proc. of IEEE Conference on Instrumentation and Measurement Technology*, pp. 1319-1323, vol. 3, 2000.

[5] Magana, M. E., Holzapfel, F., “Fuzzy logic control of an inverted pendulum with vision feedback”, *IEEE Trans. On Education*, pp. 165-170, vol. 41, no. 2, 1998.

[6] Tolat, V., Widrow, B., “An adaptive ‘broom balancer’ with visual inputs”, *IEEE International Conference on Neural Networks*, pp. 641-647, vol. 2, San Diego, 1988.

[7] Kajita,S.;Tani,K., “Experimental Study of Biped Dynamic Walking in the Linear Inverted Pendulum Mode”, *IEEE Control Systems Magazine,* Volume: 16, Issue: 1, Feb. 1996, pp. 13-19.

[8] Ogasawara, K.; Kawaji, S., "Cooperative motion control for biped locomotion robots", *Systems, Man, and Cybernetics, 1999. IEEE SMC’99 Conference Proceedings, 1999 IEEE Int. Conference on,* Volume: 4, 1999, pp. 966-971 vol.4.

[9] Park, J. Kim,K.D., “Bipedal Robot Walking Using Gravity-Compensated Inverted Pendulum Mode and Computed Torque Control”, *Robotics and Automation, 1998. Proceedings, 1998 IEEE Int. Conference on,* Volume: 4, 1998, pp. 3528-3533 vol.4.

[10] Caux, S.; Mateo, E.; Zapata, R., "Balance of biped robots: special double-inverted pendulum", *Systems, Man, and Cybernetics, 1998. 1998 IEEE Int. Conference on,* Volume: 4, 1998, pp. 3691-3696 vol.4.

[11] Williams, V.; Matsuoka, K., “Learning to Balance the Inverted Pendulum Using Neural Networks”, *Neural Networks, 1991 IEEE Int. Joint Conference on,* Volume: 1, 1991, pp. 214-219.

[12] Donaldson, P., “Error decorrelation: a technique for matching a class of functions”, *Proceedings: III Int. Conf. On Medical Electronics, 1960*, pp. 173-178.

[13] Anderson, C., “Learning to Control an Inverted Pendulum Using Neural Networks”, *IEEE Control Systems Magazine, April, 1989.*

[14] Widrow, B., Smith, F. W., “Pattern-Recognizing Control Systems”, *1963 Computer and Information Sciences (COINS) Symp.Proc.,* Washington, DC: Spartan, pp. 288-317, 1964

[15] Widrow, B., “The Original Adaptive Neural Net Broom-Balancer”, *Int. Symp. Circuits and Syst.,* pp. 351-357, May, 1987.

[16] Guez, A., Selinsky, J., “A Trainable Neuromorphic Controller”, *J. Robotic Systems.,* vol. 5, no. 4, pp. 363-388, Aug. 1988.

[17] Connell, M.E., Utgoff, P.E., “Learning to Control a Dynamic Physical System”, *Proc. AAA1-87.,* vol. 2, pp. 456-460, American Association for Artificial Intelligence, Seattle, WA, 1987.

[18] Hougen, D.F., Fischer, J., Johnam, D., “A neural network pole balancer that learns and operates on a real robot in real time”, *Proceedings of the MLC-COLT Workshop on Robot Learning*, pp.73-80, 1994.

[19] Huang, Shiuh-Jer.; Huang, Chien-Lo., “Control of an Inverted Pendulum Using Grey Prediction Model”, *IEEE Transactions on Industry Applications*, vol. 36, no. 2, pp.452-458, March/April 2000.

[20] Ahmed, M.S.; Riyaz, S.H., “Design of dynamic neural observers”, *IEEE Proc.-Control Theory Appl.*, vol. 147, no. 3, pp.257-266, May 2000.

[21] Harnold, Chi-Li-Ma; Kwang, Y.Lee., “Application of the Free-Model Based Neural Networks in Model Reference Adaptive Inverse Control”, *Proceedings of the American Control Conference.,* Chicago, Illinois, pp. 1664-1668, June 2000.

[22] Mendil, B.; Benmahammed, K., “FEP Learning Algorithm : Application to Direct Self-Learning Control”, *Proc. Of the 1999 IEEE Int. Conf. On Control Applications*, Kohala Cost-Island of Hawai’i, USA, August 22-27, 1999, pp.432-435.

[23] D. Michie and R. A. Chambers. “BOXES: An experiment in adaptive control”. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137--152, Edinburgh, UK, 1968.

[24] J. Freeman and D. Skapura., *Neural Networks: Algorithms, Applications and Programming Techniques*, Reading: Addison-Wesley, 1991.

[25] Sutherland, A., “Target Tracking Systems”, *EE Honours Dissertation*, 1993, University of Western Australia.

[26] Ogata, K., *Modern Control Engineering*, Prentice-Hall, 2nd Edition, 1990.

[27] Bräunl, T., *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*, Springer-Verlag, 2003.

[28] Liu, J., “Joint stick-slip friction compensation for robotic manipulators by iterative learning”, *Intelligent Robots and Systems ’94 “Advanced Robotic Systems and the real World”, IROS ’94, Proceedings*, pp.502-509, 1994.

[29] Dupont, P., “Friction Modeling in Dynamic Robot Simulation”, *Robotics and Automation, 1990. Proceedings IEEE Int. Conference on*, pp.1370-1376, 1990.

[30] Fujimoto, Y.; Kawamura, A., "Simulation of an Autonomous Biped Walking Robot Including Environmental Force Interaction", *IEEE Robotics & Automation Magazine,* June, 1998, pp. 33-42.

[31] Huang, Kajita, Koyachi, Kaneko, Yokoi, Arai, Komoriya, Tanie, "A High Stability, Smooth Walking Pattern for a Biped Robot", *Proc. IEEE Int. Conf. Robotics & Automation,* pp. 65-71. (1999)

[32] Goddard, Zheng, Hemami, "Control of heel-off to toe-off motion of a dynamic biped gait", *IEEE Trans. Systems, Man, and Cyber.,* Vol 22, No.1, Jan/Feb 1992, pp.92-102.

[33] SEIKA/NTT, "N2, N3, N4 Inclinometers", *Technical specification document*, URL: http://www.ntt.dk/n2_n4.pdf

[34] HITEC, "GY-130 Piezo Gyro Instructions", *Technical specification document*, URL: http://www.hitecrcd.com/support/Manuals/gyro.pdf

[35] Box, D., *Essential COM*, Addison-Wesley Professional, 1st Edition, 1997.

[36] Roumeliotis, S., Sukhatme, G., Bekey, G., "Smoother based 3D Attitude Estimation for Mobile Robot Localization", *Proc. IEEE Int. Conf. Robotics & Automation,* (1999).

[37] Joker Robotics, *http://www.joker-robotics.com*, 2003.

[38] Burgard, W., Fox, D., Hennig, D., Schmidt, T., "Estimating the absolute position of a mobile robot using position probability grids", *Proc. National Conference on Artificial Intelligence (AAAI)*, pp. 896-901. (1996).

[39] Baraff, D., "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies", *Computer Graphics*, vol. 23, no. 3, pp.223-232, 1989.

[40] Meriam, J., Kraige, L., *Engineering Mechanics, vol. 2, Dynamics*, John Wiley & Sons, 2nd Edition, 1987.

[41] Craig, J., *Introduction to Robotics: Mechanics and Control*, Addison-Wesley, 2nd Edition, 1989.

[42] P.Doerschuk, V.Nguyen, A.Li, "Neural Network Control of a Three-Link Leg," *IEEE Proc. Int. Conf. on Tools with Artificial Intel.*, pp.278-281 (1995)

[43] A.Kun, W.Miller, "Adaptive Dynamic Balance of a Biped Robot Using Neural Networks," *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 240-245 (1996)

[44] P.Doerschuk, V.Nguyen, A.Li, "A Modular Approach to Intelligent Control of a Simulated Jointed Leg," *IEEE Robotics & Automation Magazine*, June 1998, pp.12-21.

[45] I.H.J.Ploemen, M.J.G. van de Molengraft, "Hybrid Modelling for Mechanical Systems: Methodologies and Applications," *Trans. ASME Journal of Dynamic Systems, Measurement, and Control*, June 1999, Vol 121, pp.270-277.

[46] Y Nakamura, Y. Xu, "Geometrical Fusion Method for Multisensor Robotic Systems," *Multisensor Integration & Fusion for Intelligent Machines and Systems*, Ablex Pub. Corp, 1995, Ch11, pp.241-259.

[47] M.-Y. Cheng and C.-S. Lin, "Genetic Algorithm for Control Design of Bipedal Locomotion," *Journal of Robotic Systems* 14(5), 365-373 (1997)

[48] J. Koechling, M. Raibert, "How Fast Can a Legged Robot Run?" *Robots and Biological Systems: Towards a New Bionics?*, NATO ASI Series F: Computer & Systems Sciences, Vol.102, 1993, pp. 239-269.

[49] Dar-Zen Chen, Jioun-Chin Su, Kang-Li Yao, "A Decomposition Approach for the Kinematic Synthesis of Tendon-Driven Manipulators", *Journal of Robotic Systems* 16(8), 433-443 (1999)

[50] A. Takanashi, "Robot Biped Walking Stabilized with Trunk Motion," *Robots and Biological Systems: Towards a New Bionics?*, NATO ASI Series F: Computer & Systems Sciences, Vol.102, 1993, pp.271-291.

[51] Goswami, A., "Foot rotation indicator (FRI) point: A new gait planning tool to evaluate postural stability of biped robots", *Robotics and Automation, 1999. Proceedings, 1999 IEEE Int. Conference on,* Volume: 1, 1999, pp. 47-52 vol.1.

[52] Jong Hyeon Park; Hyun Chul Cho, "An online trajectory modifier for the base link of biped robots to enhance locomotion stability", *Robotics and Automation, 2000. Proceedings, ICRA'00, IEEE Int. Conference on,* Volume: 4, 2000, pp. 3353-3358.

[53] Huang, Q., Kaneko, K., Yokoi, K., Kajita, S., Kotoku, T., Koyachi, N., Arai, H., Imamura, N., Komoriya, K., Tanie, K., "Balance control of a biped robot combining off-line pattern with real-time modification", *Robotics and Automation, 2000. Proceedings, ICRA'00, IEEE Int. Conference on,* Volume: 4, 2000, pp. 3346-3352.

[54] Caux, S.; Zapata, R., "Towards a Unified Form of Biped Robot Dynamic Modeling", *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE Int. Conf. On,* Volume: 4, 1997, pp. 3249-3254.

[55] Raibert, M.H., *Legged Robots that Balance,* MIT Press:Cambridge, 1986.

[56] General Workshop, School of Electrical Electronic and Computer Engineering, The University of Western Australia, http://www.ee.uwa.edu.au/support, 2005.

[57] Donelan, J. M., Kram, R., Kuo, A., "Mechanical work for step-to-step transitions is a major determinant of the metabolic cost of human walking", *Journal of Experimental Biology,* 2002 Dec;205(Pt 23), pp. 3717-3727.

[58] McGeer, T., "Passive walking with knees", *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 1640-1645 (1990)

[59] Lee, C. R., Farley, C. T., "Determinants of the center of mass trajectory in human walking and running", *Journal of Experimental Biology,* 201, 1998, pp. 2935-2944.

[60] McGhee, R. B. and Frank, A. A., “On the Stability Properties of Quadruped Creeping Gaits,” *Mathematical Biosciences*, Vol. 3, No. 3/4, pp. 331-351, October, 1968.

[61] T. Kato, A. Takanishi, and I. Kato, “The realization of the quasi-dynamic walking by the biped walking machine,” in *Fourth Symposium on Theory and Practice of Robots and Manipulators* (A. Morecki, G. Bianchi, and K. Kedzior, eds.), Warsaw, pp. 341-351, Polish Scientific Publishers, 1983.

[62] Vukobratovic, M. and Borovac, B., “Zero-moment point – thirty five years of its life,” *Int. Journal of Humanoid Robots*, Vol. 1, No. 1, pp. 157-173, 2004.

[63] Huang, Q., Kajita, S., Koyachi, N., Kaneko, K., Yokoi, K., Arai, H., Komoriya, K., and Tanie, K., “A high stability, smooth walking pattern for a biped robot”, in *Proc. IEEE Int. Conf. Robotics and Automation*, 1999, pp. 65-71.

[64] Dunn, E. and Howe, R., “Foot placement and velocity control in smooth bipedal walking”, *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 578-583 (1996)

[65] Sutherland, A., Bräunl, T., “An experimental platform for researching robot balance”, *2002 FIRA Robot World Congress,* Seoul, May 2002, pp. 14-19 (6).

[66] Pratt, J., Dilworth, P. and Pratt, G., “Virtual model control of a bipedal walking robot”, *Proc. IEEE Int. Conf. Robotics & Automation*, pp. 193-198 (1997)

[67] Wolff, K., Nordin, P., “An evolutionary based approach for control programming of humanoids”, *Proc. IEEE-RAS.Conf. on humanoid Robots, Humanoids 2003*, (2003)

[68] Wall, M., *GALib: A C++ Library of Genetic Algorithm Components (Version 2.4),* [Online]. Available: http://lancet.mit.edu/ga/dist/galibdoc.pdf, (1996).

[69] Wolff, K., Nordin, P., “Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming.”, in *Proc. Genetic and Evolutionary Computational Conference (GECCO-2003)*, Chicago: Springer-Verlag, 12-16 July 2003, pp. 495-506.

[70] Zhang, R., Vadakkepat, P., “An evolutionary algorithm for trajectory based gait generation of biped robot.”, in *Proc. Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, 2003.

[71] Newton Game Dynamics, [Online]. Available: http://www.newtondynamics.com/, (2006).

[72] Simple DirectMedia Layer (SDL), [Online]. Available: http://www.libsdl.org/, (2006).

[73] Bräunl, T., Sutherland, A., Unkelbach, A., “Dynamic balancing of a humanoid robot”, *FIRA 1st Humanoid Robot Soccor Workshop (HuroSot),* Daejeon, Korea, MRDEC, 1: pp. 43-68 (2002).

[74] Boeing, A., Hanham, S., Bräunl, T., “Evolving autonomous biped control from simulation to reality”, *Int. Conf. on Autonomous Robots and Agents, ICARA 2004,* Palmerston North, New Zealand, pp. 440-445 (2004).

[75] Honda Worldwide | ASIMO, [online]. Available: http://www.world.honda.com/ASIMO/, 2003.

[76] Morimoto, J., Cheng, G., "A simple reinforcement learning algorithm for biped walking", *Proc. IEEE Int. Conf. Robotics and Automation, ICRA'04,* 2004, pp.3030-3035.

[77] Morimoto, J., Nakanishi, J., Endo, G., Cheng, G., "Poincare-map-based reinforcement learning for biped robot", *Proc. IEEE Int. Conf. Robotics and Automation, ICRA'05,* 2005.

[78] Sharon, D., van de Panne, M., "Synthesis of controllers for stylized planar bipedal walking", *Proc. IEEE Int. Conf. Robotics and Automation, ICRA'05,* 2005.

[79] Sutherland, A., Bräunl, T., "Learning to balance an unknown system", *Proc of the IEEE-RAS Intl. Conf. On Humanoid Robots, Humanoids 2001,* Waseda University, Tokyo, Nov. 2001, pp. 385-391 (7).

[80] Pratt, J., "Virtual model control of a biped walking robot", *Masters Thesis*, 1995, Massachusetts Institute of Technology.

[81] Williamson, M., "Series elastic actuators", *Masters Thesis*, 1995, Massachusetts Institute of Technology.

[82] Boeing, A., Bräunl, T., "Evolving Splines: An alternative locomotion controller for a bipedal robot", *Proceedings of the Seventh International Conference on Control, Automation, Robotics and Vision (ICARV 2002)*, CD-ROM, Nanyang Technological University, Singapore, Dec. 2002, pp 1-5 (5)

[83] Long, G., Anderson, J., Borenstein, J., "The kinematic design of the OmniPede: A new approach to obstacle traversion", *Proceedings of the 2002 IEEE Conference on Robotics and Automation (ICRA '02)*, Washington DC, 10-17 May 2002, pp 714-719.

[84] PAL – Physics Abstraction Layer*,* [Online]. Available: http://pal.sourceforge.net/, (2006).

[85] Pratt, J., Chew, C., Torres, A., Dilworith, P., Pratt, G., "Virtual Model Control: An intuitive approach for bipedal locomotion", *International Journal of Robotics Research*, 20:(2) , 2001, pp 129-143.

[86] Da Vinci, L., *Leonardo's Notebooks*, Black Dog and Leventhal Publishers, Inc., editor: H. Anna Suh, 2005.

[87] Shin, D., Yoshigahara, T., Ohteru, S., Takanishi, A., Takeya, T., Kato, I., "Realization of obstacle avoidance by biped walking robot equipped with vision system", *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems '89,* 1989.

[88] Huang, Q., Kajita, S., Koyachi, N., Kaneko, K., Yokoi, K., Kotoku, T., Arai, H., Komoriya, K., Tanie, K., "Walking patterns and actuator specifications for a biped robot", *Proceedings of the 1999 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems,* 1999, pp. 1462-1468.

[89] Baird, L., Klopf, A., “Reinforcement learning with high-dimensional, continuous actions”, *Technical Report WL-TR-93-1147*, Wright-Patterson Air Force Base, Ohio, 1993.

[90] Pendrith, M., “On reinforcement learning of control actions in noisy and non-Markovian domains”, *Technical Report UNSW-CSE-TR-9410*, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia, 1994.

[91] Lendaris, G., Paintz, C., “Training strategies for critic and action neural networks in dual heuristic programming method”, *Proc. Of ICNN’97, Houston, IEEE*, 1997, pp.712-717.

[92] Lee, C., “Tin men”, *Time Magazine*, July 2002.