THESIS REPORT

Master's Degree

Control of Biped Locomotion Using Oscillators

by S. Mishra
Advisor: P.S. Krishnaprasad

M.S. 93-12

The Institute for Systems Research is supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Control Of Biped Locomotion Using Oscillators

Sanjay Mishra

Thesis submitted to the Faculty of The Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1993

Advisory Committee:

Professor Krishnaprasad Chairman/Advisor
Associate Professor Avis Cohen
Associate Professor W. Dayawansa

Abstract

Title of Thesis: Control of Biped Locomotion Using Coupled Oscillators

Name of Degree Candidate: Sanjay Mishra

Degree and Year: Master of Science, 1993

Thesis Directed By: Professor P.S. Krishnaprasad
Department of Electrical Engineering

The seeming ease with which vertebrates move indicates the versatility of biological control systems. Experiments have revealed that control of locomotion is to a large degree localized in the spinal cord. It has been proposed that a Central Pattern Generator consisting of coupled non-linear neural oscillators, residing in the spinal cord, may be responsible for the generation of rhythmic patterns governing locomotion.

The idea that Central Pattern Generators form a natural way of thinking about the control of repetitive activities is presented and the design of Central Pattern Generators investigated. A new design for Central Pattern Generators involving weak, continuous, background coupling between the oscillators, strong pulse coupling between the oscillators and the controlled system and incorporating sensory feedback is presented. Following the work of Mochon and McMahon, a biped is modelled as a four link mechanism and dynamic models developed. The Central Pattern Generators designed are used to control ballistic walking and running. A distributed numerical/graphical simulation carried out on Mathematica and X windows. An object oriented user interface is designed. The results obtained show that dynamically stable walking and running are achieved.

Acknowledgments

I would like to thank my advisor Prof. Krishnaprasad for the patience and the encouragement he provided throughout the course of this thesis.

Discussions held with Prof. Cohen proved very helpful in understanding the biology behind Central Pattern Generators. Discussions with Dr. Dayawansa enlarged my understanding of the mathematical aspects of the problem.

Dr. Josip Loncaric and Dimitrios P. Tsakiris provided valuable help in figuring out the computer systems in the laboratory.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The seeming ease with which vertebrates move indicates the versatility of the biological control systems governing such movement. An investigation into the structure of such control systems seems worthwhile.

1.1 Hierarchical Control

Experiments indicate that motor control in vertebrates is hierarchical. The hierarchy may be constructed as follows:

1. Cerebellum. The cerebellum smooths body motion and integrates it with sensory input below the level of consciousness. It does this by damping out oscillations, predicting the position of different parts of the body and their relation to surrounding objects, and by maintaining body equilibrium [5].
2. Sensory Motor Cortex and Basal Ganglia. The cerebral cortex is central to learning new skills. Muscles are represented in the cerebral cortex to the extent they have to perform fine, learned movements - the degree of representation of muscles of thumbs and fingers and of throat regions is much larger than that of trunk muscles. Basal ganglia are well connected to the sensorimotor cortex. Diseases which affect basal ganglia severely affect motor control [5].
3. Spinal Cord. A large portion of control required to bring about locomotion is present in the spinal cord. When the spinal cord is cut at the level of first cervical vertebrae just below the brain stem, the animal retains a number of motor skills. Reflexes associated with scratching, withdrawal and stretching are still present. When kittens are subject to such a spinal transection one or two weeks after birth

some of them develop the ability to walk. If such a kitten is loosely supported by a sling and the hindlimbs allowed to touch a treadmill, the hindlimbs display a walking motion. Gait changes are induced as the speed of the treadmill is increased [4].

The spinal cord performs repetitive stereotyped actions leaving the conscious portion of the brain free to concentrate on more important tasks.

1.2 Form of Spinal Controller

The experiments mentioned in the previous section demonstrated that control of locomotion is to a large degree localized in the spinal cord. However the exact form of the controller is not known. It could, for example, be thought that locomotion is nothing but a series of reflexes, the completion of one instigating the following one. Such control is similar to modelling the spinal cord as a finite state machine. A finite state machine approach has been suggested by Tomovic, et. al.[17]. They have derived a finite state model for locomotion of a cat relying on sensory input of joint angles and foot contacts.

This form of control requires sensory input to be available to the spinal cord for it to know the current state of the body. However cats with spinal cords deafferentiated by cutting all dorsal roots (places where neurons carrying sensory input enter the spinal cord) are still able to walk on a treadmill. Hence it seems that the pattern generator responsible for generating neural signals controlling walking is intrinsic to the spinal cord.

1.3 Overview

In Chapter 2, various types of oscillator networks are described. Methods to achieve prescribed phase differences between the oscillators constituting such networks are discussed and the effects of external disturbances are examined.

Chapter 3 presents a model for ballistic walking in bipeds. The differential equations governing movement are derived and a control strategy based on pulse-coupled oscillators is described. A parallel finite state model is also described. The role of sensory input in stabilizing the walk is considered.

In the following chapter a similar model is constructed for the case of biped running. The necessity of calculating ground forces explicitly complicates the model.

In Chapter 5 various strategies for implementing the models are described. Components of the simulation such as the dynamical model generator, differential equation solver, graphical display and user interface are described. Current windowing systems, inter-process communication methods and object-oriented programming techniques used are described.

We conclude by showing the relevance of the work done here to related fields and by pointing out directions of future research.

Chapter 2 Oscillators as Controllers

Certain systems are required to produce repetitive output. The desired output of such a system may be thought of as evolving on a torus. A reference model for such a system must generate output lying on a corresponding torus. A system of coupled oscillators is capable of producing such trajectories. Control signals can be generated by comparing the state of the system to that of that of the oscillator generated reference signals. In such a network of oscillators control information is represented in terms of equilibria and limit cycles.

One of the ways to use a network of oscillators is to sequence various actions through it. Each action can be associated with the phase of a particular oscillator. It can be initiated when the phase of the oscillator reaches a certain threshold value. Thus a collection of oscillators with set phase differences between them can be used to sequence various actions within a cycle. The period of each oscillator must correspond to the period of the cycle.

Other schemes using oscillators as controllers are possible. Pattern recognition techniques have been used to identify common features in the control signals. Oscillators are used to generate these feature signals which are then combined to produce the desired control signals [12].

2.1 Method to achieve set phase differences

The basis of this method is a theorem due to Yuasa and Ito[22]. Consider a connected graph with an oscillator at each node as shown below. The directions

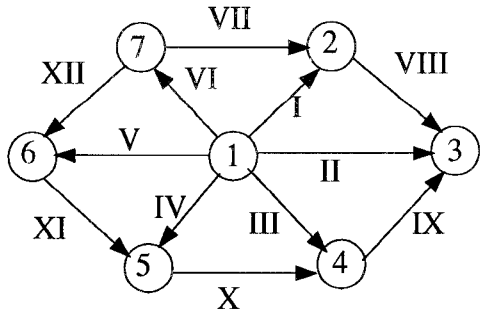


Figure 2.1.1 A network of oscillators. Hindu-Arabic numerals signify oscillators at the nodes. Roman numerals represent the branches connecting various nodes.

of branches don't have any special significance. Any two oscillators connected by a branch affect each other. Branches are assigned a direction so as to be able to construct an incidence matrix corresponding to the graph of the network. The rows of the incidence matrix correspond to nodes while its columns correspond to branches. If a_{ij} is the element corresponding to the i^{th} row and j^{th} column of the incidence matrix A, then

$$\begin{aligned} a_{ij} &= 1 \text{ if branch } j \text{ leaves node } i, \\ &= -1 \text{ if branch } j \text{ enters node } i, \\ &= 0 \text{ if branch } j \text{ does not connect to node } i. \end{aligned} \tag{2.1.1}$$

For example the incidence matrix corresponding to the network in figure 2.1.1 is

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.1.2}$$

For a network of oscillators,

θ_i is the phase of the i^{th} oscillator,

$\theta = [\theta_1, \theta_2, ...\theta_n]^T$,

n is the number of oscillators,

N is the number of branches,

$\phi_{ij} = \theta_i - \theta_j$, is the branch phase

and

A is the incidence matrix of the graph.

ϕ is an N-tuple but only n-1 components of ϕ are independent. ϕ is given by

$$\phi = A^T\theta. \tag{2.1.3}$$

Let

$$\frac{d\theta_i}{dt} = f_i\left(\theta_i, \theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_{n_i}}\right) \tag{2.1.4}$$

where,

θ_i is the phase of the i^{th} oscillator,

$\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_{n_i}}$ represent phases of oscillators interacting with i^{th} oscillator,

n_i is the number of such oscillators.

Now,

$$\dot{\phi} = A^T f. \tag{2.1.5}$$

We require ϕ to settle down to a prescribed value irrespective of initial conditions enabling phase differences corresponding to various branches to be used to sequence

actions. Our purpose would be served if we could construct a potential function on the phase space such that

$$\frac{\partial V}{\partial \phi} = -\left(A^T f\right)^T. \tag{2.1.6}$$

Since $\overline{f} = \omega[1, 1, \ldots, 1]^T$ where ω is a positive number lies in the null space of A^T, f can be decomposed as

$$f = \overline{f} + \tilde{f}, \tag{2.1.7}$$

where $\tilde{f}$ corresponds to the coupling. Only $\tilde{f}$ affects the branch phase dynamics. Also, any change which affects all oscillators equally i.e. either all oscillators are speeded up or slowed down, leaves the branch phases unchanged. The following theorem has been stated in the literature. The proof below appears to be original.

Theorem 2.1.1 [Yuasa & Ito]

There exists a potential function on the branch phase space such that

$$\frac{\partial V}{\partial \phi} = -\left(A^T f\right)^T,$$

if and only if the phase dynamics of each oscillator can be written as

$$f_i = \tilde{f}_i(\psi_i) + \overline{f},$$

where

$$\psi_i = \sum_{k=1}^{n_i} (\theta_{i_k} - \theta_i).$$

Then

$$V(\phi) = \sum_{i=1}^{n} \int_0^{\psi_i} \tilde{f}_i(\psi_i) d\psi_i$$

Here θ evolves on a spiral, while ϕ and ψ belong to R^N and R^n respectively.

Proof

(only if)

Suppose a potential function V exists. Let

$$\phi_{ij} = \theta_i - \theta_j. \tag{2.1.8}$$

Now

$$\frac{\partial V}{\partial \phi_{ij}} = -\frac{d\phi_{ij}}{dt} = -\frac{d(\theta_i - \theta_j)}{dt} = \tilde{f}_j - \tilde{f}_i. \tag{2.1.9}$$

Consider two branches which have no node in common. Since only branches connected to a node influence the oscillator corresponding to that node,

$$\frac{\partial^2 V}{\partial \phi_{kl} \partial \phi_{ij}} = \frac{\partial\left(\tilde{f}_j - \tilde{f}_i\right)}{\partial \phi_{kl}} = 0. \tag{2.1.10}$$

Considering two branches which have a node in common, as shown in figure 2.1.2,,

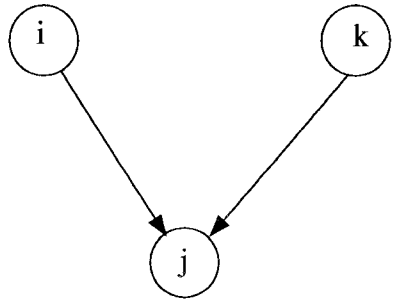


Figure 2.1.2 Three Interacting Oscillators

$$\frac{\partial^2 V}{\partial \phi_{jk} \partial \phi_{ij}} = \frac{\partial\left(\tilde{f}_j - \tilde{f}_i\right)}{\partial \phi_{jk}} = \frac{\partial \tilde{f}_j}{\partial \phi_{jk}}, \tag{2.1.11}$$

and

$$\frac{\partial^2 V}{\partial \phi_{ij} \partial \phi_{jk}} = \frac{\partial\left(\tilde{f}_k - \tilde{f}_j\right)}{\partial \phi_{ij}} = -\frac{\partial \tilde{f}_j}{\partial \phi_{ij}}. \tag{2.1.12}$$

Since

$$\frac{\partial^2 V}{\partial \phi_{jk} \partial \phi_{ij}} = \frac{\partial^2 V}{\partial \phi_{ij} \partial \phi_{jk}}, \tag{2.1.13}$$

$$\frac{\partial \tilde{f}_j}{\partial \phi_{jk}} = -\frac{\partial \tilde{f}_j}{\partial \phi_{ij}}. \tag{2.1.14}$$

Therefore the effect on an oscillator is equal in magnitude for all branches connected to the corresponding node, with the sign depending on whether the branch enters the node or whether it leaves the node. Branches not connected to the node have no effect on the oscillator.

The form of $\tilde{f}_i$ in the theorem follows as shown below. Consider a node at which all the branches connected to it are directed towards it as shown in figure 2.1.3. Then

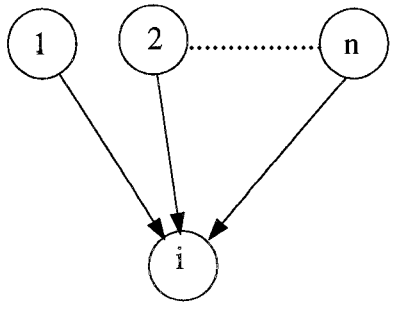


Figure 2.1.3 Interacting Oscillators

$$\frac{\partial \tilde{f}_i}{\partial \phi_{1i}} = \frac{\partial \tilde{f}_i}{\partial \phi_{2i}} = \cdots = \frac{\partial \tilde{f}_i}{\partial \phi_{ni}}. \tag{2.1.15}$$

To determine $\tilde{f}_i$ we integrate along the axes. Integration need only be performed for those branches which connect to the node. Integrating,

$$\begin{aligned}
\tilde{f}_i(\phi_{1i}, \phi_{2i}, \cdots, \phi_{ni}) &= \int_0^{\phi_{1i}} \frac{\partial \tilde{f}(x_1, 0, \cdots, 0)}{\partial \phi_{1i}} dx_1 + \int_0^{\phi_{2i}} \frac{\partial \tilde{f}(\phi_{1i}, x_2, \cdots, 0)}{\partial \phi_{2i}} dx_2 + \cdots \\
&= \int_0^{\phi_{1i}+\phi_{2i}} \frac{\partial \tilde{f}(x_1, 0, \cdots, 0)}{\partial \phi_{1i}} dx_1 + \\
&\qquad \int_0^{\phi_{3i}} \frac{\partial \tilde{f}(\phi_{1i}+\phi_{2i}, 0, x_3, \cdots, 0)}{\partial \phi_{3i}} dx_3 + \cdots \\
&= \int_0^{\phi_{1i}+\phi_{2i}\cdots+\phi_{ni}} \frac{\partial \tilde{f}(x_1, 0, \cdots, 0)}{\partial \phi_{1i}} dx_1 \\
&= \tilde{f}(\phi_{1i} + \phi_{2i} \cdots + \phi_{ni}, 0, \cdots, 0) \\
&= \tilde{f}(\phi_{1i} + \phi_{2i} \cdots + \phi_{ni}) \\
&= \tilde{f}\left(\sum_{k=1}^{n} (\theta_k - \theta_i)\right).
\end{aligned} \tag{2.1.16}$$

Similarly $\tilde{f}_i$ can be shown to have the requisite form for a node with branches both coming in and going out. This completes the first part of the proof.

(If)

The phase dynamics for each oscillator can be written as

$$f_i = \tilde{f}_i(\psi_i) + \overline{f}. \tag{2.1.17}$$

Construct the candidate potential function as

$$V(\phi) = \sum_{i=1}^{n} \int_0^{\psi_i} \tilde{f}_i(\psi_i) d\psi_i. \tag{2.1.18}$$

Now

$$\frac{\partial V(\phi)}{\partial \phi} = \frac{\partial V}{\partial \psi} \frac{\partial \psi}{\partial \phi}. \tag{2.1.19}$$

ψ_i is simply the difference between sums of relative phases of branches coming into node i and the sum of relative phases of branches going out of node i. Hence,

$$\psi = -A\phi, \tag{2.1.20}$$

and

$$\therefore \quad \frac{\partial V}{\partial \phi} = \left(\tilde{f}^T\right)(-A) = -\left(A^T \tilde{f}\right)^T. \tag{2.1.21}$$

2.2 Subtleties

Suppose one cycle of the oscillators is of unit length, that is [0,1), [1.2), [2,3) and so on. Now suppose that it is required for two oscillators to be in phase and that one of the oscillators is a quarter of a cycle ahead of the other. Then

1. The oscillator which is leading could slow down and the oscillator which is lagging could speed up. This happens if the branch phase difference has magnitude of 0.25.
2. The oscillator which is leading could speed up and the oscillator which is lagging could further slow down. This happens if the magnitude of branch phase difference is 0.75.

In both ways the oscillators would eventually be in phase. However the second method is faster than the first. Which of these two happens depends on how we set up the initial branch phase differences. If the branch phase is normalized to lie between -0.5 and 0.5 the faster method will be adopted. This normalization can be perfomed by adding or subtracting a fixed number of cycles from the values obtained

by applying A^T to the initial oscillator phases. The values added or subtracted initially are used each time the branch phase difference vector is calculated.

This also takes care of oscillators being on different turns of the spiral. [0,1) is counted as the first turn of the spiral, [1,2) is counted as the second turn of the spiral and so on. Suppose oscillator 1 is at 0. Then oscillator 2 being at 0.25 or 1.25 should result in the same branch phase difference.

Since all the oscillators have the same frequency and a branch phase difference is obtained by taking the difference between the phase differences of two oscillators, the initial normalization holds for all subsequent times.

2.3 Choice of Potential Functions

Some authors [22] have chosen trigonometric functions as potential functions. However such functions have the drawback that the system may have undesirable convergence properties. We desire the system to converge faster the further away it is from the desired operating point. Suppose the desired operating point is the origin and we chose $\sin(\phi\pi - \frac{\pi}{2}) + 1$ as the potential function. Then the rate of convergence becomes slower as the branch phase increases beyond $\frac{\pi}{4}$ or decreases below $-\frac{\pi}{4}$. In fact the system has an unstable equilibrium at $\frac{\pi}{2}$. Later on in this chapter we show a potential function which does not have such undesirable convergence properties.

2.4 Effect of Disturbances

Any network of oscillators which can function as a CPG should be robust against small disturbances. To function as a CPG the branch phase difference system must converge in an exponentially stable manner to the phase desired. This would imply

that arbitrarily small perturbations will not result in large steady state deviations from the origin. This can be seen from the following theorem proved in [8].

Theorem 2.4.1 [Miller, Michell, Coreless and Leitmann]

Let x=0 be an exponentially stable equilibrium point of the system

$$\dot{x} = f(t, x). \tag{2.4.1}$$

Consider the perturbed system

$$\dot{x} = f(t, x) + g(t, x) \tag{2.4.2}$$

Let V(t,x) be a Lyapunov function for the system 2.4.1satisfying

$$c_1\|x\|_2^2 \leq V(t, x) \leq c_2\|x\|_2^2, \tag{2.4.3}$$

$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(t, x) \leq -c_3\|x\|_2^2, \tag{2.4.4}$$

and

$$\left\|\frac{\partial V}{\partial x}\right\|_2 \leq c_4\|x\|_2 \tag{2.4.5}$$

in $[0, \infty) \times D$, where $D = \{x \in R^n \mid \|x\|_2 < r\}$. Suppose the perturbation term g(t,x) satisfies

$$\|g(t, x)\|_2 \leq \delta < \frac{c_3}{c_4}\sqrt{\frac{c_1}{c_2}}\theta r \tag{2.4.6}$$

for all t ≥ 0, all $x \in D$, and some positive constant θ<1. Then, for all $\|x(t_0)\|_2 < \sqrt{c_1/c_2}r$, the solution of the perturbed system satisfies

$$\|x(t)\| \leq k \exp\left[-\gamma(t - t_0)\right]\|x(t_0)\|, \ \forall t_0 \leq t < t_1, \tag{2.4.7}$$

and

$$\|x(t)\| \leq b, \ \forall t \geq t_1, \tag{2.4.8}$$

for some finite time t_1, where

$$k = \sqrt{\frac{c_2}{c_1}}, \quad \gamma = \frac{(1-\theta)c_3}{2c_2}, \quad b = \frac{c_4}{c_3}\sqrt{\frac{c_2}{c_1}}\frac{\delta}{\theta} \ . \qquad \blacksquare$$

We can write the system 2.1.5 as

$$\dot{\theta} = \omega U + \tilde{f}(\psi) + f_d(t) \tag{2.4.9}$$

where,

ω is a constant,

$U = [1, 1, ...1]^T$,

ψ= —$AA^T\theta$ and

$f_d(t)$ is the disturbance.

$f_d(t)$ can be written as

$$f_d = (\Delta\omega(t))U + \alpha(t), \tag{2.4.10}$$

where $(\Delta\omega(t))$ is chosen such that $\|\alpha(t)\|_2$ is minimized.

$$\therefore \ \dot{\theta} = (\omega + \Delta w(t))U + \tilde{f}(\psi) + \alpha(t) \ . \tag{2.4.11}$$

This means that the nominal system of oscillators is speeded up or slowed down by $\Delta\omega(t)$. Since $U \in Ker(A^T)$,

$$\dot{\phi} = A^T\tilde{f}(\psi) + A^T\alpha(t). \tag{2.4.12}$$

We have decomposed the disturbance into a part affecting all oscillators equally and another part affecting the phase differences between oscillators.

The nominal system is

$$\dot{\phi} = A^T \tilde{f}(\psi) \tag{2.4.13}$$

For the nominal system

$$V(\phi) = \sum_{i=1}^{n} \int_0^{\psi_i} \tilde{f}_i(\psi_i) d\psi_i \tag{2.4.14}$$

To ensure exponential stability $\tilde{f}(\psi_i)$ is chosen such that equations 2.4.3, 2.4.4 and 2.4.5 are satisfied.

2.4.1 A particular case

Suppose we choose,

$$\begin{aligned} \tilde{f}(\psi) &= [\psi - \psi_{desired}] \\ &= -A[\phi - \phi_{desired}] \\ &= -AA^T[\theta - \theta_{desired}] \,. \end{aligned} \tag{2.4.15}$$

Here θ_{desired} reflects the phase differences between the various oscillators constituting the system. If we choose the first oscillator as the reference oscillator we can write θ_{desired} as

$$\theta_{desired} = \begin{bmatrix} 0 \\ \Delta\theta_2 \\ \vdots \\ \Delta\theta_i \\ \vdots \\ \Delta\theta_n \end{bmatrix}, \tag{2.4.16}$$

where $\Delta\theta_{\text{i}}$ represents the desired phase difference between the i^{th} oscillator and the first oscillator. It is to be noticed that to set the phase differences between various oscillators all that is required is a setting of $\theta_{desired}$ at each oscillator. $\Delta\theta_{\text{i}}$ is chosen to lie between —0.5 and 0.5.

$$
\begin{aligned}
V(\phi) &= \frac{1}{2}[\psi - \psi_{desired}]^T[\psi - \psi_{desired}] \\
&= \frac{1}{2}[\phi - \phi_{desired}]A^T A[\phi - \phi_{desired}] \\
&= \frac{1}{2}[\theta - \theta_{desired}]AA^T AA^T[\theta - \theta_{desired}]
\end{aligned}
\tag{2.4.17}
$$

We would like to show that this potential function satisfies conditions specified in equations 2.4.3, 2.4.4 and 2.4.5.

Since A is the incidence matrix of a connected graph having n nodes, it has a rank of n-1. Hence AA^TAA^T has a rank of n-1. The eigenvectors of A and AA^TAA^T corresponding to the eigenvalue of zero are of the form ωU where $\omega \in R$. These map to the origin of branch phase space. Hence V(0)=0 and c_1 of equation 2.4.3 is the minimum eigenvalue of A^TA greater than zero. c_2 of equation 2.4.3 is the maximum eigenvalue of A^TA.

Since

$$
\begin{aligned}
\dot{V}(\phi) &= -\tilde{f}(\psi)AA^T\tilde{f}(\psi) \\
&= -[\phi - \phi_{desired}]^T A^T AA^T A[\phi - \phi_{desired}]^T,
\end{aligned}
\tag{2.4.18}
$$

in equation 2.4.4 c_3 is the square of the minimum eigenvalue of A^TA greater than zero.

Similarly c_4 of equation 2.4.5 is the maximum eigenvalue of A^TA.

2.5 Design of oscillator networks

The first step in the design of an oscillator network is to associate an oscillator with each action to be sequenced and to get the phase differences between the actions. This gives us the phase difference across various branches. We then specify a potential function which has a minimum corresponding to the desired phase difference vector. The coupling term at the i^{th} oscillator is obtained by differentiating

the potential function with respect to ψ_i for that oscillator. The coupling term is then summed with the dynamic term common to all oscillators, ω, to obtain the dynamics at each oscillator. This process will be illustrated in the following chapters.

We would like the network of such oscillators to converge as quickly as possible to the steady state value.

$$\frac{dV}{dt} = -\left(A^T \tilde{f}\right)^T \left(A^T \tilde{f}\right) . \tag{2.5.1}$$

If $\left|\frac{dV}{dt}\right|$ is large, convergence is faster. Hence we would like to choose A and $\tilde{f}$ so as to make it as large as possible.

There may be some gain limitations on the coupling. These imply that

$$\left|\tilde{f}_i\right| \leq \gamma_i, \tag{2.5.2}$$

where γ_i is some positive constant.

Since $\tilde{f}_i$ is a continuous function defined on a compact set, it attains a maximum and a minimum. It attains a maximum absolute value. Once the form of $\tilde{f}_i$ has been chosen it can be normalized using $\frac{\gamma_i}{\left|\tilde{f}_i\right|_{max}}$.

Adding more branches increases the dimension of $\left(A^T \tilde{f}\right)$, hence the rate of convergence will increase with an increase in number of branches. Therefore the oscillator network should have as many branches as possible.

If groups of actions are to be sequenced it may be useful to construct a hierarchy of oscillators. For each group of action we assign one oscillator. Then for actions within the group a subnetwork of oscillators is designed. For example if it is desired to control the locomotion of quadrupeds, we can construct a set of oscillators which give the timing of each leg separately. A further set of oscillators corresponding to each leg can be constructed to control individual muscles.

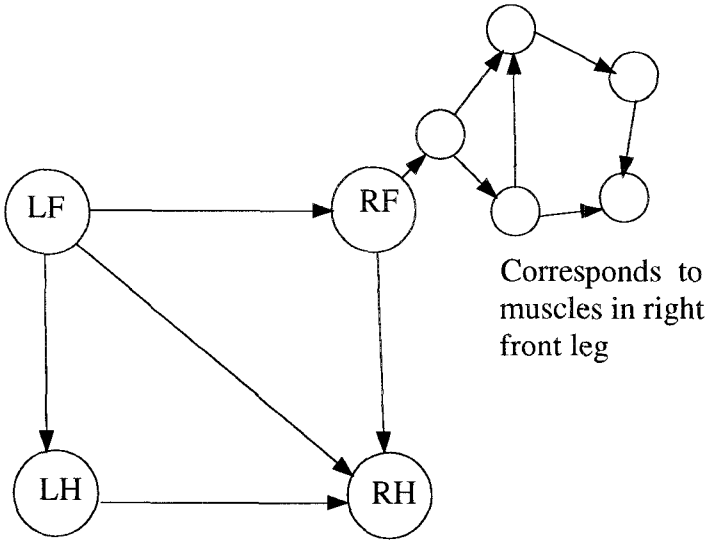


Figure 2.5.1 Oscillator hierarchy

2.6 Pulse Coupled Oscillators

The oscillators described till now interact with each other continuously. Peskin[13] proposed a scheme for synchronization of two oscillators coupled by intermittent pulses. Mirollo and Strogatz[10] generalized his idea. They considered a network of identical oscillators with each oscillator being coupled to all the remaining oscillators. Oscillators are of "integrate and fire" type. Their dynamics are characterized by a variable rising towards the threshold with a time evolution which is monotonic and concave down. When an oscillator reaches the threshold it "fires" sending out a pulse to all other oscillators. This pulse pushes the influenced oscillator up by a fixed amount or to threshold, whichever is less.

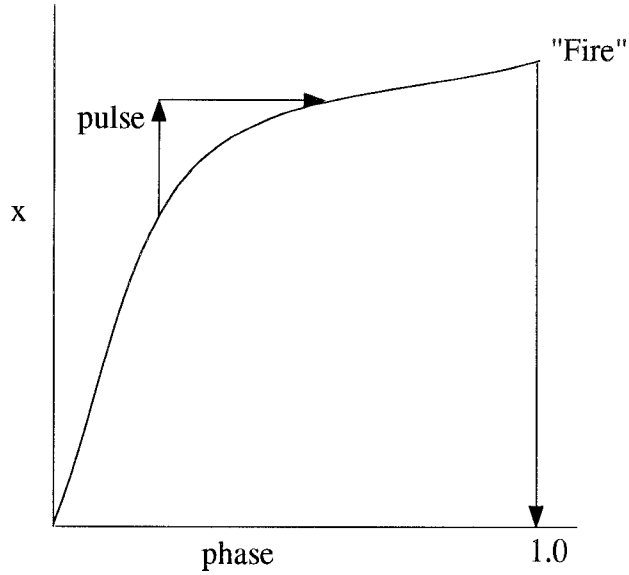


Figure 2.6.1 Integrate and fire type oscillator affected by a pulse

Mirollo and Strogatz proved that for any number of oscillators and for almost all initial conditions the oscillators constituting the network become synchronized.

Pulse coupling can also be used to achieve set phase differences between two oscillators. Consider an oscillator on which the influence of incoming pulses, whether inhibitory or excitatory depends on its phase. At a particular switch over phase the influence of the incoming pulse changes from inhibitory to excitatory. Now consider two such oscillators. For the first oscillator the switch over phase is at α_1. Above this phase the influence of incoming pulse is inhibitory and below this phase the influence of the incoming pulse is excitatory. The corresponding switch phase for the other oscillator is $(1-\alpha_1)$. Then when the second oscillator fires the first oscillator is moved towards α_1 phase and when the first oscillator fires the second oscillator is moved towards phase $(1-\alpha_1)$. Oscillatory motion results if the impulse given

is such that the phase of the influenced oscillator overshoots the switch over phase. The greater the impulse given, the faster the settling time but the more the overshoot.

This problem can be overcome by making the magnitude of the influence of the phase near the switch over phase dependent on the difference between the current phase and the switch over phase,

$$\begin{aligned} \theta_{after} &= \theta_{before} + \Delta. \\ \Delta &= -(\theta - \alpha), \text{ for } |(\theta - \alpha)| \leq \delta, \\ &= \delta, \text{ for } (\theta - \alpha) > \delta, \\ &= -\delta, \text{ for } (\theta - \alpha) < \delta, \end{aligned} \tag{2.6.1}$$

where,

θ_{before}, θ_{after} are the phases before and after the pulse,

Δ is the actual influence of the pulse,

δ is the influence of the pulse away from the switch over phase and

α is the switch over phase.

2.7 Almost Oscillators

It is not necessary that the building blocks of a Central Pattern Generator controller oscillate. Instead they may have some special properties that can cause them to oscillate if they are subject to oscillatory input or if some parameters are changed.

2.7.1 Damped Oscillators

Slightly damped oscillators may become oscillatory if they are subject to sensory feedback at appropriate points of their cycle. For example, consider an oscillator whose oscillations die out with time. Consider a section along the positive half of

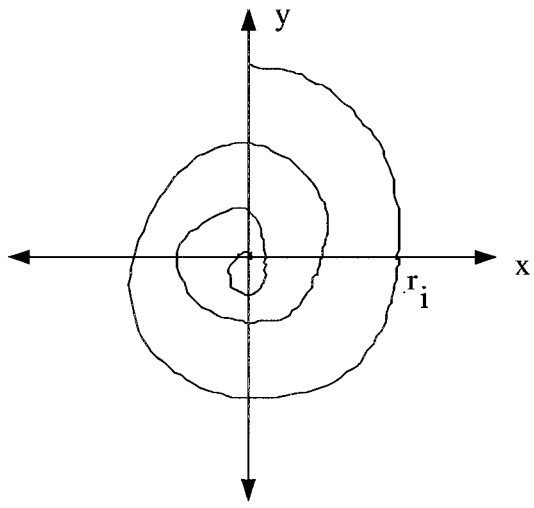


Figure 2.7.1 Damped Oscillator

the x axis. Let r_i be the position on the positive half of x axis the i^{th} time the trajectory crosses the section. Let

$$g_u(r_n) = r_n - r_{n+1}, \tag{2.7.1}$$

where the subscript u denotes the unperturbed system. Now if the oscillator is subject to a pulse just before its trajectory hits the section,

$$r_{n+1} = r_n - g_u(r_n) + p, \tag{2.7.2}$$

where p is the strength of the pulse. The system has equilibrium point at $g_u^{-1}(p)$. This will be asymptotically stable if

$$0 < \frac{\partial g}{\partial r}|_{g_u^{-1}(p)} < 2\ . \tag{2.7.3}$$

This is always true for linear systems with eigenvalues of the form $-\alpha \pm \beta$ where $\alpha, \beta > 0$.

The cycle may be started consciously, become self sustaining and be controlled unconsciously. Initially motion is initiated through conscious action, sensory feedback builds up oscillations in the damped oscillators. They start functioning as Central Pattern Generators and take over the control of motion.

2.7.2 Excitable Systems

Such a system may be described as an "one shot oscillator". Without stimulation such a system remains in stable equilibrium. A small change in conditions away from stable equilibrium point leads to a large excursion before the system returns to rest. These may be used in a way similar to damped oscillators with sensory input providing the disturbances required to cause them to oscillate.

Chapter 3 Walking

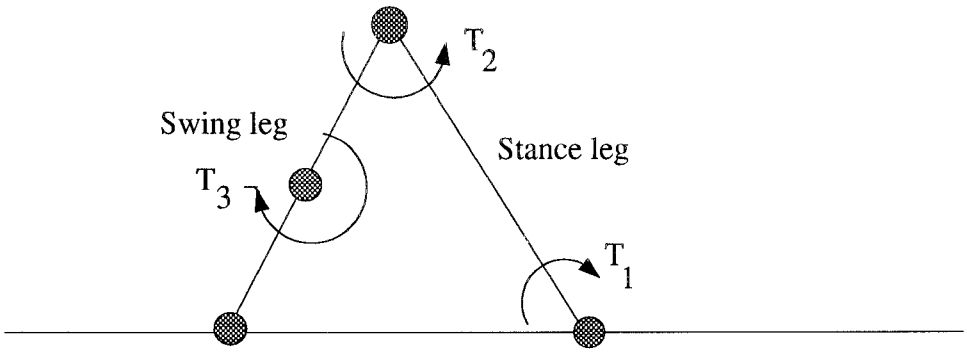
One of the tasks for which Central Pattern Generators might be employed is locomotion. Locomotion in mammals is a complex phenomenon. Each mammal moves in its own characteristic way. Gait changes with speed further complicate the issue. Human walking will be examined in this chapter.

3.1 Ballistic Walking

Electromyographic records using electrodes in leg muscles of humans show that there is very little activity in the swing leg during walking at normal speed except at the beginning and at the end of swing phase[1]. This led McMahon to propose a model in which the joints velocities at the start of swing are established by muscular forces after which the leg swings through under the influence of gravity and its established inertia[11]. Since no joint torques act on the swing leg while it is in motion, this model of walking is called the ballistic walking model. We used a model adapted from this.

3.2 Model

The model consists of four rigid links. The upper portion of the body is modelled as a point mass concentrated at the end of the stance leg. Since the foot of the stance leg is planted on ground, it is not taken into consideration while deriving the dynamical equations. The swing leg has three links one each for the thigh, the shank and the foot. The numerical values of system parameters are given in the appendix. At start, the legs have configuration as shown in figure 3.2.1



Initial position and torques

Figure 3.2.1 Initial position and torques

Torques are applied as shown in the figure. In human walking the foot of the swing leg pushes back against the ground to generate equivalent T_1. Under the action of these torques, the stance leg begins to rotate forward, the thigh of the swing leg moves forward and the shank of the swing leg lifts up to give the necessary clearance from ground.

3.2.1 Derivation of dynamical equations

The dynamical equations are derived using the Newton-Euler method. The stance leg is considered fixed to the ground. In practice this means that the coefficient of friction is high enough so that the foot does not slip.

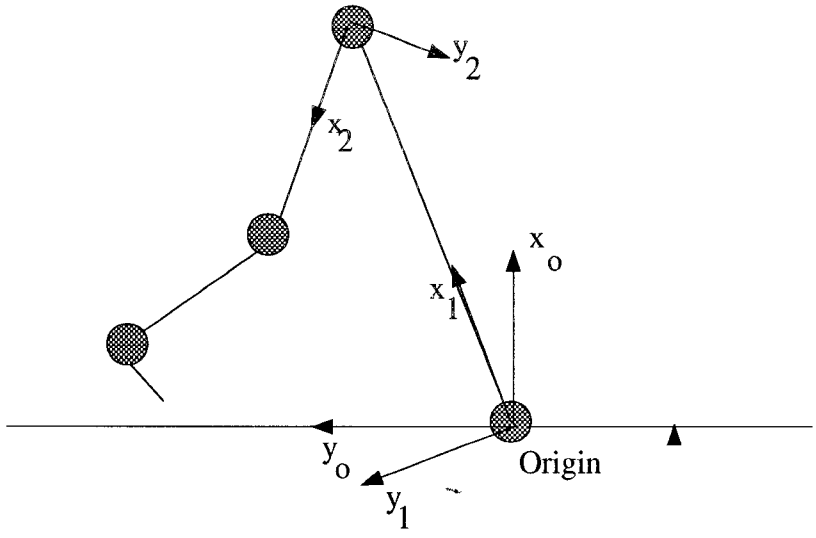


Figure 3.2.2 Reference frames

A reference frame is assigned to each link. The z axis is chosen perpendicular to the plane and pointing outwards, the x axis is chosen to lie along the link and the y axis is chosen so as to form a right handed coordinate system. Ground is chosen as link 0. The place where the heel of the stance leg is planted on the ground is chosen as the origin of the system and that of reference frame 0. x axis for link 0 is chosen to be vertical and pointing away from the ground, as shown in figure 3.2.2. A forward recursion starting from the shank of the stance leg and going out to the foot of the swing leg is used to calculate the velocities and accelerations of the links constituting the legs,

$$
\begin{aligned}
{}^{i}\omega_i &= {}^{i-1}R_i^T\,{}^{i-1}\omega_{i-1} + e_z\dot{q}_i,\\
{}^{i}\dot{\omega}_i &= {}^{i-1}R_i^T\,{}^{i-1}\dot{w}_{i-1} + e_z\ddot{q}_i + \left({}^{i-1}R_i^T\,{}^{i-1}\omega_{i-1}\right) \times e_z\dot{q}_i,\\
{}^{i}\ddot{p}_i &= {}^{i-1}R_i^T\left[{}^{i-1}\ddot{p}_{i-1} + {}^{i-1}\dot{\omega}_{i-1} \times^{i-1} \hat{p}_i + {}^{i-1}\omega_{i-1} \times \left({}^{i-1}\omega_{i-1} \times^{i-1} \hat{p}_i\right)\right],\\
{}^{i}\ddot{s}_i &= {}^{i}p_i + {}^{i}\omega_i \times^{i} \hat{s}_i + {}^{i}\omega_i \times \left({}^{i}\omega_i \times^{i} \hat{s}_i\right),
\end{aligned}
\tag{3.2.1}
$$

where,

the superscript denotes the reference frame and the subscript the link,

ω is the angular velocity of a link,

${}^{i}p_i$ is the vector from origin of the system to joint i in reference frame i,

${}^{i-1}R_i$ is the transformation matrix from reference frame i to $i-1$,

e_z is the joint axis,

$\dot{q}_i$ is the joint velocity,

${}^{i-1}\hat{p}_i$ is the vector from joint $i-1$ to joint i in frame $i-1$,

${}^{i}\hat{s}_i$ is the vector from origin of system to center of gravity of the link in frame i

and

T denotes transpose.

A backward recursion starting from the foot of the swing leg and going back to the shank of the stance leg is used to calculate the forces and joint moments exerted on each link:

$$
\begin{aligned}
&{}^{i}\hat{f}_i = m_i {}^{i}\ddot{s}_i \ , \\
&{}^{i}\hat{n}_i = {}^{i}I_i {}^{i}\dot{\omega}_i + {}^{i}\omega_i \times \left({}^{i}I_i {}^{i}\omega_i\right) \ , \\
&{}^{i}f_i = {}^{i}R_{i+1} {}^{i+1}f_{i+1} + {}^{i}\hat{f}_i, \\
&{}^{i}n_i = {}^{i}R_{i+1} {}^{i+1}n_{i+1} + {}^{i}\hat{n}_i + {}^{i}\hat{s}_i \times {}^{i}\hat{f}_i + {}^{i}\hat{p}_{i+1} \times \left({}^{i}R_{i+1} {}^{i+1}f_{i+1}\right) \ , \\
&\tau_i = e_z^{T\,i}n_i \ , \text{where i is 1,2, 3,4.}
\end{aligned}
\tag{3.2.2}
$$

Here,

${}^{i}\hat{f}_i$ is the total external force on link i in frame i,

${}^{i}\hat{n}_i$ is the total external moment on link i frame i,

${}^{i}I_i$ is the inertia tensor of link i in frame i,

${}^{i}n_i$ and ${}^{i}f_i$ are the force and moment exerted by link $i-1$ on link i

and

τ_i is the joint driving force.

Taking the acceleration of the zero link, i.e. the ground, as negative of the gravitational acceleration accounts for gravity.

3.3 Description of motion

The cycle starts with the applied torques as shown in figure 3.2.1. Under the action of these torques the stance leg starts moving forward. The shank of the swing leg lifts upwards and the swing leg swings forwards. This lift of the shank of the swing leg is important as it allows clearance for the swing leg to move forwards without its toe touching the ground. This is shown in figure 3.3.1

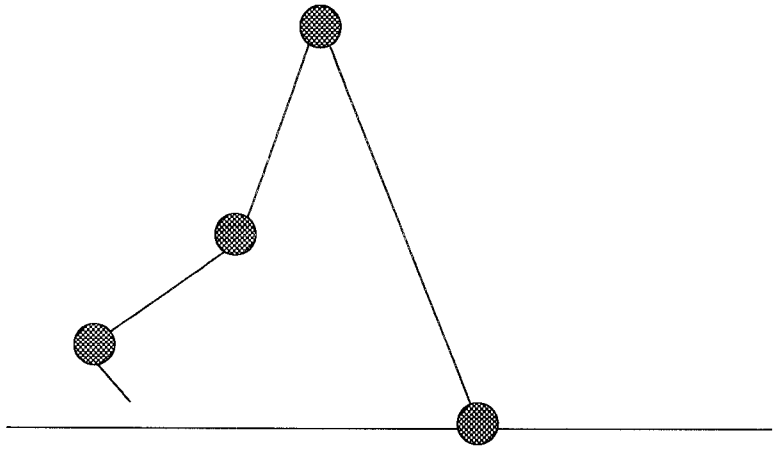


Figure 3.3.1 Take-off.

The stance leg continues to rotate forwards, till it becomes vertical and finally tips over as shown in figure 3.3.2. Since the mass of the upper body is concentrated on top of the stance leg its angular velocity increases rapidly.

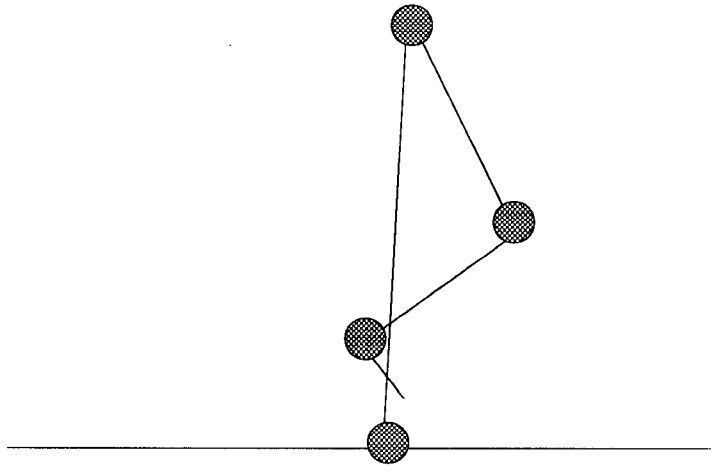


Figure 3.3.2 Midswing.

The swing leg clears the ground as shown in figure 3.3.3. This is a critical stage since the toe of the swing leg can be stubbed against the ground if clearance is not enough.

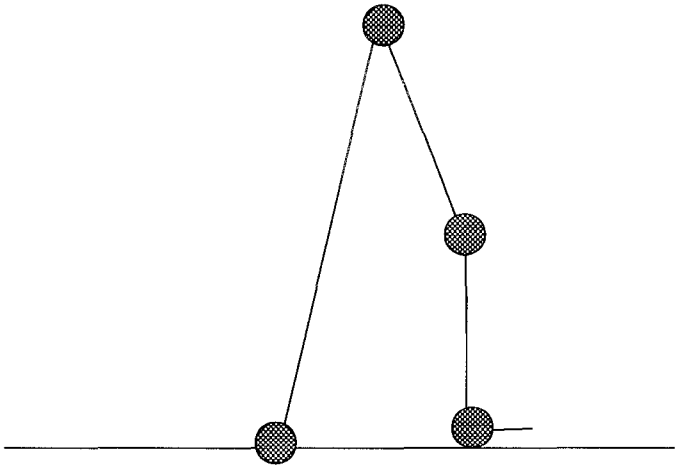


Figure 3.3.3 Clearing the ground.

The shank of the swing leg continues to swing forward and straightens out. Ideally the heel of the swing leg should strike the ground at this precise moment, but this is difficult to achieve in practice. A small clearance exists between ground and the heel of the swing leg when the swing leg straightens out. This is shown in figure 3.3.4

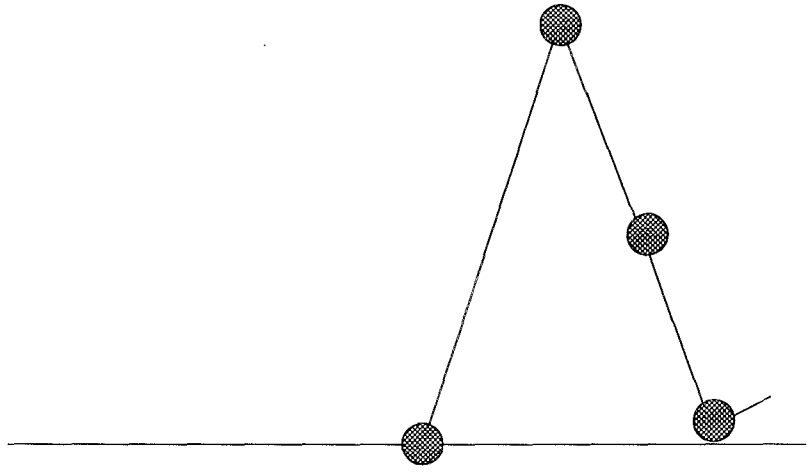


Figure 3.3.4 Touchdown.

Torque applied at the hip joint brings the swing leg down and the whole cycle repeats itself.

3.4 Energy Considerations

There is a continuous interchange between kinetic energy and potential energy while walking. When the body mass is at its highest position, i.e. the stance leg is vertical the potential energy of the system is high. As the stance leg tips forward this potential energy is converted into kinetic energy. At the beginning of the next swing the momentum is carried over. Only that component of linear momentum of the body mass which is perpendicular to the new stance leg is transferred. The momentum component which is parallel to the new stance leg is dissipated. The greater the arc of the swing, the greater is the dissipation of energy. Humans adopt a variety of means like pelvic rotation, pelvic tilt and stance leg knee flexion to flatten the arc along which the body moves and hence to reduce the energy dissipation[9]. These adaptations contribute to the characteristic human walk.

3.5 Limitations of the ballistic model

Human gait cannot be adequately modelled in two dimensions. Various adaptations to make the trajectory of our walk flatter require a three dimensional model. The effects of pelvic rotation, pelvic tilt, stance leg knee flexion, plantar flexion of the stance ankle and lateral displacement of the pelvis may be added to make the model more realistic.

The absence of muscular torque during the swing phase and elastic storage of energy in the tendons implies that the ballistic walking model cannot be extended to model running. Muscular torques during the swing phase and elastic storage of energy play a significant role in running.

3.6 Control

The thing to notice about the system is that in the desired steady state its behavior is cyclic. Therefore the most natural description of the state of the system would be as evolving on a torus. Hence the output of a set of oscillators which evolves on a torus can serve as the reference model for such a system or apply control toques at various points.

Since starting torques are applied in synchrony at all joints of the same leg, the set of oscillators needed to perform control can be simple.

Consider two sets of oscillators corresponding to the two legs. Each set has three oscillators corresponding to the ankle joint of the leg, the hip joint and the knee joint of the other leg. Ideally, the oscillators in a set are in phase with each other but approximately 180° out of phase with oscillators in the other set. The oscillators of a particular set have zero phase when the leg to which the oscillators correspond

strikes the ground and becomes the stance leg. The oscillators are identical and their phases θ_i evolve as follows:

$$\dot{\theta}_i = \omega, \text{ for } i = 1, 2, 3 \, . \tag{3.6.1}$$

The oscillator's phase evolves as a ramp. When the oscillator's phase reaches unity, the oscillator "fires" (is reset).

$$\theta_i = 0 \text{ if } \theta_i \geq 1, \text{ for } i = 1, 2, 3 \ . \tag{3.6.2}$$

Torques are applied at a particular joint when the oscillator corresponding to that joint has phase below a certain cutoff value c. When the other leg strikes the ground, the set of oscillators corresponding to that leg are responsible for the application of the requisite torques.

The only thing which we have to take care of is that the oscillators have zero phase when their corresponding leg strikes the ground. For this purpose pulse coupling between the mechanical links and the oscillators is used.

Here the frequency of the oscillators, ω, is kept lower than the frequency of the system. i.e.

$$\omega < \frac{1}{2t_s}, \tag{3.6.3}$$

where t_s is the time elapsed between successive steps.

Then when the leg corresponding to a particular set of oscillators strikes the ground, a pulse, Δ, is sent to the ankle oscillator. The phase of the ankle oscillator after

the strike becomes,

$$\theta_{as} = \theta_{bs} + \Delta,$$

where,

θ_{bs} is the oscillator phase before strike, (3.6.4)

Δ corresponds to the stength of pulse sent and

θ_{as} is the oscillator phase after strike .

If $\theta_{as} > 1$, the oscillator fires, θ_{as} is reset to zero and a pulse Δ_o is sent to the next oscillator, the hip oscillator. The hip oscillator behaves similarly. If is sufficiently close to firing, i.e.$\theta + \Delta_o > 1$, it fires sending a pulse to the next oscillator, the knee oscillator of the leg which is about to begin its swing. Thus all the oscillators fire roughly in unison with the swing leg strike. This allows the entrainment of the oscillators and the strikes of the corresponding leg and hence the proper application of torques.

In nature too, the frequency of deafferentiated Central Pattern Generator is less than that of the afferentiated Central Pattern Generator[16]. This seems to indicate that this lower frequency may be an important component in entrainment of oscillators and the system they control.

3.6.1 Pulse coupled oscillators

A problem with the above model is that it requires sensory input from ground interaction to function. In the absence of sensory input the oscillators could drift apart due to disturbances. We need to provide some stability to the phase differences between the oscillators.

One way to do this would be to use pulse coupling as discussed in section 2.6. The three oscillators providing take-off torque for each leg could be pulse coupled together. This would ensure that they remain in synchrony and that the take-off torques are applied at all joints simultaneously.

However pulse coupling only provides a way to ensure that oscillators are in synchrony. There is no natural way to achieve set phase differences between oscillators. The take-off oscillators for the two legs would have to remain independent of each other.

To achieve the desired set phase differences the method discussed next proves useful.

3.6.2 Continuously coupled oscillators

We would like the network of oscillators controlling the motion to be responsive to sensory input as well as be able to generate necessary control signals in a robust fashion in the absence of sensory input. To achieve this the following strategy seems useful. Construct a network of oscillators such that the equilibrium point in the branch phase space corresponds to the desired phase difference between oscillators. Also let the oscillators respond to sensory input as discussed earlier. Here the coupling maintains a set phase difference between the oscillators while the sensory input synchronizes the oscillators with physical events. Such a system may be considered to be a hybrid of continuously coupled and pulse coupled systems. There is a *weak background of continuous inter-oscillator coupling* upon which a *strong pulse coupling between the oscillators and mechanical links* is superimposed.

Consider the network of oscillators as shown figure 3.6.1 The incidence matrix

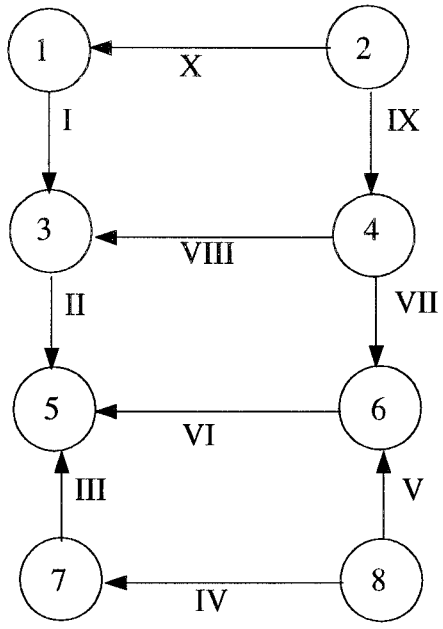


Figure 3.6.1 A network of oscillators to control walking

corresponding to this network is

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.6.5}$$

Oscillators 1, 2 and 3 are responsible for take-off torques exerted when leg 1 strikes the ground. Oscillator 4 controls the hip torque applied on leg 2 once its knee joint has reached full extension. The take-off torque when leg 2 strikes the ground is controlled by oscillators 5, 6 and 7. Oscillator 8 controls the hip torque for leg 1.

By examining the sequence of activities involved we can set

$$\theta_{desired} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.4 \\ 0.5 \\ 0.5 \\ 0.5 \\ -0.1 \end{bmatrix}. \tag{3.6.6}$$

These settings arise in a very natural fashion. We take one complete cycle to consist of two consecutive steps. Each leg becomes the stance leg once in each cycle. The two legs are roughly out of phase with each other. So if we take the oscillators responsible for take-off torques for leg 1 as the reference the desired phase difference for the take-off oscillators for leg 2 becomes 0.5. Hip torque for leg 2 is determined by oscillator 4 and is applied a little before leg 2 hits the ground, hence the desired phase difference for oscillator 4 may be set as 0.4. A similar reasoning gives the desired phase difference for oscillator 8 as —0.1. It is not necessary to set the phase differences exactly since feedback in the form of pulses takes care of any discrepancies.

We now construct the potential function as discussed in subsection 2.4.1. Take

$$V(\phi) = k(\theta - \theta_{desired})AA^TAA^T(\theta - \theta_{desired}). \tag{3.6.7}$$

The value of k determines how fast the convergence takes place. It needs to be chosen properly so as to determine an appropriate balance between sensory input and internal coupling. Choosing the potential function as in equation 3.6.7 gives

$$\dot{\theta} = \omega[1 \quad 1 \quad \cdots \quad 1]^T - AA^T[\theta - \theta_{desired}]. \tag{3.6.8}$$

The following figures show how the coupling described here achieves the set phase difference. In these figures the oscillators have a normalized frequency of one. Figure 3.6.2 shows the first and second oscillators of the network becoming in phase. Figure 3.6.3 shows second and sixth oscillators becoming out of phase.

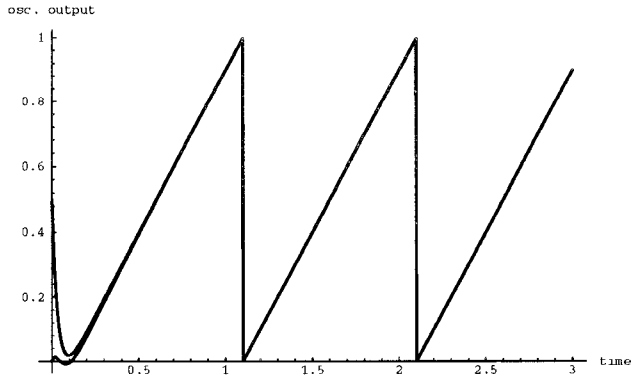


Figure 3.6.2 First and second oscillators become in phase.

Similarly first and fourth oscillators achieve a phase difference of 0.4. The first and

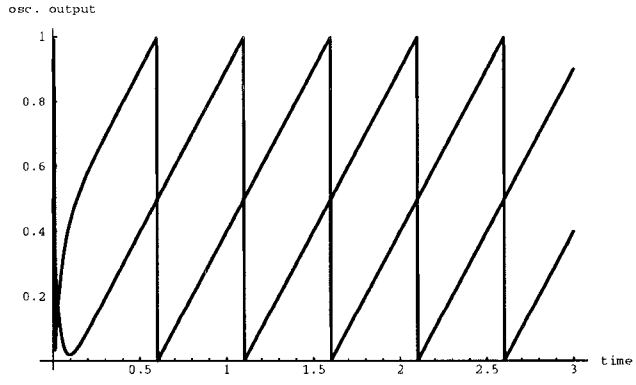


Figure 3.6.3 Second and sixth oscillators become out of phase.

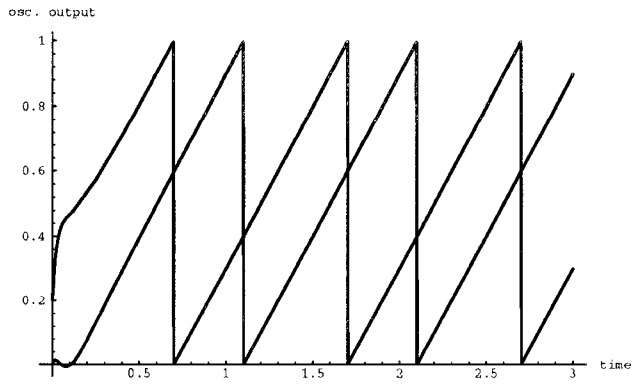


Figure 3.6.4 First and fourth oscillators achieve a phase difference of 0.4

eighth oscillator achieve a phase difference of 0.1 as shown in figure 3.6.5. As can

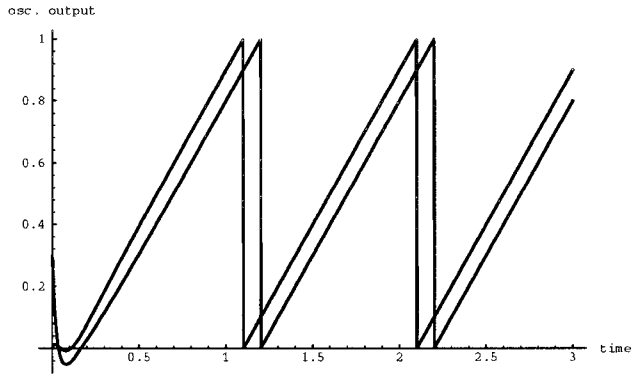


Figure 3.6.5 First and eighth oscillator with a phase difference of 0.1

be seen from these graphs coupling between oscillators may cause the value of the oscillator output to drop below zero.

Oscillators 1, 2 and 3 receive a pulse if leg 1 comes in contact with the ground. This pulse causes the oscillators to fire and they come down to ground state. Torques are applied while their outputs are below a certain value. Similarly, oscillator 4 receives a pulse when the knee joint of leg 2 reaches full extension and it fires causing the hip torque necessary to bring leg 2 to ground to be applied. We would like the hip torque applied to become zero once leg 2 has come in contact with the

ground. Therefore in addition to pulses to oscillators 5, 6 and 7 a pulse is also sent to oscillator 4 when leg 2 strikes the ground. It drives oscillator 7 out of the range in which hip torque is applied. A similar mechanism is responsible for application of torques by oscillators 5, 6, 7 and 8.

3.7 A finite state model

To construct a finite state model we require an input alphabet, an output alphabet, a set of state values and two mappings. One mapping maps the input alphabet to the set of state values and the other mapping maps the set of state values to the output alphabet. Let

$$U = \{u(1), u(2), \cdots, u(m)\}, \tag{3.7.1}$$

be the input alphabet,

$$X = \{x(1), x(2), \cdots, x(n)\}, \tag{3.7.2}$$

be the set of internal states and

$$Y = \{y(1), y(2), \cdots, y(p)\}, \tag{3.7.3}$$

be the output alphabet.

For our system we can take the input alphabet to consist of the 2^5 combinations of five elements. These correspond to the sensory input available. The elements provide information about

1. Full extension of knee joint of leg 1,
2. Full extension of knee joint of leg 2,

3. Contact of leg 1 with ground,

4. Contact of leg 2 with ground ,

5. Whether thigh of leg 1 is ahead of thigh of leg 2.

We can take the state of the system as corresponding to various torques applied on the system. The set of internal states has five elements corresponding to

I. No torque applied on the system,

II. Take-off torque for leg 1,

III. Take-off torque for leg 2,

IV. Hip torque to bring leg 1 to the ground after its knee has reached full extension at the end of swing and

V. Hip torque to bring leg 2 to ground after its knee has reached full extension at the end of swing.

The output alphabet can be taken as directly corresponding to the set of internal states. The mapping from the set of internal states to the output alphabet is hence an identity mapping.

The mapping from the input alphabet to the set of internal states is shown in table 3.7.1. The table shows the input alphabet and the internal states through a complete step cycle. Initially both legs are in contact with the ground with leg 1 being ahead of leg 2. Take-off occurs as a result of torques applied. After take-off no torques are applied and leg 2 swings freely forward till such a time as its knee joint reaches full extension. Hip torque brings leg 2 down to ground. A similar cycle take place for leg 1.

Knee joint of leg 1 fully extended?	Knee joint of leg 2 fully extended?	Leg 1 in contact with ground?	Leg 2 in contact with ground?	Is leg 1 the front leg?	System State
1	1	1	1	1	II
1	0	1	0	1	I
1	0	1	0	0	I
1	1	1	0	0	IV
1	1	1	1	0	III
0	1	0	1	0	I
0	1	0	1	1	I
1	1	0	1	1	V

Table 3.7.1 A finite state model for ballistic walking.

The following figure shows the transition diagram corresponding to the above model.

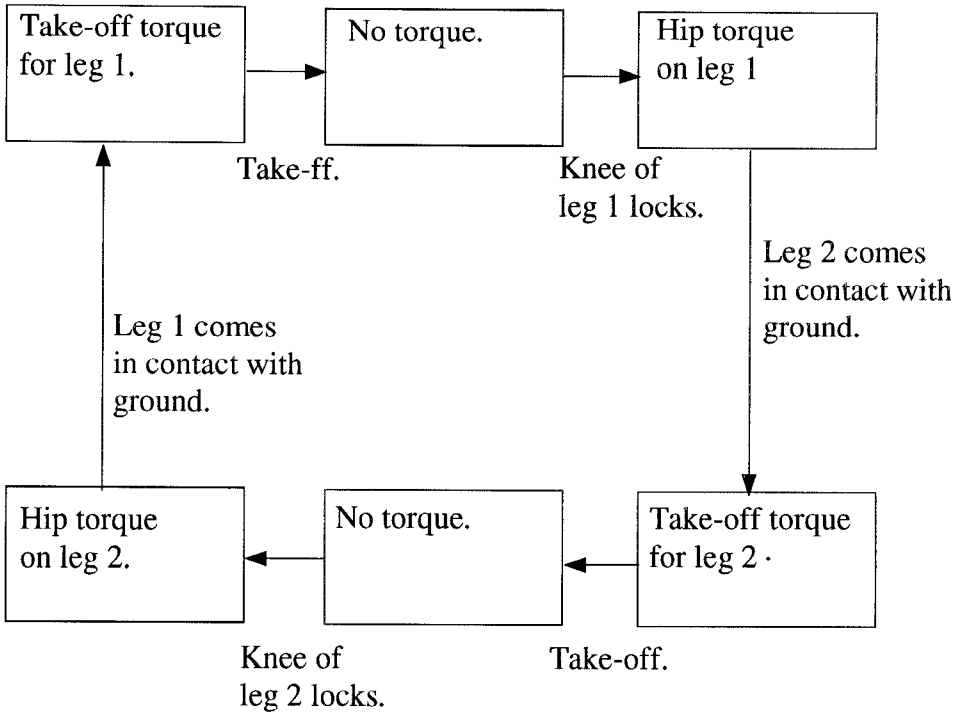


Figure 3.7.1 Transition diagram.

3.8 Stability of Walk

We would like the biped to take steps of a fixed length. Although the system is difficult to analyze if we look at the dynamical equations governing it, physical insight provides valuable clues about the type of control needed. Step length is given by the angle the legs of the biped make in the double support phase as shown in figure 3.2.1. We would like this angle to remain constant over all steps. If we consider the swing leg to be swinging freely under the influence of gravity, the more time it has, the further ahead it will swing and the greater will be the step length. The time the swing leg has depends on how fast the body weight moves on the stance leg. This in turn depends on the initial torque applied to the stance leg. The

greater the initial torque on the stance leg, the less time the swing leg has to move and the shorter the step length.

A simple feedback control strategy has been devised from this. If the step length is greater than desired, increase the initial torque applied to the stance leg and if the step length is shorter than desired decrease the initial torque applied to the stance leg.

One might be tempted to consider the swing leg as an inverted jointed pendulum, derive its time period and try to work out the time taken for swing, the step length and other parameters. However since the swing leg is coupled to the rest of the body mass at hip the values so obtained are not accurate.

In addition the walk may be destabilized by the

1. The toe of the swing leg stubbing against the ground. This is guarded against by giving sufficient clearance to the shank of the swing leg to swing through. This clearance results in the knee joint of the swing leg reaching full extension at a small height above the ground.
2. The body tipping over and falling before the swing leg has been brought to ground. Hip torque on the swing leg prevents this from happening by bringing the swing leg down to ground once it has straightened out. A similar hip torque is applied by humans at the end of swing phase.

Sensory input giving information about the state of the system is essential for stabilization.

3.9 Experimental Results

3.9.1 Biped positions

The following figure shows the positions of the biped at intervals of 0.05 second.

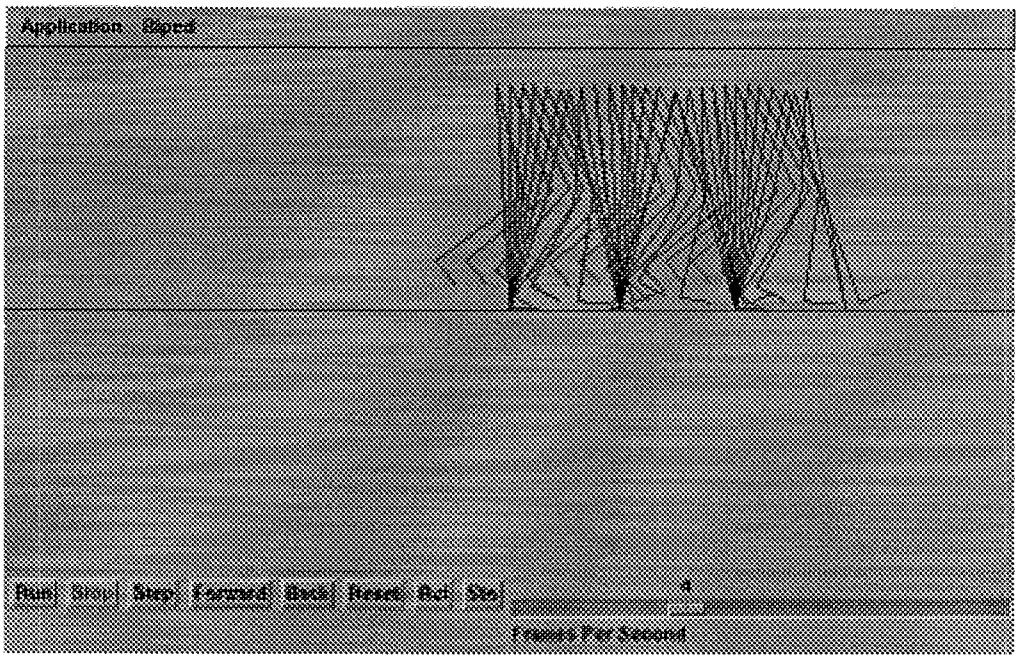


Figure 3.9.1 Positions assumed by the biped figure while walking.

3.9.2 Plots of results

The following figure shows the relationship between the output of the first oscillator and the torque on the stance leg.

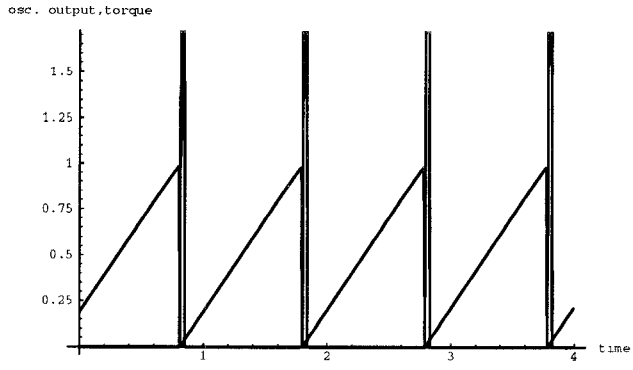


Figure 3.9.2 Output of the first oscillator and torque on stance leg.

Figure 3.9.3 shows the phase difference between the first and fifth oscillators in the actual experimental setup.

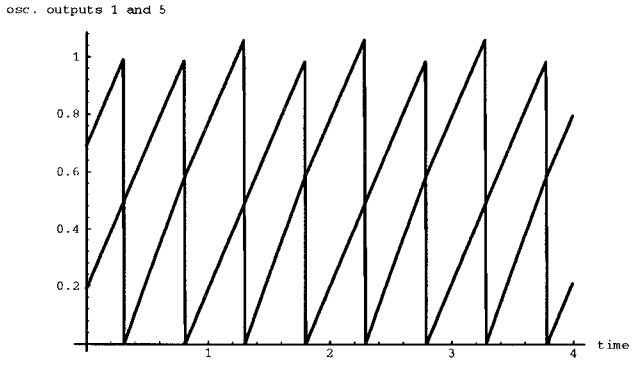


Figure 3.9.3 Outputs of the first and fifth oscillators.

Similarly the following figure shows the phase difference between the first and eighth oscillators obtained during simulation.

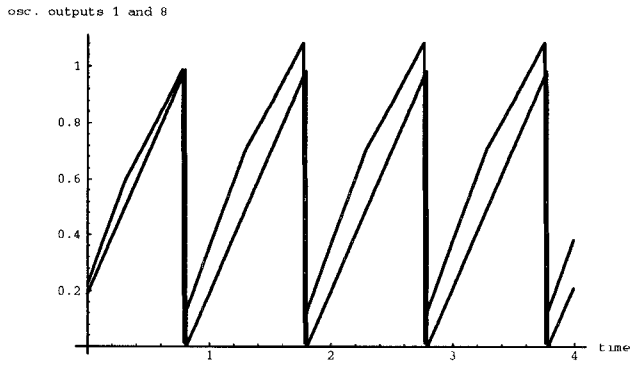


Figure 3.9.4 Outputs of the first and eighth oscillators.

The relationship between the output of the fourth oscillator and torque on the swing leg is shown in figure 3.9.5.

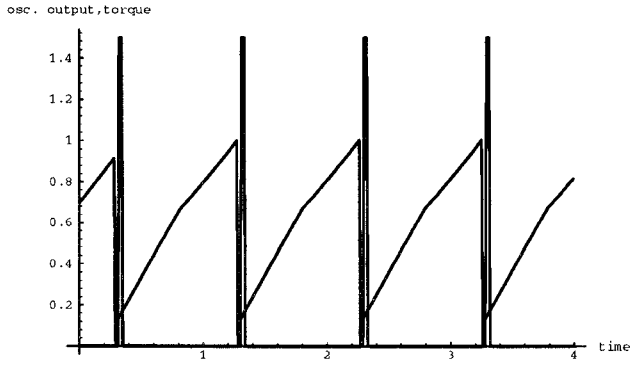


Figure 3.9.5 Output of the fourth oscillator and the torque on swing leg.

3.9.3 Movement of body and knees

The body mass at the top of swing leg moves in an arc as can be seen from figure 3.9.6.

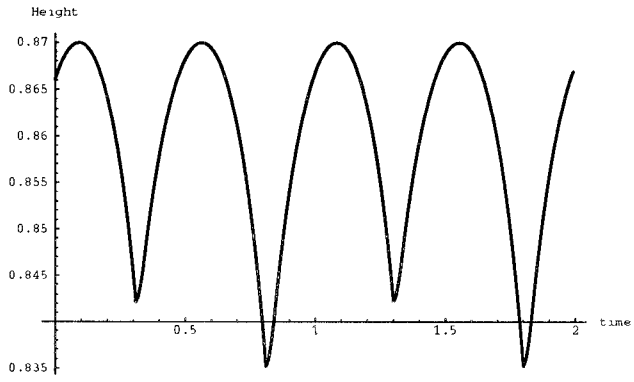


Figure 3.9.6 Height of body above ground.

The walking speed remains almost constant as can be seen from figure 3.9.7.

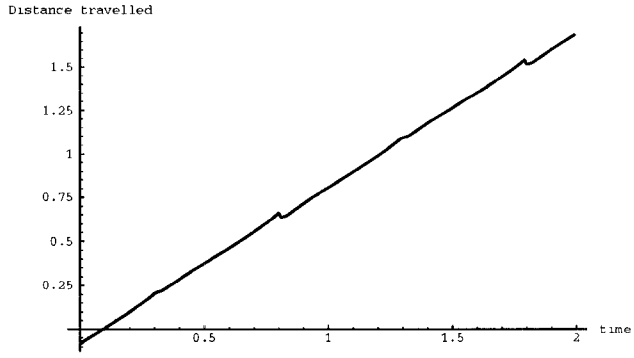


Figure 3.9.7 Distance travelled by the body mass.

The two legs maintain a fixed phase difference from each other as can be seen from figure 3.9.8. It shows the variation of heights of the two knees above ground.

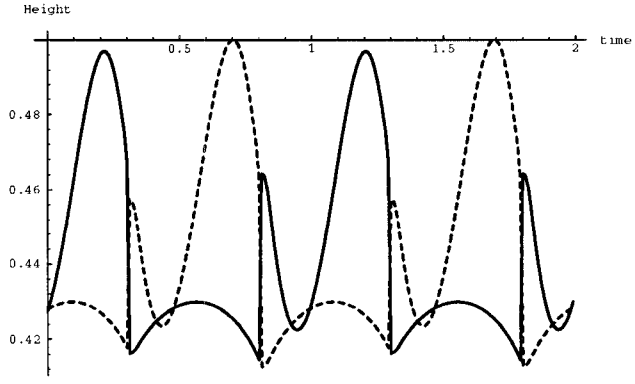


Figure 3.9.8 Height of the knees above ground.

Chapter 4 Running

In walking interchange between potential and kinetic energies played a significant role. In a run kinetic energy and potential energy reach their maxima and minima at approximately the same time. Both kinetic and potential energies are at their minimum in the middle of the support phase and both reach their maximum when the runner is off the ground. Energy is stored in an elastic form in stretched tendons, ligaments, muscles and, possibly, bent bones in running[9].

4.1 Model

If we model the reaction force from the ground can as acting on the ends of leg shanks, feet can be neglected. The upper portion of the body is considered as a point mass with two identical legs attached to it. Each leg is modelled as a chain of two links. One heavy link represents the thigh and the other the shank. The numerical values of system parameters are given in the appendix.

4.1.1 Deriving Dynamical Equations

Each of the two legs is considered separately.

The body mass is taken as the origin of the system and the reference axes are assigned as is shown in figure 4.1.1. Consider leg 1. A forward recursion is carried

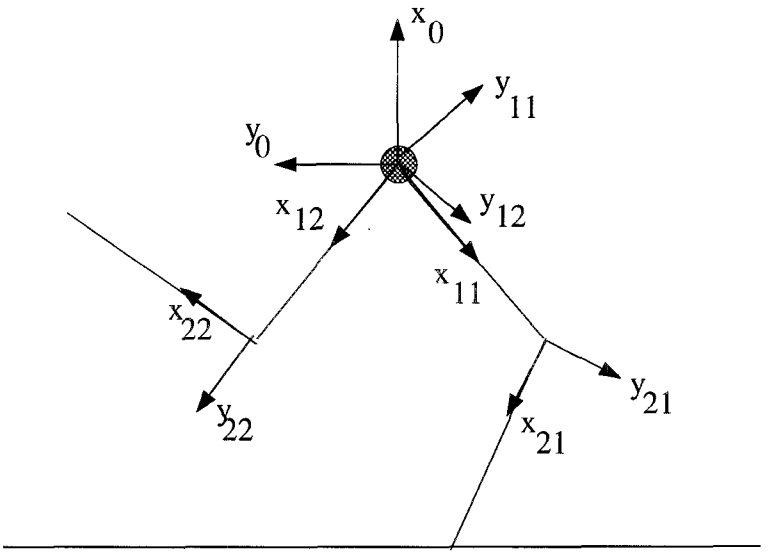


Figure 4.1.1 Reference axes

out to determine the velocity and acceleration of the center of gravity, and the angular velocity and the angular acceleration of each of the two links making up the leg.

$$
\begin{aligned}
{}^{i}\omega_i &= {}^{i-1}R_i^T\,{}^{i-1}\omega_{i-1} + e_z\dot{q}_i,\\
{}^{i}\dot{\omega}_i &= {}^{i-1}R_i^T\,{}^{i-1}\dot{w}_{i-1} + e_z\ddot{q}_i + \left({}^{i-1}R_i^T\,{}^{i-1}\omega_{i-1}\right)\times e_z\dot{q}_i,\\
{}^{i}\ddot{p}_i &= {}^{i-1}R_i^T\left[{}^{i-1}\ddot{p}_{i-1} + {}^{i-1}\dot{\omega}_{i-1}\times{}^{i-1}\hat{p}_i + {}^{i-1}\omega_{i-1}\times\left({}^{i-1}\omega_{i-1}\times{}^{i-1}\hat{p}_i\right)\right],\\
{}^{i}\ddot{s}_i &= {}^{i}p_i + {}^{i}\omega_i\times{}^{i}\hat{s}_i + {}^{i}\omega_i\times\left({}^{i}\omega_i\times{}^{i}\hat{s}_i\right),
\end{aligned}
\tag{4.1.1}
$$

where,

the superscript denotes the reference frame and the subscript the link,

ω is the angular velocity of a link,

${}^{i}p_i$ is the vector from origin of the system to joint i in reference frame i,

${}^{i-1}R_i$ is the transformation matrix from reference frame i to $i-1$,

e_z is the joint axis,

$\dot{q}_i$ is the joint velocity,

${}^{i-1}\hat{p}_i$ is the vector from joint $i-1$ to joint i in frame $i-1$ and

${}^{i}\hat{s}_i$ is the vector from origin of system to center of gravity of the link in frame i .

A backward recursion starting from the forces exerted by the ground on the leg gives us the torque acting at each joint and the force exerted by the leg on the upper body mass:

$$
\begin{aligned}
&{}^{i}\hat{f}_i = m_i {}^{i}\ddot{s}_i, \\
&{}^{i}\hat{n}_i = {}^{i}I_i {}^{i}\dot{\omega}_i + {}^{i}\omega_i \times \left({}^{i}I_i {}^{i}\omega_i\right), \\
&{}^{i}f_i = {}^{i}R_{i+1} {}^{i+1}f_{i+1} + {}^{i}\hat{f}_i \ , \\
&{}^{i}n_i = {}^{i}R_{i+1} {}^{i+1}n_{i+1} + {}^{i}\hat{n}_i + {}^{i}\hat{s}_i \times {}^{i}\hat{f}_i + {}^{i}\hat{p}_{i+1} \times ({}^{i}R_{i+1} {}^{i+1}f_{i+1}), \\
&\tau_i = e_z^{T\,i}n_i \ , \text{where i is 1,2, 3,4.}
\end{aligned}
\tag{4.1.2}
$$

Here,

${}^i\hat{f}_i$ is the total external force on link i in frame i,

${}^i\hat{n}_i$ is the total external moment on link i frame i,

${}^iI_i$ is the inertia tensor of link i in frame i,

${}^in_i$ and ${}^if_i$ are the force and moment exerted by link $i-1$ on link i

and,

τ_i is the joint driving torque.

The force exerted by each leg is summed up to give the force exerted on the upper body mass by the legs. Adding the gravitational force to this we get the total force on the body mass m_o.

$$
\begin{aligned}
&{}^0f_{m_o} = {}^0f^1_{m_o} + {}^0f^2_{m_o} + {}^0f_g, \\
&\text{where,} \\
&{}^0f_{m_o} \text{ is the total force on } m_o, \\
&{}^0f^1_{m_o} \text{and } {}^0f^2_{m_o} \text{are the forces exerted on } m_o \text{ by legs 1 and 2} \\
&\text{and} \\
&{}^0f_g \text{ is the force on } m_o \text{ due to gravity.}
\end{aligned}
\tag{4.1.3}
$$

Acceleration of the body mass can be obtained by dividing the total force by the body mass.

4.1.2 Modelling Muscles

An excellent exposition of muscle mechanics is given in [9]. A model which has become extremely useful in the analysis of skeletal muscles against load is the

"Active State Model". Here we model the muscles in a similar manner. The muscle system at each joint is considered as a spring and damper arrangement. The torque exerted by the muscle at a joint is given by

$$\tau = -k_\theta(\theta - \theta_0) - k_\omega\dot{\theta}, \tag{4.1.4}$$

where,

τ is the torque exerted,

k_θ is the spring constant,

θ is the joint angle,

θ_0 is the joint rest angle,

k_ω is the velocity constant and

$\dot{\theta}$ is the joint angular velocity.

The rest torque at each joint θ_0 and the joint stiffness k_θ are determined by the control system.

4.1.3 Ground Forces

Ground forces can be very tricky to model. Ground forces result in abrupt, large forces being applied to the system. These forces produce large changes in the state of the system. The time interval between two successive iterations has to be made small to preserve accuracy.

Here ground has been modelled as a two dimensional damped spring. The place where the leg first strikes the ground, (x_o, y_o), is considered as the rest position of the spring. Horizontal and vertical forces exerted by ground on the end of leg are

calculated as follows:

$$F_x = -k_x(x - x_o) - k_{vx}\frac{dx}{dt}, \qquad Fy = -k_y(y - y_o) - k_{yx}\frac{dy}{dt}, \tag{4.1.5}$$

where,

F_x, F_y are the horizontal and vertical forces exerted,

k_x, k_y are the spring constants in horizontal and vertical directions,

k_{vx}, k_{vy} are the damping constants in the horizontal and vertical directions.

These spring constants also take into account the elasticity of the leg. This elasticity is present in the tissue of the landing leg's foot and at the knee joint.

This model serves two purposes. It allows us to calculate ground forces. Moreover, if the spring constants are chosen correctly, ground forces exerted are not too large. This allows the use of a larger time interval between successive iterations enabling real time simulation.

4.2 Stages of Running

Running is very similar to hopping with the unnecessary leg being kept out of way. Running can be divided into various stages.

4.2.1 Impact

The leg which is to land is brought in proper position. On impact with the ground, it bends at the knee. Part of the energy is stored in the compression of the torsional spring at the knee joint.

4.2.2 Take-off

Torque is applied to open out the knee joint. The decompression of the compressed torsional spring at the knee aids in take-off.

4.2.3 Flight

The leg which is to take the weight is brought forward. The applied torques swing this leg forward at the hip and lower its shank. The leg through which takeoff just took place is brought up so that it does not interfere with the leg about to land.

4.3 Control of Speed

The result of ground impact on speed depends largely on the position of the leg which is to take the weight of the body. When the leg hits the ground, a reaction force acts on the shank of the leg. The leg tends to bend at the knee joint. If torque is now applied at the knee joint to open it out, the tendency to bend is counteracted. The knee joint tends to open out pressing the shank against the ground and accelerating the body mass. The body mass accelerates upwards. It moves upwards and takeoff occurs. The horizontal component of the reaction force depends on the orientation of the landing leg. If the leg is bent over backwards the body mass is accelerated in the backward direction. The body mass is accelerated in the forward direction if the configuration is such that the body mass is tipped in the forward direction. Whether the body mass will gain or lose speed due to impact depends on the average of horizontal acceleration during impact.

Collision with the ground results in a loss of energy. Energy is lost in the dampers of the springs used to model the elasticity of the ground and the leg.

The body mass is moving forwards when the impact occurs. If the leg is bent over backwards when impact occurs, the initial acceleration will be in a backward direction. However due to the forward velocity of the body, the body mass will

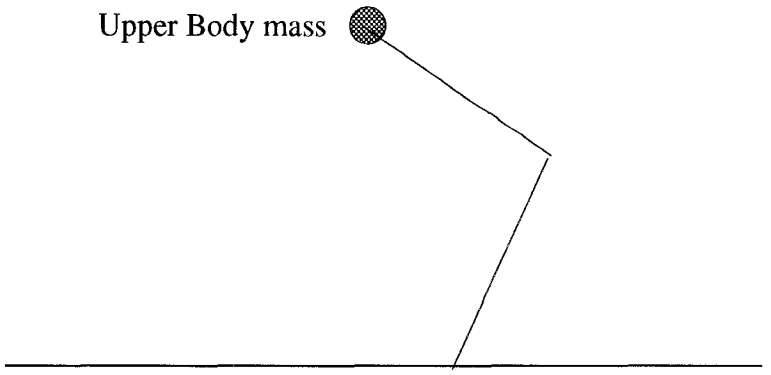


Figure 4.3.1 Landing Configuration.

tip over and acceleration will be in the forward direction. If the average force on the body mass is directed upwards and forwards the leg accelerates in the forward direction. If the average force is directed upwards and backwards, the forward speed is reduced.

What the average component of reaction force in the horizontal direction will be depends on the position of the leg at landing and the time the leg is in contact with ground. For a given speed of running, at a particular configuration of the landing leg, the reaction force is such that the horizontal speed before this impact and before the next impact is unchanged . Configurations of the leg which lie ahead of this equilibrium configuration result in acceleration and those which lie backwards result in deacceleration as shown in figure 4.3.2 This provides a convenient method to

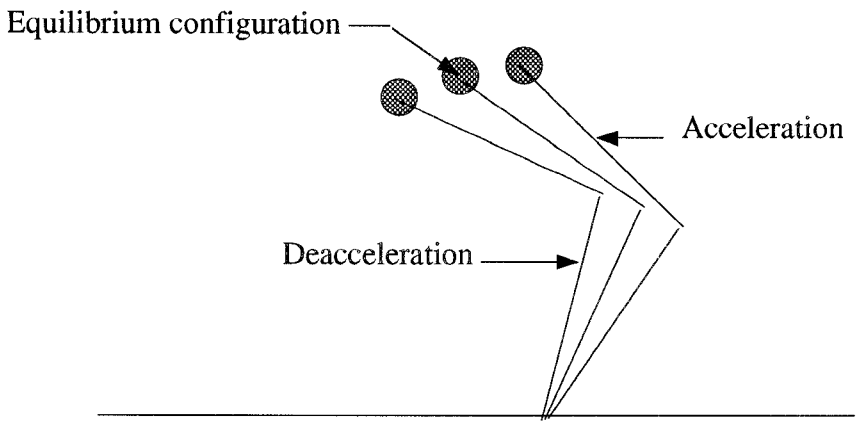


Figure 4.3.2 Acceleration and deacceleration during impact

control the speed of running — to increase speed in the forward direction place the landing leg ahead of the equilibrium configuration, and to decrease speed in the forward direction place the landing behind the equilibrium configuration.

The spring constants determine how fast the legs will swing to their new positions after take-off. The spring constant which controls the position of the thigh of landing leg is particularly important since it controls the orientation of the landing leg. A feedback control mechanism is used to determine this spring constant. At each landing, the difference between the desired value and actual value of the joint angle of the thigh of the landing leg is used to make modifications in the corresponding spring stiffness. If the actual value is less than the desired value, the swing leg has not moved far enough and the spring stiffness needs to be decreased, and vice versa.

4.4 Control

Ideally one would like to analyze the equations governing the system and to draw conclusions about system behavior and the control needed to achieve the desired objectives from such an analysis. However the complexity of the dynamical equations

governing such a system and the difficulty of analyzing impact lead us to search for other strategies.

Daily contact with the physical world has over the years built within us a remarkable degree of intuition about rigid and flexible bodies, torques, forces and impacts. We have through the pressure and force sensors present in our bodies processed a large amount of information about such systems. Our intuition can provide us with insights into the behavior and control of the system. These can then be checked against results obtained by simulating the derived equations.

For proper motion we must ensure that legs are brought into right positions for landing, that takeoff occurs properly and that motion is stabilized. By stability we mean the ability of motion to continue indefinitely in the forward direction with a uniform speed and step length. Each of these aspects of control are considered separately.

4.4.1 Takeoff Torque

The take-off torque is applied after landing. In the real world the take-off torque is applied until the time the body leaves the ground. We have modelled the ground as a spring. If takeoff torques are applied till the time the body loses contact with ground, too high a vertical speed is achieved. Hence takeoff torque is applied only till such a time that the upward velocity of the body is below a certain speed. Even if the desired upward velocity has not been achieved no torque is applied if the leg in contact with the ground has straightened out. This prevents the knee joint from being bent over backwards.

4.4.2 Leg Positioning

After takeoff the leg which is to take the weight of the body is to be brought forward while the other leg is to be kept out of the way. To accomplish this the joints are modelled as spring and damper arrangements with adjustable rest positions. After takeoff the rest positions of the legs are interchanged. This results in appropriate positioning of the legs.

4.4.3 Stability of Motion

Takeoff torque is exerted only while the vertical velocity is below a certain value. This provides us with a nearly uniform vertical velocity at all takeoffs. We now only need provide control for the forward speed at takeoff. This is largely dependent on the approach angle of the leg which is to take the weight of the body. A simple feedback mechanism as described in section 4.3 sufficed to ensure that the landing leg hits the ground with a consistent approach configuration.

Striking the ground with a consistent approach angle will result in a nearly uniform forward velocity. If at the time of impact the biped is moving with a forward velocity higher than the equilibrium velocity for impact, the backward reaction it will suffer will also be correspondingly higher and it will get slowed down. The reverse happens if it is moving slower than the equilibrium velocity for the time of impact.

Controlling the forward takeoff velocity for a given vertical takeoff velocity effectively results in control of step length and the time between steps. For a given takeoff velocity the time before next impact can be approximated as follows:

$$t = \frac{2v_{v,takeoff}}{g}, \tag{4.4.1}$$

where,

t is the time between takeoff and impact,

v_v is the vertical takeoff velocity and

g is the acceleration due to gravity.

Step length can be calculated from this and the forward velocity.

4.5 Control using oscillators

Take one cycle to consist of two consecutive steps. Then there are four tasks to be accomplished in each cycle — take-off torques have to be exerted for each of the two legs and after take-off rest positions of the joints have to be switched so that the legs are brought into proper position for the next step.

Control can be accomplished using a network of three oscillators as shown in figure 4.5.1. The oscillators and their coupling are the same as discussed in the

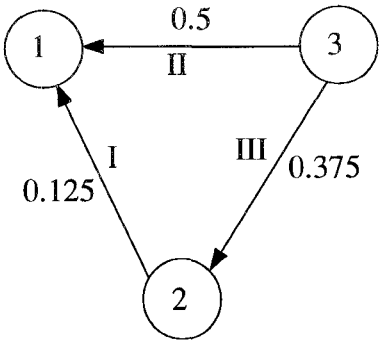


Figure 4.5.1 Network of oscillators to control running

previous chapter. Oscillator 1 is responsible for take-off torque at the knee joint of leg 1. When leg 1 hits the ground a pulse is sent to oscillator 1 causing it to fire and reach ground state. Take-off torque is exerted while the output of oscillator 1 is below a certain value. When the upward velocity of the body exceeds a set value another pulse is sent to oscillator 1 causing its output to go out of the range in which

torque is exerted. A similar mechanism through oscillator 3 results in take-off torque being applied on leg 2. Rest positions of joints are controlled through oscillator 2. Oscillator 2 is reset upon take-off by leg 1. While the output of oscillator 2 is less than half of its maximum output the rest positions at the joints are such that leg 2 is the landing leg. Take-off by leg 2 causes the output value of oscillator 2 to cross the halfway mark, the rest positions of the joints are switched and leg 1 is now brought forward for landing.

The incidence matrix for this graph,

$$A = \begin{bmatrix} -1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{4.5.1}$$

The desired values of phase differences for the oscillators are

$$\theta_{desired} = \begin{bmatrix} 0 \\ 0.125 \\ 0.5 \end{bmatrix} \tag{4.5.2}$$

Using the method discussed in section 2.4.1, we choose

$$V(\phi) = k(\theta - \theta_{desired})AA^TAA^T(\theta - \theta_{desired}), \tag{4.5.3}$$

where k is a positive real number determining the strength of the coupling. Hence,

$$\dot{\theta} = \omega - AA^T(\theta - \theta_{desired}). \tag{4.5.4}$$

The following graph shows oscillators with a normalized angular velocity settling down to the set phase differences.

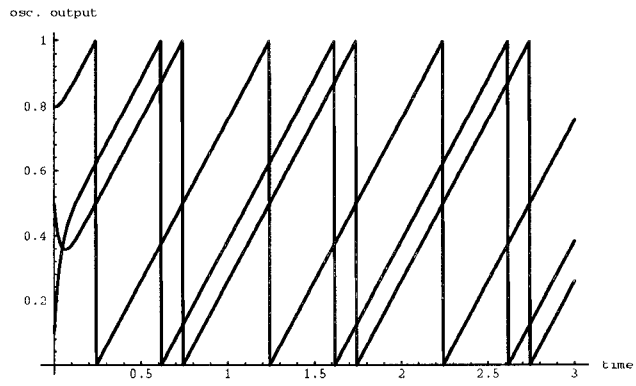


Figure 4.5.2 Oscillators settling down to set phase difference.

4.6 Finite State Model

A finite state model similar to the one constructed for walking can be constructed for running. The input can be taken to consist of

1. Whether leg 1 is in contact with ground,
2. Whether leg 2 is in contact with ground and
3. Whether the vertical velocity of the body mass is less than the desired vertical velocity.

The eight resulting combinations constitute the input alphabet. The set of internal states of the system can be taken to consist of the eight combinations of

I. Whether leg 1 is the landing leg,

II. Whether take-off torque is applied at the knee joint of leg 1 and

III. Whether take-off torque is applied at the knee joint of leg 2

The output alphabet corresponds directly to the set of internal states since I determines the joint rest angle and II and III determine the take-off torques.

The following table shows how the states change in response to input. The system remains in its current state till input corresponding to the next state in the table arrives.

Is leg 1 in contact with ground ?	Is vertical velocity of body mass < desired ?	Is leg 2 in contact with ground ?	Is leg 1 the landing leg ?	Is take off torque being applied at knee joint of leg 1 ?	Is take off torque being applied at knee joint of leg 2 ?
0	1	0	1	0	0
1	1	0	1	1	0
1	0	0	1	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	0	0	1
0	0	1	0	0	0
0	0	0	1	0	0

Table 4.6.1 Finite state model for running

4.7 Experimental Results

4.7.1 Positions of the biped figure

Figure 4.7.1 shows the position of the biped at fixed intervals of time. As can be seen from the figure, the landing leg sinks into the ground. This is because the ground has been modelled as a spring of rather low stiffness.

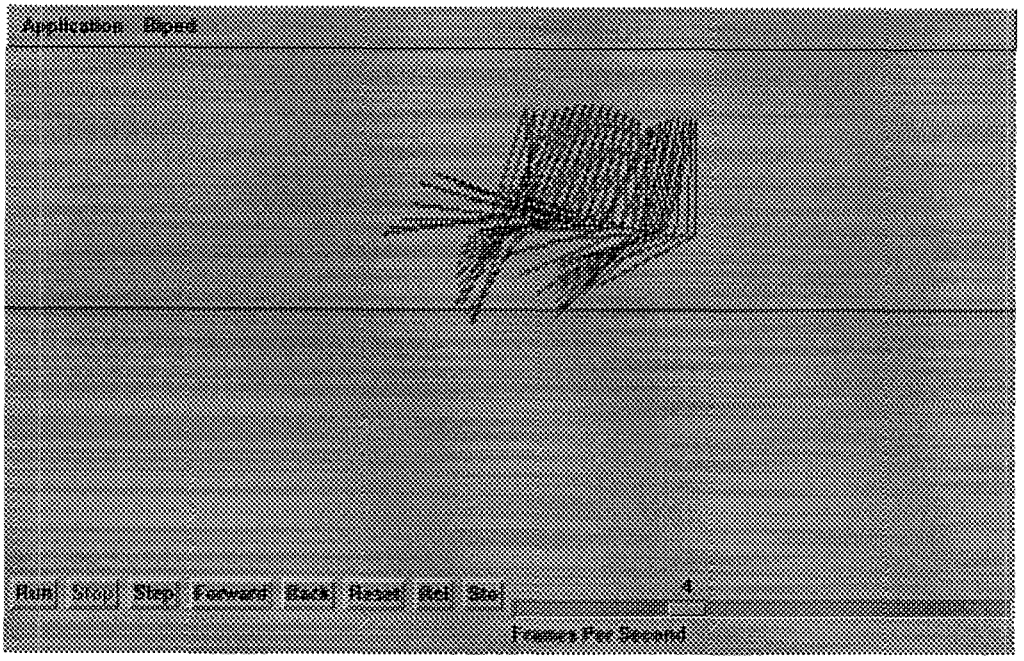


Figure 4.7.1 Positions assumed by the biped while running.

As in walking the body mass at the hip moved in an arc as shown in figure 4.7.2. The bump in the arc is due to the slight rebound from the ground when the body mass strikes the ground. A nearly constant running speed is maintained as

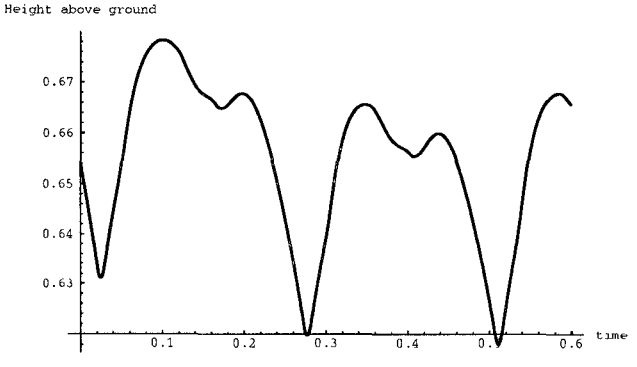


Figure 4.7.2 Height of the body mass above ground

shown in figure 4.7.3.

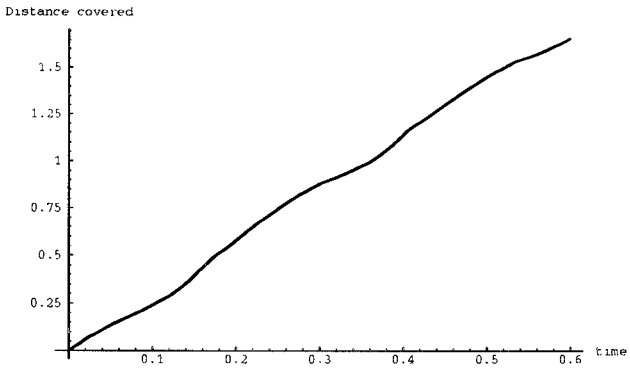


Figure 4.7.3 Constant running speed

The two knees maintain a fixed phase difference as can be seen from figure 4.7.4.

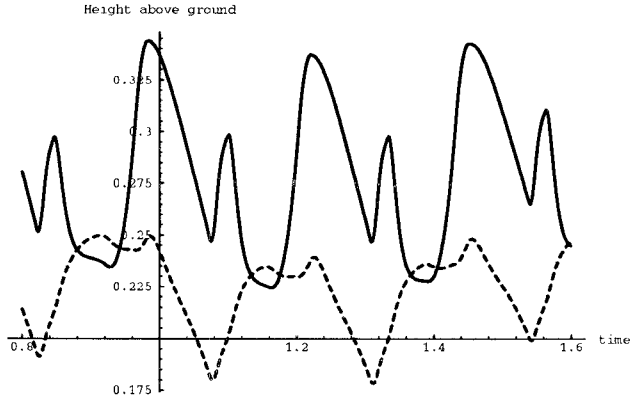


Figure 4.7.4 Fixed phase difference between two knees

4.7.2 Plots of results

Figure 4.7.5 shows the output of the second oscillator and the corresponding knee torque. The pulses in the torque correspond to the take—off torque. When take-off has occurred the oscillator receives a pulse causing a sudden jump in its output. Its output is no longer in the range for which torque is to be applied. Torque

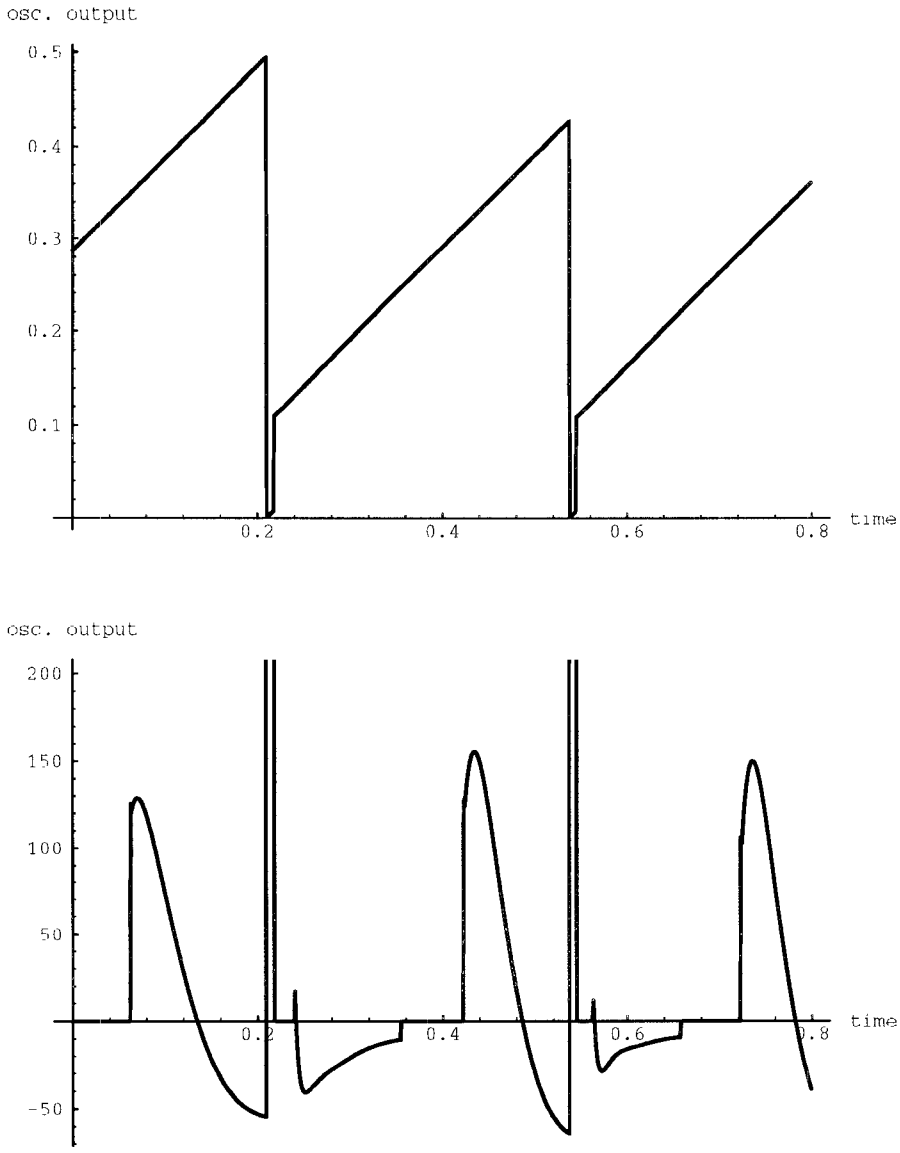


Figure 4.7.5 The output of second oscillator and take-off torque.

other than take-off torque corresponds to the torque applied to bring the leg into appropriate rest positions.

Figure 4.7.6 shows the output of the third oscillator and the rest position of the thigh of leg 1. When the leg is about to land the rest position is greater than negative

π so the leg is out in front. When leg 1 is not the landing leg, the rest position is such that leg 1 is kept out of way of leg 2.

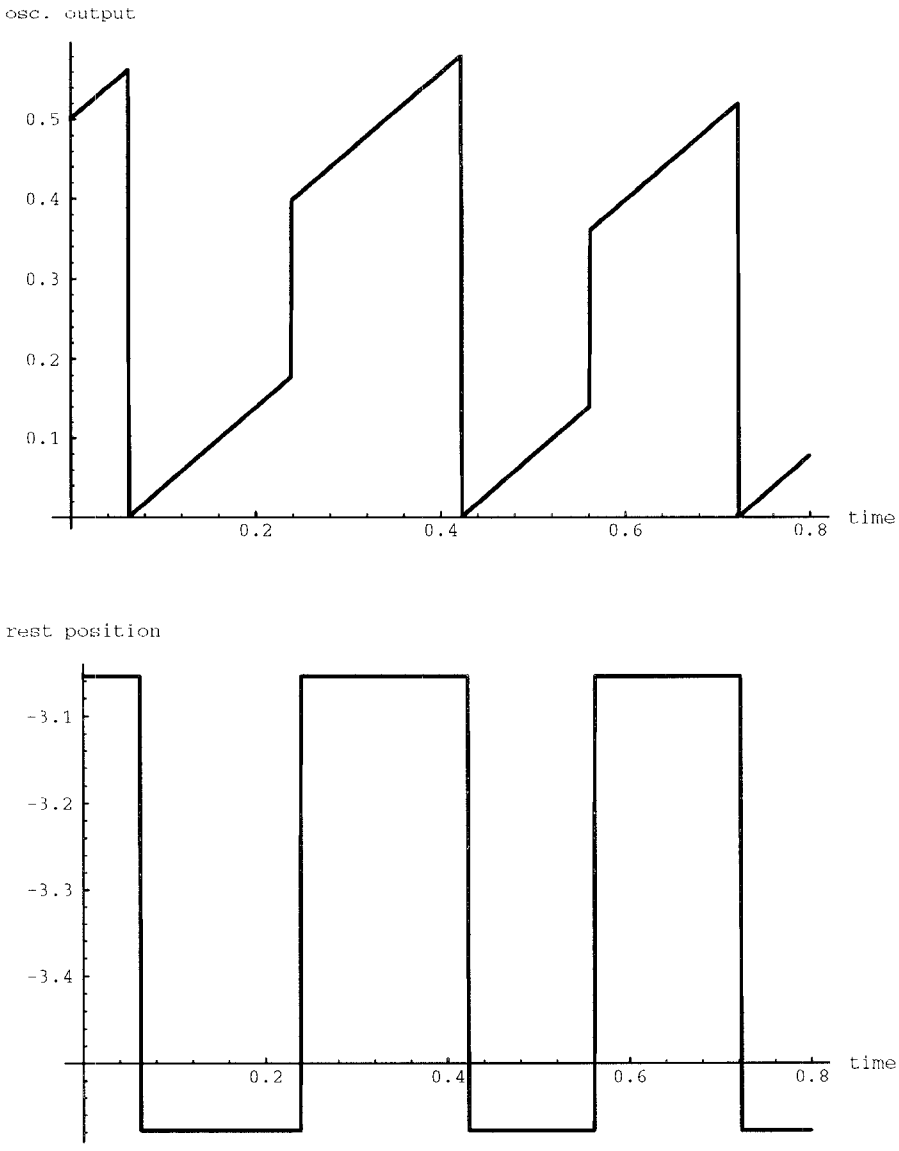


Figure 4.7.6 The output of third oscillator and rest position.

Chapter 5 Simulation

The complexity of the dynamic models involved necessitates reliance on numerical simulation for insight into behavior. Such simulation can be decomposed into

1. Dynamic Model Generation
2. System Simulation

5.1 Derivation of Dynamic Model

The first step in simulation is the actual derivation of dynamical equations. Derivation by hand is time-consuming and prone to error. To overcome these limitations packages like Mathematica and MACSYMA, which are capable of symbolic computation, are used.

The equations used here were derived using Mathematica. The methods attributed to Newton and Euler, Lagrange and Kane are three commonly used techniques for deriving dynamical equations. Although only equations generated using Newton-Euler's method were used, programs were written to generate the equations using all three methods. Programs using Lagrange's method got stuck in symbolic computation for systems with more than four degrees of freedom. Programs using the Newton-Euler method and Kane's method were able to derive the equations for systems with six degrees of freedom as needed. Equations generated by programs based on Lagrange's and Kane's methods provided a useful check on the equations generated by the programs based on the Newton-Euler method.

Although Kane's method gives the dynamical equations directly, the Newton-Euler method was used because of the physical insight it affords. It generates forces

and torques exerted by one link on another for intermediate calculations. These provide valuable insight into system behavior.

An initial run with Mathematica generates the equations in a very long form. Computation using these equations would be time consuming since many extra terms would have to calculated. These equations must be simplified. We can use the "Simplify" function of Mathematica to do so. Simplification becomes faster as the number of symbols decreases. Hence, before simplification numerical values of various parameters such as lengths of links, the position of their centers of gravity, etc. were substituted.

5.2 Simulation

To simulate the system we must solve the dynamical equations governing the system and display the result in a readily understandable form. To arrive at a proper control system, repeated simulations are required. This requires that our simulations take place in real time and that it be relatively easy to change the dynamic model. The set-up for the simulation met both of these requirements.

Simulation was done on two machines. Dynamic equations were solved using Mathematica on one machine. These results were communicated to an interface program running on another machine which used these results to produce an animation. The structure of the simulation program is as shown in figure 5.2.1

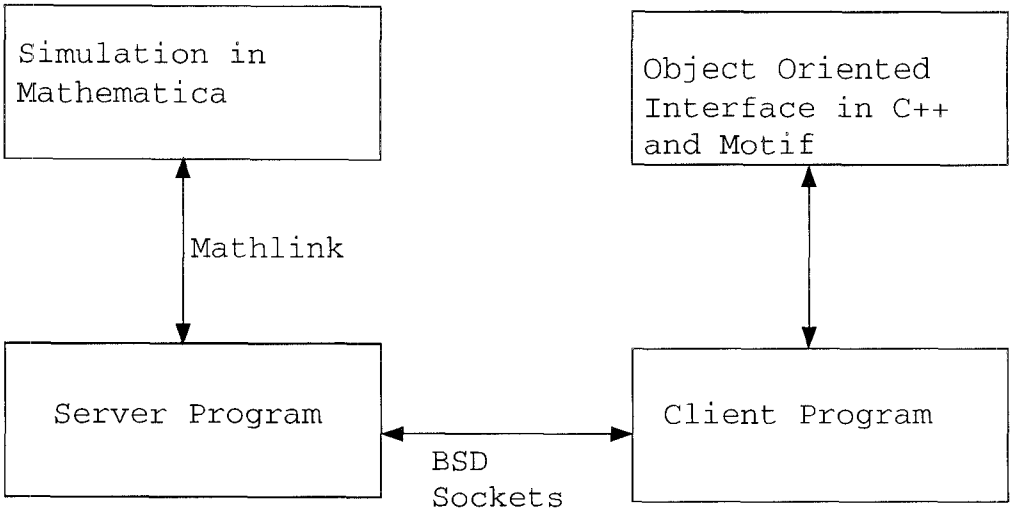


Figure 5.2.1 Structure of the Simulation

At first the user starts up the communication server on the Mathematica machine. The animation program is then started up. The animation program sends a request to the server. The server then starts up Mathematica. All the necessary equations and simulation routines are placed in the "init.m" file so that they are automatically loaded at start-up. The server sends a request to Mathematica to give the result after next iteration. A data structure in the relay program is filled up with the received result. This data structure is then transmitted to the animation program using the Berkeley sockets mechanism.

The animation program uses the received data structure to update the state of the animation. It also receives commands from the user. Depending on the command, the animation program either carries out the command or conveys it back to Mathematica for action.

5.3 Dynamical Equation Solver

Initially animation was carried out through C programs on an IRIS machine. The dynamical equations were solved through the fourth order Runge Kutta method using the math libraries available. The dynamical equations generated by Mathematica are in a form suitable for use by it. These must be processed further before they can be used by a C program. Although Mathematica provides such a function, "CForm", the function has certain limitations. Trigonometric and power functions generated by it do not match the version of C used on IRIS machines. These had to be edited before use.

A problem was encountered in the compilation of these programs on IRIS machines. The equations generated are long and have many terms. The IRIS C compiler cannot handle such long equations. Long expressions had to be split up before they could be compiled.

Converting dynamical equations into C form, editing them and compiling them is a time consuming process. This becomes especially cumbersome when different models of the system are to tried out. It was felt that there was a need for a more efficient way to go about performing the simulation. Instead of performing the simulation through a C program, simulation was performed in Mathematica.

To achieve speed of simulation on Mathematica, use was made of the compilation facility available. This facility can provide speedups up to twenty to thirty times [19].

5.4 Mathlink

Mathematica cannot directly communicate with the animation programs on another machine as it would require Mathematica to be installed on both machines.

This difficulty was overcome by having Mathematica communicate to a program on the same machine using a built-in Mathematica package known as "Mathlink". This intermediate program then relays the information to the animation program on another machine.

Mathlink is a general high-level communication mechanism that allows data and commands to be exchanged between Mathematica and external programs. It provides mechanisms for external programs to call Mathematica, and to be called by Mathematica [15]. Here we have used Mathlink to implement our own front-end for Mathematica. Mathlink is based on the Berkley sockets facility.

5.5 Inter-process communication

The Berkley sockets mechanism is a generalization of the file system of Unix and is designed to incorporate network protocols. It allows applications to choose among different available protocols and to determine the destination each time sockets are used. This interface to the various protocols was used to set up communication between the simulation running on a Hewlett Packard workstation in Mathematica, and the animation running in X windows on another machine. The server program running on the Mathematica machine had Mathlink and Berkley sockets calls interwoven together so as to effectively function as a relay. Here we describe the Berkley sockets portion of the server. The work done was influenced by [2].

5.5.1 Server

To set up communication, the server creates a socket, *ls*, using the "socket" system call. This socket is called an unbound socket because it has no addresses

associated with it. The operating system cannot demultiplex datagram packets to the correct socket without an address. The bind routine binds an address to the local end of the socket.

Sockets can be of various types, e.g. SOCK_STREAM and SOCK_DGRAM. A SOCK_STREAM provides sequenced, reliable, two way connection based byte streams. A SOCK_DGRAM supports datagrams (connectionless , unreliable messages of a fixed maximum length). SOCK_STREAM type sockets were used in the programs to establish communications.

Servers accept connections from remote clients. The listen system call is used to establish a queue for incoming connections. The listen system call has a parameter for specifying the maximum allowable length of the queue for pending connections. Since we were interested in having only one simulation running at a time, the length of the queue was set equal to one.

The "accept" system call is used to actually open connections. It extracts the first connection on the queue of pending connections, creates a new socket, *s*, with the same properties and allocates a new file descriptor for the socket. The new socket created cannot be used to accept more connections. The old socket remains open after the accept system call. In our server program, since the old socket is not needed anymore, the old socket is closed using the "close" system call.

After this the server enters the server loop. It receives the command from the client using the "receive_structure" subroutine. This subroutine uses the "read" system call to get the command sent by the client. After reading the current command the program relays to the animation program the results of the previous command.

The current command is then relayed to Mathematica using Mathlink and the results obtained stored in a data structure.

As can be seen above the receipt of a command by the server prompts it to display the results of the previous commands. This is so as to have a certain degree of parallelism in display and calculation. While Mathematica is evaluating the results of the current command, the animation program is displaying the results of the previous command.

5.5.2 Client

The client program on the graphics machine use similar subroutines. It uses a "connect_sky" subroutine to connect to the host running Mathematica. This subroutine uses "gethostbyname" system call to get the Internet address corresponding to the name of the host machine running Mathematica (in our case "skylab").

Advantage is taken of the same representation of floating point numbers on both workstations to avoid complicated formatting routines. The data structure is sent and the other end fills in its own data structure. This system call returns a "hostent" structure. This structure is used to specify the address of the host running Mathematica.

Connections are initiated by the client using the "connect" system call. This call returns with a value of zero on success and a value of negative one on failure. To open a connection to the remote host the port number corresponding to the server socket is needed. A specific predefined port number, 22370, is used to set up communications here. Port numbers below a certain value are reserved for system connections. The package won't work if the port 22370 is already in use. Since the possibility for

this is miniscule it was decided to go ahead with this scheme rather than to set up complicated schemes to get over the problem.

5.6 User Interface

The state of the system whether it walking or running cannot be readily visualized without an animation. Animation was done using the X window system. The following description of the X window system is based on [20] and [21].

5.6.1 The X window system

The X window system is a standard windowing system that provides a portable base for applications with graphical user interfaces. Motif is a higher level toolkit that provides common user interface components needed by X based applications.

The architecture of X is based on a client-server model. A single process known as a server is responsible for all physical input and output devices. The server creates and manipulates windows on screen, displays text and graphics and receives all input. Applications based on X never draw anything directly to the screen, but instead request the X server to perform the drawing. Similarly, X applications never receive input directly from the keyboard or mouse. Instead applications rely on the X server to inform them when input is available from these devices. By handling all input and output, the server provides a portable layer between applications and the display hardware. X-based applications can display windows, text or graphics on any hardware that supports the X protocol.

A client is an application that uses the facilities provided by the X server. A client communicates with the X server via a network connection. Multiple clients can be connected to a single server connection and an individual client can also

connect to one or more servers. The X server is a process, as are the clients. The server and the client can execute on the same machine or on different machines.

5.6.2 Requests and Events

When a client wishes to use a service provided by X it sends a request to the X server. The X server processes the requests from any given client in the order in which the client makes the requests. However, the server queues all requests and may not process them immediately. Since the server and the client run asynchronously from each other it is not possible to guarantee the order in which the server will process requests from multiple clients.

The server notifies each client when input is available by sending the client an event. X presents each event to the clients as a union of C structures. Each event indicates the specific type of event and also supplies additional information about the event.

All X applications must be designed to poll the event queue and respond to events as quickly as possible, for several reasons. One reason is that X windows are not persistent — they lose their contents if they are moved, resized, covered and then uncovered. X maintains a background pattern for each window and an optional border around the window. Anything a client chooses to display inside the window is transient and can be lost. X notifies applications by sending an event to the client whenever this happens.

5.6.3 Windows

The X server's primary responsibility is to display and manipulate windows. A window in X is a region on the screen. The server creates, destroys and manipulates

windows at the request of a client. Each window has a unique structure which the client must provide in all requests to manipulate that window.

Windows in X form a hierarchial structure. When the X structure starts it creates one special window, called the root window, that covers the entire screen. All other windows are either children of the root window or direct descendents.

When the server first creates a window it is not visible. Clients can request the server to display a window on the screen by issuing a map request. Windows can be removed from the screen by issuing a unmap request. Windows can be manipulated whether they are mapped or not.

Windows are created at the request of an individual client and go away when that client exits or disconnects from the server. While a window exits any client that knows that window's ID can request the X server to manipulate the window.

5.6.4 Window Managers

In X, the term window manager refers to an X client that allows the user to manipulate each application's topmost window. An X window manager is simply a client that uses the server to manipulate other clients' windows. All window managers are expected to obey a specific protocol for interaction with clients. This protocol is documented in Inter-Client Communications Conventions Manual (ICCCM).

5.6.5 X based libraries

Our application uses three distinct libraries. The highest level library is the Motif widget library, which contains common interface components such as buttons, scrollbars and menus. The second library is the XtIntrinsics, which provides some

common facilities needed to construct the components defined by Motif and defines the architecture used by all widgets. The lowest library is the Xlib C language interface to X, which provides some primitive operations that allow applications to create windows, draw text and simple graphics and so on. Figure 5.6.1 shows the relation of the libraries to one another.

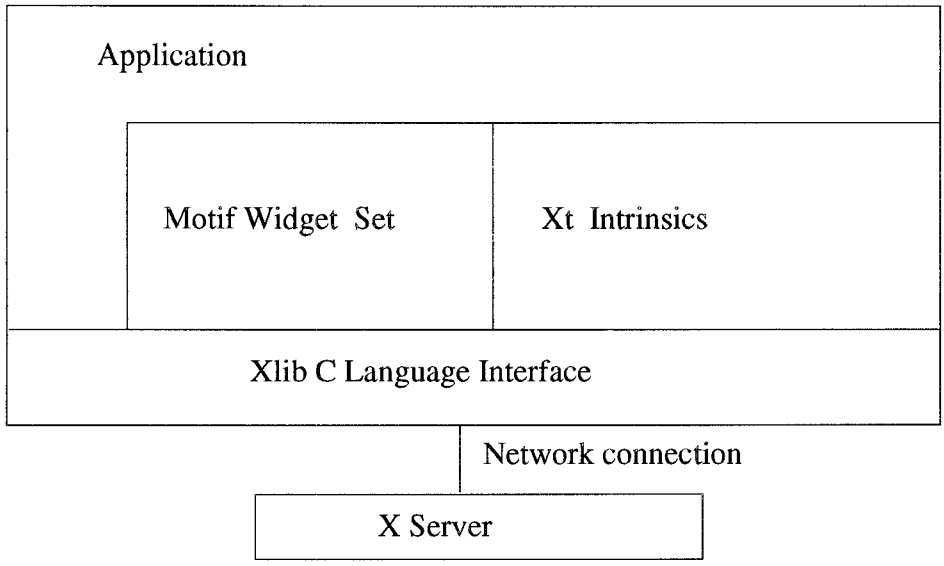


Figure 5.6.1 Architecture of the X window system

5.6.6 Widgets

The XtIntrinsics is an object oriented library written in C built on top of Xlib. Xt and the widget sets built on top of it reduce the tedium of programming in Xlib. Xt defines user interface components known as widgets. Each widget in X has a window associated with it. A widget consists of a structure that contains data and supports procedures that operate on this data.

5.6.7 Programming steps

All Xt Intrinsics based toolkits require programs using them to follow a definite sequence of steps. These steps are:

1. **Initialize Xt.** This step initializes Xt internal data structures, opens a connection to the X server and loads a resource database.
2. **Create a shell.** All applications must create one or more shell widgets to act as containers for all other widgets in the program.
3. **Create widgets.**
4. **Add callbacks.** Callbacks are functions which perform some action in response to user input.
5. **Manage widgets.** Widgets form a hierarchy. Widgets that have children act as containers that group other widgets into some logical collection. Each container widget determines the size and position of its children. This process is known as managing widgets.
6. **Realize widgets.** This causes the widgets to display themselves on screen.
7. **Enter event loop.** Xt applications are event driven — all actions are performed in response to some event.

5.7 Object Oriented Design

A new paradigm for thinking about systems, object oriented design, has emerged and gained popularity in recent years. The steps involved in object oriented design are

1. Identify the basic components of the system as classes,
2. Define the member variables and functions of each class,
3. Separate the implementation of each class from its interface,
4. Restrict the access to member which define the implementation of the object type and

5. Where two or more objects have something in common to derive them from the common object type.

Languages and programming techniques have evolved to support this design method.

5.8 Object Oriented Programming

Object oriented programming follows from object oriented design. Object oriented programming techniques differ from procedural programming techniques in grouping code and data together into "objects", in restricting access to data and in using inheritance.

Abstract data types group logically related data and code into user defined types. These types are treated the same as built in types. Separation of the interface from the implementation allows for alternative implementations of abstract data types [3]. Object oriented programs implement abstract data types as *classes*. These are templates from which objects are created and are treated just like built in types. They define a set of attributes and operations. Instances of classes are automatically initialized.

Inheritance provides various benefits such as reduced code duplication and a greater structure in design. Base classes implement functionality which is shared by one or more derived classes. New derived classes can be defined in terms of their difference from base classes. The ability of programs to treat many different forms of a class as if they are one is known as *polymorphism*. Polymorphism shields the programmer from details of derived classes. Having more than one base class for a derived class is known as *multiple inheritance*.

Abstract data types, inheritance and polymorphism form the core of object oriented programming.

5.9 C++

Although object oriented programming has been done in C (the X toolkit) using a mechanism of public and private header files to provide insulation between the implementation and the interface with the user, this is not entirely safe. C++ provides inbuilt support for object oriented programming. It was designed to allow existing C programs to be used with ease alongwith programs written in it. This has greatly contributed to its current popularity. Before using C functions they must be declared to be C functions using the extern "C" command and be provided with prototypes.

C++ differs from C in providing special features which support object oriented programming and in providing various enhancements. Object oriented programming in C++ revolves around classes and objects, member variables and functions, public and private members, constructors and destructors, derived classes, access control and polymorphism. Functions defined in the base class can be redefined in derived classes. This allow derived classes to be customized. In C++ a function can be declared *virtual* causing the correct derived class version of the function to be chosen at run time. A virtual function which has no body in the base class and must be defined in the derived class is called a *pure virtual function.* An *abstract class* is itself never instantiated but used only to derive further classes.

C++ enhancements include a new comment style, a different input output library, allowance for variable declarations to be included anywhere within a block and allowance for parameter declarations to be included inside function headers. Further

enhancements include provision for declaration of constants using the typed const command. C++ allows constant pointers, reference variables, reference parameters, inline functions, constant parameters, function prototypes and default parameter values. Overloading of functions and operators proves very useful.

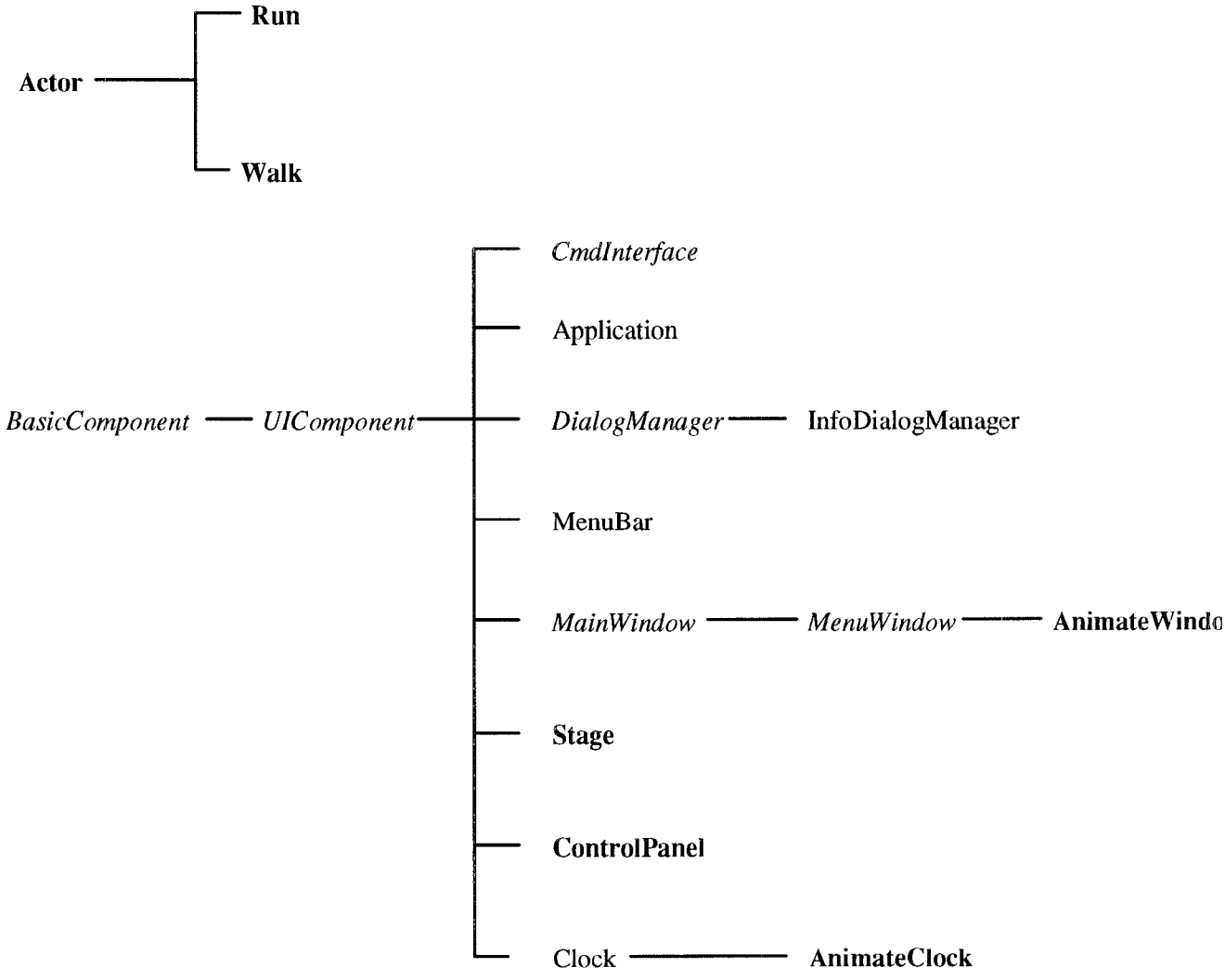
5.10 Structure of the Interface

The application is based on the MotifApp framework developed by Young in his book [21]. The application program is an adaptation of the Bouncing Ball program in the same book.

The user interface displays the motion of the biped, both running and walking. A control panel allows the user

1. to start and stop the animation,
2. to control the speed of animation,
3. to reset the simulation.
4. to move the viewing point forwards and backwards.

The screen where the simulation takes place occupies most of the window. Below the screen is the control area which provides the buttons to control the various actions. The menubar on top of the screen contains two pulldown menu panes. The first pane provides the Undo Command and a Quit Command. The second pane allows the user to choose between running and walking. The following paragraphs describe the implementation.


Run
Actor
Walk
CmdInterface
Application
BasicComponent
UIComponent
DialogManager
InfoDialogManager
MenuBar
MainWindow
MenuWindow
AnimateWindo
Stage
ControlPanel
Clock
AnimateClock

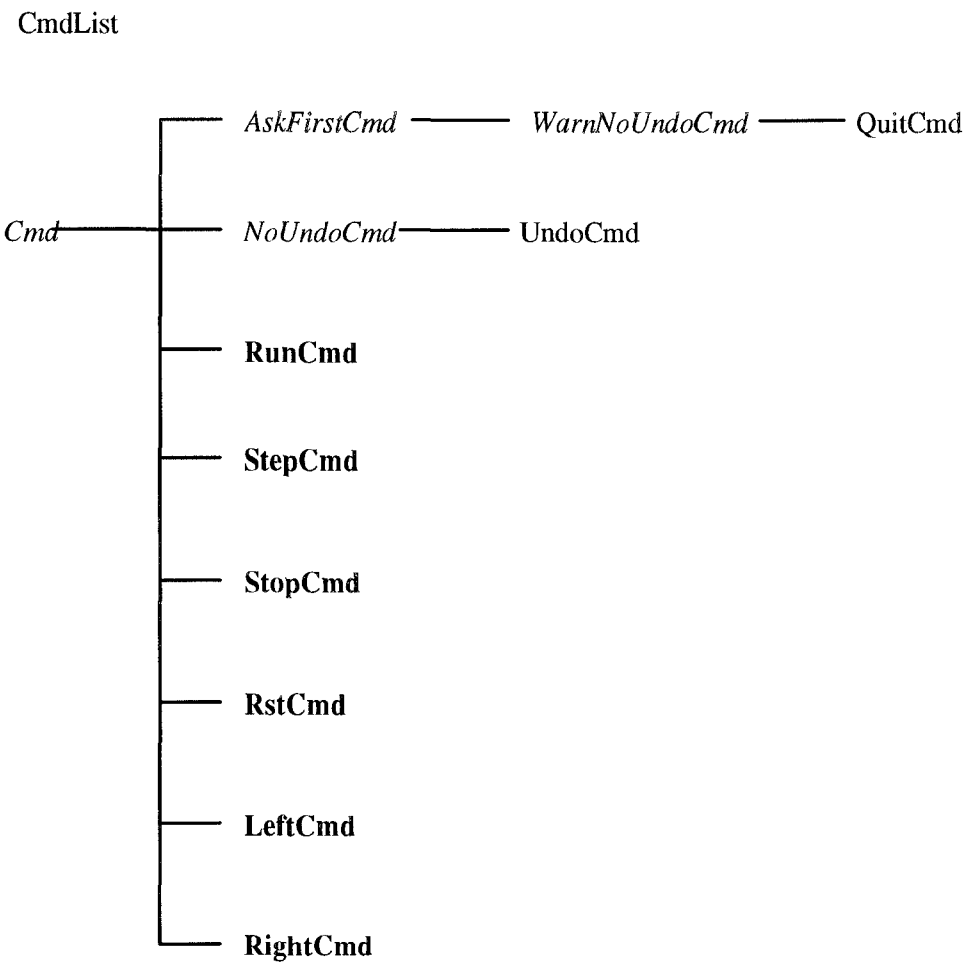


Figure 5.10.1 Structure of the user interface.

Basic Component Class is a simple abstract class and is intended to serve as a base class for other classes. It contains a widget member, _w, which its constructor initializes to NULL. It does not create any widgets. That task is left for derived classes. The widget, _w, serves as the root of the widget tree.

UI Component Class is derived from the Basic Component Class and implements several additional facilities which are useful to many user interface classes. It provides for the correct handling of widget destruction and the use of X resource manager to customize the behavior of the user interface component.

Main Window is an abstract class. It provides support for elements common to most independent top level windows. It supports a shell widget and a Motif XmMain Window widget. Derived classes are responsible only application specific work area which caters to the widgets in the application.

Menu Bar constructs menu panes from lists of command objects.

Application Class is a concrete class derived from UI Component. It handles X/Xt initialization, encapsulates the Xt event loop, creates a main shell that serves as a parent for all other top-level windows, maintains global data structures and opens/closes/iconifies all windows.

Stage Class provides an area in which the biped figure is displayed. To prevent flickering of the animation *double buffering* must be used. Double buffering is a technique whereby the new graphic is drawn in a second buffer and when needed. The two buffers are swapped. Double buffering is implemented here using pixmaps. A pixmap is offscreen memory used for storing images.

Actor, Walk and Run Classes. The actor class is an abstract class which defines a basic protocol for all objects to be animated. It defines a pure virtual function "nextframe" which is implemented by derived classes Walk and Run to produce the animation. Walk and Run classes are responsible for receiving the current state of the system from communication programs.

Clock Class is derived from UIComponent and calls a pure virtual function at a rate determined by the user through a slider.

AnimateClock Class is derived from Clock Class. It implements the pure virtual function defined in the Clock Class to drive the animation.

ControlPanel Class. This class instantiates each of the command classes and creates an interface from them.

Cmd Class. This is an abstract class based on the idea that every command object should support an action and also a way to reverse that action. A command object represents a logical operation as well as the state or data associated with that object. The Cmd Class provides a public interface for each of these operations. This class supports member functions that cab be used to enable or disable commands, along with any interface associated with the command. Each command object also supports a list of other commands that need to be enabled or disabled when this command is executed making it easy to set up dependencies between commands.

CmdList Class. This class is used internally by the Cmd Class to maintain lists of Cmd objects.

No Undo Cmd Class is an abstract class derived from the Cmd Class which supports commands which are difficult to undo.

AskFirst Cmd Class is an abstract class derived from the Cmd Class. It posts a dialog asking the user to confirm a command before executing it.

Warn No Undo Cmd Class is derived from the AskFirst Cmd Class. It warns the user that there is no undo available for a particular command and asks for confirmation.

CmdInterface Class is an abstract class derived from UI Component. It adds a callback function that can be used to execute the associated Cmd object. It accepts activate and deactivate messages from Cmd objects to activate and deactivate the user interface component associated with the command.

StartCmd, StopCmd and RunCmd Classes These affect the simulation by affecting the clock which drives the animation.

LeftCmd and RightCmd Classes. These move the viewing point around by changing the location at which the biped figure is drawn.

RstCmd Class. This resets the animation by sending a reset command to Mathematica.

Chapter 6 Conclusions and Future Directions

6.1 Summary

This investigation into biological control systems has revealed a new perspective for looking at control tasks. Instead of a centralized controller crunching away at numbers we can think in terms of a distributed controller. In Chapter 2 we discussed a theorem which proved very useful in designing such controllers.

Biped walking and running were examined. Actual biped structure was simplified to a model suitable for our purposes in such a way that it retained the essential features of the real system. Dynamical equations were derived, ground forces modelled and necessary simplifications made. Networks of oscillators were used to sequence the torques necessary for proper locomotion.

Any simulation of such complexity makes exacting demands on our ability to generate dynamical equations, to numerically solve them and to display the results in a readily understandable form. These demands were met by taking advantage of special facilities available on different computers and by tying up the whole distributed system through interprocess communication.

6.2 Future Directions

A number of interesting ideas were introduced in the second chapter. These could be further developed. Particularly interesting seem to be ideas having to do with damped oscillators and oscillator hierarchies.

Pulse coupled oscillators merit further investigation. Currently available theoretical results deal only with oscillators which have all to all coupling and are in

synchrony. It would be interesting to see whether these results can be generalized to account for less restrictive forms of coupling such as oscillators being arranged in the form of connected graphs. Also the possibilities of achieving set phase differences between oscillators should be investigated.

It might be interesting to look at the conditions under which such Central Pattern Generators can arise. A network of interacting neurons can be considered and events or conditions leading to the formation of Central Pattern Generators examined. Such work may be tied up with biological investigations of the evolution of Central Pattern Generators.

Mammals move with different gaits at different speeds. The different gaits may be thought to correspond to different phase difference settings between the oscillators controlling such locomotion. If the coupling is a function of the speed of locomotion, it may cause the equilibrium point in the phase difference space to bifurcate at certain speeds of locomotion resulting in gait changes.

The work done here has an interesting intersection with the burgeoning field of computer graphics. One of the goals of computer graphics is achieve visual realism in animations. With the increase in computing power available, the use of real time simulation of dynamical equations in producing realistic animations has become feasible. There is an effort underway to generate control algorithms capable of producing the characteristic gaits of a wide variety of animal [14].

Appendix A Parameters

The parameters for the models used are adapted from [18].

A.1 Walking

The model for ballistic walking used parameters as shown in table A.1.1.

Link	Length (m)	Mass (kg)	Distance of center of gravity from last revolute joint (m)
1	0.87	53.15	0.77
2	0.44	8.41	0.22
3	0.43	4.74	0.28
4	0.12	1.00	0.05

Table A.1.1 Parameters of the ballistic walking model.

A.2 Running

The parameters used for running are shown in the following table.

Link	Length (m)	Mass (kg)	Distance of center of gravity from last revolute joint (m)
1	0.43	8.41	0.22
2	0.44	4.74	0.28
3	0.43	8.41	0.22
4	0.44	4.74	0.28

Table A.2.1 Parameters of the model for running.

Mass of the body $m_o = 40\ kg$.

Appendix B Generation of Dynamical Equations

B.1 Newton Euler's Method

This involves a forward recursion to find velocities and accelerations and a backward recursion to find the torques and forces that each link exerts on another. This has already been illustrated in the derivation of dynamical equations for the walking biped and for the running biped.

B.2 Kane's Method

This method is described in [6] and illustrated for the case of a robot arm in [7]. Here we will try to give a brief summary, based primarily on [6], of the theory behind Kane's method.

Important to Kane's method are the notions of generalized coordinates, generalized speeds, generalized active forces and generalized inertia forces.

The configuration of a set of N_μ particles, S, is known whenever the position vector of each particle with respect to a point fixed in the reference frame, A, is known. If the motion of the set of particles is affected by bodies which come in contact with one or more of the particles, restrictions are imposed on the positions that the affected particles may occupy, and the set S is said to be subject to configuration constraints. Equations expressing such constraints are called *holonomic constraint equations*. If there are M such holonomic constraint equations, only

$$n \triangleq 3N_\mu - M \tag{B.2.1}$$

of the $3N_\mu$ Cartesian coordinates are independent of each other. Under these circumstances one can express each of the components along x, y and z axis of the position vector $\mathbf{p_i}$, x_i, y_i, z_i $(i = 1, \cdots, N_\mu)$ as a single valued function of time t and n functions of t, say, $q_1(t), \cdots, q_n(t)$, in such a way that the constraint equations are satisfied identically for all values of t and $q_1, \cdots, q_n$ in a given domain. The quantities $q_1, \cdots, q_n$ are called *generalized coordinates* for S.

Expressions for angular velocities of rigid bodies and velocities of points of the system S whose configuration in a reference frame A is characterized by n generalized coordinates $q_1, \cdots, q_n$ can be brought into the following form

$$u_r \triangleq \sum_{s=1}^{n} Y_{rs}\dot{q}_s + Z_r, \tag{B.2.2}$$

where Y_{rs} and Z_{rs} are functions of $q_1, \cdots, q_n$ and the time t. $u_1, \cdots, u_n$ are called the generalized speeds of for S in A. The functions $Y_{rs}$and Z_{rs} must be chosen such that equation B.2.2 can be solved uniquely for $\dot{q}_1, \cdots, \dot{q}_n$. It may happen that for physical reasons the generalized speeds $u_1, \cdots, u_n$ are not independent of each other. In that case the system is said to be subject to motion constraints and an equation that relates $u_1, \cdots, u_n$ to each other is called a *nonholonomic constraint equation.* A system which is not subject to motion constraint equations is called a *holonomic system* while a system subject to such motion constraints is called a *nonholonomic* system.

When all nonholonomic constraint equations can be expressed as the m relationships

$$u_r = \sum_{s=1}^{p} A_{rs}u_s + B_r\ , \qquad (r = p+1, \cdots, n), \tag{B.2.3}$$

where,

$$p \triangleq n - m, \tag{B.2.4}$$

and where A_{rs} and B_r are functions of $q_1, \cdots, q_n$, and the time t, S is referred to as a simple nonholonomic system possessing p degrees of freedom in A. For this system $\boldsymbol{\Omega}$, the angular velocity of a rigid body in S, and $\mathbf{v}$, the velocity of a particle P belonging to S, can be expressed *uniquely* as

$$\begin{aligned} \boldsymbol{\Omega} &= \sum_{r=1}^{p} \tilde{\boldsymbol{\Omega}}_r u_r + \tilde{\boldsymbol{\Omega}}_t \\ &\text{and} \\ \mathbf{v} &= \sum_{r=1}^{p} \tilde{\mathbf{v}}_r u_r + \tilde{\mathbf{v}}_t, \end{aligned} \tag{B.2.5}$$

where $\tilde{\boldsymbol{\Omega}}_r$, $\tilde{\mathbf{v}}_r (r = 1, \cdots, p)$, $\tilde{\boldsymbol{\Omega}}_t$, and $\tilde{\mathbf{v}}_t$ are functions of $q_1, \cdots, q_n$ and t the vector $\tilde{\boldsymbol{\Omega}}_r$ is called the r$^{\text{th}}$ nonholonomic partial angular velocity of the rigid body in A, while $\tilde{\mathbf{v}}_r$ is known as the r$^{\text{th}}$ nonholonomic partial angular velocity of P in A.

If $u_1, \cdots, u_n$ are generalized speeds for the simple nonholonomic system S possessing p degrees of freedom in a reference frame A, p quantities $\tilde{\mathbf{F}}_1, \cdots, \tilde{\mathbf{F}}_p$, called *nonholonomic generalized active forces* are defined as

$$\tilde{\mathbf{F}}_r \triangleq \sum_{i=1}^{N_\mu} \tilde{\mathbf{v}}_r^{P_i} . \mathbf{R}_i, \qquad (r = 1, \cdots, p), \tag{B.2.6}$$

where N_μ is the number of particles comprising S, P_i is a typical particle of S, $\tilde{\mathbf{v}}_r^{P_i}$ is a nonholonomic partial velocity of P_i in A, and $\mathbf{R}_i$ is the resultant of all contact forces and distance forces acting on P_i.

Similarly n quantities called *nonholonomic generalized inertia forces* for S in A are defined as

$$\tilde{\mathbf{F}}_r \triangleq \sum_{i=1}^{v} \tilde{\mathbf{v}}_r^{P_i} . \mathbf{R}_i^*, \qquad (r = 1, \cdots, p). \tag{B.2.7}$$

Here

$$\mathbf{R}_i^* \triangleq -m_i \mathbf{a}_i, \tag{B.2.8}$$

where m_i is the mass of P_i and $\mathbf{a}_i$ is the acceleration of P_i in A. There exist reference frames N such that if S is a simple nonholonomic system possessing p degrees of freedom in N, and $\tilde{\mathbf{F}}_r$ and $\tilde{\mathbf{F}}_r^*$ $(r = 1, \cdots, p)$ are, respectively, the nonholonomic generalized active forces and the nonholonomic generalized inertia forces for S in N, then the equations

$$\tilde{\mathbf{F}}_r + \tilde{\mathbf{F}}_r^* = 0, \qquad (r = 1, \cdots, p), \tag{B.2.9}$$

govern all motions of S in any reference frame. The reference frames N are called Newtonian or inertial reference frames and the equations B.2.9 are known as *Kane's Dynamical equations.*

The Mathematica program "kane.m" is based on these ideas and derives dynamical equations governing the motion of a robot arm whose links are connected by a revolute joints. Setting the parameter n at the beginning of the program specifies the number of links in the arm.

B.3 Lagrange's Method

This method uses the concept of Lagrangian which is related to kinetic energy. From Newton's equation of motion we have

$$\mathbf{F}_i = m_i \ddot{\mathbf{p}}_i \, . \tag{B.3.1}$$

Taking the inner product of $\partial \mathbf{p}_i / \partial q_i$ with equation B.3.1 and summing for all particles of the system, we have

$$\sum_i \mathbf{F}_i^T \frac{\partial \mathbf{p}_i}{\partial q_i} = \sum_i m_i \ddot{\mathbf{p}}_i \frac{\partial \mathbf{p}_i}{\partial q_i}, \quad i = 1, 2, \cdots, n. \tag{B.3.2}$$

We have

$$\begin{aligned} \dot{\mathbf{p}}_i &= \sum_{j=1}^{n} \frac{\partial \mathbf{p}_i}{\partial q_j}\dot{q}_j + \frac{\partial \mathbf{p}_i}{\partial t}, \\ \frac{\partial \dot{\mathbf{p}}_i}{\partial \dot{q}_j} &= \frac{\partial \mathbf{p}_i}{\partial q_j}. \end{aligned} \tag{B.3.3}$$

Hence equation B.3.1 can be rewritten as

$$Q_j = \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_j}\right) - \frac{\partial K}{\partial q_j}, \tag{B.3.4}$$

where

$$K = \sum_i \frac{m_i}{2}\dot{\mathbf{p}}_i^T \dot{\mathbf{p}}_i \tag{B.3.5}$$

and

$$Q_j = \sum_i \mathbf{F}_i^T \frac{\partial \mathbf{p}_i}{\partial q_j}. \tag{B.3.6}$$

K is the kinetic energy of the system and Q_j is called the generalized force corresponding to q_j.

The force $\mathbf{F}_i$ can be divided in two parts: the force due to potential fields and the remainder. The potential force can be expressed in terms of the potential field as

$$\mathbf{F}_i = -\frac{\partial P}{\partial \mathbf{p}_i}. \tag{B.3.7}$$

Taking the Lagrangian, L,

$$L = K - P, \tag{B.3.8}$$

we obtain

$$Q_{jb} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i}, \tag{B.3.9}$$

where

$$Q_{jb} = \sum_i \mathbf{F}_{ib}^T \frac{\partial \mathbf{p}_i}{\partial q_j}. \tag{B.3.10}$$

Equation B.3.9 is called Lagrange's equation of motion.

Mathematica programs written using this method were able to derive equations of motion for systems with four degrees of freedom.

Bibliography

[1] J. V. Basmajian. *The Human Bicycle. Biomechanics*, volume 5-A. Baltimore: Univ. Park Press, 1976.

[2] R. H. Byrne. Interactive graphics and dynamical simulation in a distributed processing environment. Master's thesis, University of Maryland, 1990.

[3] James O. Coplien. *Advanced C++, Programming Styles and Idioms*. Addison-Wesley, 1992.

[4] S. Grillner. Locomotion in vertebrates: central mechanisms and reflex interactions. *Physiol. Rev.*, 1975.

[5] A.C. Guyton. *Physiology of the human body*. Saunders College Publishing, 1984.

[6] Thomas R. Kane and David A. Levinson. *Dynamics: Theory and Applications*. McGraw-Hill Book Company, 1983.

[7] Thomas R. Kane and David A. Levinson. The use of kane's dynamical equations in robotics. *The International Journal of Robotics Research*, 1983.

[8] Hassan K. Khalil. *Nonlinear Systems*. Macmillan Publishing Company, 1992.

[9] Thomas A. McMahon. *Muscles, Reflexes and Locomotion*. Princeton University Press, 1984.

[10] Renato E. Mirollo and Steven H. Strogatz. Synchronization of pulse coupled biological oscillators. *SIAM Journal of Applied Math*, 50(6):1645–1662, December 1990.

[11] S. Mochon and T.A. McMahon. Ballistic walking. *Journal of Biomechanics*, 13:49–57, 1980.

[12] Aftab E. Patla. Analytic approaches to the study of outputs from central pattern generators. *Neural Control of Rhythmic Movements in Vertebrates, Editors: Avis H. Cohen and Serge Rossignol and Sten Grillner, Wiley-Interscience*, 1988.

[13] C.S. Peskin. *Mathematical Aspects of Heart Physiology*. Courant Institute of Mathematical Sciences, New York University, 1975.

[14] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4), July 1991.

[15] Wolfram Research. *Mathlink, Technical Reference*. Wolfram Research Inc., 1991.

[16] Serge Rossignol, James P. Lund, and Trevor Drew. Role of sensory inputs. *Neural Control of Rhythmic Movements in Vertebrates, Editors: Avis H. Cohen and Serge Rossignol and Sten Grillner, Wiley-Interscience*, 1988.

[17] R. Tomovic, R. Anastasijevic, J. Vuco, and D. Tepavac. The study of locomotion by finite state models. *Biological Cybernetics*, pages 271–276, 1990.

[18] M. Vukobratovic, B. Borovac, D.Surla, and D.Stokic. *Scientific Fundamentals of Robotics 7: Biped Locomotion - Dynamics, Stability, Control and Application*. Springer Verlag, 1990.

[19] Stephen S. Wolfram. *Mathematica: A System of Doing Mathematics by Computer*. Addison Wesley Publishing Company, 1991.

[20] Douglas A. Young. *The X Window System, Programming and Applications with Xt*. Prentice Hall, 1990.

[21] Douglas A. Young. *Object Oriented Programming with C++ and OSF/Motif.* Prentice Hall, 1992.

[22] H. Yuasa and M. Ito. Coordination of many oscillators and generation of locomotory patterns. *Biological Cybernetics*, 1990.