Analysis and Simulation of Elasticae under Friction and Buckling Forces for the Design of a New Type of Bill-Handling Machine

by

Aris G. Constantinides

B. Eng. Mechanical Engineering
Imperial College of Science, Technology and Medicine, 1991

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 1993

Signature of Author _____
Department of Mechanical Engineering
May 7, 1993

Certified by _____
Harry West
Thesis Supervisor

Accepted by _____
Ain A. Sonin
Chairman, Departmental Graduate Committee

Analysis and Simulation of Elasticae under Friction and Buckling Forces for the Design of a New Type of Bill-Handling Machine

by

Aris G. Constantinides

Submitted to the Department of Mechanical Engineering on May 7, 1993
in partial fulfillment of the requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Mathematical models of the behavior of paper currency in a novel bill handling machine were developed and the reliability of the machine was analyzed in terms of two problems: double-buckling and slip. Double-buckling was analyzed by considering the mechanics of layers of notes under friction forces in their plane. Slip was analyzed by comparing the coefficient of friction to the ratio of the tangential to the normal force at the point of contact with the buckled bill. To calculate the forces, an elastica model of the bill was developed and solved using Runge-Kutta integration combined with the shooting method. The results were in the form of a 2-dimensional map of possible slip regions in the operating space of the bill handling machine. The mathematical models were found to compare well with an experimental prototype. The analysis has yielded insights and design rules to improve the speed and reliability of the machine. In a more general context it has served as a template for modeling and simulating flexible material handlers.

Thesis Supervisor: Harry West
Title: Associate Professor of Mechanical Engineering

Στους γονεις μου, Γιωργο και Εφη,
και στην αδελφουλα μου Δαφνη

To my parents, Ray and Effie,
and to my sister Daphne

Acknowledgements

I would like to thank my advisor, Prof. Harry West, for his help, encouragement and inspiration on this project. I will always remember our weekly brainstorming meetings as intellectually stimulating. I am also grateful to the OMRON Corporation for the funding of this project.

A number of people helped me with this work, generously providing technical or non-technical advice. Among them I would like to thank:

• All the OMRON engineers in what used to be the EMB division: Nishikoji Tomikazu, Kubo Ichiro, Onomoto Ryuichi, Morimoto Masaru and Sugitate Yoshimasa. My working in Japan has been an outstanding experience in my life.

• My friends at the Mechatronics Laboratory: Nathan Delson, Mohamed Khemira, Ross Levinsky, James Pinkney, Anjay Skoskiewicz. The late-night food scene, the 2.70 packing times, the high-school contest days, the fighting over the MACs and other great debates will always remain vivid.

◦ My teachers at Imperial College, but especially Professor Jim Whitelaw, Dr. Alex Taylor, Dr. Ramsey Gohar and Dr. Barry Hill.

• My great friends since the Athens College days, George Alexopoulos, Haralambos Eleftheriadis, Konstantinos Janetos; *retired* "General" Thanos Siapas for being a great room mate; Giannis Daskalantonakis, Chris Konstantarakis for the fiery debates, Emmy Korodima, Dimitri Panagiotou (the "gross hacker") for his friendship and UNIX help, Katerina Panagiotou, Thanos Papadimitriou, George Varsamis and Duke Xanthopoulos.

But most of all, I would like to thank my family: my parents Ray and Effie and my sister Daphne; for standing by me with their love, support and affection through all these years.

Contents

List of Figures

Nomenclature

a_x	acceleration in x direction
a_y	acceleration in y direction
a_θ	angular acceleration
b_i	damping coefficient (aerodyanmics)
B_i	damping coefficient (internal)
C_n	trial contact point
d	vertical displacement
ds	element length
dx	element length in x direction
dy	element length in y direction
E	bending stiffness
fx	force in x direction
fy	force in y direction
F	force
F_b	buckling force
F_f	frictional force
F_n	normal force
F_t	tangential force
F	discrepancy vector
I	second moment of area
L	length of bill
L^*	length at which angle is zero
L_1	length of left elastica
L_2	length of right elastica
L_c	initial contact length
L_f	length fed by roller
m	bending moment
m_i	mass per unit element
M[]	added moment for curved bills
M_l	moment from left elastica
M_r	moment from right elastica
r	curvature
R	ratio of tangential to normal forces
s	arc length
s_c	actual contact length
T_n	smallest period of the elastica
v	trial vector for shooting
v_x	velocity in x direction
v_y	velocity in x direction
v_θ	angular velocity

w	weight per unit length
x	x position
y	y position

Greek Symbols

$[\alpha]$	Jacobian matrix
$\Delta\theta$	angular step for roller rotation
Δt	time step for dynamics
Δt_{cr}	critical time step
δv	increment to trial vector
δv^*	clipped increment
θ	angle with horizontal
θ_1	angle for curved bills
θ_{seg}	angle of segmented roller's arc
μ	coefficient of friction
Ω	angular velocity of roller

Subscripts

1	top bill
2	second bill
3	third bill
c	contact
new	new bills
old	old bills
r	center of roller
t	current time-step
t-Δt	previous time-step
t+Δt	next time-step

Superscripts

k, old	previous iteration
k+1, new	new iteration

1. Introduction

1.1 Background

Automated flexible-material handling machines have become increasingly important in many applications: from bill-counters and automated teller machines (ATMs) used for paper currency processing, to paper-feeders in copier machines, to fabric-handling devices used in automated garment assembly lines. Technologically, these devices are complex, due to the particular way in which the material is processed: high flexibility and large deformations. These two features make analysis and design of the automated flexible-material handling machines difficult.

An example of such a device is the automatic bill in-out (ABIO) machine, designed by OMRON Corporation of Japan, and is used in their line of ATM machines. Its purpose is to count bills from a stack and feed them to the next stage of processing. OMRON has been producing machines in this market since 1969; they currently have 20% of the ATM market and more than 20 years experience in the design of ATMs. Until recently the design of their bill handlers has been based mostly on their experience and experiment (prototype testing). It has become clear, however, that to increase the efficiency of their machines and to reduce the design time, they needed to enhance their knowledge on the analytical aspect of bill-handling and develop simulation techniques for testing their new machines.

As such devices became more widely used and complex, research on flexible materials has developed. The motivation has been the need to model the flexible medium analytically in order to, first, understand its behavior under the effect of various forces

and deformations and, second, develop computational simulations to avoid prototype building.

The traditional approach for designing devices for handling flexible materials has been to use intuition, experience and a large number of prototype iterations; an inefficient and slow process. The lack of general purpose computational packages, such as finite element programs, capable of analyzing bill handling machines has precluded any other approach to design. Our research has been aimed at establishing methods for analytical modeling and simulation through the design of bill-handling equipment. This goal has been met by the development of a computational template for designers.

1.2 The Bill Feeder

The bill feeding devices used by OMRON for its ATM machines have been based on two feeding methods: pinch-friction and gap-friction methods with several modifications. The failure rate of these devices is low (highest rate has been 3 errors per 1500 transactions, 20 notes per transaction, or 0.01 % error) and the highest speeds attainable during operation are 10 notes per second. The most frequent problems encountered in these types of feeders are skew and jamming, the latter depending on the condition of bills. Previous work on these feeders has been done by MIT, mainly in developing a self-correcting deskewing mechanism to correct the skew problem [Levinsky, 1992, Kotovsky, 1990]. The deskewing mechanism is connected in series with the bill feeder and corrects the skew of the bills leaving the feeder.

OMRON is interested in developing completely new concepts for bill handling. The motivation is the need to radically improve the ATM design in terms of reliability, speed and size in order to have a lead in the next generation of ATM machines. To achieve this, a team of MIT and OMRON engineers brainstormed new ideas for the development of a new device, with the potential for improvement over the existing technology. The objective of the new design was to increase the speed from 10 notes/sec to as high as 20 notes/sec, while improving the reliability of the processing.

The result of the brainstorming session was a radically new method in bill feeding. A prototype of the new bill feeder was developed and available for testing. This

device can be described as a 'buckling feeder', since its principal action is to buckle the bills from a stack and to separate them for processing. The simple design of the feeder consists of a main roller, to which three smaller rollers are attached. The smaller rollers rotate independently of the main roller, at a constant angular velocity. The bills are placed in a stack, under the roller and are firmly clamped at one end by a stabilizer. Combined rotation and vertical pressure on the stack results in buckling of the top bill. Consequently, the top bill is lifted from the stack by the roller, by virtue of contact forces that arise from frictional contact between the rubber-coated roller and the bill. Hence, the device's main characteristics are frictional contact and buckling. Figure 1.1 shows a schematic diagram of the machine's end-effector (roller).

The bill feeding process is separated into three stages to facilitate understanding and modeling of its operation. The initial stage, termed Stage 1, is defined when both the roller and the bill are in contact with the stack. This is the stage of pure buckling. Stage 2 is defined when the roller is in contact with the bill at some point (and not in contact with the stack), while the bill still is in contact with the stack at some other point. Stage 3 is considered the lifting stage, since the roller is in contact with the bill but neither is in contact with the stack. The lifted bill is then grasped by a mechanical arm which keeps it from falling back on the roller or the stack. The counting function is done during the lifting stage, as the lifted bill passes through an infra-red beam, triggering a counter.

The reliability of this feeder is limited by a number of problems that arise during its operation, particularly when bills of different conditions are processed. The main problems are: double-buckling and slipping. Double-buckling occurs in Stage 1 when the roller buckles more than just the top bill of the stack. This results in two bills being lifted (in Stages 2 and 3) and, hence, double-counting. The problem of slip arises during the lifting stage, when the bill slides from the roller and falls back to the stack.

Initial work on this device can be found in [Dotterer, 1991], which initially explored the problems of double-buckling and slip. Dotterer assumed that the shape of the buckled bill represented a sine curve and solved the model equations analytically. As a result of this preliminary work it was decided that a more detailed numerical analysis was needed.

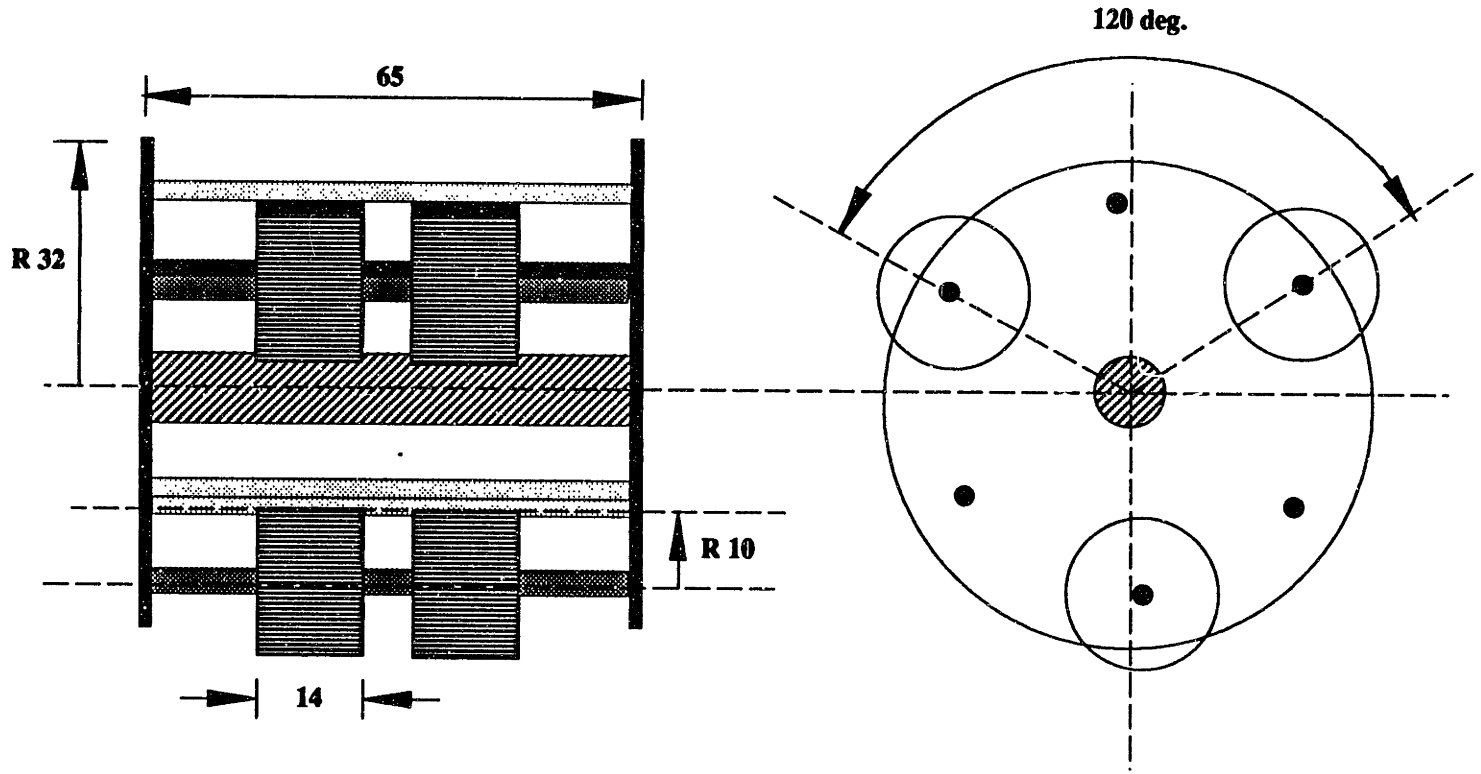


Figure 1.1: Schematic diagram of the end-effector of the feeder (the roller)

1.3 Objective of this Work

Computational analysis of the design of mechatronic devices is becoming increasingly important. The present work on the buckling feeder is two-fold; first, in a design context, to determine the causes of the various problems mentioned above and the design modifications necessary to eliminate them. Second, to develop a computational simulation of the device's operation and, in extension, a generally-applicable computational methodology and framework for approaching similar problems in the future.

Three tasks were undertaken. The first involved developing a finite element model of the feeder on a commercially available package, testing the operating performance of the model. The second was to analyze the prototype from a general design perspective, identifying possible modifications and improvements. The third was to develop a numerically solvable analytical model of the feeder and computationally simulate its operation. Based on the computational results, a modified design of the buckling feeder was produced. The computational model was altered to accommodate the particularities of the new design. In this work, the three tasks and the results obtained from each will be presented in the following order.

Chapter 2 discusses the initial-buckling stage, with the objective of eliminating the problem of double-buckling. The approach towards this problem is in terms of analyzing the process of the initial buckling stage: the material properties of Yen paper-currency is considered and the mechanics of pure buckling are described. Conclusions are drawn regarding the success of this stage and design modifications are proposed.

Chapters 3 and 4 present the numerical models of the bill-feeding process, which serve as the underlying framework for the computational simulation discussed above. Chapter 3 emphasizes on the theoretical standpoint of these models, while Chapter 4 stresses on their application to the buckling feeder. The theoretical background includes discussion of the numerical methods employed for the analysis, reasons for choosing the particular methods as well as the presentation of the boundary conditions applicable to the different stages of the operation. The discussion of the application of these models to the buckling feeder focuses on each of the operating stages separately, justifying the

boundary conditions applicable to each. Chapter 4 also elaborates on the problems involved with modeling as well as the simulation of curved and folded bills.

Chapter 5 presents and discusses the results obtained from the solution of the models of Chapters 3 and 4. The results are initially presented for Stages 2 and 3 separately, for different bill conditions, then for combined Stages 2 and 3 and are finally compared to the experimental results obtained from the prototype.

Appendix I presents the material characteristics of Yen. The last two appendices contain information regarding the finite element modeling and computational modeling of the operation respectively. Appendix II briefly describes the finite element approach, the model and the results obtained. Appendix III includes a listing of the codes developed that simulate the operation of the prototype device.

2. Initial Buckling

This chapter presents and discusses the analysis of the initial buckling stage, defined earlier as Stage 1 buckling, during which, both the roller and the bill are in contact with the floor. In this context the term 'floor' designates the top of the stack, on which rests the roller and the processed bill. The primary problem of this stage, is the problem of double-buckling (or double-feeding) defined when two (or more) bills are simultaneously buckled and are not separated during the lifting operation (Stages 2 and 3). The analysis of this stage has been developed by taking into account the varying material characteristics of the bills and the manner in which they affect the fundamental mechanics of the buckling operation. Initially, the various types of bills are defined and categorized depending on their conditions. Secondly, the buckling mechanics are formulated and compared for different bills. Finally, alternative designs eliminating the double-buckling problem are briefly discussed.

2.1 Material Properties of Yen

Reliable operation of the buckling feeder has to be guaranteed for bills of different conditions, such as those found in circulation. There are several parameters which characterize the condition of a bill and, hence, the performance of the buckling operation. These parameters are:

a. Bending stiffness (E), for buckling in the longitudinal direction of the bill.
b. Coefficient of friction, determined by the surface characteristics of the bill (roughness).
c. Permanent deformations in the bill, as in the case of folded, creased or crumpled bills.

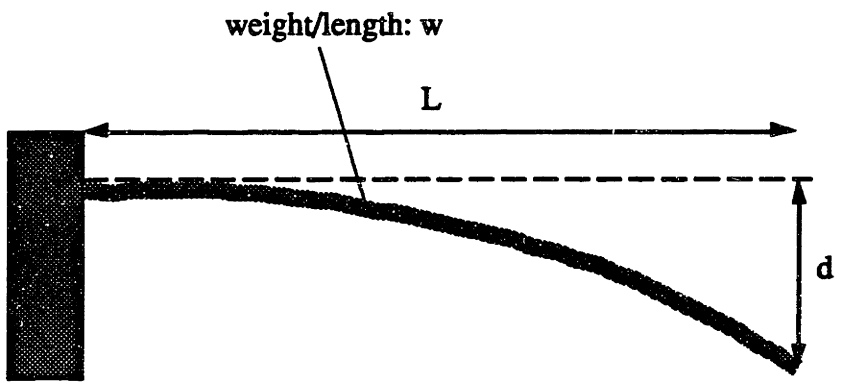


Figure 2.1 : The stiffness test, used for the calculation of the bending stiffness of bills.

For analysis of initial buckling, only the first two parameters were taken into account quantitatively; the third was included in the calculation of the stiffness of the bill in stage 2 & 3 analysis (see Section 4.6). The bills were grouped, for modeling purposes, under two main categories: new (crisp) bills and old (used) bills. There is, of course, a continuum of bill conditions, but only the two extreme cases of new and old need to be considered. If reliable operation is possible for these two extremes, it will also be guaranteed for groups in between.

The experimental tests to determine the properties of new bills were carried-out on 'dummy' Yen notes, provided by OMRON, while for old bills, worn dollars were used. The reason for using dummy Yen notes is that they are used by OMRON for their own tests and that they share the same properties as real Yen. Worn-out dollars were used to simulate the old bills since, on average, the condition of used Yen bills is better than the condition of 'old bill' extreme.

A stiffness test was used to measure the bending stiffness (E) for a particular bill and a friction test was carried out to measure its coefficient of friction (μ). The first test involves clamping the bill at one end, allowing the other end to droop (cantilevered), and measuring the resulting deformation characteristics (horizontal spread and vertical drop). The bending stiffness of the bill is then given by:

$$E = \frac{w L^4}{8 d I} \qquad (2\text{-}1)$$

where w is the weight per unit length (taken as 0.00785 kg/m), I is the second moment of area (for the particular geometry calculated as 6.33×10^{-12} m^4), d the vertical displacement of the free end and L the horizontal spread of the bill.

Figure 2.1 shows the stiffness test for measurement of the bending stiffness of a bill. This measurement was done for both old and new bills and the values of L and d were found by experiment. It was found that the stiffness for new and old bills was:

$$\text{old: } E_{old} = 4.13\ 10^6 \text{ N/m}$$
$$\text{and}$$
$$\text{new: } E_{new} = 4.1\ 10^9 \text{ N/m}$$

The ratio of the stiffness of new bills to old bills is about 3 orders of magnitude.

The second test for measuring the coefficient of friction of the bill is illustrated in Figure 2.2. The bill was placed on an inclined surface, with frictional coefficient corresponding to the condition of the second bill, and a mass of known weight was placed on the bill, to simulate realistic contact conditions. The coefficient of friction describes the frictional characteristics of one surface against another. It must be noted that the coefficient of friction for a pair of bills of different conditions is independent of the order of the two bills. Hence,

$$\mu(\text{new,old}) = \mu(\text{old,new})$$

Three trials were held for different bill conditions: one for new against new, a second for new against old and a third for old against old. For each of these combinations the coefficient of friction of the pair was calculated and found to be:

$$\mu(\text{new,new}) = 0.231 \quad \mu(\text{old,old}) = 0.662 \quad \mu(\text{new,old}) = 0.43$$

The coefficient of friction also varied depending on the two surfaces. The lowest value was found for new bills against new bills, whereas the highest was found for old against

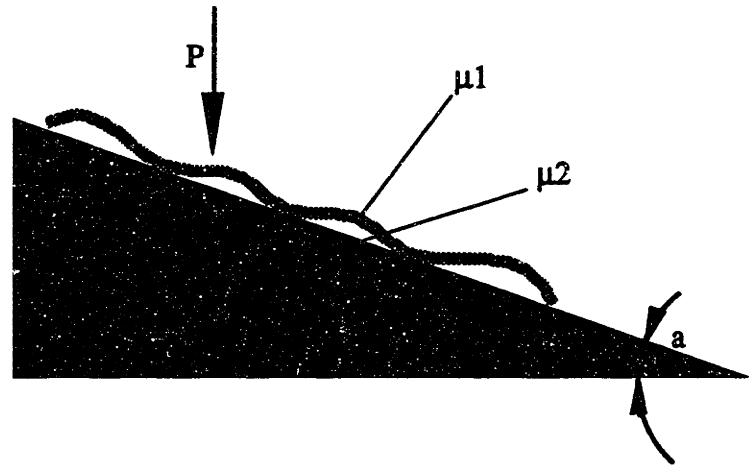


Figure 2.2 : The friction test, used for the calculation of the coefficient of friction between bills of different quality.

old. This can be explained from the fact the surfaces of old bills are rougher than new bills'. However, brand new bills (bills that have not yet been circulated) were not considered in this test. The coefficient of friction between these types can reach as high as unity, due to the stickiness of the ink, which is still fresh on the bill's surface.

2.2 Mechanical Analysis of the Buckling Stage

The main cause of error for Stage 1 is double-buckling which occurs at the beginning of the stage, when both the roller and the bill are still in contact with the floor. It is not necessary to model this process by developing a numerical model; the bills can be represented as flexible beams which are buckled in the longitudinal direction in a simple 2D model [Timoshenko, 1961].

Consider the top three bills in the stack (Figure 2.3). The top bill is buckled by a horizontal frictional force applied by the roller, due to the vertical pressure from the roller on to the stack of bills. This force is given by:

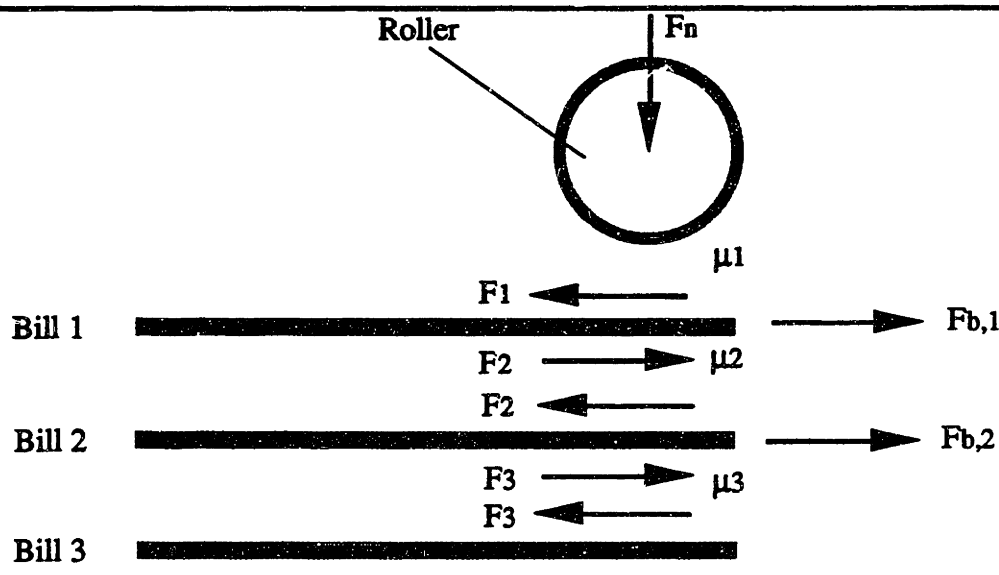


Figure 2.3 : Schematic representation of the forces acting on the top three bills. Note that the reactions are also included in the figure.

$$F_1 = F_n \mu_i \tag{2-2}$$

where F_1 is the horizontal frictional force, F_n the normal force from the roller and μ_0 the coefficient of friction between the roller and the top bill. The horizontal forces between the top and second bills and between the second and third bills are termed F_2 and F_3 respectively. The corresponding friction coefficients are termed μ_1 and μ_2. Similarly, these forces are given by:

$$F_2 = F_n \mu_2 \tag{2-3}$$

and

$$F_3 = F_n \mu_3 \tag{2-4}$$

where F_n is the same normal force as in (2-2). This is an approximation neglecting the lateral stiffness of the flat bills.

The horizontal buckling forces for bills 1 and 2 are analytically given by:

$$F_{b,1} = \frac{4\pi^2 E_1 I}{L^2} \tag{2-5}$$

and

$$F_{b,2} = \frac{4\pi^2 E_2 I}{L^2} \tag{2-6}$$

where E is the stiffness of each bill, I the second moment of area and L is the length of the bill. Hence, in order to buckle the top bill, the following relation must be satisfied:

$$F_1 - F_2 > F_{b,1} \tag{2-7}$$

Similarly, to avoid buckling the second bill, the following must hold:

$$F_2 - F_3 < F_{b,2} \tag{2-8}$$

Therefore,

$$\frac{F_{b,1}}{(\mu_1 - \mu_2)} < F_n < \frac{F_{b,2}}{(\mu_2 - \mu_3)} \tag{2-9}$$

Hence, the normal force F_n applied from the roller to the bills must be between these two values if double-buckling is to be avoided.

Table I - Values of the normal force for combinations of new/old conditions for the top three bills in stack: 'N' = New, 'O' = Old.

Bill 1	N	N	N	N
Bill 2	N	N	O	O
Bill 3	N	O	O	N
Min. Fn (N)	0.153	0.153	0.218	0.218
Max. Fn (N)	--	--	--	--

Bill 1	O	O	O	O
Bill 2	O	O	N	N
Bill 3	O	N	N	O
Min. Fn (N)	0.000157	0.000157	0.00008	0.00008
Max. Fn (N)	--	0.000162	0.514867	--

To include the effect of bills of different characteristics in the analysis, all possible combinations of bill conditions were considered for the three top bills. There are 8 possible combinations for new and old bills. These are shown in Table I. Below each combination, the maximum and minimum value of the normal force is given, as calculated from the above equations. The cells containing no entry indicate that there is no maximum value (ceiling) for the normal force according to the analysis. This arises when $\mu_1 = \mu_2$ or when $\mu_2 > \mu_1$. In these cases the value of F_n must only be greater than a minimum value to achieve buckling, and consequently, double-buckling is impossible.

The following conclusions can be drawn from the analysis. First, when a stack consists of bills of the same quality (all old, or all new), the double-buckling problem can be completely avoided if the normal force is close to, but higher than the minimum value shown in Table I. In some combinations, when mixed bills are present in a stack, two extreme values of the normal force are given. These dictate the range of normal forces to avoid double-buckling. Since this range of forces is small for cases such as the OON (old-old-new) combination, it can be inferred that the problem of double-buckling will be significant under such bill combinations and that the normal force applied has to be

precisely controlled. Second, considering all combinations, it is clear that there exists no common value of the normal force which can be used to avoid double-buckling. This leads to the conclusion that when bills of different conditions are present, it is likely that two bills will be buckled simultaneously.

In reality, however, the problem of double-buckling is not as unavoidable as may be inferred from the previous observations, for two reasons. Experiments done on the prototype indicated that, even if two bills are buckled initially, such as in the OON case above, once the roller moves into the lifting stage, the second bill may drop back onto the stack and only the first will remain fixed to the roller. Secondly, there are alternative methods that can be used for initial buckling (discussed in Section 2.3) which can guarantee that only one bill will be picked by the roller at a time.

This analysis has simplified the modeling of double-buckling in the following ways: first, the two extreme cases were examined quantitatively, those of new bills and those of very old bills. Second, the quantitative analysis was done by taking the two most important parameters (stiffness and coefficient of friction) into account. The permanent deformations of the bill were only qualitatively included into the above parameters. Third, the effects of other parameters, such as the flexibility of the stack or the width-wise deformation of the bills were not considered.

2.3 Design Alternatives

To overcome double-buckling in the case of multiple bill types, several modifications to the original design were proposed. Each of these alternative ideas has originated from the analysis presented in the previous section. The same principle of roller/bill interaction has been kept in all the alternative solutions.

1. Suction roller

Using a roller with airflow, would significantly decrease the possibility of slip, while at the same time separate only the top bill from the stack, due to the upwards force caused by the airflow. Therefore, the problem of double-buckling would be negligible. This method is advantageous in that it would reduce slip and double-buckling problems

greatly while allowing very high speeds of operation. The major drawback, however, is that it would require an air-pump which would increase cost, increase noise levels and increase the size of the machine.

2. Buckling in different direction

In order to avoid the creases in the bills, which are usually directed in the width direction, the roller can buckle the bill at an angle of 45º or 90º to the longitudinal direction. This method would solve the problem of very old bills which have small stiffness in the length direction. A serious disadvantage of this method is that old bills that have been buckled at an angle can be easily torn, especially at high speeds.

3. Eliminate creases by pressure

Another method to eliminate the effect of the creases on successful buckling is to apply some pressure on the center of the bills (at the position of the creases) which would imply shortening the buckling length by a factor of two. Once the half-bill has been buckled, the pressure would be removed and the normal operation would resume. A disadvantage of this alternative is that it would require an additional component to the end-effector (apart from the basic roller) which would reduce the speed of operation.

4. Two-stage buckling

As an intermediate stage between buckling and lifting, a weight test could be applied to see if one or more bills have been lifted by buckling. Including however this intermediate stage would imply loss of operating speed and impose an additional problem of handling the second bill, assuming it has been lifted accidentally by double-buckling.

5. Non-constant normal force

By adjusting the trajectory of the roller in such a way so as to provide a non-constant normal force on the bills (eg. starting from a 'just-touch' position to a position of 'jamming' relative to the stack), the whole range of necessary normal forces (as described in Section 2.2) could be covered. However, this method would be dependent, not only on the size of a particular stack, but also on its compliance. It would be difficult to provide a controllable normal force.

3. Numerical Solution of the Elastica

The following chapters concentrate on the lifting operation (Stages 2 and 3) and the problems of roll and slip. It was necessary to analyze these stages by developing a numerical solution to the mathematical models describing them. This chapter develops these models, while Chapter 4 emphasizes their application to the feeder's operation. Both chapters consider a quasi-static analysis, while Chapter 6 introduces a dynamic model.

3.1 Mathematical Model of the Elastica (Governing Equations)

An elastica is a mathematical model of a thin sheet or a slender elastic rod. The basic theory of the elastica has been established by Euler, who defined the elastica as being a beam whose deformation is only due to bending, such that the local curvature is proportional to the local moment experienced. Heavy elasticae, defined as having non-negligible density, have many engineering applications, ranging from textiles and paper to metal sheets and plastics. A review of many applications is given in [Wang 1986].

The elastica is discretized into elements, each being subject to the governing differential equations that characterize the elastica. For each node, seven variables are defined:

x: x position
y: y position
θ: angle
$\frac{\partial\theta}{\partial s}$: curvature
fx: force in the x-coordinate direction

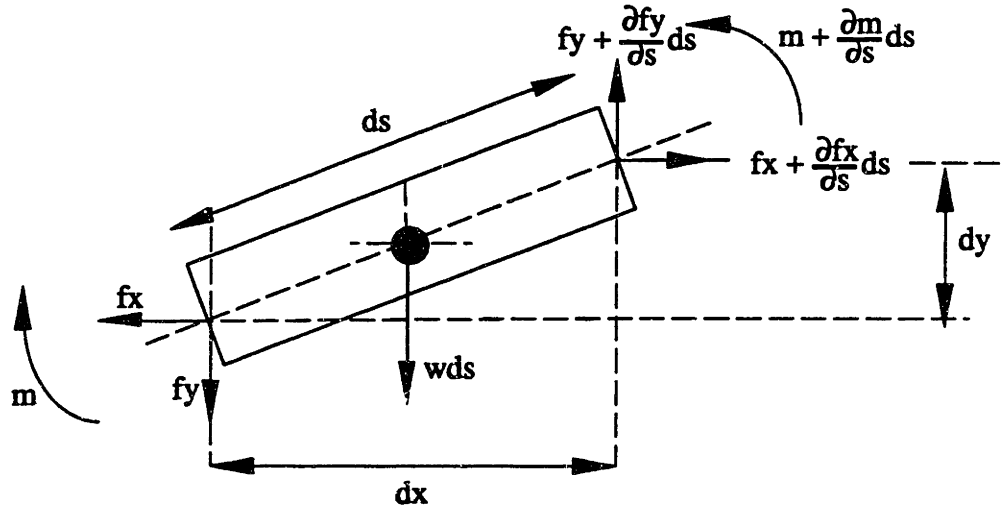


Figure 3.1: Static Equilibrium of an element of the elastica

fy: force in the y-coordinate direction

m: bending moment

The purpose of the analysis is to calculate the values of each of these variables for all the nodes of the elastica. Then, the shape of the elastica can be completely defined.

The conditions of static equilibrium for a single element of the elastica are shown below and refer to Figure 3.1.

Equation (3-1) represents the static equilibrium of the elastica with respect to the x coordinate.

$$-fx + fx + \frac{\partial fx}{\partial s} ds = 0 \Rightarrow \boxed{\frac{\partial fx}{\partial s} = 0} \qquad (3\text{-}1)$$

The next equation is obtained by considering the y coordinate static equilibrium.

$$-fy + fy + \frac{\partial fy}{\partial s}ds - wds = 0 \Rightarrow \boxed{\frac{\partial fy}{\partial s} = w} \tag{3-2}$$

Equation (3-3) is an expression of the rotational equilibrium of the element.

$$-m + m + \frac{\partial m}{\partial s}ds - wds\frac{dx}{2} + fydx - fxdy = 0 \Rightarrow$$

$$\frac{\partial m}{\partial s}ds = fx\frac{dy}{ds} - fy\frac{dx}{ds} \Rightarrow$$

$$\boxed{\frac{\partial m}{\partial s}ds = fx\sin\theta - fy\cos\theta} \tag{3-3}$$

Finally, theory of elastic beams under bending stresses relates the bending moment, m, to the curvature of the beam, r, through

$$\frac{1}{r} = \frac{m}{EI} \tag{3-4}$$

where EI is the beam's flexural rigidity. Mathematically, the inverse curvature can be written as

$$\frac{1}{r} = \frac{\frac{\partial^2 y}{\partial x^2}}{\left(1 + \left(\frac{\partial y}{\partial x}\right)^2\right)^{\frac{3}{2}}} \tag{3-5}$$

where the numerator and denominator contain the second and first derivative of y with respect to x respectively. Equating and re-writing (3-4) and (3-5):

$$\boxed{\frac{\partial \theta}{\partial s} = \frac{m}{EI}} \tag{3-6}$$

which is the large deflection form of the Bernoulli-Euler beam equation.

3.2 Numerical Solution of the Elastica Model

The solution to the governing equations of the elastica is not completely defined by these equations alone. What remains to be established is the nature of its boundary conditions, which are algebraic conditions imposed on these equations at specific values in the solution domain. The computational domain for this particular problem is discretized into thirty-four elements, as shown in Figure 3.2. This number was chosen as a compromise between speed of solution and simulation accuracy. The equations derived previously are solved numerically. The type of boundary conditions specifies the type of numerical problem and determines which numerical methods should be used.

There are two main types of numerical problems, characterized by the nature of boundary conditions: boundary value problems (BVP) and initial value problems (IVP). In the first type of problems, boundary conditions are specified at two (or more) different points in the solution domain, with different conditions specified at each point. Therefore, these problems are often termed two-point boundary value problems. In the second class of problems, all the conditions are given at the start of the solution. Techniques for solving BVPs include relaxation methods (finite-difference techniques) or finite element methods. For IVPs, popular methods include several integrators, such as the Runge-Kutta integrator or predictor-corrector methods.

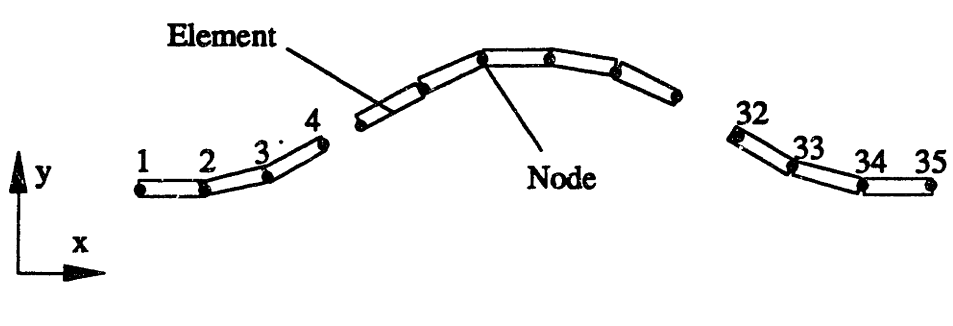


Figure 3.2: Discretization of the elastica into 34 elements

This particular problem can be classified as a two-point boundary value problem. However, instead of using one of the BVP methods mentioned above, an IVP method, known as the shooting method using fourth-order Runge-Kutta integration was used. This was chosen for two reasons: first, it is simpler to use and provides more flexibility to alterations of the boundary conditions of the problem and, second it is considerably more accurate with a small number of iterations. Since this is a BVP, not all the boundary conditions are specified at the start of the solution domain, which is one of the requirements of an IVP method. Therefore, the Runge-Kutta integrator was combined with the shooting method, which is discussed in more detail in the next section.

3.3 The Shooting Method

The shooting method is based on a simple principle: if the boundary conditions at one end of the elastica are unknown, guess them! So, the shooting method, as illustrated in Figure 3.3, consists of a basic strategy: guess - integrate - compare. Once the initial guesses are made, an initial value method is used to march throughout the solution domain, with some conditions at the start known and some guessed. By integrating, the conditions at the other end are determined. The calculated conditions at the end of the solution domain are then compared to the desired solutions at that end, and the difference is used to provide new, better, guesses at the start. This iterative loop is repeated until the difference between actual and desired conditions reaches the required level of accuracy. At this point, the elastica has been completely solved and all its characteristic variables have been determined. The theory of the shooting method is described in detail by [Press, 1988] and is discussed briefly below.

At the start of the solution x_1, the unknown boundary conditions are placed in a vector **v**, whose components are guessed before each integration, while the vector **y** contains the values of all the elastica variables at each point in the domain. The dimension of **v** is, therefore, the number of unknown boundary conditions at x_1, while the dimension of **y** is the total number of boundary conditions at x_1. By integrating the governing equations from x_1 to x_2, the end of the solution domain, using the boundary conditions in $\mathbf{y}(x_1)$, the values of $\mathbf{y}(x_2)$ are determined. The difference between these values and the desired boundary conditions at x_2 are placed in a discrepancy vector **F**, of dimension equal to the number of unknown conditions at x_1.

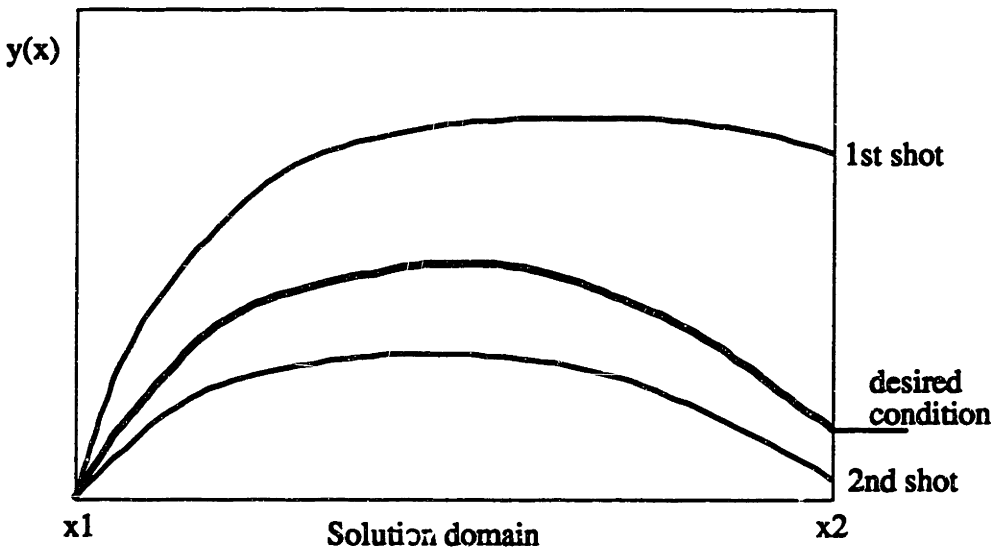


Figure 3.3: Schematic representation of the shooting method. The trial shots are compared to the desired condition and an improved guess is tried.

The shooting method is a multi-dimensional Newton-Raphson method that seeks to find a vector v that will reduce **F** to zero. This is done by iteratively solving the set of equations

$$[\alpha]\,\delta v = -F \tag{3-7}$$

and

$$v^{new} = v^{old} + \delta v \tag{3-8}$$

where v^{new} is the new guess vector, based on the old vector (of the previous iteration) v^{old} , and the difference δv. This difference is calculated from (3-7) by inverting the matrix $[\alpha]$ and multiplying it to the discrepancy vector **F**. The elements of $[\alpha]$ are given by:

$$\alpha_{ij} = \frac{\partial F_i}{\partial V_j} \tag{3-9}$$

which is approximated by

$$\frac{\partial F_i}{\partial V_i} \approx \frac{F_i(V_1, ..., V_j + \Delta V_j, ...) - F_i(V_1, ..., V_j, ...)}{a_j} \tag{3-10}$$

The matrix [α] is essentially the stiffness matrix of the elastica, containing elements proportional to the difference of the discrepancy vectors for two consecutive iterations. The solution, therefore, converges when the vector **F** becomes smaller after each iteration. Crucial to the convergence of the solution is the choice of a_j's, whose maximum value is limited by the accuracy and minimum by the stability of the solution. It was empirically determined that the best values of the a_j's should be:

$$j = 1, 2, ..., n \qquad a_j = 0.5\ 10^{-9}$$

In order to make the solution more stable, the increments δv of (3-8) are adjusted so that only half of the calculated increment is added back to the trial vector. By this way, the increments are much smoother. Hence,

$$v^{new} = v^{old} + \delta v^* \tag{3-11}$$

where,

$$\delta v^* = 0.5\ \delta v \tag{3-12}$$

A drawback of the shooting method, however, is the dependency on the initial (guessed) conditions. For fast convergence, it is required that the initial guesses are very close to the actual conditions. Otherwise, the solution may converge to the wrong value (Section 4.7).

3.4 Application to the Buckling Feeder (Boundary Conditions)

The purpose of this section is to elaborate on the formulation of the numerical methods (Runge-Kutta and shooting method) as applied to the buckling feeder. Initially, the particular features of the problem are presented and, then, a strategy is developed.

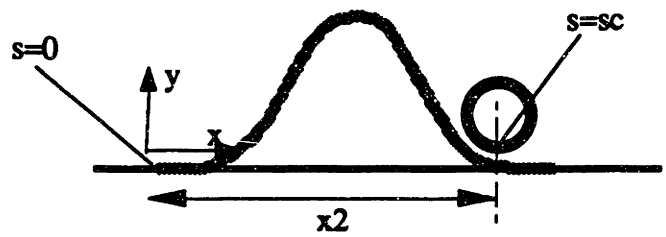
The three functional steps of the buckling feeder have been discussed in the first chapter; they are: Stage 1, initial buckling of the bill; Stage 2, start of lifting stage and Stage 3, lifting of the bill by the roller. These three steps completely define an operating cycle of the device.

The separation into functional stages is both necessary and meaningful for the modeling of the buckling process. It seems reasonable, therefore, to separate the stages of the simulation in terms of boundary conditions.

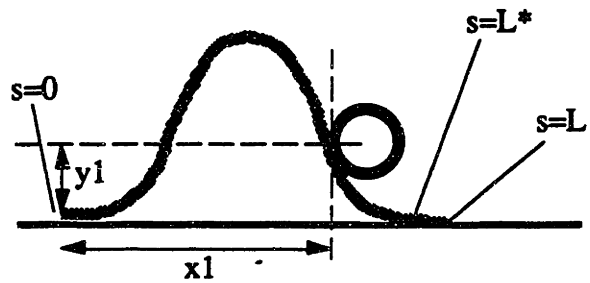
Recall that the left side of the bill is "clamped" by a support which remains firm throughout the operation. This implies that the boundary conditions at the left end (Start of the solution) will remain constant at all times. It therefore remains to determine the boundary conditions that apply at the other end of the elastica or at some point in its length.

Three stages are enough to characterize the whole bill feeding process in terms of the different boundary conditions that apply to the elastica. The first stage, termed Stage 1, applies when the roller is in contact with the elastica which, in turn, is in contact with the floor *at the point of contact with the roller*. This situation is illustrated in Figure 3.4-a. Stage 2, the intermediate stage, is more difficult to solve and is defined when the roller is in contact with the elastica at some point, this point not touching the floor, but, unlike Stage 3, some other point of the elastica *is* in contact with the floor. This situation is depicted in Figure 3.4-b. The final stage, termed Stage 3, becomes effective when the roller is in contact with the elastica, at some point along its length, but the elastica is no longer in contact with the floor.

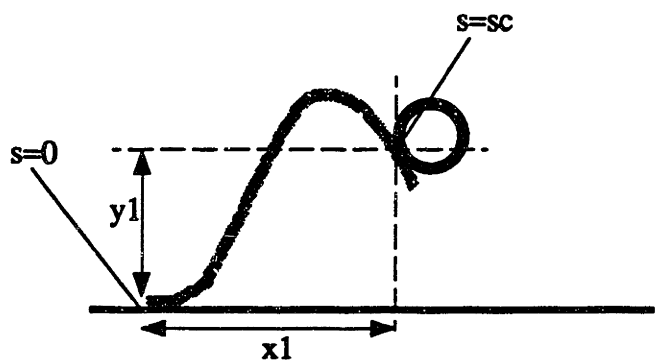
All three stages have different boundary conditions, which necessitates their separate solution. Initially, the boundary conditions that apply to the start of the elastica are discussed. Since this end is clamped, the angle of the elastica (θ) at this point will be



(a)



(b)



(c)

Figure 3.4: (a) Stage 1 - pure buckling of the elastica
(b) Stage 2 - the intermediate stage
(c) Stage 3 - buckling and lifting

equal to zero. Furthermore, since the origin of the coordinate system has been chosen to coincide with this point, the values of x and y positions are also zero. Hence:

$$x(0) = 0$$
$$y(0) = 0$$
$$\theta(0) = 0$$

The boundary conditions that define Stage 1 (at the end of the solution domain, ie. the contact point between the roller and the elastica) are given by:

$$x(s_c) = x_2$$
$$y(s_c) = 0$$
$$\theta(0) = 0$$

where, s_c is the contact point on the elastica, or the end of the solution domain, and x_2 is a known distance from the origin, in the x-direction, to which the roller has moved. This is illustrated in Figure 3.4-a.

Stage 2 is essentially a three-point boundary value problem, since there are boundary conditions specified at three distinct points along the length of the elastica. The first, at the start, has already been examined. The second boundary point is at the point of contact of the elastica with the roller and, the third point is the contact between the elastica and the floor. The boundary conditions at the third point are:

$$y(L^*) = 0$$
$$fx(L^*) = \mu fy(L^*)$$
$$m(L^*) = 0$$

These conditions are evaluated at $s=L^*$ instead of $s=s_c$ (as with Stage 1), since the end of the solution domain now coincides with the first point where the elastica is in contact with the floor, before the physical end of the elastica. Referring to Figure 3.4-b, this point has a zero y coordinate, since it is in contact with the floor, but has a frictional force applied to it, equal to the magnitude of the normal force fy, adjusted by a factor equal to μ (the frictional coefficient between the elastica and the floor).

For the point of contact between the roller and the elastica, the following boundary conditions apply:

$$x(s_c) = x_1$$
$$y(s_c) = y_1$$

where s_c is again the contact point along the arc length and x_1, y_1 are the coordinates of the contact point at a particular time. However, a third boundary condition, necessary for defining this problem, cannot be found. To overcome this problem, the elastica for Stage 2 will be considered from now on as *two distinct elasticae* : the first one runs from the start of the solution (coordinate origin) to the point of contact with the roller, with arc length s_1, and the second from the point of contact with the roller to the physical end of the elastica, with arc length s_2. Geometry continuity implies that the coordinates of the point of contact (x_1, y_1) and ending angle are common for both the left and right elasticae. For shape continuity, the curvature of the elastica at the point of contact has to be continuous. Therefore, the third, missing boundary condition can be written as:

$$m(s=s_1, \text{left part}) = -m(s=0, \text{right part})$$

where "left part" and "right part" refer to the two elasticae respectively and the arc lengths are given for each of the two elasticae. Of course, another equivalent condition for this point would be:

$$\theta(s=s_1, \text{left part}) = \theta(s=0, \text{right part})$$

referring to the continuity of the angle, but the first one is used for modeling simplicity.

The boundary conditions applying to Stage 3 refer to Figure 3.4-c:

$$x(s_c) = x_1$$
$$y(s_c) = y_1$$
$$m(s_c) \approx 0$$

where s_c is the point of contact of the elastica and the roller, along the arc length of the elastica, and x_1, y_1 are the x and y coordinates of the *current* roller contact point. Finally, the third boundary condition states that the moment at the point of contact is negligible. This is an approximation, since the "tail" of the elastica (the length after the contact point)

is hanging free and therefore applies a moment to the point of contact. This moment is small relative to the moment m[s=0], about 0.1%, so it can be neglected.

Once the governing equations for the elastica have been established and the appropriate boundary conditions for each operating stage have been introduced, it remains to apply the numerical method with these conditions to model the buckling feeder completely. The next chapter discusses this application in more detail and analyzes the problems that were encountered during the modeling and the steps taken towards overcoming them.

4. Modeling of Buckling with Large Deflection

In this chapter the mathematical foundation of Chapter 3 is applied to the particular problem of analyzing the operation of the buckling feeder. The first section introduces the objective of the analysis, or the parameters that this numerical solution will simulate. The next sections deal with the cases of Stages 1, 2 and 3 respectively, as defined in the previous chapter; the solution algorithm is presented and the difficulties and modeling assumptions are discussed. The last two sections discuss the modeling of folded and curved bills and the difficulties arising in the numerical solution of the elastica problem.

4.1 Objectives of the Analysis

The purpose of this project as a whole has been to increase the reliability of the operation of the buckling feeder and, at the same time, increase the speed in terms of number of bills counted per second. To achieve this goal, the operation of the buckling feeder has been simulated with the intention of gaining insight into which particular trajectories will be the most reliable. This section defines the parameters which are used as indicators of, not only the device's reliability, but also of the modeling process as a whole.

First, the conditions for which the operation of the buckling feeder is regarded as 'safe' are described. One of the problems in the operation of the feeder has been the case of slipping of the bill from the roller. Figure 4.1 shows a detail of the roller and the bill in contact. The force that is exerted between the roller and the bill can be separated into two

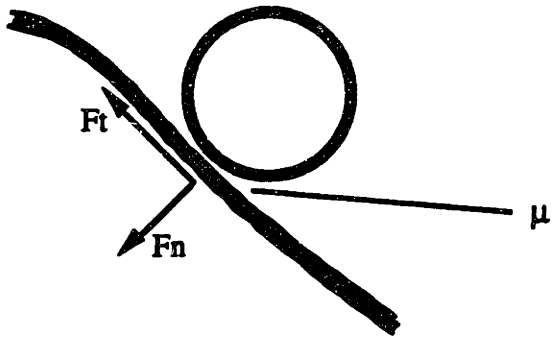


Figure 4.1: Detail of the roller and the bill in frictional contact.

components: one that acts tangentially to the surface of the bill and one that acts in the normal direction at the contact point. Since the question of stick or slip is being addressed, the horizontal component of the force is compared to the frictional force that arises between the roller and the bill due to their contacting surfaces with friction coefficient μ. The frictional force can be written as:

$$F_f = \mu F_n \tag{5-1}$$

where F_f is the frictional force between the roller and the bill, F_n the normal force from the roller to the bill and μ the coefficient of friction between the two surfaces. Slip will occur when the tangential force acting on the bill, F_t, is greater in absolute value than the frictional force. Hence, in the case of slip:

$$|F_t| > |F_f| \Rightarrow |F_t| > \mu |F_n| \Rightarrow \boxed{\left|\frac{F_t}{F_n}\right| > \mu} \tag{5-2}$$

To calculate the ratio at the point of contact, the following relationships are used:

$$F_t = | fx[c] \cos(\theta[c]) + fy[c] \sin(\theta[c]) |$$

$$F_n = | fx[c] \sin(\theta[c]) + fy[c] \cos(\theta[c]) |$$

$$R = F_t / F_n \qquad (5\text{-}3)$$

where the symbols have their usual meaning, and are evaluated at the node currently in contact with the elastica and R is defined as the ratio between the tangential and normal forces. Hence, by calculating the ratio of the tangential force to the normal force we can determine if the bill is likely to slip from the roller. Of course, this ratio applies for only one particular position of the bill and roller, so, in order to determine the overall reliability, this ratio has to be calculated for every discrete position in the bill's trajectory.

The second test, which gave some insight into the accuracy of the analysis, was the representation of the bill's shape under varying buckling conditions. The shapes determined by the numerical solution were compared to the shapes of real bills under similar deflection conditions; it was found that the shapes matched exactly.

A 2-dimensional 'map' of the stick/slip regions was produced by calculating the ratio at all points inside the operating region of the device. For this purpose, a mesh was generated (Figure 4.2), with mesh spacing 2mm in the x and y directions, and mesh size 140mm in the x direction and 60mm in the y direction. The ratio of quasi static tangential and normal forces was calculated at each node of the mesh, a node defined as the point of intersection of two mesh lines. Essentially each node in the mesh was tested as if it were the actual contact point. A contour plot was generated of the value of the ratio at all these points. The results obtained by numerical solution at multiple points in the operational area were compared with the experimental output and found to match the very closely. This comparison is given in Chapter 5.

To produce a typical contour plot, for some particular values of the design parameters (contact length and bill condition), only one initial condition has to be determined: the condition at rightmost bottom corner of the plot, ie. the starting point for

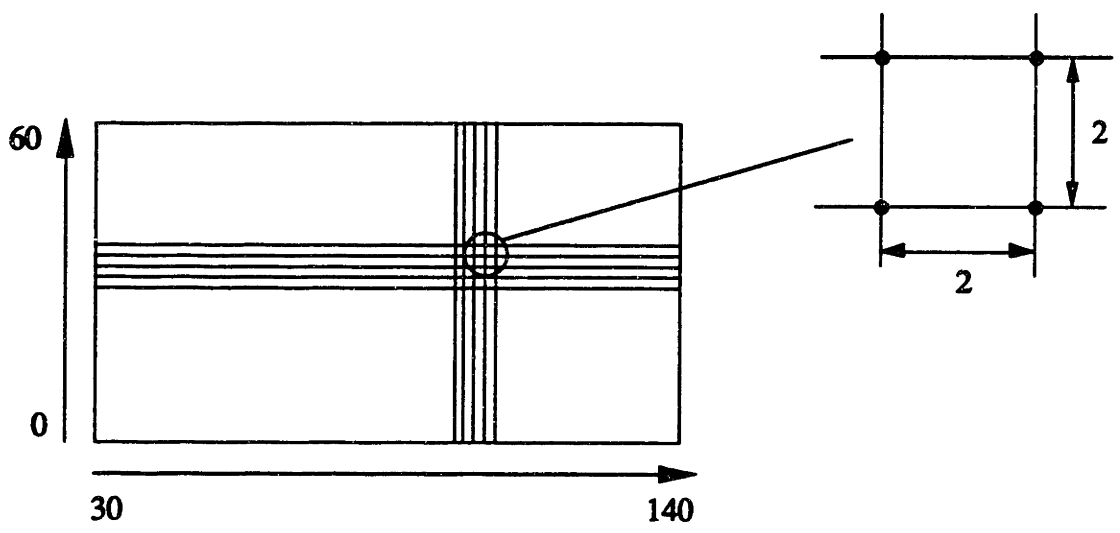


Figure 4.2: Dimensions of the computational domain (mesh) and of the spacing size used. (All dimensions in mm)

the sweep. Once the initial conditions for this point are given, the true conditions are found by applying the numerical method to solve for that point. These conditions are then used to calculate the point above, which is at a vertical distance of dy=2 mm. The value of fx[0], fy[0] and m[0] - the guessed variables - at the second point are then used as an input for the third point etc. Once all the points in a particular column have been solved for, the values of the three missing conditions are used as guesses for the initial conditions of the corresponding points of the same y-position and exactly to the left (Figure 4.3). By using this sweep, the whole domain is covered.

The following sections give an analytical description of the solution method, accompanied by specific examples for each of the three different stages.

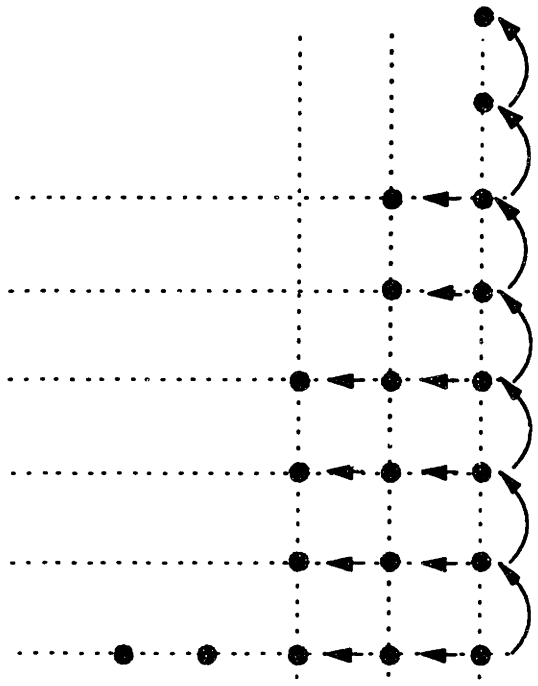


Figure 4.3 : Sweep method for calculating the ratio at all nodes in the computational domain. The initial values for the start of the solution are passed as shown.

4.2 Analysis of Stage 1

The boundary conditions at the end of the elastica are given by:

$$x(s_c) = x_2$$
$$y(s_c) = 0$$
$$\theta(s_c) = 0$$

The position to which the roller moves is given in terms of the distance x_2 from the origin. The contact length s_c is also known as the distance from the origin at which the roller initially makes contact. It is reasonable to impose the condition of the angle θ to be

equal to zero at the end of the elastica, since there are no forces acting on the elastica after the point of contact which would change the angle. It is therefore expected that a Stage 1 bill will be symmetrical with respect to the line $x=(x_2)/2$, not only in regard to its shape but also in terms of the forces and moments.

The problem of sticking or slipping is not an issue in Stage 1 analysis, as explained in Chapter 2, since the roller can always apply a sufficient normal force to the bill in order to avoid slip. However, the problem of double-buckling may arise, if the normal force exceeds a certain limit; this has been dealt with previously (Section 2.2). The focus of this section is, therefore, to produce the bill shapes that correspond to this stage of the paper-feeding process, through a specific example.

Considering the following example of a bill buckled at $x_2 = 10$ cm, with $s_c = 12$ cm, an estimate of the missing initial conditions is required for the unknown quantities at the s=0 (fx, fy and m). As a first reasonable estimate for these values, the following approximations could be considered: fx should be greater than the horizontal buckling force, hence:

$$fx[0] > F_b = \frac{4\pi^2 E I}{L^2} = -0.23 \text{ N}$$

which is calculated with the material values found in Appendix III and a margin of about 10 times. The normal force fy can be found from the weight relationship:

$$fy[0] = \frac{w L}{2} = 0.0055 \text{ N}$$

Finally, the moment can be estimated by using the relationship relating it to the local curvature:

$$m[0] = \frac{E I}{r} = -0.0047 \text{ Nm}$$

by estimating the radius of curvature at s=0 to be about 2 cm. With these initial guesses, the solution converged to the final shape (Figure 4.4), the number of iterations needed being 8 and the solution time under 1 second, for an error of 2×10^{-5} with respect to the end position of the elastica.

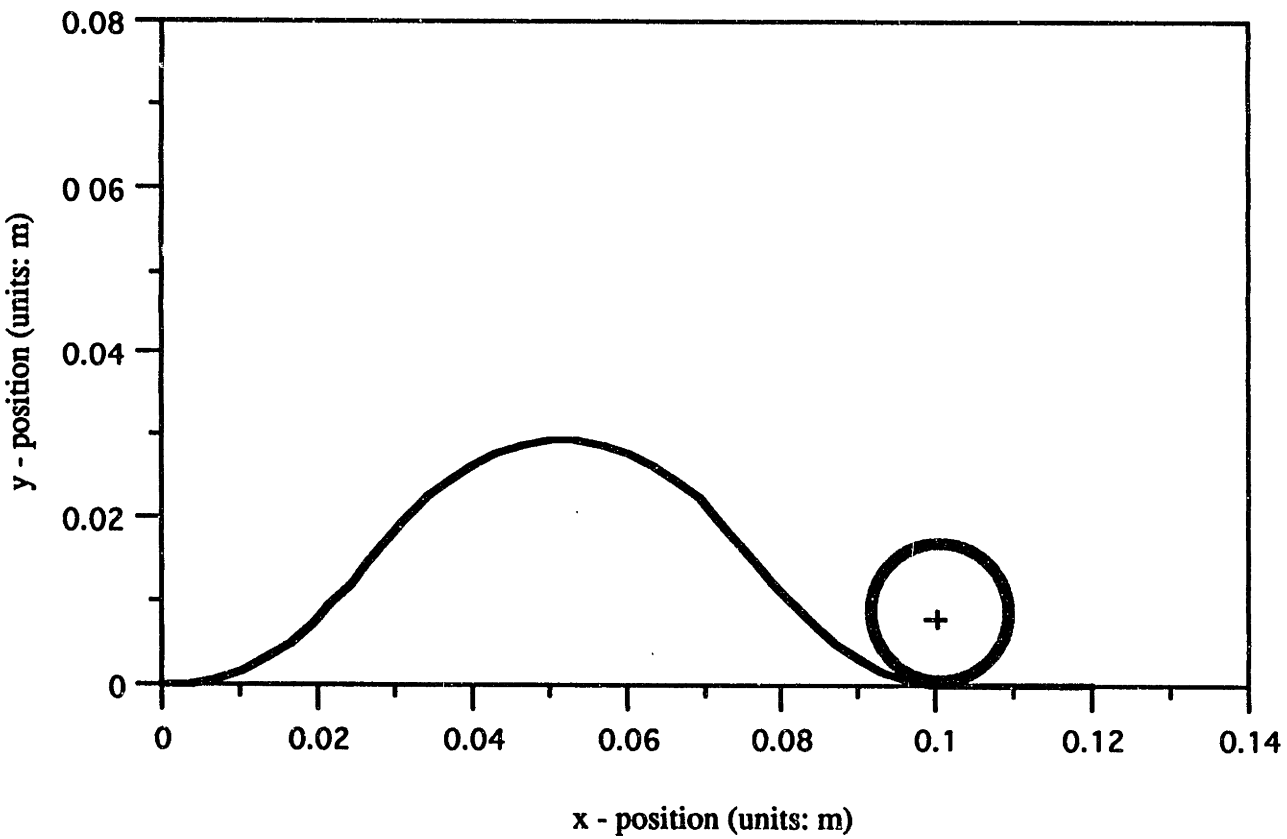


Figure 4.4 : Shape of bill under Stage 1 buckling (to scale).

4.3 Analysis of Stage 2

The methods for solving the governing equations for elasticae under deformations corresponding to Stage 2 were more difficult. Stage 2 is essentially a three-point boundary value problem, since there are three points along the length of the elastica which are in contact with some external boundary: the start, the end and the point of contact with the roller.

The development of the algorithm which calculates Stage 2 bills was derived from the concept that Stage 2 is actually an intermediate stage between Stages 1 and 3. Actually, Stage 1 imposes large restrictions on the shape of the bill by forcing a large change in curvature near the contact point. Just as the roller is lifted from the floor to a higher position, the restriction on the shape of the bill is relaxed and the curvature and moment of the bill decreases.

The transition from Stage 2 to Stage 3 buckling occurs when the end of the bill is lifted off the floor. The mechanics of the transition (Figure 4.5) show that just as the bill is lifted off the floor, the moment at the point of contact becomes positive and equal to the moment of the 'tail', caused by the weight of the hanging section. A positive moment implies a downwards-concave curvature. On the contrary, a negative moment indicates an upwards-concave curvature. The limiting case, when the moment is equal to zero corresponds to a straight elastica. Therefore, the transition from Stage 2 to Stage 3 can be detected when the moment at the contact point is approximately equal to the positive moment of the weight.

Stage 2 was solved by separately calculating two distinct elasticae which join at the point of contact with the roller. For the left elastica the boundary conditions at the point of contact are:

$$x(L_c) = x(C_n)$$
$$y(L_c) = y(C_n)$$
$$\theta(L_c) = \theta(C_n)$$

where L_c is the current contact length of the bill, marking the end of the left elastica, and C_n the trial contact point whose coordinates and angle of contact have to be determined.

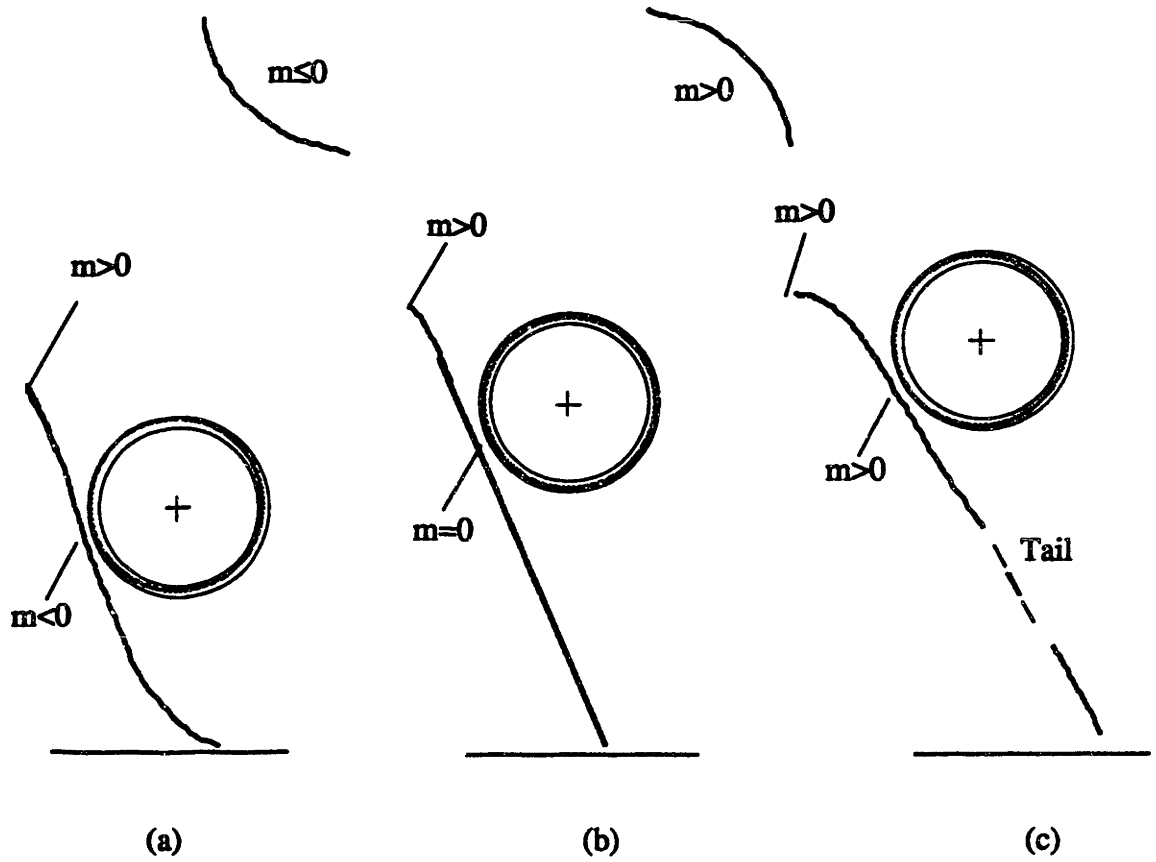


Figure 4.5 : Transition between Stages 2 and 3; (a) moment at contact point with roller is negative, (b) moment at contact with roller is equal to zero (bill just touching floor), (c) moment at contact is positive (bill is not touching the floor).

To solve for the right elastica, the conditions given above were used as initial known conditions. Hence,

$$x(0) = x(C_n)$$
$$y(0) = y(C_n)$$
$$\theta(0) = \theta(C_n)$$

and, for the end of the right part,

$$y(L^*) = 0$$
$$fx(L^*) = \mu\, fy(L^*)$$
$$m(L^*) = 0$$

It must be noted that L^* corresponds to the arc length of the bill at which the angle is first equal to zero and not the physical end of the bill, since the no forces are applied to the resting part of the bill (from L^* to L), which can be neglected. The distance of L^*, however, is not known, so the boundary conditions are initially applied to the physical end of the bill (ie. at s=L). The problem that arises with these ending conditions is that the bill sometimes 'goes below the floor' (Figure 4.6). This is an acceptable solution numerically but cannot be possible under real conditions. In order to find the real contact point at L^*, the length of the right elastica is decreased and the solution is tried again until the bill does not fall below the floor, ie. until the correct length L^* is determined. The subtracted length ($L - L^*$) is added back to the elastica; this is the region that rests on the floor.

As a first step to calculating Stage 2, the last position in Stage 1 buckling was considered, with the center of the roller at point A. The effect of an upwards movement of the roller to a point B was then examined (Figure 4.7). The old contact point is designated by C_1, while the new contact point, which is not known, will lie somewhere in the arc C_2

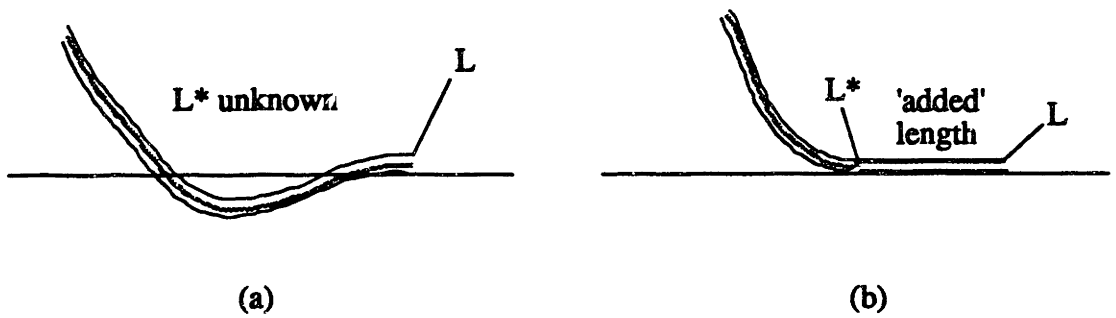


Figure 4.6 : If the elastica goes below the floor - (a), its length is decreased until the correct shape and the first contact point with the floor is found (L^*) - (b).

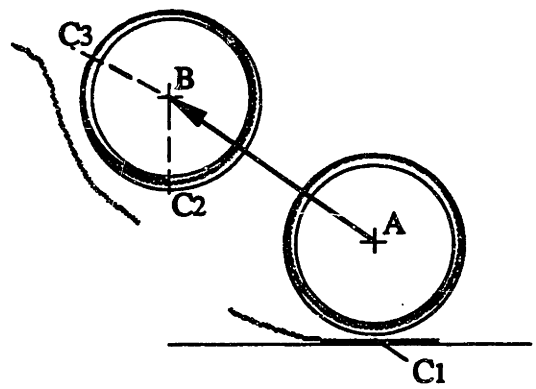


Figure 4.7 : Analysis of transition from Stage 1 to Stage 2

C_3, since this is the area of the roller likely to be touching the elastica. The exact point of contact was not known, so several points lying on the arc C_2 C_3 were tested, starting from C_2, each time adding a constant increment to the angle to obtain the new point. For each point, the left elastica was solved and the moment at its end (the possible contact point) was determined (M_l). Then, the right elastica was solved and the moment at its start (M_r) was calculated. A separate loop in the program checked to see if the right elastica slipped below the floor and, if so, made the necessary adjustment.

The two moments from the left and right elastica (M_l and M_r) were compared to determine if the test point was actually the contact point. Since the two elasticae are continuous at the point of contact, it follows that position, angle and moment continuity hold true at that point. If the two moments are not equal, then the angle is updated and the next point is tested. If, however, M_l and M_r are equal at a particular point between C_2 C_3, that point will be the contact point. If both moments are negative, the resulting shape of the bill corresponds to Stage 2 deformation whereas both moments positive indicates Stage 3 conditions. This is the criterion for transition. Figure 4.8 shows the flowchart of the algorithm used to find the contact point.

As an example, the following case is considered, starting at the last point of Stage 1 buckling, with a contact length s_c = 12 cm and roller position x_2 = 10 cm (the example

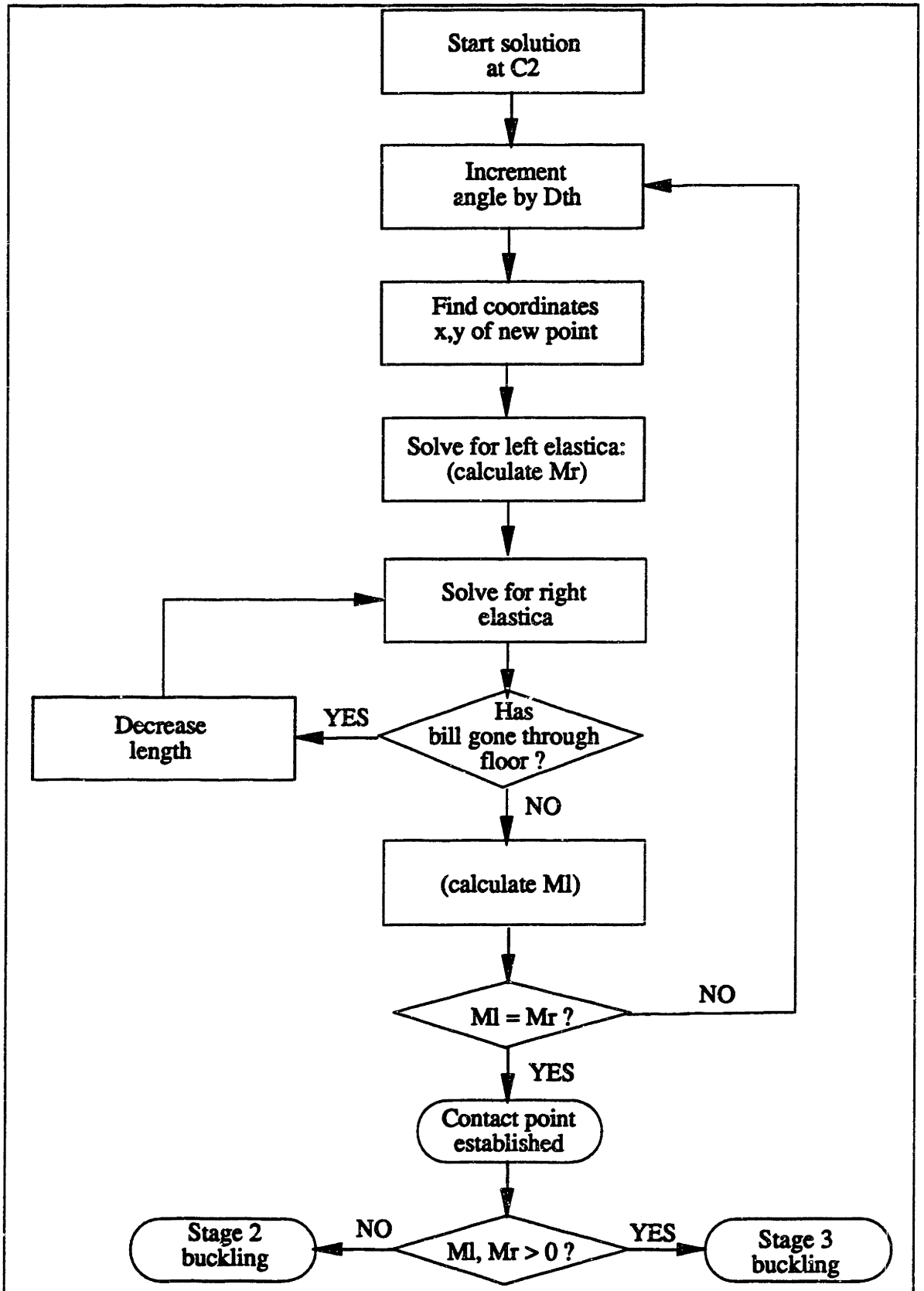


Figure 4.8 : Flow chart of the contact point estimation algorithm

of Section 4.2) and moving the roller from (x, y) = (10, 2) cm to (x, y) = (4 , 2) cm in 2 cm increments. The resulting shapes of the bill under Stage 2 buckling are presented in Figures 4.9 to 4.12.

4.4 Analysis of Stage 3

The boundary conditions at the end of the solution domain are:

$$x(s_c) = x_1$$
$$y(s_c) = y_1$$
$$m(s_c) \approx 0$$

the symbols having been defined in Section 3.4. The coordinates x_1 and y_1 are the desired coordinates for the end of the elastica and the roller is assumed to be contacting the elastica at this point.

To calculate the exact bill shape under conditions of Stage 3 buckling, several assumptions were made. First, it was assumed that the coordinates of the contact point were known. This assumes that the bill has not slipped from the roller, equivalent to 'fixing' the contact point to the roller, until it reaches this point. In reality, however, slip might have occurred at an earlier stage so the contact point would change. But when all the possible 'slip areas' in the operational space are considered, they can be avoided by changing the trajectory of the roller.

Another simplification was that the moment at the contact point was approximately equal to zero. This is a reasonable assumption, since the moment due to the hanging part of the elastica is approximately 0.1% of the highest moment in the elastica. Furthermore, when the boundary condition was altered to compensate for the moment at the point of contact, the resulting change in the angle was of the order of 0.1 degrees, which was beyond the required level of accuracy.

Throughout this analysis for Stage 3, it was assumed that there is only one contact point between the bill and the roller. This may be an approximation for some cases, nonetheless adequate for modeling purposes, because it introduces an error on the

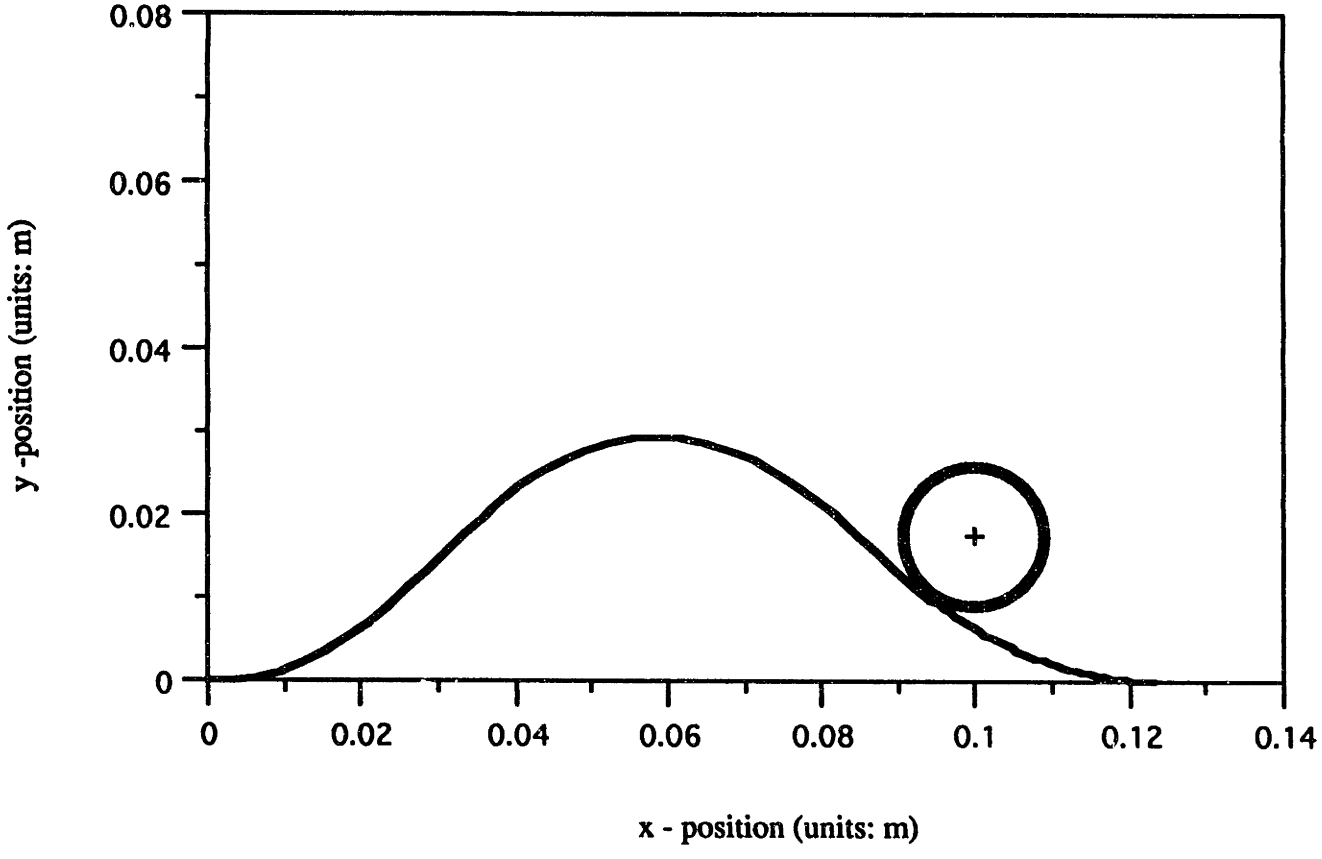


Figure 4.9 : Shape of bill under Stage 2 buckling, roller coords (0.10, 0.018) [TO SCALE]

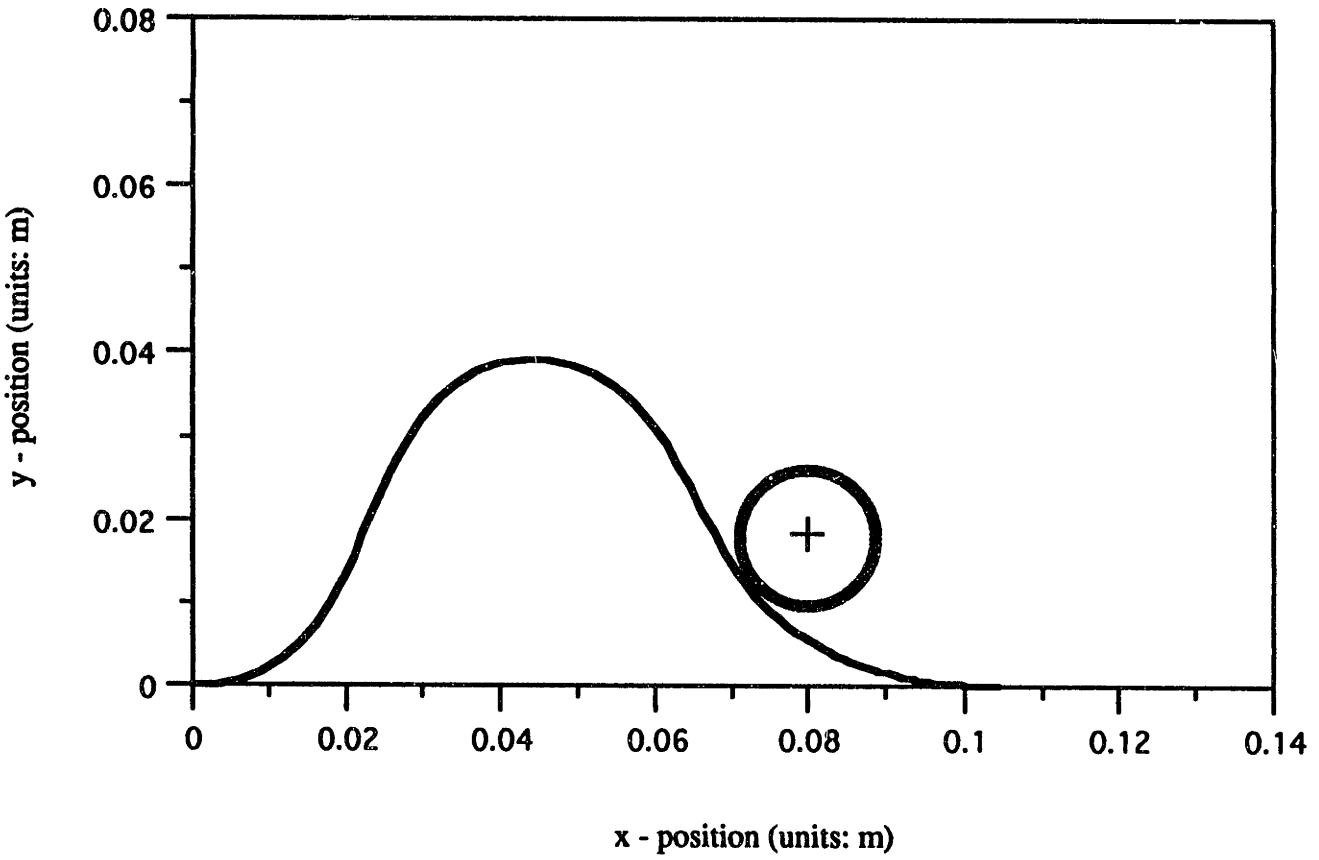


Figure 4.10: Shape of bill under Stage 2 buckling, roller coords (0.06, 0.018)
[TO SCALE]

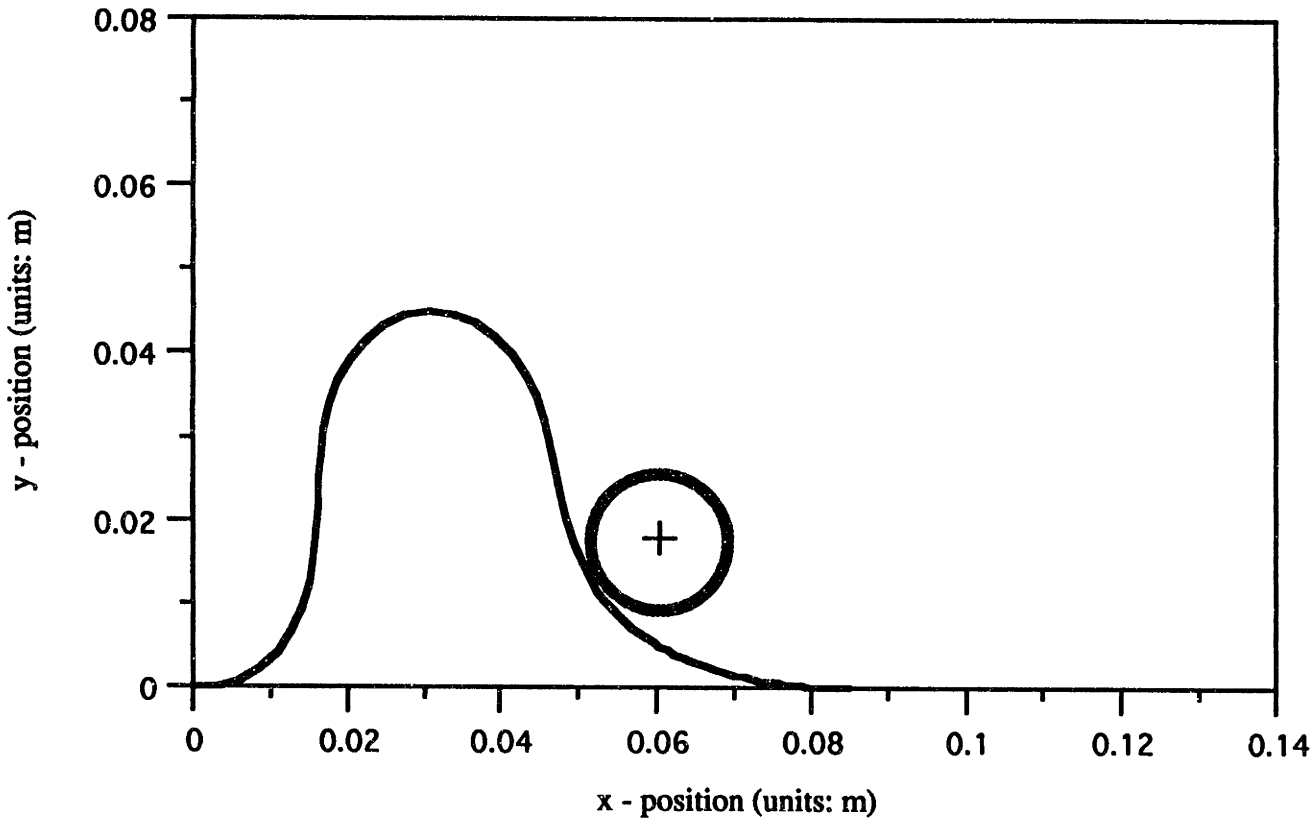


Figure 4.11: Shape of bill under Stage 2 buckling, roller coords. (0.06, 0.018) [TO SCALE]

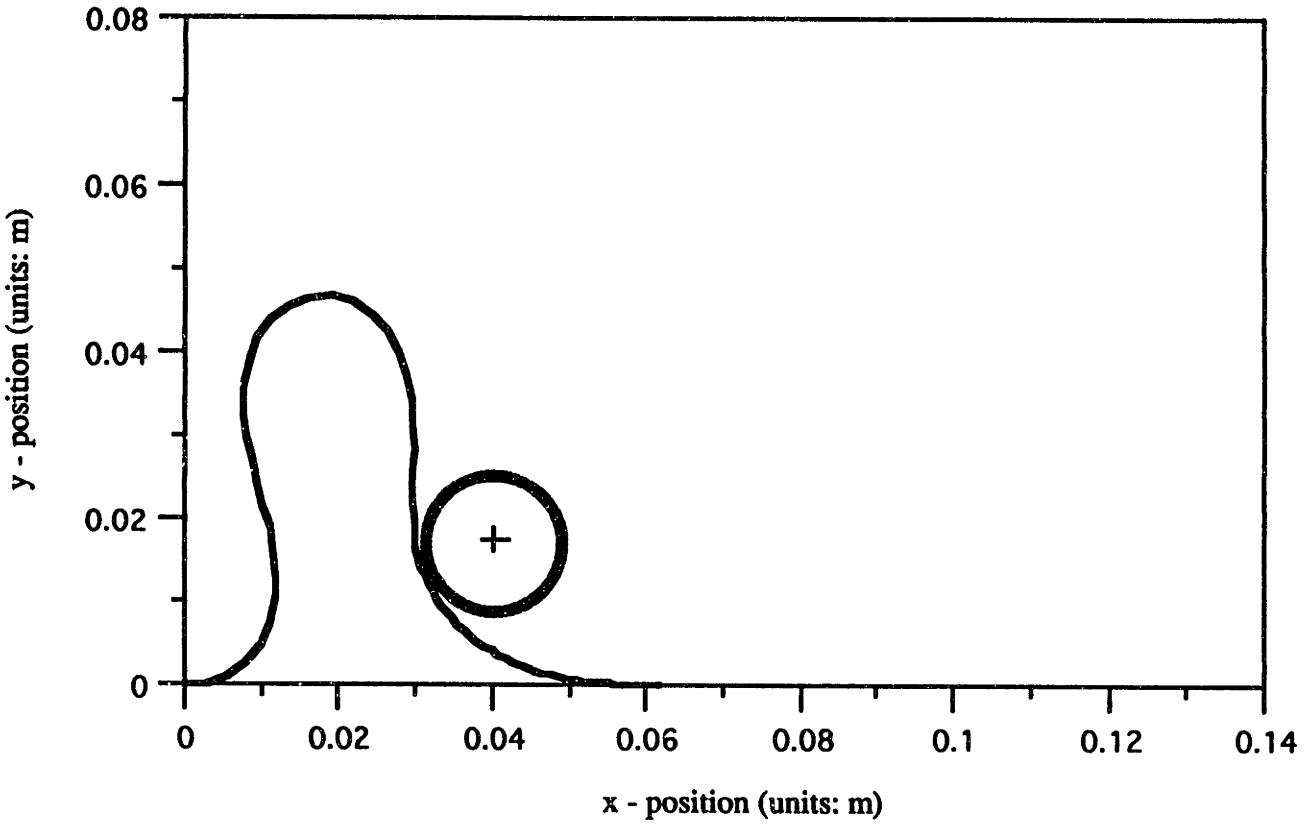


Figure 4.12 : Shape of bill under Stage 2 buckling, roller coords (0.04, 0.018)
[TO SCALE]

conservative side; the larger the contact area the less likely does slip become. Furthermore the analysis is simplified to a large extent.

One of the final problems in Stage 3 analysis was the problem of roll, during which the contact point between the bill and the roller changes, without the bill having slipped from the roller. A three-step algorithm was developed to test if the bill had rolled and adjust the contact length and contact point once roll had occurred. First, the roller was moved to a new position; it was hypothesized that the bill did not roll on the roller and that the contact length remained the same when the roller moved to the new position. Then, the x, y coordinates of the new contact point were calculated and the numerical solution was applied with the new boundary conditions, solving for the forces fx and fy and the ending angle θ. Finally, the new contact point coordinates were found from:

$$x_c^{k+1} = x_c^k + (x_r^{k+1} - x_r^k)$$

$$y_c^{k+1} = y_c^k + (y_r^{k+1} - y_r^k)$$

$$s_c^{k+1} = s_c^k \qquad (5\text{-}4)$$

The equations refer to Figure 4.13. The superscript indicates the iteration step; the subscripts c and r indicate the contact point and the roller center respectively. The basic variables have their usual meaning. The angle θ^{k+1} was compared to the ending angle of the previous position θ^k; roll had occurred when the angles differed. New x, y coordinates were then established, based on the new ending angle:

$$x_c^{k+2} = x_c^{k+1} - r\,(\sin\theta^{k+1} - \sin\theta^k)$$

$$y_c^{k+2} = y_c^{k+1} - r\,(\cos\theta^{k+1} - \cos\theta^k)$$

$$s_c^{k+2} = s_c^{k+1} - r\,(\theta^{k+1} - \theta^k) \qquad (5\text{-}5)$$

The superscripts and subscripts have the same meaning as above and r is the radius of the roller. This scheme is applied iteratively, as shown in the flowchart of Figure 4.13, until the coordinates of the contact point converge to a steady value. These are the final

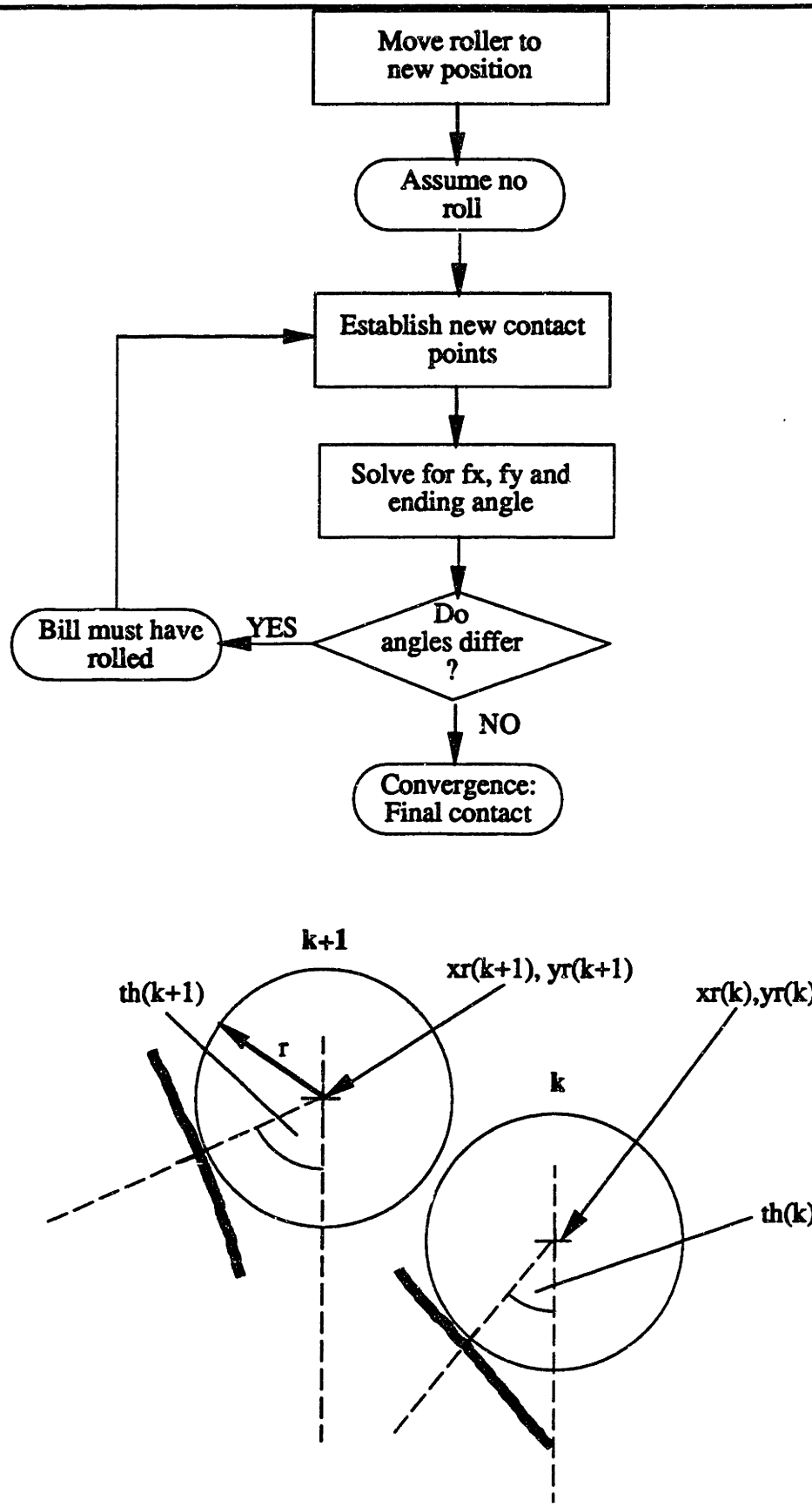


Figure 4.13 : Flow chart and description of roll algorithm

coordinates and new contact length, adjusted for roll. The ratio can be therefore be calculated for these new coordinates.

To illustrate Stage 3 analysis the following example is given for a bill having an initial contact length of s_c = 12 cm. For this example, it was assumed that the end of the elastica was located at the point with x,y coordinates 8 and 2 cm respectively.

Again, it was necessary to provide guesses to the missing initial conditions. Based on similar reasoning as with the previous example of Stage 1, it was calculated that:

$$fx[0] = -0.23 \text{ N}$$
$$fy[0] = w\,L = 0.011 \text{ N}$$
$$m[0] = -0.0047 \text{ Nm}$$

The required number of iterations for the solution this problem was 8, and the solution time, again, under 1 second for a position error of 2×10^{-5} with respect to the desired ending position. The resulting shape of the bill for these coordinates is shown in Figure 4.14.

4.5 A Different Design

To this point, the analysis of the bill feeder has been based on the roller design introduced in Chapter 1. The model was developed, tested and results were produced (Chapter 5). Based on these results, suggestions were made regarding the ideal trajectory of the roller. The design was modified to eliminate the problems the analysis had discovered. This section discusses the model developed to simulate the operation of the new roller; the modeling principles are exactly the same as before. Hence, this section serves as an illustration of the application of our methodology to new designs. The modeling process becomes very easy when the boundary conditions applicable to each stage are identified and dealt with accordingly.

The new roller design is similar to the old one. The full roller has been substituted by a 'segmented' roller, a section of the old roller, extending to a specific arc length (θ_{seg}). The center of the roller is fixed at a certain point and the roller is only allowed to rotate clockwise while buckling the top bill (Figure 4.15).

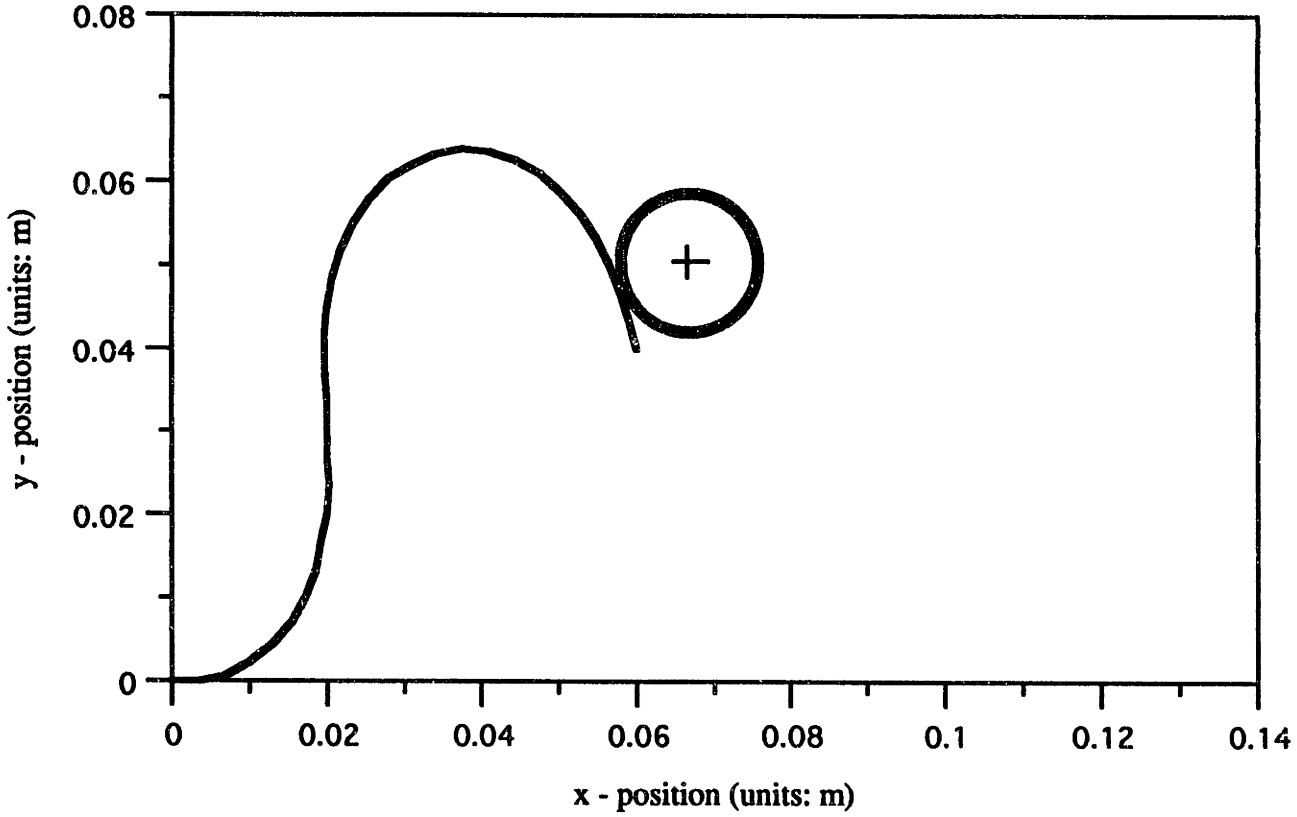


Figure 4.14 : Shape of bill under Stage 3 buckling, roller coords (0.066, 0.05)
[TO SCALE]

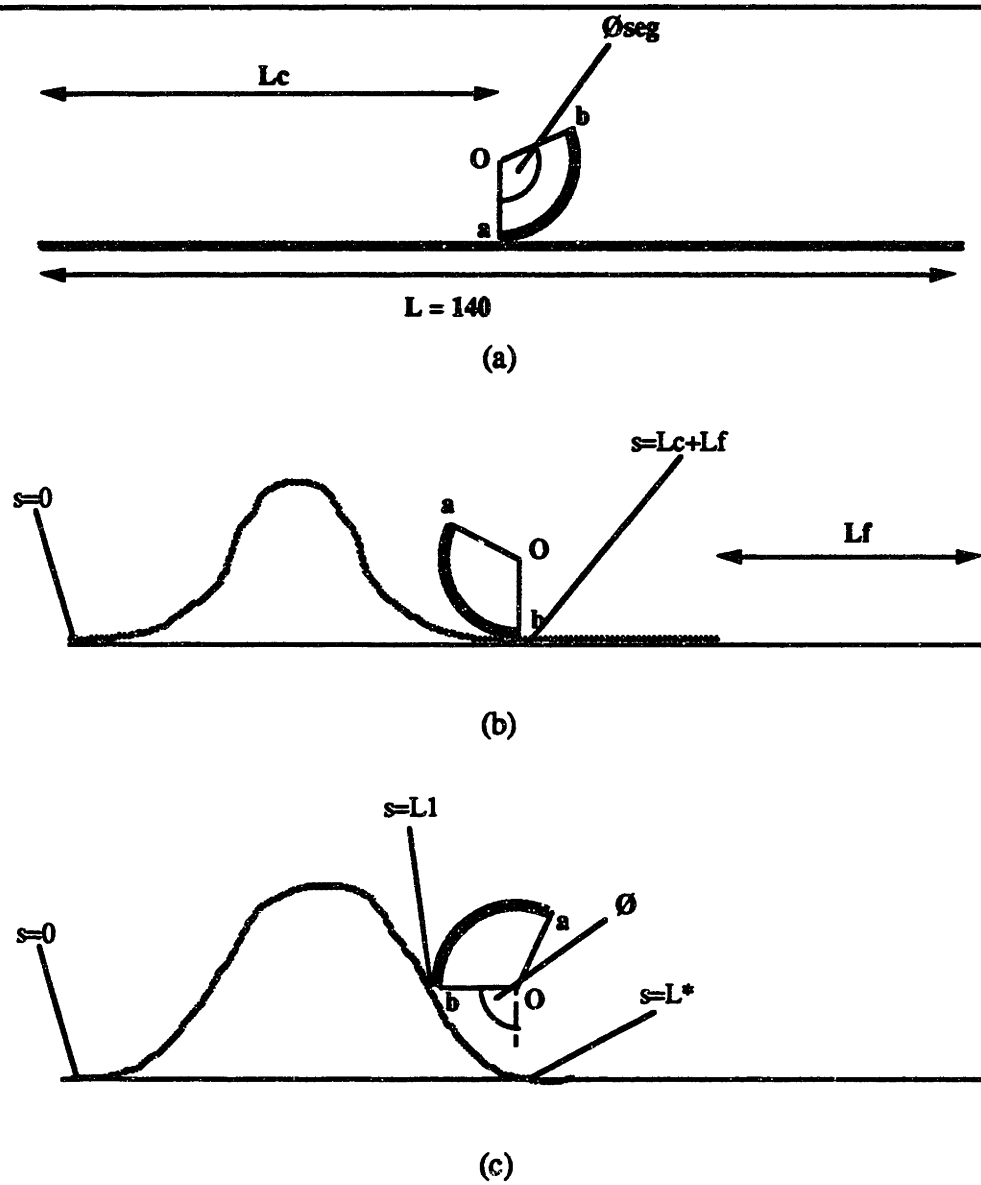


Figure 4.15: Operation of the new roller design; (a) to (b) feeding, (b) to (c) lifting.

The motion of the roller can be, once again, separated into two phases: The first is the 'feeding' phase, when one part of the arc length of the roller is always touching the

floor (Figure 4.15-a to 4.15-b), and the second when only point 'b' of the roller is touching the bill (Figure 4.15-c).

Inspection of the above statement reveals that for the first part, the relevant boundary conditions are those applicable to Stage 1 (as defined earlier); for the second part, the boundary conditions are equivalent to those of Stage 2. Making this formulation more formal, for the lifting stage first:

$$x(s=0) = 0 \qquad x(s=Lc+Lf) = Lc$$
$$y(s=0) = 0 \qquad y(s=Lc+Lf) = 0$$
$$\theta(s=0) = 0 \qquad \theta(s=Lc+Lf) = 0$$

where the left column of B.C.'s refers to the start of the elastica and the right column to the point at which the roller is touching the floor. The length Lc indicates the initial contact length of the roller on the bill, while Lf refers to the length of the bill fed by the roller during the feeding phase. It should now be clear that the bill length to the left of point 'b' (in Figure 4.15-b) is the sum of the two lengths, Lc+Lf.

Figure 4.15-c is reminiscent of Stage 2, as discussed in Section 4.3. Again, it is useful to separate the bill into two distinct elasticae which have the boundary conditions:

-for the left elastica:

$$x(s=0) = 0 \qquad x(s=L_1) = xc$$
$$y(s=0) = 0 \qquad y(s=L_1) = yc$$
$$\theta(s=0) = 0$$

-for the right elastica:

$$x(s=0) = xc \qquad fx(s=L^*) = \mu\, fy(s=L^*)$$
$$y(s=0) = yc \qquad y(s=L^*) = 0$$
$$m(s=L^*) = 0$$

where L^* is the arc length corresponding to the first point of the right elastica touching the floor and xc, yc are the coordinates of the contact point with the roller, given by:

$$xc = Lc - r\sin\theta$$
$$yc = r\,(1 + \sin(\theta - \pi/2))$$

where r is the roller radius and θ is the angular deviation of the segment Ob with the vertical.

As with Stage 2, only two boundary conditions can be specified at the common point of the two elasticae. The third condition has to be found iteratively. Contrary to Stage 2, however, the bill can be considered 'pinned' to the roller at the contact point (b). This means that the elastica will assume the angle that will minimize the moment at the point of contact. To find the correct angle of the elastica at that point, an iterative scheme was used: a trial angle was used as the third boundary condition (for the left and right elasticae) and the numerical solution was sought for that angle. The resulting moments from the left and right were compared and an updated guess was used, until the moments converged to the same value.

Using principles and guidelines developed in Chapters 3 and 4, a model was developed for the simulation of the modified feeder. The simplicity of the model's creation suggests that the developed methodology is generally applicable to most processing operations for flexible materials.

4.6 Modeling of Imperfect Bills

There are many types of bills found in a typical stack, ranging from crisp new bills to torn and delicate old bills. The modeling should include bills of different conditions. Predictions of the effect of the varying bill conditions on the double-buckling problem have been given in Section 2.2. The modeling of bill of various conditions in the lifting operating step is given below.

Two types of deformed bills, which are the ones most frequently found, are considered: folded and curved bills. Both types are shown in Figures 4.16 and 4.17 respectively, under conditions of Stage 3 buckling. The first type of bill is creased in the center of the bill in the direction of the width, such as the crease caused by folding a piece of paper. The second type is the result of putting the bill in a wallet: it contains a defect in the same direction as before, but it is not folded as hard.

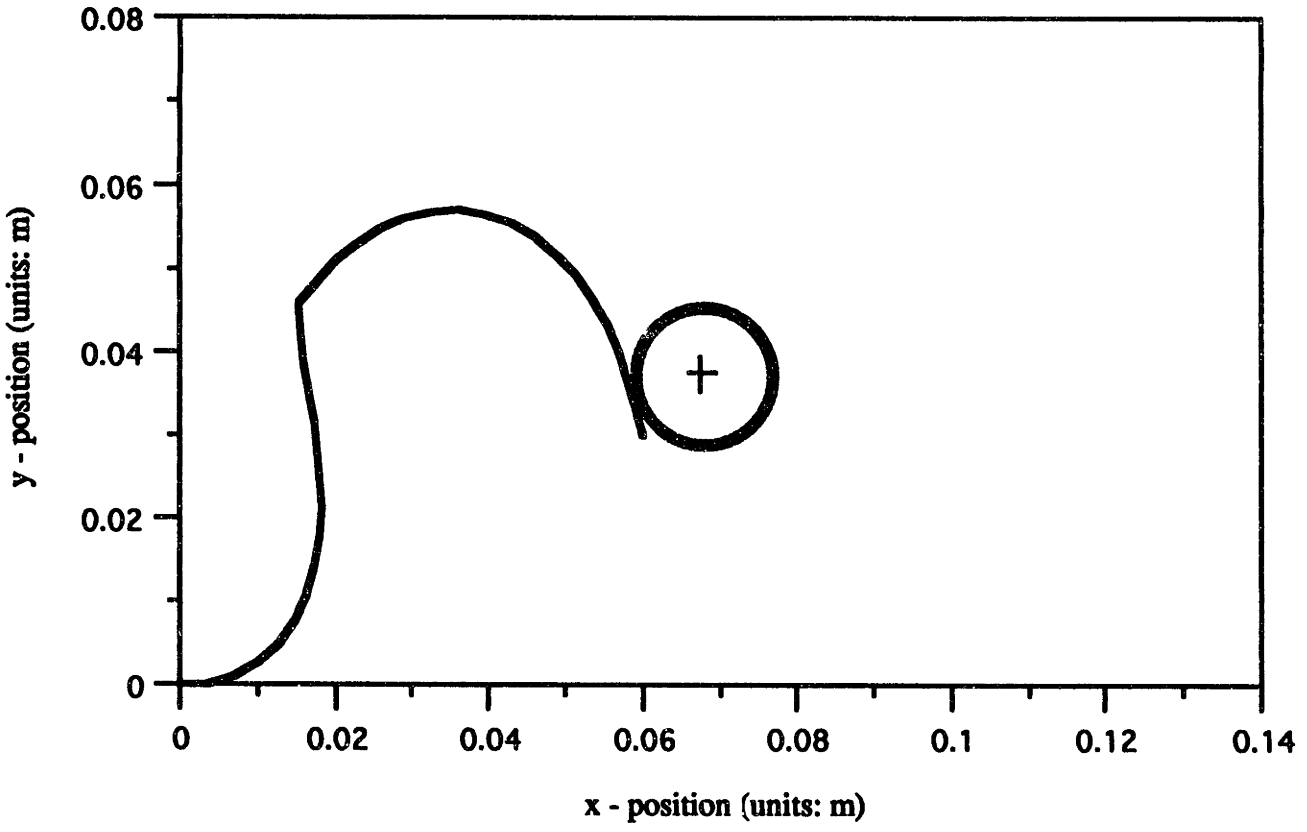


Figure 4.16 : Shape of folded bill under Stage 3 buckling
[TO SCALE]

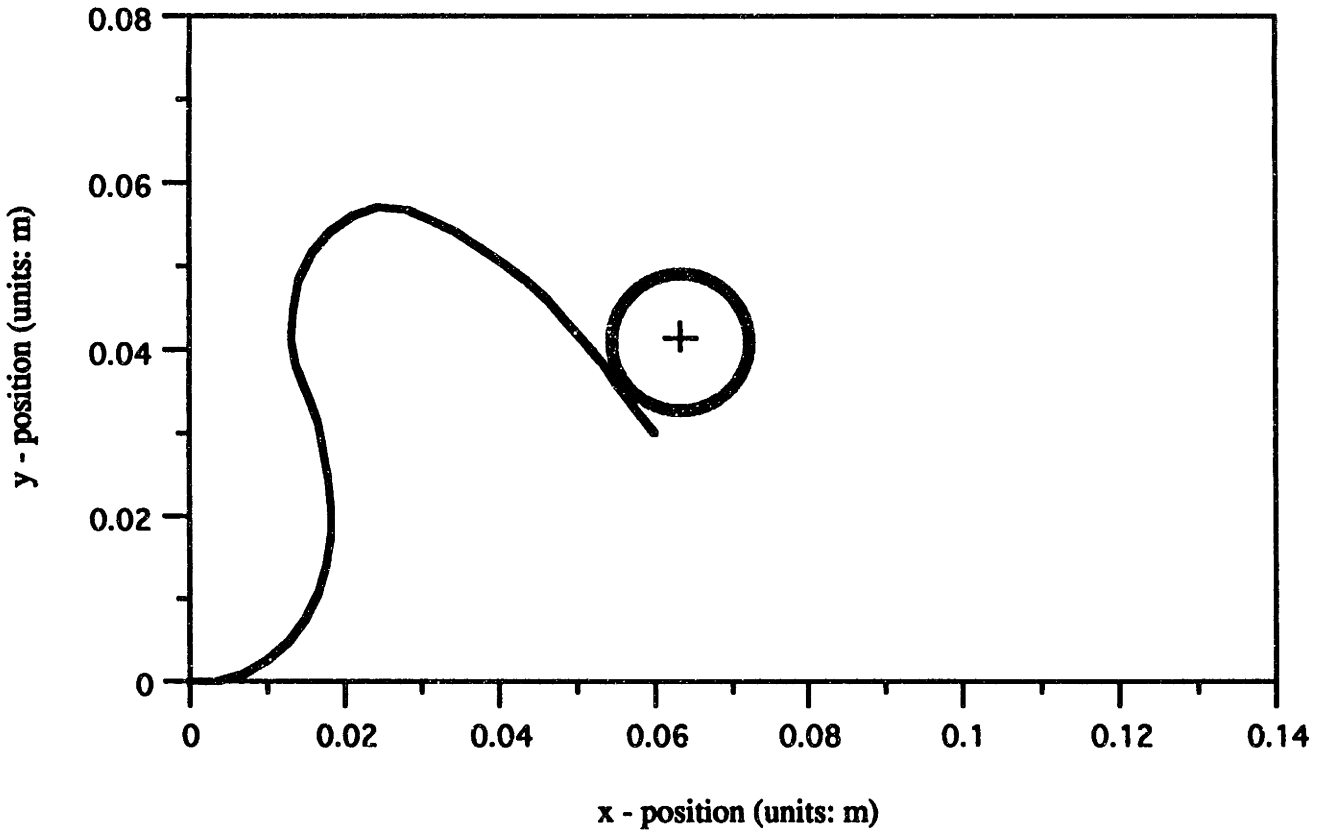


Figure 4.17 : Shape of curved bill under Stage 3 buckling
[TO SCALE]

The characteristics of the first defect is that the bill is very weak at the point where it is folded, while the two 'flaps' have the same stiffness as the original bill. To model this type of bill, the stiffness of the central element (which contains the crease) was decreased by a factor of 10 relative to the other elements. The value of 1/10 is an arbitrary factor, empirically derived, but produces results that match the experimentally measured behavior of folded bills well.

Modeling of curved bills is a more complicated procedure. The effect of placing the bills in a wallet is to induce a permanent deformation in the shape of the bill, which increases the curvature (locally) close to the center of the bill. This higher curvature corresponds to an increase in the moment at each of these elements that are in the deformed section. The question arises, however, as to how to determine the right moment for these central elements. Figure 4.18 illustrates this procedure: The radius and extent of curvature are determined by observation of a deformed bill specimen. The curved part of the bill is then treated separately from the rest of the bill, as a distinct elastica, and is considered fixed at A. The initial and final conditions are then:

$$x[0]=0,\ x[L_c]=AB$$
$$y[0]=0,\ y[L_c]=0$$
$$\theta[0]=\theta_1,\ \theta[L_c]=180-\theta_1$$

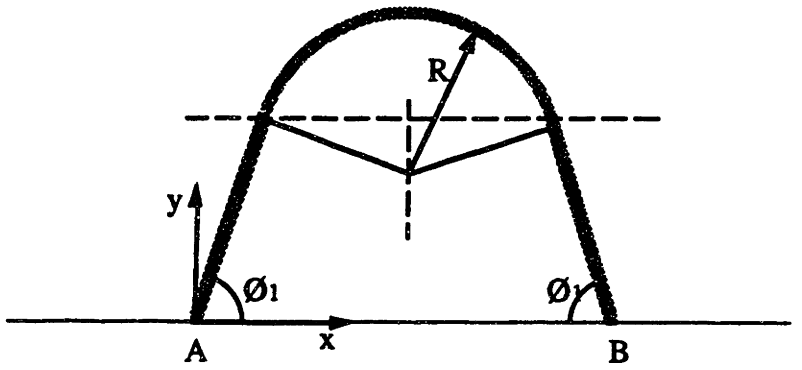


Figure 4.18 : Modeling of curved bills: dimensions taken from a specimen bill

where L_c is the length of the curved area, θ_1 the angle with the horizontal, and AB is the x-direction distance between the ends of the curved section. Solving this particular problem using the same procedure described above, with guesses for fx[0], fy[0] and m[0], yields values for the moment at each element of the curved section. These moments are then added to the moment m of equation (3-6) which becomes:

$$\boxed{\frac{\partial\theta}{\partial s} = \frac{m+M[i]}{EI}} \qquad (5\text{-}6)$$

where the symbols have their usual meaning; M[i] is the added moment of the i'th element. Elements not belonging to the curved region have M[i]=0.

4.7 Problems in the Calculation of the Shape of the Elastica

In developing numerical methods, amongst the most important problems are the speed and robustness of convergence. Since the mathematical model is non-linear, the convergence of the solution depends on the choice of initial conditions. The initial conditions for this problem, for all three stages, are the values of fx[0], fy[0] and m[0] at the start of the solution, which are the missing boundary conditions. Guessing these initial conditions may prove to be a difficult task.

If the guessed initial conditions are not 'close' to the actual conditions, where 'close' varies from an error of 0.1% to 5% depending on how constrained is the particular solution, the solution may either converge to the 'wrong' answer or not converge at all. An example of a 'wrong' solution is given in Figure 4.19. This is a mathematically acceptable solution; in physical terms, however, it does not make any sense. Therefore, solutions as such were discarded from the analysis. To overcome solutions which are meaningless in terms of the model, a rule was used, based on the number of changes of curvature of the bill's shape. If that number exceed two (in the case of Stage 3 buckling) or if the number was greater then three (in the case of Stage 1 and 2 buckling) then the solution was not acceptable.

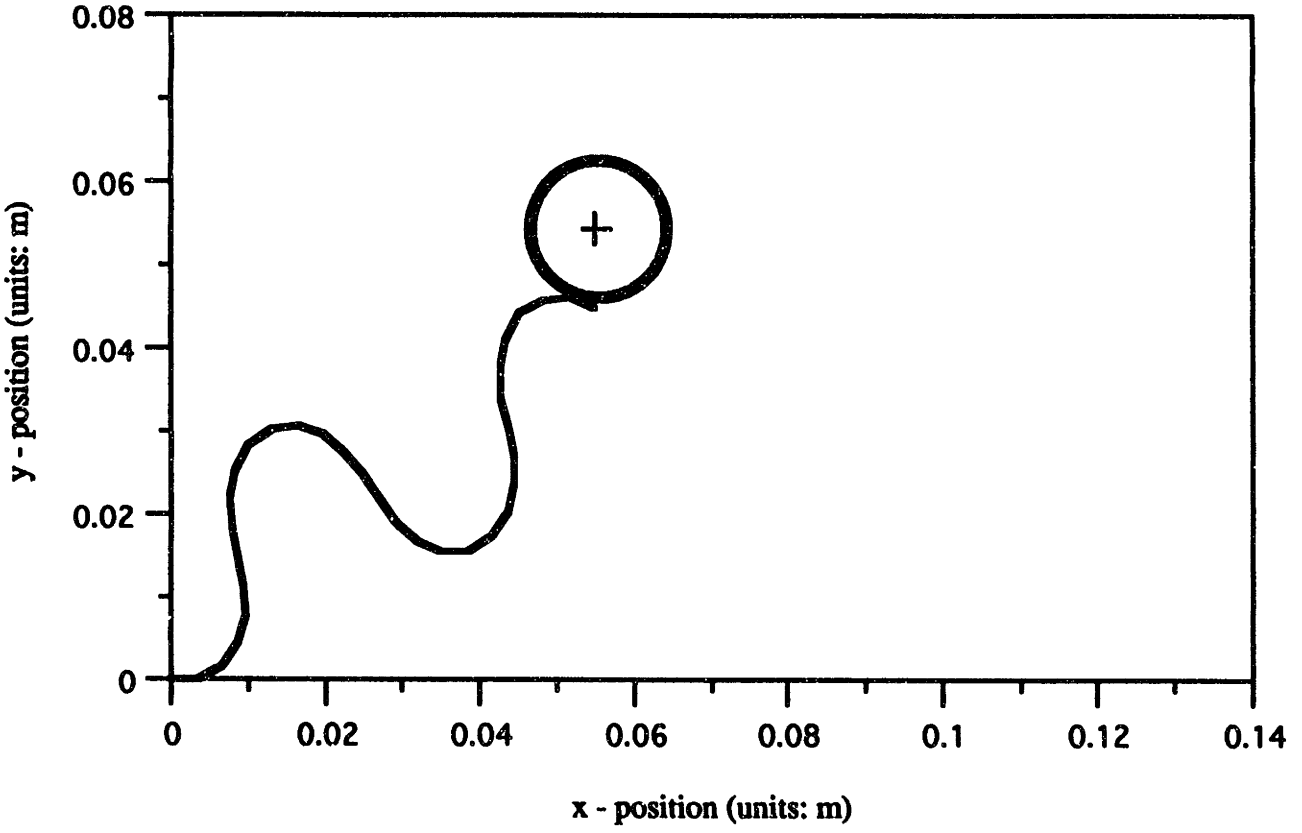


Figure 4.19 : An example of convergence to the 'wrong' solution
[TO SCALE]

There is no systematic and reliable method to calculate the initial conditions for the elastica. A guess is made and, depending on the output (how quickly the solution converges), the guess is changed so that 'better' initial conditions are used. However, once the initial conditions for a single point have been established (for a certain value of the contact length and bill condition) initial conditions at neighboring points can be calculated as described in Section 4.1.

5. Results of the Numerical Solution

This chapter presents the results of the numerical simulation of the original design of the buckling feeder, predicting possible slip regions in the operating space of the feeder. The results are given in the form of a 2D-contour plot, the two dimensions being the x and y distance from the support, and the z variable being the ratio of Ft/Fn in the operating space of the roller. The numerically-calculated results are compared to the measurements taken from the prototype buckling feeder.

The contour plot is presented in Figure 5.1. The ratio of the forces is denoted by different colors in the contour plot. The contour plots were drawn from tables of data of the ratio Ft/Fn, generated by the numerical simulation for a new bill and for a contact length l=12 cm. Contour-plotting software was used to interpolate between the data and to produce the color contour plots. A custom-written algorithm was developed to allow the plotting software to read the numerical data.

The downward-sloping black line indicates the transition from Stage 2 buckling to Stage 3 buckling. The data for each stage was calculated by separate programs (see Appendix II). The diagonal 'cut' in the upper-right part of the contour plot indicates areas which the bill cannot reach, limited by its length. The calculation of data ended at a horizontal distance of about x=3 cm, since there is no reason for the roller to advance further to the left.

Yellow and red regions on the 2D map indicate slip regions, for which

$$F_t / F_n > 1.0$$

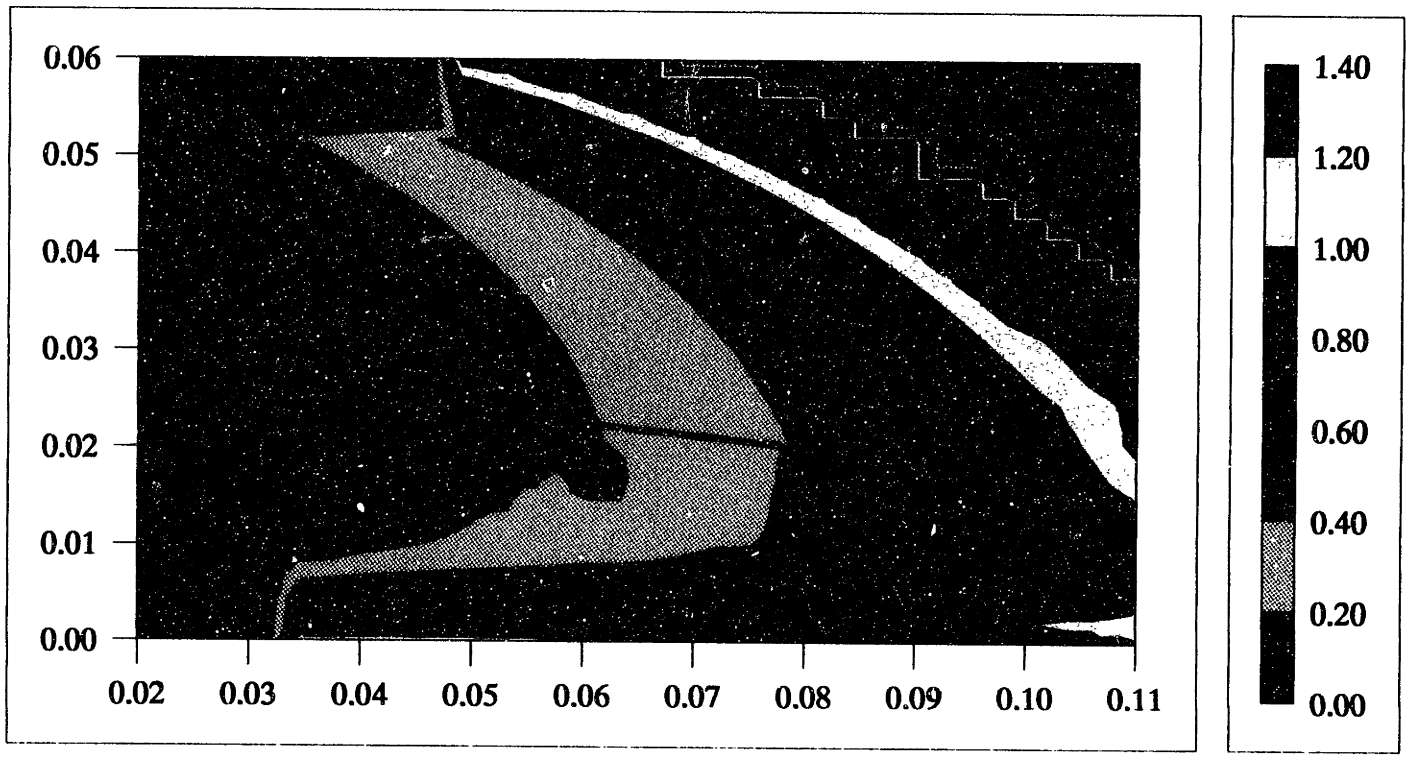


Figure 5.1: Variation of ratio of tangential to normal force at the contact point for different positions in the x,y operating space (in meters). [COLOR PLATE]

For a reasonable safety margin, the area where $F_t / F_n > 0.8$ should also be considered as a slip region. These regions dominate the upper-right part of the plot. The explanation lies in the geometry of the bill: if the ending angle is small, the spring force will act almost tangentially to the circumference of the roller and will tend to separate the contact (Figure 5.2-a). If the ending angle of the bill is large, the spring force on the bill will act normally to the roller (Figure 5.2-b).

The transition between Stage 2 and 3 is indicated on the contour plot at points where the moment at the contact, for Stage 2 analysis, became approximately equal to zero. Points above the transition were calculated using Stage 3 boundary conditions. The transition is not a straight line since, the more the bill is buckled in the x-direction the greater the length of the tail becomes, as the contact point changes. Hence, the transition height (measured in the y-direction) is greater at small values of x than for larger x.

The best trajectory for the roller to follow, avoiding all the possible slip areas, can be inferred from the contour plot, This is depicted in Figure 5.3. The bill should be buckled well away from the slip regions, until about x=5 cm, in Stage 1, before lifting from the stack. This would minimize the probability of slip. There is a trade-off, however, between reliability (buckling as far as possible from the slip regions, say to x=2 cm) and speed. An additional 2 cm of horizontal buckling may add up to 10-20% of operating

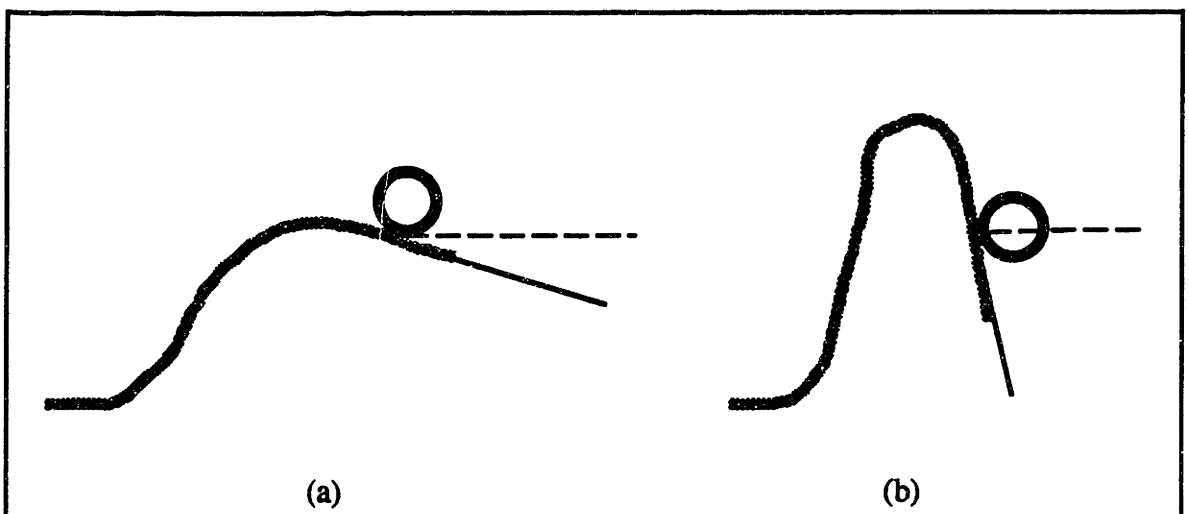


Figure 5.2: (a) Small angle makes slipping of bill highly possible (b) Big ending angle implies low possibility of slipping

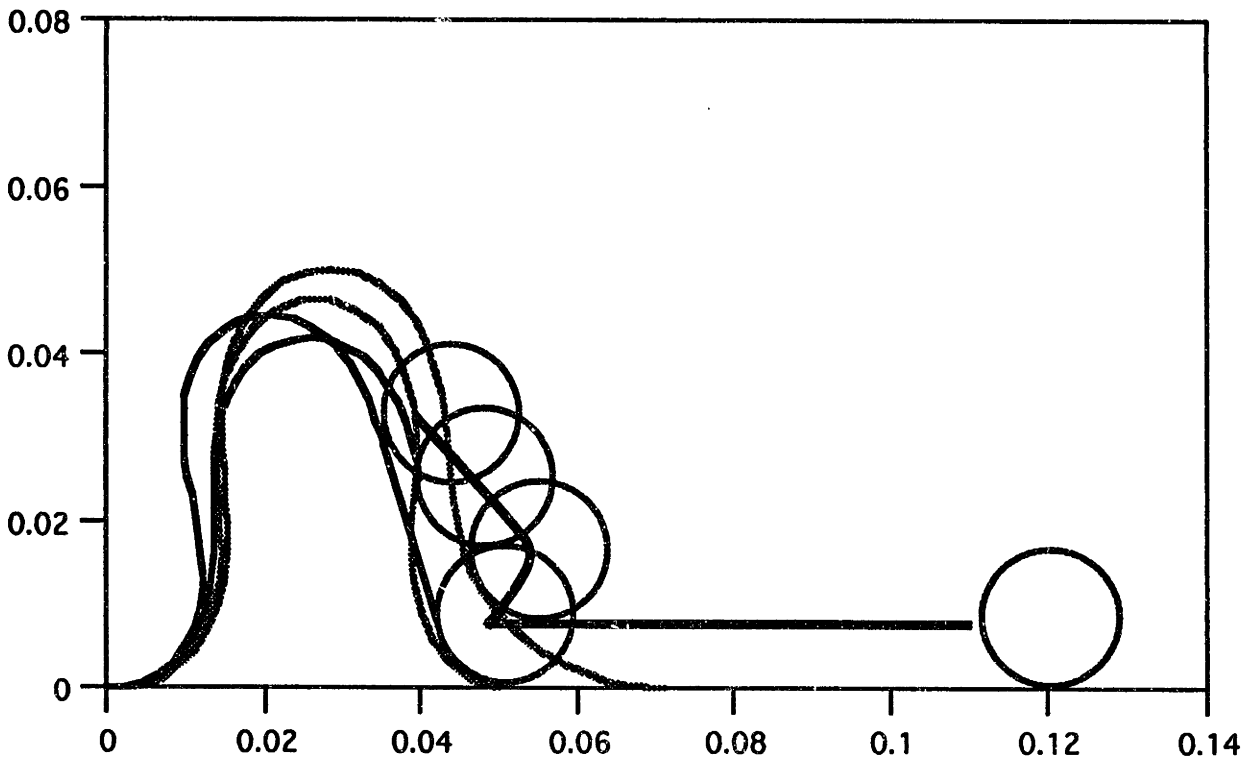


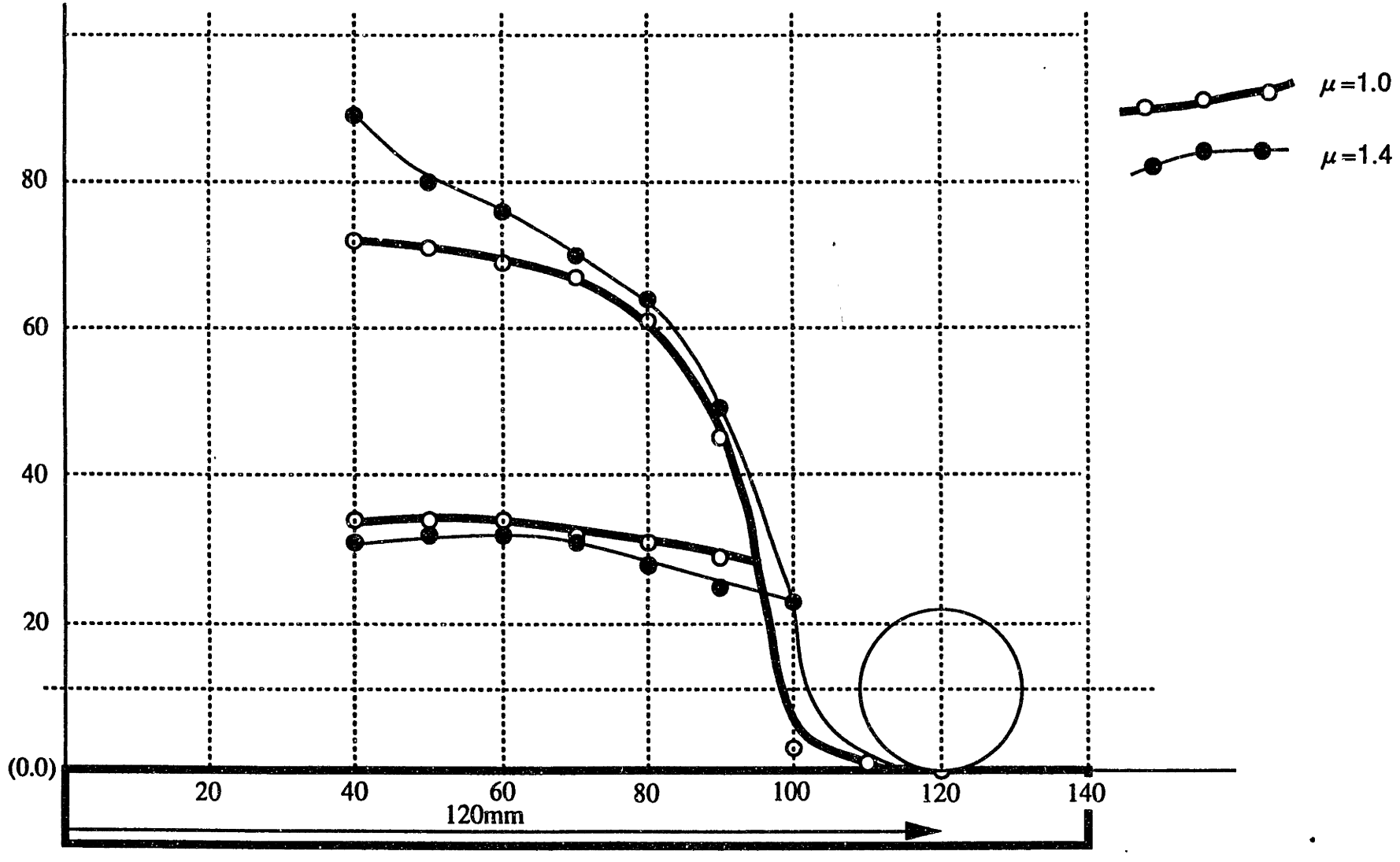
Figure 5.3 : Minimum slip trajectory of the roller

time per cycle, assuming the total trajectory of the roller is 10-20 cm long. Hence, if the frequency is at 20 notes/second this additional distance might decrease it to 16-18 notes/second.

Figure 5.4 shows the experimental results obtained from the prototype, for a new bill at initial contact length l=12 cm. The top pair of lines shows the positions for which slip occurs during Stage 3 buckling. The bold line is for a coefficient of friction between bill and roller of 1.0, while the thin line is for a coefficient of friction of 1.4. Compared to the numerical results of Figure 5.1, it can be seen that the shape of these lines closely match the shape of the contour plots in Stage 3. This leads to the conclusion that the numerical prediction has been successful in predicting the regions of slip.

The lower pair of lines in Figure 5.4 (at a height of about y=2.5 cm) indicates the transition from Stage 2 to Stage 3 for the same bill for different values of the coefficient of friction. Again, the shape of this line the same as the transition line found numerically (Figure 5.1). Hence, the numerical solution correctly predicted the boundary between the two stages, confirming the mathematical model used.

The buckling feeder has been tested repeatedly at OMRON with the same bill type, using the trajectory described above. The success rate has been 100% for new bills, dropping to 90% when older bills were used. This analysis can be also applied to defective bills, using the models described in Section 4.5, to establish a contour plot indicating regions of safe operation.


μ=1.0
μ=1.4
80
60
40
20
(0.0)
20
40
60
80
100
120
140
120mm

6. Dynamics

This chapter presents the inclusion of dynamic analysis in the modeling of the buckling feeder. When acceleration-dependent inertia forces and velocity-dependent damping forces become important, especially at high roller speeds, simple static analysis cannot any longer be justified. With the dynamic feeder this is often the case, since the design principle has been to create a high-speed machine. The methodology of including dynamics is presented and the results of the dynamic effects are examined.

The following variables are introduced for the inertial-force analysis:

vx: velocity in the x direction
vy: velocity in the y direction
vth: angular velocity

ax: acceleration in the x direction
ay: acceleration in the x direction
ath: angular acceleration

These variables are defined at each node of the discretized bill. In addition, all variables defined at each node (both the ones used in static analysis and the above variables) obtain a second dimension, in time. The time-dependency of variables will be now denoted as:

$x^{t-\Delta t}$: value of the variable x at time $t=t-\Delta t$

x^{t}: value of the variable x at time $t=t$

$x^{t+\Delta t}$: value of the variable x at time $t=t+\Delta t$

Dynamic analysis was treated essentially as a time-dependent static analysis. The static equilibrium of the elastica (including the effect of the inertia and damping forces) at discrete time intervals Δt was solved for.

Consider a particular element of the discretized bill (Figure 6-1). The element is in static equilibrium at position (A) and is moved in time Δt to position (B), another static equilibrium position, displaced by Δx in the horizontal direction, Δy in the vertical direction and by an angle $\Delta\theta$. The velocities are calculated as:

$$v_x^t = \frac{x^t - x^{t-\Delta t}}{\Delta t}$$

$$v_y^t = \frac{y^t - y^{t-\Delta t}}{\Delta t}$$

$$v_\theta^t = \frac{\theta^t - \theta^{t-\Delta t}}{\Delta t}$$

The accelerations on each element are then calculated by:

$$a_x^t = \frac{v_x^t - v_x^{t-\Delta t}}{\Delta t} = \frac{x^{t+\Delta t} - 2x^t + x^{t-\Delta t}}{\Delta t^2}$$

$$a_y^t = \frac{v_y^t - v_y^{t-\Delta t}}{\Delta t} = \frac{y^{t+\Delta t} - 2y^t + y^{t-\Delta t}}{\Delta t^2}$$

$$a_\theta^t = \frac{v_\theta^t - v_\theta^{t-\Delta t}}{\Delta t} = \frac{\theta^{t+\Delta t} - 2\theta^t + \theta^{t-\Delta t}}{\Delta t^2}$$

This way of calculating the accelerations and velocities is also called the central difference method. The acceleration at a position t in time is found by considering the positions at times t, t - Δt and t + Δt.

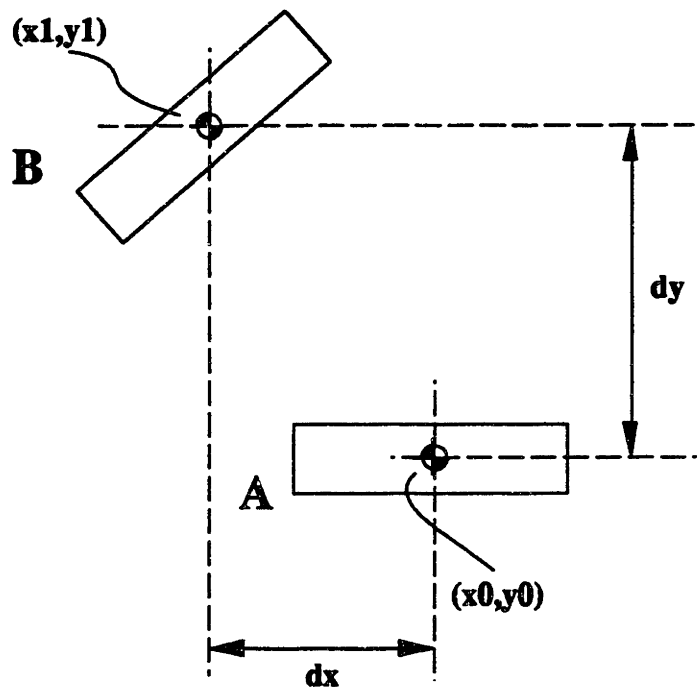


Figure 6-1: Calculation of velocities and accelerations from element's planar motion.

Once the accelerations and velocities were determined for time t, the inertia and damping forces were then calculated and added to the static forces. The new governing equations for the elastica, cf. (3-1) to (3-6), take the form:

$$\frac{\partial fx}{\partial s} = m_i\, a_x^t - b_i\, v_x^t \sin\theta$$

$$\frac{\partial fy}{\partial s} = -w + m_i\, a_y^t - b_i\, v_y^t \cos\theta$$

$$\frac{\partial m}{\partial s} = fx \sin\theta - fy \cos\theta + I_i\, a_\theta^t - B_i\, v_\theta^t$$

$$\frac{\partial \theta}{\partial s} = \frac{m}{EI}$$

where b_i is the damping coefficient due to aerodynamics effects, B_i the rotational damping coefficient due to internal damping, m_i the mass per element and I_i the rotational inertia per element. Note that the damping forces are perpendicular to the surface of the element so the coefficients must be adjusted for that by the appropriate trigonometric functions.

Inspection of these equations reveals that the inertial-force analysis is simply a 'superposition' on the static analysis. If the accelerations and velocities are zero, the resulting equations will be identical to those derived in Chapter 3. The equations were integrated using the fourth-order Runge-Kutta integrator combined with shooting.

The dynamic algorithm is illustrated in Figure 6-2. Initially, the first static solution is sought. The velocities and accelerations are determined from the difference of the last two static positions. The dynamic solution is then calculated. New static and dynamic solutions were calculated for every angular increment of the roller.

The inertial force analysis requires two additional inputs: the time step (Δt) and the damping coefficients (b_i and B_i). The time step has to be smaller than a critical time, Δt_{cr}. This critical time is calculated by considering the highest frequency of the elastica. Hence,

$$\Delta t << \Delta t_{cr} = T_n$$

where T_n is the smallest period of the elastica. The time-step also determines the angular velocity of the roller (Ω) since:

$$\Omega = \Delta\theta / \Delta t$$

where $\Delta\theta$ is the angular increment used in the static and dynamic calculation and Δt is the dynamic time step. Note that a very small time-step may lead to very large values in the calculated inertial and damping forces; the numerical solution will thus not converge under these circumstances. For a given roller speed, therefore, both the time-step and roller angular step-size have to be adjusted so as to meet the above conditions.

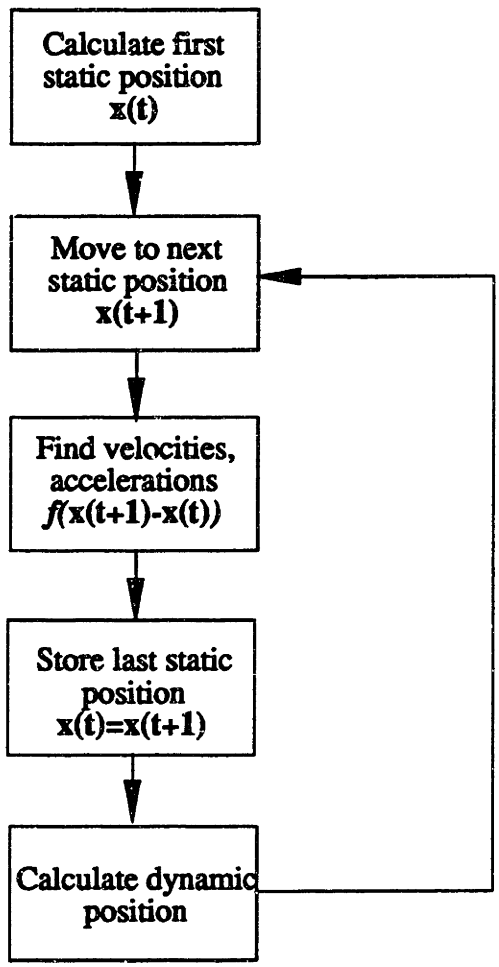


Figure 6.2: Schematic flowchart of the dynamic analysis. Bold characters represent vectors.

The table below shows different values of Δt for several roller speeds (for an angular step of 0.174533 rad (10 deg.).

Machine Speed (notes/sec)	Roller Velocity (rad/sec)	Time-step (sec)
25	157.1	0.001111
20	125.7	0.001389
10	62.8	0.002778
5	31.4	0.005556
1	6.3	0.027778

It was found that for roller speeds lower than 1 note per second dynamic analysis ceased to become important since the inertial and damping terms were small compared to the static terms. The corresponding dynamic solution did not differ from the static solution in terms of bill shape and value of the ratio.

The damping factor is caused by the aerodynamic effects on the bill for high roller speeds. The correct damping coefficient should be determined by an aerodynamic model, taking into account the airflow in the vicinity of the bill and roller. Such a model has not been included in the dynamic program but it can be easily added in the function for the damping coefficient.

7. Conclusions

The objectives of this work, as set out in the introduction, were to analyze and simulate the operation of a new type of paper-currency feeder, the buckling feeder, in view of increasing the speed of operation and eliminating the potential causes of malfunction: double-buckling, roll and slipping. The purpose of the present chapter is to draw a conclusion to the study and summarize the findings of the previous chapters.

To meet these objectives, a finite element model was initially developed. In extension to that original approach, a numerical model of the operation of the buckling feeder was solved using the initial value numerical techniques fourth-order Runge-Kutta and the shooting method. The analytical aspect of these models was to predict the possibility of slipping, which was quantified by the ratio of the tangential to the normal forces at the point of contact. The results of the analysis were compared to the experimental results measured from the prototype and were found to match closely. The operation of the buckling feeder was divided into three stages, in terms of their respective boundary conditions, to facilitate modeling. Stage 1 was the initial buckling stage, Stage 2 the transitional stage, and Stage 3 the lifting stage.

The problem of double-buckling, which occurred during Stage 1 when two or more bills were lifted simultaneously, was approached by analyzing the buckling mechanism at the initial stages of buckling, for bills of different stiffness and coefficient of friction. It was found that the problem of double-buckling greatly depended on the value of the normal force applied from the roller to the bill, which had to be between two limits. The lower limit was dictated from the minimum buckling force required to buckle the top bill, while the upper limit originated from the possibility of buckling the lower bills as well. It was concluded that the chances of double-buckling were particularly high

when a stack of bills of different conditions were considered and several alternative designs were proposed.

The problem associated with Stage 2 was mainly the slipping problem. Stage 2 was simulated by considering two separate elasticae (representing the bill), joined at the point of contact with the roller. The conditions at the joint were position, angle and moment continuity. The point of contact with the roller, which was unknown, was determined by testing a number of points on the roller's circumference. The true contact was established at the point where moment continuity was observed between the two elasticae.

During Stage 3, it was possible for the bill to slip or roll from the roller. Roll occurred when the contact point between the bill and the roller changed without the bill having slipped, but which could lead to slipping of the bill. The boundary conditions for the end of the elastica were formed by assuming that the end point of the elastica was known and, in addition, was the point of contact. This point then was changed by adjusting for roll and the ratio was calculated at the new point.

The solution of the feeder model was applied to all the points in the operating space of the device, by generating a mesh and calculating the ratio at all nodes of the mesh. This produced a contour lot, with the z-variable being the ratio, which predicted the areas in which slip was likely to occur. It was found that the contour plots were very close to the observations of the experimental prototype.

The last aspect of the numerical solution was to simulate the shapes of deformed bills under conditions of buckling in all three stages. The shapes that were produced were found to exactly represent real bills (with the same material characteristics) under similar conditions of deformation.

There are many possible extensions to the modeling of this problem that, if considered, would provide an interesting area for further study. These extensions range from simply improving the mathematical model of the device to producing a general purpose design package, with a graphical user-friendly interface, that will be allow the user to analyze and simulate similar problems involving flexible materials.

Extensions to the modeling include relaxing certain assumptions that were made, such as including the effect of the angular velocity of the roller, including airflow in the analysis and perhaps extending the analysis the 3-dimensions.

Developing a powerful design package for flexible materials, would involve reproducing much of this work , in particular the simulation aspect, as far as real-time simulation of the flexible materials is concerned. The requirements for such a project would be to reduce the solution time, particularly for Stage 2 simulation, and the ability to distinguish between the different boundary conditions at each point in the simulation.

References

[1] Bathe K-J., "Finite Element Procedures in Engineering Analysis", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982

[2] Briggs J., "Automated Handling of Flexible Materials", S.M. Thesis, Department of Mechanical Engineering, MIT, 1988

[3] Clapp T.G. and Peng H., "Buckling of Woven Fabrics, Part I: Effect of Fabric Weight", *Textile Research Journal*, 60, pp. 228-234 (1990)

[4] Clapp T.G. and Peng H., "Buckling of Woven Fabrics, Part II: Effect of Weight and Frictional Couple", *Textile Research Journal*, 60, pp. 285-292

[5] Crandall S.H., "Engineering Analysis: a Survey of Numerical Procedures", McGraw Hill, NewYork, N.Y., 1956

[6] Dotterer H.J., "Design and Analysis of a Bulk Note Feeding Device", S.B. Thesis, Department of Mechanical Engineering, MIT, 1991

[7] Kotovsky J., "Design of a De-Skewing System for Automatic Bill Handling Machines", S.B. Thesis, Department of Mechanical Engineering, MIT, 1990

[8] Levinsky R. B., "Design, Analysis and Control of a Bill Deskewing Device for Automated Teller Machines", S.M. Thesis, Department of Mechanical Engineering, MIT, 1992

[9] Press W. H., "Numerical Recipes: the Art of Scientific Computing", Cambridge University Press, 1986

[10] Timoshenko S. and Gere J.M., "Theory of Elastic Stability", McGraw Hill, NewYork, N.Y., 1961

[11] Wang C.Y., "Large Deformations of a Heavy Cantilever", *Quarterly of Applied Mathematics*, 39, pp. 261-273 (1981)

[12] Wang C.Y., "A Critical Review of the Heavy Elastica", *International Journal of Mechanical Science*, 28, 8, pp. 549-559 (1986)

Appendix I: Characteristics of Yen

There are three types of Yen notes currently in circulation in Japan, the denominations of which are ¥1,000, ¥5,000 and ¥10,000. The bills that are considered in this report are of the latter type. The geometrical and material characteristics of ¥10,000 notes are given below.

Geometry:

Length (L): 165 mm

Width (b): 76 mm

Thickness (h): 0.1 mm

Second moment of area for longitudinal bending (I): 6.33×10^{-12} m^4

(It must be noted that the total length of the bills considered in the anlaysis is 140 mm, since 25 mm are under the support.)

Weight:

Mass: 1.1 gram

Material:

Bending Stiffness: (see Chapter 2, but taken as $E=15 \times 10^6$ Nm for lifting analysis)

Coefficient of friction: (see Chapter 2)

Coefficient of friction of roller: 1.0

Appendix II: Finite Element Analysis of the Buckling Feeder

It was proposed initially by OMRON, to use a finite element approach to analyzing the operation of the buckling feeder, with the same goals of examining slip, roll and double-buckling and of simulating the bill under conditions of frictional buckling. This appendix describes the finite element model developed for the buckling feeder, the results obtained from it, and briefly comments on the application of the finite element method to the simulation of paper feeding.

The finite element package ADINA was used for the development of the finite element model. ADINA was chosen because of its ability of handling contact between surfaces effectively. It was required to calculate the ratio of the tangential to the normal forces between the roller and bill for different points along the roller's trajectory.

The finite element model (Figure I.1) consisted of the bill to be buckled, resting on a fixed surface. The bill was discretized into 25 beam elements, while the fixed surface was represented by a single beam element with all the degrees of freedom deleted. The roller circumference was modeled by a series of 64 beam elements constrained to move with the center of the circle. The contacts between the bill and the 'floor' element and between the bill and the roller elements was initially modeled using frictional contact. The material and geometry characteristics defined in Appendix I were used in this model.

This model was tried on ADINA without success for two main reasons: the geometry and the material properties of the bill. The irregular geometry of the bill implied a very high aspect ratio for the elements (ratio of length of the element to its thickness), while paper, having a low stiffness, was highly deformable. Numericaliy,

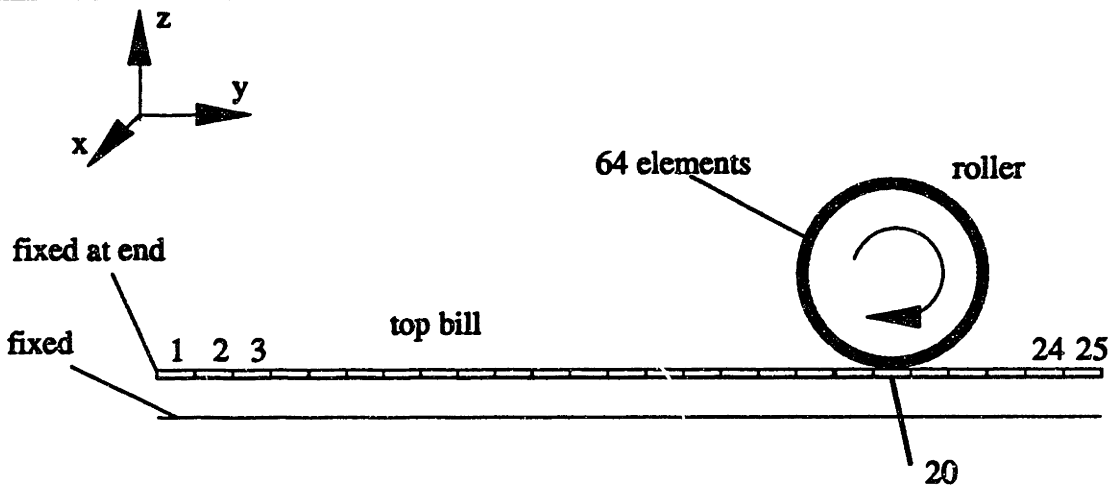


Figure I.1 : The finite element model of the roller and bill in contact

these two problems caused the stiffness matrix of the model to be ill-conditioned. The iteration method (full-Newton method) was not capable of manipulating the matrix, thus causing termination of the solution.

In order to overcome the difficulties of buckling the bill using frictional forces, a simplified model was considered, in which the bill was 'attached' to the roller, using a single beam element. The normal and tangential forces at the point of contact were calculated by measuring the forces applied on the connecting element (thus this element acted as a type of strain gauge). The results (the ratio of the tangential to normal forces against the angle of rotation of the roller) obtained from this model are presented in Figure I.2 for different initial contact lengths.

For all contact lengths, the value of the ratio at smaller angles of rotation is considerably higher that at larger angles, since initial buckling requires a larger tangential force to be applied. The graph shows the line where the ratio of the forces is equal to one, so points above the line indicate a large possibility of slip. It can be observed from the graph that the largest number of points in the 'non-slip' region belong to initial lengths of l=0.105 m and l=0.12 m.

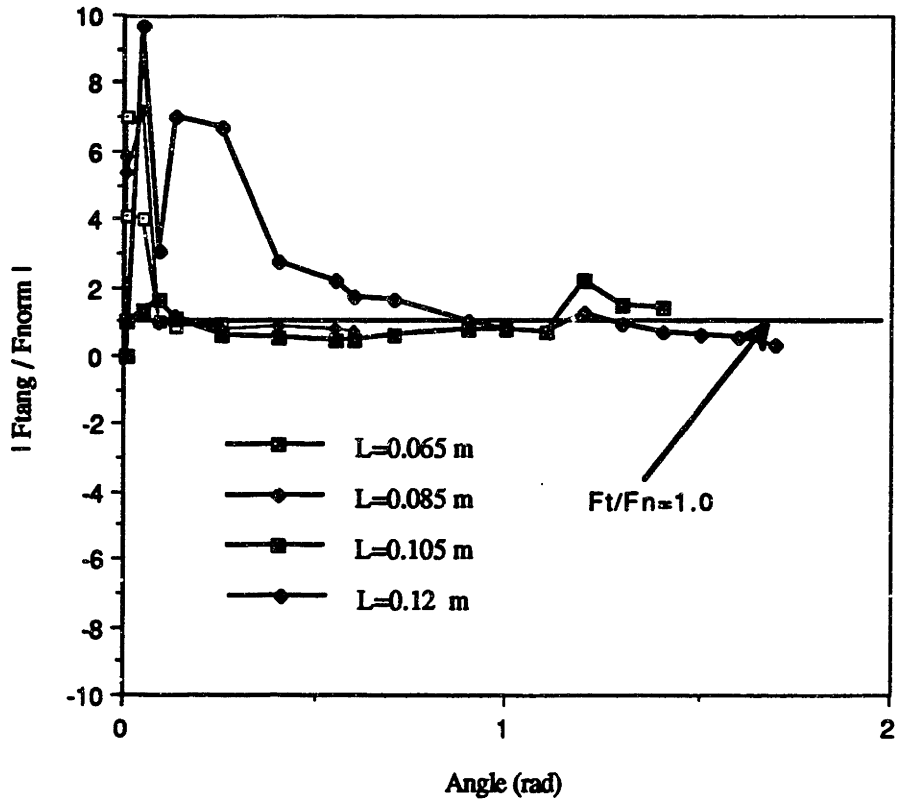


Figure I.2 : Variation of the ratio of the tangential to the normal force with angle of rotation of the roller for different initial contact lengths.

The difficulty of using frictional contact surfaces in finite element analysis for the modeling of the buckling operation, lead to the development of the numerical algorithm described in this work. If the results of the two approaches are compared, significant differences will arise. However, since the results produced using the numerical methods of Chapters 3 and 4 agree with the experimental results, it is reasonable to assume that the finite element model described above is not very realistic.

Appendix III: Computer Program Listings

This section contains the computer programs that have been written based on the models described in the earlier chapters. These programs were developed for two purposes: first, to allow editing of the machine's design through the model, without the need to buiild mechanical prototypes and second, to test the operation of each design and identify potential problems.

Three programs have been included, all written in ANSI C, in the UNIX/XWindows environment:

(1)	**roller.c**	Simulates the operation of the simple roller (quasi-static).
(2)	**segment.c**	Same as above, but for the segmented roller (quasi-static).
(3)	**dynamics.c**	Includes inertial-force analysis in the modeling of (2).

The codes can be compiled under UNIX by issuing the command: **make** <program>, where <program> is the name of the code required. To effectively compile each program the following files are required for each (this is also the order in which the listings appear in this appendix):

a. An instruction file for compilation ("Makefile")
b. A common header file ("common.h")
c. A "main" function for each program ("main.c" and "main_roller.c")
d. A list of auxiliary functions

The program "dynamics.c", which contains the inertial-force analysis, can be compiled separately. This is listed at the end of the appendix.

1. CODE "Roller":

Inputs required:
* Starting Position of Roller
(defined from x=0 - units: m)
* Feeding length
(Length of paper to be buckled - units: m)
* Roller Height
(Maximum vertical distance - units: m)

Program Output:
* Graphical simulation of roller movement and paper deformation
* Printout of ratio at each point of roller trajectory

2. CODE "Segment":

Inputs required:
* Starting Position of Roller
(defined from x=0 - units: m)
* Feeding angle of roller
(Total angle of rotation of roller - units: rad)

Program Output:
* Graphical simulation of roller movement and paper deformation
* Printout of ratio at each point of roller trajectory
* Graph of ratio vs. angle or roller

3. CODE "Dynamics":

Inputs required:
* Starting Position of Roller
(defined from x=0 - units: m)
* Feeding angle of roller
(Total angle of rotation of roller - units: rad)

Program Output:

* Graphical simulation of roller movement and paper deformation
* Printout of ratio at each point of roller trajectory
* Graph of ratio vs. angle or roller

List of files:

The following pages contain the files in the following order:

(1.) Makefile
(2.) common.h
(3.) main.c
(4.) main_roller.c
(5.) set_initial_conditions.c
(6.) set_boundary_conditions.c
(7.) shoot.c
(8.) calculate_ratio.c
(9.) below_floor.c
(10.) check_shape.c
(11.) Runge_Kutta.c
(12.) derivs.c
(13.) matrix.c
(14.) simulation.c
(15.) simulation_roller.c
(16.) load_font.c
(17.) draw_text.c
(18.) graphics_init.c
(19.) graphics_init_roller.c
(20.) dynamics.c

File: Makefile

```
# Makefile for compiling and linking Appendix codes
#
# Kind of gross hack :-)

TARGETS = segment roller

SEGMENT_SRCS = main.c Runge_Kutta.c below_floor.c calculate_ratio.c
check_shape.c \
derivs.c draw_text.c graphix_init.c load_font.c matrix.c
set_boundary_conditions.c set_initial_conditions.c \
shoot.c simulation.c
SEGMENT_OBJS = main.o Runge_Kutta.o below_floor.o calculate_ratio.o
check_shape.o \
derivs.o draw_text.o graphix_init.o load_font.o matrix.o
set_boundary_conditions.o set_initial_conditions.o \
shoot.o simulation.o

ROLLER_SRCS = main_roller.c Runge_Kutta.c below_floor.c
calculate_ratio.c check_shape.c \
derivs.c draw_text.c graphix_init_roller.c load_font.c matrix.c
set_boundary_conditions.c set_initial_conditions.c \
shoot.c simulation_roller.c
ROLLER_OBJS = main_roller.o Runge_Kutta.o below_floor.o
calculate_ratio.o check_shape.o \
derivs.o draw_text.o graphix_init_roller.o load_font.o matrix.o
set_boundary_conditions.o set_initial_conditions.o \
shoot.o simulation_roller.o

#PROG1_SRCS =
#PROG1_OBJS =

#PROG1_SRCS =
#PROG1_OBJS =

HDRS = common.h
LIBS = -L/usr/X11R5/lib -lX11 -lm


CFLAGS = -I/usr/X11R5/include
# CFLAGS = -O2 -I/usr/X11R5/include

CC = cc
LINT = lint
SABER = xsaber
DEPEND = makedepend
DEBUG = dbx
INSTALL = install
RM = /bin/rm -f
TAGGER = etags -t
LPR = enscript -2Gqr


.SUFFIXES: .o .c
```

```
.c.o:
	$(CC) $(CFLAGS) -c $<

all: $(TARGETS)

segment: $(SEGMENT_OBJS) $(HDRS)
	$(CC) $(CFLAGS) -o segment $(SEGMENT_OBJS) $(LIBS)

roller: $(ROLLER_OBJS)
	$(CC) $(CFLAGS) -o roller $(ROLLER_OBJS) $(LIBS)

#prog3: $(PROG3_OBJS)
#	$(CC) $(CFLAGS) -o prog3 $(PROG3_OBJS) $(LIBS)

clean:
	$(RM) *.o *~ *.bak .#*  $(TARGETS)
```

Header: common.h

```
/* common.h */

/* This is the header file common to all three thesis programs */

/*          Written by Aris Constantinides   05/03/93          */


#ifndef _common.h_
#define _common.h_

/* Include all the includes here... */

#include <stdio.h>
#include <math.h>
#include <X11/Xlib.h>                  /* Needed for the X-Windows */
#include <X11/Xutil.h>                 /* environment */
#include <X11/X.h>
#include <X11/keysym.h>

/* Include all the defines here... */

#define MAX_NODES 35                   /* Maximum number of nodes */
#define NUMBC 3                        /* Max. number of BC's */
#define N     2                        /* Max. number of BC's */
#define NUMBER_OF_NODES 35
#define SHAPES 20
#define PI 3.141592654

#define NUM_EQN 5                      /* No. of eqns integrated */
#define TINY 1.0e-20                   /* For inverse routine */


/* Properties of the yen notes */

#define L 0.14                         /* Length of bill */
#define B 0.076                        /* width of bill */
#define t 1.0e-4                       /* thickness of bill */
#define w 1.1e-3/0.14                  /* weight per unit length */
#define cof 0.15                       /* coefficient of friction */
                                       /* between lifted and top bill */
#define E 15000000                     /* elasticity of bending */
#define I 6.33e-12                     /* second moment of area */


/* Global Variables */

extern double x2_desired[NUMBC];       /* Desired BC at x2 */
extern double V[NUMBC];                /* Free BC at x1 */
extern double delv[NUMBC];             /* Change in V for Jacobian */
extern double fx_guess;                /* Guessed force_x at x1 */
extern double fy_guess;                /* Guessed force_y at x1 */
extern double m_guess;                 /* Guessed moment at x1 */
extern double old_fx_guess1;           /* Prev. guessed fx at x1 (lp) */
extern double old_fy_guess1;           /* Prev. guessed fy at x1 (lp)*/
```

```
extern double old_m_guess1;             /* Prev. guessed m at x1  (lp)*/
extern double old_fx_guess2;            /* Prev. guessed fx at x1 (rp) */
extern double old_fy_guess2;            /* Prev. guessed fy at x1 (rp)*/
extern double old_m_guess2;             /* Prev. guessed m at x1  (rp)*/

extern double ratio;                    /* Ratio Ft/Fn at contact */
extern double clip;                     /* Changes in V[] */
extern double tol1;                     /* Tolerance for left part */
extern double ratio_display,old_ratio;/* Ratio Ft/Fn at contact */
extern double dy_roll;                  /* y-momvt of roller */
extern double del_th;                   /* Angular increment for calcs */
                                        /* in radians */
extern double l;                        /* Length of yen (metres) */
extern double l1;
extern double l2;
extern double l21;
extern double l22;
extern double l1,l2,l21,l22,l_set;      /* Contact length with roller */
extern double l_rem;                    /* Length Remaining in tail */
extern double dl;                       /* Step size for bill length */
extern double xc0;                      /* Initial x-coord. of contact */
extern double lc0;                      /* Initial contact length */
extern double r_radius;                 /* Roller radius */
extern double theta,th_roll;            /* Current roller angle */
extern double th_seg,theta1;            /* some angle */
extern double x_cont;                   /* x trial contact point */
extern double y_cont;                   /* y trial contact point */
extern double left_mom;                 /* moment from left */
extern double right_mom;                /* moment from right */
extern double avg_mom;                  /* Average of left & right */
extern double wt;                       /* Total weight */
extern double ls;                       /* Length of segment 1/n) */
extern double th_fin,th_cont;           /* Segemented roller arc length */
extern double th_trial,th_old;          /* To find the equal moments */


/* Variables defined at each node of the elastica */

extern double fx[];                     /* force in x-direction */
extern double fy[];                     /* force in y-direction */
extern double m[];                      /* moment in bill */
extern double x_pos[];                  /* x-position of nodes */
extern double y_pos[];                  /* y-position of nodes */
extern double th[];                     /* angle of nodes with horiz. */
extern double dthds[];                  /* curvature of bill */
extern double add_mom[];                /* additional moment in bill */


/* Various flags and integers */

extern int n;                           /* Number of nodes */
extern int flag;                        /* Flags set of B.C.'s at end */
                                        /* flag=0: x, y, th */
                                        /* flag=1: fx, y, m */
extern stage3;                          /* Stage 3 flag */
extern int len_flag;                    /* Flag for change in bill length
*/
extern int eq_mom;                      /* Equal moments flag */
```

```
/* XWindows variables */

extern Display *display;                /* Pointer to display */
extern Window sim_window, data_window;/* Windows */
extern Window graph_window;
extern int screen;                      /* Screen ID */
extern GC gc;                           /* Graphic Content */
extern Font f;                          /* Font for text display */
extern XSizeHints hint;                 /* Window specification */
extern unsigned long foreground,background; /* Pixel values */
extern XColor DarkSlateGrey,Magenta;    /* Color Variables */
extern XColor White;
extern Colormap cmap, cmap1;            /* Default colormap */
extern XFontStruct *font_info;

extern int fail;                        /* Colors/Black-White */
extern double factor;                   /* Color factor */


#endif
```

Function: main.c

```
#include "common.h"


/* This is the main subroutine for code "segment.c" */
/* The modeling algorithms are mostly contained in this */
/* function. */

/* Global Variables declared in common.h */

double x2_desired[NUMBC];              /* Desired BC at x2 */
double V[NUMBC];                       /* Free BC at x1 */
double delv[NUMBC];                    /* Change in V for Jacobian */
double fx_guess;                       /* Guessed force_x at x1 */
double fy_guess;                       /* Guessed force_y at x1 */
double m_guess;                        /* Guessed moment at x1 */
double old_fx_guess1;                  /* Prev. guessed fx at x1 (lp) */
double old_fy_guess1;                  /* Prev. guessed fy at x1 (lp)*/
double old_m_guess1;                   /* Prev. guessed m at x1  (lp)*/
double old_fx_guess2=0.0;              /* Prev. guessed fx at x1 (rp) */
double old_fy_guess2=0.0;              /* Prev. guessed fy at x1 (rp)*/
double old_m_guess2=0.0;               /* Prev. guessed m at x1  (rp)*/
double ratio;                          /* Ratio of tangential to normal
forces */
double clip;                           /* Changes in V[] */
double tol1 = 0.0001;                  /* Tolerance for left part */
double ratio_display,old_ratio;        /* Ratio Ft/Fn at contact */
double dy_roll = 0.002;                /* y-momvt of roller */
double del_th = 0.2;                   /* Angular increment for calcs */
                                       /* in radians */
double l;                              /* Length of yen (metres) */
double l1;
double l2;
double l21;
double l22;
double l_set;
double l_rem;                          /* Length Remaining in tail */
double dl = 0.005;                     /* Step size for bill length */
double xc0;                            /* Initial x-coord. of contact */
double lc0;                            /* Initial contact length */
double r_radius = 0.01;                /* Roller radius */
double theta,th_roll;                  /* Current roller angle */
double th_seg,theta1;                  /* some angle */
double x_cont;                         /* x trial contact point */
double y_cont;                         /* y trial contact point */
double left_mom;                       /* moment from left */
double right_mom;                      /* moment from right */
double avg_mom;                        /* Average of left & right */
double wt;                             /* Total weight */
double ls;                             /* Length of segment l/n) */
double th_fin,th_cont;                 /* Segemented roller arc length */
double th_trial,th_old;                /* To find the equal moments */


/* Variables defined at each node of the elastica */
```

```
double fx[MAX_NODES];
double fy[MAX_NODES];
double m[MAX_NODES];
double x_pos[MAX_NODES];
double y_pos[MAX_NODES];
double th[MAX_NODES];
double dthds[MAX_NODES];
double add_mom[MAX_NODES];


/* Various flags and integers */

int n=35;                                   /* Number of nodes */
int flag;                                   /* Flags set of B.C.'s at end */
                                            /* flag=0: x, y, th */
                                            /* flag=1: fx, y, m */
int len_flag;                               /* Flag for change in bill length
*/
int eq_mom=1;                               /* Equal moments flag */


/* XWindows variables */

Display *display;                           /* Pointer to display */
Window sim_window, data_window;             /* Declare windows */
Window graph_window;
int screen;                                 /* Screen ID */
GC gc;                                      /* Graphic Content */
Font f;                                     /* Font for text display */
XSizeHints hint;                            /* Window specification */
unsigned long foreground,background;        /* Pixel values */
XColor DarkSlateGrey,Magenta;               /* Color Variables */
XColor White;
Colormap cmap, cmap1;                       /* Default colormap */
XFontStruct *font_info;

int fail;                                   /* Colors/Black-White */
double factor=257;                          /* Color factor */


FILE *fpo, *fopen();

main(argc,argv)

     int argc;
     char **argv;

{
  double testval;
  double Fx_left;                           /* Resultant force in x-dir */
  double Fy_left;                           /* Resultant force in y-dir */

  double outflag=999;                       /* Flag for output file */
  float dd3,dd4,dd5,dd6,dd7;                /* For keyboard input */
  int floor_node;                           /* Node where bill passes floor */
  int rept=1;                               /* scanf variable */
```

```
  int i,j;                              /* Various counters */
  int n_iter;                           /* Shoot counter */
  int in;                               /* Main iteration counter */
  int ang_count=0;                      /* for angular increments (graph)
*/


  double bc0,bc1,bc2;                   /* boundary conditions */
  double ic0,ic1,ic2;                   /* initial conditions  */

  double x_roll,y_roll,r_roll;          /* for graphics */
  double x_draw[2][MAX_NODES];
  double y_draw[2][MAX_NODES];


  /* Open file for the shape of the elastica (optional) */
  /* fpo = fopen("elshapenew","w"); */

  r_roll = 0.01;

  printf("\nContact length          ");
  scanf("%f",&dd7);
  lc0 = dd7;
  l1 = lc0;

  xc0 = lc0;
  x_roll = lc0;

  printf("\nFinal roller angle    (max. 3.216125 ");
  scanf("%f",&dd7);
  th_fin = dd7;
  y_roll = r_roll;


  graphix_init(argc,argv);              /* Initialize graphics */


  /* Correction terms for fx and fy */

  delv[0]=0.0000000005;                 /* Correction for fx */
  delv[1]=0.0000000005;                 /* Correction for fy */
  delv[2]=0.0000000005;                 /* Correction for m  */
  clip = 0.5;                           /* Clip value */


/* Choose guesses for initial conditions depending on the */
/* location of the roller */

  if (xc0 >=0.07) {
    old_fx_guess1 = -0.5;
    old_fy_guess1 = 0.06;
    old_m_guess1 = -0.007;
    old_fx_guess2 = 0.0;
    old_fy_guess2 = 0.0;
    old_m_guess2 = 0.0;

  }
```

```
  if (xc0 < 0.07) {
    old_fx_guess1 = -0.8;
    old_fy_guess1 = 0.09;
    old_m_guess1 = -0.008;
    old_fx_guess2 = 0.0;
    old_fy_guess2 = 0.0;
    old_m_guess2 = 0.00;
  }

  ratio_display = 0.0;
  flag=0;                              /* Set flag for left part first */
  th_roll = 0.174533;                  /* Angular step-size of roller */
  th_seg = 1.74533;                    /* Roller arc length 100 deg. */



/*-------------------------------------------------------------------*/
/* Rotation Stage of Roller */
/*-------------------------------------------------------------------*/

  theta = 0.0;                         /* Set angles to zero */
  theta1 = 0.0;



  while (theta1 <= th_seg) {           /* Until rotation stage completed
*/

    theta1 += th_roll;                 /* Update angle*/
    l1 += r_radius*th_roll;            /* Update lengths */
    l2 = L - l1;

    x_cont = xc0;
    y_cont = dy_roll;
    ls = l1/(1.0*(n-1.0));             /* Segment length */
    l = l1;                        /* Length of left part */



/* Initialize variables */
    for (j=0; j<MAX_NODES ; j++) {

      m[j]=0.0;
      fx[j]=0.0;
      fy[j]=0.0;
      x_pos[j]=0.0;
      y_pos[j]=0.0;
      th[j]=0.0;
      add_mom[j] = 0.0;
    }


/* This is additional moment data representing curved bills */
/* To calculate for straight bills, remove additional moment */
/*
          add_mom[13] = 0.007319;
          add_mom[14] = 0.006592;
```

```
            add_mom[15] = 0.006113;
            add_mom[16] = 0.006000;
            add_mom[17] = 0.006205;
            add_mom[18] = 0.006000;
            add_mom[19] = 0.006113;
            add_mom[20] = 0.006592;
            add_mom[21] = 0.007319;

*/


/* Set initial conditions */

    fx_guess = old_fx_guess1;
    fy_guess = old_fy_guess1;
    m_guess = old_m_guess1;

    set_initial_conditions(0.0,0.0,0.0);  /* x,y,th at s=0 */


/* Set boundary conditions */

    set_boundary_conditions(x_cont,0.0,0.0);
    /* x-coord of contact point */
    /* y-coord of contact point */
    /* ending angle */

    y_pos[n-1] = 1.0;                  /* Trick to pass into loop */


/* Shoot for desired conditions at the end */

    n_iter=0;                          /* check for tolerance */
                                       /* or number of iterations */
    while( (fabs(fabs(y_pos[n-1]) - x2_desired[1]) > tol1) && (n_iter <
30)) {


      n_iter++;

      fx[0]=V[0];
      fy[0]=V[1];
      m[0]=V[2];

      Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

      shoot(V,x2_desired,delv,clip,flag);

    }                      /* Closes while(fabs) */


    old_fx_guess1 = V[0];              /* for next guess, use */
    old_fy_guess1 = V[1];              /* solved initial conditions */
    old_m_guess1 = V[2];

    check_shape(x_draw,y_draw,flag);
```

```
    simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count);

  }

/*-------------------------------------------------------------------*/
/* Lifting Stage of Roller */
/*-------------------------------------------------------------------*/


/* Start with left part first */

  flag=0;                                  /* flag for left part first */

  theta = 0.0;                             /* set the angle to zero */
  l_rem = 0.0;
  l22 = l2;
  l21 = L - l22;
  th_old = theta;


  while (theta1 <= th_fin) {               /* until the end of lifting */

    theta1 += del_th;
    theta += del_th;

    x_cont = xc0 - r_radius*sin(theta);
    y_cont = r_radius*(1 + sin(theta-1.5708));


    th_trial = th_old;                     /* First start with the roller */
                                           /* ending angle */


    while(eq_mom) {                        /* loop to check that Mr = Ml */

      len_flag=1;
      l_rem = 0.0;

      while(len_flag) {                    /* Check if bill below floor */


      flag=0;

      ls = l21/(1.0*(n-1.0));              /* Segment length */
      l = l21;                             /* Length of left part */


      for (j=0; j<MAX_NODES ; j++) {

        m[j]=0.0;
        fx[j]=0.0;
        fy[j]=0.0;
        x_pos[j]=0.0;
        y_pos[j]=0.0;
        th[j]=0.0;
```

```
        add_mom[j] = 0.0;
      }

      fx_guess = old_fx_guess1;
      fy_guess = old_fy_guess1;
      m_guess = old_m_guess1;


      set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

      set_boundary_conditions(x_cont,y_cont,th_trial);
      /* x-coord of contact point */
      /* y-coord of contact point */
      /* ending moment */

      y_pos[n-1] = 1.0;          /* Trick to pass into loop */


      n_iter=0;
      while( (fabs(fabs(y_pos[n-1]) - x2_desired[1]) > tol1) && (n_iter
< 30)) {

        n_iter++;

        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        shoot(V,x2_desired,delv,clip,flag);


      }                          /* Closes while(fabs) */


      left_mom = m[n-1];

      old_fx_guess1 = V[0];
      old_fy_guess1 = V[1];
      old_m_guess1 = V[2];

      check_shape(x_draw,y_draw,flag);

      Fx_left = fx[n-1];
      Fy_left = fy[n-1];



/* Now solve for the right part */

      flag=1;


      ls = l22/(1.0*(n-1.0));          /* Segment length */
      l = l22;                         /* Length of left part */
```

```
      for (j=0; j<MAX_NODES ; j++) {

        m[j]=0.0;
        fx[j]=0.0;
        fy[j]=0.0;
        x_pos[j]=0.0;
        y_pos[j]=0.0;
        th[j]=0.0;
        add_mom[j] = 0.0;
      }

      fx_guess = 0.0;
      fy_guess = 0.0;
      m_guess = 0.0;

      set_initial_conditions(x_cont,-y_cont,th_trial); /* x,y,m at s=0
*/

      y_pos[n-1] = 1.0;          /* Trick to pass into loop */


/* Shoot loop for right part */

      n_iter=0;
      while( (fabs(fabs(y_pos[n-1]) - x2_desired[0]) > tol1) && (n_iter
< 30)) {

        n_iter++;

        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        set_boundary_conditions(0.0,cof*fabs(fy[n-1]),0.0);
        /* x-coord of contact point */
        /* y-coord of contact point */
        /* ending angle */

        shoot(V,x2_desired,delv,clip,1);

      }                          /* Closes while(fabs) */


      right_mom = m[0];
      old_fx_guess2 = V[0];
      old_fy_guess2 = V[1];
      old_m_guess2 = V[2];

      below_floor(floor_node);

      if (floor_node>33) len_flag=0;


      check_shape(x_draw,y_draw,flag);

      }                                /* Closes while(len_flag) */
```

```
/* End this run by drawing the shape and calculating the ratio */

      calculate_ratio(Fx_left,Fy_left);

      XClearWindow(display,data_window);
      draw_text(data_window,gc,font_info,hint.width,hint.height);
      XFlush(display);

      if(left_mom<0.0) th_trial += 10.0*(fabs(fabs(right_mom)-
fabs(left_mom)));
      if( (right_mom<0.0) && (fabs(right_mom) > fabs(left_mom)))
      th_trial -= 20*(fabs(fabs(right_mom)-fabs(left_mom)));

      122=12;
      if ( fabs(fabs(right_mom)-fabs(left_mom))<0.001 ) eq_mom = 0;

    }

    ratio_display = ratio;
    th_old = th_trial;
    eq_mom=1;

    check_shape(x_draw,y_draw,flag);

    ang_count++;
    simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count);
    old_ratio = ratio;
    printf("\n\n ********* MOVE TO NEW ANGLE *********\n\n");

  }                              /* Closes while(y_roll...) */

/* Clear window */
  sleep (5);
  XClearWindow(display,sim_window);
  XFlush(display);

/* Termination */

  XFreeGC (display,gc);
  XDestroyWindow (display,sim_window);
  XCloseDisplay (display);
  exit(0);


}                                /* Closes MAIN */
```

Function: main_roller.c

```
#include "common.h"

/* This is the main subroutine for code "segment.c" */
/* The modeling algorithms are mostly contained in this */
/* function. */

/* Global Variables declared in common.h */

double x2_desired[NUMBC];              /* Desired BC at x2 */
double V[NUMBC];                       /* Free BC at x1 */
double delv[NUMBC];                    /* Change in V for Jacobian */
double fx_guess;                       /* Guessed force_x at x1 */
double fy_guess;                       /* Guessed force_y at x1 */
double m_guess;                        /* Guessed moment at x1 */
double old_fx_guess1;                  /* Prev. guessed fx at x1 (lp) */
double old_fy_guess1;                  /* Prev. guessed fy at x1 (lp)*/
double old_m_guess1;                   /* Prev. guessed m at x1  (lp)*/
double old_fx_guess2=0.0;              /* Prev. guessed fx at x1 (rp) */
double old_fy_guess2=0.0;              /* Prev. guessed fy at x1 (rp)*/
double old_m_guess2=0.0;               /* Prev. guessed m at x1  (rp)*/
double clip;                           /* Changes in V[] */
double tol1 = 0.0001;                  /* Tolerance for left part */
double ratio,ratio_display;            /* Ratio Ft/Fn at contact */
double dy_roll = 0.002;                /* y-momvt of roller */
double del_th = 0.087266;              /* Angular increment for calcs */
                                       /* in radians */
double l;                              /* Length of yen (meters) */
double l1;                             /* Contact length with roller */
double l2;
double l21;
double l22;
double l_set;
double y_set;
double l_rem;                          /* Length remaining in tail */
double dl = 0.005;                     /* Step size for bill length */
double xc0;                            /* Initial x-coord. of contact */
double lc0;                            /* Initial contact length */
double r_radius = 0.01;                /* Roller radius */
double theta,th_roll;                  /* Current roller angle */
double x_cont;                         /* x trial contact point */
double y_cont;                         /* y trial contact point */
double left_mom;                       /* moment from left */
double right_mom;                      /* moment from right */
double avg_mom;                        /* Average of left & right */
double wt;                             /* Total weight */
double ls;                             /* Length of segment l/n) */

/* Variables defined at each node of the elastica */

double fx[MAX_NODES];
double fy[MAX_NODES];
double m[MAX_NODES];
double x_pos[MAX_NODES];
double y_pos[MAX_NODES];
```

```
double th[MAX_NODES];
double dthds[MAX_NODES];
double add_mom[MAX_NODES];


/* Various flags and integers */

int n=35;                                   /* Number of nodes */
int flag,eq_mom;                            /* Left part : flag=0 */
                                            /* Right part: flag=1 */
int len_flag;                               /* Flag for change in bill length
*/
int stage3=0;


/* XWindows variables */

Display *display;                           /* Pointer to display */
Window sim_window,data_window;              /* Window variables */
int screen;                                 /* Screen ID */
GC gc;                                      /* Graphic Content */
Font f;                                     /* Font for text display */
XSizeHints hint;                            /* Window specification */
unsigned long foreground,background;        /* Pixel values */
XColor DarkSlateGrey,Magenta;               /* Color Variables */
XColor White;
Colormap cmap,cmap1;                        /* Default colormap */
XFontStruct *font_info;

int fail;                                   /* Colors/Black-White */
double factor=257;                          /* Color factor */



FILE *fpo, *fopen();

main(argc,argv)

     int argc;
     char **argv;

{
  double testval;
  double Fx_left;                           /* Resultant force in x-dir */
  double Fy_left;                           /* Resultant force in y-dir */

  double outflag=999;                       /* Flag for output file */
  float dd3,dd4,dd5,dd6,dd7;                /* For keyboard input */
  int floor_node;                           /* Node where bill passes floor */
  int rept=1;                               /* scanf variable */
  int i,j;                                  /* Various counters */
  int n_iter;                               /* Shoot counter */
  int in;                                   /* Main iteration counter */

  double bc0,bc1,bc2;                       /* boundary conditions */
  double ic0,ic1,ic2;                       /* initial conditions  */
  double y_carry;
```

```
    double x_roll,y_roll,r_roll;        /* for graphics */
    double x_draw[2][MAX_NODES];
    double y_draw[2][MAX_NODES];


    /* Open file for the shape of the elastica (optional) */
    /*  fpo = fopen("elshapenew","w"); */

    r_roll = 0.01;

    printf("\nContact length          ");
    scanf("%f",&dd7);
    lc0 = dd7;
    l1 = lc0;

    xc0 = lc0;
    x_roll = lc0;

    printf("\nLength to be fed      ");
    scanf("%f",&dd7);
    l_set = dd7;

    printf("\nFinal y-position      ");
    scanf("%f",&dd7);
    y_set = dd7;
    y_roll = r_roll;


    graphix_init(argc,argv);


    /* Correction terms for fx and fy */

    delv[0]=0.0000000005;          /* Correction for fx */
    delv[1]=0.0000000005;          /* Correction for fy */
    delv[2]=0.0000000005;          /* Correction for m  */
    clip = 0.5;                    /* Clip value */


  /* Choose guesses for initial conditions depending on the */
  /* location of the roller */

    if (xc0>=0.07) {
      old_fx_guess1 = -0.5;
      old_fy_guess1 = 0.06;
      old_m_guess1 = -0.007;
      old_fx_guess2 = 0.0;
      old_fy_guess2 = 0.0;
      old_m_guess2 = 0.0;

    }

    if (xc0<0.07) {
      old_fx_guess1 = -0.8;
      old_fy_guess1 = 0.09;
      old_m_guess1 = -0.008;
      old_fx_guess2 = 0.0;
      old_fy_guess2 = 0.0;
```

```
    old_m_guess2 = 0.0;
  }

  ratio_display = 0.0;
  theta = 0.0;
  flag=0;                                   /* Set for left part first */
  th_roll = 0.174533;                       /* Angular step size of roller */


/*-------------------------------------------------------------------*/
/* Rotation Stage of Roller */
/*-------------------------------------------------------------------*/

  while (l1 <= l_set) {                     /* Until rotation is completed */

    l1 += r_radius*th_roll;
    l2 = L - l1;

    x_cont = xc0;
    y_cont = dy_roll;


    ls = l1/(1.0*(n-1.0));                  /* Segment length */
    l = l1;                                 /* Length of left part */


    for (j=0; j<MAX_NODES ; j++) {

      m[j]=0.0;
      fx[j]=0.0;
      fy[j]=0.0;
      x_pos[j]=0.0;
      y_pos[j]=0.0;
      th[j]=0.0;
      add_mom[j] = 0.0;
    }


/* This is additional moment data representing curved bills */
/* To calculate for straight bills, remove additional moment */

/*
          add_mom[13] = 0.007319;
          add_mom[14] = 0.006592;
          add_mom[15] = 0.006113;
          add_mom[16] = 0.006000;
          add_mom[17] = 0.006205;
          add_mom[18] = 0.006000;
          add_mom[19] = 0.006113;
          add_mom[20] = 0.006592;
          add_mom[21] = 0.007319;

*/
```

```
/* Set initial and boundary conditions for roller */

    fx_guess = old_fx_guess1;
    fy_guess = old_fy_guess1;
    m_guess = old_m_guess1;


    set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

    set_boundary_conditions(x_cont,0.0,0.0);
    /* x-coord of contact point */
    /* y-coord of contact point */
    /* ending angle */

    y_pos[n-1] = 1.0;           /* Trick to pass into loop */


/* Shoot for desired conditions at the end */


    n_iter=0;
    while( (fabs(fabs(y_pos[n-1]) - x2_desired[1]) > tol1) && (n_iter <
30)) {


      n_iter++;

      fx[0]=V[0];
      fy[0]=V[1];
      m[0]=V[2];

      Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

      shoot(V,x2_desired,delv,clip,flag);


    }

    old_fx_guess1 = V[0];
    old_fy_guess1 = V[1];
    old_m_guess1 = V[2];

    check_shape(x_draw,y_draw,flag);

    simulation(x_roll,y_roll,r_roll,x_draw,y_draw);

  }

/*-----------------------------------------------------------------*/
/* Stage 2 Analysis */
/*-----------------------------------------------------------------*/


/* Start from left part first */


  while (y_roll <= y_set) {          /* Until limit of y-pos. */
```

```
      eq_mom=1;
      y_roll += dy_roll;
      theta = theta/1.5;


      while(eq_mom) {                    /* Loop to check that Mr = Ml */


        theta += del_th;
        l_rem = 0.0;
        l22 = l2 + r_radius*theta;
        l21 = L - l22;
        x_cont = xc0 - r_radius*sin(theta);
        y_cont = (y_roll - r_roll) + (1 - cos(theta))*r_radius;
        len_flag=1;


        while(len_flag) {

        flag=0;

        ls = l21/(1.0*(n-1.0));          /* Segment length */
        l = l21;                         /* Length of left part */


        for (j=0; j<MAX_NODES ; j++) {
          m[j]=0.0;
          fx[j]=0.0;
          fy[j]=0.0;
          x_pos[j]=0.0;
          y_pos[j]=0.0;
          th[j]=0.0;
          add_mom[j] = 0.0;
        }

        fx_guess = old_fx_guess1;
        fy_guess = old_fy_guess1;
        m_guess = old_m_guess1;


        set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

        set_boundary_conditions(x_cont,y_cont,theta);
        /* x-coord of contact point */
        /* y-coord of contact point */
        /* ending angle */

        y_pos[n-1] = 1.0;          /* Trick to pass into loop */


/* Shoot for known boundary conditions */

        n_iter=0;
        while( (fabs(fabs(y_pos[n-1]) - x2_desired[1]) > tol1) && (n_iter
< 30)) {

          n_iter++;
```

```
        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        shoot(V,x2_desired,delv,clip,flag);

      }                          /* Closes while(fabs) */


      left_mom = m[n-1];
      old_fx_guess1 = V[0];
      old_fy_guess1 = V[1];
      old_m_guess1 = V[2];

      check_shape(x_draw,y_draw,flag);

      Fx_left = fx[n-1];
      Fy_left = fy[n-1];


/* Now solve for the right part */

      flag=1;

      ls = l22/(1.0*(n-1.0));          /* Segment length */
      l = l22;                         /* Length of left part */


      for (j=0; j<MAX_NODES ; j++) {

        m[j]=0.0;
        fx[j]=0.0;
        fy[j]=0.0;
        x_pos[j]=0.0;
        y_pos[j]=0.0;
        th[j]=0.0;
        add_mom[j] = 0.0;
      }

      fx_guess = 0.0;
      fy_guess = 0.0;
      m_guess = 0.0;

      set_initial_conditions(x_cont,-y_cont,theta); /* x,y,th at s=0 */

      y_pos[n-1] = 1.0;          /* Trick to pass into loop */


/* Shoot for boundary conditions at the right part */

      n_iter=0;
      while( (fabs(fabs(y_pos[n-1]) - x2_desired[0]) > tol1) && (n_iter
< 30)) {

        n_iter++;
```

```
            fx[0]=V[0];
            fy[0]=V[1];
            m[0]=V[2];

            Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

            set_boundary_conditions(0.0,cof*fabs(fy[n-1]),0.0);
            /* x-coord of contact point */
            /* y-coord of contact point */
            /* ending angle */

            shoot(V,x2_desired,delv,clip,1);

          }


          right_mom = m[0];

          old_fx_guess2 = V[0];
          old_fy_guess2 = V[1];
          old_m_guess2 = V[2];

          below_floor(floor_node);

          if (floor_node>33) len_flag=0;

          check_shape(x_draw,y_draw,flag);

          }

/* End this run by drawing the shape and calculating the ratio */

          calculate_ratio(Fx_left,Fy_left);

          XClearWindow(display,data_window);
          draw_text(data_window,gc,font_info,hint.width,hint.height);
          XFlush(display);


          if ( (right_mom <=0.0) && ( fabs(right_mom) > fabs(left_mom))) {
          simulation(x_roll,y_roll,r_roll,x_draw,y_draw);
          avg_mom = (left_mom + right_mom)/2.0;
          if (avg_mom >= -0.001000) {
            y_carry = y_roll;
            y_roll = 999.0;
          }
          eq_mom=0;

          printf("\n Changing to different y - position ");
          }

       }
       ratio_display = ratio;

     }

/*-----------------------------------------------------------------*/
```

```
/* Stage 3 Analysis */
/*-----------------------------------------------------------------*/


  flag = 0;                        /* flags that Stage 3 starts */
  stage3=1;
  y_roll = y_carry;


  while (y_roll <= y_set) {

    y_roll += dy_roll;
    x_cont = xc0 - r_radius*sin(theta);
    y_cont = (y_roll - r_roll) + (1 - cos(theta))*r_radius;
    ls = l21/(1.0*(n-1.0));            /* Segment length */
    l = l21;                           /* Length of left part */


    for (j=0; j<MAX_NODES ; j++) {

      m[j]=0.0;
      fx[j]=0.0;
      fy[j]=0.0;
      x_pos[j]=0.0;
      y_pos[j]=0.0;
      th[j]=0.0;
      add_mom[j] = 0.0;
    }

    fx_guess = old_fx_guess1;
    fy_guess = old_fy_guess1;
    m_guess = old_m_guess1;

    set_initial_conditions(0.0,0.0,0.0);

    set_boundary_conditions(x_cont,y_cont,theta);

    y_pos[n-1] = 1.0;                  /* Trick to pass into loop */


/* Shoot for boundary conditions at right end */

    n_iter=0;
    while( (fabs(fabs(y_pos[n-1]) - x2_desired[1]) > tol1) && (n_iter <
30)) {

      n_iter++;

      fx[0]=V[0];
      fy[0]=V[1];
      m[0]=V[2];

      Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

      shoot(V,x2_desired,delv,clip,flag);


    }                        /* Closes while(fabs) */
```

```
      old_fx_guess1 = V[0];
      old_fy_guess1 = V[1];
      old_m_guess1 = V[2];


      check_shape(x_draw,y_draw,0);

      ratio = (fabs(fx[n-1]*cos(th[n-1]) + fy[n-1]*sin(th[n-1]))) /
        (fabs(fx[n-1]*sin(th[n-1]) +fy[n-1]*cos(th[n-1])));

      ratio_display = ratio;

      simulation(x_roll,y_roll,r_roll,x_draw,y_draw);

    }


    sleep (5);
    XClearWindow(display,sim_window);
    XClearWindow(display,data_window);
    XFlush(display);

/* Termination */
    XFreeGC (display,gc);
    XDestroyWindow (display,sim_window);
    XDestroyWindow (display,data_window);
    XCloseDisplay (display);
    exit(0);


}
```

Function: set_initial_conditions.c

```
#include "common.h"

/* Enters values for known and unknown initial conditions */

set_initial_conditions(ic0,ic1,ic2)

     double ic0,ic1,ic2;

{

/* Known boundary conditions at s=0 for left part*/

  x_pos[0]=ic0;
  y_pos[0]=ic1;
  th[0]=ic2;

/*  Unknown Boundary Conditions at s=0 (guessed) */

  V[0] = fx_guess;
  V[1] = fy_guess;
  V[2]  = m_guess;

}
```

Function: set_boundary_conditions.c

```
#include "common.h"

/* Enters values for the known boundary conditions */

set_boundary_conditions(bc0,bc1,bc2)

     double bc0,bc1,bc2;

{

/* Generic case */

  x2_desired[0]=bc0;          /* first  bound. condition */
  x2_desired[1]=bc1;          /* second bound. condition */
  x2_desired[2]=bc2 ;         /* third  bound. condition */


}
```

Function: shoot.c

```
#include "common.h"

/* This is the actual shooting algorithm, described in Chapter 3 */

shoot(V,x2_desired,delv,clip,flag)

     double V[NUMBC];                  /* Free BC at x1 */
     double x2_desired[NUMBC];         /* Desired BC at x2 */
     double delv[NUMBC];               /* Change in V for Jacobian */
     double clip;                      /* Changes in V[] */
     int flag;                         /* Distinguish between left and */
                                       /* right part */

{
  double F[NUMBC];                     /* Discrepancy vector at x2 */
  double s[NUMBC];                     /* Discr. vector */
  double dv[NUMBC];                    /* Temp. discrepancy vector */
  double dxc,dyc;                      /* Discrepancy terms for x,y_pos*/
  double dxc1,dyc1;                    /* Discr. terms for x,y_pos */
  double dfdv[NUMBC][NUMBC];           /* Jacobian matrix */
  double a[NUMBC][NUMBC];              /* Jacobian matrix backup */
  double inv_dfdv[NUMBC][NUMBC];       /* Inverse Jacobian matrix */
  double det_dfdv;                     /* Determinant of Jacobian matrix
*/
  double sav;                          /* Storage variable */
  double mom_cal;


  int i,j,k,q;                         /* Various counters */

/* ------------------------------------------------------------ */
/* Trial Integration of Equations */
/* ------------------------------------------------------------ */

  if(!flag){
    F[0] = x_pos[n-1] - x2_desired[0];
    F[1] = fabs(y_pos[n-1]) - x2_desired[1];
    F[2] = th[n-1] - x2_desired[2];
  }
  if(flag){
    F[0] = fabs(y_pos[n-1]) - x2_desired[0];
    F[1] = fx[n-1] - x2_desired[1];
    F[2] = m[n-1] - x2_desired[2];
  }


/* ------------------------------------------------------------ */
/* Vary fx[0] */
/* ------------------------------------------------------------ */

  sav = fx[0];
  fx[0] += delv[0];
```

```
  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

  if(!flag){
    dv[0] = x_pos[n-1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1]) - x2_desired[1];
    dv[2] = th[n-1] - x2_desired[2];
  }
  if(flag){
    dv[0] = fabs(y_pos[n-1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
  }


  dfdv[0][0] = (dv[0] - F[0])/delv[0];
  dfdv[1][0] = (dv[1] - F[1])/delv[0];
  dfdv[2][0] = (dv[2] - F[2])/delv[0];

  fx[0] = sav;                          /* Return fx to original */
                                        /* value before increment */

/* ------------------------------------------------------------ */
/* Vary fy[0] */
/* ------------------------------------------------------------ */

  sav = fy[0];
  fy[0] += delv[1];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);


  if(!flag){
    dv[0] = x_pos[n-1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1]) - x2_desired[1];
    dv[2] = th[n-1] - x2_desired[2];
  }
  if(flag){
    dv[0] = fabs(y_pos[n-1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
  }

  dfdv[0][1] = (dv[0] - F[0])/delv[1];
  dfdv[1][1] = (dv[1] - F[1])/delv[1];
  dfdv[2][1] = (dv[2] - F[2])/delv[1];

  fy[0] = sav;                          /* Return fy to original */
```

```
                                        /* value before increment */

/* ------------------------------------------------------------ */
/* Vary m[0] */
/* ------------------------------------------------------------ */

  sav = m[0];
  m[0] += delv[2];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

  if(!flag){
    dv[0] = x_pos[n-1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1]) - x2_desired[1];
    dv[2] = th[n-1] - x2_desired[2];
  }
  if(flag){
    dv[0] = fabs(y_pos[n-1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
  }

  dfdv[0][2] = (dv[0] - F[0])/delv[2];
  dfdv[1][2] = (dv[1] - F[1])/delv[2];
  dfdv[2][2] = (dv[2] - F[2])/delv[2];

  m[0] = sav;                           /* Return m to original */
                                        /* value before increment */

  for (i=0 ; i<3 ; i++) {
    for(j=0 ; j<3 ; j++)
      a[i][j]=dfdv[i][j];
  }



  calculate_inverse(a,inv_dfdv);


  s[0]=F[0];
  s[1]=F[1];
  s[2]=F[2];


  multiply(inv_dfdv,s,dv);

  dv[0] = -dv[0];
  dv[1] = -dv[1];
  dv[2] = -dv[2];
```

```
  if ( fabs(dv[0]) > (clip*fabs(dv[0])) || fabs(dv[1]) >
(clip*fabs(dv[1]))
      || fabs(dv[2]) > (clip*fabs(dv[2])) ) {

    dv[0] *= clip;
    dv[1] *= clip;
    dv[2] *= clip;
  }


  fx[0] += dv[0];
  fy[0] += dv[1];
  m[0]  += dv[2];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

  mom_cal=0.0;
  for (j=1 ; j<n ; j++) {
    mom_cal += (w*ls) * ((x_pos[j-1]+x_pos[j])/2.0);
  }



}                                        /* Closes shoot */
```

Function: calculate_ratio.c

```
#include "common.h"

/* Calculates the ratio at the point of contact with the roller */

calculate_ratio(Fx_left,Fy_left)

     double Fx_left,Fy_left;


{
  double Fx_res,Fy_res;
  double F_tangent,F_normal;
  int r_node;

  Fx_res = fx[0] + Fx_left;              /* Resultant force in x-dir. */
  Fy_res = fy[0] + Fy_left;              /* Resultant force in y-dir. */

  F_tangent = fabs(Fx_res*cos(th[0]) + Fy_res*sin(th[0]));
  F_normal  = fabs(Fx_res*sin(th[0]) + Fx_res*cos(th[0]));

  ratio = F_tangent / F_normal ;

}
```

Function: below_floor.c

```
#include "common.h"

/* Checks to see if bill has gone below the floor */


below_floor(floor_node)

     int floor_node;
{
  int i;

  len_flag = 0;

  for (i=0 ; i<34 ; i++) {
    if(y_pos[i]>0.0002) len_flag=1;
    floor_node = i;
  }

  if (len_flag) {
    l22 -= dl;
    l_rem += dl;
    printf("\n Bill has gone below floor: new length %f",l22 );
    y_pos[n-1] = 1.0;
  }

}
```

Function: check_shape.c

```
#include "common.h"

/* Checks calculated shape of bill to see if solution is physically */
/* acceptable. Also prepares drawing subroutine.*/


check_shape(x_draw,y_draw,flag)

     double x_draw[2][MAX_NODES];
     double y_draw[2][MAX_NODES];
     int flag;

{

  int cnt1,i;

  cnt1=0;

  if(m[0]<0.0){

    for (i=1 ; i<34 ; i++) {
      if((m[i]*m[i-1])<0.0) cnt1++;
    }
/* if (cnt1>1)printf("\nWrong shape !!; %d changes in
curvature\n",cnt1);  */
  }


  for (i=0; i<MAX_NODES ; i++) {
    x_draw[flag][i]=x_pos[i];
    y_draw[flag][i]=-y_pos[i];
  }
}
```

Function: Runge_Kutta.c

```
#include "common.h"

/* ------------------------------------------------------------ */
/*                      Runge-Kutta integrator                  */
/* ------------------------------------------------------------ */

Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n)

     double fx[],fy[],m[],th[],x_pos[],y_pos[];
     double ls;
     int n;


{
  double ybeg[NUM_EQN];          /* Begining of t-step */
  double yend[NUM_EQN];          /* End of t-step */
  int i,k;
  double l1 = 0.0;               /* Distance from origin */
  double Total_weight;           /* Total weight (correction) */


  k=0;


  ybeg[0]=fx[0];
  ybeg[1]=fy[0];
  ybeg[2]=m[0];
  ybeg[3]=th[0];
  ybeg[4]=0.0;

  while(1){

    l1 += ls;

    if (l1>l+ls/10.0) break;

    integrate_one_step(ybeg,l1,ls,yend,k);

    for (i=0; i<NUM_EQN; i++) {
      ybeg[i]=yend[i];
    }

    fx[k+1] = yend[0];
    fy[k+1] = yend[1];
    m[k+1]  = yend[2];
    th[k+1] = yend[3];
    Total_weight = yend[4];

    x_pos[k+1] = x_pos[k] + ls*cos(th[k]);
    y_pos[k+1] = y_pos[k] + ls*sin(th[k]);
```

```
      k++;
    }

  }


  /* Integrate one time step using Runge-Kutta method... */


  integrate_one_step(ybeg,x,h,yend,k)

       double ybeg[];
       double x;
       double h;
       double yend[];
       int k;

  {
    double fcn1[NUM_EQN];
    double fcn2[NUM_EQN];
    double fcn3[NUM_EQN];
    double fcn4[NUM_EQN];
    double xtmp;
    double ytmp[NUM_EQN];

    int i;

    xtmp = x-h;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i];
      derivs(xtmp,ytmp,fcn1,k);
    }
    xtmp = x-h/2.0;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn1[i]/2.0;
      derivs(xtmp,ytmp,fcn2,k);
    }
    xtmp = x-h/2.0;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn2[i]/2.0;
      derivs(xtmp,ytmp,fcn3,k);
    }
    xtmp = x;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn3[i];
      derivs(xtmp,ytmp,fcn4,k);
    }

    for (i=0 ; i<NUM_EQN ; i++) {
      yend[i] = ybeg[i] + h*(fcn1[i] + 2.0*fcn2[i] + 2.0*fcn3[i] +
  fcn4[i])/6.0;
    }

  }
```

Function: derivs.c

```
#include "common.h"

/* Governing equations of the elastica */

derivs(x,y,fcn,k)

     double x;
     double y[];
     double fcn[];
     int k;

/* Four coupled first-order equations from initial ODE */
/* eg. fcn[0] = dy[0]/ds */

{
  fcn[0]=0.0;
                          /* d(fx)/ds=0.0 */

  fcn[1]= -1.0*w;         /* d(fy)/ds=-w  */

  fcn[2]=y[0]*sin(y[3]) - y[1]*cos(y[3]);
                          /* d(m)ds=... */

  fcn[3]=(y[2]+add_mom[k])/(E*I);
                          /* d(th)/ds=m/EI */
  fcn[4]=w*x;
}
```

Function: matrix.c

```
#include "common.h"

/* Contains matrix functions used in programs:    */
/* 1. Inverse of a matrix                         */
/* 2. LU decomposition and LU back-substitution   */
/* 3. Multiplication of vector and matrix         */


/*----------------------------------------------------------------------*/
/* Calculate inverse of matrix */
/*----------------------------------------------------------------------*/

calculate_inverse(a,r)

     double a[N+1][N+1];
     double r[N+1][N+1];
{
  double l[N+1][N+1],u[N+1][N+1];
  int i,j;
  int indx[N+1];
  double b[N+1];
  double d;
  double y[N+1][N+1];
  double col[N+1];
  double x[N+1];

  ludcmp(a,N,indx,&d);


/* Decode matrix into upper and lower triangular matrices */

  for (i=0 ; i<=N ; i++) {
    for (j=0 ; j<=N ; j++) {
      u[i][j]=a[i][j];
      l[i][j]=a[i][j];
    }
  }

  for (i=0 ; i<=N ; i++) {
    l[i][i]=1.0;
    for (j=i+1 ; j<=N ; j++) l[i][j]=0.0;
  }

  for (j=0 ; j<=N-1 ; j++) {
    for (i=j+1 ; i<=N ; i++) u[i][j]=0.0;
  }



  for(j=0 ; j<=N ; j++) {

    for (i=0 ; i<=N ; i++) col[i]=0.0;
    col[j]=1.0;
```

```
    lubksb(l,u,N,indx,col,x);

    for(i=0 ; i<=N ; i++) r[j][i]=x[i];
  }


}


/*-------------------------------------------------------------------------*/
/* LU Decompose matrix */
/*-------------------------------------------------------------------------*/


ludcmp(a,n,indx,d)

     int n,indx[N+1];
     double a[N+1][N+1],*d;
{
  int i,imax,j,k;
  float big,dum,sum,temp;
  double vv[N+1];


  *d=1.0;
  for (i=0;i<=n;i++) {
    big=0.0;
    for (j=0;j<=n;j++)
      if ((temp=fabs(a[i][j])) > big) big=temp;
    if (big == 0.0) printf("SINGULAR\n");
    vv[i]=1.0/big;
  }


  for (j=0;j<=n;j++) {
    for (i=0;i<j;i++) {
      sum=a[i][j];
      for (k=0;k<i;k++) sum -= a[i][k]*a[k][j];
      a[i][j]=sum;
    }
    big=0.0;
    for (i=j;i<=n;i++) {
      sum=a[i][j];
      for (k=0;k<j;k++)
      sum -= a[i][k]*a[k][j];
      a[i][j]=sum;
      if ( (dum=vv[i]*fabs(sum)) >= big) {
      big=dum;
      imax=i;
      }
    }
    if (j != imax) {
      for (k=i;k<=n;k++) {
      dum=a[imax][k];
      a[imax][k]=a[j][k];
      a[j][k]=dum;
      }
```

```
        *d = -(*d);
        vv[imax]=vv[j];
      }
      indx[j]=imax;
      if (a[j][j] == 0.0) a[j][j]=TINY;
      if (j != n) {
        dum=1.0/(a[j][j]);
        for (i=j+1;i<=n;i++) a[i][j] *= dum;
      }
    }
  }


  /*-------------------------------------------------------------------*/
  /* LU back-substitution of matrix */
  /*-------------------------------------------------------------------*/

  lubksb(l,u,n,indx,b,x)
       double l[N+1][N+1],u[N+1][N+1],b[],x[N+1];
       int n;

  {
    int i,ii=0,jj=0,j;
    double sum;
    double y[N+1];

    for (i=0 ; i<=n ; i++)  {
      sum=0.0;
      if (i==0) ii=0;
      if (ii!=0) {
        for (j=0 ; j<=i-1 ; j++) sum += y[j]*l[i][j];
        y[i]=b[i]-sum;
      }
      else {
        y[0]=b[0];
        ii=1; }
    }

    for (i=n ; i>=0 ; i--) {
      sum=y[i];
      if (i==n) jj=0;
      if (jj!=0) {
        for (j=i+1 ; j<=n ; j++) sum -= u[i][j]*x[j];
        x[i]=sum/u[i][i];
      }
      else { x[n]=sum/u[n][n];
           jj=1; }
    }

  }

  /*-------------------------------------------------------------------*/
  /* Multiply matrix to vector */
  /*-------------------------------------------------------------------*/

  multiply(inv_a,s,dv)

       double inv_a[N+1][N+1],s[N+1],dv[N+1];
```

```
{
  int i,j;
  float tempo;
  for (i=0 ; i<=N ;i++) dv[i]=0.0;
  for(i=0 ; i<=N ; i++) {
    for(j=0 ; j<=N ; j++)
      dv[i] += (inv_a[j][i]*s[j]);
  }
}
```

Function: simulation.c

```
#include "common.h"

/* Graphically simulates the bill-feeder in window (sim_window) */
/* Graphical output of ratio (graph_window) */


simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count)

     double x_roll,y_roll,r_roll;      /* roller coords. and radius   */
     double x_draw[2][MAX_NODES],y_draw[2][MAX_NODES];
     int ang_count;                    /* Counter for angle (graph) */

{

  int xx1,yy1,xx2,yy2;                 /* x,y coords for graphics */
  int xc,yc;
  int rr;                              /* radius of roller */
  int th_deg;                          /* angle in degrees */
  int gx1,gx2,gy1,gy2;                 /* graphics coordinates for graph
*/
  int max_ang_increments;              /* angluar increments for graph */
  int length_of_increment;             /* length (in pixels) of angluar
increment */
  float xf,yf,rf;                      /* Data from input */


/* Other variables */

  int j;
  int i,q,q2;
  int con_node;                        /* Contact node number */
  float choice;                        /* Input values */
  int remlength;                       /* Remaining length for rot'n */


  max_ang_increments = (3.4 - th_seg)/del_th;
  length_of_increment = 450/max_ang_increments;

  XClearWindow(display,sim_window);
  XFlush(display);
  XClearWindow(display,data_window);

  load_font(&font_info);

  draw_text(data_window,gc,font_info,hint.width,hint.height);

/* Draw Floor Bill */
  XDrawLine(display,sim_window,gc,50,257,500,257);
  XFlush(display);

/* Draw buckled Bill - left part first  (BOTH ROTATION AND STAGE 2) */
  for (j=0 ; j<2 ; j++) {

    for (i=0 ; i<NUMBER_OF_NODES-1 ; i++) {
```

```
        xx1=3214*x_draw[j][i];
        xx2=3214*x_draw[j][i+1];
        yy1=3214*y_draw[j][i];
        yy2=3214*y_draw[j][i+1];
        XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xx2,
              255-yy2);

      }

    }


 /* ONLY FOR STAGE 2 */
   if (flag) {
     q2 = 3214*l_rem;
     XDrawLine(display,sim_window,gc,50+xx2,255-yy2,50+xx2+q2,255-yy2);

 /* Now draw graph of Ratio vs. theta in graph window */

     XDrawLine(display,graph_window,gc,50,110,500,110); /* x-axis (theta)
*/
     XDrawLine(display,graph_window,gc,50,30,50,190);   /* y-axis (Ratio)
*/
     XDrawLine(display,graph_window,gc,45,70,500,70);   /* R=1 line */

     gx1 = 50 + length_of_increment*(ang_count-1);
     gx2 = 50 + length_of_increment*ang_count;

     if (ang_count == 1) gy1 = 110;     /* If this is the first data of
ratio */
     /* use origin for first point */
     if (ang_count != 1) gy1 = 110 - old_ratio*40;

     gy2 = 110 - ratio*40;              /* Note: I assume that +y axis
goes up */
     /* to a ratio of 2 (maximum) */

     XDrawLine(display,graph_window,gc,gx1,gy1,gx2,gy2);

     XFlush(display);

   }

 /* ONLY FOR ROTATION */
   if (!flag) {
     /* Draw Flat Part (tail) */
     remlength = 3214*(12 + xc0) + 50;
     q = xc0*3214;
     XDrawLine(display,sim_window,gc,50+q,255,remlength,255);

   }


 /* Draw Roller */

   xx1 = 3214*x_roll;               /* Coordinates of centre */
   yy1 = 3214*y_roll;
   /* N O T E !! :    NOT the coordinates of the centre to be given */
```

```
   xx2 = 3214*r_roll;

   xc = xx1 - xx2*sin(theta);
   yc = yy1 - xx2*cos(theta);

   XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xc,255-yc);

   /* theta1 is theta plus the angle of the segment of the roller */
   xc = xx1 - xx2*sin(theta1);
   yc = yy1 - xx2*cos(theta1);

   th_deg = 1*(theta*180/PI);
   XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xc,255-yc);

/* XDrawArc draws the arc between the straight segements of the roller.
The arguments are: */
/* ..... , angle from vertical of first clock-wise segment, -(angular
distance of 2nd straight side) */

  XDrawArc(display,sim_window,gc,50+xx1-xx2,255-yy1-
xx2,2*xx2,2*xx2,(270-th_deg)*64,-(100)*64);


  XFlush(display);

}
/*closes simulation */
```

Function: simulation_roller.c

```
#include "common.h"

/* Graphically Simulates the bill-feeder in window (sim_window) */


simulation(x_roll,y_roll,r_roll,x_draw,y_draw)

     double x_roll,y_roll,r_roll;      /* roller coords. and radius   */
     double x_draw[2][MAX_NODES],y_draw[2][MAX_NODES];
{

  int xx1,yy1,xx2,yy2;                 /* x,y coords for graphics */
  int xend,yend;

  float xf,yf,rf;                      /* Data from input */


  /* Other variables */

  int j;
  int i,q1,q2;
  int con_node;                        /* Contact node number */
  float choice;                        /* Input values */
  int remlength;                       /* Remaining length for rot'n */

  XClearWindow(display,sim_window);
  XFlush(display);

  XClearWindow(display,data_window);

  load_font(&font_info);

  draw_text(data_window,gc,font_info,hint.width,hint.height);

  /* Draw Floor Bill */
  XDrawLine(display,sim_window,gc,50,257,500,257);
  XFlush(display);

  if (!stage3) {
    /* Draw buckled Bill - left part first*/
    for (j=0 ; j<2 ; j++) {

      for (i=0 ; i<NUMBER_OF_NODES-1 ; i++) {
      xx1=3214*x_draw[j][i];
      xx2=3214*x_draw[j][i+1];
      yy1=3214*y_draw[j][i];
      yy2=3214*y_draw[j][i+1];
      XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xx2,
              255-yy2);
      }

    }

    if (flag) {
```

```
        q2 = 3214*l_rem;
        XDrawLine(display,sim_window,gc,50+xx2,255-yy2,50+xx2+q2,255-yy2);
      }

      if (!flag) {
        /* Draw Flat Part (tail) in case of pure rotation*/
        remlength = 3214*(12 + xc0) + 50;
        q1 = xc0*3214;
        XDrawLine(display,sim_window,gc,50+q1,255,remlength,255);
      }
    }

    if (stage3) {

      for (i=0 ; i<NUMBER_OF_NODES-1 ; i++) {
        xx1=3214*x_draw[0][i];
        xx2=3214*x_draw[0][i+1];
        yy1=3214*y_draw[0][i];
        yy2=3214*y_draw[0][i+1];
        XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xx2,
              255-yy2);
      }

      xend = 3214*122*cos(theta);
      yend = 3214*122*sin(theta);

      XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xx1+xend,255-
yy1+yend);

    }                                   /* closes if(stage3) */


    /* Draw Roller */
    xx1 = 3214*x_roll;                  /* Coordinates of centre */
    yy1 = 3214*y_roll;

    xx2 = 3214*r_roll;

    XDrawArc(display,sim_window,gc,50+xx1-xx2,255-yy1-
xx2,2*xx2,2*xx2,0,360*64);

    XFlush(display);


}
/*closes simulation */
```

Function: load_font.c

```
#include "common.h"

/* Loads font from XWindows lib. to be used in data window */

load_font(font_info)
     XFontStruct **font_info;
{
  char *fontname = "9x15";

  if((*font_info = XLoadQueryFont(display,fontname)) == NULL)

    {
      (void) fprintf( stderr,"Basic: Cannot open 9x15 font \n");
      exit(-1);
    }
}
```

Function: draw_text.c

```
#include "common.h"

/* Writes stuff in the data_window: moment convergence info, ratio */


draw_text(data_window,gc,font_info,win_width,win_height)

     Window data_window;
     GC gc;
     XFontStruct *font_info;
     unsigned int win_width, win_height;

{
  int y = 50;
  char *string1 = "Ratio of tangential to normal forces at contact";
  char *string2 = "This is another test for the ratio";
  int len1, len2, len3;
  char cd_ratio[50], cd_lm[50], cd_rm[50];
  int font_height;
  int initial_y_offset, x_offset;



  (void) sprintf(cd_ratio, "RATIO (Ft/Fn):  %f ", ratio_display);
  (void) sprintf(cd_lm,  "Left-part moment:   %f",left_mom);
  (void) sprintf(cd_rm,  "Right-part moment:  %f",right_mom);

  font_height = font_info->max_bounds.ascent + font_info-
>max_bounds.descent;

  initial_y_offset = win_height/2 - font_height - font_info-
>max_bounds.descent;

  x_offset = (int) win_width/4;

  len1 = strlen(cd_ratio);
  len2 = strlen(cd_lm);
  len3 = strlen(cd_rm);


  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset,
cd_ratio,len1);

  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset +
5*font_height, cd_lm,len2);
  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset +
6*font_height, cd_rm,len3);

}
```

Function: graphix_init.c

```
#include "common.h"

/* Prepares and initializes screen for window and graphics output. */

graphix_init(argc,argv)
     int argc;
     char **argv;

{

  display = XOpenDisplay("");          /* Screen Initialization */
  screen = DefaultScreen(display);
  cmap = XDefaultColormap(display,screen);


  Magenta.red=255*factor;
  Magenta.green=0*factor;
  Magenta.blue=255*factor;

  if(XAllocColor(display,cmap,&Magenta)==0) fail=1;

  DarkSlateGrey.red=47*factor;
  DarkSlateGrey.green=79*factor;
  DarkSlateGrey.blue=79*factor;

  if(XAllocColor(display,cmap,&DarkSlateGrey)==0) fail=1;

  White.red=255*factor;
  White.green=255*factor;
  White.blue=255*factor;

  if(XAllocColor(display,cmap,&White)==0) fail=1;

  /* Determine if color or black/white to be used */

  fail=1;

  if(fail) {
    background = WhitePixel(display,screen);
    foreground = BlackPixel(display,screen);

  }

  else{
    background = White.pixel;
    foreground = Magenta.pixel;
  }

  /* Specify window position and size for first window (simulation) */

 hint.x = 10;
 hint.y = 30;
 hint.width = XDisplayWidth(display,screen)/2;
 hint.height = XDisplayHeight(display,screen)/2;
```

```
   hint.flags = USPosition | USSize;

   /* Window creation */
   sim_window = XCreateSimpleWindow (display, DefaultRootWindow(display),
hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);


XSetStandardProperties(display,sim_window,"simulation","simulation",None
,
                    argv,argc,&hint);


   /* Specify window position and size for second window (data) */

   hint.x = 600;
   hint.y = 30;
   hint.width = XDisplayWidth(display,screen)/4;
   hint.height = XDisplayHeight(display,screen)/4;
   hint.flags = USPosition | USSize;


   data_window = XCreateSimpleWindow (display,
DefaultRootWindow(display), hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);

   XSetStandardProperties(display,data_window,"data","data",None,
                    argv,argc,&hint);


   /* Specify window position and size for third window (graphics) */

   hint.x = 10;
   hint.y = 550;
   hint.width = XDisplayWidth(display,screen)/2;
   hint.height = XDisplayHeight(display,screen)/4;
   hint.flags = USPosition | USSize;


   /* Window creation */
   graph_window = XCreateSimpleWindow (display,
DefaultRootWindow(display), hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);

   XSetStandardProperties(display,graph_window,"R / th
Graph","graph",None,
                    argv,argc,&hint);


   /* Restore these so that data_window doesn't get screwed-up */
```

```
    hint.width = XDisplayWidth(display,screen)/4;
    hint.height = XDisplayHeight(display,screen)/4;

    /* GC creation and initialization */

    gc = XCreateGC(display,sim_window,0,0);
    gc = XCreateGC(display,data_window,0,0);
    gc = XCreateGC(display,graph_window,0,0);

    XSetBackground(display,gc,background);
    XSetForeground(display,gc,foreground);

    /* Load and set font */

    f = XLoadFont(display,"9x15");
    if (!f) {
      fprintf("Couldn't load font %s.  \n",f);
      exit(-1);
    }

    /* Window mapping */

    XMapRaised (display,sim_window);
    XMapRaised (display,data_window);
    XMapRaised (display,graph_window);

    XFlush(display);
    sleep(2);


}
```

Function: graphix_init_roller.c

```
#include "common.h"

/* Prepares and initializes screen for window and graphics output. */

graphix_init(argc,argv)

     int argc;
     char **argv;

{

  display = XOpenDisplay("");           /* Screen Initialization */
  screen = DefaultScreen(display);
  cmap = XDefaultColormap(display,screen);


  Magenta.red=255*factor;
  Magenta.green=0*factor;
  Magenta.blue=255*factor;

  if(XAllocColor(display,cmap,&Magenta)==0) fail=1;

  DarkSlateGrey.red=47*factor;
  DarkSlateGrey.green=79*factor;
  DarkSlateGrey.blue=79*factor;

  if(XAllocColor(display,cmap,&DarkSlateGrey)==0) fail=1;

  White.red=255*factor;
  White.green=255*factor;
  White.blue=255*factor;

  if(XAllocColor(display,cmap,&White)==0) fail=1;

  /* Determine if color or black/white to be used */

  fail=1;

  if(fail) {
    background = WhitePixel(display,screen);
    foreground = BlackPixel(display,screen);

  }

  else{
    background = White.pixel;
    foreground = Magenta.pixel;
  }

  /* Specify window position and size */

  hint.x = 10;
  hint.y = 30;
  hint.width = XDisplayWidth(display,screen)/2;
```

```
  hint.height = XDisplayHeight(display,screen)/2;
  hint.flags = USPosition | USSize;   /* Program sets position */


  /* Window creation */
  sim_window = XCreateSimpleWindow (display, DefaultRootWindow(display),
hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);

XSetStandardProperties(display,sim_window,"simulation","simulation",None
,
                    argv,argc,&hint);

  /* GC creation and initialization */


  hint.x = 600;
  hint.y = 30;
  hint.width = XDisplayWidth(display,screen)/4;
  hint.height = XDisplayHeight(display,screen)/4;
  hint.flags = USPosition | USSize;

  data_window = XCreateSimpleWindow (display,
DefaultRootWindow(display), hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);
  XSetStandardProperties(display,data_window,"data","data",None,
                    argv,argc,&hint);

  hint.width = XDisplayWidth(display,screen)/4;
  hint.height = XDisplayHeight(display,screen)/4;

  gc = XCreateGC(display,sim_window,0,0);
  gc = XCreateGC(display,data_window,0,0);

  XSetBackground(display,gc,background);
  XSetForeground(display,gc,foreground);


  /* Load and set font */

  f = XLoadFont(display,"9x15");
  if (!f) {
    fprintf("Couldn't load font %s.  \n",f);
    exit(-1);
  }


  /* Window mapping */

  XMapRaised (display,sim_window);
  XMapRaised (display,data_window);
  XFlush(display);
  sleep(2);
```

}

Program: dynamics.c

```
#include <stdio.h>
#include <math.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/X.h>
#include <X11/keysym.h>

#define MAX_NODES 35                    /* Maximum number of nodes */
#define NUMBC 3                         /* Max. number of BC's */
#define N     2                         /* Max. number of BC's */
#define NUMBER_OF_NODES 35
#define SHAPES 20
#define PI 3.141592654

#define NUM_EQN 5                       /* No. of eqns integrated */
#define TINY 1.0e-20                    /* For inverse routine */


/* Properties of yen */

#define L 0.14
#define B 0.076
#define t 1.0e-4
#define w 1.1e-3/0.14
#define cof 0.15
#define E 15000000
#define I 6.33e-12


/* Variables */

double x2_desired[NUMBC];               /* Desired BC at x2 */
double V[NUMBC];                        /* Free BC at x1 */
double delv[NUMBC];                     /* Change in V for Jacobian */
double fx_guess;                        /* Guessed force_x at x1 */
double fy_guess;                        /* Guessed force_y at x1 */
double m_guess;                         /* Guessed moment at x1 */
double old_fx_guess1;                   /* Prev. guessed fx at x1 (lp) */
double old_fy_guess1;                   /* Prev. guessed fy at x1 (lp)*/
double old_m_guess1;                    /* Prev. guessed m at x1  (lp)*/
double old_fx_guess2=0.0;               /* Prev. guessed fx at x1 (rp) */
double old_fy_guess2=0.0;               /* Prev. guessed fy at x1 (rp)*/
double old_m_guess2=0.0;                /* Prev. guessed m at x1  (rp)*/
double ratio,ratio_display,old_ratio;   /* Ratio Ft/Fn at contact */
double clip;                            /* Changes in V[] */
double tol1 = 0.0001;                   /* Tolerance for left part */
double tol2 = 0.000002;                 /* Tolerance for right part */
double dt =0.003;                       /* Time step for dynamics */


/* Variables for new position */

double dy_roll = 0.002;                 /* y-momvt of roller */
double del_th = 0.2;                    /* Angular increment for calcs */
```

```
                                        /* in radians */
double l1,l2,l21,l22,l_set;             /* Contact length with roller */
double l_rem;                           /* Length Remaining in tail */
double th_seg,th_fin,th_cont;           /* Segemented roller arc length */
double dl = 0.005;                      /* Step size for bill length */
double xc0;                             /* Initial x-coord. of contact */
double lc0;                             /* Initial contact length */
double r_radius = 0.01;                 /* Roller radius */
double theta,theta1,th_roll;            /* Current roller angle */
double th_trial,th_old;                 /* To find the equal moments */
double x_cont;                          /* x trial contact point */
double y_cont;                          /* y trial contact point */
double left_mom;                        /* moment from left */
double right_mom;                       /* moment from right */
double avg_mom;                         /* Average of left & right */
double wt;                              /* Total weight */
double ls;                              /* Length of segment 1/n) */


/* Variables at each node */

double fx[MAX_NODES];                   /* static force in x direction */
double vx[MAX_NODES][2];                /* velocity in x direction */
double ax[MAX_NODES];                   /* acceleration in x direction */

double fy[MAX_NODES];                   /* static force in y direction */
double vy[MAX_NODES][2];                /* velocity in y direction */
double ay[MAX_NODES];                   /* acceleration in y direction*/

double m[MAX_NODES];                    /* moment */
double omega[MAX_NODES][2];             /* angular velocity */
double ath[MAX_NODES];                  /* angular acceleration */

double x_pos[MAX_NODES][2];             /* x-pos (two time-steps) */
double y_pos[MAX_NODES][2];             /* y_pos (two time-steps) */
double th[MAX_NODES][2];                /* angle with the horizontal */
double dthds[MAX_NODES];                /* 'curvature' of element */
double add_mom[MAX_NODES];              /* additional internal */
                                        /* moment caused by folding */


int n=35;                               /* Number of nodes */
int nn;                                 /* Iteration counter */
int cnt1;                               /* General counter1 */
int flag,th_calc;                       /* Left part : flag=0 */
                                        /* Right part: flag=1 */
int len_flag;                           /* Flag for change in bill length
*/
int eq_mom=1;                           /* Equal moments flag */


double l;                               /* Length of yen (metres) */

/* Declare pointers for output files */
double xx1,yy1;                         /* For output to file */
double dummy;                           /* scanf var. */
```

```
/* XWindows variables */

Display *display;                          /* Pointer to display */
Window sim_window, data_window;            /* Windows */
Window graph_window;
int screen;                                /* Screen ID */
GC gc;                                     /* Graphic Content */
Font f;                                    /* Font for text display */
XSizeHints hint;                           /* Window specification */
unsigned long foreground,background;       /* Pixel values */
XColor DarkSlateGrey,Magenta;              /* Color Variables */
XColor White;
Colormap cmap, cmap1;                      /* Default colormap */
XFontStruct *font_info;

int fail;                                  /* Colors/Black-White */
double factor=257;                         /* Color factor */



FILE *fpo, *fopen();

main(argc,argv)

     int argc;
     char **argv;

{
  double testval;
  double Fx_left;                          /* Resultant force in x-dir */
  double Fy_left;                          /* Resultant force in y-dir */

  double outflag=999;                      /* Flag for output file */
  float dd3,dd4,dd5,dd6,dd7;               /* For keyboard input */
  int floor_node;                          /* Node where bill passes floor */
  int rept=1;                              /* scanf variable */
  int i,j,kk;                              /* Various counters */
  int n_iter;                              /* Shoot counter */
  int in;                                  /* Main iteration counter */
  int ang_count=0;                         /* for angular increments (graph)
*/
  int clear_flag=0;                        /* 0 to clear window, 1 to keep */

  double bc0,bc1,bc2;                      /* boundary conditions */
  double ic0,ic1,ic2;                      /* initial conditions  */

  double x_roll,y_roll,r_roll;             /* for graphics */
  double x_draw[2][MAX_NODES];
  double y_draw[2][MAX_NODES];
  double x_left[MAX_NODES][2];
  double y_left[MAX_NODES][2];
  double th_left[MAX_NODES][2];
  double x_right[MAX_NODES][2];
  double y_right[MAX_NODES][2];
  double th_right[MAX_NODES][2];

  double y_static;                         /* the y-value of the last node */
                                           /* under the static solution */
```

```
  float dd;                              /* scanf var. */


  /* Open file 'elshape' for the shape fo the elastica */
  fpo = fopen("elshapenew","w");

  r_roll = 0.01;

  printf("\nContact length          ");
  scanf("%f",&dd7);
  lc0 = dd7;
  l1 = lc0;

  printf("\nInitial x-position      ");
  scanf("%f",&dd7);
  xc0 = dd7;
  x_roll = xc0;

  printf("\nFinal roller angle   (max. 3.216125 ");
  scanf("%f",&dd7);
  th_fin = dd7;
  y_roll = r_roll;

    graphix_init(argc,argv);


/* Correction terms for fx and fy */
  delv[0]=0.0000000005;          /* Correction for fx */
  delv[1]=0.0000000005;          /* Correction for fy */
  delv[2]=0.0000000005;          /* Correction for m  */
  clip = 0.5;                    /* Clip value */

  if (xc0 >=0.07) {
    old_fx_guess1 = -0.5;
    old_fy_guess1 = 0.06;
    old_m_guess1 = -0.007;
    old_fx_guess2 = 0.0;
    old_fy_guess2 = 0.0;
    old_m_guess2 = 0.0;

  }

  if (xc0 < 0.07) {
    old_fx_guess1 = -0.8;
    old_fy_guess1 = 0.09;
    old_m_guess1 = -0.008;
    old_fx_guess2 = 0.0;
    old_fy_guess2 = 0.0;
    old_m_guess2 = 0.00;
  }


  for (j=0; j<MAX_NODES ; j++) {


    m[j]=0.0;
```

```
      omega[j][0]=0.0;
      omega[j][1]=0.0;
      ath[j]=0.0;

      fx[j]=0.0;
      vx[j][0]=0.0;
      vx[j][1]=0.0;
      ax[j]=0.0;

      fy[j]=0.0;
      vy[j][0]=0.0;
      vy[j][1]=0.0;
      ay[j]=0.0;

      x_pos[j][0]= lc0*((1.0*j)/(1.0*(MAX_NODES-1)));;
      x_pos[j][1]=0.0;
      y_pos[j][0]=0.0;
      y_pos[j][1]=0.0;
      th[j][0]=0.0;
      th[j][1]=0.0;
      dthds[j]=0.0;

    }


    ratio_display = 0.0;
    theta = 0.0;
    theta1 = 0.0;
    flag=0;                                  /* Set for left part first */
    th_roll = 0.52;
/*     th_roll = 0.174533;*/
     th_seg = 1.74533;                       /* 100 deg. */

/*************************************************************************/
/*      Main iteration loop - Different angles of theta on roller      */
/*************************************************************************/

/* Rotation FIRST */


   while (theta1 <= th_seg) {

     theta1 += th_roll;
     l1 += r_radius*th_roll;
     l2 = L - l1;

     x_cont = xc0;
     y_cont = dy_roll;

     printf("\n %f",l1);



/* ------------------------------------------------------------ */
/* Calculate elastica and element properties for left part     */
/* ------------------------------------------------------------ */
```

```
    ls = ll/(1.0*(n-1.0));              /* Segment length */
    l = ll;                             /* Length of left part */



/* This is additional moment data representing curved bills */
/* To calculate for straight bills, remove additional moment */

/*
          add_mom[13] = 0.007319;
          add_mom[14] = 0.006592;
          add_mom[15] = 0.006113;
          add_mom[16] = 0.006000;
          add_mom[17] = 0.006205;
          add_mom[18] = 0.006000;
          add_mom[19] = 0.006113;
          add_mom[20] = 0.006592;
          add_mom[21] = 0.007319;

*/

/* ------------------------------------------------------------ */
/*        Set Boundary Conditions at End for rotation           */
/* ------------------------------------------------------------ */


/********************************  STATIC SOLUTION START  **********/


    for(j=0 ; j<MAX_NODES ; j++) {

      ax[j] = 0.0;
      vx[j][1] = 0.0;

      ay[j] = 0.0;
      vy[j][1] = 0.0;

      ath[j] = 0.0;
      omega[j][1] = 0.0;

    }

    fx_guess = old_fx_guess1;
    fy_guess = old_fy_guess1;
    m_guess = old_m_guess1;

    set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

    set_boundary_conditions(x_cont,0.0,0.0);
    /* x-coord of contact point */
    /* y-coord of contact point */
    /* ending angle */



    y_pos[n-1][1] = 1.0;                /* Trick to pass into loop */
```

```
/* ------------------------------------------------------------ */
/* Shoot loop for left part */
/* ------------------------------------------------------------ */


    n_iter=0;
    while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[1]) > tol1) && (n_iter
< 30)) {


      n_iter++;

      fx[0]=V[0];
      fy[0]=V[1];
      m[0]=V[2];

      Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

      shoot(V,x2_desired,delv,clip,flag);

    }

    y_static = y_pos[14][1];

    old_fx_guess1 = V[0];
    old_fy_guess1 = V[1];
    old_m_guess1 = V[2];


/* ------------------------------------------------------------ */
/*    Determine shape of elastica (coordinates of nodes)        */
/* ------------------------------------------------------------ */

    check_shape(x_draw,y_draw,flag);

    simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count,clear_flag);


    for (i=0 ; i<35 ; i++) {
      xx1 = x_pos[i][1];
      yy1 = -y_pos[i][1];
    }

/************************************  STATIC SOLUTION END **********/


/*********************************** DYNAMIC SOLUTION START **********/

    for(kk=0 ; kk<1; kk++) {          /* Iterations to convergence */
                                      /* of dynamic solution */

      for(j=0; j<MAX_NODES ; j++) {

      m[j]=0.0;
      omega[j][0]=0.0;

      fx[j]=0.0;
```

```
        fy[j]=0.0;

        dthds[j]=0.0;

        }

        for(j=0 ; j<MAX_NODES ; j++) {

/* Velocities */

        vx[j][1] = (x_pos[j][1] - x_pos[j][0])/dt;
                                        /* x-velocity of elements */

        vy[j][1] = (y_pos[j][1] - y_pos[j][0])/dt;
                                        /* y_velocity of elements */

        omega[j][1] = (th[j][1] - th[j][0])/dt;
                                        /* ang. velocity of elements */


/* Accelerations */

        ax[j] = (vx[j][1] - vx[j][0])/dt;
                                        /* x-acceleration of elements */

        ay[j] = (vy[j][1] - vy[j][0])/dt;
                                        /* y-acceleration of elements */

        ath[j] = (omega[j][1] - omega[j][0])/dt;
                                        /* ang acceleration of elements */

        }

        for(j=0 ; j<MAX_NODES ; j++)  {


        x_pos[j][0] = x_pos[j][1];
        y_pos[j][0] = y_pos[j][1];
        th[j][0] = th[j][1];
        vx[j][0] = vx[j][1];
        vy[j][0] = vy[j][1];
        omega[j][0] = omega[j][1];

        }

        set_initial_conditions(0.0,0.0,0.0);

        set_boundary_conditions(x_cont,0.0,0.0);


        n_iter=0;

        y_pos[n-1][1] = 1.0;            /* Trick to pass into loop */


        while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[1]) > tol1) &&
(n_iter < 30)) {
```

```
      n_iter++;

      fx[0]=V[0];
      fy[0]=V[1];
      m[0]=V[2];

      Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

      shoot(V,x2_desired,delv,clip,flag);


      }                              /* Closes while(fabs) */


      check_shape(x_draw,y_draw);
      background = DarkSlateGrey.pixel;
      foreground = White.pixel;
      XSetBackground(display,gc,background);
      XSetForeground(display,gc,foreground);
      clear_flag=1;


simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count,clear_flag);

      clear_flag=0;

      background = DarkSlateGrey.pixel;
      foreground = Magenta.pixel;
      XSetBackground(display,gc,background);
      XSetForeground(display,gc,foreground);

      sleep(2);


    }                              /* Closes for(kk..) - dynamics */
                                   /* iteration loop */


/*********************************  DYNAMIC SOLUTION END  **********/


  }                                /* Closes while(theta1<=th_seg)  */

  for(j=0 ; j<MAX_NODES ; j++) {

    x_left[j][0] = x_pos[j][1];
    y_left[j][0] = y_pos[j][1];
    th_left[j][0] = th[j][1];

    x_right[j][0] = xc0 + 12*((1.0*j)/(1.0*MAX_NODES));
    y_right[j][0] = 0.0;
    th_right[j][0] = 0.0;

  }

  scanf("%f",&dd);
```

```
/* This loop is for the different y-positions of the roller */
  flag=0;
  theta = 0.0;

  l_rem = 0.0;
  l22 = l2;
  l21 = L - l22;

  th_old = theta;

  while (theta1 <= th_fin) {
    th_calc=1;


    theta1 += del_th;
    theta += del_th;

    x_cont = xc0 - r_radius*sin(theta);

    y_cont = r_radius*(1 + sin(theta-1.5708));



    th_trial = th_old;            /* First start with the roller */
                                  /* ending angle */


/************************************ STATIC SOLUTION START *****/


    while(eq_mom) {

      len_flag=1;
      l_rem = 0.0;

      while(len_flag) {

/* Left Part First */
      flag=0;

      ls = l21/(1.0*(n-1.0));          /* Segment length */
      l = l21;                         /* Length of left part */


      for(j=0 ; j<MAX_NODES ; j++) {

        ax[j] = 0.0;
        vx[j][1] = 0.0;

        ay[j] = 0.0;
        vy[j][1] = 0.0;

        ath[j] = 0.0;
        omega[j][1] = 0.0;

      }
```

```
      fx_guess = old_fx_guess1;
      fy_guess = old_fy_guess1;
      m_guess = old_m_guess1;


      set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

      set_boundary_conditions(x_cont,y_cont,th_trial);
      /* x-coord of contact point */
      /* y-coord of contact point */
      /* ending moment */

      y_pos[n-1][1] = 1.0;            /* Trick to pass into loop */

/* ------------------------------------------------------------ */
/* Shoot loop for left part */
/* ------------------------------------------------------------ */


      n_iter=0;
      while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[1]) > tol1) &&
(n_iter < 30)) {

        n_iter++;

        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        shoot(V,x2_desired,delv,clip,flag);


      }                          /* Closes while(fabs) */


      left_mom = m[n-1];
      old_fx_guess1 = V[0];
      old_fy_guess1 = V[1];
      old_m_guess1 = V[2];

      check_shape(x_draw,y_draw,flag);

      for(j=0 ; j<MAX_NODES ; j++) {

      x_left[j][1] = x_pos[j][1];
      y_left[j][1] = y_pos[j][1];
      th_left[j][1] = th[j][1];

      }


      Fx_left = fx[n-1];
      Fy_left = fy[n-1];
```

```
/* RIGHT  Part Next */

      flag=1;


      ls = 122/(1.0*(n-1.0));          /* Segment length */
      l = 122;                         /* Length of left part */


      fx_guess = 0.0;
      fy_guess = 0.0;
      m_guess = 0.0;

      set_initial_conditions(x_cont,-y_cont,th_trial); /* x,y,m at s=0
*/


      y_pos[n-1][1] = 1.0;             /* Trick to pass into loop */

/* ------------------------------------------------------------ */
/* Shoot loop for right part */
/* ------------------------------------------------------------ */


      n_iter=0;
      while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[0]) > tol1) &&
(n_iter < 30)) {

        n_iter++;

        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        set_boundary_conditions(0.0,cof*fabs(fy[n-1]),0.0);
        /* x-coord of contact point */
        /* y-coord of contact point */
        /* ending angle */

        shoot(V,x2_desired,delv,clip,1);

      }                              /* Closes while(fabs) */


      right_mom = m[0];
      old_fx_guess2 = V[0];
      old_fy_guess2 = V[1];
      old_m_guess2 = V[2];

      below_floor(floor_node);

      if (floor_node>33) len_flag=0;


      check_shape(x_draw,y_draw,flag);
```

```
      for(j=0 ; j<MAX_NODES ; j++) {

        x_right[j][1] = x_pos[j][1];
        y_right[j][1] = y_pos[j][1];
        th_right[j][1] = th[j][1];

      }

      }                                   /* Closes while(len_flag) */

/* End this run by drawing the shape and calculating the ratio */


      calculate_ratio(Fx_left,Fy_left);

      XClearWindow(display,data_window);
      draw_text(data_window,gc,font_info,hint.width,hint.height);
      XFlush(display);

      printf("\n*** Moments:  L: %f
R:%f",left_mom,right_mom);

      if(left_mom<0.0) th_trial += 10.0*(fabs(fabs(right_mom)-
fabs(left_mom)));
      if( (right_mom<0.0) && (fabs(right_mom) > fabs(left_mom)))
      th_trial -= 20*(fabs(fabs(right_mom)-fabs(left_mom)));

      l22=l2;
      if ( fabs(fabs(right_mom)-fabs(left_mom))<0.001 ) eq_mom = 0;


    }
                                  /* Closes eq_mom */



    ratio_display = ratio;
    th_old = th_trial;
    eq_mom=1;

    check_shape(x_draw,y_draw,flag);

    ang_count++;
    simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count,clear_flag);
    old_ratio = ratio;
    printf("\n\n ********* MOVE TO NEW ANGLE *********\n\n");

/************************************ STATIC SOLUTION END *****/

/********************************** DYNAMIC SOLUTION START ******/
/* First for the left Part */

    len_flag=1;
```

```
      for(kk=0 ; kk<1 ; kk++) {

        while(len_flag) {

        ls = l1/(1.0*(n-1.0));
        l = l1;

        flag=0;


        for (j=0 ; j<MAX_NODES ; j++) {

                                    /* Calculate velocities */
          vx[j][1] = (x_left[j][1] - x_left[j][0])/dt;

          vy[j][1] = (y_left[j][1] - y_left[j][0])/dt;

          omega[j][1] = (th_left[j][1] - th_left[j][0])/dt;

                                    /* Calculate accelerations */
          ax[j] = (vx[j][1] - vx[j][0])/dt;

          ay[j] = (vy[j][1] - vy[j][0])/dt;

          ath[j] = (omega[j][1] - omega[j][0])/dt;

        }


        for (j=0 ; j<MAX_NODES ; j++){

          printf("\n [%d.] x=%f vx=%f ax=%f   y=%f vy=%f ay=%f  th=%f
vth=%f ath=%f",j,x_pos[j][1]

,vx[j][1],ax[j],y_pos[j][1],vy[j][1],ay[j],th[j][1],omega[j][1],ath[j]);

          x_left[j][0] = x_left[j][1];
          y_left[j][0] = y_left[j][1];
          th_left[j][0] = th_left[j][1];

        }

        scanf("%f",&dd);

        fx_guess = old_fx_guess1;       /* Prepare for left part shooting
*/
        fy_guess = old_fy_guess1;
        m_guess = old_m_guess1;

        set_initial_conditions(0.0,0.0,0.0); /* x,y,th at s=0 */

        printf("\n %f %f %f      %f %f
%f",x_cont,y_cont,th_trial,fx_guess,fy_guess,m_guess);

        set_boundary_conditions(x_cont,y_cont,th_trial);

        /* x-coord of contact point */
```

```
        /* y-coord of contact point */
        /* ending moment */

        y_pos[n-1][1] = 1.0;            /* Trick to pass into loop */

 /* ------------------------------------------------------------ */
 /* Shoot loop for left part */
 /* ------------------------------------------------------------ */


        n_iter=0;
        while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[1]) > tol1) &&
 (n_iter < 30)) {

        n_iter++;

        fx[0]=V[0];
        fy[0]=V[1];
        m[0]=V[2];

        Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

        shoot(V,x2_desired,delv,clip,flag);


        }                           /* Closes while(fabs) */

        for (j=0 ; j<MAX_NODES ; j++) {

          printf("\n %f %f %f",x_pos[j][1],y_pos[j][1],th[j][1]);

        }

        left_mom = m[n-1];
        old_fx_guess1 = V[0];
        old_fy_guess1 = V[1];
        old_m_guess1 = V[2];

        check_shape(x_draw,y_draw,flag);

        for(j=0 ; j<MAX_NODES ; j++) {

          x_left[j][1] = x_pos[j][1];
          y_left[j][1] = y_pos[j][1];
          th_left[j][1] = th[j][1];

        }

        Fx_left = fx[n-1];
        Fy_left = fy[n-1];


 /* Right Part Next */

        flag=1;
        len_flag=1;

        ls = l22/(1.0*(n-1.0));
```

```
        l=122;

        for (j=0 ; j<MAX_NODES ; j++) {

                                    /* Calculate velocities */
          vx[j][1] = (x_right[j][1] - x_right[j][0])/dt;

          vy[j][1] = (y_right[j][1] - y_right[j][0])/dt;

          omega[j][1] = (th_right[j][1] - th_right[j][0])/dt;

                                    /* Calculate accelerations */
          ax[j] = (vx[j][1] - vx[j][0])/dt;

          ay[j] = (vy[j][1] - vy[j][0])/dt;

          ath[j] = (omega[j][1] - omega[j][0])/dt;

        }

        for (j=0 ; j<MAX_NODES ; j++){

          x_right[j][0] = x_right[j][1];
          y_right[j][0] = y_right[j][1];
          th_right[j][0] = th_right[j][1];

        }

        fx_guess=0.0;
        fy_guess=0.0;
        m_guess=0.0;


        set_initial_conditions(x_cont,-y_cont,th_trial); /* x,y,m at s=0
*/


        y_pos[n-1][1] = 1.0;            /* Trick to pass into loop */

/* ------------------------------------------------------------ */
/* Shoot loop for right part */
/* ------------------------------------------------------------ */


       n_iter=0;
       while( (fabs(fabs(y_pos[n-1][1]) - x2_desired[0]) > tol1) &&
(n_iter < 30)) {

         n_iter++;

         fx[0]=V[0];
         fy[0]=V[1];
         m[0]=V[2];

         Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

         set_boundary_conditions(0.0,cof*fabs(fy[n-1]),0.0);
         /* x-coord of contact point */
```

```
        /* y-coord of contact point */
        /* ending angle */

        shoot(V,x2_desired,delv,clip,1);

      }                          /* Closes while(fabs) */


      right_mom = m[0];
      old_fx_guess2 = V[0];
      old_fy_guess2 = V[1];
      old_m_guess2 = V[2];

      below_floor(floor_node);

      if (floor_node>33) len_flag=0;

      for(j=0 ; j<MAX_NODES ; j++) {

        x_right[j][1] = x_pos[j][1];
        y_right[j][1] = y_pos[j][1];
        th_right[j][1] = th[j][1];

      }

      }                        /* closes while(len_flag) */

      for(j=0 ; j<MAX_NODES ; j++) {

      printf("\n [%d.]  xl= %f  yl= %f  thl= %f    xr= %f  yr=%f
thr=%f",j,x_left[j][1],y_left[j][1],
            th_left[j][1],x_right[j][1],y_right[j][1],th_right[j][1]);

      }

      scanf("%f",&dd);

      check_shape(x_draw,y_draw,flag);

      background = DarkSlateGrey.pixel;
      foreground = White.pixel;
      XSetBackground(display,gc,background);
      XSetForeground(display,gc,foreground);
      clear_flag=1;


simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count,clear_flag);

      clear_flag=0;

      background = DarkSlateGrey.pixel;
      foreground = Magenta.pixel;
      XSetBackground(display,gc,background);
      XSetForeground(display,gc,foreground);
```

```
    }                              /* closes for(kk...) - the dynamic loop
*/


/************************************ DYNAMIC SOLUTION END *****/
  }                                /* Closes while(y_roll...) */


/*-------------------------------------------------------------------*/
    /* Clear window */
  sleep (5);
  XClearWindow(display,sim_window);
  XFlush(display);

/* Termination */

  XFreeGC (display,gc);
  XDestroyWindow (display,sim_window);
  XCloseDisplay (display);
  exit(0);


}                                          /* Closes MAIN */


set_initial_conditions(ic0,ic1,ic2)

     double ic0,ic1,ic2;

{

/* Known boundary conditions at s=0 for left part*/
  if(!flag){
    x_pos[0][1]=ic0;
    y_pos[0][1]=ic1;
    th[0][1]=ic2;
  }

/* Known boundary conditions at s=0 for left part*/
  if(flag) {
    x_pos[0][1]=ic0;
    y_pos[0][1]=ic1;
    th[0][1]=ic2;
  }

/*  Unknown Boundary Conditions at s=0 (guessed) */
  V[0] = fx_guess;
  V[1] = fy_guess;
  V[2]  = m_guess;

}


set_boundary_conditions(bc0,bc1,bc2)

     double bc0,bc1,bc2;
```

```
{

/* Generic case */

  x2_desired[0]=bc0;                    /* first  bound. condition */
  x2_desired[1]=bc1;                    /* second bound. condition */
  x2_desired[2]=bc2;                    /* third  bound. condition */


}



shoot(V,x2_desired,delv,clip,flag)

/* Variables */

     double V[NUMBC];                   /* Free BC at x1 */
     double x2_desired[NUMBC];                /* Desired BC at x2 */
     double delv[NUMBC];                /* Change in V for Jacobian */
     double clip;                  /* Changes in V[] */
     int flag;                          /* Distinguish between left and */
                                  /* right part */

{
  double F[NUMBC];                      /* Discrepancy vector at x2 */
  double s[NUMBC];                        /* Discr. vector */
  double dv[NUMBC];                     /* Temp. discrepancy vector */
  double dxc,dyc;                  /* Discrepancy terms for x,y_pos */
  double dxc1,dyc1;                        /* Discr. terms for x,y_pos */
  double dfdv[NUMBC][NUMBC];            /* Jacobian matrix */
  double a[NUMBC][NUMBC];               /* Jacobian matrix backup */
  double inv_dfdv[NUMBC][NUMBC];        /* Inverse Jacobian matrix */
  double det_dfdv;                      /* Determinant of Jacobian matrix
*/
  double sav;                           /* Storage variable */
  double mom_cal;


  int i,j,k,q;                     /* Various counters */

/* ------------------------------------------------------------ */
/*                Trial Integration of Equations                */
/* ------------------------------------------------------------ */

  if(!flag){
    F[0] = x_pos[n-1][1] - x2_desired[0];
    F[1] = fabs(y_pos[n-1][1]) - x2_desired[1];
    F[2] = th[n-1][1] - x2_desired[2];
  }
  if(flag){
    F[0] = fabs(y_pos[n-1][1]) - x2_desired[0];
    F[1] = fx[n-1] - x2_desired[1];
    F[2] = m[n-1] - x2_desired[2];
  }


/* ------------------------------------------------------------ */
```

```
/* Vary fx[0] */
/* ------------------------------------------------------------ */

  sav = fx[0];
  fx[0] += delv[0];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

  if(!flag){
    dv[0] = x_pos[n-1][1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1][1]) - x2_desired[1];
    dv[2] = th[n-1][1] - x2_desired[2];
      }

  if(flag){
    dv[0] = fabs(y_pos[n-1][1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
  }

  dfdv[0][0] = (dv[0] - F[0])/delv[0];
  dfdv[1][0] = (dv[1] - F[1])/delv[0];
  dfdv[2][0] = (dv[2] - F[2])/delv[0];

  fx[0] = sav;                          /* Return fx to original */
                                /* value before increment */

/* ------------------------------------------------------------ */
/* Vary fy[0] */
/* ------------------------------------------------------------ */

  sav = fy[0];
  fy[0] += delv[1];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);


  if(!flag){
    dv[0] = x_pos[n-1][1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1][1]) - x2_desired[1];
    dv[2] = th[n-1][1] - x2_desired[2];
  }
  if(flag){
    dv[0] = fabs(y_pos[n-1][1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
```

```
  }


  dfdv[0][1] = (dv[0] - F[0])/delv[1];
  dfdv[1][1] = (dv[1] - F[1])/delv[1];
  dfdv[2][1] = (dv[2] - F[2])/delv[1];

  fy[0] = sav;                          /* Return fy to original */
                                  /* value before increment */

/* ------------------------------------------------------------ */
/* Vary m[0] */
/* ------------------------------------------------------------ */


  sav = m[0];
  m[0] += delv[2];


  V[0] = fx[0];
  V[1] = fy[0];
  V[2] = m[0];


  Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

  if(!flag){
    dv[0] = x_pos[n-1][1] - x2_desired[0];
    dv[1] = fabs(y_pos[n-1][1]) - x2_desired[1];
    dv[2] = th[n-1][1] - x2_desired[2];
  }
  if(flag){
    dv[0] = fabs(y_pos[n-1][1]) - x2_desired[0];
    dv[1] = fx[n-1] - x2_desired[1];
    dv[2] = m[n-1] - x2_desired[2];
  }

  dfdv[0][2] = (dv[0] - F[0])/delv[2];
  dfdv[1][2] = (dv[1] - F[1])/delv[2];
  dfdv[2][2] = (dv[2] - F[2])/delv[2];

  m[0] = sav;                                 /* Return m to original */
                                        /* value before increment */

  for (i=0 ; i<3 ; i++) {
    for(j=0 ; j<3 ; j++)
      a[i][j]=dfdv[i][j];
  }



  calculate_inverse(a,inv_dfdv);


  s[0]=F[0];
  s[1]=F[1];
  s[2]=F[2];
```

```
    multiply(inv_dfdv,s,dv);

    dv[0] = -dv[0];
    dv[1] = -dv[1];
    dv[2] = -dv[2];


    if ( fabs(dv[0]) > (clip*fabs(dv[0])) || fabs(dv[1]) >
  (clip*fabs(dv[1])) || fabs(dv[2]) > (clip*fabs(dv[2])) ) {

      dv[0] *= clip;
      dv[1] *= clip;
      dv[2] *= clip;
    }


    fx[0] += dv[0];
    fy[0] += dv[1];
    m[0]  += dv[2];


    V[0] = fx[0];
    V[1] = fy[0];
    V[2] = m[0];


    Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n);

    mom_cal=0.0;
    for (j=1 ; j<n ; j++) {
      mom_cal += (w*ls) * ((x_pos[j-1][1]+x_pos[j][1])/2.0);
    }




}                                       /* Closes shoot */


calculate_ratio(Fx_left,Fy_left)

     double Fx_left,Fy_left;

{
  double Fx_res,Fy_res;
  double F_tangent,F_normal;
  int r_node;

  Fx_res = fx[0] + Fx_left;
  Fy_res = fy[0] + Fy_left;

  F_tangent = fabs(Fx_res*cos(th[0][1]) + Fy_res*sin(th[0][1]));
  F_normal  = fabs(Fx_res*sin(th[0][1]) + Fx_res*cos(th[0][1]));
```

```
    ratio = F_tangent / F_normal ;

  }


  below_floor(floor_node)

       int floor_node;

  {

    int i;

    len_flag = 0;

    for (i=0 ; i<34 ; i++) {

      if(y_pos[i][1]>0.0002) len_flag=1;
      floor_node = i;
    }

    if (len_flag) {

      l22 -= dl;
      l_rem += dl;
      printf("\n BELOW %f %f %d",l22,l_rem,floor_node);
      y_pos[n-1][1] = 1.0;
    }

  }


  check_shape(x_draw,y_draw,flag)

       double x_draw[2][MAX_NODES];
       double y_draw[2][MAX_NODES];
       int flag;

  {

    int cnt1,i;

    cnt1=0;


    if(m[0]<0.0){
      for (i=1 ; i<34 ; i++) {
        if((m[i]*m[i-1])<0.0) cnt1++;
      }
      /*          if (cnt1>1)
            printf("\nWrong shape !!; %d changes in curvature\n",cnt1);
  */
    }

    /*      if(m[0]>=0.0) printf("\nWrong shape !! : Initial moment
  m[0]>=0\n");*/
```

```
  for (i=0; i<MAX_NODES ; i++) {
    x_draw[flag][i]=x_pos[i][1];
    y_draw[flag][i]=-y_pos[i][1];
  }
}


/* ------------------------------------------------------------ */
/*                      Runge-Kutta integrator                  */
/* ------------------------------------------------------------ */

Runge_kutta(fx,fy,m,th,x_pos,y_pos,ls,n)

     double
fx[],fy[],m[],th[MAX_NODES][2],x_pos[MAX_NODES][2],y_pos[MAX_NODES][2];
     double ls;
     int n;


{
  double ybeg[NUM_EQN];          /* Begining of t-step */
  double yend[NUM_EQN];          /* End of t-step */
  int i,k;
  double l1 = 0.0;                    /* Distance from origin */
  double Total_weight;           /* Total weight (correction) */


  k=0;


  ybeg[0]=fx[0];
  ybeg[1]=fy[0];
  ybeg[2]=m[0];
  ybeg[3]=th[0][1];
  ybeg[4]=0.0;

  while(1){

    l1 += ls;

    if (l1>l1+ls/10.0) break;

    integrate_one_step(ybeg,l1,ls,yend,k);

    for (i=0; i<NUM_EQN; i++) {
      ybeg[i]=yend[i];
    }

    fx[k+1] = yend[0];
    fy[k+1] = yend[1];
    m[k+1]  = yend[2];
    th[k+1][1] = yend[3];
    Total_weight = yend[4];

    x_pos[k+1][1] = x_pos[k][1] + ls*cos(th[k][1]);
    y_pos[k+1][1] = y_pos[k][1] + ls*sin(th[k][1]);
```

```
      k++;
    }

  }

  /* ------------------------------------------------------------ */
  /*     Integrate one step using Runge-Kutta method...          */
  /* ------------------------------------------------------------ */

  integrate_one_step(ybeg,x,h,yend,k)

       double ybeg[];
       double x;
       double h;
       double yend[];
       int k;

  {
    double fcn1[NUM_EQN];
    double fcn2[NUM_EQN];
    double fcn3[NUM_EQN];
    double fcn4[NUM_EQN];
    double xtmp;
    double ytmp[NUM_EQN];

    int i;

    xtmp = x-h;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i];
      derivs(xtmp,ytmp,fcn1,k);
    }
    xtmp = x-h/2.0;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn1[i]/2.0;
      derivs(xtmp,ytmp,fcn2,k);
    }
    xtmp = x-h/2.0;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn2[i]/2.0;
      derivs(xtmp,ytmp,fcn3,k);
    }
    xtmp = x;
    for (i=0 ; i<NUM_EQN ; i++) {
      ytmp[i] = ybeg[i] + h * fcn3[i];
      derivs(xtmp,ytmp,fcn4,k);
    }

    for (i=0 ; i<NUM_EQN ; i++) {
      yend[i] = ybeg[i] + h*(fcn1[i] + 2.0*fcn2[i] + 2.0*fcn3[i] +
fcn4[i])/6.0;
    }

  }

  /* ------------------------------------------------------------ */
```

```
/*    Evaluate the functions to be integrated...            */
/* ------------------------------------------------------------ */

derivs(x,y,fcn,k)

     double x;
     double y[];
     double fcn[];
     int k;

/* Four coupled first-order equations from initial ODE */
/* eg. fcn[0] = dy[0]/ds */

{
  double B1(),B2();

  fcn[0]=w*ax[k]-B1()*vx[k][1];
                                  /* d(fx)/ds=0.0 */
  fcn[1]= -1.0*w+w*ay[k]-B1()*vy[k][1];
                                  /* d(fy)/ds=-w  */

  fcn[2]=(y[0])*sin(y[3]) - y[1]*cos(y[3]) + I*ath[k]-B2()*omega[k][1];
                                  /* d(m)ds=... */

  fcn[3]=(y[2]+add_mom[k])/(E*I);
                                  /* d(th)/ds=m/EI */

  fcn[4]=w*x;
}



/*-------------------------------------------------------------------*/
calculate_inverse(a,r)

     double a[N+1][N+1];
     double r[N+1][N+1];
{
  double l[N+1][N+1],u[N+1][N+1];
  int i,j;
  int indx[N+1];
  double b[N+1];
  double d;
  double y[N+1][N+1];
  double col[N+1];
  double x[N+1];
  /*float r[N+1][N+1];*/

  ludcmp(a,N,indx,&d);


/* Decompose LU matrix */

  for (i=0 ; i<=N ; i++) {
    for (j=0 ; j<=N ; j++) {
      u[i][j]=a[i][j];
      l[i][j]=a[i][j];
    }
```

```
  }

  for (i=0 ; i<=N ; i++) {
    l[i][i]=1.0;
    for (j=i+1 ; j<=N ; j++) l[i][j]=0.0;
  }

  for (j=0 ; j<=N-1 ; j++) {
    for (i=j+1 ; i<=N ; i++) u[i][j]=0.0;
  }



  for(j=0 ; j<=N ; j++) {

    for (i=0 ; i<=N ; i++) col[i]=0.0;
    col[j]=1.0;

    lubksb(l,u,N,indx,col,x);

    for(i=0 ; i<=N ; i++) r[j][i]=x[i];
  }


}



ludcmp(a,n,indx,d)

     int n,indx[N+1];
     double a[N+1][N+1],*d;
{
  int i,imax,j,k;
  float big,dum,sum,temp;
  double vv[N+1];


  *d=1.0;
  for (i=0;i<=n;i++) {
    big=0.0;
    for (j=0;j<=n;j++)
      if ((temp=fabs(a[i][j])) > big) big=temp;
    if (big == 0.0) printf("SINGULAR\n");
    vv[i]=1.0/big;
  }


  for (j=0;j<=n;j++) {
    for (i=0;i<j;i++) {
      sum=a[i][j];
      for (k=0;k<i;k++) sum -= a[i][k]*a[k][j];
      a[i][j]=sum;
    }
    big=0.0;
    for (i=j;i<=n;i++) {
      sum=a[i][j];
      for (k=0;k<j;k++)
```

```
        sum -= a[i][k]*a[k][j];
        a[i][j]=sum;
        if ( (dum=vv[i]*fabs(sum)) >= big) {
        big=dum;
        imax=i;
        }
      }
      if (j != imax) {
        for (k=i;k<=n;k++) {
        dum=a[imax][k];
        a[imax][k]=a[j][k];
        a[j][k]=dum;
        }
        *d = -(*d);
        vv[imax]=vv[j];
      }
      indx[j]=imax;
      if (a[j][j] == 0.0) a[j][j]=TINY;
      if (j != n) {
        dum=1.0/(a[j][j]);
        for (i=j+1;i<=n;i++) a[i][j] *= dum;
      }
    }
}


/* Back-substitute LU matrix */

lubksb(l,u,n,indx,b,x)
     double l[N+1][N+1],u[N+1][N+1],b[],x[N+1];
     int n;

{
  int i,ii=0,jj=0,j;
  double sum;
  double y[N+1];

  for (i=0 ; i<=n ; i++)  {
    sum=0.0;
    if (i==0) ii=0;
    if (ii!=0) {
      for (j=0 ; j<=i-1 ; j++) sum += y[j]*l[i][j];
      y[i]=b[i]-sum;
    }
    else {
      y[0]=b[0];
      ii=1; }
  }

  for (i=n ; i>=0 ; i--) {
    sum=y[i];
    if (i==n) jj=0;
    if (jj!=0) {
      for (j=i+1 ; j<=n ; j++) sum -= u[i][j]*x[j];
      x[i]=sum/u[i][i];
    }
    else { x[n]=sum/u[n][n];
         jj=1; }
```

```
    }
}


/* Multiply vector to matrix */

multiply(inv_a,s,dv)

     double inv_a[N+1][N+1],s[N+1],dv[N+1];

{
  int i,j;
  float tempo;
  for (i=0 ; i<=N ;i++) dv[i]=0.0;
  for(i=0 ; i<=N ; i++) {
    for(j=0 ; j<=N ; j++)
      dv[i] += (inv_a[j][i]*s[j]);
  }
}



simulation(x_roll,y_roll,r_roll,x_draw,y_draw,ang_count,clear_flag)

     double x_roll,y_roll,r_roll;      /* roller coords. and radius   */
     double x_draw[2][MAX_NODES],y_draw[2][MAX_NODES];
     int ang_count;                    /* Counter for angle (graph) */
     int clear_flag;                   /* 0 to clear window, 1 to keep */
{

  int xx1,yy1,xx2,yy2;                 /* x,y coords for graphics */
  int xc,yc;
  int rr;                          /* radius of roller */
  int th_deg;                           /* angle in degrees */
  int gx1,gx2,gy1,gy2;                  /* graphics coordinates for graph
*/
  int max_ang_increments;               /* angluar increments for graph */
  int length_of_increment;              /* length (in pixels) of angluar
increment */
  float xf,yf,rf;                  /* Data from input */


/* Other variables */

  int j;
  int i,q,q2;
  int con_node;                         /* Contact node number */
  float choice;                         /* Input values */
  int remlength;                   /* Remaining length for rot'n */


  max_ang_increments = (3.4 - th_seg)/del_th;
  length_of_increment = 450/max_ang_increments;

  if(!clear_flag){
```

```
      XClearWindow(display,sim_window);
      XFlush(display);
    }

    XClearWindow(display,data_window);

    load_font(&font_info);

    draw_text(data_window,gc,font_info,hint.width,hint.height);

    /* Draw Floor Bill */
    XDrawLine(display,sim_window,gc,50,257,500,257);
    XFlush(display);

    /* Draw buckled Bill - left part first  (BOTH ROTATION AND STAGE 2) */
    for (j=0 ; j<2 ; j++) {

      for (i=0 ; i<NUMBER_OF_NODES-1 ; i++) {
        xx1=3214*x_draw[j][i];
        xx2=3214*x_draw[j][i+1];
        yy1=3214*y_draw[j][i];
        yy2=3214*y_draw[j][i+1];
        XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xx2,
              255-yy2);

      }

    }


    /* ONLY FOR STAGE 2 */
    if (flag) {
      q2 = 3214*l_rem;
      XDrawLine(display,sim_window,gc,50+xx2,255-yy2,50+xx2+q2,255-yy2);

    /* Now draw graph of Ratio vs. theta in graph window */

      XDrawLine(display,graph_window,gc,50,110,500,110); /* x-axis (theta)
*/
      XDrawLine(display,graph_window,gc,50,30,50,190);   /* y-axis (Ratio)
*/
      XDrawLine(display,graph_window,gc,45,70,500,70);   /* R=1 line */

      gx1 = 50 + length_of_increment*(ang_count-1);
      gx2 = 50 + length_of_increment*ang_count;

      if (ang_count == 1) gy1 = 110;     /* If this is the first data of
ratio */
                                  /* use origin for first point */
      if (ang_count != 1) gy1 = 110 - old_ratio*40;

      gy2 = 110 - ratio*40;               /* Note: I assume that +y axis
goes up */
                                  /* to a ratio of 2 (maximum) */

      XDrawLine(display,graph_window,gc,gx1,gy1,gx2,gy2);

      XFlush(display);
```

```
    }

    /* ONLY FOR ROTATION */
    if (!flag) {
      /* Draw Flat Part (tail) */
      remlength = 3214*(12 + xc0) + 50;
      q = xc0*3214;
      XDrawLine(display,sim_window,gc,50+q,255,remlength,255);

    }

        /* Draw Roller */
    xx1 = 3214*x_roll;              /* Coordinates of centre */
    yy1 = 3214*y_roll;
        /* N O T E !! :    NOT the coordinates of the centre to be given
*/
    xx2 = 3214*r_roll;

    xc = xx1 - xx2*sin(theta);
    yc = yy1 - xx2*cos(theta);

    XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xc,255-yc);

/* theta1 is theta plus the angle of the segment of the roller */
    xc = xx1 - xx2*sin(theta1);
    yc = yy1 - xx2*cos(theta1);

    th_deg = 1*(theta*180/PI);
    XDrawLine(display,sim_window,gc,50+xx1,255-yy1,50+xc,255-yc);

/* XDrawArc draws the arc between the straight segements of the roller.
The arguments are: */
/* ..... , angle from vertical of first clock-wise segment, -(angular
distance of 2nd straight side) */
    XDrawArc(display,sim_window,gc,50+xx1-xx2,255-yy1-
xx2,2*xx2,2*xx2,(270-th_deg)*64,-(100)*64);



        XFlush(display);

}
/*closes simulation */

load_font(font_info)
      XFontStruct **font_info;
{
    char *fontname = "9x15";

    if((*font_info = XLoadQueryFont(display,fontname)) == NULL)

      {
        (void) fprintf( stderr,"Basic: Cannot open 9x15 font \n");
        exit(-1);
      }
}
```

```
draw_text(data_window,gc,font_info,win_width,win_height)

     Window data_window;
     GC gc;
     XFontStruct *font_info;
     unsigned int win_width, win_height;
{
  int y = 50;
  char *string1 = "Ratio of tangential to normal forces at contact";
  char *string2 = "This is another test for the ratio";
  int len1, len2, len3;
  int width1, width2;
  char cd_ratio[50], cd_lm[50], cd_rm[50];
  int font_height;
  int initial_y_offset, x_offset;



  len1 = strlen(string1);
  len2 = strlen(string2);


  width1 = XTextWidth(font_info, string1,len1);
  width2 = XTextWidth(font_info, string2,len2);

/*  XDrawString(display,data_window,gc,(win_width -
width1)/2,y,string1,len1); */
/*  XDrawString(display,data_window,gc,(win_width -
width2)/2,(int)(win_height - 35),string2,len2);*/


  (void) sprintf(cd_ratio, "RATIO (Ft/Fn):  %f ", ratio_display);
  (void) sprintf(cd_lm,  "Left-part moment:   %f",left_mom);
  (void) sprintf(cd_rm,  "Right-part moment:  %f",right_mom);

  font_height = font_info->max_bounds.ascent + font_info-
>max_bounds.descent;

  initial_y_offset = win_height/2 - font_height - font_info-
>max_bounds.descent;

  x_offset = (int) win_width/4;

  len1 = strlen(cd_ratio);
  len2 = strlen(cd_lm);
  len3 = strlen(cd_rm);


  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset,
cd_ratio,len1);

  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset +
5*font_height, cd_lm,len2);
  XDrawString(display, data_window,gc,x_offset, (int) initial_y_offset +
6*font_height, cd_rm,len3);

}
```

```
/* Damping coefficient function - for x and y directions */
/* Aerodynamic model should be input here */

double B1()

{

/* For now, just return a constant */
  return (0.001);

}

/* Damping coefficient function - for rotation */
/* Aerodynamic model should be input here */

double B2()

{

/* For now, just return a constant */
  return (0.001);

}


graphix_init(argc,argv)

     int argc;
     char **argv;

{
/* INITIALIZE GRAPHICS */

  display = XOpenDisplay("");          /* Screen Initialization */
  screen = DefaultScreen(display);
  cmap = XDefaultColormap(display,screen);


  Magenta.red=255*factor;
  Magenta.green=0*factor;
  Magenta.blue=255*factor;

  if(XAllocColor(display,cmap,&Magenta)==0) fail=1;

  DarkSlateGrey.red=47*factor;
  DarkSlateGrey.green=79*factor;
  DarkSlateGrey.blue=79*factor;

  if(XAllocColor(display,cmap,&DarkSlateGrey)==0) fail=1;

  White.red=255*factor;
  White.green=255*factor;
  White.blue=255*factor;
```

```
   if(XAllocColor(display,cmap,&White)==0) fail=1;

     /* Determine if color or black/white to be used */

   fail=0;

   if(fail) {
     background = WhitePixel(display,screen);
     foreground = BlackPixel(display,screen);

   }

   else{
     background = DarkSlateGrey.pixel;
     /*      background = White.pixel;*/
     foreground = Magenta.pixel;
   }

     /* Specify window position and size for first window (simulation) */

   hint.x = 10;
   hint.y = 30;
   hint.width = XDisplayWidth(display,screen)/2;
   hint.height = XDisplayHeight(display,screen)/2;
   hint.flags = USPosition | USSize;

   /* Window creation */
   sim_window = XCreateSimpleWindow (display, DefaultRootWindow(display),
hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);


XSetStandardProperties(display,sim_window,"simulation","simulation",None
,
                     argv,argc,&hint);


     /* Specify window position and size for second window (data) */

   hint.x = 600;
   hint.y = 30;
   hint.width = XDisplayWidth(display,screen)/4;
   hint.height = XDisplayHeight(display,screen)/4;
   hint.flags = USPosition | USSize;


   data_window = XCreateSimpleWindow (display,
DefaultRootWindow(display), hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);

   XSetStandardProperties(display,data_window,"data","data",None,
                     argv,argc,&hint);
```

```
    /* Specify window position and size for third window (graphics) */

   hint.x = 10;
   hint.y = 550;
   hint.width = XDisplayWidth(display,screen)/2;
   hint.height = XDisplayHeight(display,screen)/4;
   hint.flags = USPosition | USSize;


    /* Window creation */
  graph_window = XCreateSimpleWindow (display,
DefaultRootWindow(display), hint.x,
                                hint.y, hint.width, hint.height, 0,
foreground,
                                background);

  XSetStandardProperties(display,graph_window,"R / th
Graph","graph",None,
                    argv,argc,&hint);



/* Restore these so that data_window doesn't get screwed-up */
  hint.width = XDisplayWidth(display,screen)/4;
  hint.height = XDisplayHeight(display,screen)/4;

    /* GC creation and initialization */

  gc = XCreateGC(display,sim_window,0,0);
  gc = XCreateGC(display,data_window,0,0);
  gc = XCreateGC(display,graph_window,0,0);

  XSetBackground(display,gc,background);
  XSetForeground(display,gc,foreground);

/* Load and set font */

  f = XLoadFont(display,"9x15");
  if (!f) {
    fprintf("Couldn't load font %s.  \n",f);
    exit(-1);
  }

/* Window mapping */

  XMapRaised (display,sim_window);
  XMapRaised (display,data_window);
  XMapRaised (display,graph_window);

  XFlush(display);
  sleep(2);

/* INITIALIZE GRAPHICS END */

}
```