Human Hip Joint Mechanics - An Investigation into the Effects of Femoral Head Endoprosthetic Replacements using *In Vivo* and *In Vitro* Pressure Data

by

Kjirste Lynn Carlson

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

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1993

Author ....[illegible]..........
Department of Mechanical Engineering
May 21, 1993

Certified by ....
Robert W. Mann
Professor Emeritus
Thesis Supervisor

Accepted by ....[illegible]..........
Ain A. Sonin
Chairman, Departmental Committee on Graduate Students

Human Hip Joint Mechanics - An Investigation into the Effects of Femoral Head Endoprosthetic Replacements using *In Vivo* and *In Vitro* Pressure Data

by

Kjirste Lynn Carlson

Submitted to the Department of Mechanical Engineering
on May 21, 1993, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Abstract

The development and implantation of two pressure-instrumented femoral head prostheses and the subsequent posthumous recovery of one of the prostheses provide an unique opportunity for study, both *in vivo* and *in vitro*, of the human hip joint. *In vivo* data, including kinematics and external forces coordinated with the local cartilage pressures were acquired over a period of 5 years beginning in June 1984 from the first prosthesis, and thus far for a year from a second prosthesis, implanted in December 1991. Comparison of the data from the two subjects provide information on the effect of small mismatches (< 1 mm) in prosthesis fit and on the changes cartilage undergoes when in contact with a prosthetic femoral head for extended periods of time *in vivo.* Data from these two subjects also provides valuable information on the magnitude and location of maximum stresses in the hip joint, the timing of pressure changes, on the presence of muscle co-contraction forces during some movements, and on the effects of rehabilitation exercises.

The acetabular location of high pressure regions in general supports Wolff's law on bone response to loading; the highest stresses on the pelvis corresponded to the most dense bone. Location and size differences in the region over which high stresses were measured for the two prostheses can be attributed to the slightly over-sized and slightly under-sized fit of the prostheses; regions of high pressure were more localized for the under-sized prosthesis in analogy to Hertz stresses. Changes in the distribution of high pressures on the acetabulum were found to occur over the implantation period for both subjects. The location and distribution of stresses also varied with the movement performed.

Pressure measurements from the two subjects were normalized by body weight over height squared. Maximum normalized pressures were found to change over the implantation period, in a similar manner, for both subjects for the post-operative year. Maximum normalized pressures were usually found to be slightly higher for the slightly under-sized prosthesis, however the maximum pressures and maximum normalized pressures covered a similar range for the two subjects, despite significant differences in their morphology, mobility and coordination. The movements for which the highest normalized pressures were measured were different for the two subjects; rising from a chair resulted in the highest pressure measurements from the first subject while climbing and descending stairs produced higher measurements for the second subject. The highest pressures measured from the second subject were during jumping off a step, a movement not performed by the first subject. The normalized maximum pressures during gait are in the range of 20-25 (MPa/(weight/ht^2)) for both subjects during the first year post-operative.

Thesis Supervisor: Robert W. Mann
Title: Professor Emeritus

Acknowledgements

This thesis has taken longer to complete than many and as a result the number of people who have helped along the way is greater than the number of pages in this document. I wish I could thank each one appropriately here but in the interest of conserving paper and time I can only mention a few people.

The work done for this thesis would not have been possible without the support and encouragement of Professor Robert Mann. His vision has guided the work done on this project through a sucession of students over a period of nearly 30 years. I feel extremely fortunate to have been in the Newman lab for Biomechanics and to have worked with Professor Mann on a part of the "hip project". I have a great deal of respect for his work in assembling and motivating students, for his skills as an educator as well as a researcher.

I also want to thank the folks in the lab in general, the characters change over the course of years, but it has been a wonderful place to work throughout the time I've spent here. I believe that the Newman lab is possibly the best working environment for graduate students at M.I.T., through a combination of the people in it and the vision and style of the faculty members associated with the lab.

Funding for this research has come in part from the National Institute on Disability and Rehabilitation Research, grant REC-H133E80024. Many thanks also to the Whitaker Foundation, which contributed to the support of this research.

This thesis might never have been completed with out the help of my wonderful boyfriend, John Morrell, who helped me enormously during the difficult final phases this year. I hope I can be as supportive and helpful in return someday.

Finally, I want to thank my parents, Nels and LaVerne Carlson, for believing in me all along.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Goals

The human hip joint has been extensively studied both analytically and through experiment. The joint itself is spherical, a ball and socket, and simpler mechanically than many other joints in the human body [Figure 1-1]. Despite its relatively simple geometry and kinematics the lifetime for replacement prostheses is only 5-10 years for a replacement of only the femoral head with an endoprosthesis, or 10-15 years for total joint replacement (THR) at the hip. [14, 24] Every year over 200,000 hip joint replacement operations are performed, of these, 40% are replacement of the femoral head alone. Within 5-10 years these prostheses require revision surgery. This short lifetime suggests that our understanding of the natural history of the joint following surgery is lacking in understanding and that improvements in replacement procedure and hardware are possible. The research presented in this thesis has two aims: (1) to better understand the mechanics of the human hip joint and its components; (2) to provide information to improve the success of the endoprosthetic hip joint.

Towards these goals, as part of a research investigation of synovial joint biomechanics and osteoarthritis, a pressure-instrumented endoprosthesis was developed in the Newman Laboratory for Biomechanics at MIT. Two of these prostheses were implanted in consenting human subjects, making possible extended studies of the stresses on cartilage, and the pathophysiological response, in the human hip joint.

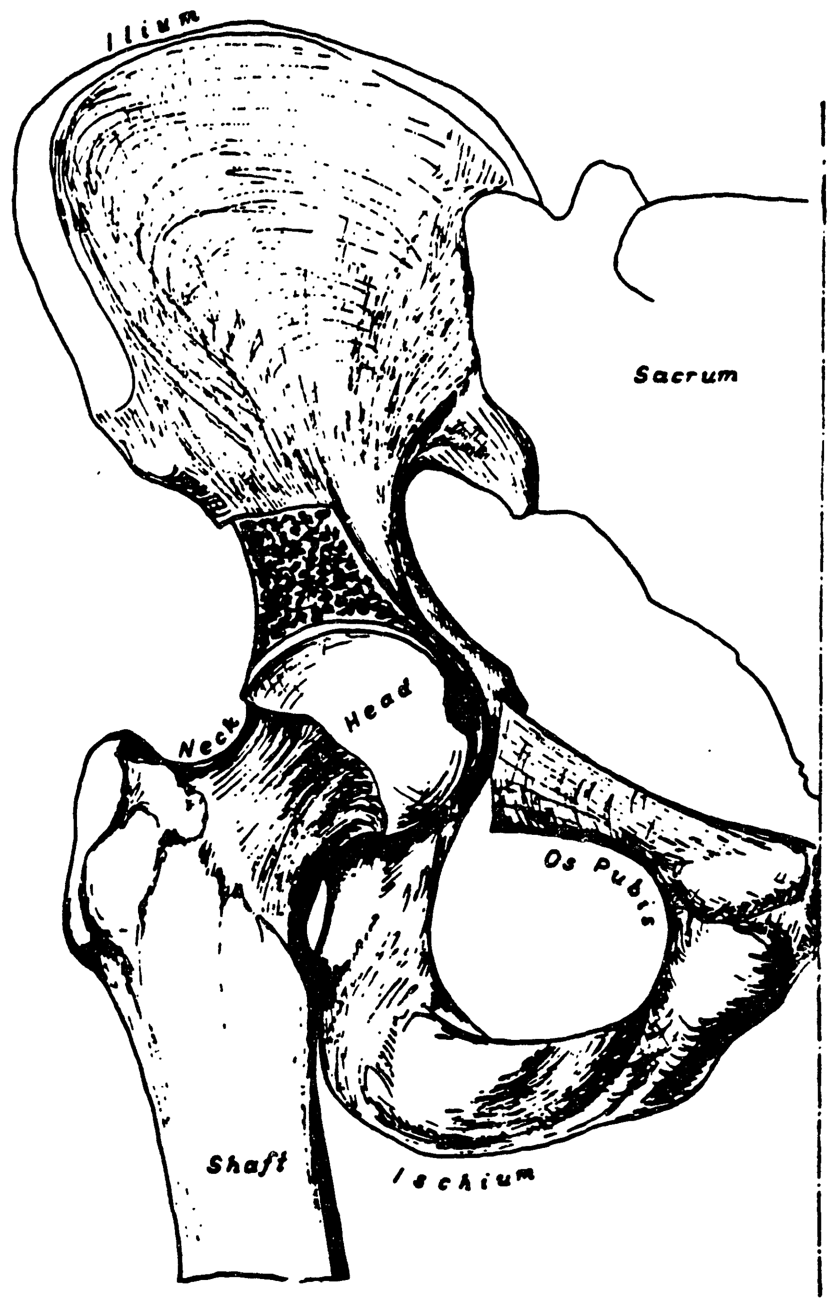


Figure 1-1: Human Hip Joint: Frontal Cross-section [from Tepic [22]]

The two implanted prostheses were each very slightly different in size from the natural femoral heads which they were replacing, one being slightly oversized, the other slightly undersized in diameter. This difference was not planned, rather a consequence of the preordained size of the implant and the difficulty, during surgery, of sizing the natural acetabulum, but these circumstances presented the opportunity to evaluate the effect of small misfits in prosthesis size. Investigation into the effects of prosthesis misfit on the stresses in the human hip joint *in vivo* and on the physiological adaptation process that occurs to the human hip joint following implantation of an endoprosthesis is the focus of this thesis. This information may help elucidate how mechanical factors lead to osteoarthritis and identify the contribution of muscular co-contraction forces in producing high joint pressures, as well as explicating reasons for failure of endoprosthetic joints.

The measurements, analyses, and comparisons herein were performed in part to investigate the hypothesis that even a small misfit may affect the stress distribution in the human hip joint and the performance of the prosthetic joint. For the two subjects who have had instrumented femoral head endoprostheses, the prosthesis implanted in the first subject was 0.6 mm over-sized in diameter relative to the natural femoral head, and the prosthesis implanted in the second subject was 0.7 mm under-sized in diameter. Previous *in vitro* studies by our group, at a time when commercial endoprostheses were produced in 1/8" [3 mm] increments in diameter have shown that mismatches of 1 or 2 mm in femoral head diameter can produce profound differences in the loading distribution of the endoprosthesis on the cartilage in the joint. [20] This information led to the development, testing, and then wide use of sizing gages to determine the proper prosthesis size to implant and to the availability of more closely spaced prosthesis sizes. Femoral head sizes are available now in 1 mm increments, which means that many people are fitted with a prosthesis that is .5 mm too large or too small. No *in vivo* data on the effect of this small a mismatch had ever been available. Orthopedic surgeons sometimes operate under the premise that if a mismatch is unavoidable, implantation of an oversized prosthesis is preferable to implantation of an undersized prosthesis. This rule appeals to common sense, the area of contact between the femoral head and cartilage will be larger and therefore the stresses should be more evenly distributed, however there has been no experimental proof or disproof. The chance occurrences which led to the implantation of two instrumented prostheses, each slightly mismatched to the particular subject in opposing directions, have produced a unique opportunity to investigate this hypothesis.

Data obtained from the implanted instrumented prostheses over long periods of time provide information on the adaptation process of the natural cartilage of the acetabulum to the rigid metal endoprosthesis ball. Initially the cartilage presumably adapts to the different size of the replacement femoral head. More global remodelling may take place as the prosthesis performs as a part of the joint for longer periods of time; this frequently culminates in the protrusion of the prosthetic femoral head through the acetabular cartilage and the underlying bone. A common failure mode of femoral head replacement prostheses, the reasons for this occurance are not well understood. While this type of failure has not occurred with either subject, the acetabular cartilage of the first subject, who succumbed to an unrelated pathology, did show a substantial difference between the time of implantation and after 5 years of contact with the endoprosthesis. This study will shed light both on what changes occurred to the cartilage and on why those changes took place.

Of general concern to all endoprosthetic procedures, the difference in long-term success of slightly oversized versus undersized prostheses is not known at all. Although the second prosthesis has been implanted for a only year thus far, some indication of what may occur in the future is evident in the data obtained thus far. What is clear is that over the long term the cartilage in constant contact with a prosthetic femoral head experiences degradation of a different type than that found in a natural joint.

The initial thrust of this thesis was the comparison of joint forces *in vivo* to estimates of joint forces made from measurements of external forces to assess the importance of muscle co-contraction forces. In both subjects the maximum pressures recorded during activities such as rising from a chair or ascending stairs was considerably higher than would be expected from the dynamics alone. Concurrent recording of electromyographic signals from agonist and antagonist muscle pairs about the hip joint confirmed that muscular co-contraction was augmenting the dynamic and inertial forces at the hip. Attempts to estimate the hip force vector from the pressure data were unsatisfactory due to the paucity of reporting transducers, leading to the decision to design a new prosthesis which measures force directly. This research will be reported in future efforts of our group. As part of this thesis much effort was directed at assessing the effect of impact forces at the foot on the loading in the hip joint. However it proved difficult to obtain consistent data for impact loading tests, with synchronization of external kinematic and dynamic variables and the internally measured pressures. This will also be addressed in the future with the new prosthesis design.

1.2 Investigation Procedure

Pressure data from two subjects are compared in this thesis, primarily the comparison of maximum pressures measured in each acetabular region. Since the pressure transducers are in the metal femoral head, while of interest is where they are on acetabular cartilage, kinematic variables in the relationship between femur and pelvis are employed throughout to make this transition. Comparison of data between the two subjects is done through normalization of the pressures using subject weight and height and representation of data as in a right hip orientation. (One prosthesis was implanted in a right hip, the other in a left hip.) The change in pressures over the implantation period for a standardized test protocol provides information on the effect of joint adaptation to the prosthesis.

The first pressure-instrumented prosthesis was implanted in June 1984, in a 73-year-old woman, who was 1.68 m tall and weighed 663 N. The data reported herein was acquired during testing every 3 to 6 months for the next 5 years. In late 1989 she died of an unrelated cause. Arrangements had been for donation of the prosthesis and the matching hemipelvis. The second prosthesis was implanted in December 1991 when a suitable subject was found, in this case an 82-year-old man, who was 1.6 m tall and weighed 529 N. Data has been acquired from this subject for a year as reported herein, and data continues to be taken every few months.

During the immediate post-operative phase of patient management primarily pressure data is gathered, supplemented at times with external measurements of leg angle or force exerted by the leg against a force transducer. From 10 days post-surgical the subject is capable of coming to the Biomotion Laboratory at Massachusestts General Hospital (MGH); where kinematic and forceplate data can be synchronized with the pressure data. Sets of synchronized pressure, kinematic, and forceplate data exist for 5 years of test sessions with the first subject, and now for three test sessions with the second subject.

In addition to investigating the changes that occurred over time and the differences between the two prostheses, pressures were also compared for changes in activity level, for instance, walking at 60 beats per minute (BPM), walking at 120 BPM, unpaced walking, and walking in place. Forceplate information is used on data comparison, primarily the timing of pressure changes vs. that of rise of external force. Additionally, pressures measured *in vitro* on the excised hemipelvis and retrieved prosthesis in the MIT hip simulator were compared to those previously obtained *in vivo* for similar kinematic and loading configurations. Ultrasound measurements of the acetabular cartilage region of the explant were made by another researcher [26] and these data were used to compare acetabular thickness to regional maximum pressures.

1.3 Thesis Organization

Chapter 2 presents relevant aspects of the history of the development of the pressure instrumented prostheses and discusses the results of other researchers in this field. Chapter 3 describes the equipment used to take data, while Chapter 4 discusses the processing of data. Chapter 5 presents results for selected tests organized by movement and by subject. A discussion of the trends observed in these results and their implications is presented in Chapter 6. Detailed information on transducer calibration and on the location of transducers in the prostheses are presented in the appendices, as is information on the computer programs for data acquisition programs and test equipment circuitry.

Chapter 2

Background

2.1 Instrumented Prosthesis History

Experimental and analytical study of the human hip joint has been underway at MIT in the Newman Laboratory since 1966. At that time C.E. Carlson began design of a pressure instrumented femoral head prosthesis for measurement of the loading in the human hip joint. [6, 17] The currently implanted instrumented prostheses are very similar to his prototype, although several aspects were altered in response to results obtained from *in vitro* testing of the original prostheses. Information about the pressure-instrumented prosthesis structure and data output essential to this thesis are in Chapters 3 and 4 and in Appendices A, B and C. More information on the design and on the telemetry electronics can be found in [6, 7, 5].

Human implantation of a pressure-instrumented prosthesis was a goal of the project from the outset. This intention dictated some of the initial design parameters and necessitated some of the redesign process. Extensive *in vitro* testing, using the same pressure instrumentation, but with hard-wired output, preceded implantation. But no degree of *in vitro* results can substitute for *in vivo* data. Among other considerations *in vivo* data was a priority in light of the fact that the muscle forces acting across a joint in life are not known. Design specifications for the prosthesis included: subject safety; performance characteristics identical or superior to standard prostheses of this type; high data accuracy; adequate high frequency response to faithfully record effects of human motion; minimum patient encumberence during data acquisition; as many pressure transducers as possible given the joint geometry; data acquisition ability for more than 2 years post-implantation.

The prosthesis itself is similar to a Moore endoprosthesis, however at 14 locations on the femoral ball the spherical wall thickness is reduced to a thin diaphragm. Deflection of these diaphragms due to a localized pressure between the ball and acetabular cartilage is measured and output via radio-telemetry. Power is supplied to the electronics in the femoral ball externally through a magnetic power induction link. Calibration of the assembled prostheses (covered in detail in Appendices B and C) indicated that transducer output could be interpreted accurately from 10 transducers in the first implanted prosthesis and from 12 transducers in the more recently implanted prosthesis. Although the number of transducers is as large as possible for the available space, unfortunately they do not report enough data to accurately interpolate pressure contours over the entire surface at any given instant in a test. [9] Thus, *in vivo* data on the force vector during movements has not been possible with this prosthesis design.

2.2 Other Research on Joint Forces and Cartilage Pressures

The most common approach to the problem of estimating joint forces has been analytical, based on experimental kinematic, kinetic, body morphological, and electromyographic data. Some such studies have considered the results of *in vitro* tests. The analytical approach allows consideration of the normal joint; however much information on relevant parameters is lacking and many simplifying assumptions must be made to reduce the problem to a manageable one.

The basic technique is inverse Newtonian analysis; foot-floor forces and moments can be measured with satisfactory accuracy and precision. The foot is then assumed to be a solid object with kinematic variables, positions and rotations measured by some stereographic movement analysis system. Most of these read position data from markers on the body segment. Calibration issues, recording fidelity, 3-D reconstruction algorithms, system noise (enhanced by differentiation to determine velocities and then accelerations), all contaminate the kinematic data to various degrees. Then the mass, center of mass, and several inertias of the limb segment must be estimated by methods ranging from anthromorphic scaling of cadaver experiments to use of computer-tomographic data. With acceleration and mass-inertial estimates, the causative forces and moments can be estimated, for the example here, at the ankle. This analytical procedure is repeated treating the shank as an isolated body, to get estimates of the knee forces and moments, followed by similar treatment of the thigh to arrive at estimates of hip forces and moments.

Given the range of fidelity of inertial and kinematic data and the processing involved it is perhaps not surprising that estimates of hip forces in normal level walking range in the literature by almost a factor of 10. [21, 12, 19] Beyond these uncertainties the analytical procedure cannot include those forces across the joint which result from co-contraction of agonist-antagonist muscle pairs. Motion analyses can only reflect muscle pair imbalances which produce motion, ie. the difference between opposing muscle forces, whereas the joint experiences the force due to the sum of co-contracting muscles.

That co-contraction is commonplace in virtually all movement is becoming increasingly clear both from the hip joint research reported in this thesis as well as from studies that show the movement control areas of the brain and spinal cord control the impedance as well as the motion of anatomical joints. [15]

In vivo results for the force at the human hip joint were obtained by Rydell. [21] This study used a strain-gaged instrumented, modified Austin-Moore endoprosthesis with an over-long neck implanted in two subjects. The only data was taken at one test session 6 months post-operative, at which time the subjects may not have been fully recovered. The brevity of testing was dictated by the means of transmitting data; wires from the strain gages passed directly through an incision in the skin. The maximum joint force recorded was 4.3 times body weight for running. During walking the highest force measured was 3.3 times body weight.

Recently Bergmann et. al. have implanted 3 force instrumented total hip replacements: two in one subject and one in another. [3] These are total hip replacements, the entire natural joint has been replaced by a new metal ball and plastic socket, thus they intrinsically change the natural joint geometry, compared to endoprostheses where only the femoral head is replaced.

In most technical respects the prostheses implanted by Bergmann et al [3] are similar to the ones from which results are reported here. They also inductively power the prostheses and use telemetered output. However, their prostheses are total hip replacements and they measure forces acting at the joint. No kinematic data concurrent with the internal force data is available, except for video of the subject during tests. Their results which are of interest here are that the resultant load direction is always close to the femoral-neck axis and that the load direction is nearly invariable at higher loading and longer time post-op.[2] The peak forces reported during walking were 300% BW. [2] Although they possess a forceplatform and are in a position to concurrently measure external forces and internal force at the joint, no information of this type has been published by this group of researchers.

A number of *in vitro* studies have also been done on the human hip joint. Estimation or measurement of the surface pressure distribution has been more common in these investigations. Prior to 1980 researchers doing these studies estimated pressure at only a few locations using indirect methods and then inferred a uniform or axisymmetric sinusiodal pressure distribution. The first direct *in vitro* pressure measurements were made early in this project by Rushfeldt using an instrumented prosthesis similar to the ones discussed in this thesis in the hip simulator. [20] Results from his *in vitro* studies indicated that local pressures were much higher than previously assumed, and that non-uniform, steep pressure gradients can exist in hip joint articular cartilage.

Similar pressure distributions were then found by Brown in experiments in which recesses were machined into the cartilage surface layer on the femoral head and accept small piezoresistive pressure transducers. [4]

Chapter 3

Apparatus

The equipment used to collect the data reported in this thesis includes two pressure instrumented endoprostheses, the kinematic data acquisition system in the Biomotion Laboratory at Massachusetts General hospital (MGH), a pressure data acquisition system, and the hip simulator at MIT. Information describing each of these subsystems is presented here; other sources can be consulted for further information on the prostheses, the kinematic data acquisition system, and the hip simulator. Additional new equipment was designed and fabricated to allow calibration of the pressure instrumented prostheses; this equipment is discussed in Appendix B.

3.1 Instrumented Prosthesis

3.1.1 Overview

The pressure instrumented prosthesis is essentially a Moore-type femoral head replacement; it takes the place of the natural femoral head and neck and is attached to a shaft which extends into the femoral canal. Figure 3-1 shows the external form of the prosthesis. This type of prosthesis is typically implanted in an individual who has sustained a fracture of the femoral neck and has no degradation of the acetabular cartilage. As a result the joint better replicates the natural human hip joint than does a total hip replacement prosthesis, in which both the ball (femoral head) and the socket (acetabulum) are replaced. Since the acetabulum is natural, the ball location is predetermined. Provided the geometry between the natural femur and the natural femoral head is reproduced, an endoprosthesis maintains the anatomy of the natural joint. By contrast with total hip replacements, the removal of bone stock for the artificial components; and deliberate choices by the surgeon to reduce loading in the artificial joint usually changes the natural geometry of the joint.

The implanted prostheses material is a cobalt chromium alloy (Stellite 21) a biocompatible alloy commonly used in endoprostheses. The stem and integral lower half of the femoral head was cast by Howmedica; the upper hemisphere containing the transducers was hot isostatically pressed from a sintered version of this same material and donated by Zimmer. The pressure transducers consist of 3 mm wells formed in the inner surface of this hemisphere by electron discharge machining. These wells are arranged with one central transducer surrounded by a ring of six, and an outer ring of six or seven transducers (differs in the two prostheses). The arrangement of the transducer wells for each prosthesis is shown in Appendix A. Figure 3-2 indicates the relationship of transducer 1 (the central transducer) to the prosthesis stem.

The mechanical structure of the pressure transducers is shown in Figure 3-3. The thinner hemisphere diaphragm (0.46 mm) of the well is connected by a pin (restrained by a Teflon sleeve) to a strain-gaged, silicon-crystal cantilever beam. Deflection of the diaphragm (0.00028 mm per MPa) is proportional to the difference in pressure between the external and internal sides of the hemisphere wall and is transmitted via the sliding pin to the free end of the cantilever beam. The actual deflection of the much thicker (3 mm) shell is negligible and does not unduly influence the pressure measurement. The natural frequency of the transducer is 12 kHz. The second implanted prosthesis also contains a thermistor in another well in the inner surface of the hemisphere, location shown on the diagram of prosthesis 43 in Appendix A. The well in which the thermistor is placed is similar to the other wells but was not counter-bored.

The transducers are powered and read through a radio-telemetry system contained in the prosthesis. External to the subject and the prosthesis are the power supply and

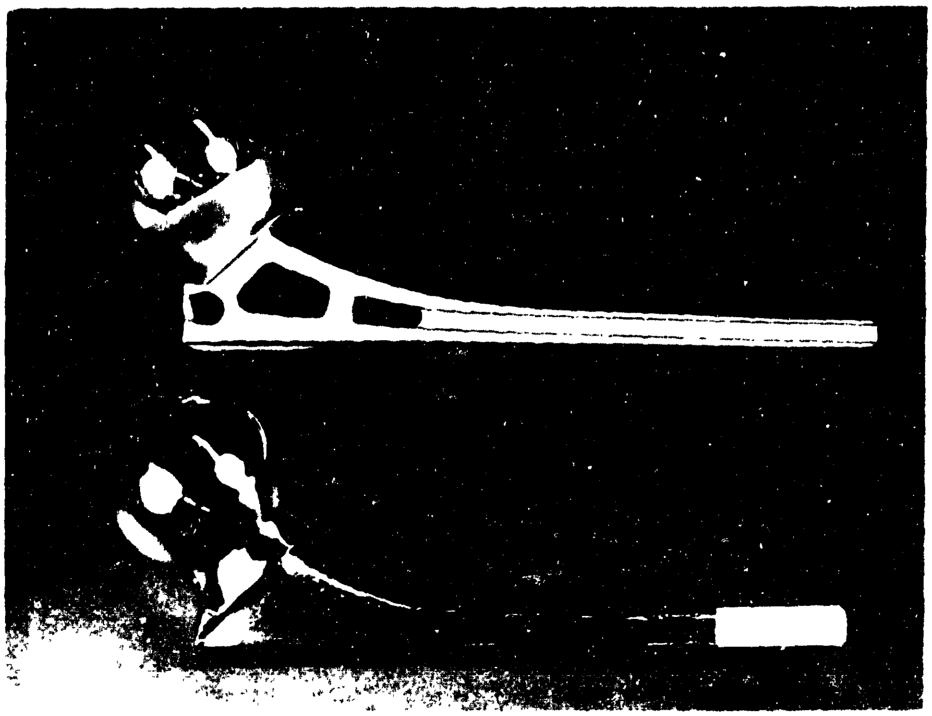


Figure 3-1: Pressure Instrumented Prosthesis: External View [from Carlson [6]]

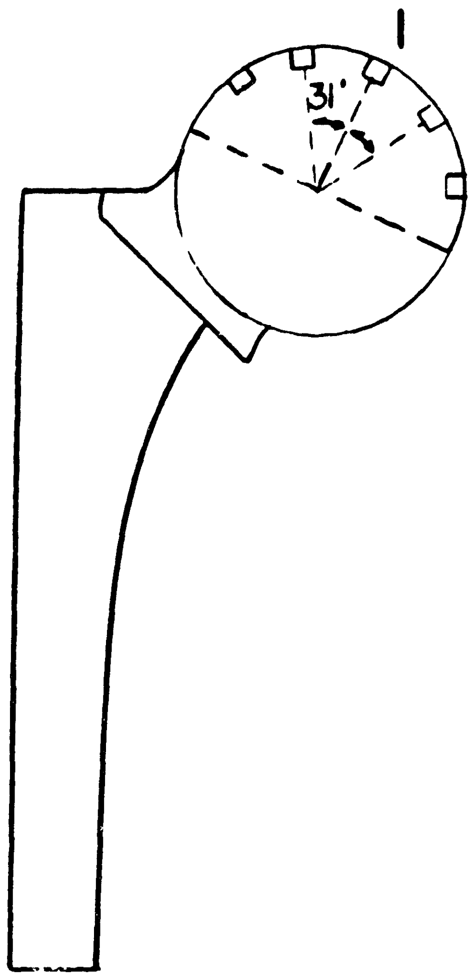


Figure 3-2: Location of central transducer

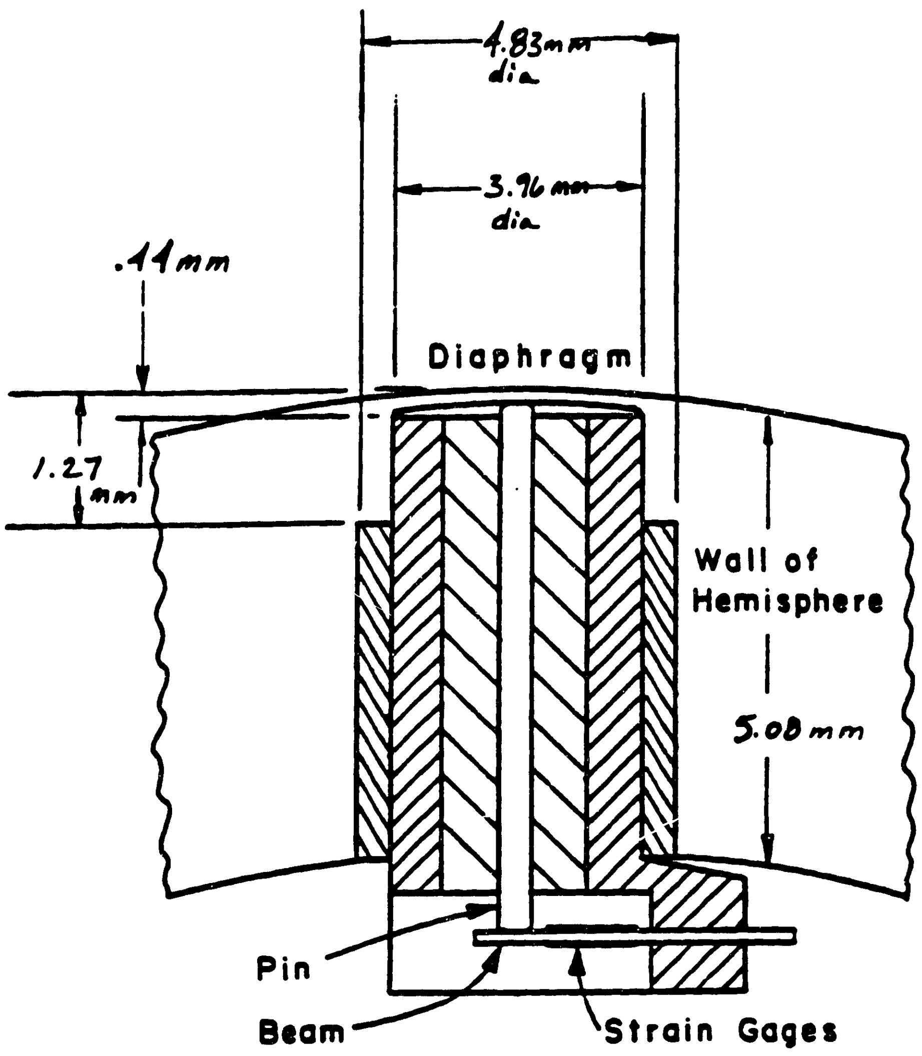


Figure 3-3: Pressure Transducer Mechanical Structure

signal receiver. Electronic circuitry in the ball sequentially powers the transducers and multiplexes the output signal from all transducers. A pair of insulated silver leads run down the stem of the prosthesis to the antenna inside a Teflon cap at the end of the stem. The prostheses are externally powered at 100 kHz. Data are output from the prostheses via radio-telemetry. The transducers are each operated at 250 or 500 Hz (500 Hz in the case of the newer prosthesis), and a single analog output carries information from all transducers serially. In addition this signal carries samples of a known voltage (1 V or 2.5 V) and of 0 V at the start of each sampling sequence. These signals are used for scaling of transducer output magnitude. Chapter 4 contains information on the signal format and interpretation of this signal. The telemetry device itself is more thoroughly described in [6, 7].

The electronic circuitry of the device isolates each transducer output and amplifies the pulse-amplitude-modulated (PAM) signal in an operational amplifier stage. The amplified signal is used to frequency modulate a 100 MHz oscillator. The resulting FM signal is transmitted to the external receiver through the antenna coil wound on a ferrite core at the end of the prosthesis stem.

The signal collection and transmission system of the prosthesis is powered externally through a magnetic power induction link, which eliminates the lifetime restrictions (and biologically incompatible materials) of internal batteries. There are three components active in power transmission: 1) a 100 kHz power oscillator (HP Model 20SAH); 2) a primary coil contained either in a teflon sleeve which fits over the prosthesis stem prior to implantation or a garter that fits over the subject's thigh; 3) a secondary coil inside the Teflon tip of the prosthesis stem, the same coil which acts as the PAM/FM transmitting antenna. Approximately 700mW are delivered to the telemetry system through the power system for the first implanted prosthesis, the newer prosthesis only requires approximately 30 mW.

The hermetically sealed single-unit stucture of the prosthesis ensures protection of the subject from electronic materials. The equator between the stem part and the instrumented hemisphere is welded, after which the interior is sterilized by a baking and nitrous oxide gas process, after which the access hole is plugged and welded. The leads from the ball interior to the stem pass through a glass to metal feed through. The integrity of the prosthesis structure also protects the electronic instrumentation and assures the extended lifetime and accuracy of the transducers. To the subject the prosthesis is indistinguishable from a normal Austin-Moore prosthesis whether powered or not, and it is only powered during data acquisition sessions in the lab. The risk of infection has proven non-existent.

Interpretation and accuracy of the data transmitted from the prosthesis is addressed in Chapter 4 and in appendices B and C. The relationship between transducer output and applied pressure is linear for most of the transducers and the gain of that relationship was found to be invariant with time and temperature. With no applied pressure the transducer output value was found to change over time and to depend on temperature as well. Appendices B and C contains information on how the transducers were calibrated and the results of that calibration process. The essential facts are that the transducers from which results are reported here behaved in a linear fashion and that recalibration *in vivo* of the transducer offset was accomplished by taking data with the subject relaxed and lying down. Previous work indicated that relaxed lying produced transducer outputs as low or lower than those obtained with traction applied to the leg. [8]

There are 14 pressure transducers in the first implanted prosthesis, serial 33, and 13 pressure transducers in the second implanted prosthesis (serial 43). Of these, 4 were found to have non-linear behavior in serial 33 and 1 was found to be non-linear in serial 43. Results from non-linear transducers have not been reported.

Estimation of the misfit of the two instrumented prostheses to their respective recipients was made from measurement of X-ray images of the femoral head. The femoral heads were also passed through a set of sizing gauges during the implantation process to ensure that the correct size prosthesis was implanted. The diameter of each prosthesis was measured with a micrometer.

3.2 Data Acquisition

The data used in this thesis was collected over the period from 1984 to 1993. During this time computer systems were changing rapidly and as a result the data acquisition systems used changed too. One system was used throughout the implantation period for the first prosthesis and another set of systems has been used with the more recently implanted prosthesis. The earlier *in vivo* pressure, kinematic, and forceplate data were all collected by the same computer, a PDP 11/60 running the RSX-11M operating system. Now pressure data is collected on a PC while kinematic and forceplate data is collected on another computer (at first the same PDP 11/60 and now a 486 PC). The information that will be presented in this chapter can be supplimented with [8] for the data acquisition and processing from the first implanted prosthesis and information on the kinematic data acquisition system can be found in [1].

3.2.1 Kinematic data acquisition

All the kinematic data that will be reported in this thesis was collected at the MGH Biomotion Laboratory. This lab uses a version of a data acquisition system, called TRACK, developed in the Newman Laboratory for Biomechanics at MIT by Tetewsky, Conati, Ottenheimer, Antonsson, Mansfield and Lord. [22, 11, 1, 18, 16] Bilateral kinematic data are acquired by two sets of opto-electronic Selspot cameras. Each set of cameras can sense the locations of up to 32 infra-red light emitting diodes in the laboratory reference frame. These LEDs are mounted on rigid arrays of known dimensions attached to the subject's body segments. for now - put back in later The location and rotation of each body segment is computed by the TRACK software system. Kinematic data is stored in processed form as the 3-D location and orientation of coordinate systems fixed to each body segment.

Forceplate data is acquired from two Kistler forceplates. The three components of forceplate data are saved from each forceplate. Forceplate and kinematic data are acquired simultaneously at the same rate, which has typically been 153 Hz.

3.2.2 Pressure data acquisition

Data from the first implanted prosthesis was acquired by the same computer as the kinematic and forceplate data at a rate of 250 Hz per transducer. Many aspects of the data and the processing have not changed between the two prostheses, thus this section will deal explicitly with the newer prosthesis and note differences between the two. For an explanation of the data and processing from the older prosthesis only, refer to [8].

Pressure data are currently acquired on an AT&T 6300+ personal computer, this computer has a 80286 microprocessor but with an XT bus, using an Analog Devices RTI-815F multifunction card for the A/D. Data acquisition uses 2 or 3 A/D channels at 8 kHz for prosthesis 43, 4 kHz for prosthesis 33. This card stores data on the computer via DMA (direct memory access). Figure 3-4 shows the system as set up for data acquisition of 3 channels of data. The interface box between the receiver and the computer performs two additional functions; it buffers the computer from the signal which is used to trigger data sampling when data collection has not yet started and it contains a circuit which allows the computer to collect data at a lower rate. Data collection at a lower frequency was necessary because the DMA controller can only access a 32K block of memory locations and there is insufficient time while running the data acquisition program to transfer data from memory to a file. Appendix E covers the manner in which the interface circuit performs. Of the three channels of analog data, one is the actual transducer outputs, a second contains information required for processing and interpretation of the data, and the third contains auxiliary input (goniometer, force instrumented cane, foot switches, forceplates, TRACK synch signal) which are multiplexed onto a single channel by the auxiliary-multiplexer box.

Data processing and analysis has been carried out on another personal computer, a 486 in the Newman Lab. This computer is also equipped to take data concurrently from a prosthesis and from the hip simulator and to control the hip simulator.

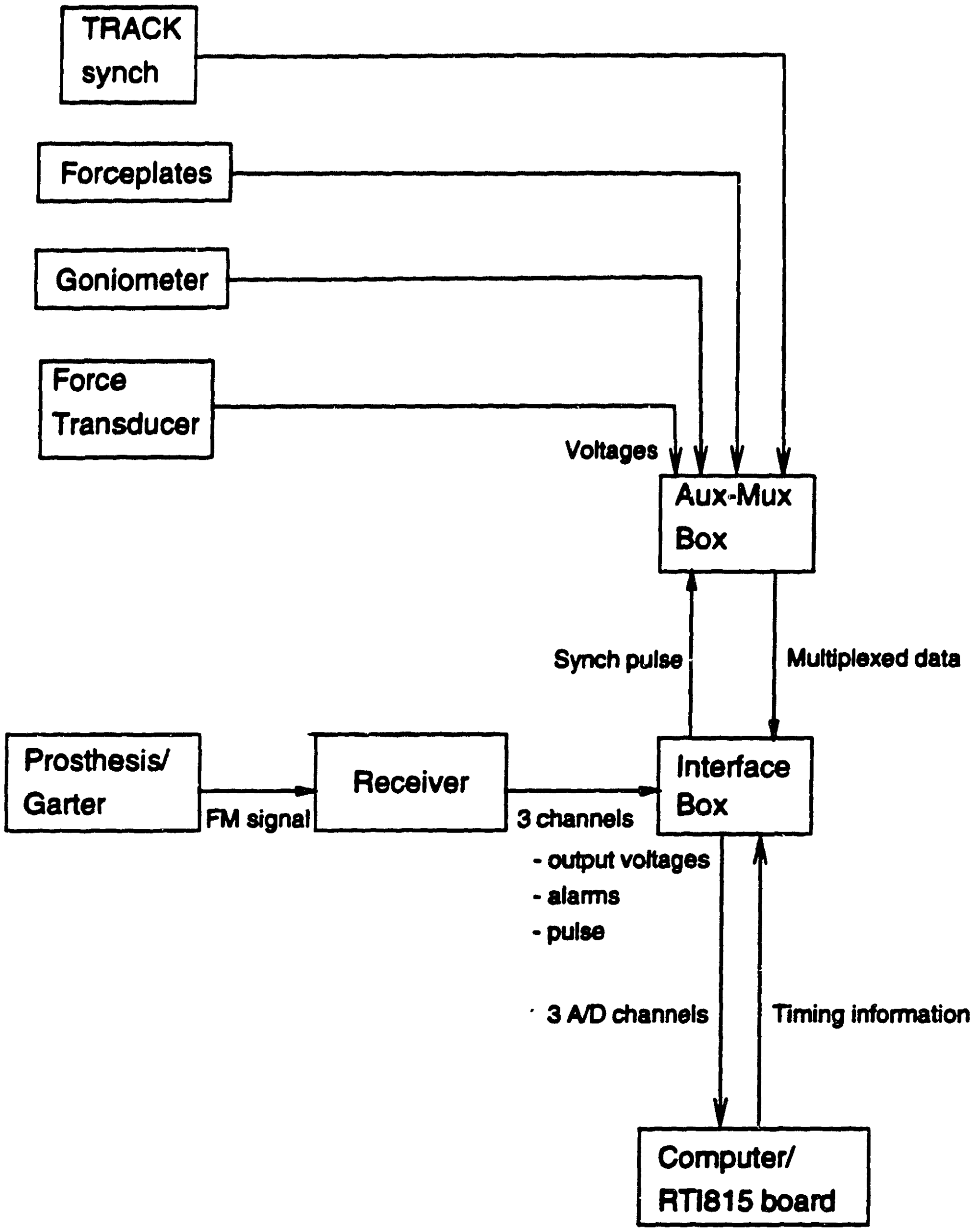


Figure 3-4: Equipment setup for Data Acquisition

3.2.3 Hip Simulator

The hip simulator is a multi-axis electro-hydraulic machine in which a cadaver hip joint can be loaded in anatomical positions. Flexion angle, rotation, and load can be controlled (only two at a time). Measurements of those angles, the load, torque applied, and deflection can be obtained. This machine and the controller for it were developed by Carlson, Rushfeldt and Palmer. [20]

Chapter 4

Processing

4.1 Interpretation of Transducer Output

Data from all transducers are multiplexed onto a single analog signal which is input to an A/D channel on the computer used for data acquisition. Data processing includes correctly demultiplexing and interpreting this data; information on this aspect of processing for prosthesis 33 can also be found in [8]. The procedure determines frame boundaries, identifies the scaling channels, demultiplexes the data, uses the thermal information, and applies transducer calibrations to determine the pressure corresponding to the transducer output signal.

Figure 4-1 shows an example of the data signal collected from prosthesis 43. Also shown is the signal on a second A/D channel used to interpret the data channel. This second channel, referred to as the alarm channel, indicates any conditions which may indicate that the data is bad. The alarm channel also determines the start of each

Table 4.1: Alarm values

Alarms	**Field 1 (V)**	**Field 2 (V)**	Comments
None	-4.95	+0.31	ΔV between fields $\cong$ 5.3 V
$\overline{Tuned}$	-4.3	+0.95	Signal from prosthesis not tuned
$\overline{AGC}$	-3.7	+1.6	Auto-gain for signal not set
$\overline{Tuned}\ \overline{AGC}$	-3.05	+2.3	Not tuned and gain not set
$\overline{Lock}\ \overline{AGC}$	-1.05	+4.2	Phase-lock loop not synchronized

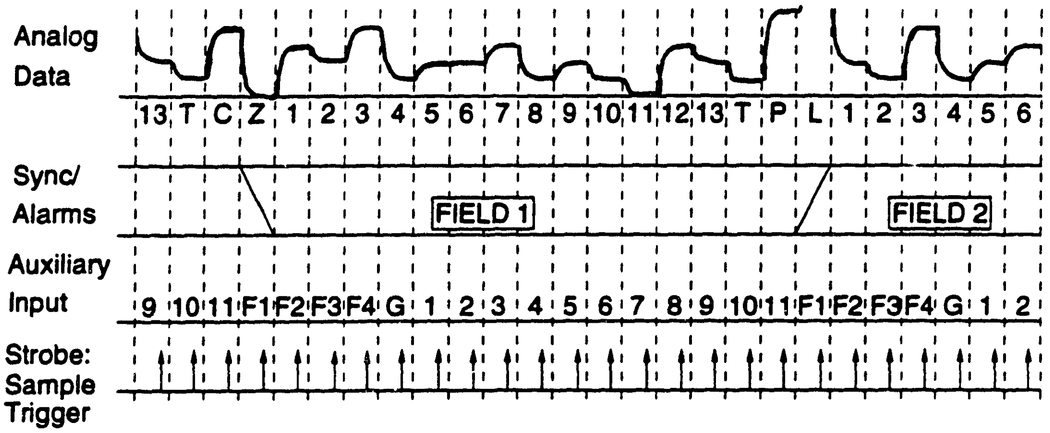
frame of data. Table 4.1 contains the possible alarm values and their interpretation. A third A/D signal is also shown in Figure 4-1; this channel primarily contains data not required for pressure data processing; this channel however, has a signal which indicates when kinematic data is being acquired. In summary, three channels of A/D are required whenever kinematic data is acquired in order to determine the temporal relationship of the pressure and kinematic data.

The data and alarm signal from prosthesis 43 is transmitted in two "fields". Each field includes 16 output values on the data channel. Both fields have the output from all pressure transducers and from the thermistor, field A also contains two constant voltage signals used to scale the transducer outputs and field B carries information defining the threshold above which data is compressed. The determination of which field is active is established by inspecting the alarm channel; the transition from one field to the other is used to find the frame start.

Appendices B and C discuss calibration of the prostheses and Appendix E contains the programs used for initial processing of pressure data. Figure 4-2 shows a block diagram of the procedure. Factors that need to be considered are: thermal effects; decompression of data; scaling of the output; alarm conditions; offset of transducer output at zero pressure; and gain of transducer output with pressure.

The thermal measurements for the test are averaged and interpreted via equation 4.1, where T is in degrees Fahrenheit and y is the thermistor output. This relationship was determined during the transducer calibration as detailed in Appendix C. The temperature at which measurements are made influences the offset in each transducer's pressure relation (equation 4.2). In this equation b_i represents the offset of transducer i, a_i and c_i are the thermal adjustment factors for that transducer, and $\overline{T}$ is the mean temperature for that test. The thermal adjustment for each transducer offset is shown in Table 4.2. The programs that process data from prosthesis 33 define temperature in degrees C, while the processing programs for prosthesis 43 calculate temperature in degrees F.

$$T = \sqrt{(y - 2861.0)/0.339} \tag{4.1}$$


Analog Data
13 T C Z 1 2 3 4 5 6 7 8 9 10 11 12 13 T P L 1 2 3 4 5 6
Sync/ Alarms
FIELD 1
FIELD 2
Auxiliary Input
9 10 11 F1 F2 F3 F4 G 1 2 3 4 5 6 7 8 9 10 11 F1 F2 F3 F4 G 1 2
Strobe: Sample Trigger

F1, F2, F3, F4: Foot Switches	C: Scaling signal 2.5 V
G: Goniometer	Z: Scaling signal 0 V
T: Thermistor output	P: Power level
	L: Compression Level

Figure 4-1: Signal from Prosthesis 43

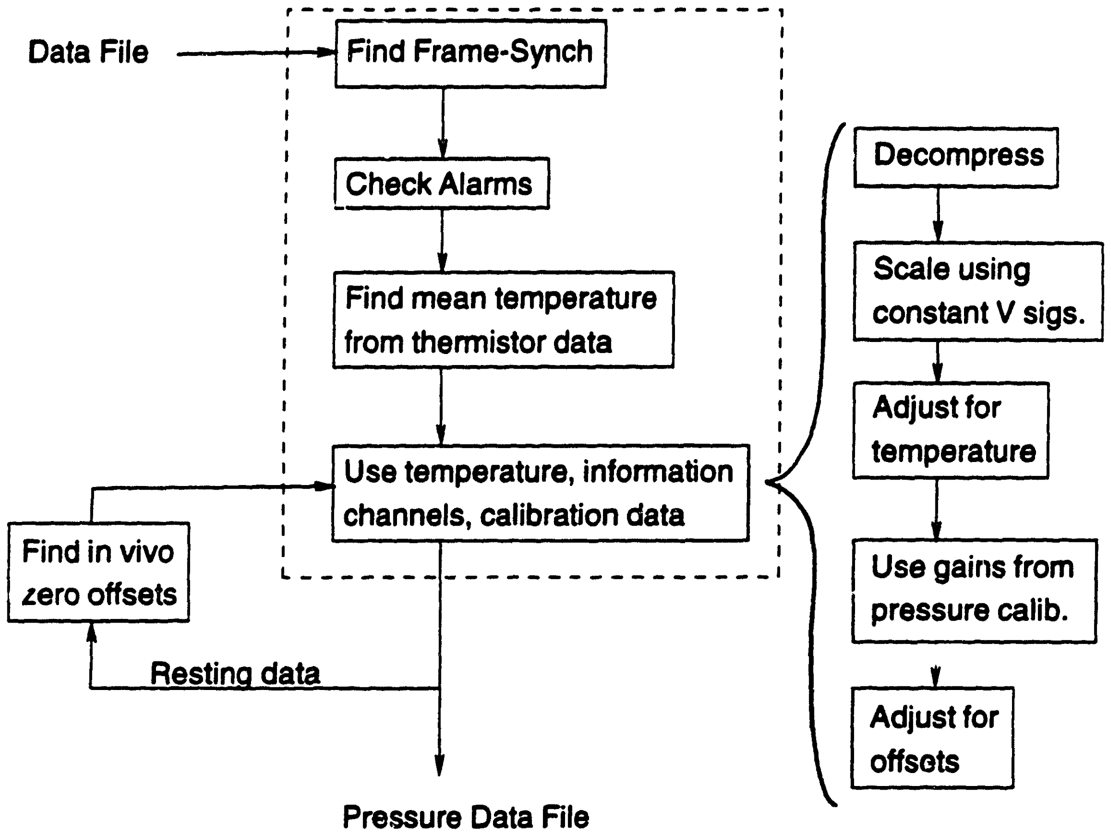

Data File
Find Frame-Synch
Check Alarms
Find mean temperature from thermistor data
Use temperature, information channels, calibration data
Find in vivo zero offsets
Resting data
Pressure Data File
Decompress
Scale using constant V sigs.
Adjust for temperature
Use gains from pressure calib.
Adjust for offsets

Figure 4-2: Procedure for processing pressure data

Table 4.2: Thermal adjustment to transducer offsets for both prostheses

Prosthesis	**Transducer**	a_i	c_i
33	1	-2.67	0
	2	-8.45	0
	3	-20.15	0
	5	2.52	0
	7	0.48	0
	9	-32.95	0
	10	-96.2	0
	11	16.13	0
	12	3.925	0
	13	-37.3	0
43	1	-0.0308	-153
	2	-0.275	1009
	3	-0.412	22
	5	-0.150	555
	6	0.201	359
	7	-0.113	150
	8	-0.0564	251
	9	-0.2899	314
	10	-0.4408	695
	11	-0.177	-159
	12	-0.0815	1068
	13	-0.0722	-175

$$b_i = b_i - a_i\overline{T} - c_i \tag{4.2}$$

One of the signals transmitted in field B is the compression level. Any data in that frame above the threshold must be decompressed. Equation 4.3 indicates how this is done; L is the compression threshold and x_i is the output of transducer i. The data is divided into the part below the compression threshold and the part above the threshold. The second part is multiplied by 4 and added to the compression threshold to obtain the actual value. Compression above a certain voltage was introduced in order to avoid saturating the A/D converter. In prosthesis 33 the compression threshold wasn't transmitted but was fixed at 3 V. Data above that value was also compressed 4:1.

$$x_i = \begin{cases} 4(x_i - L) + L & \text{if } x_i > L \\ x_i & \text{otherwise} \end{cases} \tag{4.3}$$

Two constant voltage signals are transmitted with the transducer outputs. The difference between these signals is used to appropriately scale the transducer output, as shown in equation 4.4 where Z represents a 0 V. signal and C represents a higher constant voltage. The 0 V signal is subtracted from both the data and the higher voltage signal (1 V for prosthesis 33 or 2.5 V for prosthesis 43). In this equation the result is the scaled transducer output in units of "corrected millivolts" (cmv).

$$x_i = 1000.0\left(\frac{x_i - Z}{C - Z}\right) \tag{4.4}$$

The relationship between transducer output in cmv and pressure is linear for most of the transducers and can be described by a gain and an offset (data from the nonlinear transducers have not been used in results). As described above the offset at zero pressure changes with temperature; it also varies over time on the order of months for transducers in both prostheses. This variation necessitates some means of determining the offset after the prosthesis is implanted and a known calibrating pressure cannot be applied. After implantation the zero-pressure offset is estimated by

Table 4.3: Estimated offset errors for prosthesis 33 *in vivo*

Est. ex-*vivo* offsets (temp. corrected to 38.6° C	**August 1989 *in vivo* offsets**	**Est. correction to *in vivo* pressure estimates (MPa)**
-1457	-1507	-0.7
939	905	-0.3
785	835	0.55
589	639	0.5
1673	1657	-0.2
-1051	-1041	0.1
375	425	0.44
935	871	-0.46
25	-51.7	-0.58
-638	-662	-0.2

averaging data taken while the subject lies relaxed. These "zeros" are almost certainly not all at 0 MPa. The error introduced in pressure measurements is estimated in table 4.3. These were obtained by comparing prosthesis 33 post-*vivo* zeroes with the preceding *in vivo* data from prosthesis 33, after correction for temperature. The temperature correction was made using post-*vivo* temperature calibration data; *in vivo* the temperature in the hip was estimated at 38.6° C (101.5° F), using temperature data from the integra; thermistor implanted in prosthesis 43 in the second subject. A positive value indicates that the pressure *in vivo* was probably under-estimated by that amount, while a negative sign indicates that the *in vivo* estimates were probably over-estimated by the amount shown. The estimated error is relatively small for all transducers compared to the noise in the signal from this prosthesis which was +/- 0.3 MPa for some of the transducers.

Prior to implantation the transducers were calibrated in a hydrostatic pressure chamber, and, in the case of prosthesis 33, post-*vivo*. The calibration procedure and equipment is detailed in Appendix B. The gain in the relationship between transducer output and pressure was found to be invariant with both temperature and over time. Pressures at each transducer are obtained through equation 4.5, in which x_i is the

Table 4.4: Sample Duration vs. Effective Sampling Rate for Prosthesis 43

Time [sec]	A/D Channels	Eff. Freq. [Hz]
1	3	125
2	3	83.3
3	3	62.5
5	3	41.67
7	3	31.25
9	3	25

scaled transducer output in cmv, b_i is the transducer offset adjusted for temperature, G_i is the transducer gain with pressure, $\overline{y_{z_i}}$ is the mean zero pressure reading for that day, and y_i is the estimated pressure measured at transducer i.

$$y_i = \frac{x_i - b_i}{G_i} - \overline{y_{z_i}} \tag{4.5}$$

4.2 Sampling Rate

Some frames of data must be skipped in order to acquire data for long periods of time (1-10 seconds) and not exceed the DMA limitations. The data acquisition program is in Appendix E and in part consists of clock programming which enables the system to skip a number of frames between each pair of frames collected. A pair of frames of data are required for the interpretation of the pressure data. The number of frames skipped is therefore a multiple of two and varies with the length of time for which data is to be collected, in the trade-off to collect data at as high a frequency as possible. Tables 4.4 and 4.5 show the tradeoff between the length of time of data collection and the frequency of data acquisition. Of the two frames collected together, one set of transducer outputs is discarded to maintain a single sampling frequency.

Table 4.5: Maximum time can sample for at a given effective frequency, for Prosthesis 43

Effective Frequency [Hz]	Time [sec]
500	0.9
125	1.8
83.3	2.7
62.5	3.6
50	4.5
41.67	5.4
35.7	6.3
31.25	7.25
27.78	8.1
25	9.0

4.3 Normalization of Pressure Data

The two subjects participating in this study were both implanted with the same size prosthesis, but were otherwise quite different in bodily dimensions. Therefore comparison of pressures between subjects requires some consideration of the physique of the respective subjects. Of a number of normalization factors considered, the dimensions which seem most relevant are body weight, height, acetabular area, and perhaps leg length or some indicator of the location of the center of mass. Since the two subjects have essentially the same size acetabulum, the normalization chosen was body weight over height-squared. This provides a non-dimensional number for comparison of data. A scaling factor of 1000 was applied to reduce the range of data values to 0 - 100. (The maximum value thus far is 82.3, from the second subject.) Equation 4.6 normalizes pressure data where yn_i is the normalized pressure, BW is body weight, and H is height. The first subject (implanted with prosthesis 33) weighed 665 N and was 1.68 m tall. The second subject (implanted with prosthesis 43) weighed between 530 and 578 N and was 1.6 m tall.

$$yn_i = \frac{y_i}{1000.0\left[\frac{BW}{H^2}\right]} \tag{4.6}$$

4.4 Kinematic Data Processing

The kinematic data was collected and processed at MGH using the version of the TRACK system installed at the Biomotion Laboratory. Kinematic data was processed according to the current methods used at that facility at the time of data collection, thus not all of the kinematic data has been processed identically. Functionally, the only critical difference has been the filtering applied to kinematic data. The filter cut-off frequency and filter type affect the estimates of joint angles made from processed kinematic data. All of the kinematic data used in this thesis has been low-pass filtered. Early data from prosthesis 33 was low-pass filtered using a Butterworth filter with a relatively high cut-off frequency (different depending on the day). Kinematic data taken for prosthesis 43 has all been filtered at 6 Hz using an FIR filter.

The kinematic data low-pass filtered with a high cutoff frequency produced noisier estimates of joint angles; when used to calculate acetabular locations of transducers those kinematic data produced pressures applied over more extended areas of the acetabulum. In order to compare with data from prosthesis 43, the joint angle information from prosthesis 33 (first subject) was smoothed for data acquired in Dec. 1984, May 1985, and Oct. 1986. Kinematic data from prosthesis 33 from later dates had already been filtered with a sufficiently low cut-off frequency for use in comparisons. The smoothing routine used a generalized, cross-validatory spline smoothing algorithm. This routine was made available by Professor W. Durfee; the version used was written in January 1993.

4.5 Transformation to Acetabular Coordinates

4.5.1 Change in Coordinate Systems

The stresses seen by the natural cartilage side of the joint are of primary interest since only that side experiences remodelling, adaptation and degradation. The kinematic data is crucial to the determination of where on the acetabulum measurements are made, since the kinematic data gives the relationship of the femur to the pelvis as the subject moves. The process of transforming data from the prosthesis coordinate frame to one fixed in the acetabulum is thus a critical aspect of the processing. These coordinate transformations will be outlined here and are also explained in [8] and in Appendix E which contains the program that performs coordinate transformations. The process used assumes that only rotations matter, that no translations of the pelvis relative to the femoral head occur, an eminently reasonable assumption. For both the right and left hip axes X is anterior and Y superior, thus Z is lateral for a right hip and medial for a left hip.

The transformation from the prosthesis coordinate frame to the acetabular coordinate frame requires three steps; one from the prosthesis to the reference frame fixed to the femur, one from the femur to the pelvis which is calculated from kinematic data, and one from the pelvis to the acetabular coordinate frame. The first and the last of these do not change; they are calculated once and used for the whole data set. Only the relationship between the femur and the pelvis requires kinematic data. To make the programs more easily understood, the transformation from prosthesis to femoral coordinates is described in several steps corresponding to information relating the coordinate systems.

The first step is the specification of transducer locations in the prosthesis coordinate frame. The prosthesis coordinate frame is shown in figure 4-3 and is defined with the prosthesis z coordinate along the axis from the center of the femoral head to the center of transducer 1 (the central transducer). The prosthesis y direction is towards transducer 8. Prosthesis axis x is defined to point in an anterior direction, thus the prosthesis z coordinate is in opposite directions for a left and a right hip. Transducer locations in the prosthesis are shown in Appendix A which also contains the transducer locations in the prosthesis coordinate frame. The first transformation is to "stem" coordinates, $\bar{s}$, which only involves a rotation of α_0 degrees from prosthesis coordinates, $\bar{u}$, as shown in Figure 4-4; α_0 is 25° for both of the prostheses implanted thus far. The rotation matrix for this procedure is shown in equation 4.7. This rotation requires information on prosthesis design and on the subject's side in which it is implanted, $\mathcal{R}$. $\mathcal{R}$ is 1.0 for a right hip and -1.0 for a left hip.

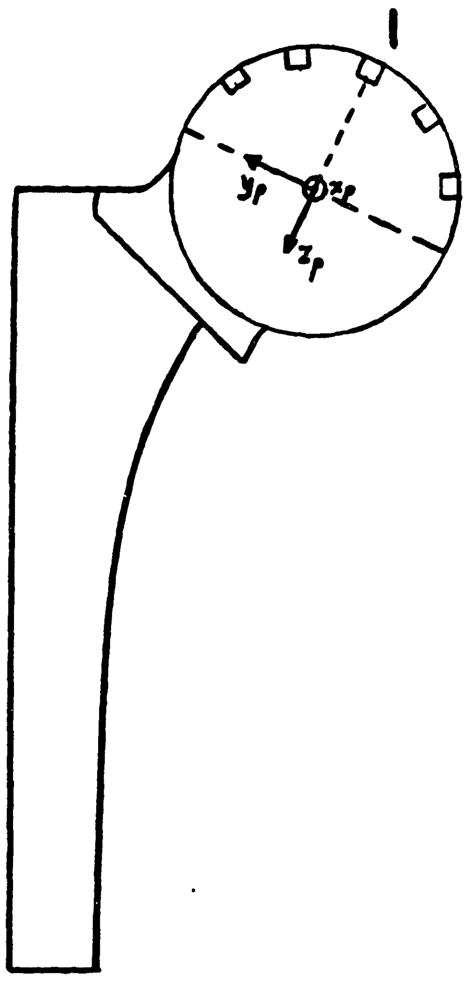


Figure 4-3: Prosthesis Coordinate Frame

$$\bar{s} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & sin\,\alpha_0 & -\mathcal{R}cos\,\alpha_0 \\ 0 & \mathcal{R}cos\,\alpha_0 & sin\,\alpha_0 \end{bmatrix} \bar{u} \tag{4.7}$$

The next rotation is from "stem" to "femoral" coordinates. This rotation uses a set of angles which describe the placement of the prosthesis in the femur. The angles used are indicated in Figure 4-5. Angle α_1 is the angle in the frontal plane between the stem y direction and the femoral y direction. Angle α_2 is defined inthe transverse plane and is the angle between the "stem" and "femoral" z directions. Angle α_3 is the angle between the stem and femoral z directions in the saggital plane. The values of these three angles were determined by the surgeon who performed the operation on both subjects and from x-rays. The rotation matrix which uses these angles to relate stem, $\bar{s}$, to femoral, $\bar{f}$, coordinates is shown in Equation 4.8. The array elements c[*i*][0] are calculated from the cross product of the c[*i*][1] and c[*i*][2] columns. $\mathcal{R}$ is 1.0 for a right hip and -1.0 for a left hip. Transducer locations are defined in femoral coordinates once, then stored for use through the whole data set.

$$\bar{f} = \begin{bmatrix} c[0][0] & \frac{-tan\,\alpha_3}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_3)^2}} & \frac{\mathcal{R}\,tan\,\alpha_2}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_2)^2}} \\ c[1][0] & \frac{1.0}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_3)^2}} & \frac{-\mathcal{R}\,tan\,\alpha_1}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_2)^2}} \\ c[2][0] & \frac{\mathcal{R}\,tan\,\alpha_1}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_3)^2}} & \frac{1.0}{\sqrt{1.0+(tan\,\alpha_1)^2+(tan\,\alpha_2)^2}} \end{bmatrix} \bar{s} \tag{4.8}$$

The rotation from femoral to pelvis coordinates uses the kinematic data. The clinical joint angles; hip flexion, abduction, and external rotation are found using the Tupling and Pierrynowski formulation [23] as is also done at the MGH Biomotion lab. The method for calculating these angles is presented in the subroutine mgh_ang.c in appendix E. Figure 4-6 shows the physical meaning of these angles. Equation 4.9 shows the rotation matrix used to relate femoral,$\bar{f}$, and pelvis,$\bar{h}$, coordinates using the clinical joint angles. Hip flexion is denoted by $\mathcal{F}$, external rotation by $\mathcal{E}$, and abduction by $\mathcal{A}$. $\mathcal{R}$ is 1.0 for a right hip and -1.0 for a left hip.

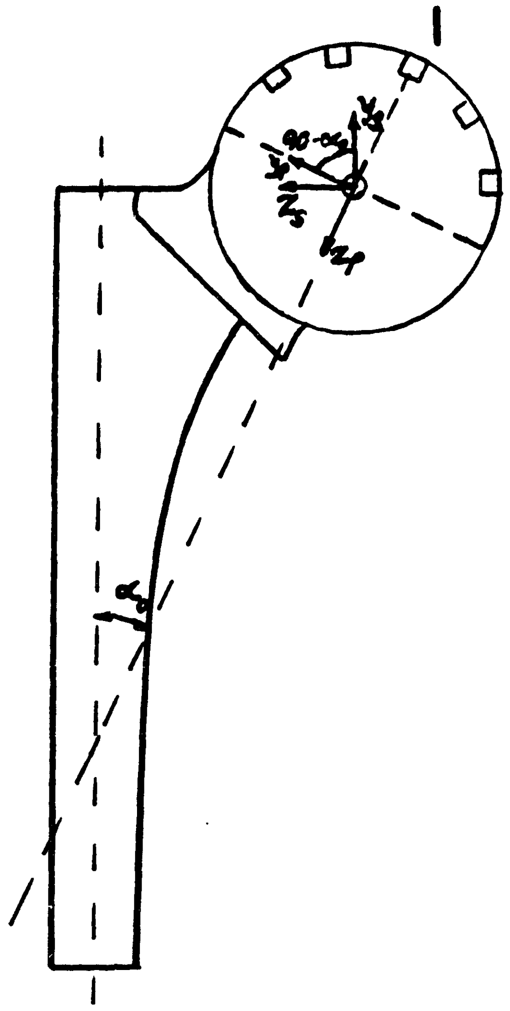


Figure 4-4: Prosthesis and Stem Coordinates

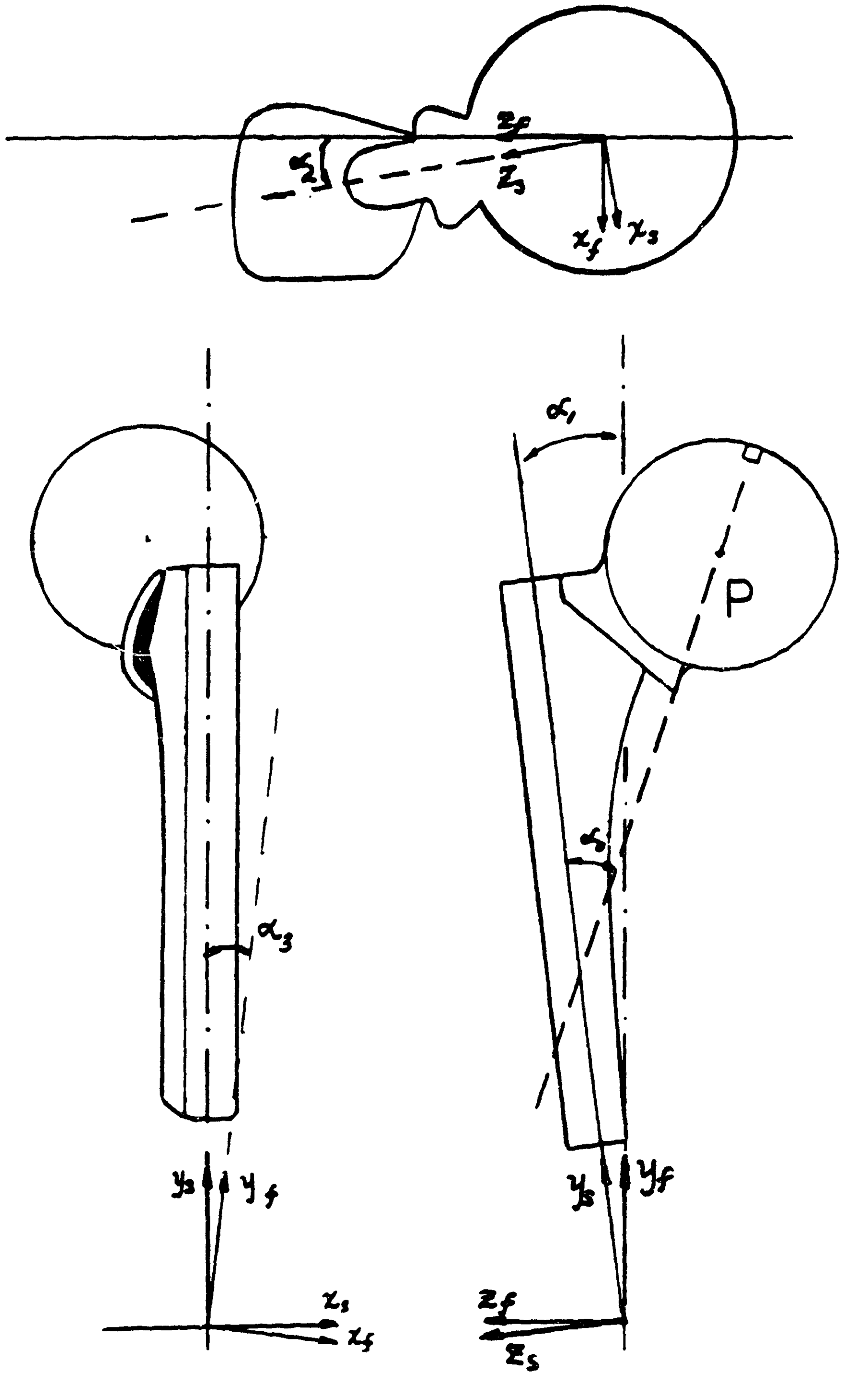


Figure 4-5: Stem to Femoral Coordinates

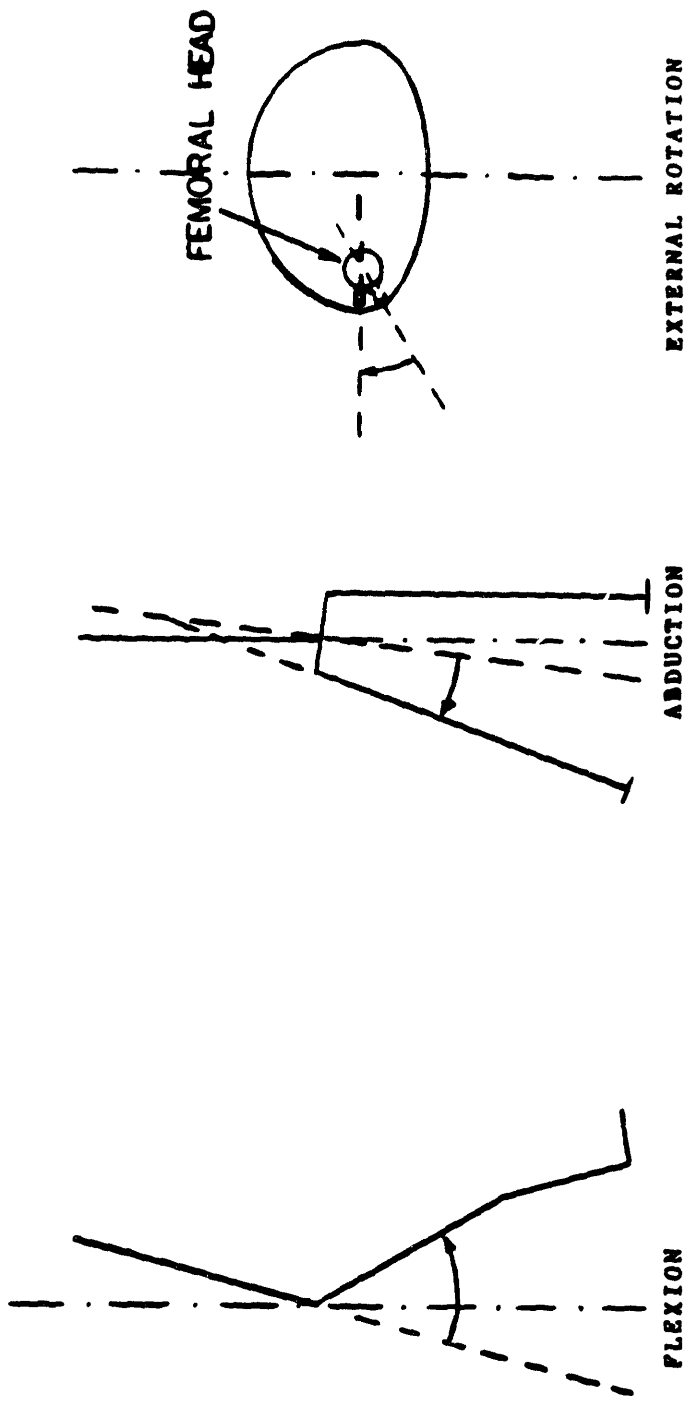


Figure 4-6: Definition of clinical joint angles

$$\overline{h} = \begin{bmatrix} cos\mathcal{E}cos\mathcal{F} & -cos\mathcal{A}sin\mathcal{F} & -\mathcal{R}sin\mathcal{E}cos\mathcal{F} \\ cos\mathcal{E}sin\mathcal{F} + sin\mathcal{E}sin\mathcal{A} & cos\mathcal{A}cos\mathcal{F} & -\mathcal{R}sin\mathcal{F}sin\mathcal{E} + \mathcal{R}cos\mathcal{E}sin\mathcal{A} \\ \mathcal{R}sin\mathcal{E}cos\mathcal{A} & -\mathcal{R}sin\mathcal{A}cos\mathcal{F} & cos\mathcal{E}cos\mathcal{A} \end{bmatrix} \overline{f} \tag{4.9}$$

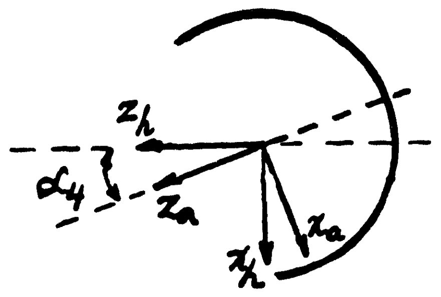
The final transformation is from the pelvis coordinates to acetabular coordinates. The orthogonal acetabular coordinate system is shown in figure 4-7, in which the x direction is defined by the medial direction and z is defined by the acetabular outward normal. Two angles are used to describe the relationship of the pelvis, $\overline{h}$, and acetabular, $\overline{a}$, coordinate systems; angle α_4 is the angle in the transverse plane between the medial-lateral direction (global Z) and the acetabular outward normal, angle α_5 is in the frontal plane between the medial-lateral direction and the acetabular outward normal. These angles are used to determine the rotation matrix shown in equation 4.10. The rotation matrix elements $c[1][i]$ are calculated by taking the cross product of the other two rows of the rotation matrix.

$$\overline{a} = \begin{bmatrix} cos\,\alpha_4 & 0.0 & -\mathcal{R}sin\,\alpha_4 \\ c[1][0] & c[1][1] & c[1][2] \\ \frac{\mathcal{R}tan\,\alpha_4}{\sqrt{1.0+(tan\,\alpha_4)^2+(tan\,\alpha_5)^2}} & \frac{-\mathcal{R}tan\,\alpha_5}{\sqrt{1.0+(tan\,\alpha_4)^2+(tan\,\alpha_5)^2}} & \frac{1.0}{\sqrt{1.0+(tan\,\alpha_4)^2+(tan\,\alpha_5)^2}} \end{bmatrix} \overline{h} \tag{4.10}$$

The acetabular coordinates for each transducer are subsequently stored as a longitude and latitude (two numbers are sufficient as the radius of the sphere is constant). The Cartesian coordinates are recalculated from these numbers by the programs that require displays in orthogonal coordinates.

4.5.2 Timing

Another issue in processing important to establishing acetabular pressure locations is the timing relationship between kinematic and pressure data. Since the two sets of data are collected by different computers, at different frequencies, and over different



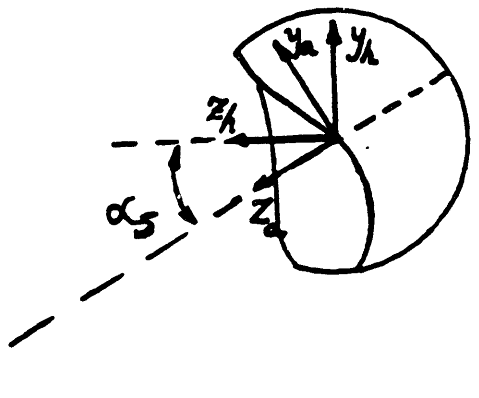


Figure 4-7: Relationship between Pelvis and Acetabular coordinates

lengths of time, a method for relating the two is needed in order to combine them. To achieve this a signal was output by the kinematic data acquisition system to the pressure data acquisition system via the auxiliary inputs. This synch signal goes low when kinematic data is being acquired thus it records the start and end of the kinematic data in the pressure data file. When pressure and kinematic data are being used together the one acquired at a lower frequency is used as is and the time for each frame of that data is calculated. The corresponding closest frame of the other type of data is calculated from the time and the known frequency of data acquisition.

4.6 Graphical Displays

Several types of data were collected in conjunction with the *in vivo* pressure data, and there are a multitude of ways in which these data can be combined and presented. Some require little explanation, such as plots of pressure vs. time for each transducer, but when kinematic and pressure data is combined some thought must be given as to how to present information effectively. The results presented here will use:

1. Discrete measured values: Maximum values reported by pressure transducers, both normalized and not, for a test, a movement, or a day.

2. Time displays: Plots of pressure vs. time, forceplate forces vs. time, and clinical joint angles vs. time.

3. Global summary displays: Plots of maximum, minimum, or mean pressure measured in each region of the acetabulum over the course of a test or a set of tests.

Figure 4-8 shows an example of plots of pressure measured at each transducer vs. time. This multi-plot type of display has been used a great deal for initial evaluation of each test session. In this particular format the data is from prosthesis 43 and the thermistor data is in the upper left-hand corner, followed by the two scaling channels, the compression level and the power level, then plots for the active transducers (all except number 4 for prosthesis 43). Additionally, plots of auxiliary data, such as goniometer or foot switch can be presented on the same page. Figure 4-9 is another example of pressure vs. time plots, in this case the pressure at each transducer is normalized and all transducers are plotted on the same graph. This format is used in the Results section of this thesis. In Chapter 5, graphs of pressure data acquired from the first subject (prosthesis 33) present smoothed pressure data. The smoothing algorithm used was the same as that used to smooth joint angle data and uses a generalized cross-validatory spline smoothing algorithm.

An example of a plot of clinical joint angles vs. time is shown in figure 4-10; in these plots the left-hand y axis is used for flexion and abduction, the right-hand y axis is used for external rotation. Kinematic data is not necessarily valid for the entire test session, as the subject may move out of view of one of the Selspot cameras. For the current subject a factor indicating whether the subject is in view or not is stored in the kinematic data files, thus only kinematic data acquired when the subject is in view of the cameras are used in processing and in displays. Kinematic data from earlier in this study did not have the same indicator, but clinical joint angles became +/- 90° when the subject was not in view. The data sets were clipped so that data when the subject was not in view were not used in smoothing or in subsequent processing. For this reason the time index in graphs does not always start at 0.

Forceplate vertical force vs. time is shown in figure 4-11, the forceplate forces are normalized by body weight. In current procedure a weight estimate for the subject is included in the kinematic data files, stored from the early post-operative phase. Static stance data was examined to determine the weight of subject 2 on the test dates from which data is presented; these estimates were used in normalizing the pressure and forceplate data for subject 2. An attempt was made to apply the same process for subject 1. No similar static stance data was available, however, weight estimates from other sets of data indicated that the original estimate of her weight was nearly correct for all the days on which data is presented in this thesis.

Calculation of the acetabular location of each pressure measurement frame enables the compilation of a set of basic statistics for each acetabular region over a given data set or a set of data sets. The values tabulated have been: regional maximum, regional minimum, regional mean, number of points measured in the region and RMS value for

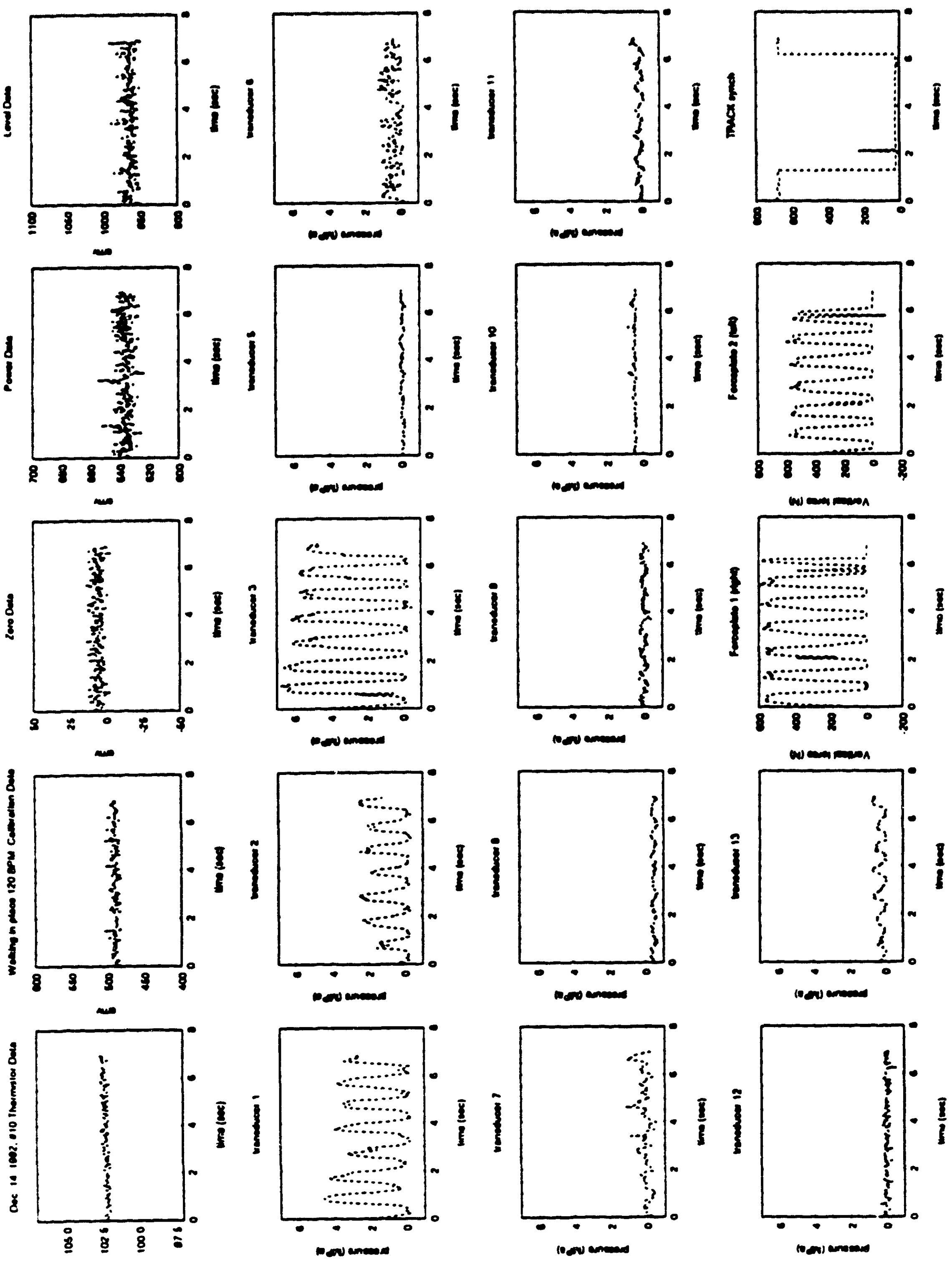


Figure 4-8: Pressure vs. Time

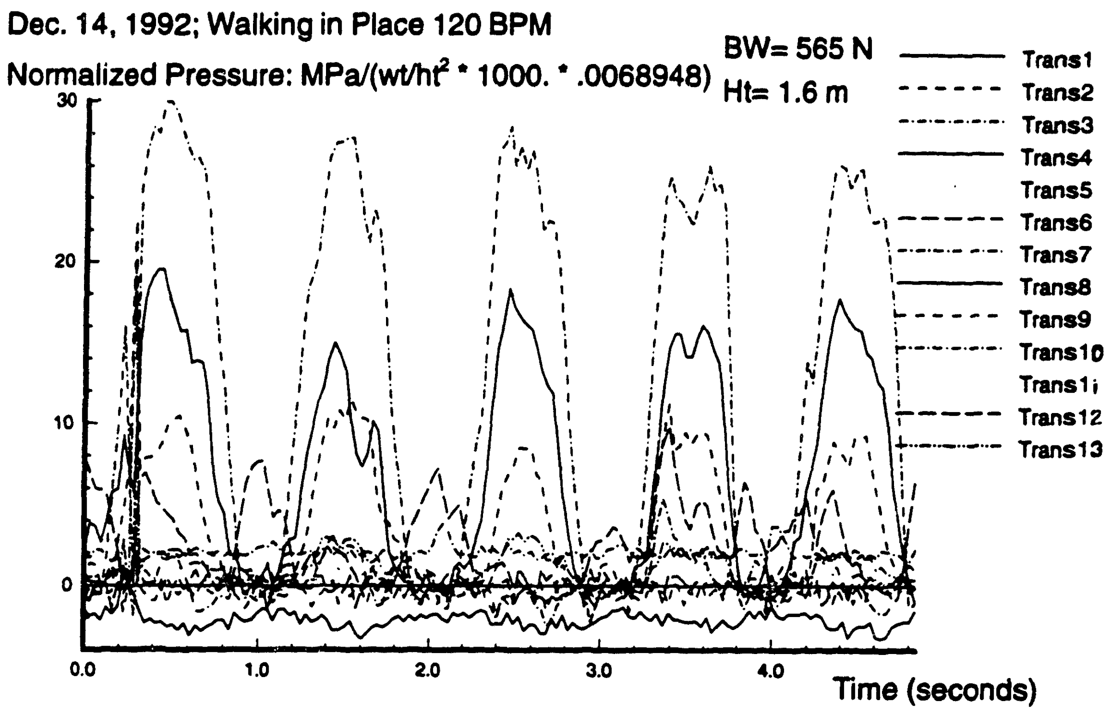


Figure 4-9: Normalized Pressure vs. Time

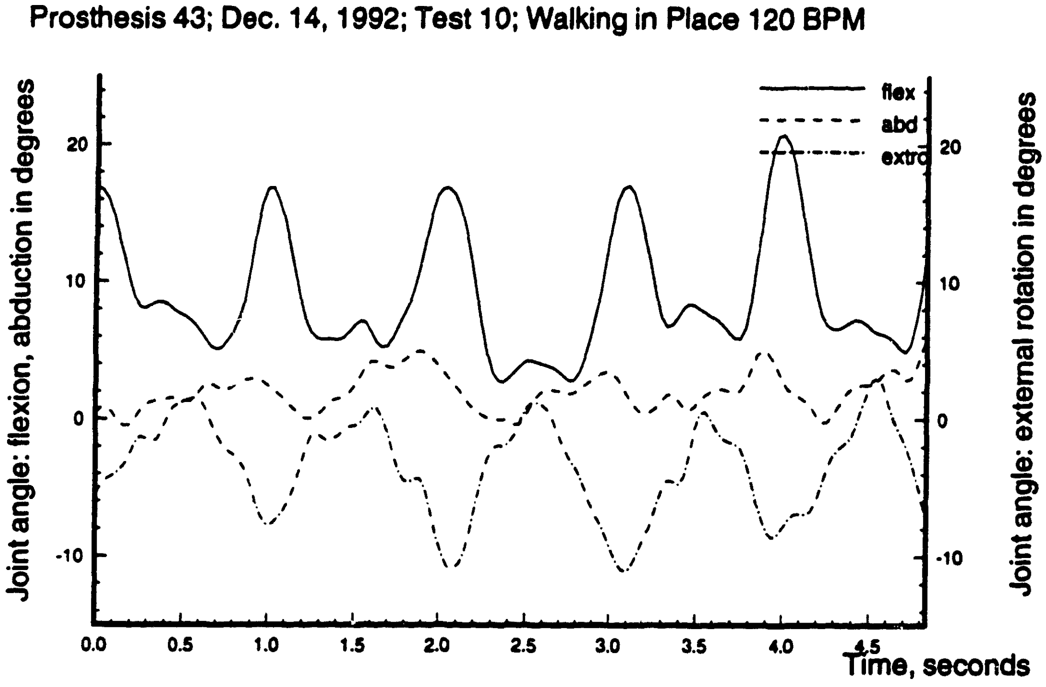


Figure 4-10: Joint Angles vs. Time

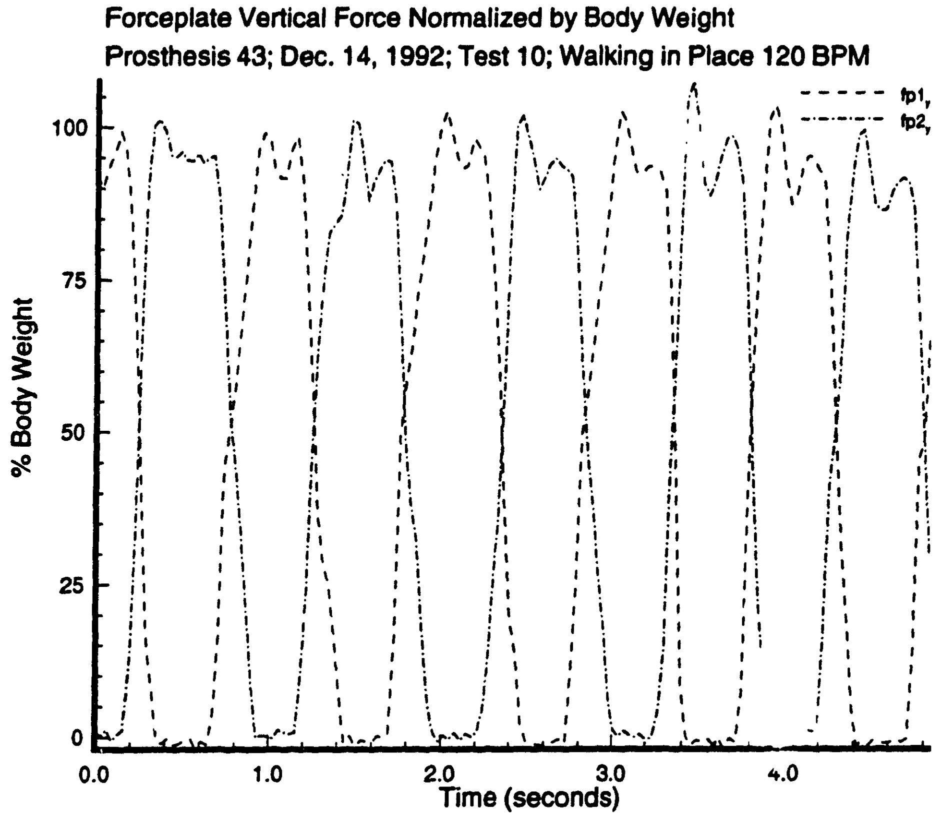


Figure 4-11: Forceplate forces vs. Time

the pressure. These tabulations permit the display of sets of data in a concise manner for comparison with other data and to determine whether high pressures were always measured in the same region. A display of maximum normalized regional pressures on the acetabular cartilage surface is shown in figure 4-12. This type of plot uses kinematic and pressure data, but does not explicitly present temporal information. The acetabular surface is sectioned into bins and the maximum, minimum, and mean pressure and the number of measurements made in each bin are recorded. Regions in which no measurements were made are left blank. The view of figure 4-12 is looking into the socket, with all data oriented for a right hip, and anterior toward the right side of the plot. In the presentation of similar results for movements such as rising from a chair and stair climbing, the acetabular surface has been rotated to show the regions of high pressure more clearly.

Similar to figure 4-12, but without the kinematic information, is the plot shown in figure 4-13. In this case the maximum pressure measured at each transducer is displayed by a symbol at that tranducer's location in the prosthesis. The view is looking into the inner surface of the femoral ball hemisphere, with data again displayed as for a right hip.

Since evaluation of the effect of prosthesis sizing on acetabular cartilage response is global in scope, displays of regional maximum pressures or normalized pressures have been used extensively in this thesis. The data sets presented in the Results section are accompanied by plots of the joint angles, forceplate forces, and normalized pressures vs. time, to convey a sense of the temporal course of the movement and to evaluate the similarity of the movement between trials and between subjects.

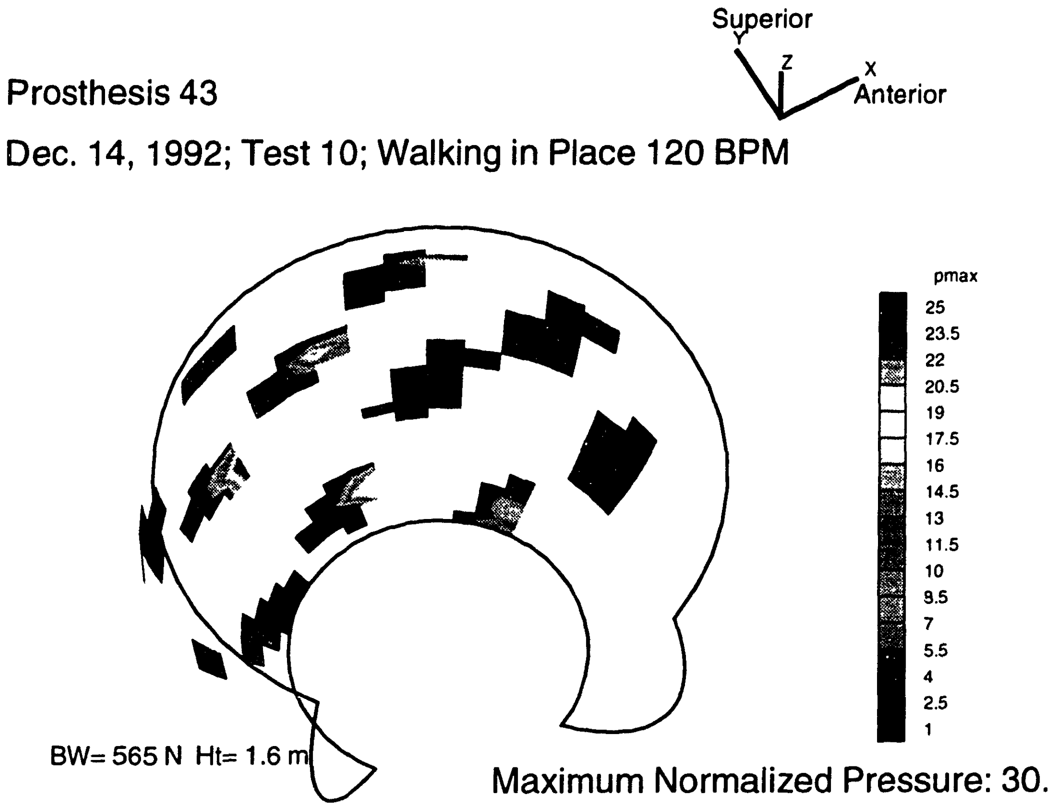
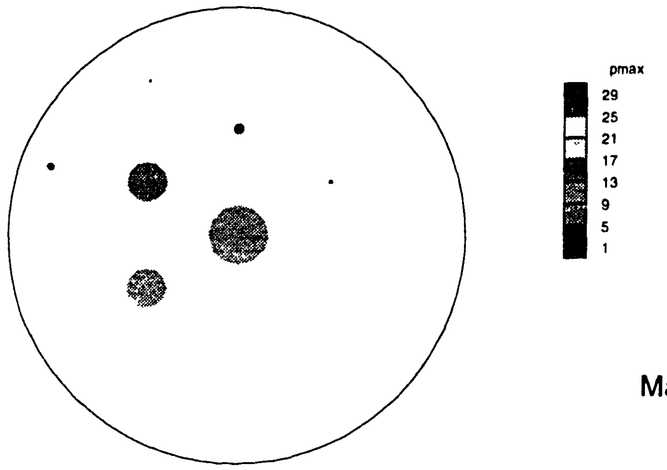


Figure 4-12: Maximum Local Acetabular Pressures

MVIC Abduction

Normalized Maximum pressures at each transducer



Prosthesis 43:
Dec. 10, 1991
Tests 2 and 3

Data displayed in
Right hip orientation

Maximum normalized pressure: 11.6
BW= 529 N
Ht = 1.6 m

Figure 4-13: Maximum Pressures displayed at Transducer Locations

Chapter 5

Results

Of the huge volume collected, data from several types of movements will be presented in this chapter. These movements were chosen for different reasons: some reported the same activity over a number of different days, some resulted in especially high pressures, and others are activities frequently considered by other researchers. Data will be presented separately for each movement and each subject. The protocol for some movements been different for the two subjects, and in many cases the protocol changed over the years the first prosthesis was implanted. In general the results will be presented in a number of different formats; pressure vs. time, forceplate forces vs. time, joint angles vs. time, and regional statistics. Evaluation of the similarity of a movement performed on different days or by a different subject may be made in part by considering the joint angle plots and the forces recorded by the forceplates.

The first subject received prosthesis number 33, the second received prosthesis 43. The prosthesis fitted to the first subject was 0.6 mm greater in diameter than her natural femoral head. The prosthesis implanted in the second subject was 0.7 mm smaller in diameter than his natural femoral head.

5.1 Gait

The movement inspected at the greatest depth is walking. In virtually every test session with each subject (after the first post-operative week or so) data were taken with the subject walking normally across the lab. There were minimal differences in the way these tests were performed over the course of time and between subjects, it is an activity performed routinely during the daily life of each subject, and it is an activity the results of which can be compared to those of other researchers. In all "unconstrained" tests the subject was instructed to walk as he or she normally would and to look ahead (rather than down at the ground). This gait data is labeled "free-speed" walking. Among the other types of gait data collected were walking tests with the pace set to some number of steps per minute using a metronome. These tests are labeled 60 beats per minute (BPM) walking, or 120 BPM, or 80 BPM, and are otherwise similar to the free-speed tests.

5.1.1 Free Speed Gait: Prosthesis 33

Figure 5-1 shows the magnitude of the local pressures at the joint during walking for the first subject for 6 months to 5 years post-operative. Data from all working transducers is shown, though not all record significant pressures. The pressures in this figure are normalized and smoothed. Figure 5-2 shows the corresponding forceplate forces during the same set of tests; the vertical force at the forceplate is shown and it is normalized by body weight. Usually the force on the instrumented leg is measured at forceplate 1 for this subject; however in the December 1984 data forceplate 2 was on the right. The joint angles for the tests shown are presented in figure 5-3. Figure 5-4 presents the regional statistics for the same time period and the same tests, though when more than one test was available they were both used to compile regional statistics.

Significant among this accumulated data are: the highest pressure during gait was measured 6 months post-operative, pressure at 11 months post-operative was nearly as high, then pressures decreased for the next 4 years; a change in the regions in which the maxiumum pressure was measured occurred between 6 months and 11 months post-op, and again between 2 years and 5 years post-operative. At 6 months post-op there are three regions in which relatively high pressures were measured, with the outer two of the three regions bearing higher stresses. Although no kinematic data is available for the pressure data taken in the early post-operative period, 4 transducers were measuring pressures on the order of 3 MPa at approximately the same times during the gait cycle, which would suggest that there were also 4 regions of relatively high pressure during gait in the first two weeks after implantation. In the 11 month post-operative data the same general region was more highly stressed than other areas, however higher pressures were measured in the more central area. This pattern is substantiated in the data from Oct. 1986, though the highest pressures have decreased in magnitude. By March 1989, however, the loading pattern has become even more diffuse, with pressures nearly as high as those measured centrally also being found in the outer regions of the acetabular surface. The acetabular cartilage had thinned by this time and the prosthesis was in the initial stages of protrusion, as is evidenced by load distribution. The subject was feeling discomfort during extended activity at this time.

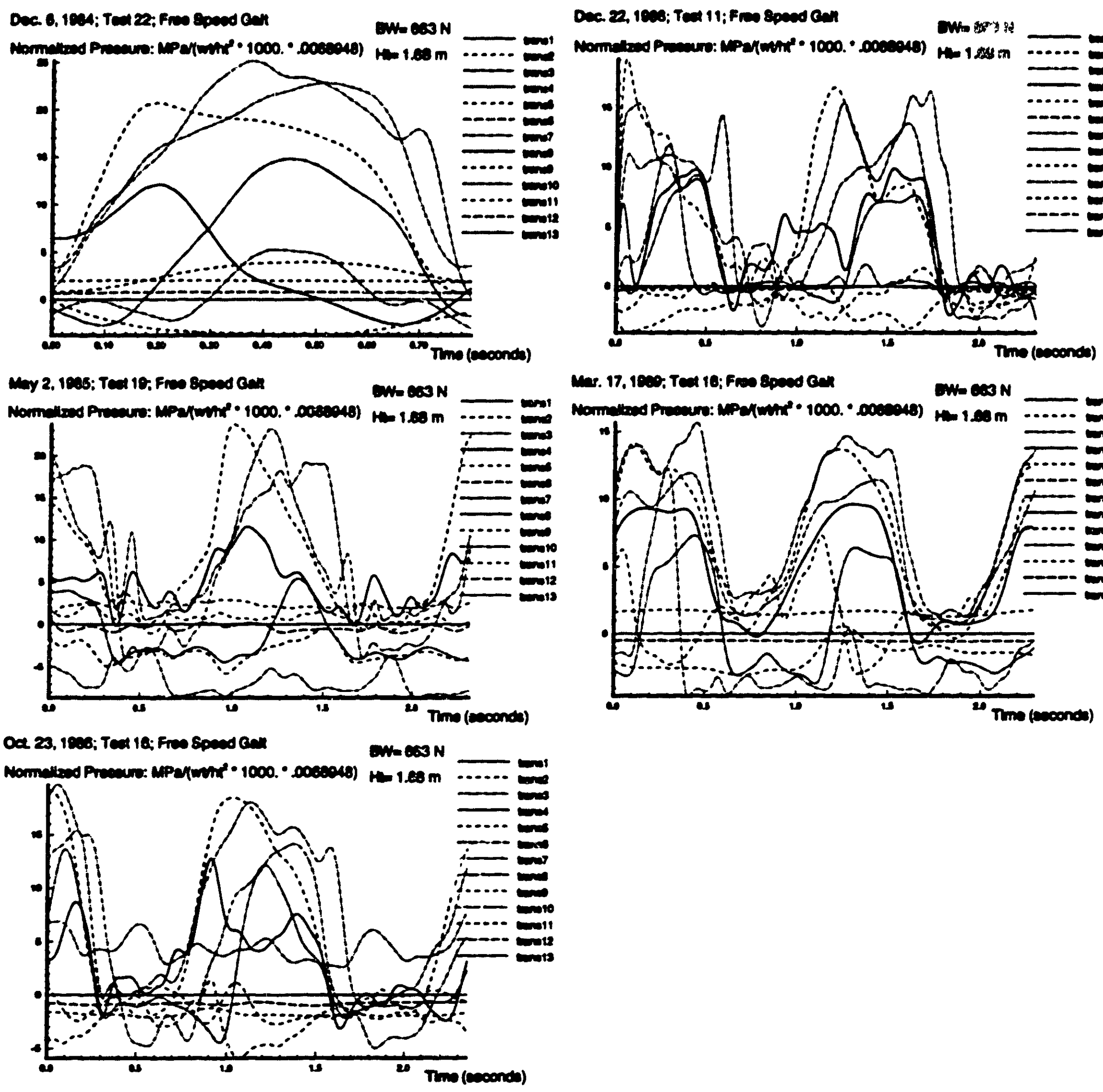
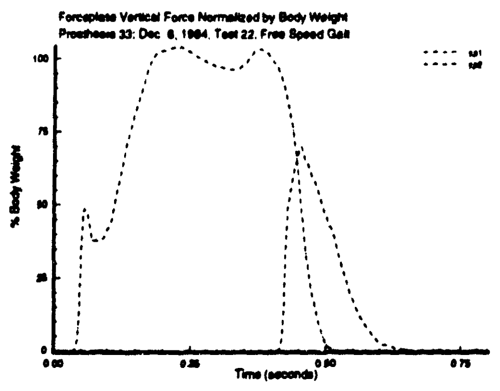
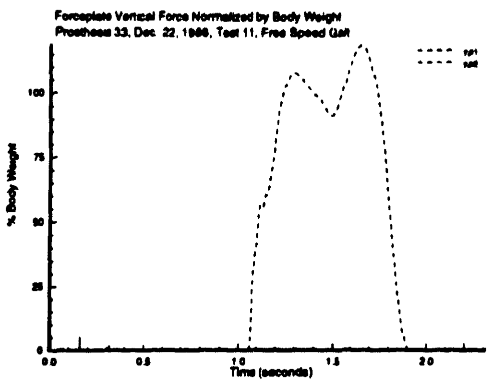
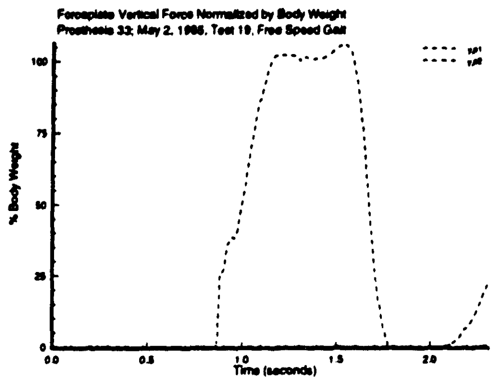
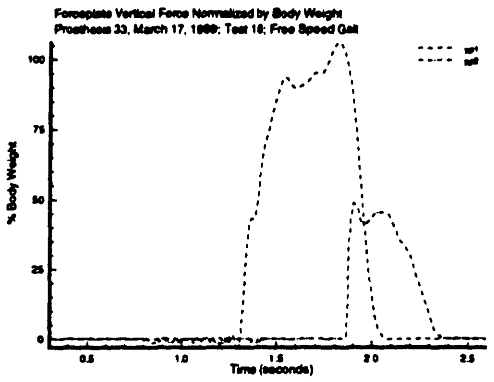


Figure 5-1: Pressures for subject 1: Free Speed Walking









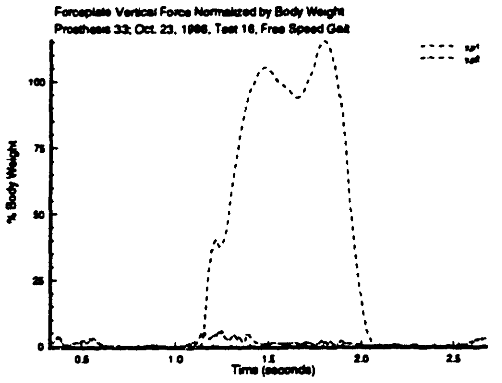


Figure 5-2: Forceplate data for subject 1: Free Speed Gait

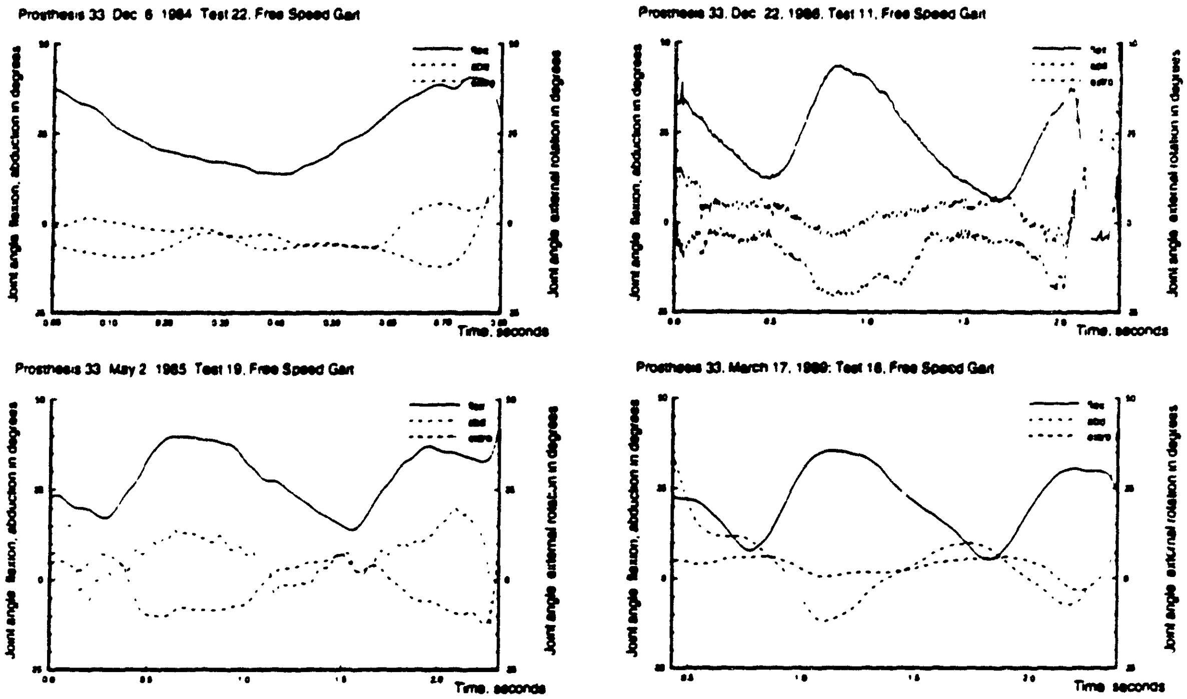


Figure 5-3: Joint Angles for Subject 1: Free Speed Gait

5.1.2 Free Speed Gait: Prosthesis 43

Figure 5-5 shows the magnitude of the local pressures at the joint during walking for the second subject for 4 months to 1 year post-operative. Data from all working transducers are shown, though not all record significant pressures. The pressures in this figure are normalized. Figure 5-6 shows the corresponding forceplate forces during the same set of tests; the vertical force at the forceplate is shown and it is normalized by body weight. In general forceplate 2 corresponds to the instrumented leg for this subject. Figure 5-7 presents the regional statistics for the same time period and the same tests, though when more than one test was available they were both used to compile regional statistics. The joint angles for the tests shown are presented in figure 5-8.

The maximum normalized pressures measured in the second subject during free speed walking rose between 4 months and 1 year post-operative. The regional maximums cover a similar range to that found in the first subject, with the normalized maximum pressures very similar for free-speed walking during the first year post-operative. In contrast to the first subject, however, in the second subject the regions in which higher pressures are measured have become more diffuse between 4 months and 1 year post-operative. No kinematic data is available coincident with the pressure data taken during the initial post-operative period, but the pressures for one set of gait data from December 1991, shown in fig 5-5, indicate that at that time only one transducer was registering significant pressures. In later data, the location of the higher pressure regions was very similar to that found in the first subject.

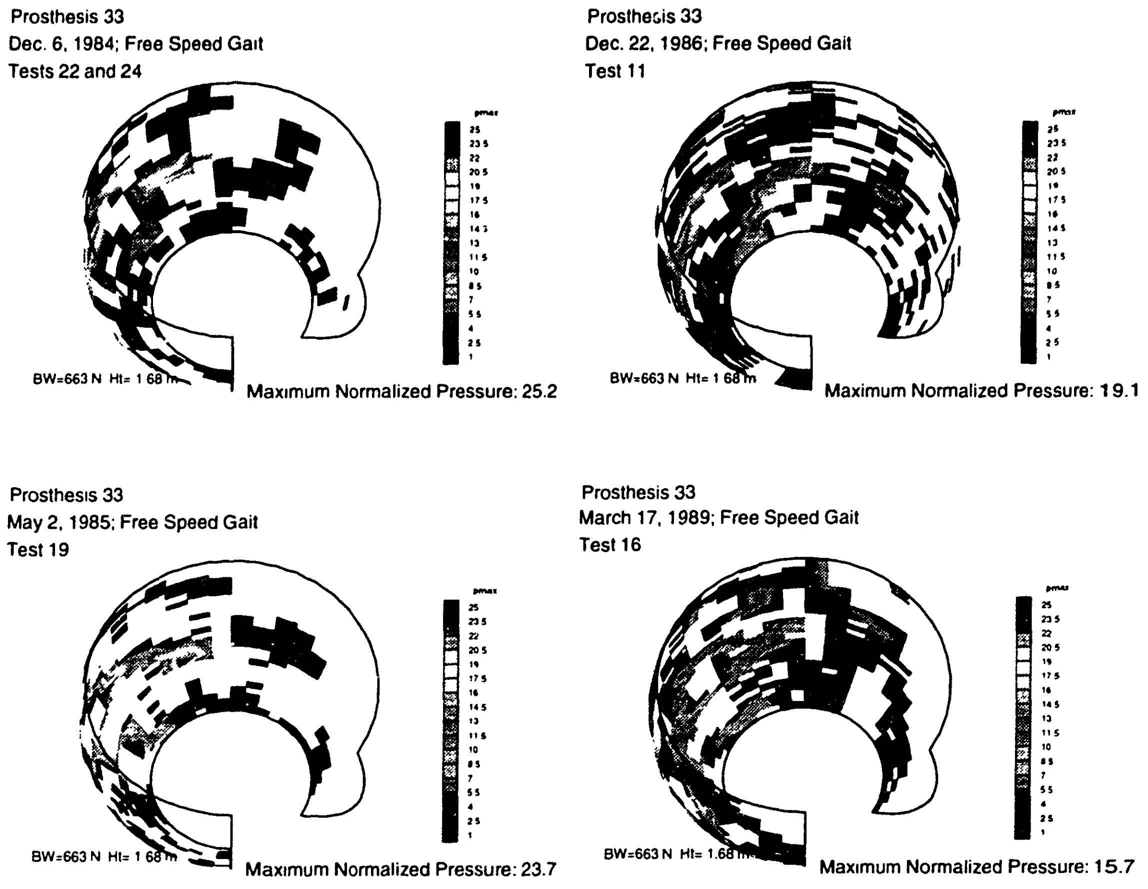


Figure 5-4: Regional Maximums for Subject 1: Free Speed Gait

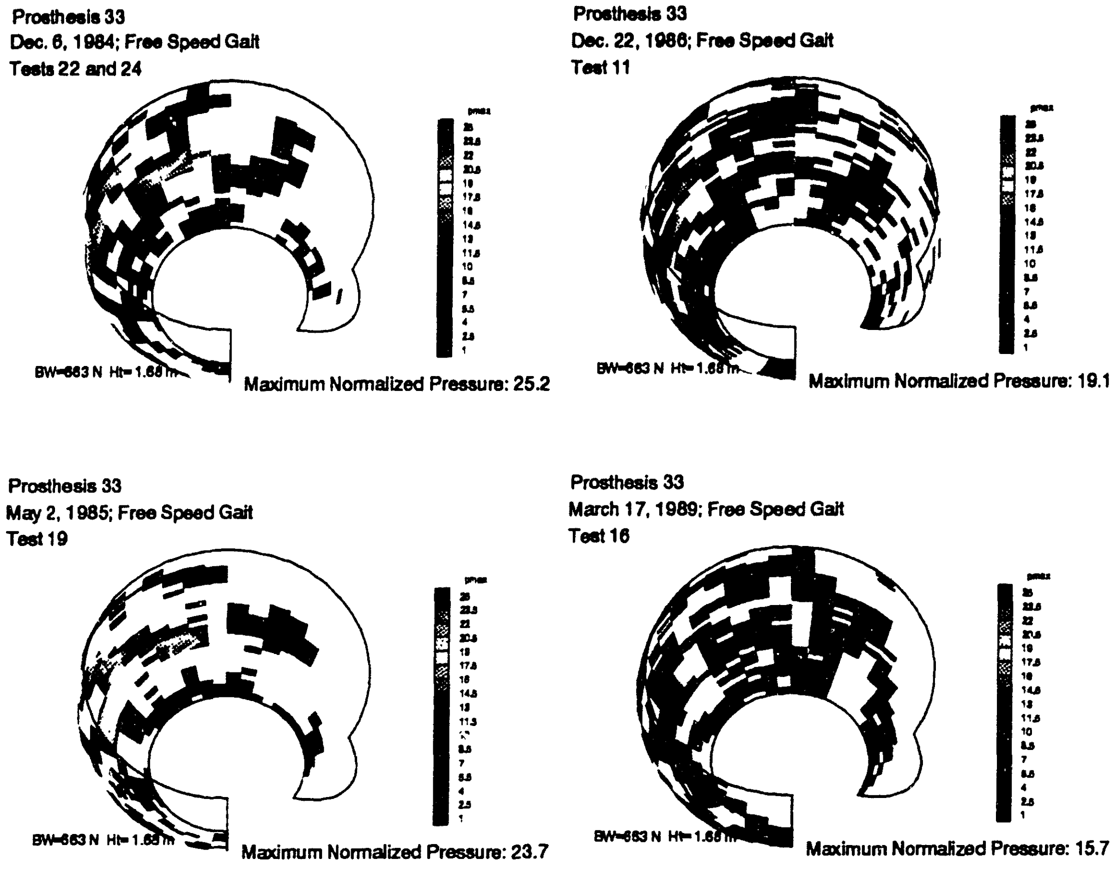
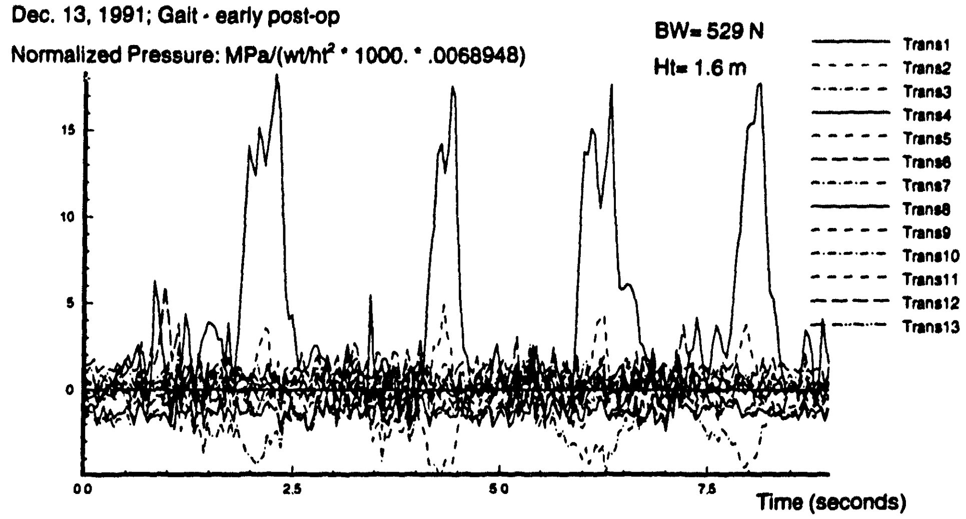
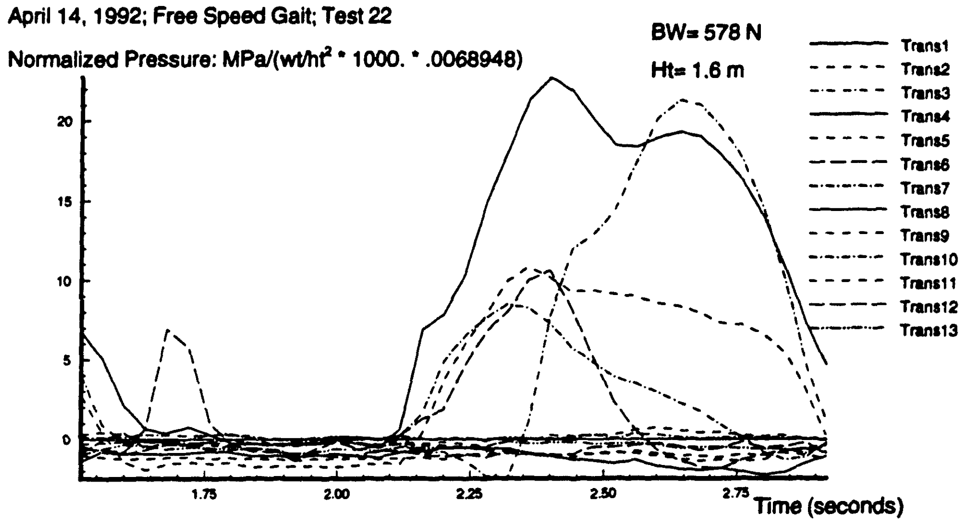


Figure 5-4: Regional Maximums for Subject 1; Free Speed Gait





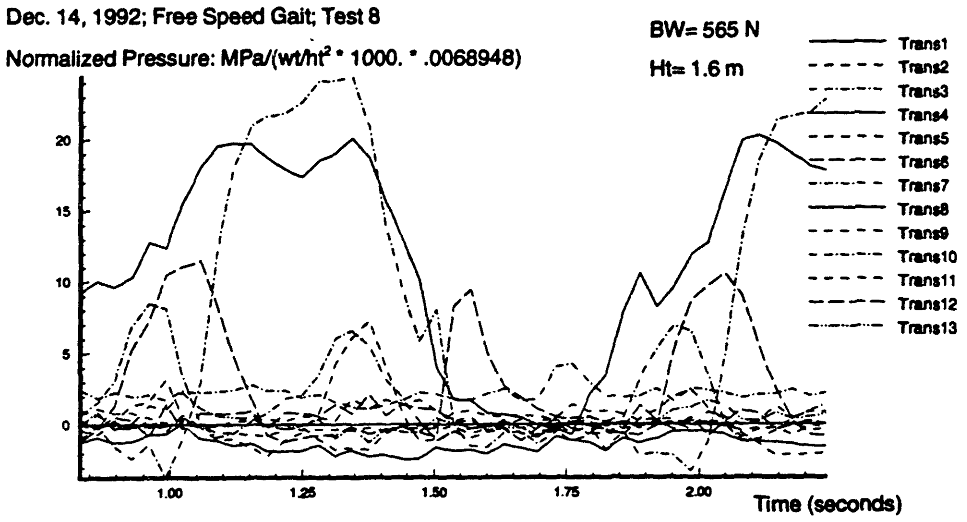
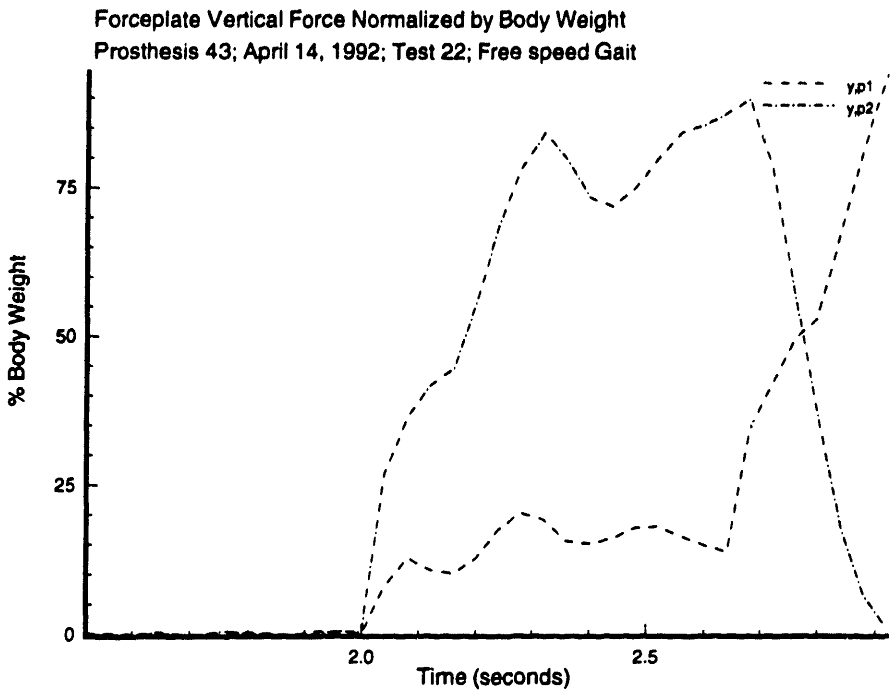


Figure 5-5: Pressures in the Hip Joint: Free Speed Walking



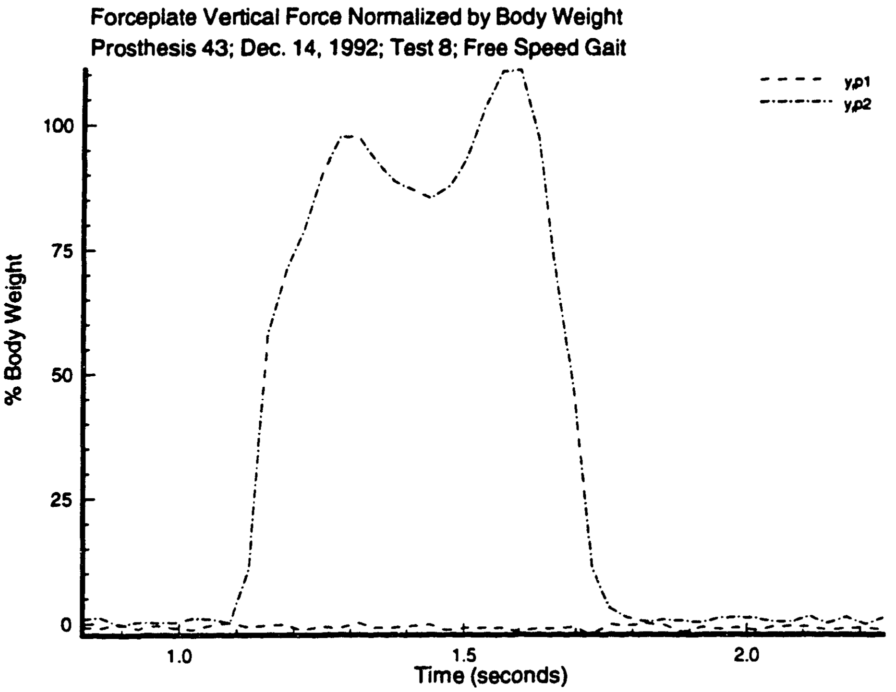
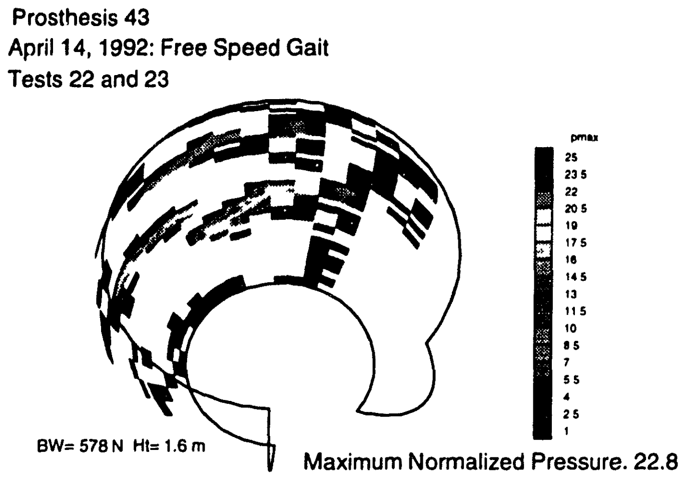


Figure 5-6: Forceplate data for subject 2: Free Speed Gait



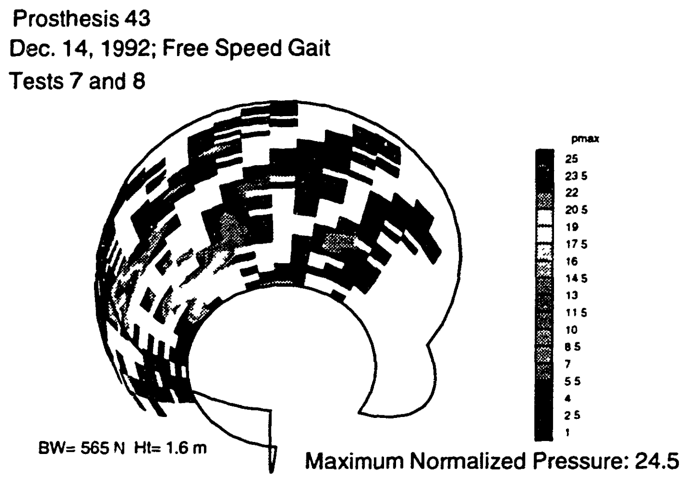
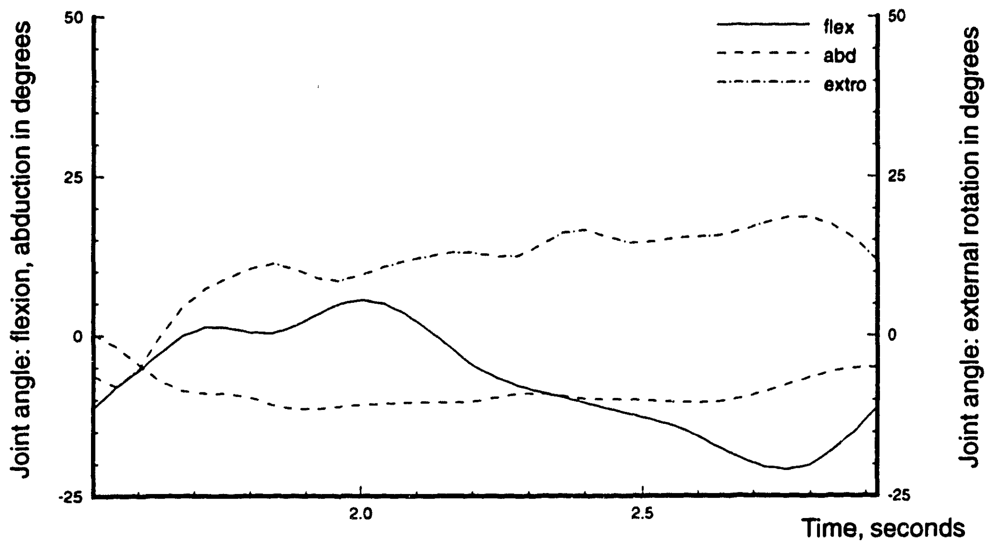


Figure 5-7: Regional Maximums for Subject 2: Free Speed Gait

Prosthesis 43; April 14, 1992; Test 22; Free Speed Gait



Prosthesis 43; Dec. 14, 1992; Test 8; Free Speed Gait

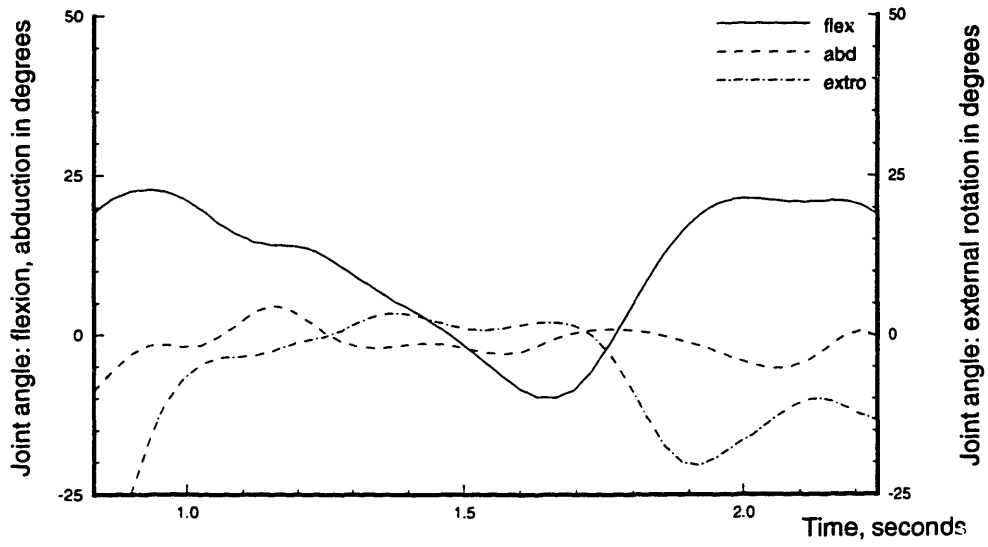
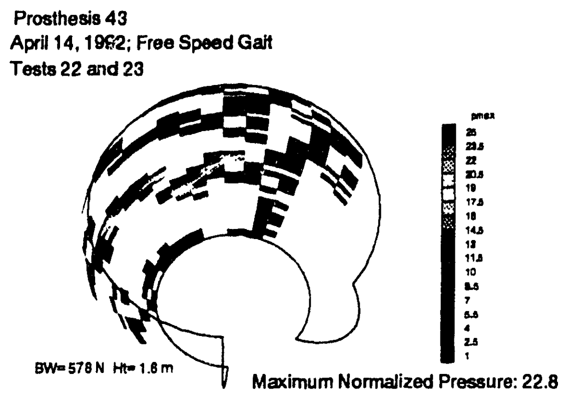


Figure 5-8: Joint Angles for Subject 2: Free Speed Gait



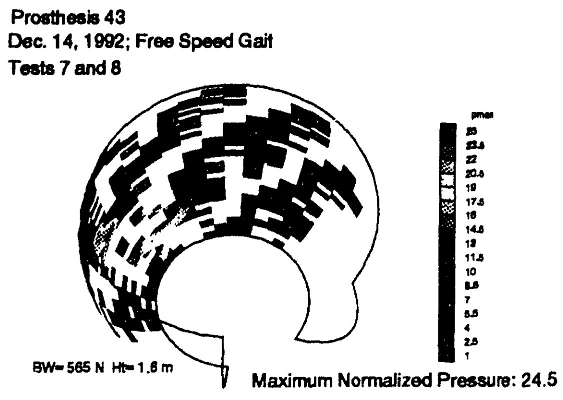


Figure 5-7 Regional Maximums for Subject 2; Free Speed Gait

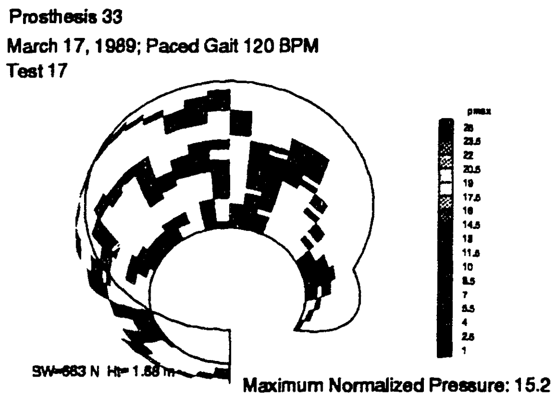


Figure 5-12; Regional Maximums for Subject 1; Walking at 120 BPM

5.1.3 Paced Gait: Prosthesis 33

Very few gait tests were done at specified paces for prosthesis 33. The data reported here is all from late in the implantation period. Figure 5-9 shows the local pressure magnitude in the joint during walking at 120 beats per minute (BPM). The pressures in this figure are normalized and smoothed. Figure 5-10 shows the corresponding forceplate forces during the same test ; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 1 for this subject. The joint angles for the test shown are presented in figure 5-11. Figure 5-15 presents the regional statistics for the same test.

The magnitude of the maximum normalized pressure for this activity was very similar to free speed walking. The only data for this movement was taken late in the implantation period, by which time the regions of high pressure had become more diffuse.

Figure 5-12 shows the local pressure magnitude in the joint during walking at 80 beats per minute (BPM). The pressures in this figure are normalized and smoothed. Figure 5-13 shows the corresponding forceplate forces during the same test; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 1 for this subject. The joint angles for the test shown are presented in figure 5-14. Figure 5-16 presents the regional statistics for the same test.

The results from this test are very similar to both free speed walking and 120 BPM gait for the day on which 80 BPM gait was taken. However, this data was taken late in the implantation period.

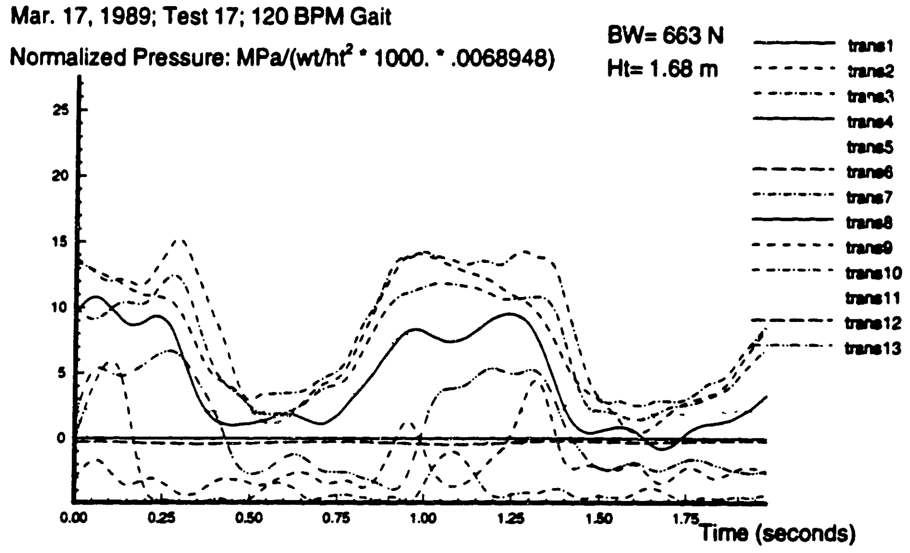


Figure 5-9: Pressures for su .ect 1: Walking at 120 BPM

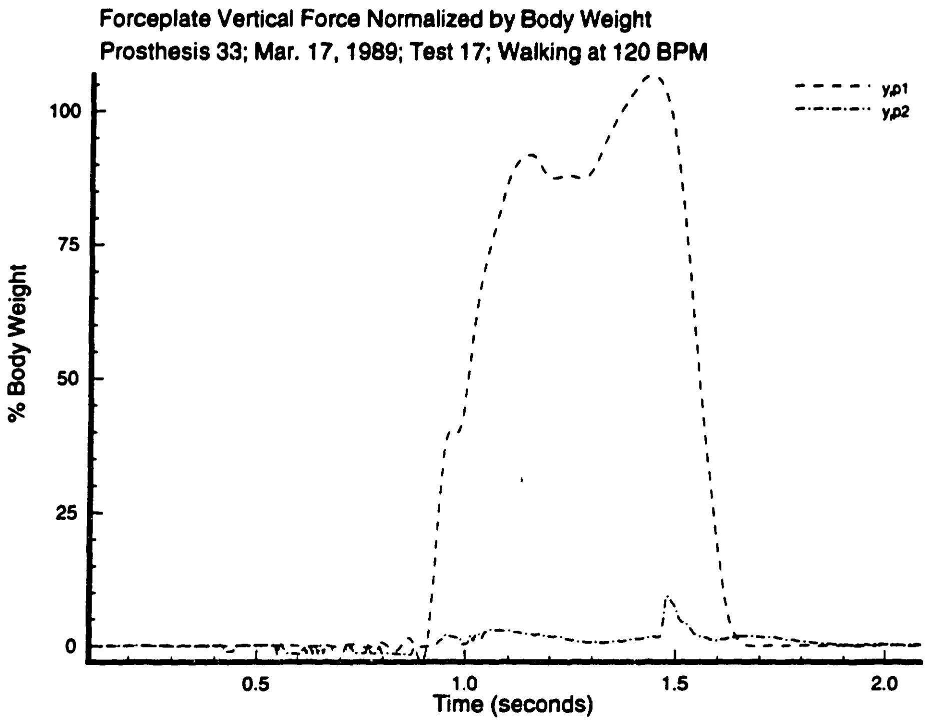


Figure 5-10: Forceplate data for subject 1: Walking at 120 BPM

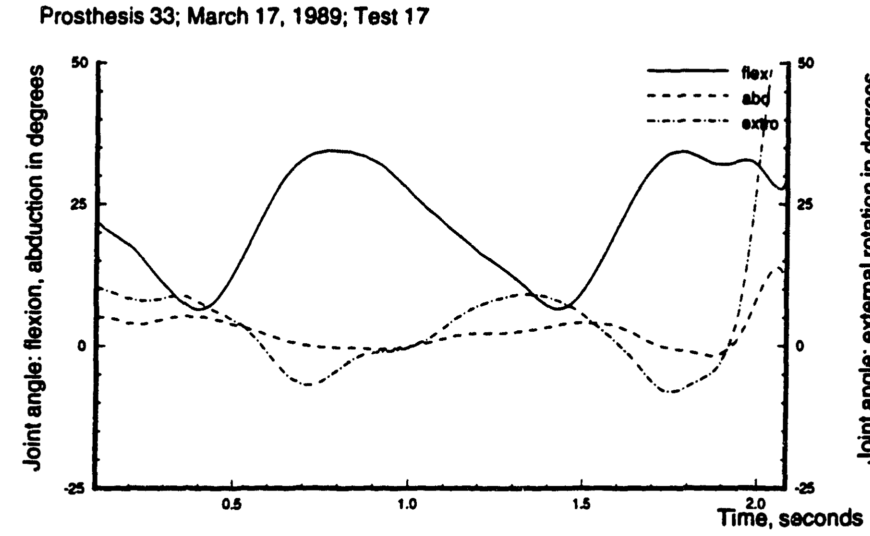


Figure 5-11: Joint Angles for Subject 1: Walking at 120 BPM

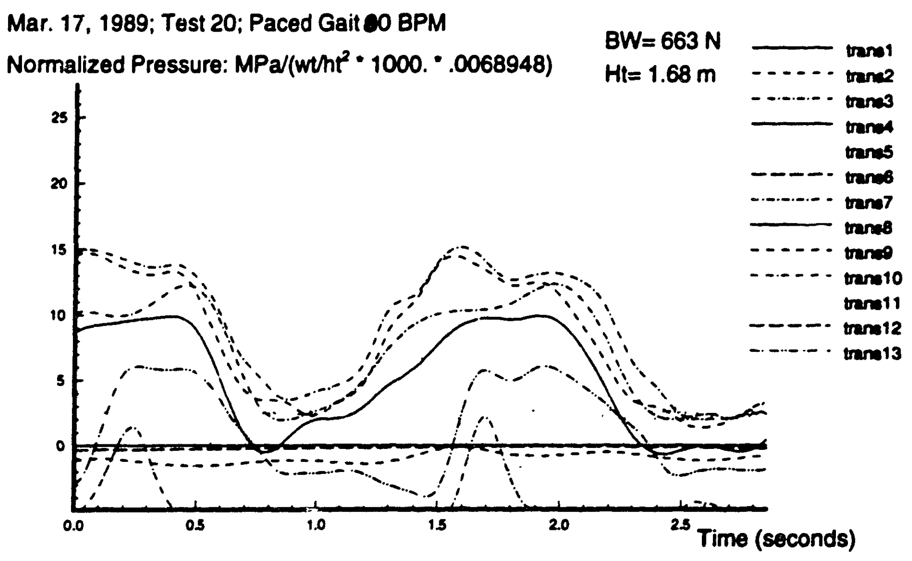


Figure 5-12: Pressures for subject 1: Walking at 80 BPM

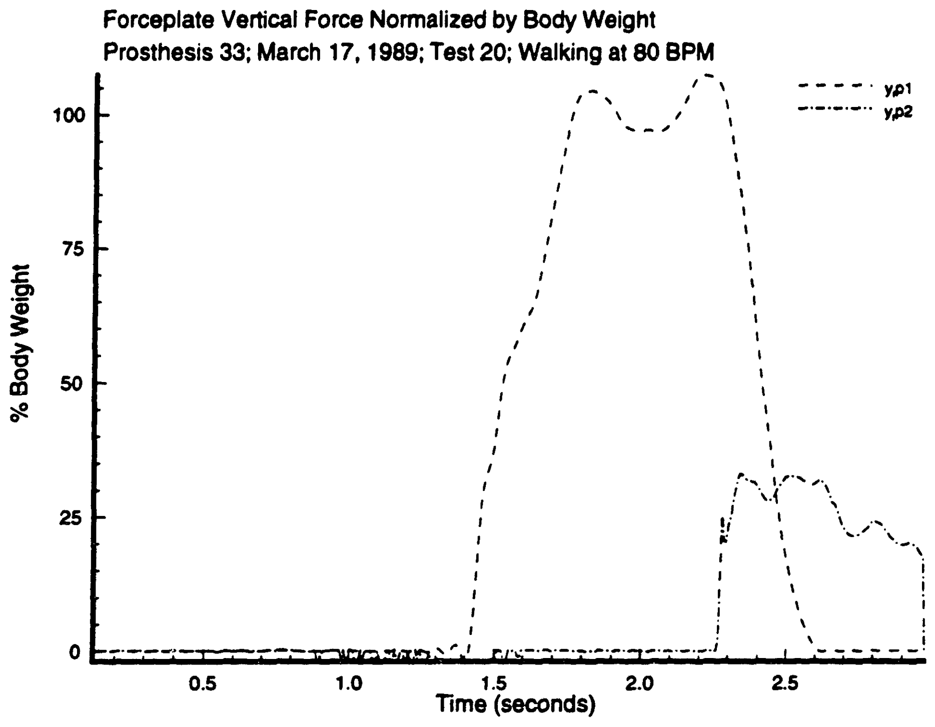


Figure 5-13: Forceplate data for subject 1: Walking at 80 BPM

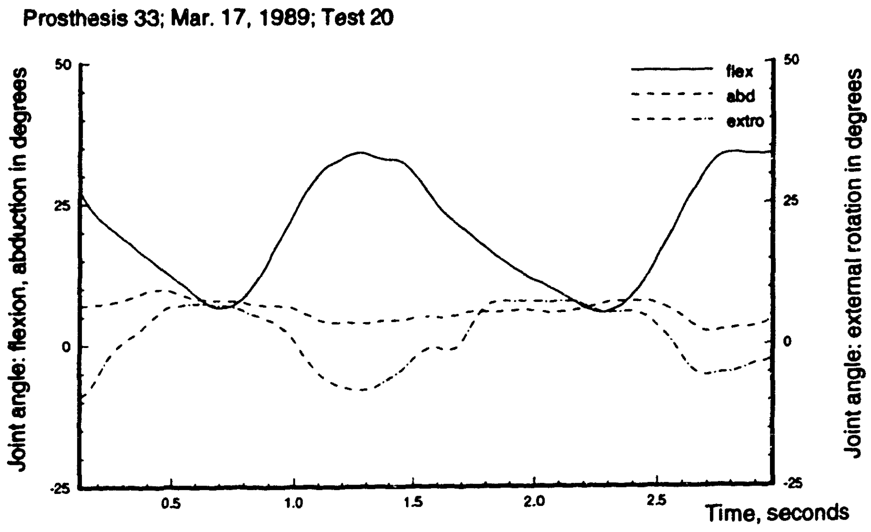


Figure 5-14: Joint Angles for Subject 1: Walking at 80 BPM

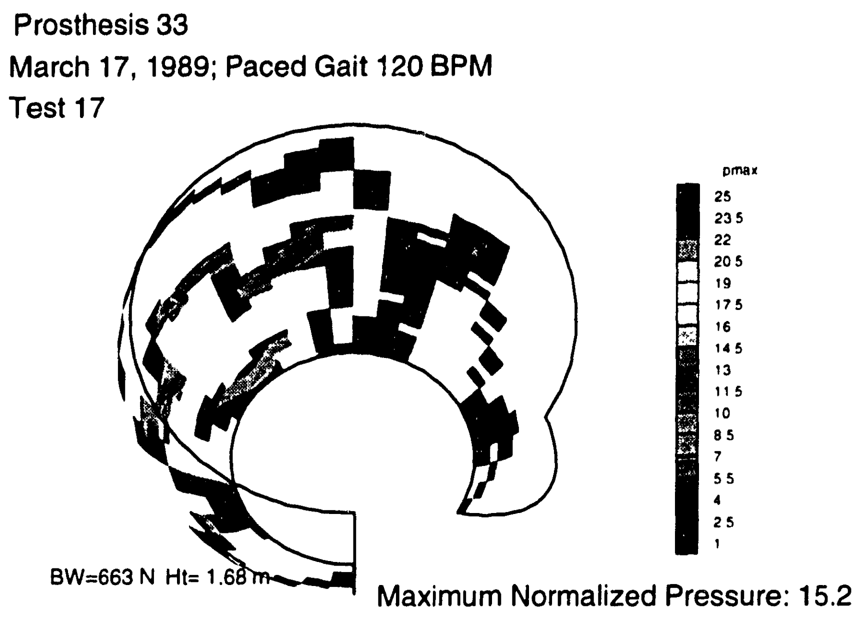


Figure 5-15: Regional Maximums for Subject 1: Walking at 120 BPM

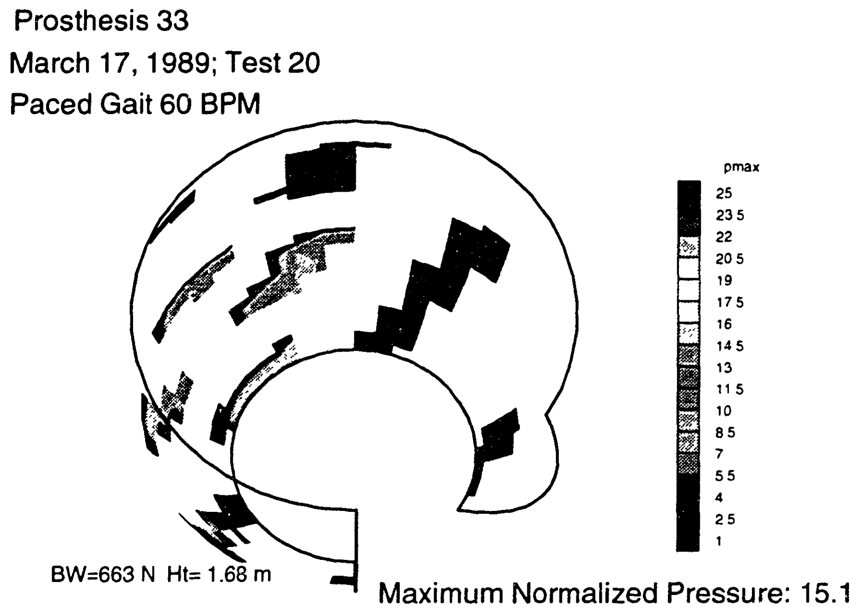
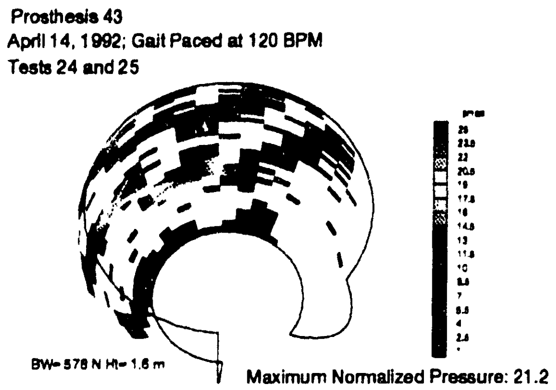


Figure 5-16: Regional Maximums for Subject 1: Walking at 80 BPM



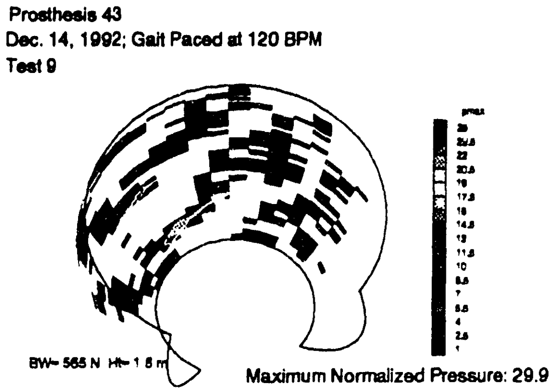


Figure 5-20: Regional Maximums for Subject 2; Walking at 120 BPM

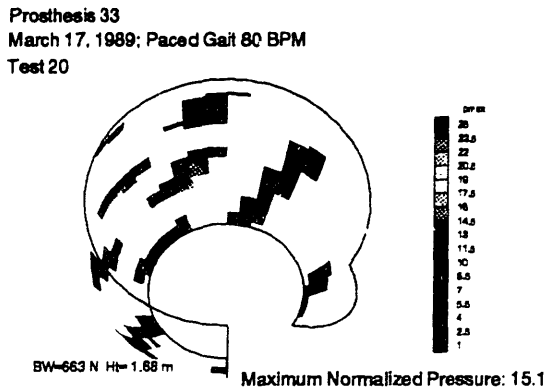


Figure 5-16: Regional Maximums for Subject 1; Walking at 80 BPM

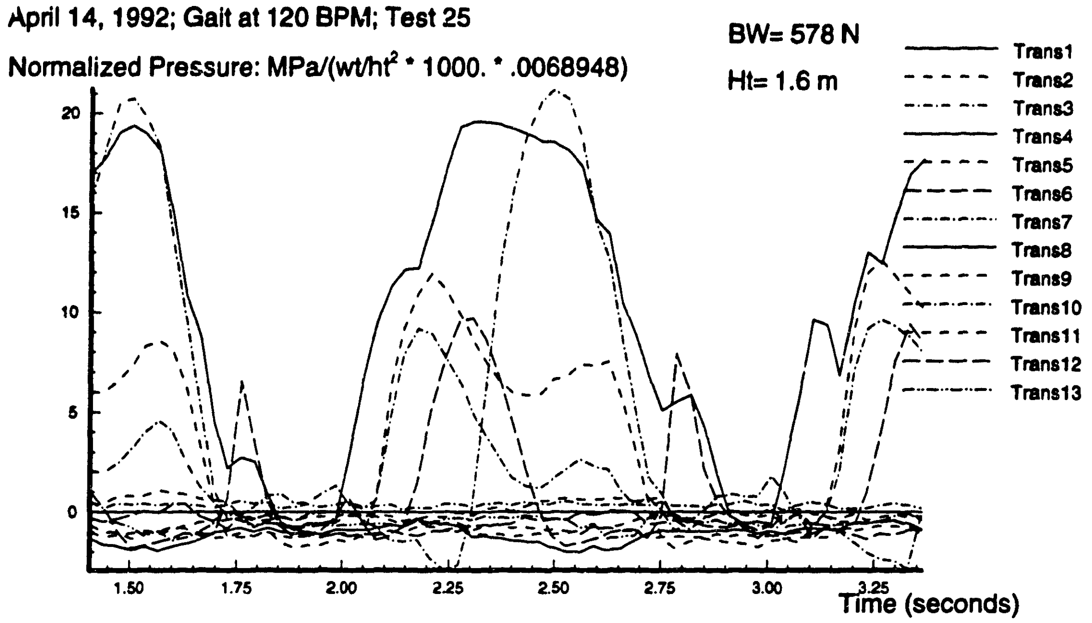
5.1.4 Paced Gait: Prosthesis 43

In contrast to the data from prosthesis 33, a metronome has been used frequently during data acquisition sessions with the second subject. However, the data is much earlier post-operative than is the paced gait data from the first subject and only reports the first year post-implantation. Figure 5-17 shows the local pressure magnitude in the joint during walking at 120 beats per minute (BPM). The pressures in this figure are normalized. Figure 5-18 shows the corresponding forceplate forces during the same set of tests; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject. The joint angles for the tests shown are presented in figure 5-19. Figure 5-20 presents the regional statistics for the same tests, when more than one test was available they were all used to compile regional statistics.

Higher pressures were measured during 120 BPM gait than during free speed gait for this subject, at a year post-operative. (At 4 months they were essentially the same.) The regions in which the highest pressures were measured are slightly different in that they are more localized for 120 BPM gait than for free speed gait.

Figure 5-21 shows the local pressure magnitude in the joint during walking at 60 beats per minute (BPM). The pressures in this figure are normalized. Figure 5-22 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject. The joint angles for the tests shown are presented in figure 5-23. Figure 5-24 presents the regional statistics for the same tests.

Higher pressures were measured during 60 BPM gait than during free speed gait for both days presented here. Pressures were also higher than those measured for 120 BPM gait on the same days. Although not presented here, the same trends in pressure were observed in data taken at 15 months post-operative. Data from a year post-operative is striking in how well-defined the area over which high pressures were measured.



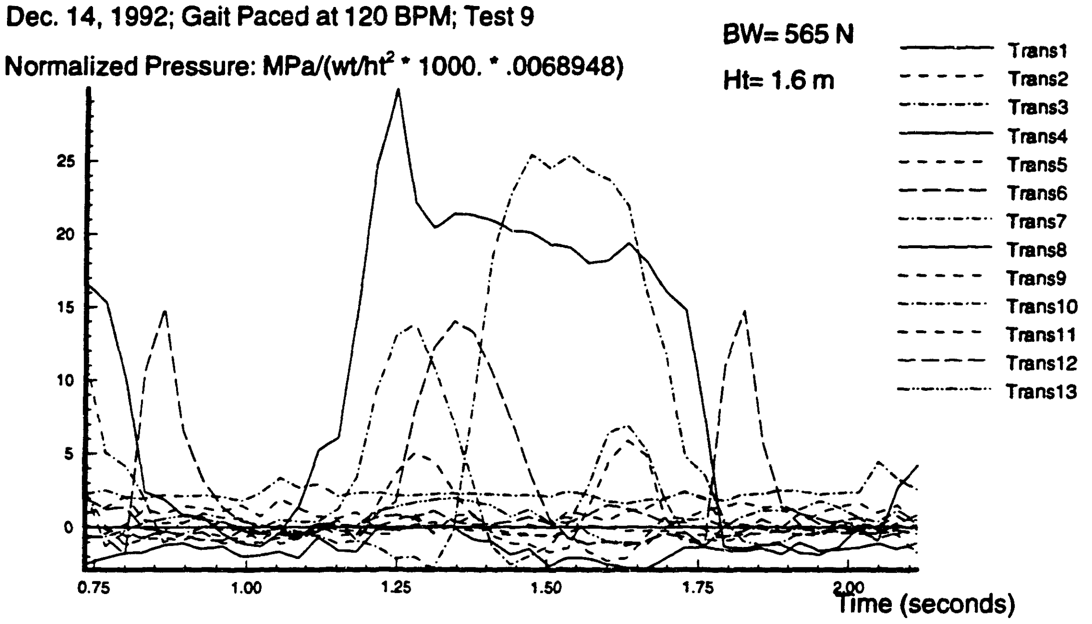
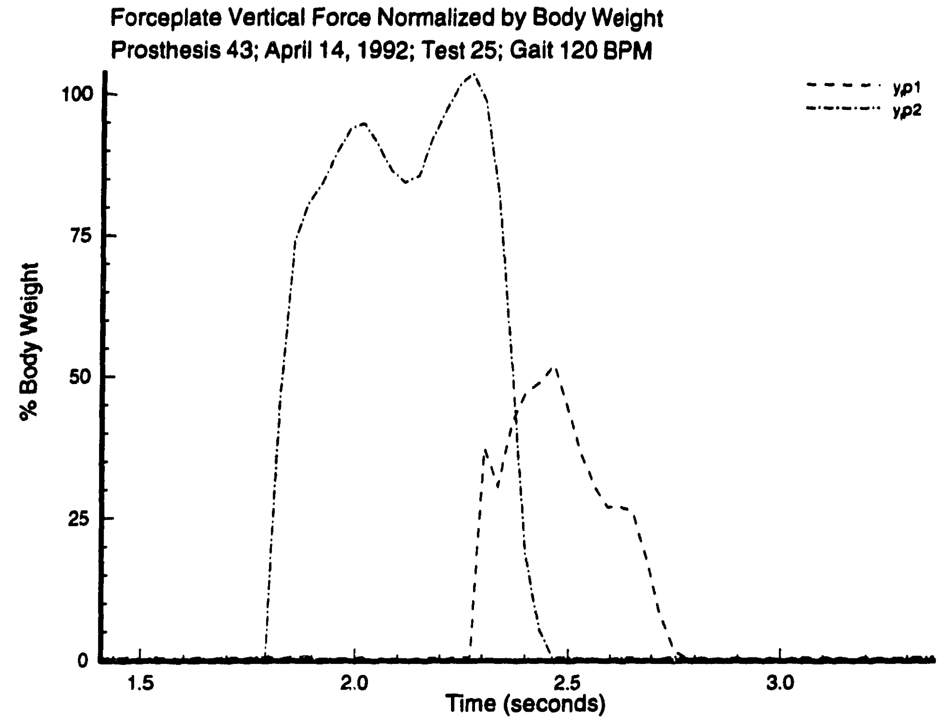


Figure 5-17: Pressures for subject 2: Walking at 120 BPM



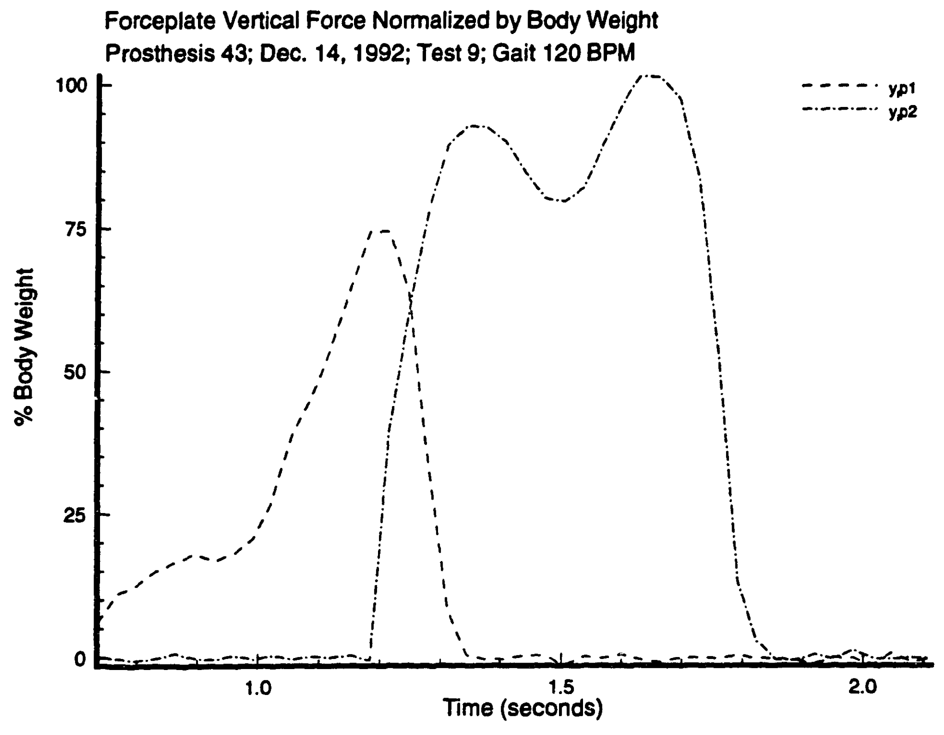
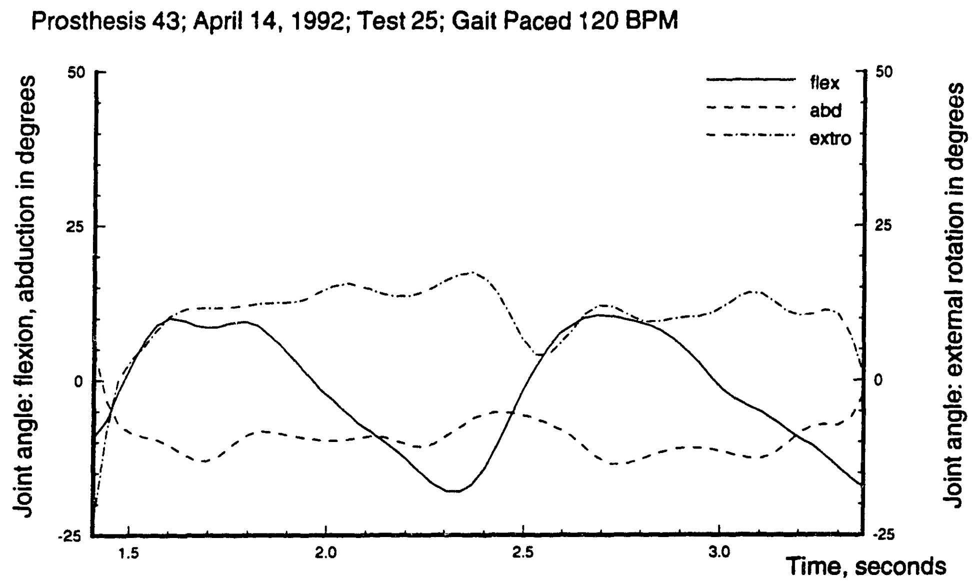


Figure 5-18: Forceplate data for subject 2: Walking at 120 BPM



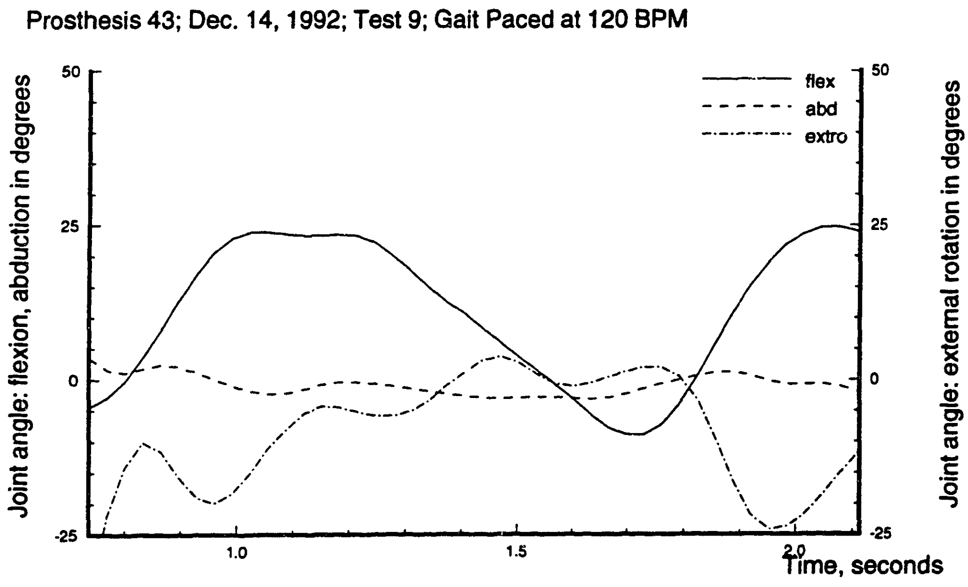
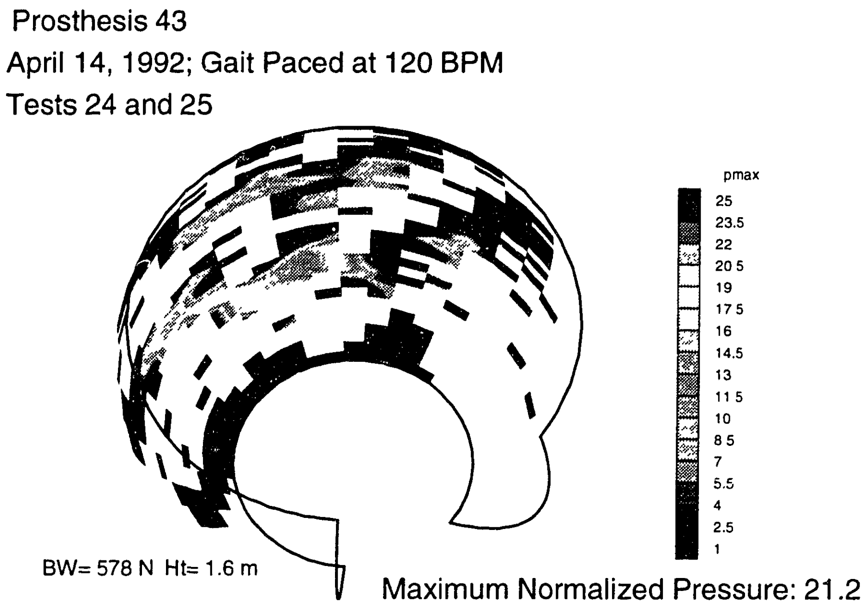


Figure 5-19: Joint Angles for Subject 2: Walking at 120 BPM



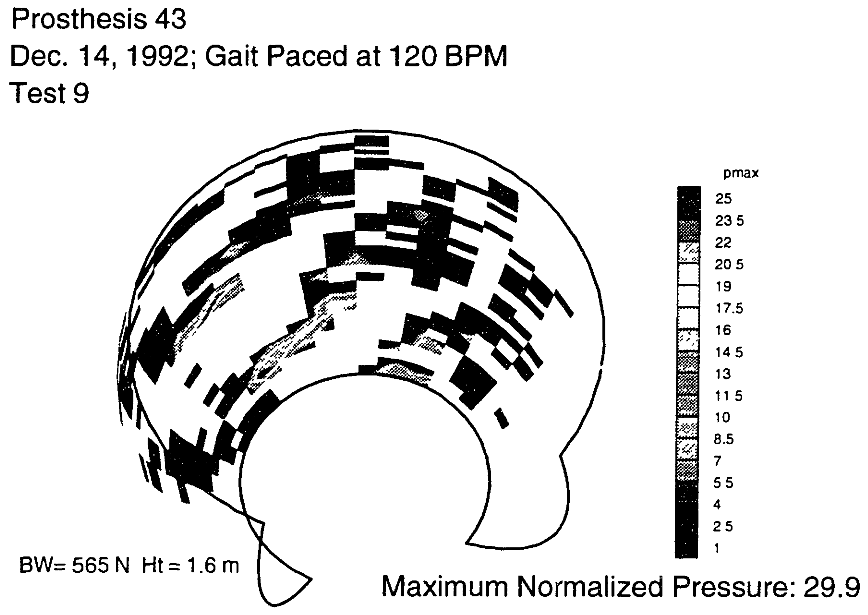
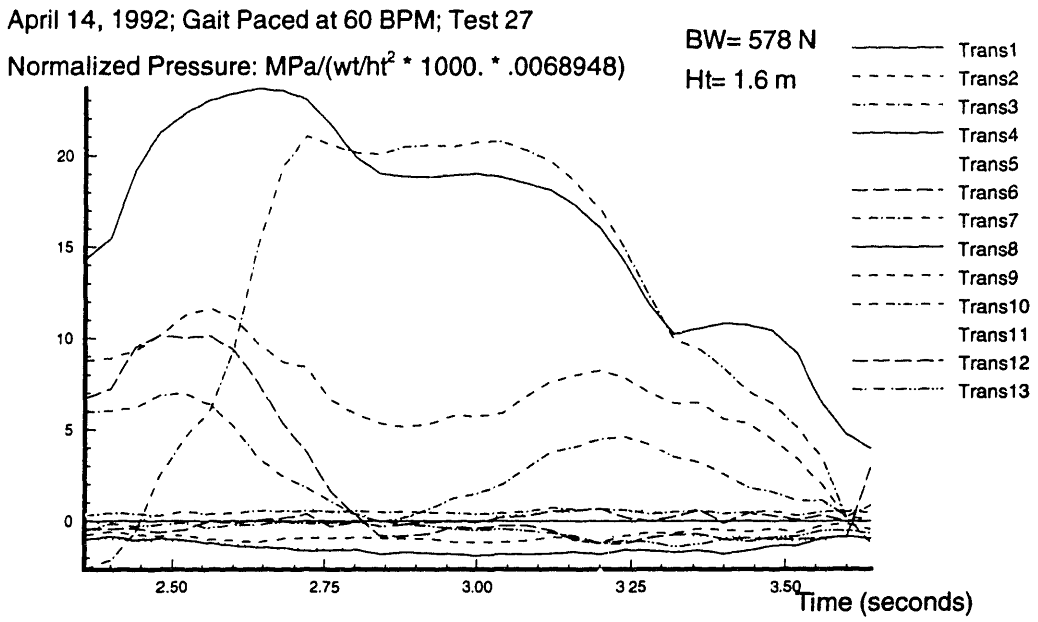


Figure 5-20: Regional Maximums for Subject 2: Walking at 120 BPM



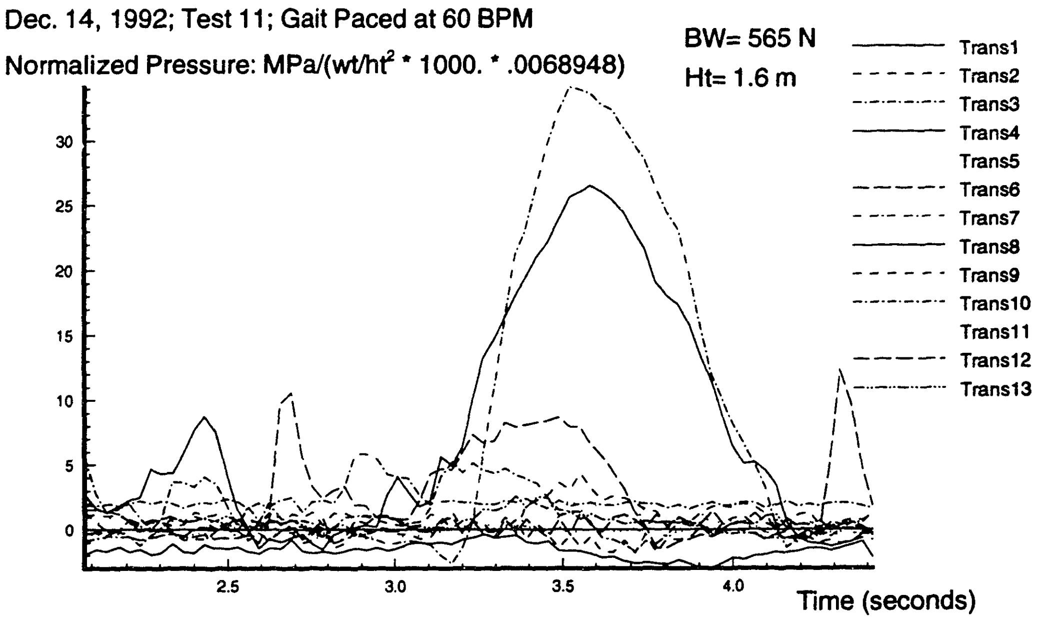
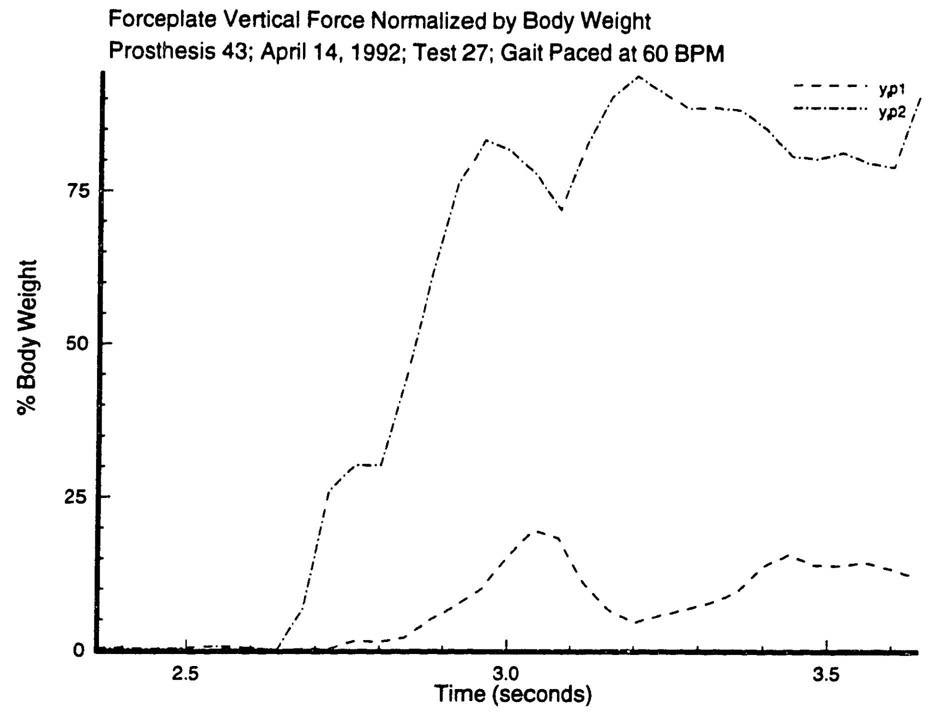


Figure 5-21: Pressures for subject 2: Walking at 60 BPM



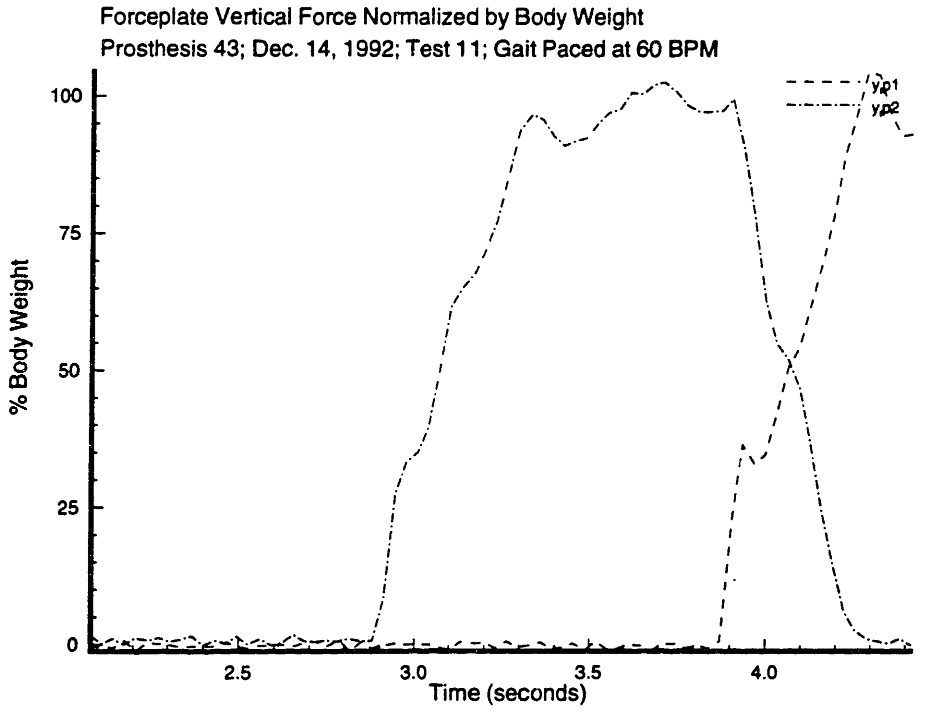
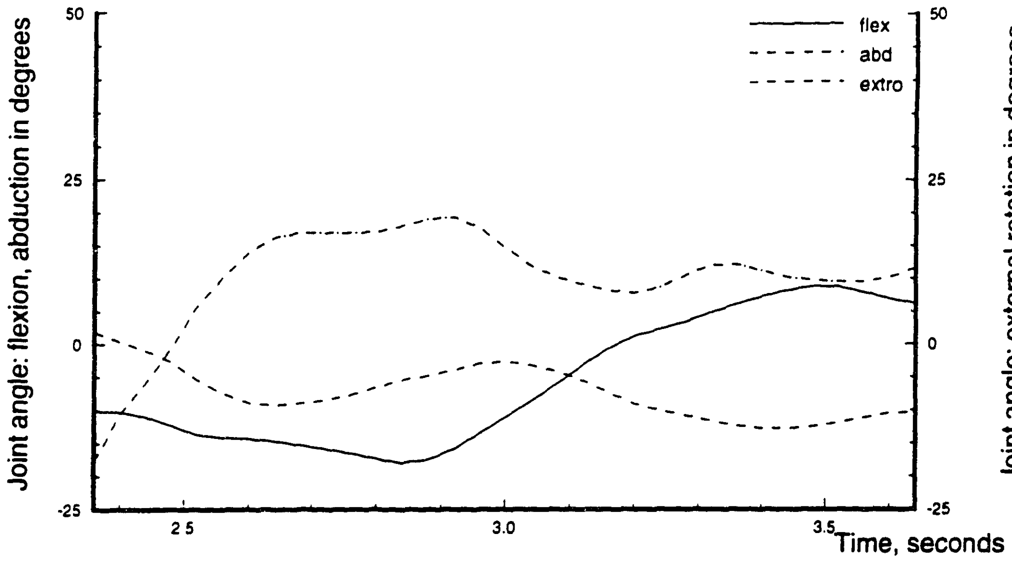


Figure 5-22: Forceplate data for subject 2: Walking at 60 BPM

Prosthesis 43; April 14, 1992; Gait Paced at 60 BPM; Test 27



Prosthesis 43; Dec. 14, 1992; Gait Paced at 60 BPM; Test 11

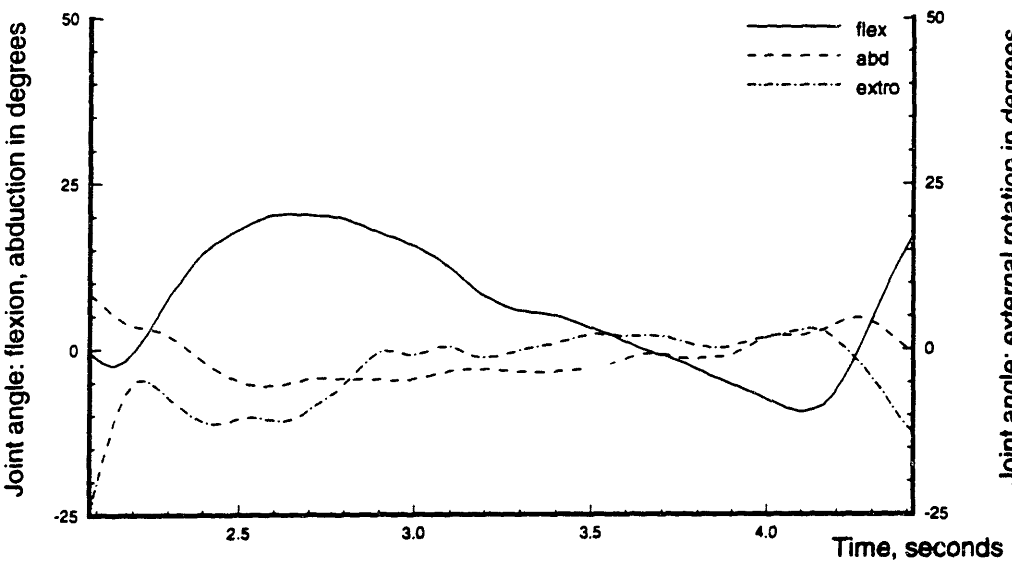
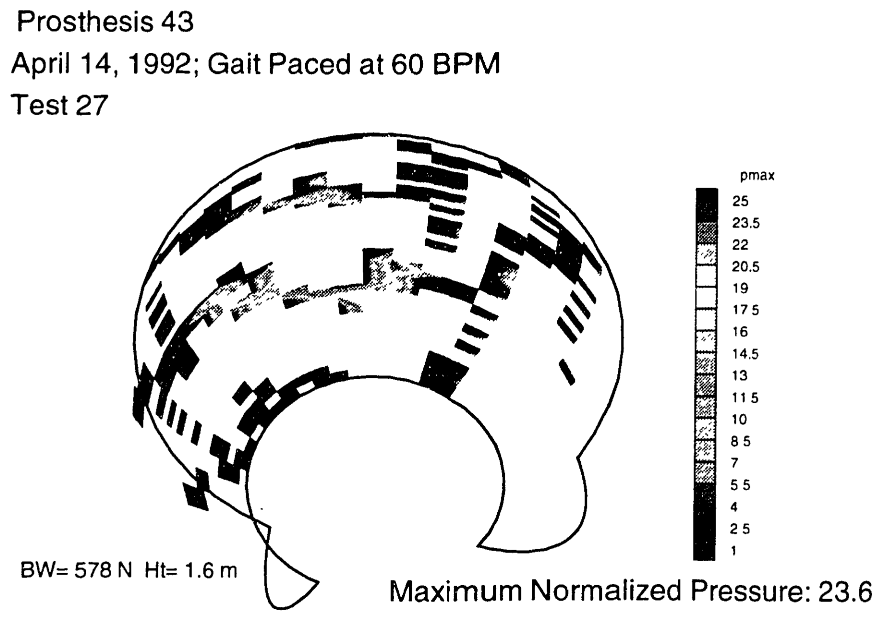


Figure 5-23: Joint Angles for Subject 2: Walking at 60 BPM



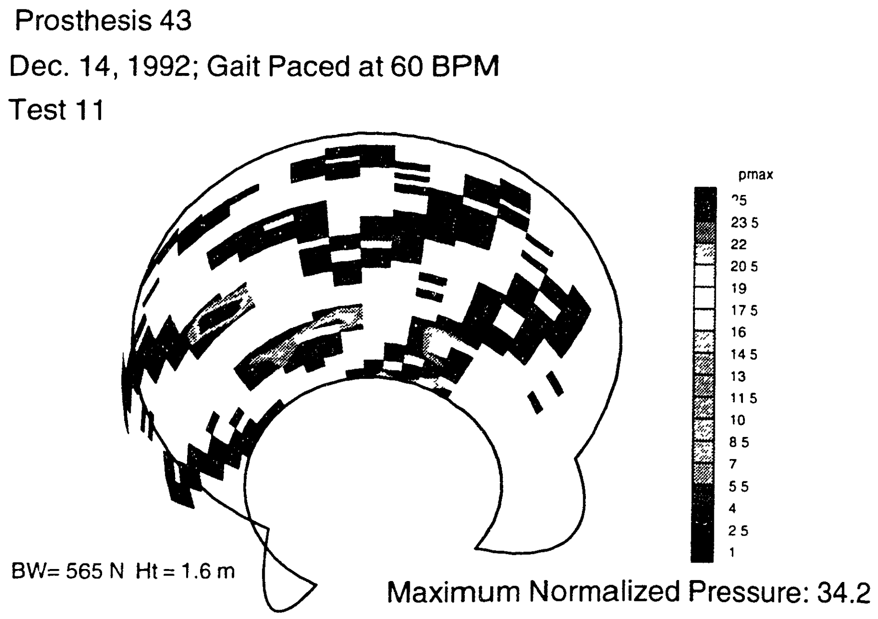


Figure 5-24: Regional Maximums for Subject 2: Walking at 60 BPM

5.2 Rising from a Chair

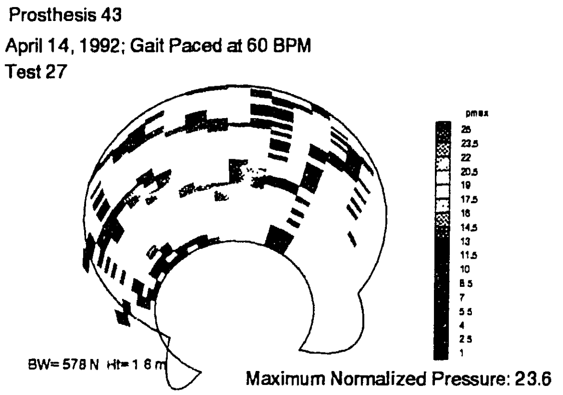
The motion of rising from a chair produced the highest pressure measurement recorded from the first subject, at 11 months post-operative. Following that measurement more attention was paid to this movement, and numerous tests were done, including a series with different height chairs. [9] Accordingly a number of tests have been done with the second subject rising from a seated position, however for this subject the maximum pressures recorded during this movement are much lower.

5.2.1 Prosthesis 33

The manner in which this movement was carried out by the first subject varied over the implantation period. Early on data was taken with the subject rising from an actual chair - sometimes a different chair than the one used the previous time data was taken. Eventually a set of blocks was built which could be arranged into different height seats or into a set of stairs, to standardize these test conditions. In general the instruction to the subject was to cross her arms in front of her body during the movement.

Figure 5-25 shows the local pressure magnitude in the joint during rising from a chair. The pressures in this figure are normalized and smoothed. Figure 5-26 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. In general the force on the instrumented leg is measured at forceplate 1 for this subject. The positioning of the chair or blocks relative to the forceplate evolved, so that during some tests only one foot was on a forceplate while during other tests one foot was on each forceplate. Also, the subject's feet may not have been entirely on the forceplates during some tests, this appears to be the case in Oct. 1986. The joint angles for the tests shown are presented in figure 5-27. These plots indicate differences in chair height, as the hip flexion angle ranged from 65 to 80 degrees for the tests shown. Figure 5-28 presents the regional statistics for the same tests.

Whatever the possible effect of chair height variation on the pressures, the maxi-



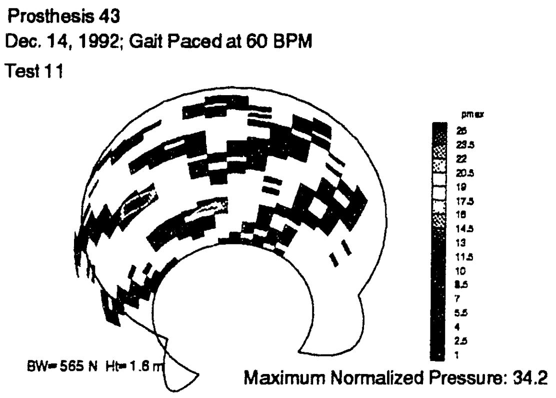


Figure 5-24: Regional Maximums for Subject 2; Walking at 60 BPM

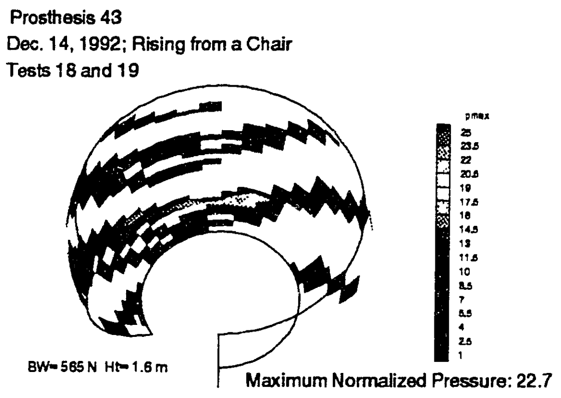


Figure 5-32: Regional Maximums for Subject 2; Rising from a Chair

mum pressures measured in May 1985 were more than double those measured during this movement in 1986. Although not all of the data has been presented here, pressure measurements during rising from a chair in August 1985 were nearly as high as those made in May 1985. In all tests it is clear that this subject initiated the movement by bringing her upper body forward (the flexion angle increases significantly) prior to rising. The time at which the maximum pressure occurred varies relative to the timing of the movement. For instance, in Dec. 1984 the maximum pressure was measured after the maximum flexion angle, while in May 1985 and Oct. and Dec. 1986 the maximum pressure was nearly coincident with the maximum flexion angle. The regions of highest pressure were located posterior on the acetabulum in all cases.

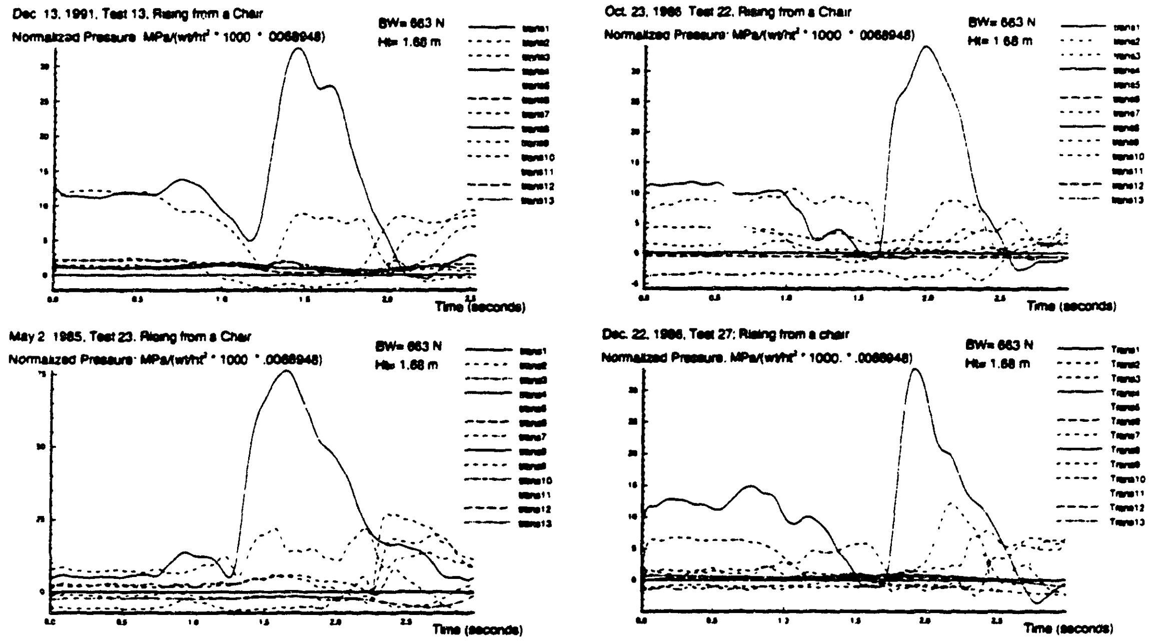


Figure 5-25: Pressures for subject 1: Rising from a chair

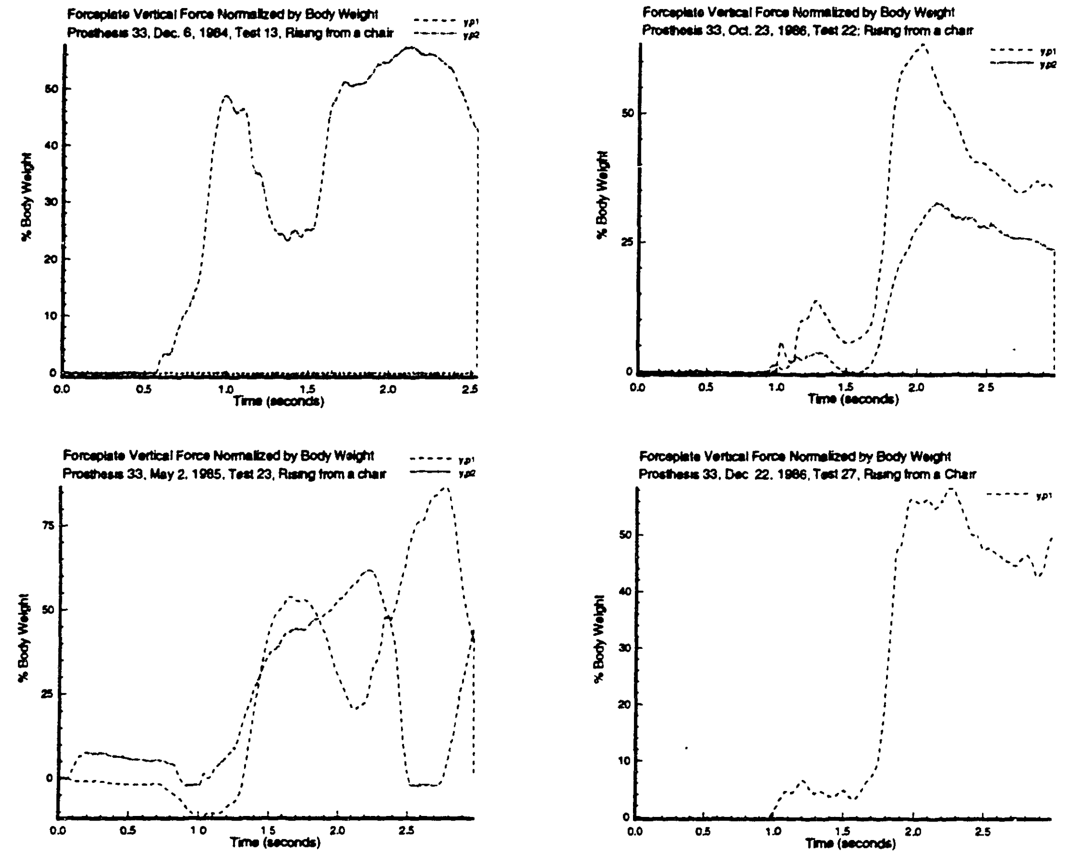


Figure 5-26: Forceplate data for subject 1: Rising from a chair

Data was taken for rising from different height chairs, this is presented and discussed in [9]. The results essentially are that higher local pressures are measured the lower the chair height.

5.2.2 Prosthesis 43

For this subject greater uniformity has been imposed in the manner in which this rising from a chair test has been conducted, although there are fewer days for which kinematic data is available. The standard test is rising from a seated position without pacing, with the seat at 100% of knee height. Tests have also been done in which the seat height was changed to 150% of knee height and 80% of knee height. The subject puts his hands on his thighs at the start of the test.

Figure 5-29 shows the local pressure magnitude in the joint during rising from a seat at 100% knee height without pacing. The pressures in this figure are normalized. Figure 5-30 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject. The joint angles for the tests shown are presented in figure 5-31. Figure 5-32 presents the regional statistics for the same tests.

In a number of respects this data is similar to that from the first subject; the chair height is similar as shown by a starting flexion angle of about 74 degrees, the subject also initiates this movement by bringing his upper body forward, and the point in the movement at which the maximum pressure is measured is similar to the results for the first subject at 6 months post-operative. However, the maximum pressures measured were much lower for the second subject and positioned more anterior than the high stresses measured for the first subject.

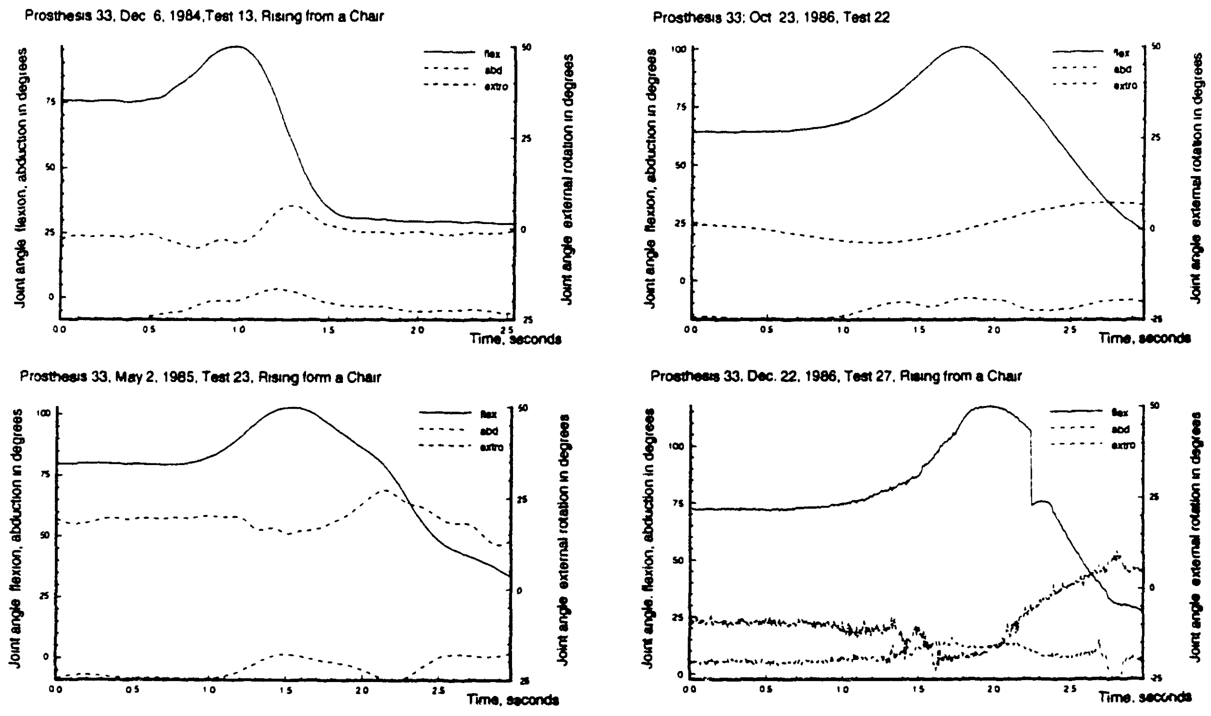


Figure 5-27: Joint Angles for Subject 1: Rising from a chair

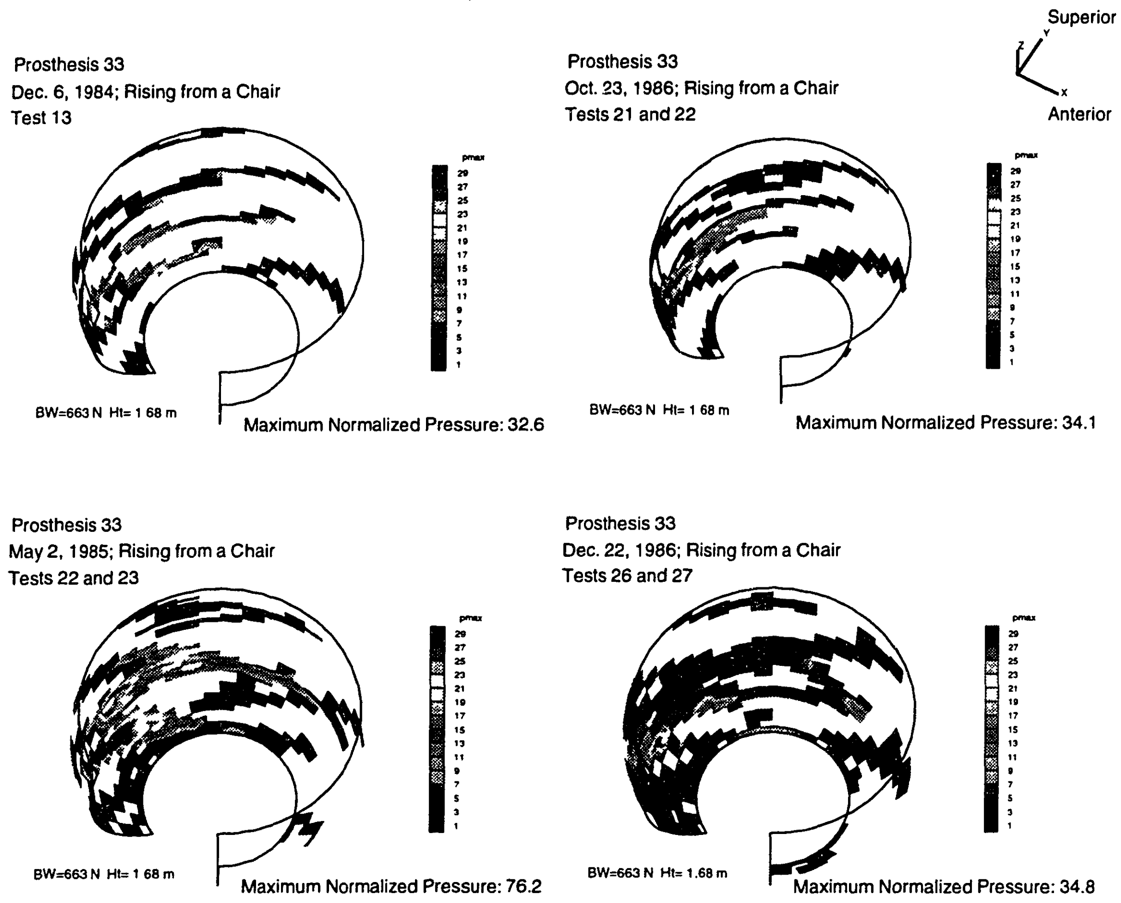


Figure 5-28: Regional Maximums for Subject 1: Rising from a chair

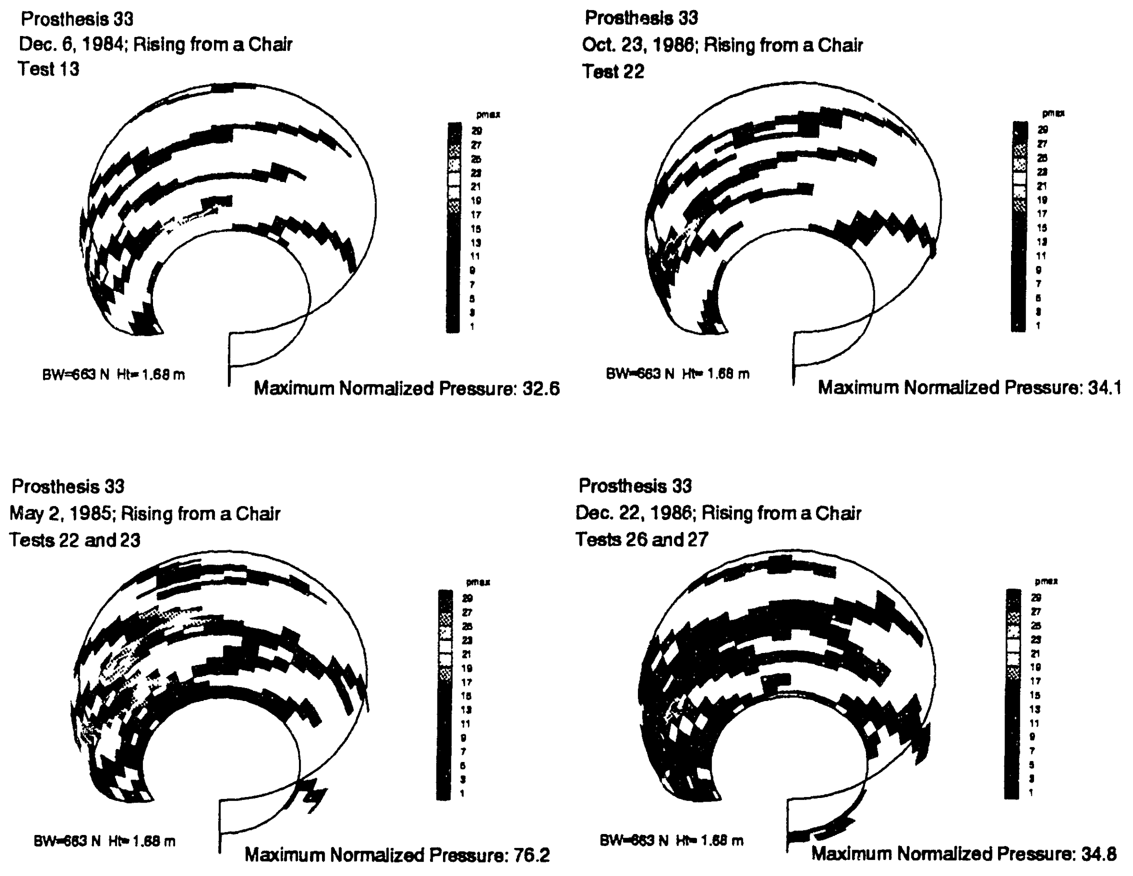


Figure 5-28: Regional Maximums for Subject 1; Rising from a Chair

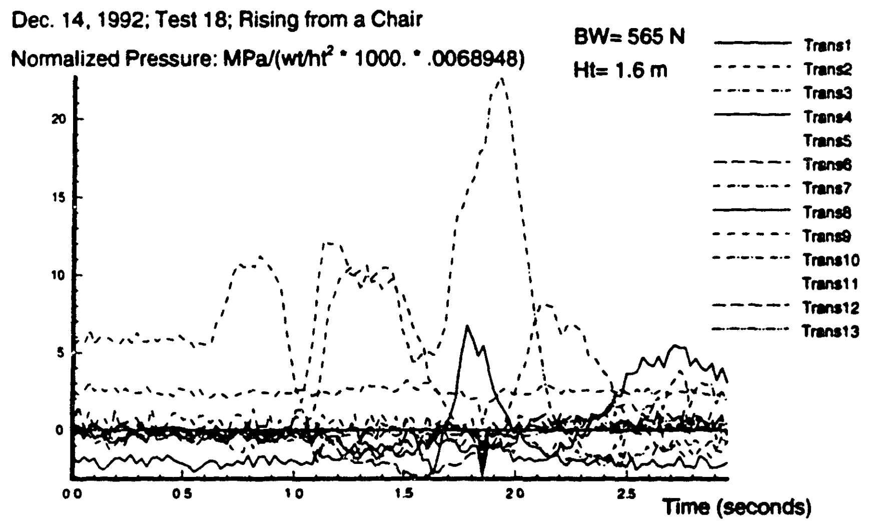


Figure 5-29: Pressures for subject 2: Rising from a chair

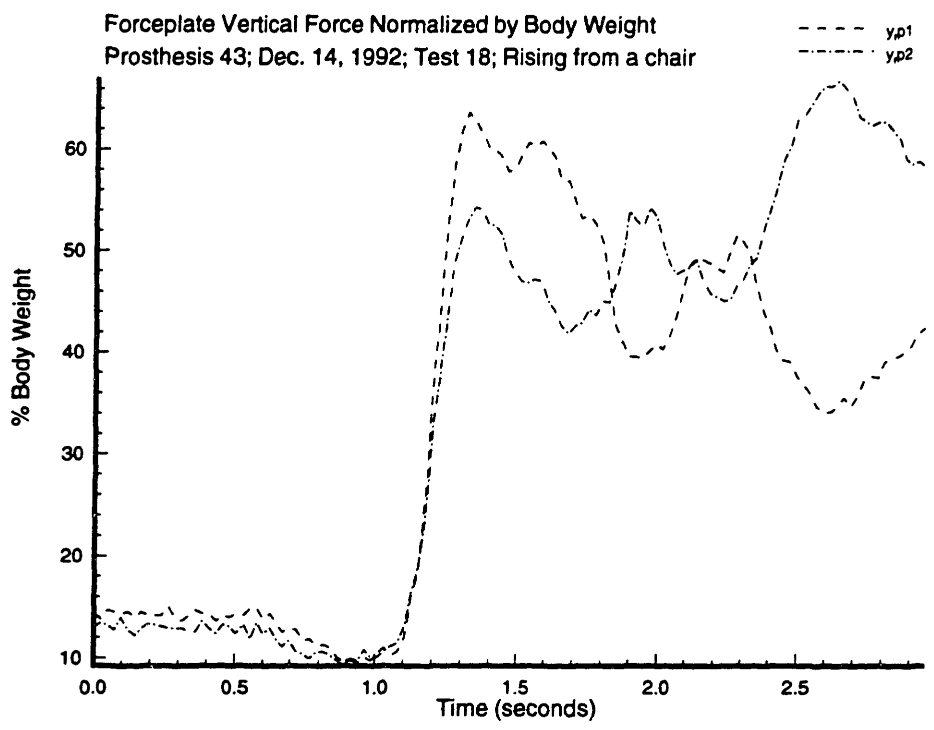


Figure 5-30: Forceplate data for subject 2: Rising from a chair

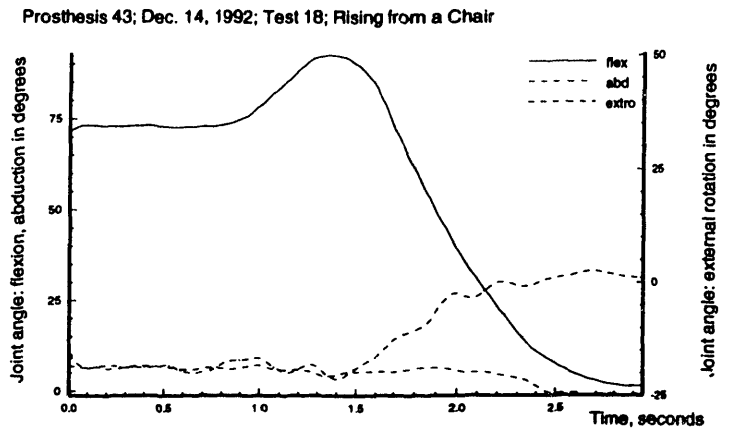


Figure 5-31: Joint Angles for Subject 2: Rising from a chair

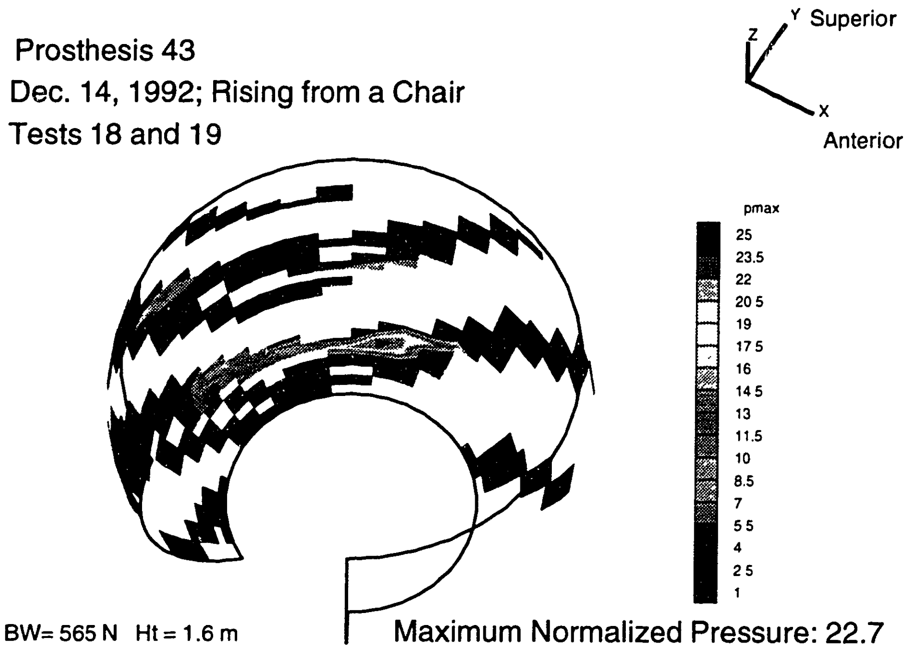


Figure 5-32: Regional Maximums for Subject 2: Rising from a chair

5.3 Stairs

Climbing and descending stairs are also movements performed frequently in daily life which were found to generate locally high pressures for the first subject. Thus far these tests have resulted in higher local pressures for the second subject than for the first, both with and without normalization. The pressures measured in the second subject during stair climbing and descending have also been higher than those found in rising from a chair for this subject. The second subject has expressed difficulty in maintaining his balance, this may contribute to the higher pressure measurements during stair climbing and descending, as the subject may be co-contracting massively. For the second subject descending stairs resulted in higher pressure measurements than did ascending stairs.

5.3.1 Prosthesis 33

The protocol for this movement varied over the implantation period, early on there was only one step, 9" high, later a set of blocks that could be arranged into a set of stairs with 3 steps was built. No armrails were provided during any test. Results for both ascending and descending stairs are presented. Figure 5-33 shows the local pressure magnitude in the joint during ascending stairs. The pressures in this figure are normalized and smoothed. Figure 5-34 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. However, the steps used were not always the same and were not always placed on the forceplates in the same manner. For instance in May 1985 the only step was on forceplate 1. By Dec. 1986 a set of stairs had been constructed and the two forceplate measurements were independant. The joint angles for the tests shown are presented in figure 5-35. Figure 5-36 presents the regional statistics for the same tests.

The pressure vs. time data for climbing stairs is quite complex and confusing, for both this subject and the second subject. The maximum pressure was generally measured during the initial loading of the foot during each step for this subject. The regions in which the highest pressures were measured were more anterior than for rising from a chair, but more posterior than in normal gait.

Figure 5-37 shows the local pressure magnitude in the joint during descending stairs. The pressures in this figure are normalized and smoothed. Figure 5-38 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject on this date. The joint angles for the tests shown are presented in figure 5-39. Figure 5-40 presents the regional statistics for the same tests.

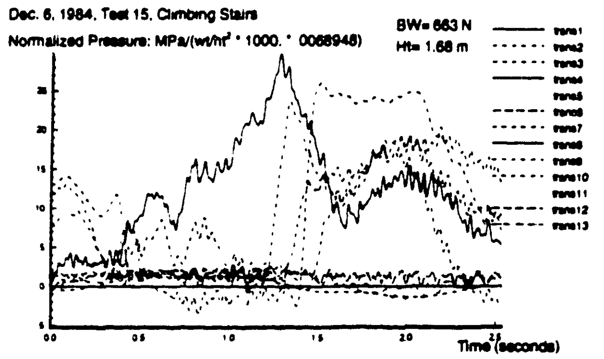
The pressures measured during this movement were not as high as in climbing stairs for this subject and, despite occuring during a period of high hip flexion, were more anterior on the acetabulum than for climbing stairs, suggesting that muscle action influences area of acetabular loading.

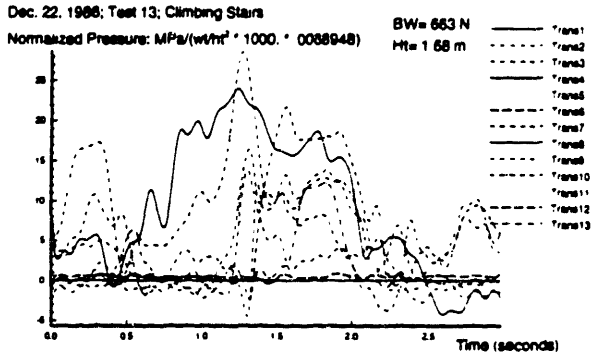
5.3.2 Prosthesis 43

This subject has always used the same set of stairs - three steps each 9" high. The stairs have been turned around for descending stairs, so that forceplate 2 always registers force from the instrumented leg. Data has been taken for both paced (at 80 BPM) and unpaced stair-climbing. No hand rails are used.

Figure 5-41 shows the local pressure magnitude in the joint during stair climbing with and without pacing. The pressures in this figure are normalized. Figure 5-42 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. The joint angles for the tests shown are presented in figure 5-43. Figure 5-44 presents the regional statistics for the same tests.

Climbing and descending stairs have resulted in the highest recorded pressures from this subject, except for tests in which he jumps off a step. High pressures have been recorded at just about all points in the gait cycle during stair climbing, though the highest pressures appear just as the leg is being raised for the next step. Regions of high stress are located both posterior and superior in the acetabulum. This





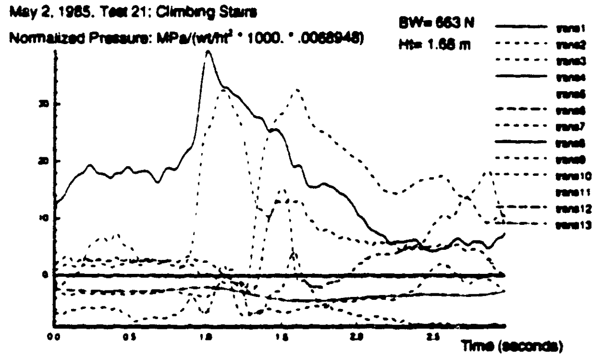
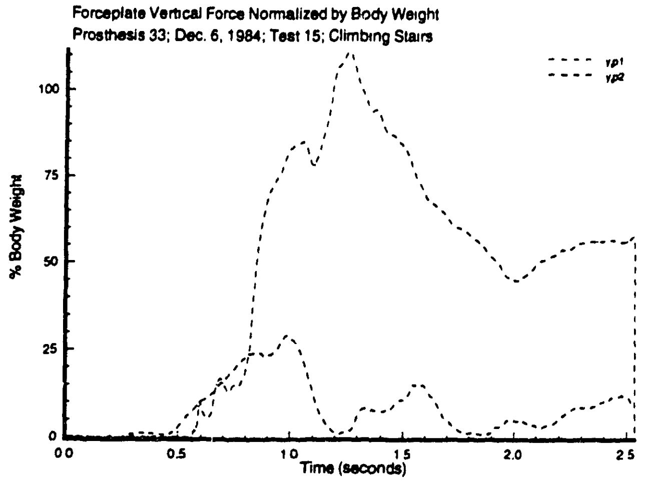
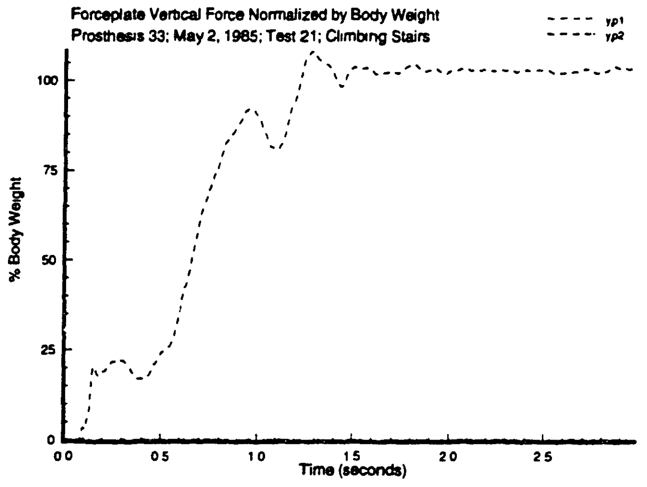


Figure 5-33: Pressures for subject 1: Ascending Stairs





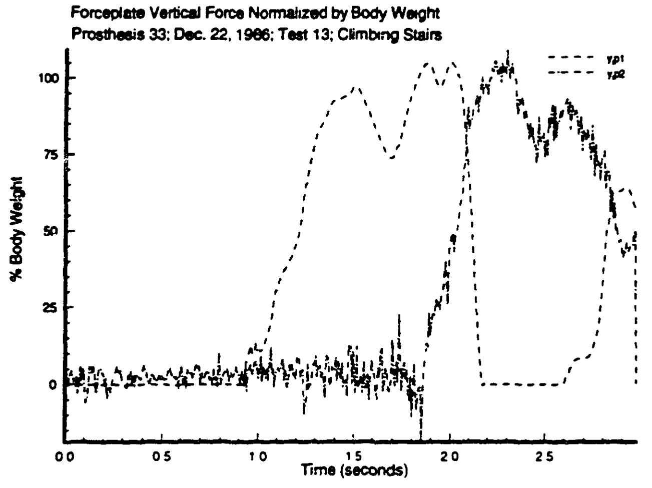


Figure 5-34: Forceplate data for subject 1: Ascending Stairs

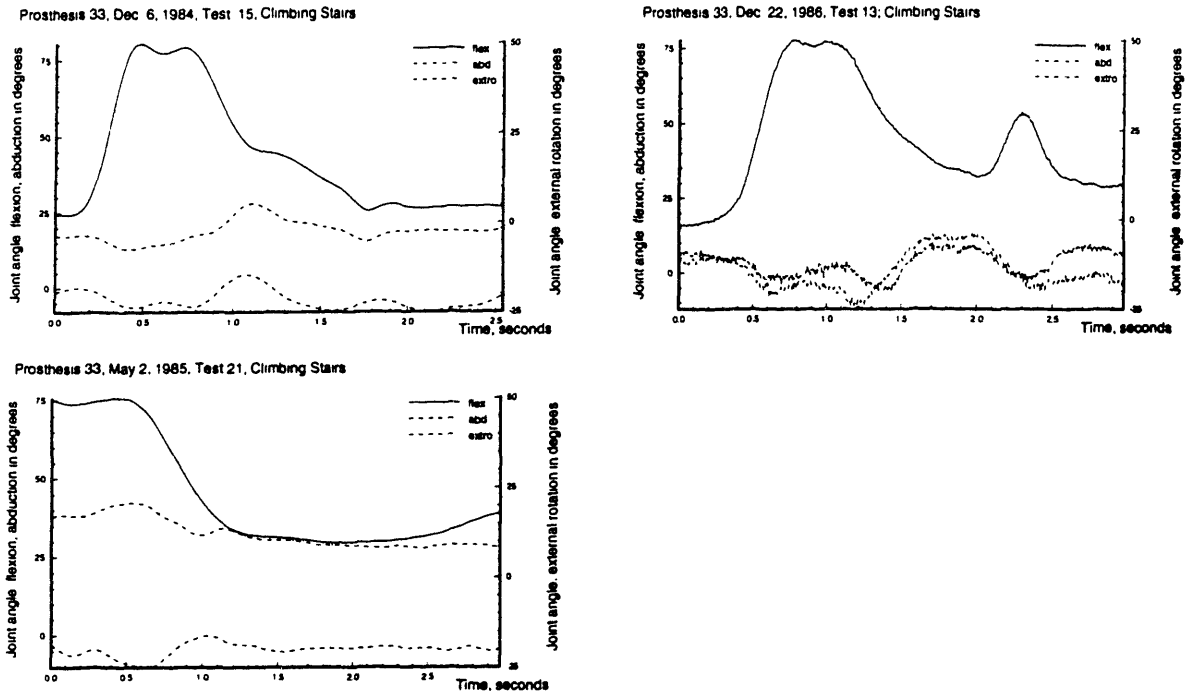


Figure 5-35: Joint Angles for Subject 1: Ascending Stairs

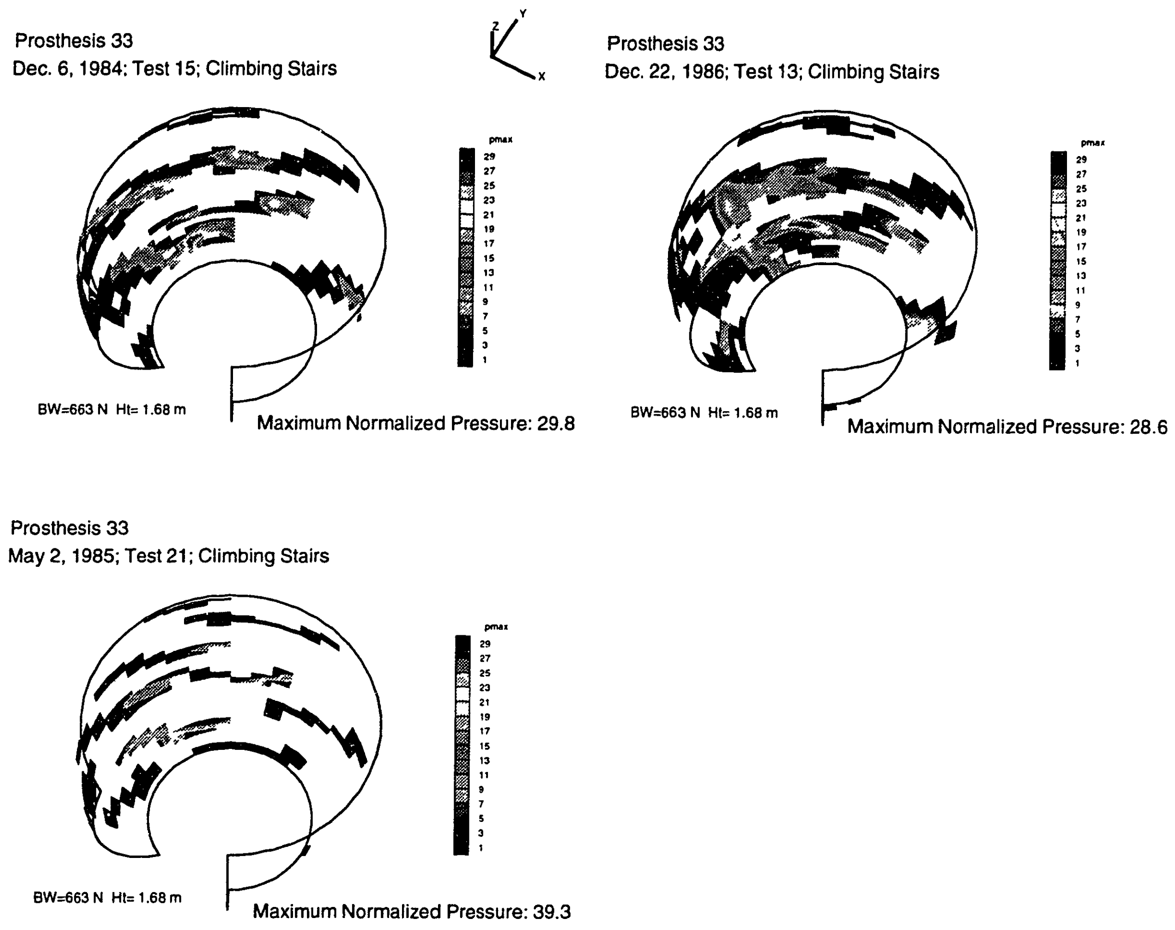
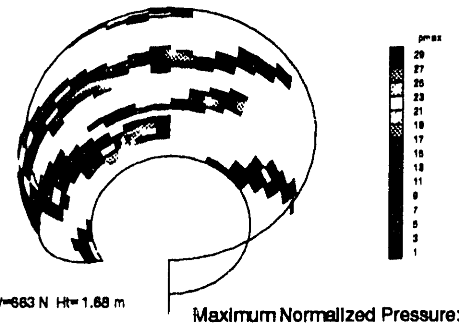
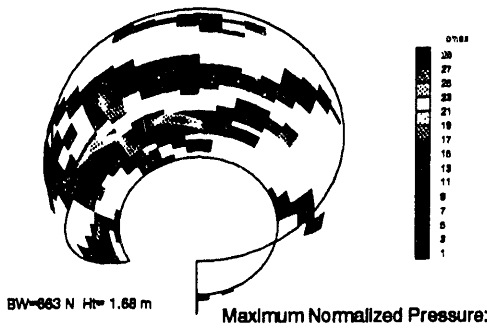


Figure 5-36: Regional Maximums for Subject 1: Ascending Stairs

Prosthesis 33
Dec. 6, 1984; Test 15; Climbing Stairs



Prosthesis 33
Dec. 22, 1986; Test 13; Climbing Stairs



Prosthesis 33
May 2, 1985; Test 21; Climbing Stairs

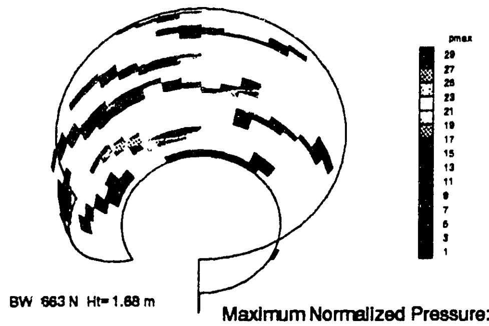


Figure 5-36: Regional Maximums for Subject 1; Ascending Stairs

Prosthesis 33
Dec. 6, 1984; Test 17; Descending Stairs

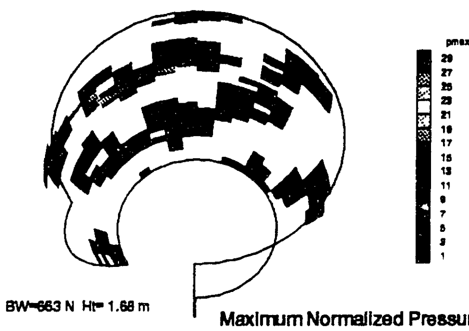


Figure 5-40: Regional Maximums for Subject 1; Descending Stairs

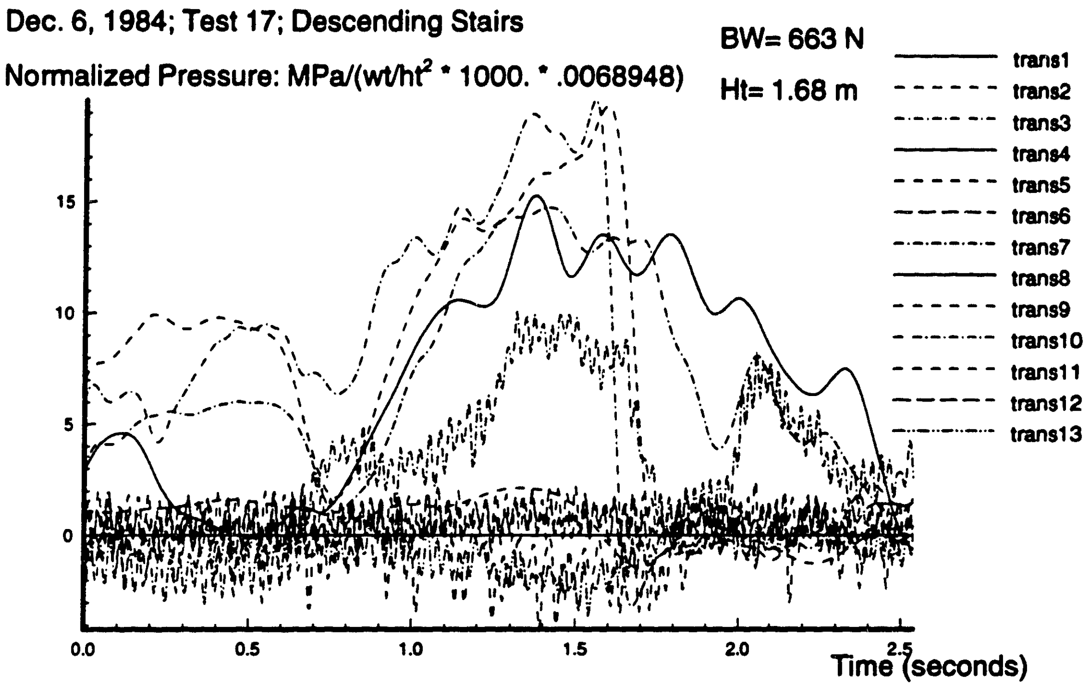


Figure 5-37: Pressures for subject 1: Descending Stairs

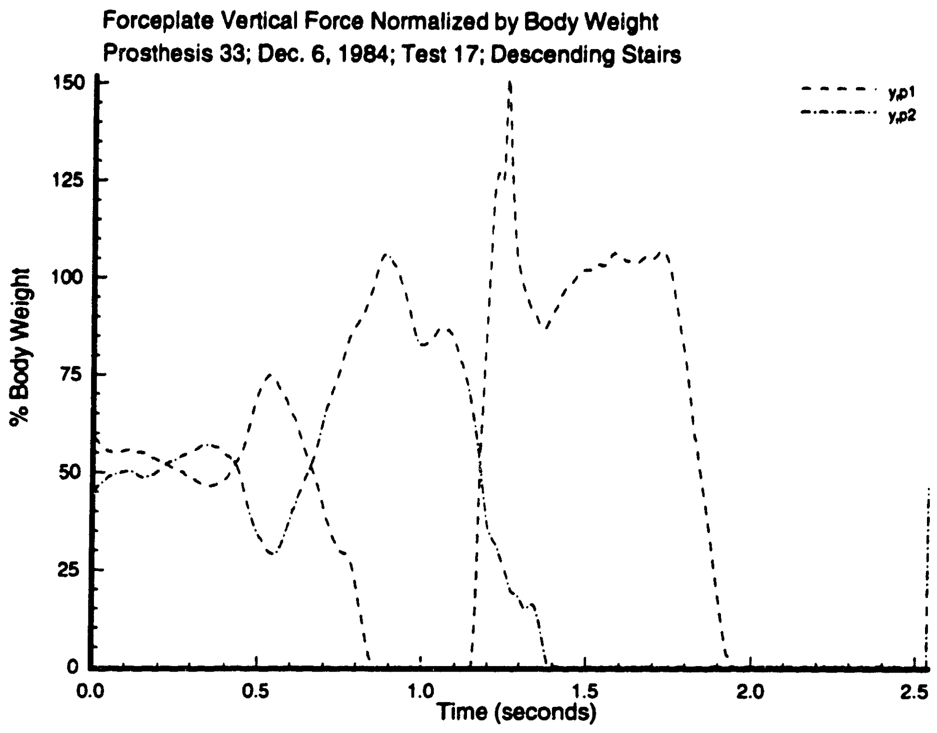


Figure 5-38: Forceplate data for subject 1: Descending Stairs

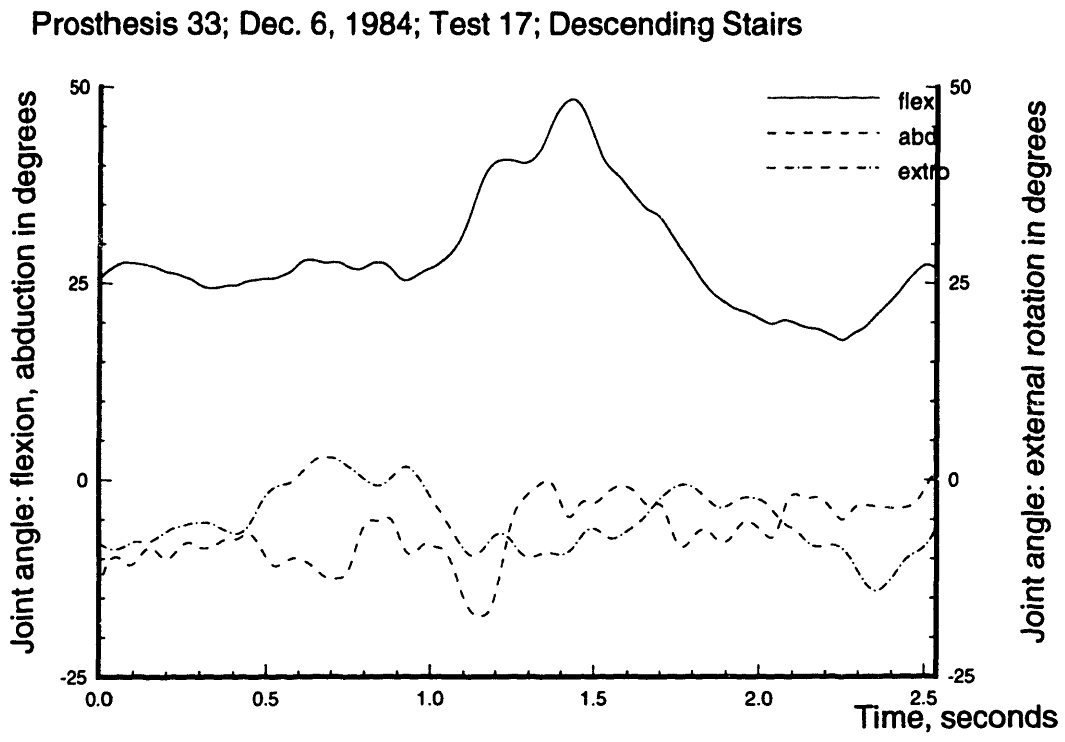


Figure 5-39: Joint Angles for Subject 1: Descending Stairs

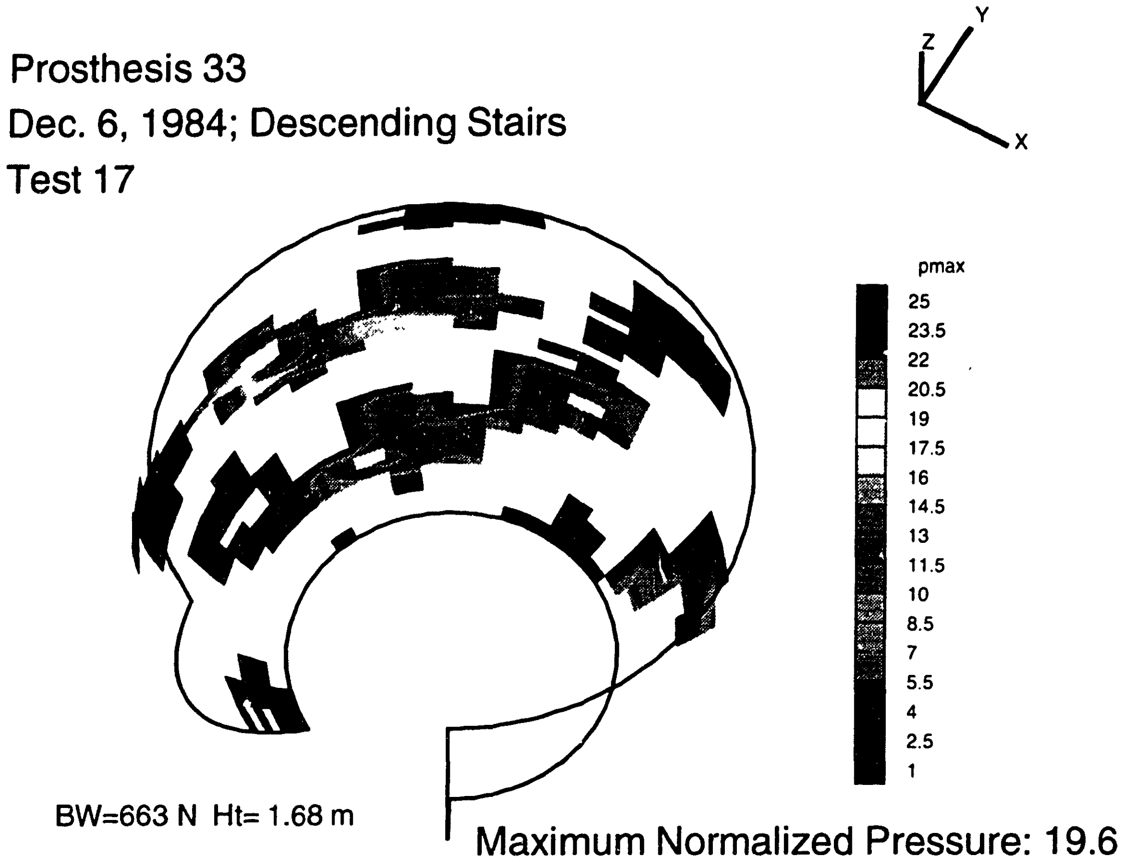
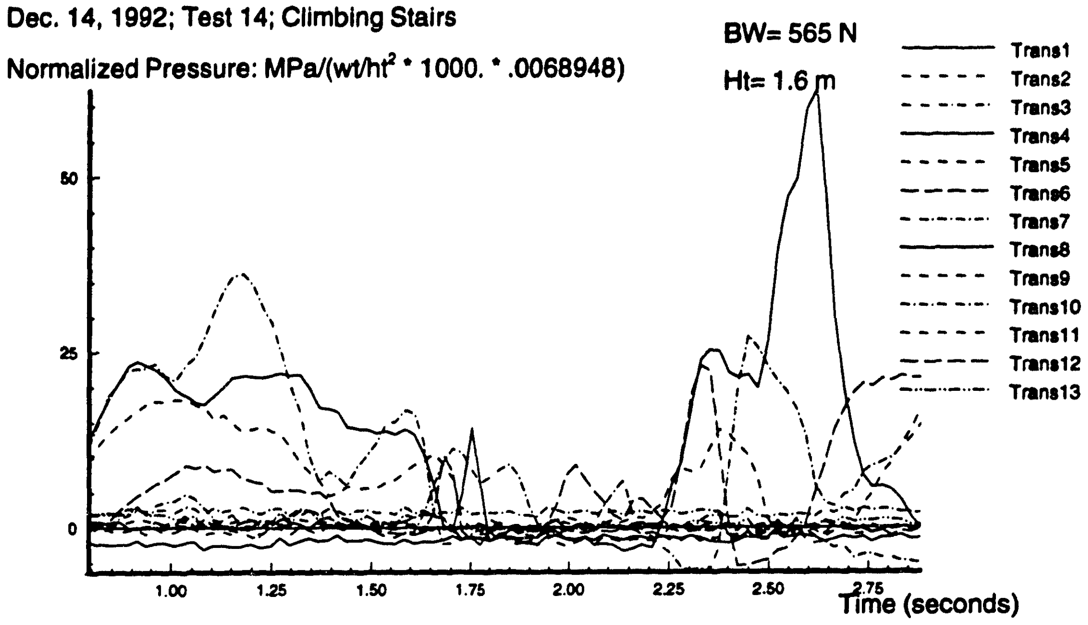


Figure 5-40: Regional Maximums for Subject 1: Descending Stairs

subject has complained of balance difficulty; it appears that this is being manifested in massive co-contraction during this movement and during descending stairs. Visually, the subject ascends and descends stairs with difficulty, particularly during descent.

Very high pressures have been measured in the second subject as he descends a set of stairs. Only data from paced descending is available, pacing was done at 80 BPM, however due to the difficulty with which he performed this movement the actual pace may be different. Figure 5-45 shows the local pressure magnitude in the joint during descending stairs. The pressures in this figure are normalized. Figure 5-46 shows the corresponding forceplate forces during the same tests; the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject. The joint angles for the tests shown are presented in figure 5-47. Figure 5-48 presents the regional statistics for the same tests.

While the kinematics for this subject while ascending stairs are similar to those from the first subject, the kinematics during descent are quite different and complex.



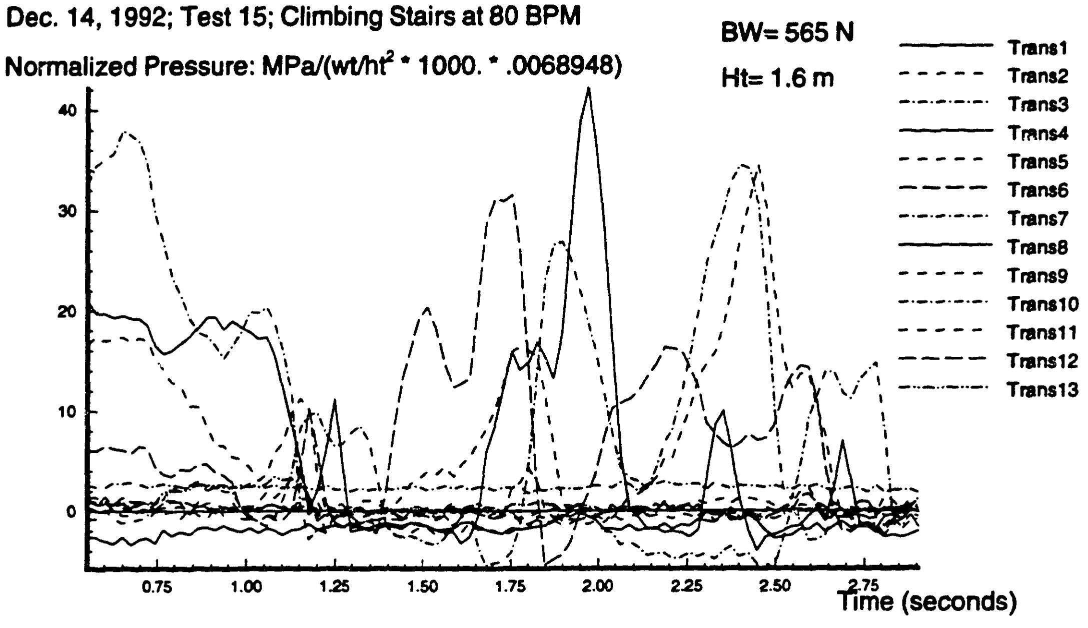
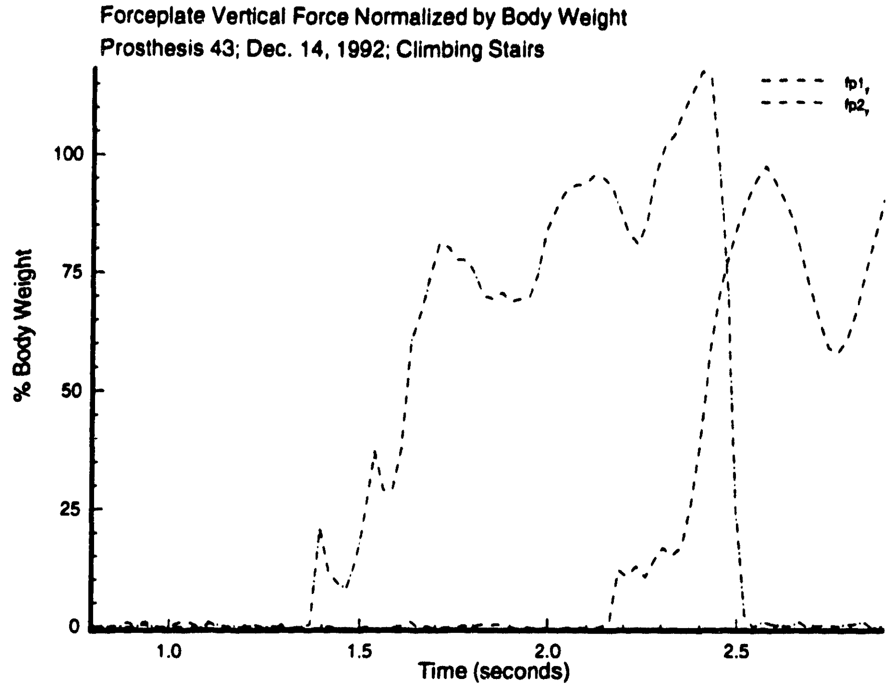


Figure 5-41: Pressures for subject 2: Ascending Stairs



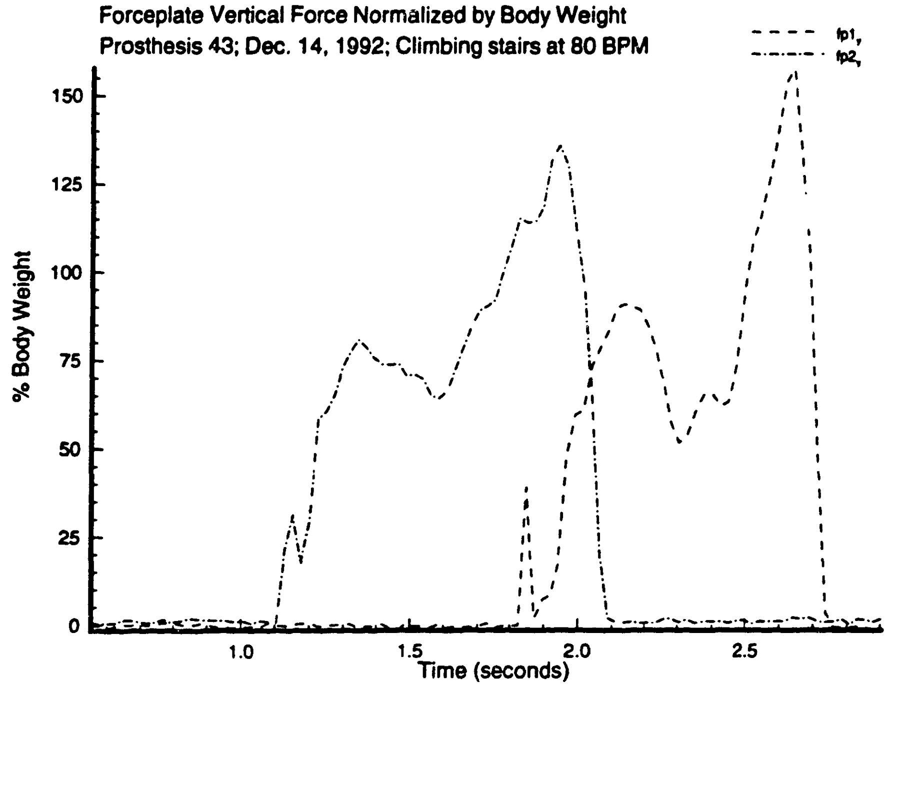
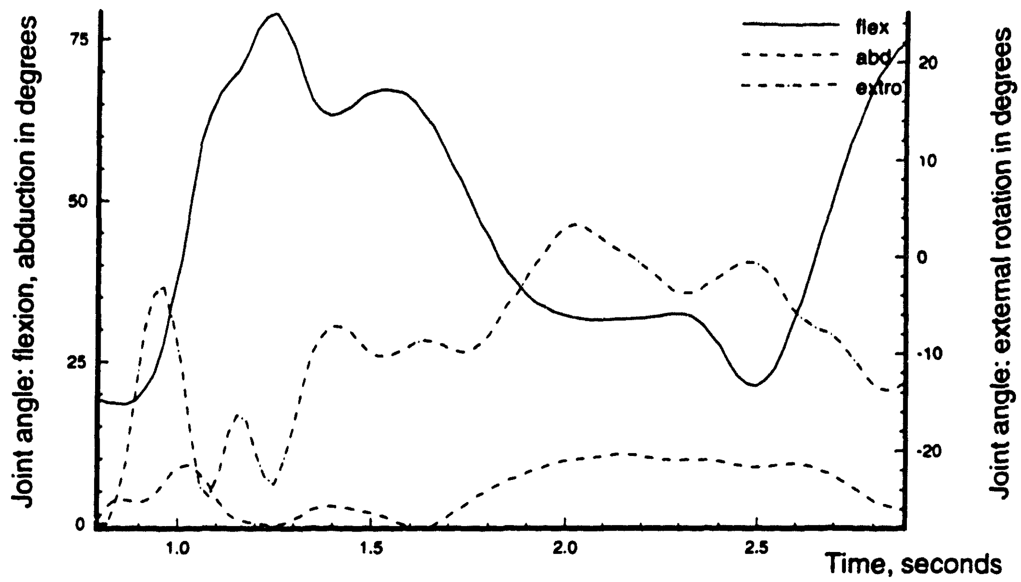


Figure 5-42: Forceplate data for subject 2: Ascending Stairs

Prosthesis 43; Dec. 14, 1992; Test 14; Climbing Stairs



Prosthesis 43; Dec. 14, 1992; Test 15; Climbing Stairs at 80 BPM

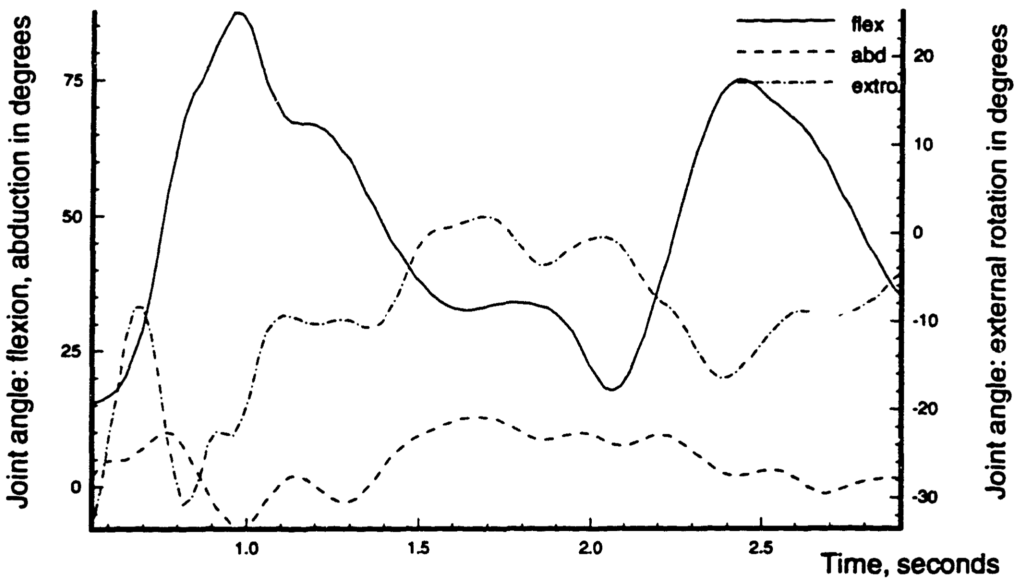
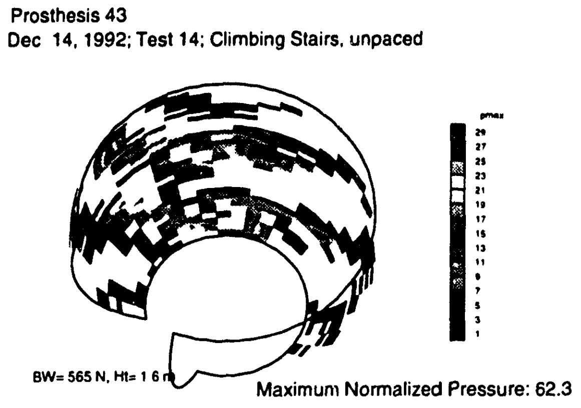


Figure 5-43: Joint Angles for Subject 2: Ascending Stairs



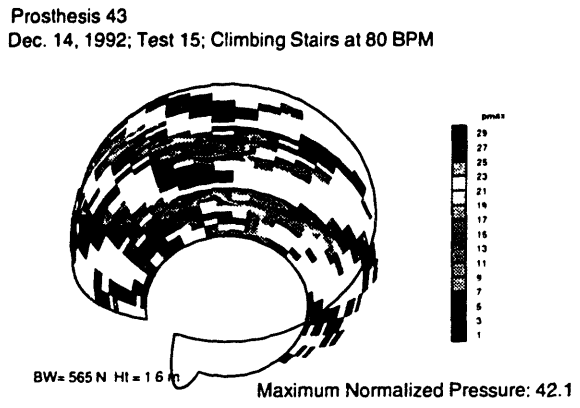
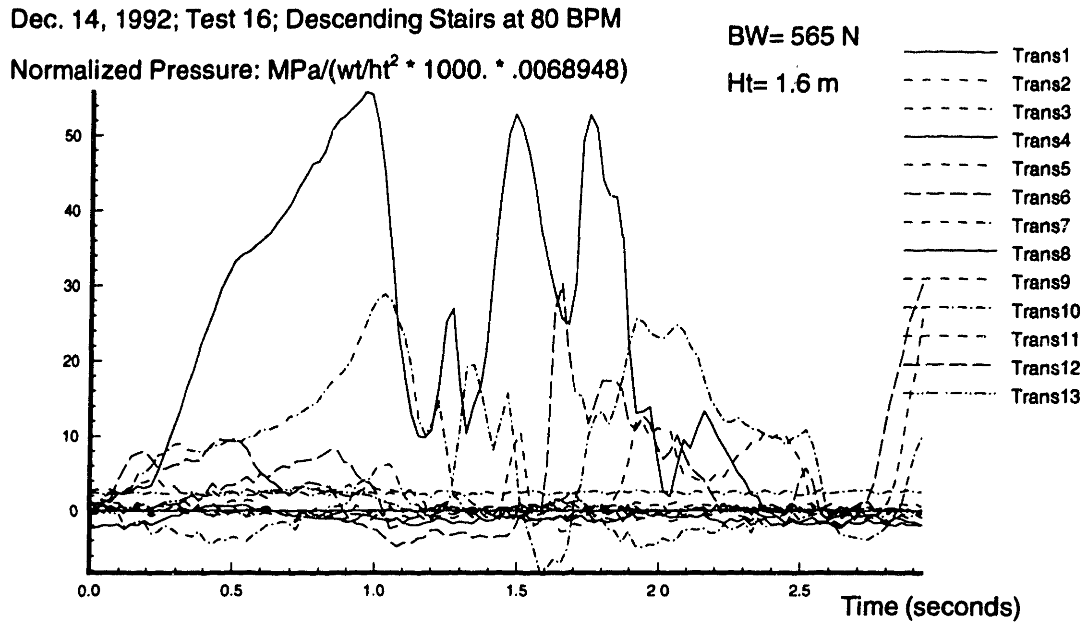


Figure 5-44: Regional Maximums for Subject 2: Ascending Stairs



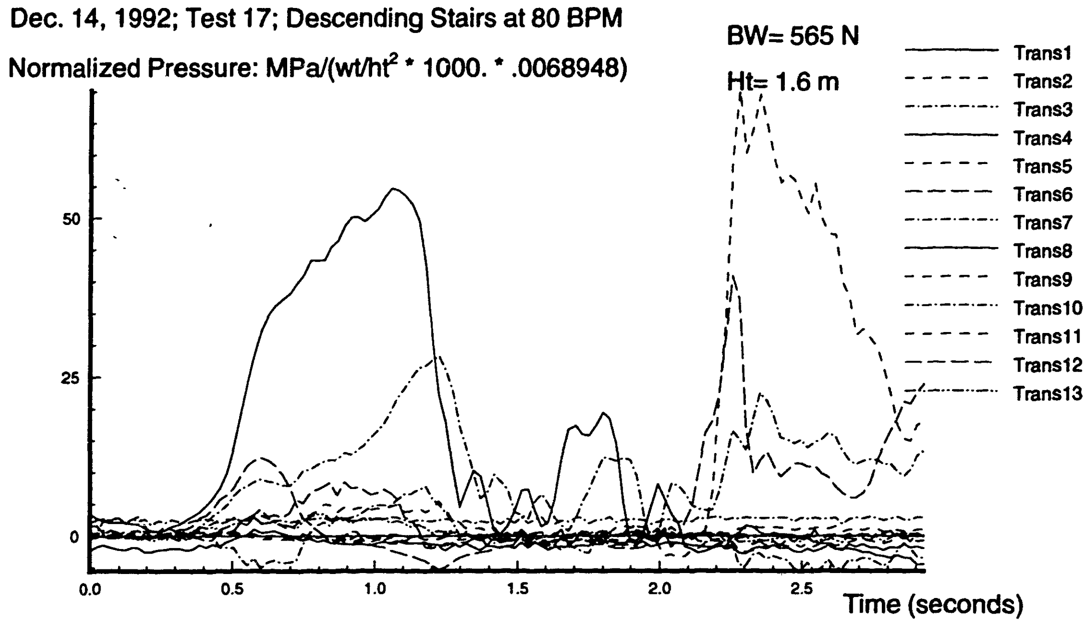


Figure 5-45: Pressures for subject 2: Descending Stairs

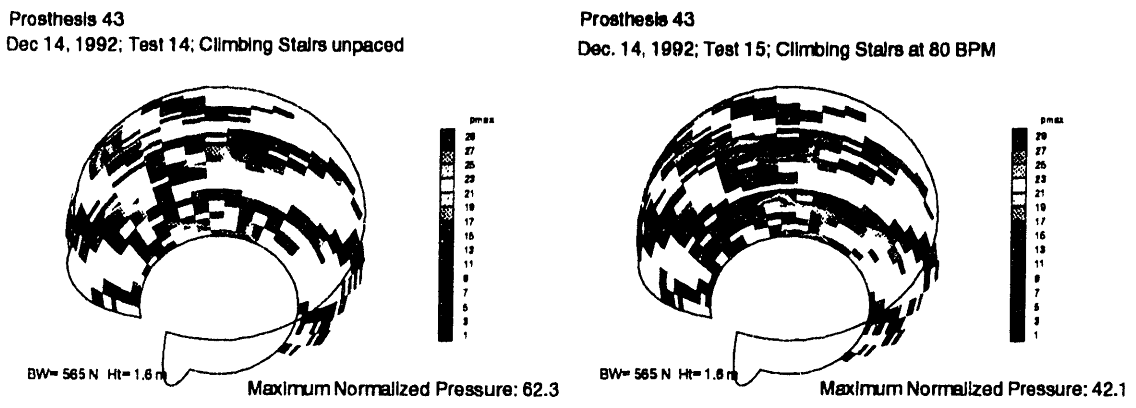


Figure 5-44: Regional Maximums for Subject 2: Ascending Stairs

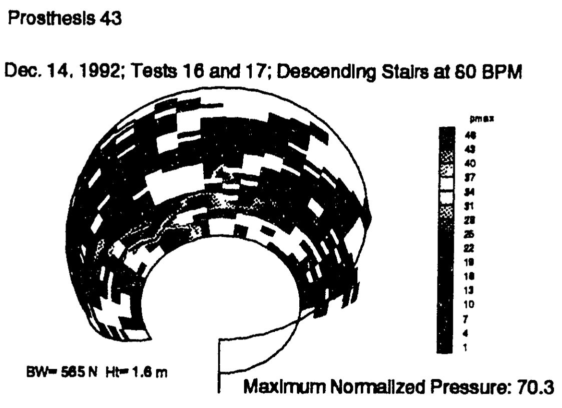
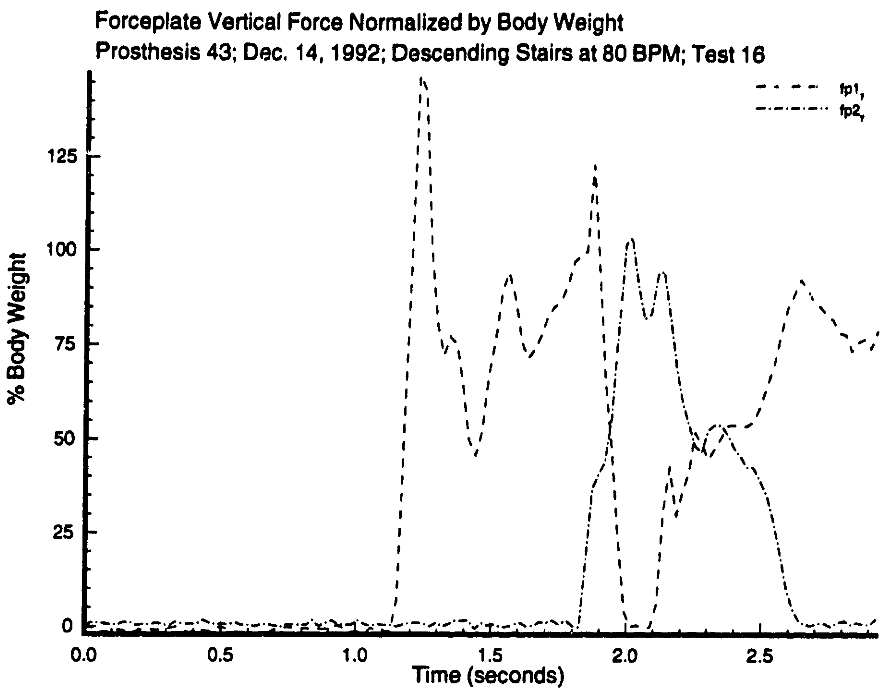


Figure 5-48: Regional Maximums for Subject 2: Descending Stairs



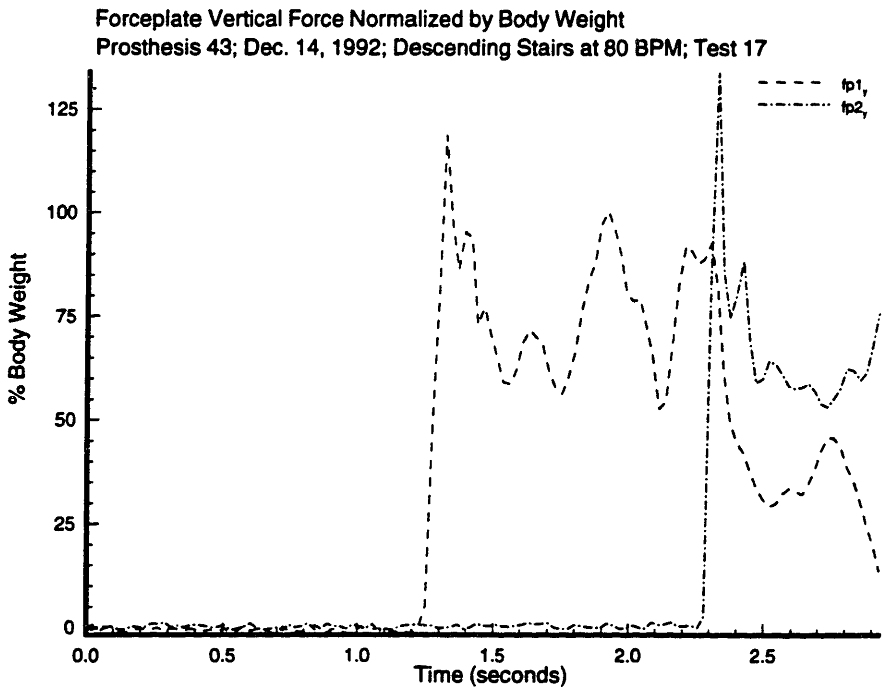
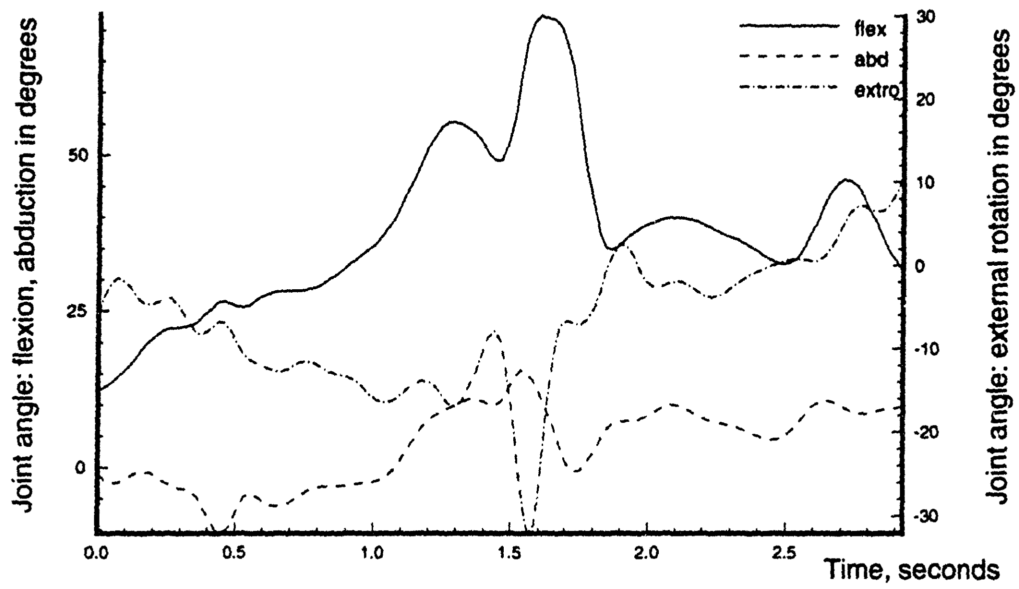


Figure 5-46: Forceplate data for subject 2: Descending Stairs

Prosthesis 43; Dec. 14, 1992; Test 16; Descending Stairs at 80 BPM



Prosthesis 43; Dec. 14, 1992; Test 17; Descending Stairs at 80 BPM

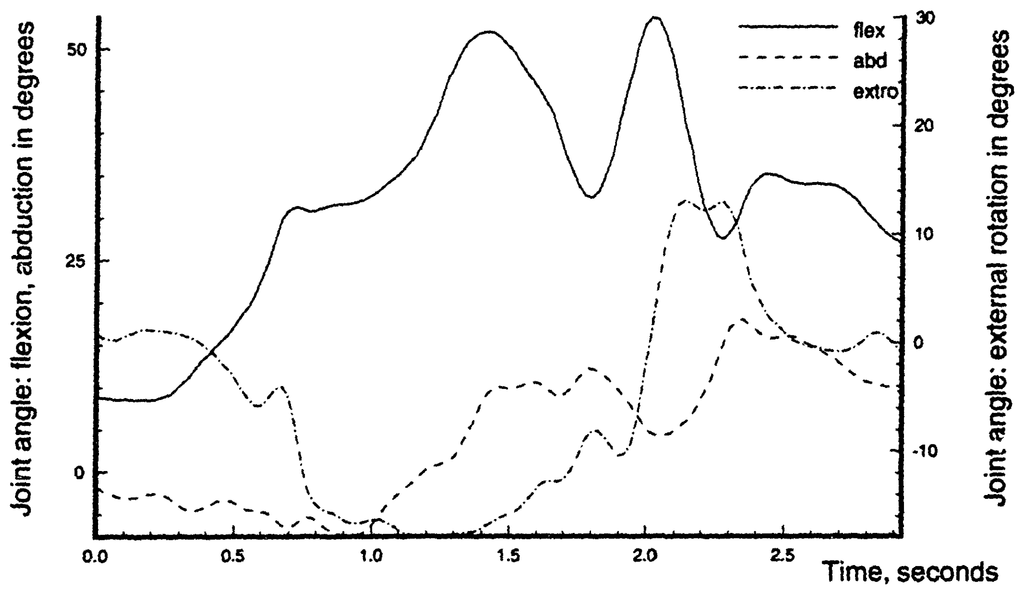
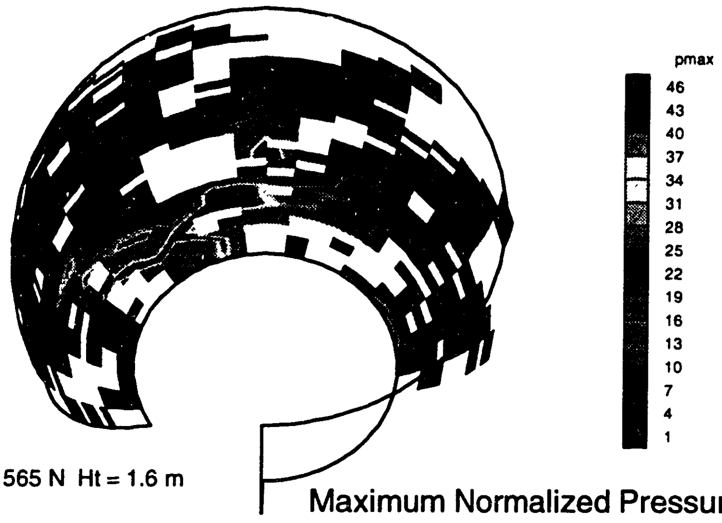


Figure 5-47: Joint Angles for Subject 2: Descending Stairs

Prosthesis 43

Dec. 14, 1992; Tests 16 and 17; Descending Stairs at 80 BPM



Maximum Normalized Pressure: 70.3

Figure 5-48: Regional Maximums for Subject 2: Descending Stairs

5.4 Isometric Abduction

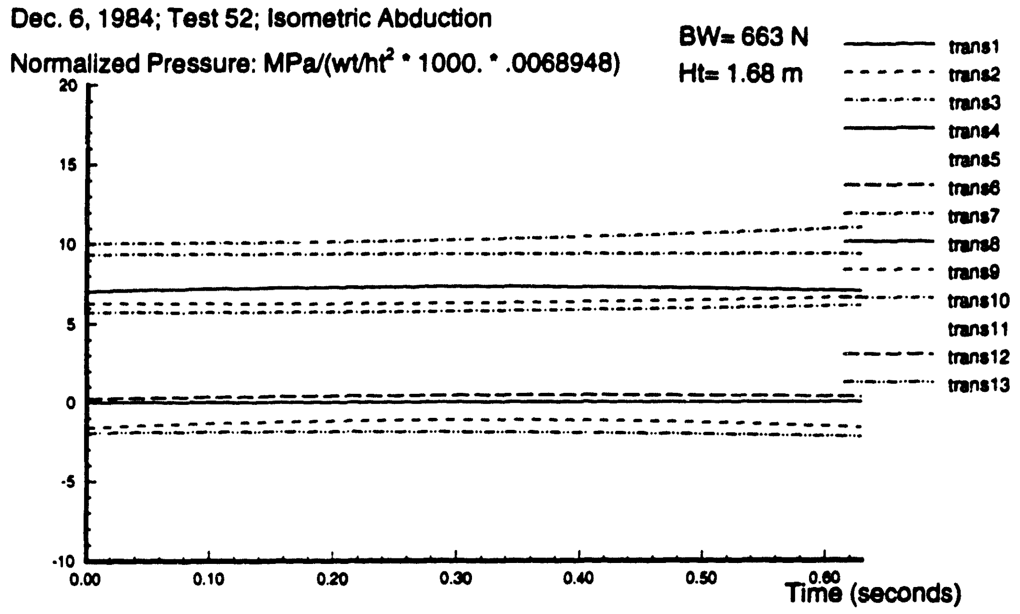
Isometric abduction has been performed each time data is acquired from the second subject and was performed several times with the first subject. The protocol with the second subject has been that he lies on a physical therapy table and pushes outward against restraint; he is instructed to push as hard as he can. Data acquisition starts just as he starts to push, so that a ramp increase in pressure is seen. No kinematic data has been acquired during this activity. Abduction tests with the first subject were performed in a different manner, and were not consistent. A description of the conditions accompanies each test for prosthesis 33.

5.4.1 Prosthesis 33

Isometric abduction data at 6 months and 2 1/2 years post-operative were taken with the subject lying on a couch and pushing out against a restraining force. This subject was not instructed to push as hard as she could, and data was taken starting after she had been performing the activity for a short time. Kinematic data were acquired at the same time. Figure 5-49 shows the local pressure magnitude in the joint during isometric abduction. The pressures in this figure are normalized and smoothed. The joint angles for the tests shown are presented in figure 5-50. Figure 5-51 presents the regional statistics for the same tests.

5.4.2 Prosthesis 43

Testing of maximum voluntary isometric (MVIC) abduction has been done at almost every test session for the second subject. Figure 5-52 shows the pressures vs. time for some of these tests. The pressures in this figure are normalized. Figures 5-53 and 5-54 present the regional statistics for the same tests, however since no kinematic data is available statistics are only displayed for each transducer in the prosthesis.



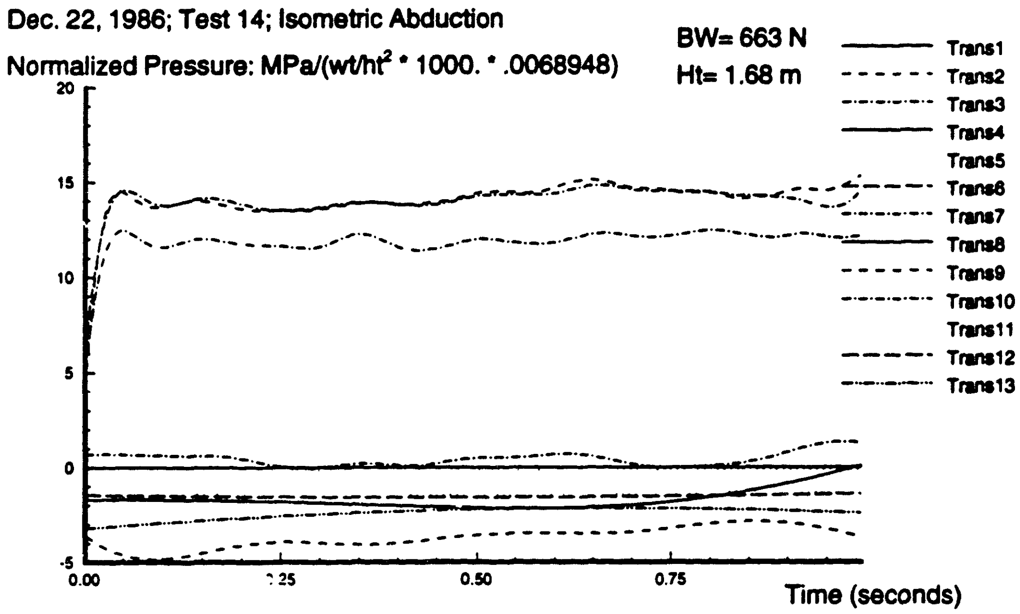
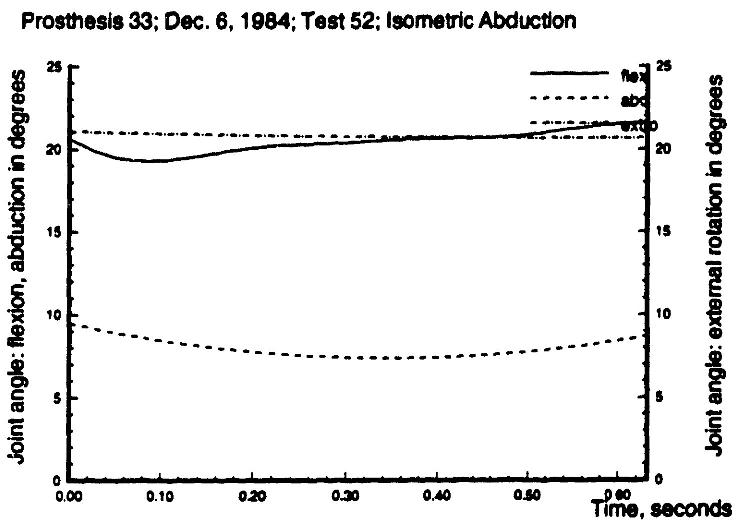


Figure 5-49: Pressures for subject 1: Isometric Abduction



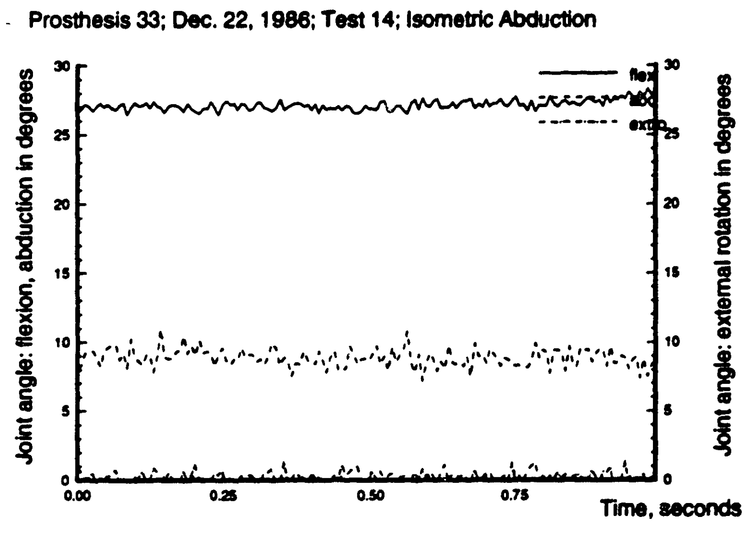


Figure 5-50: Joint Angles for Subject 1: Isometric Abduction

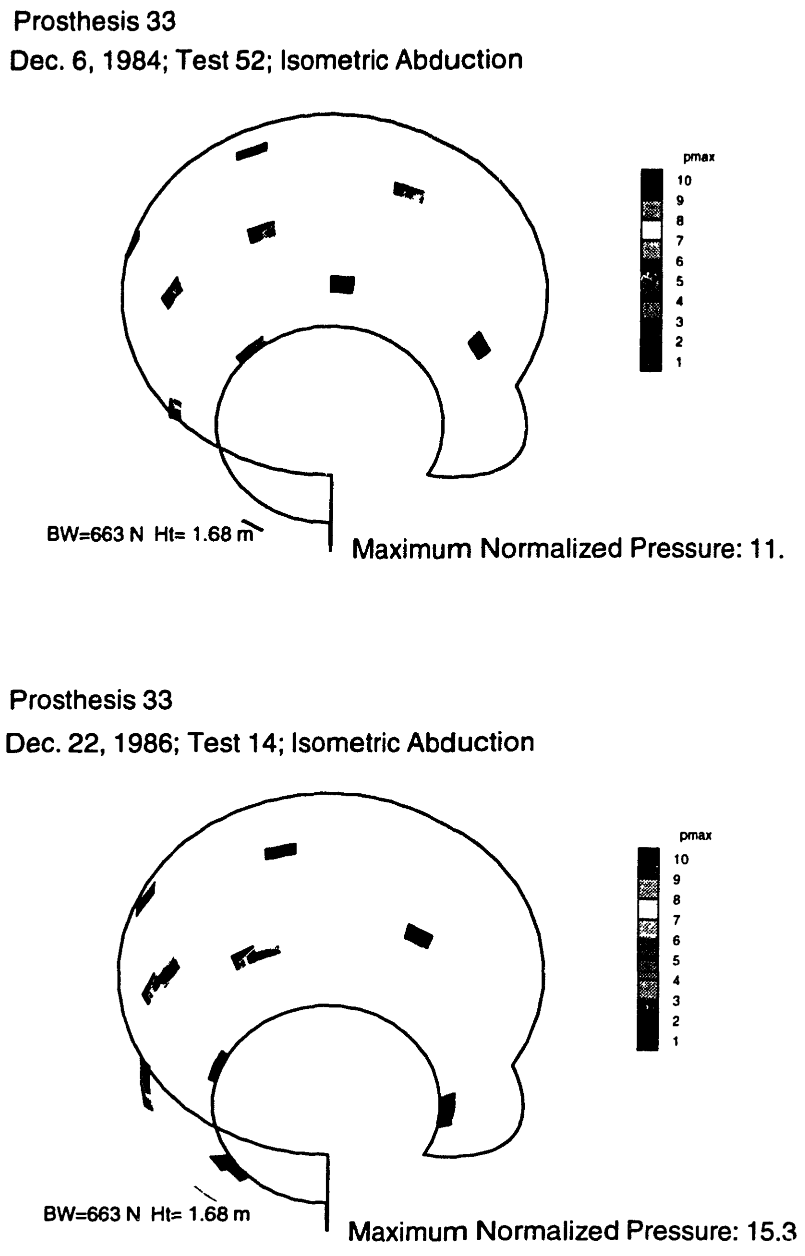


Figure 5-51: Regional Maximums for Subject 1: Isometric Abduction

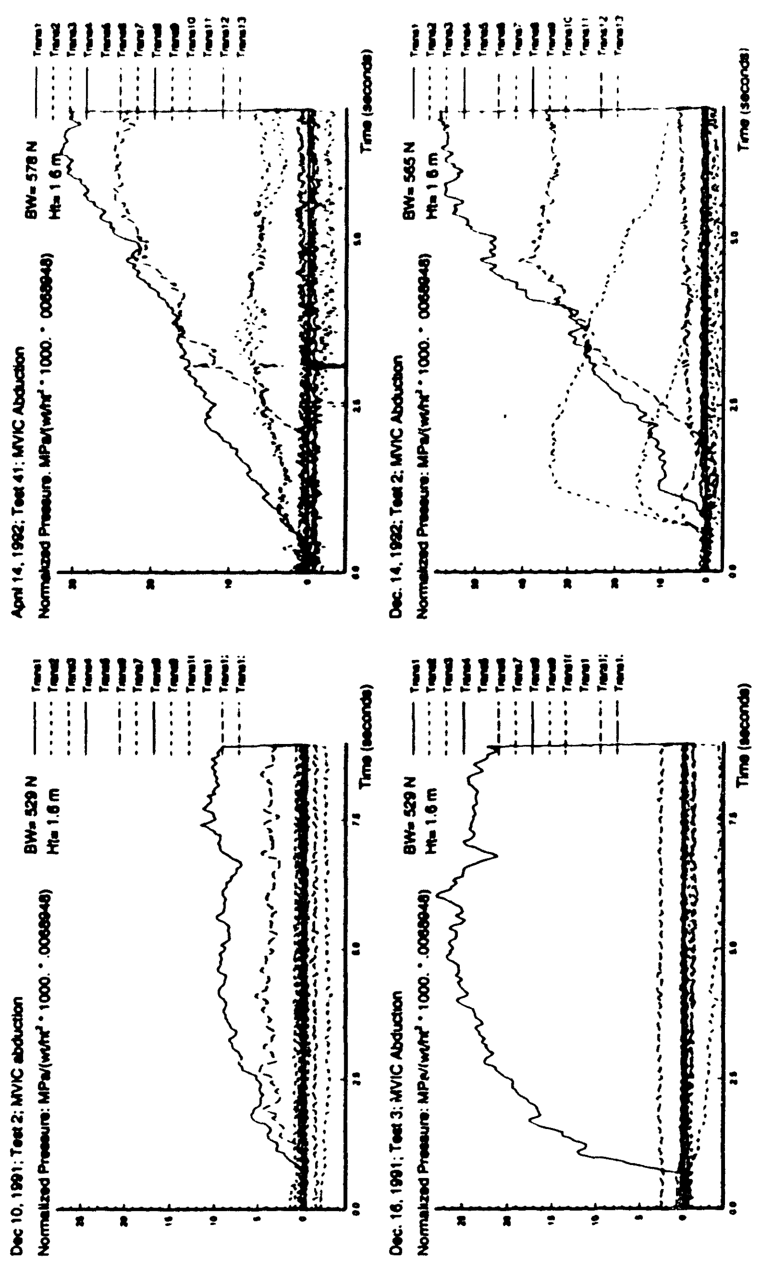
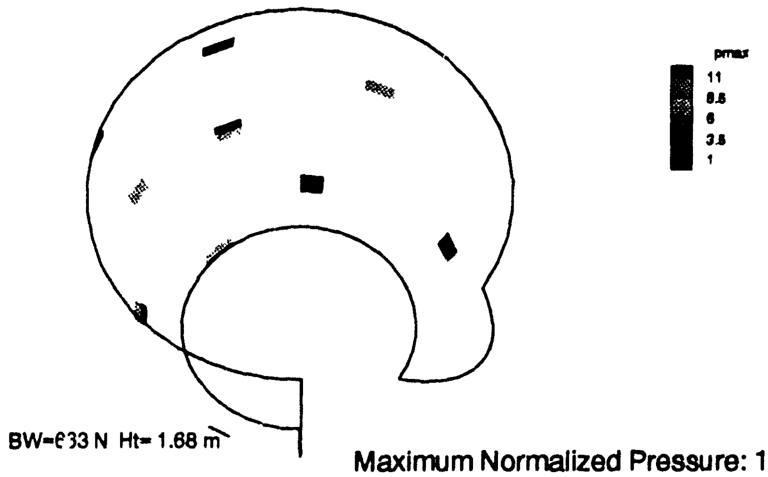


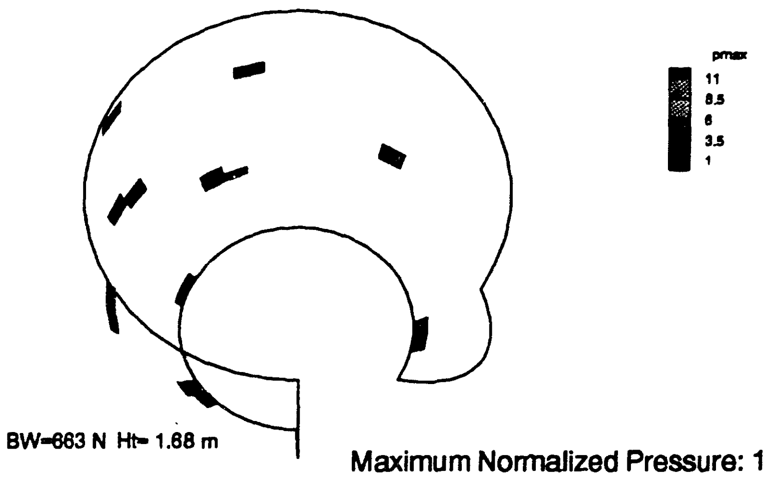
Figure 5-52: Pressures for subject 2: MVIC Abduction

Prosthesis 33
Dec. 6, 1984; Test 52; Isometric Abduction



Maximum Normalized Pressure: 11

Prosthesis 33
Dec. 22, 1986; Test 14; Isometric Abduction

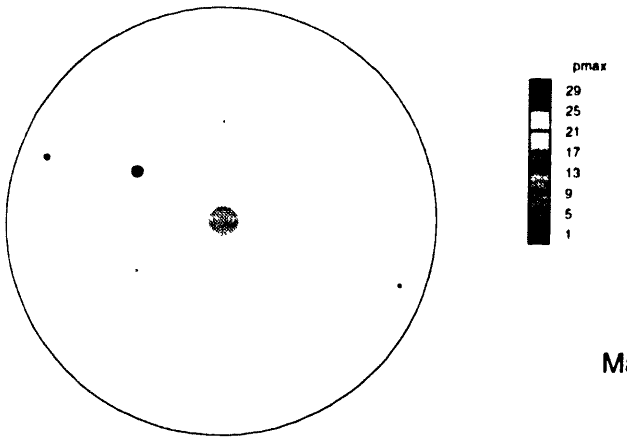


Maximum Normalized Pressure: 15.3

FIGURE 5-51

MVIC Abduction

Normalized Maximum pressures at each transducer



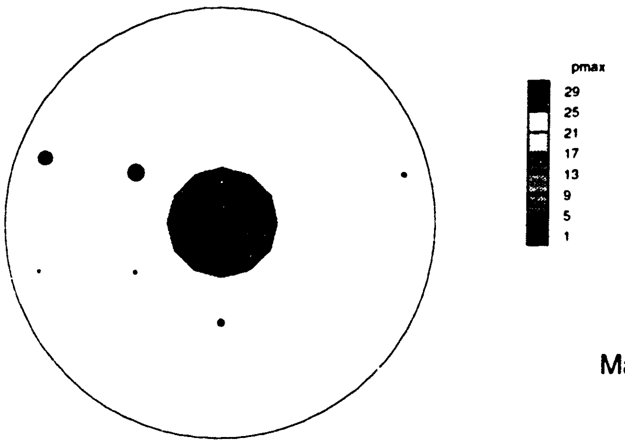
Prosthesis 43:
Dec. 10, 1991
Tests 2 and 3

Data displayed in
Right hip orientation

Maximum normalized pressure: 11.6
BW= 529 N
Ht = 1.6 m

MVIC Abduction

Normalized Maximum pressures at each transducer



Prosthesis 43:
Dec. 16, 1991
Test 3

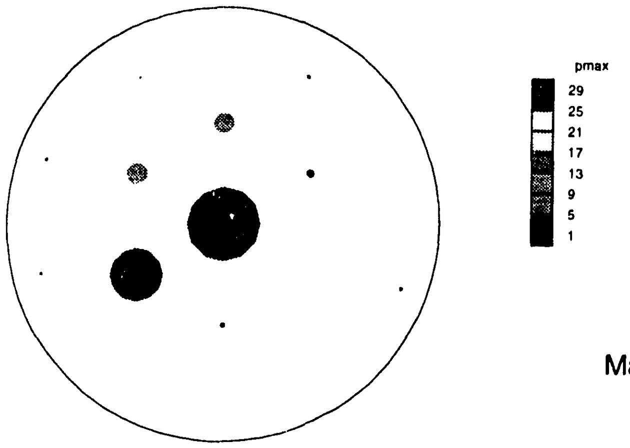
Data displayed in
Right hip orientation

Maximum normalized pressure: 27.8
BW= 529 N
Ht = 1.6 m

Figure 5-53: Regional Maximums for Subject 2: MVIC Abduction, post-operative

MVIC Abduction

Normalized Maximum pressures at each transducer



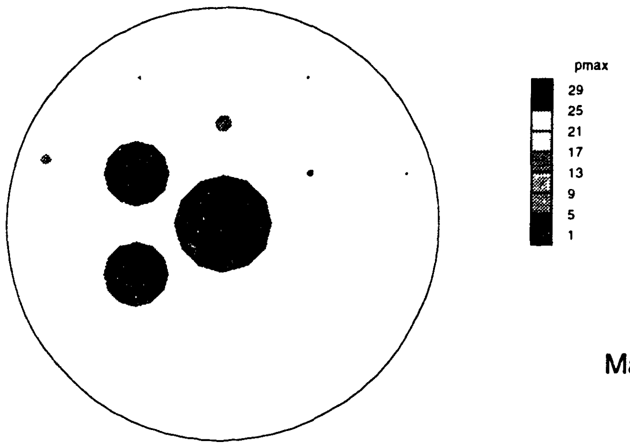
Prosthesis 43:
April 14, 1992
Tests 2 and 41

Data displayed in
Right hip orientation

Maximum normalized pressure: 31.8
BW= 578 N
Ht = 1.6 m

MVIC Abduction

Normalized Maximum pressures at each transducer



Prosthesis 43:
Dec. 14, 1992
Test 2

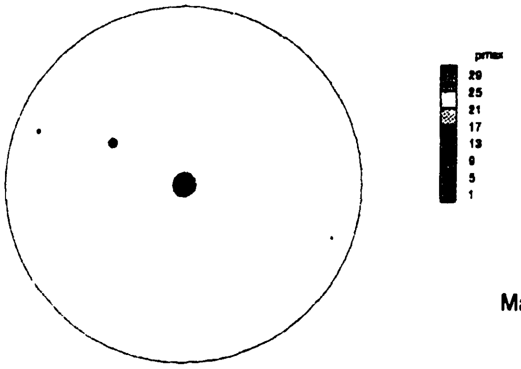
Data displayed in
Right hip orientation

Maximum normalized pressure: 58.3
BW= 565 N
Ht = 1.6 m

Figure 5-54: Regional Maximums for Subject 2: MVIC Abduction

MVIC Abduction

Normalized Maximum pressures at each transducer



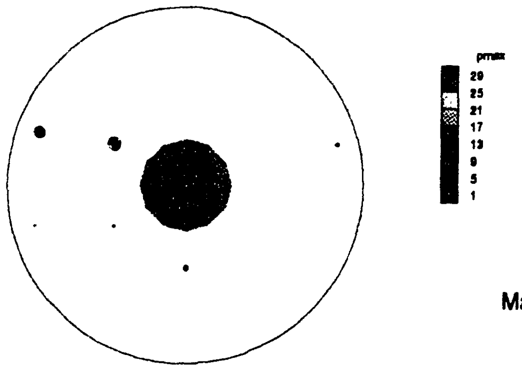
Prosthesis 43:
Dec. 10, 1991
Tests 2 and 3

Data displayed in
Right hip orientation

Maximum normalized pressure: 11.6
BW= 529 N
Ht = 1.6 m

MVIC Abduction

Normalized Maximum pressures at each transducer



Prosthesis 43:
Dec. 16, 1991
Test 3

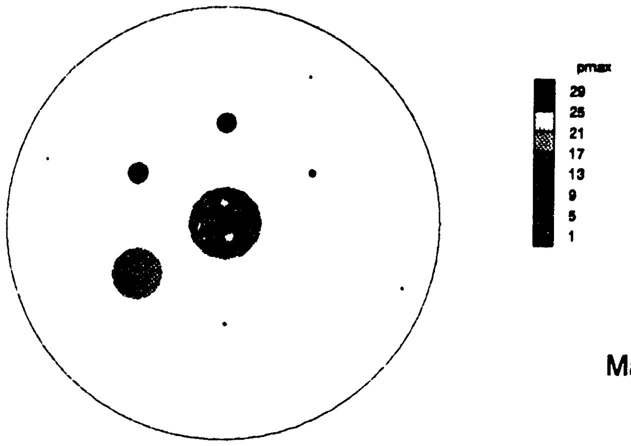
Data displayed in
Right hip orientation

Maximum normalized pressure: 27.8
BW= 529 N
Ht = 1.6 m

FIGURE 5-53

MVIC Abduction

Normalized Maximum pressures at each transducer



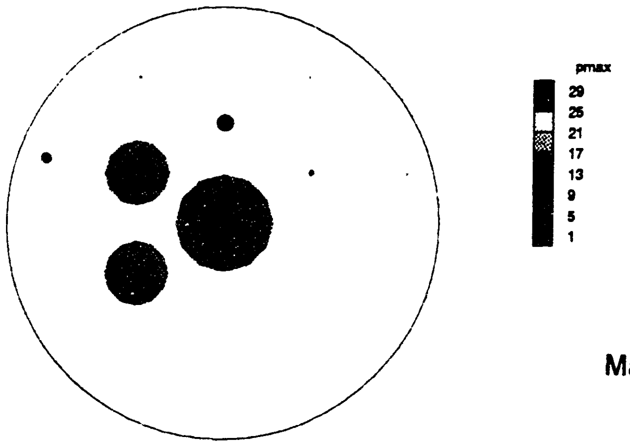
Prosthesis 43:
April 14, 1992
Tests 2 and 41

Data displayed in
Right hip orientation

Maximum normalized pressure: 31.8
BW= 578 N
Ht = 1.6 m

MVIC Abduction

Normalized Maximum pressures at each transducer



Prosthesis 43:
Dec. 14, 1992
Test 2

Data displayed in
Right hip orientation

Maximum normalized pressure: 58.3
BW= 565 N
Ht = 1.6 m

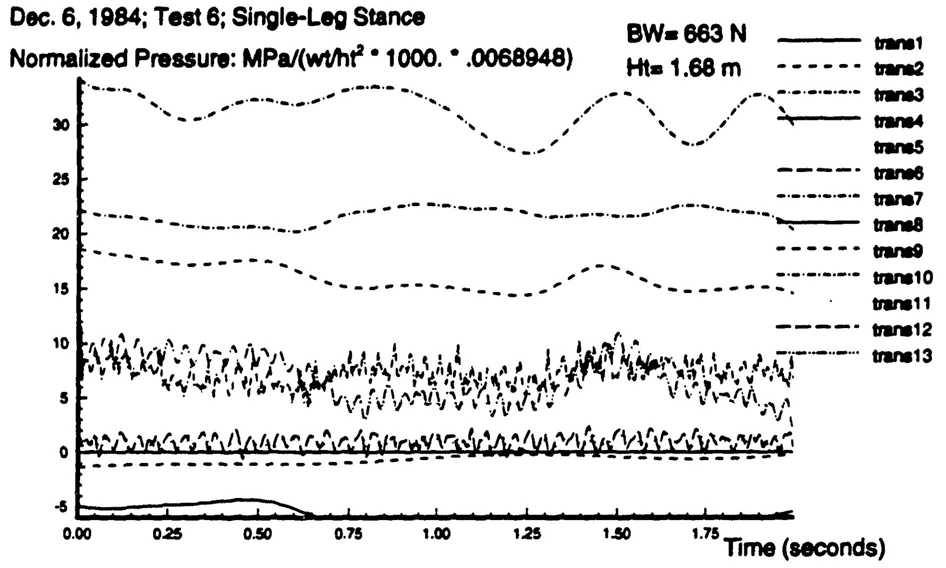
FIGURE 5-54

5.5 Single-Leg Stance

Another activity during which little movement takes place is single-leg stance, always on the leg in which the instrumented prosthesis is implanted. Instructions to both subjects were essentially the same for this test and neither subject was assisted with support. The second subject had greater difficulty performing this test, due to his balance difficulty, thus he was not always able to stand on a single leg for the duration of the test. The forceplate data and the joint angle data indicate what movement was taking place during each test.

5.5.1 Prosthesis 33

Figure 5-55 shows the local pressure magnitude in the joint during single-leg stance on the instrumented leg. The pressures in this figure are normalized and smoothed. Figure 5-56 shows the corresponding forceplate forces during the test from December 1984 (the subject was not standing on either forceplate in Dec. 1986); the vertical force at the forceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this test. The joint angles for the tests shown are presented in figure 5-57. Figure 5-58 presents the regional statistics for the same tests.



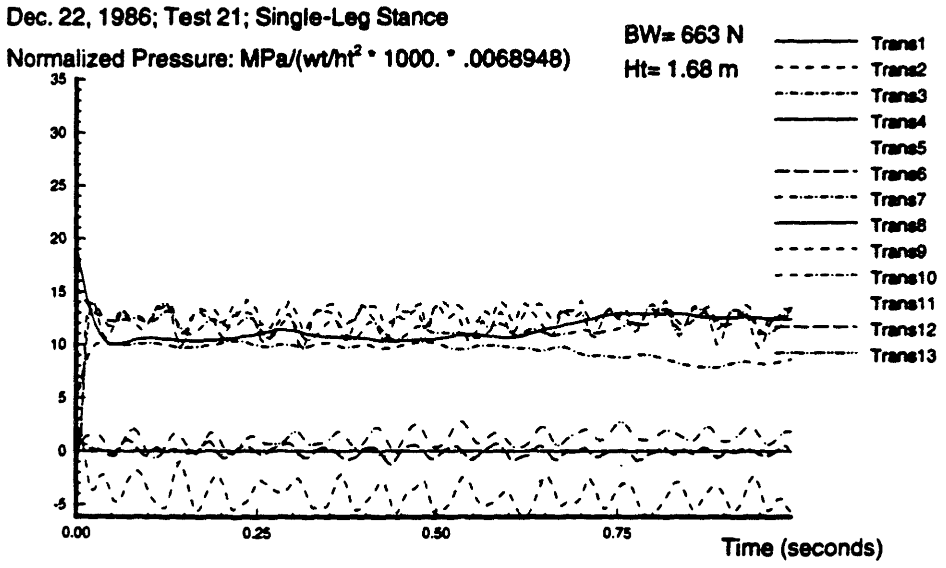


Figure 5-55: Pressures for subject 1: Single-Leg Stance

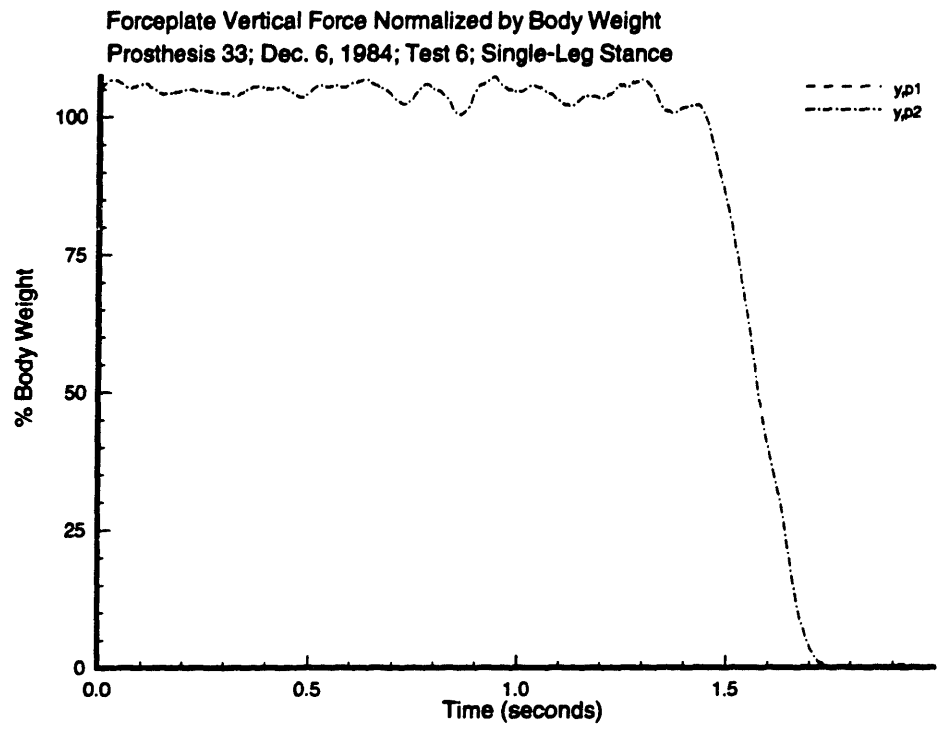


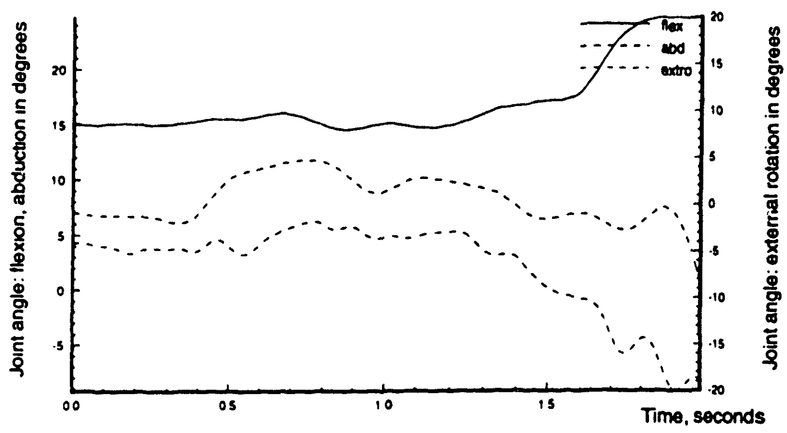
Figure 5-56: Forceplate data for subject 1: Single-Leg Stance

5.5.2 Prosthesis 43

Single-leg stance results are shown in figure 5-59 for standing on the instrumented leg. The pressures in this figure are normalized. Figure 5-60 shows the corresponding forceplate forces during the same test; the vertical force at the foıceplate is shown and it is normalized by body weight. The force on the instrumented leg is measured at forceplate 2 for this subject, and shows that the subject was not always standing on only the instrumented leg. The joint angles for the tests shown are presented in figure 5-61. Figure 5-62 presents the regional statistics for the same tests.

The pressure data from this test for this subject, indicate how large a role co-contraction forces can play in determining the loading at a joint. This subject is clearly having difficulty stabilizing himself on one foot, and must as a result be contracting the muscles about his hip joint since the pressures measured in the joint are more than double those measured during gait, even though the kinematics indicate little dynamic contribution to pressure.

Prosthesis 33; Dec. 6, 1984; Test 6; Single-Leg Stance



Prosthesis 33; Dec. 22, 1986; Test 21; Single-Leg Stance

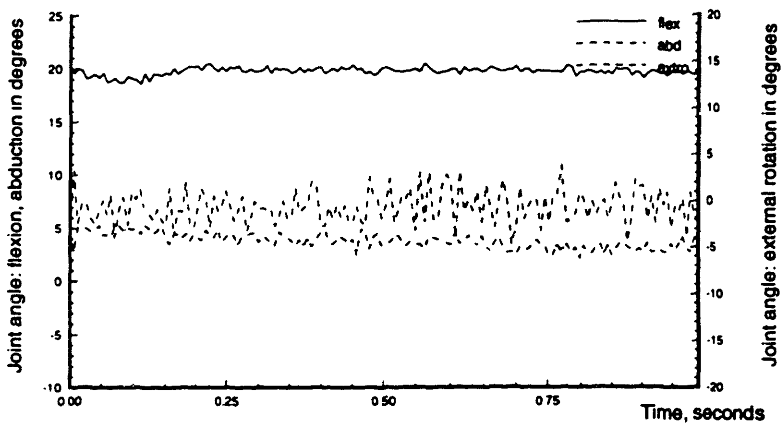
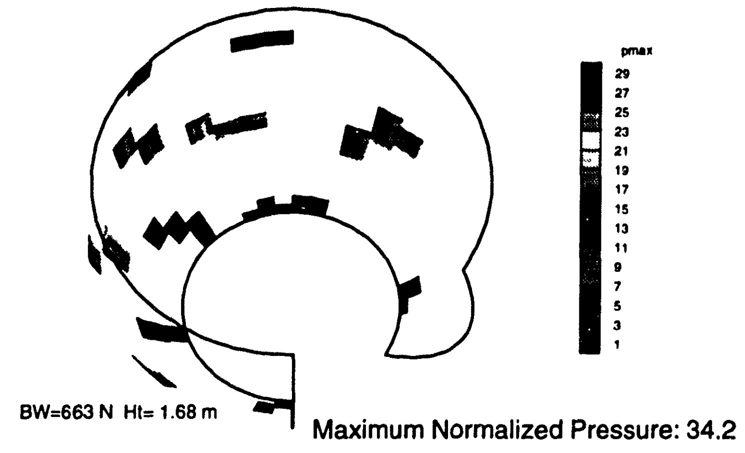


Figure 5-57: Joint Angles for Subject 1: Single-Leg Stance

Prosthesis 33
Dec. 6, 1984; Test 6; Single-Leg Stance (instrumented leg)



Prosthesis 33
Dec. 22, 1986; Test 21; Single-Leg Stance (instrumented leg)

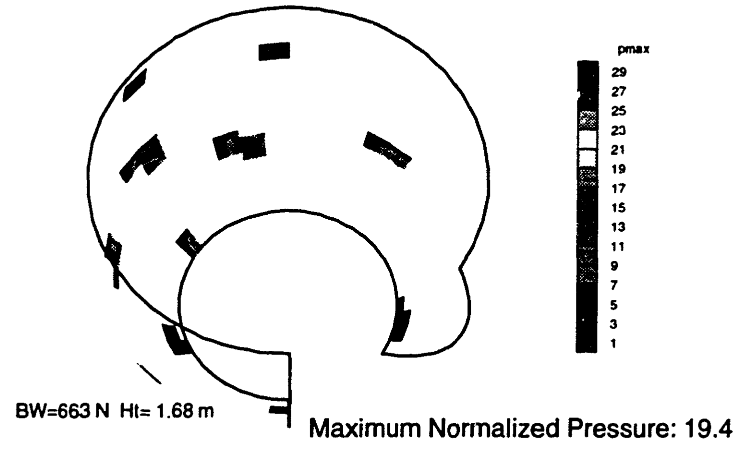


Figure 5-58: Regional Maximums for Subject 1: Single-Leg Stance

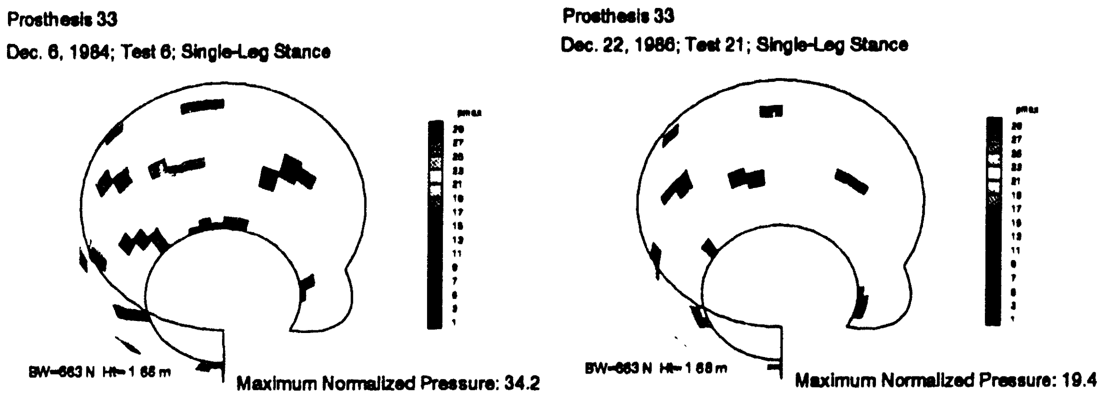


Figure 5-58: Regional Maximums for Subject 1; Single-Leg Stance

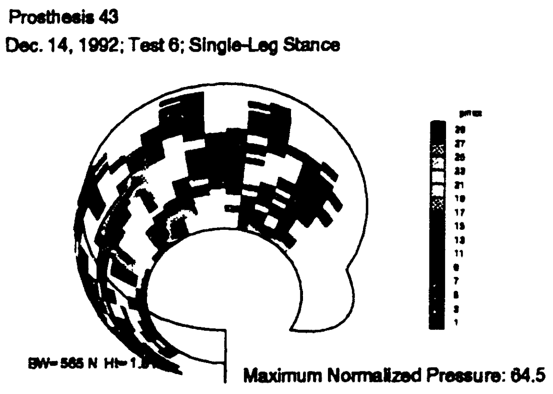


Figure 5-62: Regional Maximums for Subject 2: Single-Leg Stance

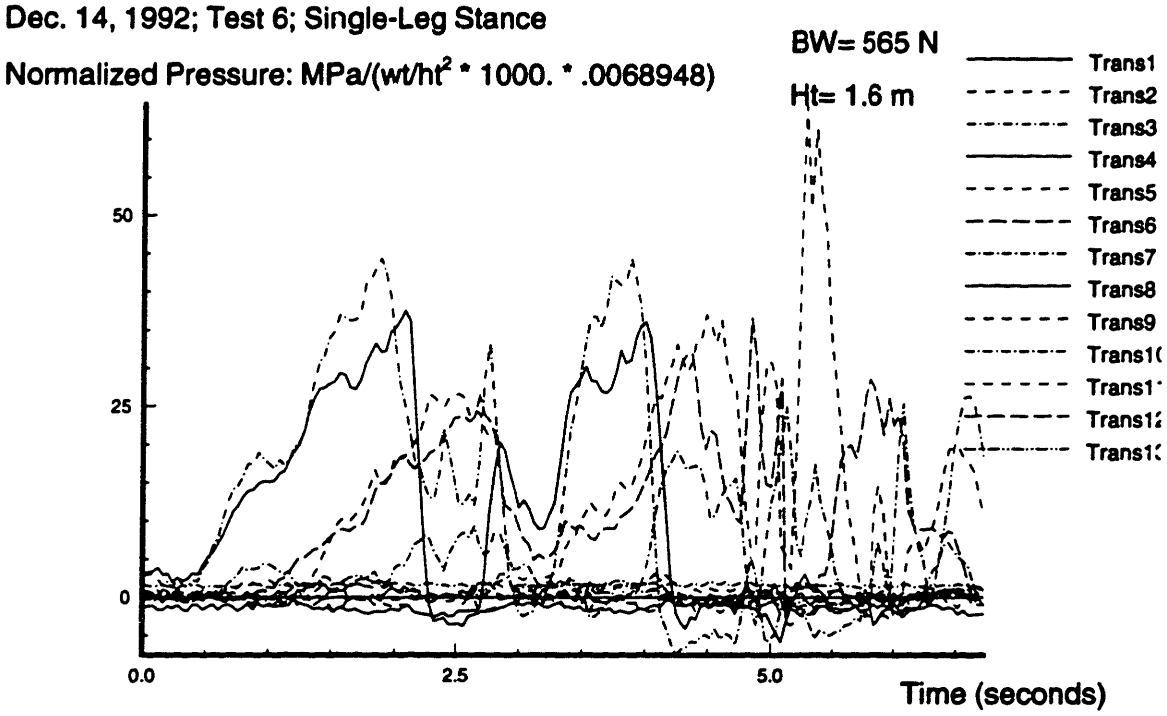


Figure 5-59: Pressures for subject 2: Single-Leg Stance

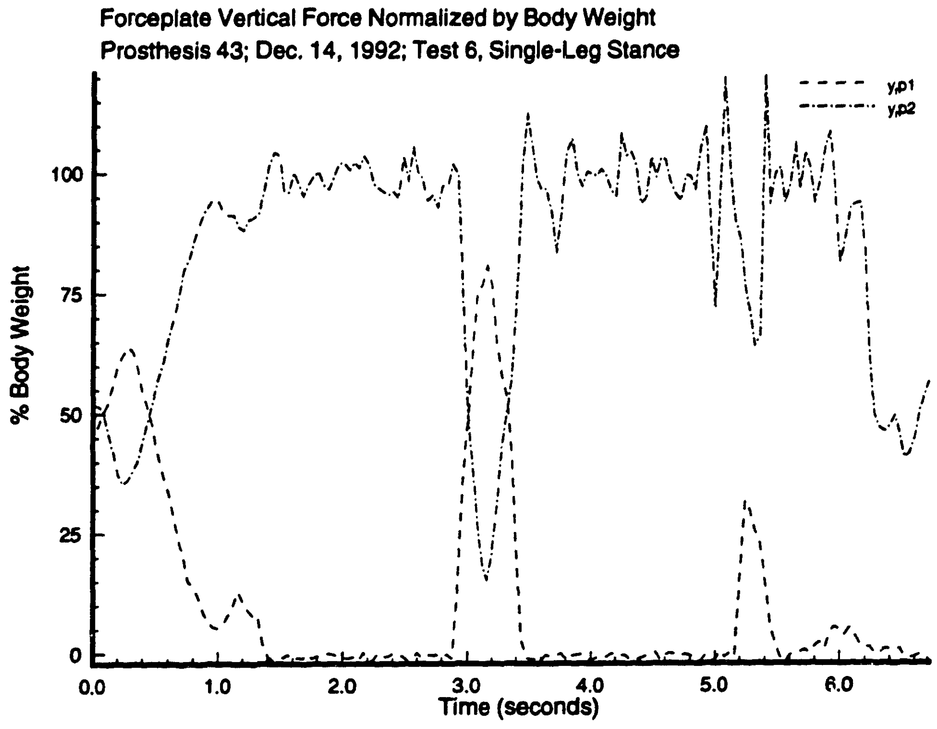


Figure 5-60: Forceplate data for subject 2: Single-Leg Stance

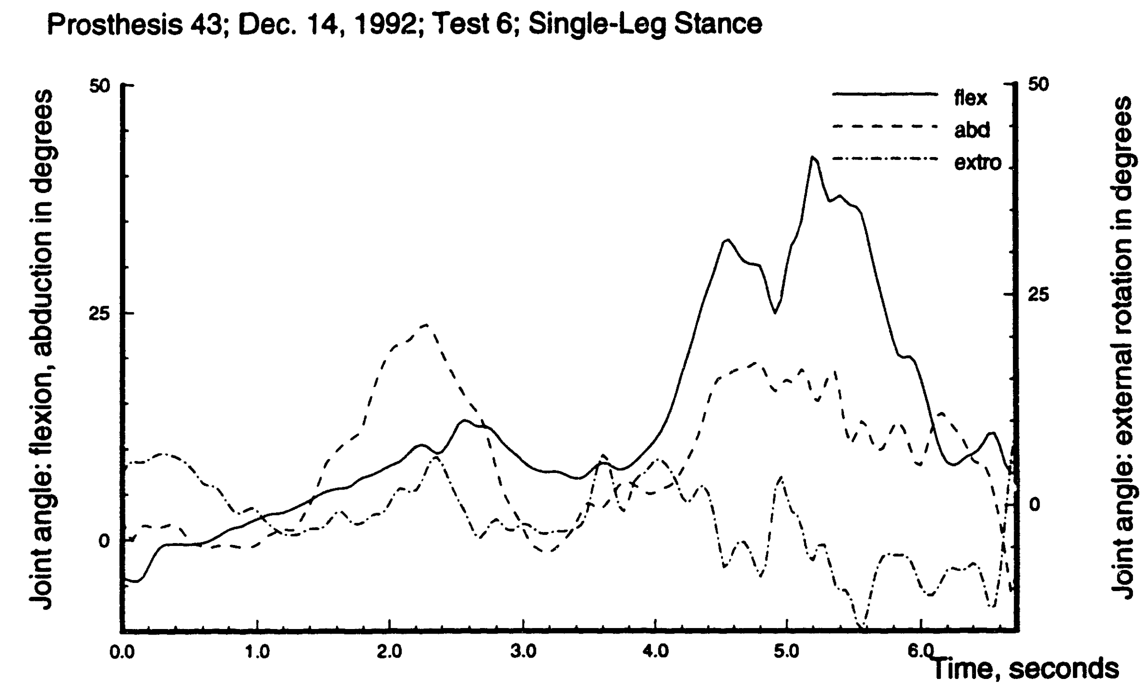


Figure 5-61: Joint Angles for Subject 2: Single-Leg Stance

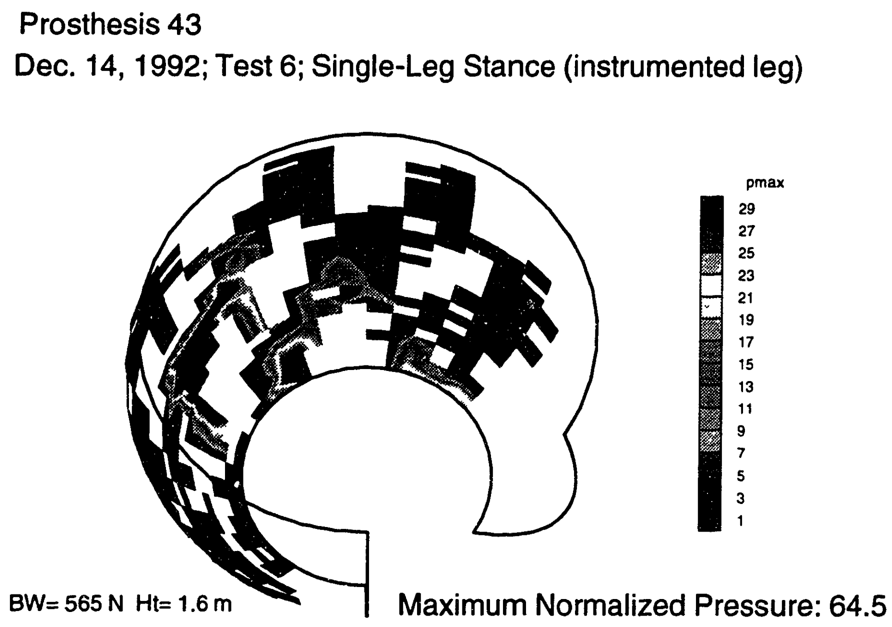


Figure 5-62: Regional Maximums for Subject 2: Single-Leg Stance

Chapter 6

Discussion

This chapter will present a discussion on several aspects of the results, the acetabular location of pressures with implications on bone and cartilage response and endoprosthesis misfit, and the magnitudes of the pressures measured. Many other aspects of the data warrant the extraction of important information, but those chosen here are the most obvious areas to first investigate.

6.1 Endoprosthesis mismatch to acetabulum

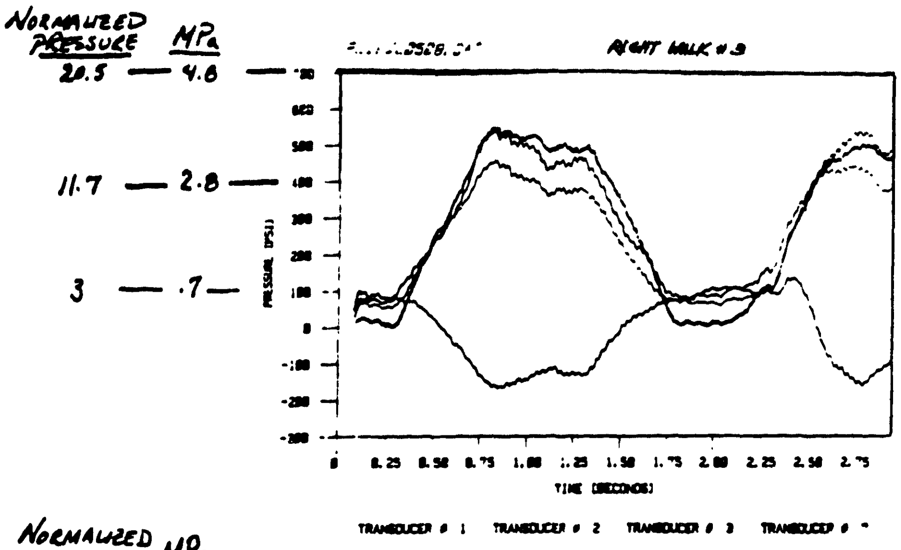
The mismatch in sizing of the prostheses was expected to produce an effect in the distribution of pressures on the acetabulum, though it was not known if the effect of such small mismatches would be discernible. The prosthesis implanted in the first subject was 0.6 mm over-sized in diameter relative to the natural femoral head, while the prosthesis implanted in the second subject was 0.7 mm undersized in diameter relative to that subject's natural femoral head. Differences in the distribution of pressures were apparent shortly after implantation even in the absence of kinematic data during the first two weeks post-operative. Only one transducer reported significant pressures from the second subject during the early post-operative period, while four transducers had reported pressures at a similar level for the first subject at the same time post-operative. Figure 6-1 shows the pressure data for the two subjects during gait data taken two weeks after implantation. All of the transducers reporting relatively high pressures for the first subject were active at the same time, indicating that several regions of the acetabulum were experiencing relatively high pressures, while only one transducer was recording similar pressures in the second subject, indicating that a much smaller region of the acetabulum was being loaded. Later data from the two subjects continues to show this effect, though changes occuring during the first 4-6 months post-operative tended to produce more similarity between the two subjects.

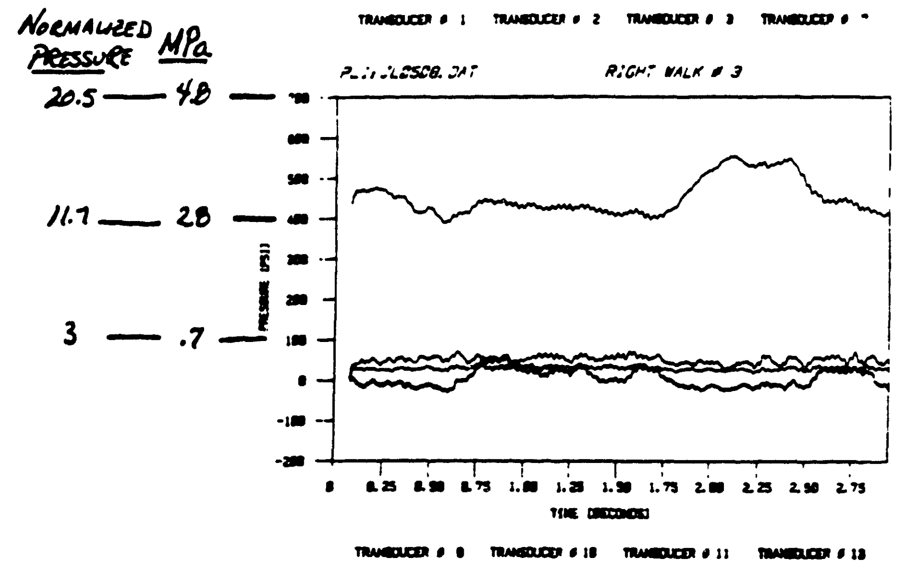
These differences in pressure distribution correspond to the Hertz stresses analytically predicted when two different diameter spherical surfaces are in contact. For rigid materials a ball smaller than the socket in which it sits will contact the socket in one location at any one time. Although the femoral head replacement is rigid, the cartilage is not and so the contact is not solely point contact, but occurs over a smaller area than it would were the ball to fit exactly. Given the consolidation of cartilage in the case of the slightly oversized prosthesis, contact occurs over a large area. Results from *in vitro* tests with instrumented prostheses and cadaver acetabula indicated the effect of prosthesis fit on pressure distribution in the acetabulum for undersized misfits of 1-3 mm in diameter. [20] The pressure data from these two subjects support the *in vitro* results, and extend them to smaller misfits and oversized prostheses.

6.2 Cartilage adaptation to pressure

Changes occurred in the pressure distribution over the implantation period for both of the two subjects. Since data is available for 5 years from the first subject and only for a year from the second subject, the data from the first subject has greater utility in discussing the adaptation process. There appear to be two phases during which the pressure distribution changes, an initial adaptation phase during the first year to 18 months post-operative and a degenerative phase late in the implantation period.

The adaptation phase was manifested differently in the two subjects, due to the different directions of prosthesis misfit. In the first subject the prosthesis was initially





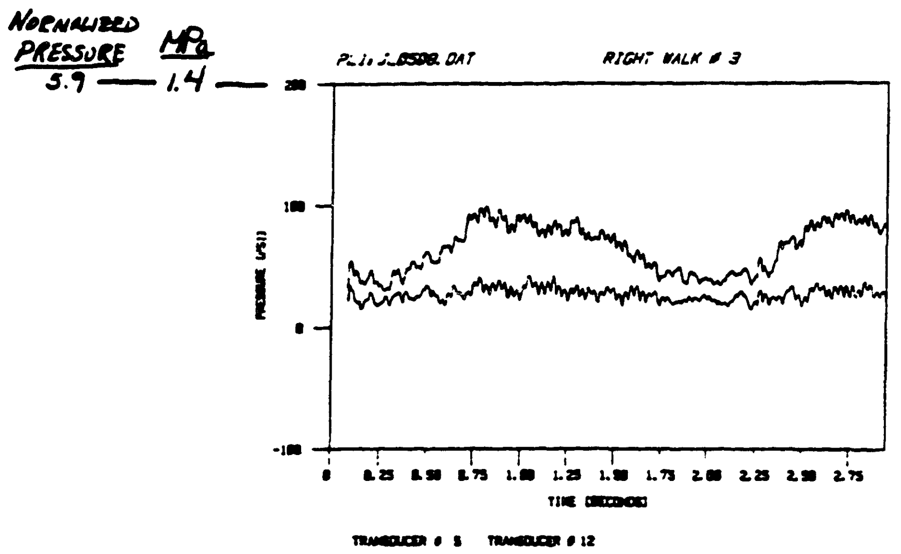


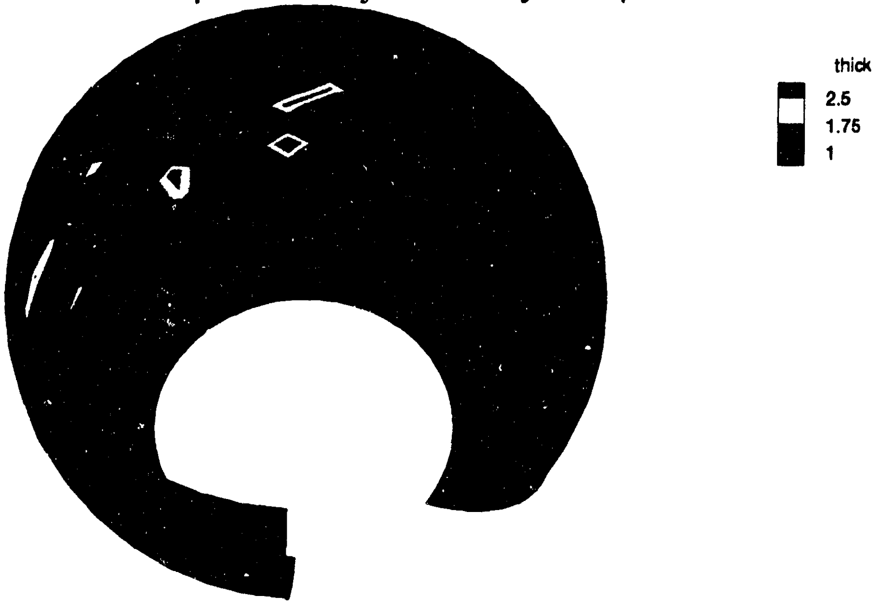
Figure 6-1: Pressures for Subject 1 walking, July 5, 1984

too large and significant pressures ($\tilde{3}$ MPa) were measured concurrently at several transducers during gait. The region over which high pressures were measured was more localized in the data taken at 11 months and 2 1/2 years post-operative than in the data taken at 6 months (figure 5-4). In the second subject the changes that occurred during the first year after undersized implantation were in the opposite direction, from data that approaches point contact at 2 weeks post-operative, to a high pressure region at 1 year which is similar in extent to that seen in the first subject at the same post-operative time (see figure 5-7).

Following the first year and up to 2 1/2 years post-operative, little change appears to take place in the location or area over which high pressures are distributed. However, data taken nearly 5 years after implantation indicate a substantial change in the extent of the acetabulum over which relatively high pressures are measured. The distribution of pressures had become much more diffuse as is evident in figure 5-4 in the data from March 1989. Relatively high pressure measurements are found at that time in the outer regions of the acetabulum. This indicates that the prosthesis has thinned the cartilage to the point where it has begun to protrude into the acetabulum.

Following the death of the first subject the prosthesis and matching hemi-pelvis were donated to the lab. The thickness of the acetabular cartilage was subsequently measured using ultrasound by M.C. Clayton [10]. A representation of his results is shown in figure 6-2. This figure presents his data in the same reference frame that the regional maximum pressure data has been presented in this thesis. Essentially he found that the acetabular cartilage was very thin, with the thickest regions anterior in the socket. The ultrasound measurement system used could not accurately measure cartilage thicknesses of less than 0.5 mm. Much of the acetabular cartilage was too thin to be measured, leading to very sparse data. Visual inspection of the acetabular surface supports the ultrasound measurement data, as the cartilage is nearly non-existent over much of the acetabulum and is also "burnished", a condition that generally occurs after prolonged contact with a metallic femoral head relacement. Measurement of normal acetabular cartilage thickness by Rushfeldt from two cadavers using the same technique indicated that the thickest cartilage was more posterior

Acetabular Cartilage Thickness
Measured post-mortem using ultrasound
Prosthesis 33 implanted in joint for 5 years previous



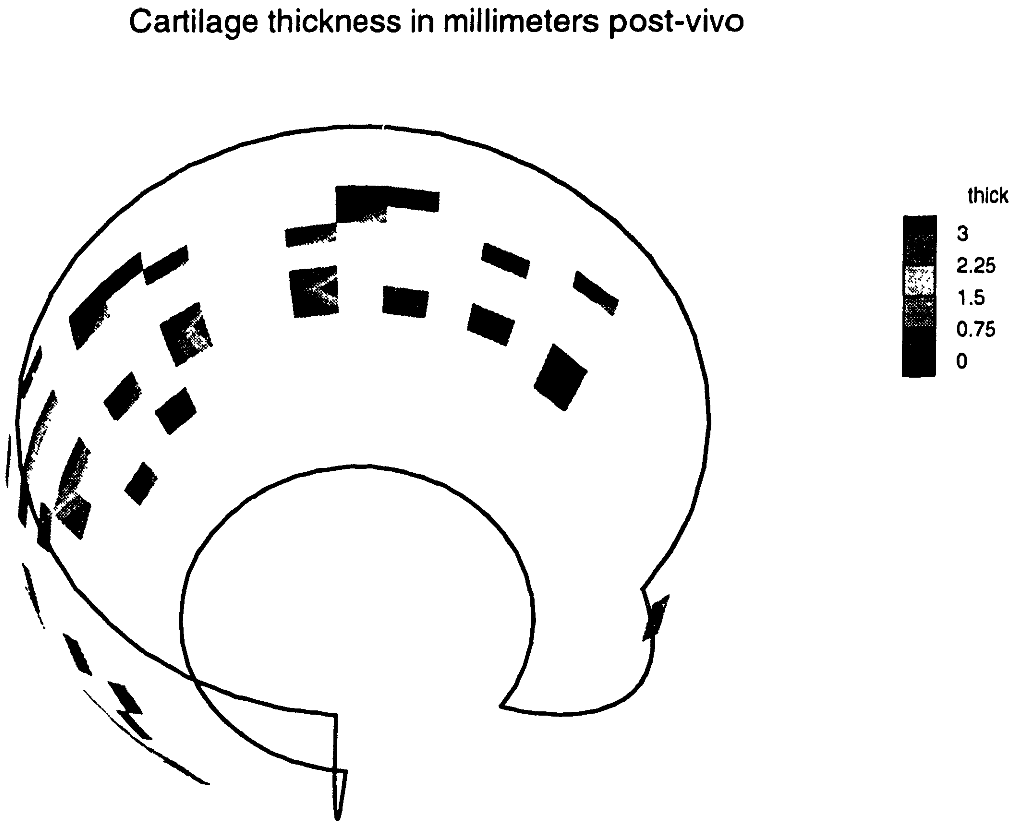


Figure 6-2: Ultrasound data on Acetabular Cartilage Thickness for Subject 1

on the acetabular surface, in much the same region that in which high pressures have been measured. [21] That the cartilage was thin in this region of the acetabulum from the first subject supports the hypothesis that the femoral head had begun to protrude.

6.3 Bone response to pressure loading

The display of regional maximum pressures on the acetabulum provides an argument in the support of Wolff's law. This hypothesis states that bone grows in response to stress so as to support the loads applied to it with a minimum of material; Wolff's observation was that the spicules in cancellous bone in the hip joint appear to form arcades that line up with the assumed principle directions of stress in the tissue. [26] The regional maximum pressure data indicates that the highest stresses are in the superior/posterior region of the acetabulum as had previously been conjectured. Figure 6-3 shows the Xray image of a right hip from and the regional maximum pressures for gait from subject 1, in approximately the same orientation. Since Xrays are absorbed by the material through which they pass, and an Xray is a negative image, the densest bone will produce the whitest image. As can be seen in this figure the greatest bone mass is in the region in which high pressures were measured during gait. While gait pressures were not the highest measured, walking and standing are much more common activities than are rising from a chair or stair climbing. Bone response to the integrated effect of load and duration seems likely.

A conceptual difficulty arises in that, if the data support Wolff's law, as they do appear to, then how does protrusion of the femoral head into the pelvic cavity occur? Shouldn't the presence of higher stresses in that region mean that bone mass in that region remains strong? So, why is this prosthesis thinning the cartilage to the point that much of the femoral head is loaded during data taken at 5 years post-operative? One possible explanation lies in the fact that the prosthesis is slightly over-sized. Perhaps the distribution of stresses over a larger area results in an abnormal loading such that the stresses are lower than normal in the high stress region. This may result in the bone mass in that region decreasing as more is formed in a less central area. This could eventually lead to the gradual inward migration of the socket area.

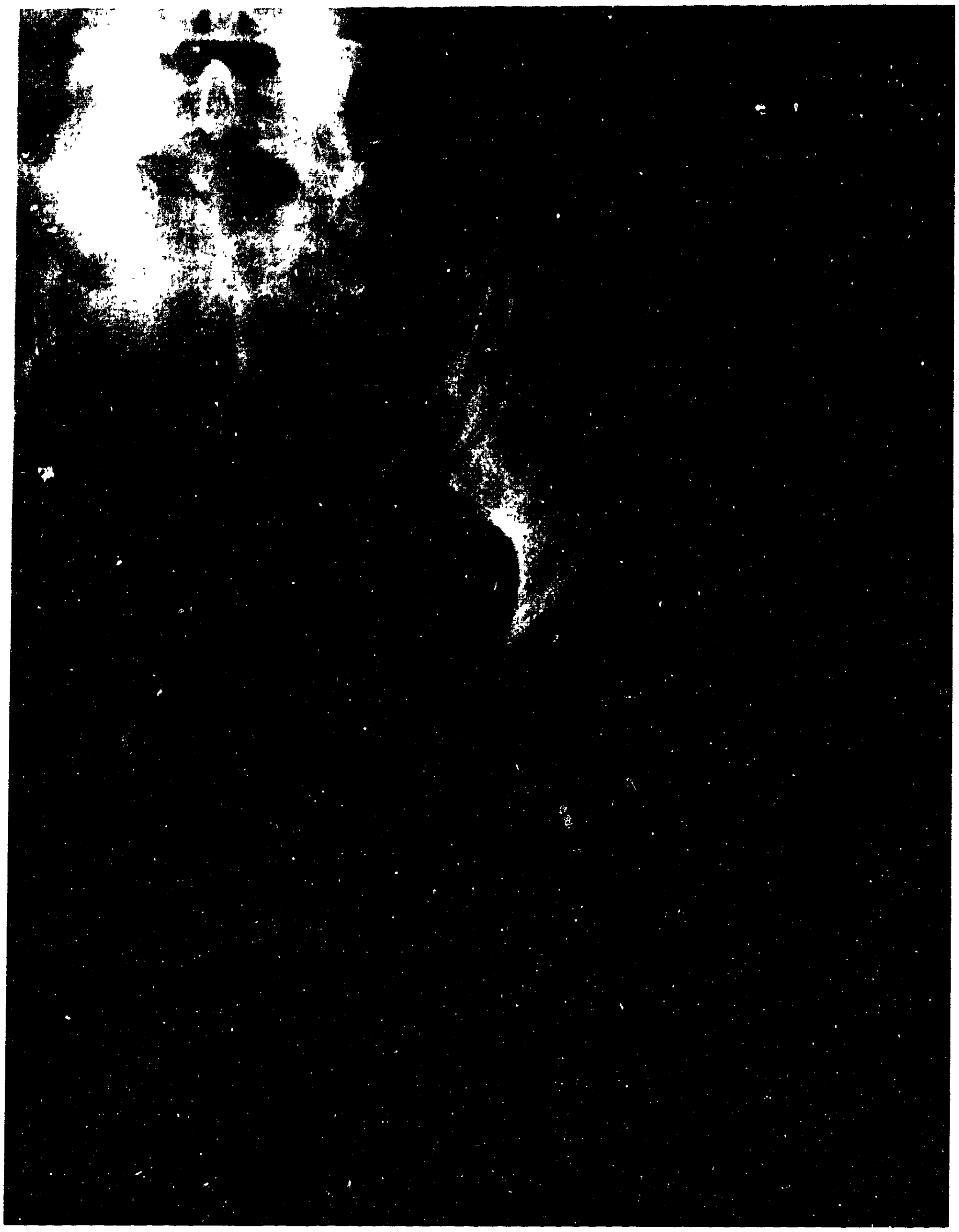


Figure 6-3: Xray of a left hip

6.4 Pressure magnitude

In addition to changes that occured in the location and distribution of pressures, there were also changes in the magnitude of pressures and normalized pressures for both subjects. The magnitude of the pressures is influenced by the rehabilitation of the subject following surgery as well as by changes in the structure and mechanics of the joint. Both subjects show some similar trends in the changes in pressure magnitude during the first year post-operative.

In the first subject the trend appeared to be that pressures increased for a particular activity until some time between 6 and 18 months, then leveled off. Maximum pressures began to gradually decrease between 2 years and 5 years post-operative for all activities.

For the first subject the maximum normalized pressure during free speed gait was reached at 6 months post-operative. At 2 1/2 years post-operative this maximum pressure was 25% lower than at 6 months. At 5 years post-operative it was nearly 40% lower than the value at 6 months post-operative. Some of this decrease can be attributed to better muscle control and coordination. In general it appears that the maximum pressures attained during an activity increased during rehabilitation as the subject became stronger, then leveled off as the subject became more skilled at doing that activity. Decreases in pressure accompany the degradation of the acetabulum.

Results from Hall et al [14] appear to indicate that cartilage synthesis increases when subjected to pressures between 5 and 15 MPa in a cyclical manner, though not for pressures between 20 and 50 MPa. As all the pressures measured were below 20 MPa it would appear that the actual magnitude of the pressures in the joint was not responsible for the cartilage degradation that took place in the acetabulum of the first subject.

For the second subject the maximum normalized pressure continued to rise for the first year post-operative, in addition to the increase in the area of the acetabulum over which those pressures were measured. During gait the maximum normalized pressures were very similar to those measured in the first subject, between 20 and

25 (MPa/(BW/Ht^2)). Although not included in this thesis, data was taken from the second subject at 15 months post-operative. The maximum normalized pressures during free speed gait at that time were slightly lower than at a year post-operative, suggesting that a leveling-off trend similar to that which was found in the first subject was beginning to take place. The maximum normalized pressures during other types of gait had also decreased slightly.

While differences between data from the two subjects are notable, the ways in which the data are similar are also important. The pressures achieved, normalized or not, achieved were very similar for both subjects. The overall pressures found for the first subject were up to 18 MPa (2600 psi), or 76 (MPa/(BW/Ht^2)) in normalized units. The highest pressure measurement made thus far from the second subject, in Sepember 1992, was 18.2 MPa, or 82.3 (MPa/(BW/Ht^2)) during descending stairs at 80 BPM. This test was not presented in this thesis because the kinematic and pressure data was not synchronized on that day. The maximum pressure measured from the second subject during the data acquisition sessions presented here was 16.2 MPa, 73.4 in normalized units, during jumping off a 7" step. During free speed gait the maximum normalized pressures measured during the first year post-operative were between 22 and 26 for both subjects. The actual maximum pressures measured were between 5 and 6 MPa for both subjects. These similarities exist in spite of gender, morphological, and neuromuscular control differences between the subjects and differences in the way tests were performed.

The measurement of such similar pressures, in similar regions of the acetabular surface, from two very different subjects, substantiates the validity of the results obtained. In addition, the *in vitro* calibration performed on the first implanted prosthesis post-*vivo* establishes that the measurements made with that prosthesis *in vivo* were accurate.

Reviewing the changes that occured over time for both subjects in both the pressure distribution and pressure magnitude, the presence of three stages during the implantation period appears. The first 6-11 months are an initial rehabilitation time, during which the cartilage adapts to the replacement femoral head. A more advanced rehabilitation period follows, with some overlap, taking place between 6 months and 18 months, during which the patient reaches a steady-state strength and coordination level when performing different activities at different times. Following this period there is a gradual decline in the pressure measurements due to cartilage degeneration.

Chapter 7

Conclusions

Nearly 100,000 femoral head endoprostheses are implanted in the U.S alone each year. A better understanding of how these prostheses perform and how they fail will have an impact on the lives of an enormous number of people. A number of conclusions can be drawn directly from the data obtained from these two instrumented prostheses. Additionally, these results can be used to support or challenge theories on joint lubrication and wear. Finally, since these results provide unique information on the actual *in vivo* loading in a human joint the data presented here are expected to be of value to other researchers investigating cartilage, synovial joints, bone, and musculo-skeletal analysis and experiments. While there have been only two subjects thus far, the similarites between the subjects' data are reassuring, while the existence of data from more than one person nonetheless provides the opportunity to separate subject-specific results from those which are more global. Recalibration of the first prosthesis post-*vivo* established the accuracy of data previously obtained *in vivo*, and validated the entire instrumented prosthesis design and data acquisition and processing system.

The major conclusions of this thesis are that: a misfit of less than 1 mm in prosthesis diameter can significantly influence the pressure distribution in the human hip; the location of high pressures on the acetabulum supports Wolff's law; significant changes in pressure distribution and magnitude occur over the implantation time of a prosthesis; the normalized pressures measured in both subjects were similar; and muscle co-contraction during routine maneuvers strongly influences the loading, and thereby pressures, in the human hip.

The collaborating surgical team took great care was taken in fitting both of these subjects with their prostheses. Even so, the prosthesis implanted in the first subject was 0.6 mm over-sized for that person, while the second subject was implanted an 0.7 mm under-sized prosthesis. Considering the care taken to match prosthesis and subject and noting that standard femoral head replacement prostheses are only available in 1 mm increments in diameter, it is likely that a very large per cent of individuals are fitted with a prosthesis that does not quite match their natural femoral head. The evidence from this research is that even such small mismatches can and do affect the distribution of load in the human hip joint.

The high pressures measured in the two subjects in this study were concentrated in specific regions of the acetabulum for each movement studied, and were generally located in the superior-posterior region. Wolff's "law", essentially contends that bone grows so as to optimally support the applied stresses with a minimum of tissue. He based his finding on observation of the structure of cancellous bone in the hip and on assumptions of the stress distribution in the joint. The data of this study support Wolff in that the location of high pressures in the joint during gait corresponds well with the most dense region of bone in the pelvis.

The distribution of pressures on acetabular cartilage changed over the implantation period for both subjects. High interior and central pressures became less localized during the first year post-operative for the (second) subject fitted with a under-sized (0.7 mm) prosthesis. The pressure distribution for the first subject fitted with an over-sized prosthesis also changed during the first post-operative year. Here the higher pressures initially were distributed over a relatively large area, with the highest pressures in the outer part of that region. Later in the year the highest pressures were measured more centrally. Adaptation of the cartilage in the acetabulum to the rigid replacement femoral ball clearly accounts for the changes seen during the first year post-operative; different geometrical adaptations are a direct consequence of the fit of the two prostheses to the joint. Late in the implantation period of the first subject the distribution of relatively high pressures became diffuse and there was no distinct region of high pressures. Degradation of the acetabular cartilage had probably occurred by this time and is believed to be the reason for the change in the distribution of high pressures.

Changes also occured in the magnitude of the pressures measured during the time the prostheses were implanted. The maximum pressures reported increase during the first year to 18 months post-operative, then level off and decrease later in the implantation period. The asymptote of maximum pressure was found to occur at different times post-operative for different movements. This trend follows the regimen of the rehabilitation process, the more strenuous the movement the longer the post-operative period before pressure measurements reached steady-state. Trends observed in these changes during the first year post-operative are similar for both subjects. Later in the implantation period the maximum pressures measured decreased significantly; accompanied by the more diffuse loading pattern mentioned above.

The pressure measurements from both subjects were similar in many ways. Data from the second subject substantiates the results previously found from the first subject. [8] Irregular pressure distributions are routine in both subjects. The magnitude range of pressures measured was similar for the two subjects, in both cases the highest pressure recorded was about 18 MPa. Maximum pressures during free speed gait are between 5 and 6 MPa for both subjects during the first year post-operative. Normalizing by body weight and height, the maximum values were between 23 and 25 for free speed gait during the first year post-operative for both subjects. The maximum pressures during other movements were not as similar due to subject-specific factors, but the range of measurements was almost the same. The locations on the acetabulum on which high pressures were measured were also similar for the two subjects in all movements.

Muscular co-contraction has been established as a major determinant of the resultant load at the hip joint. Early observation of high pressures while rising from a chair, prior to the measurement of large forces on the forceplate, indicated the importance of muscle forces on the loading experienced by the joint. That information is now supported by results from a number of movements during tests with the second subject, from stair-climbing, stair-descent, and single-leg stance in particular. The pressures measured during single-leg stance were more than twice as large as the pressures measured in stance phase of normal gait.

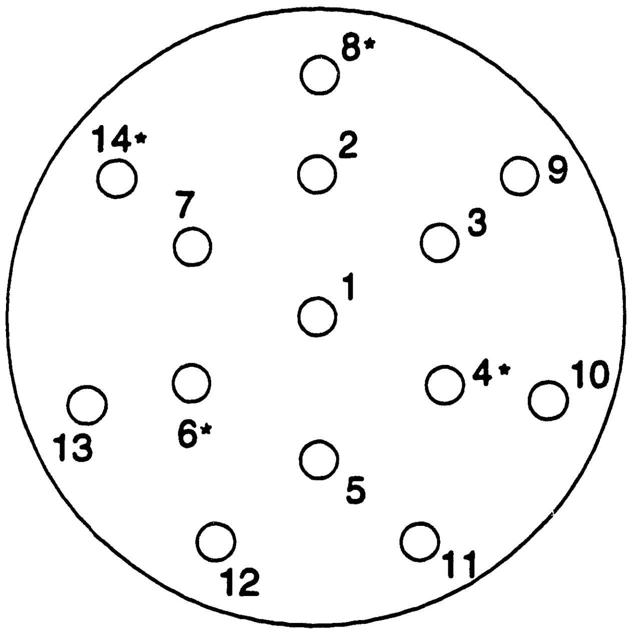
No research is ever actually complete; this project is no different. There are several ways in which this research should be continued: future data from the second subject should be acquired and analyzed; data from future instrumented prostheses should be compared to the results from these subjects; results from this work should be used in considering lubrication and wear phenomena in synovial joints. I hope that someone will continue to investigate the data acquired *in vivo* from these instrumented prostheses, and that the results presented in this thesis will be useful to other researchers in determining parameters to use in studies of articular cartilage and synovial joints.

Appendix A

Transducer Locations and Numbering

Figure A shows the transducer numbering and locations for prosthesis 33. The view shown is looking into the inner surface of the femoral ball hemisphere. Figure A shows a similar view for prosthesis 43. Table A.1 presents the transducer locations in prosthesis latitude and longitude, where transducer 1 is at one "pole", or 0° longitude (ϕ). 90° latitude (θ) is in the same direction as the positive y axis in prosthesis coordinates, toward transducer 8 for prosthesis 33, and 0° latitude is clockwise from that direction.

Location of pressure transducers in prosthesis 33, looking at inside surface of open hemisphere



<— anterior for L hip   anterior for R hip —>

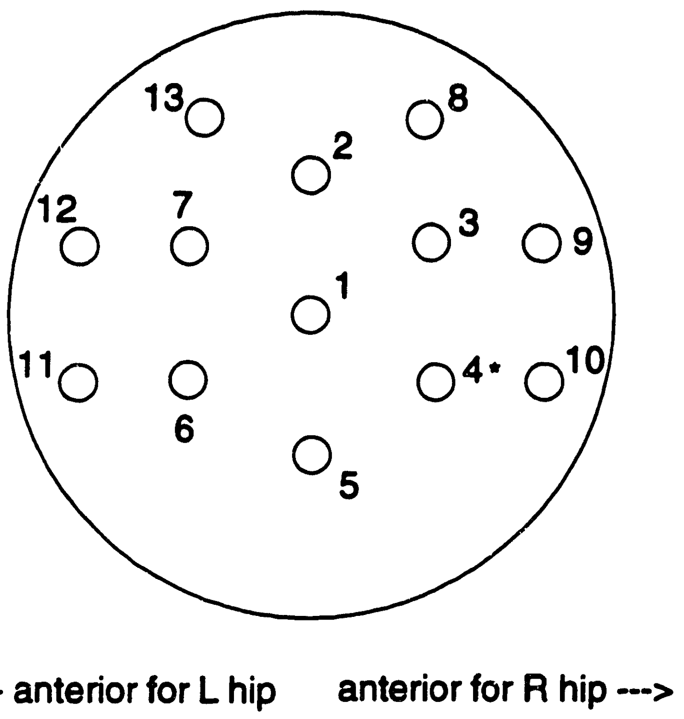
* transducer not working or non-linear response to pressure

Figure A-1: Transducer locations in Prosthesis 33

Table A.1: Transducer Locations in theta and phi

Prosthesis	Transducer	θ	ϕ
33	1	0	0
	2	90	31
	3	30	31
	4	330	31
	5	270	31
	6	210	31
	7	150	31
	8	90	62
	9	38.6	62
	10	347.2	62
	11	295.75	62
	12	244.3	62
	13	192.9	62
	14	141.4	62
43	1	0	0
	2	90	31
	3	30	31
	4	330	31
	5	270	31
	6	210	31
	7	150	31
	8	60	53
	9	17	60
	10	343	60
	11	197	60
	12	163	60
	13	120	53
	Temp	120	20

Location of pressure transducers in prosthesis 43, looking at inside surface of open hemisphere



* transducer not working or non-linear response to pressure

Figure A-2: Transducer locations in Prosthesis 43

Appendix B

Calibration of Pressure Instrumented Prostheses

The calibration process for the instrumented prostheses includes calibration of the electrical output of the pressure transducers to applied pressure, but also must account for the effect of temperature on the output of those transducers. Other variables that may affect the transducer output are: time, power level supplied to the prosthesis, and length of time over which the prosthesis is powered. The transducer offsets are known to be affected by temperature and are known to drift over long periods of time (months). Powering the prosthesis for an extended period of time was expected to only affect prosthesis 33 which dissipates more power and therefore may heat up slightly if powered continuously for a long time. In addition, since the transducer gains may change over the prosthesis lifetime, recalibration of prosthesis 33 post-mortem was done in part to check this possibility.

The major step in calibration of the pressure transducers is calibration of their output with respect to pressure. This is done prior to implantation of the prosthesis, but following the removal of prosthesis 33 it was also recalibrated because it had been over five years since the previous calibration was done and because some of the pressures measured by it were above the level for which it had been calibrated pre-implantation. A new pressure vessel was designed and constructed in order to calibrate the prostheses to higher pressures; the one used previously for number 33 pre-implantation relied on a restraint mechanism which exerted too high a stress on the cast stem region of the prosthesis.

All pressure calibration was done hydrostatically in oil, with the entire ball stressed. The pressure vessel is shown in Figure B-1, with a prosthesis shown in the cutaway view. The stem extends out of the vessel to allow data transmission and powering of the electronics in the prosthesis. n

Figure B-2 shows the entire set up for pressure calibration. A dead-weight tester was used for pressurization, up to 2500 psi (17.2 MPa) in increments of 25 psi (0.17 MPa). An external pressure gage was used primarily to check for any rapid transients in pressure, and also to double-check the dead-weight tester. During later tests the pressure vessel also could be heated via a coil which surrounded the base of the pressure vessel and was attached to a power source with thermostat control. The heating element was only used with prosthesis 43, in which there is a thermistor; there was no means for monitoring the internal temperature with prosthesis 33.

Data acquisition from the prosthesis during calibration tests was similar in equipment and procedure to data acquisition from prosthesis 43 *in vivo* as described in Chapter 3. An AT&T 6300+ PC was used, data was saved at 500 Hz for transducers in prosthesis 43 and 250 Hz for the transducers in prosthesis 33. Statistics for each data set were found; for calibration of transducer output with respect to pressure or temperature the mean of each data set is plotted vs. the variable of interest.

Calibration was done in psi (because the dead-weight tester weights and piston were calibrated for psi); four different tests were done on prosthesis 43 and it was calibrated to 2500 psi (17.2 MPa), two tests were done on prosthesis 33 and it was calibrated to 2000 psi (13.8 MPa). Table B.1 summarizes the days and the test formats for both prostheses. For prosthesis 33 both tests were at room temperature (78 °F, 25.5 °C) and one was up to 1000 psi (6.9 MPa) in increments of 50 psi (0.34 MPa) and back down in large increments (about 200 psi (1.4 MPa)). The second test with prosthesis 33 was up to 2000 psi (13.8 MPa) in increments of 100 psi (0.69 MPa) below 1100 psi (7.6 MPa) and in increments of 50 psi (0.34 MPa) above 1100 psi (7.6 MPa). During the second test, when pressure was decreased it was done in

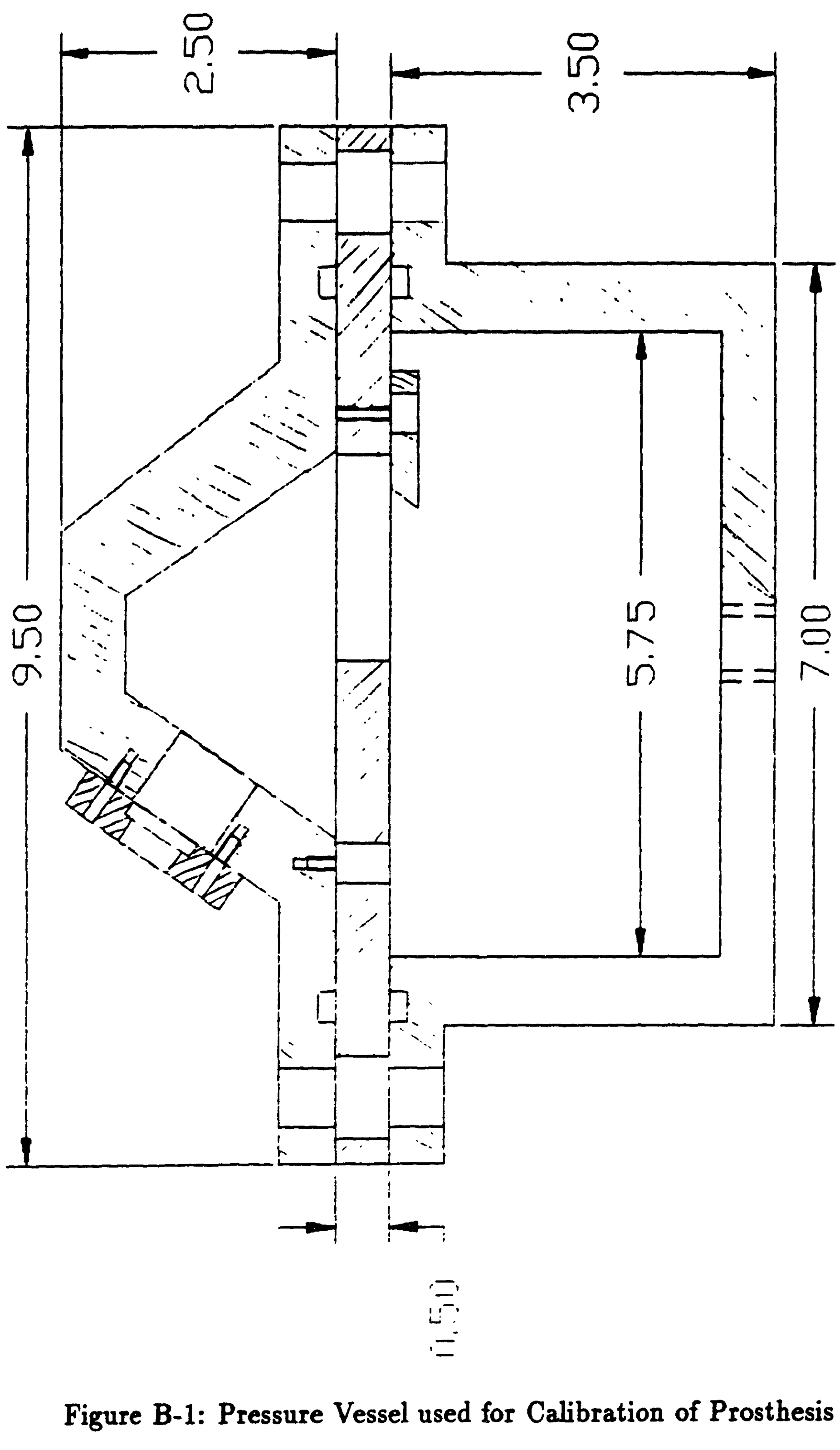


Figure B-1: Pressure Vessel used for Calibration of Prosthesis

Hydrostatic Pressure Calibration Setup

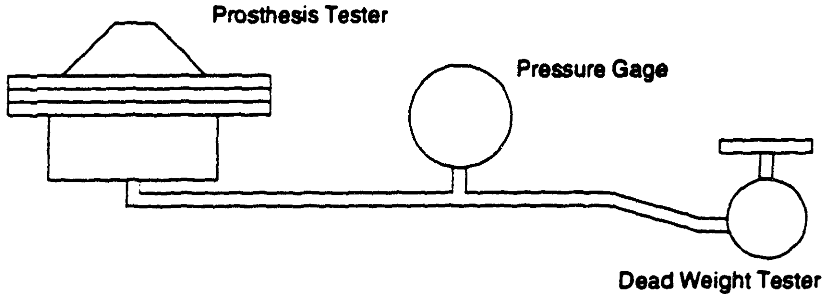


Figure B-2: Experimental Setup for Pressure Calibration of Prosthesis

Prosthesis	Date	Range [psi]	Increment [psi]	Temp. [°F]
33	Nov. 29, 1990	0-1000	50	78
		1000-0	200	78
33	Dec. 10, 1990	0-2000	100	78
		2000-0	100	78
43	Jan. 17, 1991	0-1000	50	78
		1000-0	50	78
43	Jan. 18, 1991	0-2500	100	78
		2500-0	100	78
43	Feb. 19, 1991	0-1000	100	98
43	Mar. 5, 1991	0-1000[a]	100	98

[a]Four cycles between 0 and 1000 psi

Table B.1: Pressure calibration tests

increments of 100 psi (0.69 MPa) between 2000 and 1000 psi (13.8 and 6.9 MPa) and in increments of 200 psi (1.4 Mpa) below 1000 psi (6.9 MPa).

Prosthesis 43 was calibrated with respect to pressure on four days, using four different protocols. Two of these tests were done at room temperature (78 °F, 25.5 °C) and two were attempted with the temperature of the pressure vessel raised to approximately body temperature. Of the two tests performed at room temperature, the first one was done to 1000 psi (6.9 MPa) in increments of 50 psi (0.34 MPa) both on the way up and on the way down, the second test was up to 2500 psi (17.2 MPa) in increments of 100 psi (0.69 MPa) for both increasing and decreasing pressure. For the first test with the pressure vessel heated the temperature of the vessel was raised to around 98 °F (36.7 °C) but no control was exerted over the temperature as the pressure was raised to 1000 psi (6.9 MPa) in 50 psi (0.34 MPa) increments. During the second test with the heated pressure vessel the temperature was monitored and was more stable than in the first heated test. The second heated test consisted of 4 cycles in pressure between 0 and 1000 psi (6.9 MPa). Only the first cycle was used in pressure calibration.

Results for the pressure calibration of prosthesis 33 are shown in figures B-4 through B-17. These figures show the means of the data taken at each pressure during the test run. Also shown in these figures is the best fit line to these means. The transducers which have a linear relationship with pressure are the only ones for which results herein have been reported. For these transducers the relationship of output to pressure can be described by an offset and a gain. Table B.2 summarizes the results of pressure calibration of prosthesis 33, giving the post-vivo gains and also presenting the gains used during the implantation period. Table B.3 presents the difference in pressure estimates that would result from the different gains.

Results from the pressure calibration of prosthesis 43 are presented in figures B-18 through B-30. Only transducer 4 appeared to be nonlinear with respect to pressure. Figure B-3 shows the temperature during the pressure calibration tests, these temperature estimates were obtained from the thermistor in prosthesis 43 which had previously been calibrated in a series of thermal calibration tests. Table B.4 sum-

Transducer	**1**	**2**	**3**	**5**	**7**
Nov. 29, up	0.488	0.818	0.622	0.755	0.684
Nov. 29, down	0.513	0.813	0.622	0.752	0.668
Dec. 10, up	0.482	0.776	0.590	0.755	0.629
Dec. 10, down	0.492	0.768	0.595	0.742	0.649
Nov. 29, all	0.498	0.819	0.625	0.756	0.680
Dec. 10, all	0.487	0.775	0.595	0.751	0.642
ex vivo	0.493	0.795	0.61	0.75	0.66
pre-implant	0.49834	0.7955	0.61685	0.65058	0.62767
Transducer	**9**	**10**	**11**	**12**	**13**
Nov. 29, up	0.521	0.689	0.840	0.902	0.652
Nov. 29, down	0.573	0.742	0.819	0.900	0.684
Dec. 10, up	0.551	0.726[a]	0.805	0.937	0.684
Dec. 10, down	0.576	0.751[b]	0.812	0.928	0.681
Nov. 29, all	0.544	0.715	0.834	0.902	0.667
Dec. 10, all	0.561	0.737[c]	0.808	0.932	0.683
ex vivo	0.55	0.73[d]	0.82	0.917	0.675
pre-implant	0.59583	0.78429	0.94738	0.91537	0.74245

[a]Below 1300 psi
[b]Below 1300 psi
[c]Below 1300 psi
[d]Below 1300 psi

Table B.2: Gains of prosthesis 33 pressure transducers

Transducer	ΔP [MPa]	ΔP [psi]
1	0.01	1.45
2	0.0	0.0
3	0.011	1.6
5	-0.132	-19.1
7	-0.049	-7.1
9	0.08	11.6
10	0.074	10.7
11	0.16	23.2
12	-0.002	-0.29
13	0.1	14.5

Table B.3: Difference in pressure estimates for different gains

marizes the results of pressure calibration, presenting the gains found. Offsets were found to vary with time for this prosthesis as well as for prosthesis 33. The variation in temperature during the first heated test appeared to cause a variation in transducer outputs; this data was ultimately not used in calculation of transducer calibration.

The differences found in the transducer gains for prosthesis 33 were on the same order as the different gains found prior to implant. Evaluation of the maximum pressure reading by each transducer and the maximum error in gain revealed that the largest changes would be about 0.3 MPa. This is on the order of the noise in transducer output. The transducer with the largest changes in gain did not measure very high pressures, and the transducer which transmitted the highest readings appears to have a very stable gain. Only one transducer was found to have different behavior at pressures above the level to which it had previously been calibrated. This transducer (shown in figure B-13) appears to have a change in gain above 1300 psi (8.96 MPa). Fortuitously this transducer has never reported outputs above or near that level.

Pressure tests on prosthesis 43 at body temperature as well as room temperature indicated that transducer gains were independant of the temperature of the prosthesis, as had been expected from results of calibrating other prostheses.

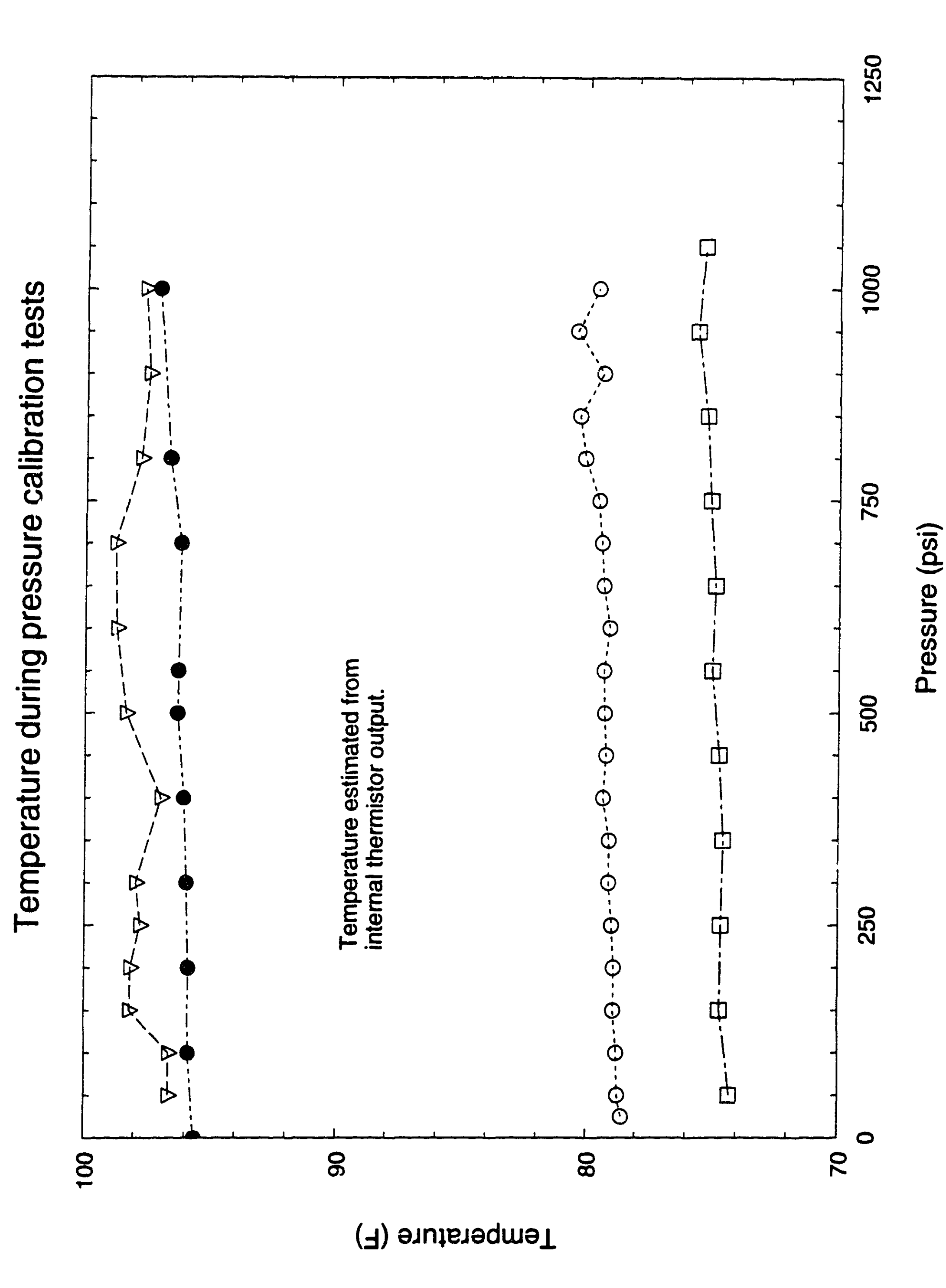


Figure B-3: Temperature during Pressure calibration tests

Transducer	Gain [cmv/psi]
1	1.22
2	0.987
3	0.678
5	0.976
6	1.23
7	0.960
8	0.870
9	0.795
10	0.885
11	0.814
12	0.839
13	0.888

Table B.4: Prosthesis 43 pressure transducer gains

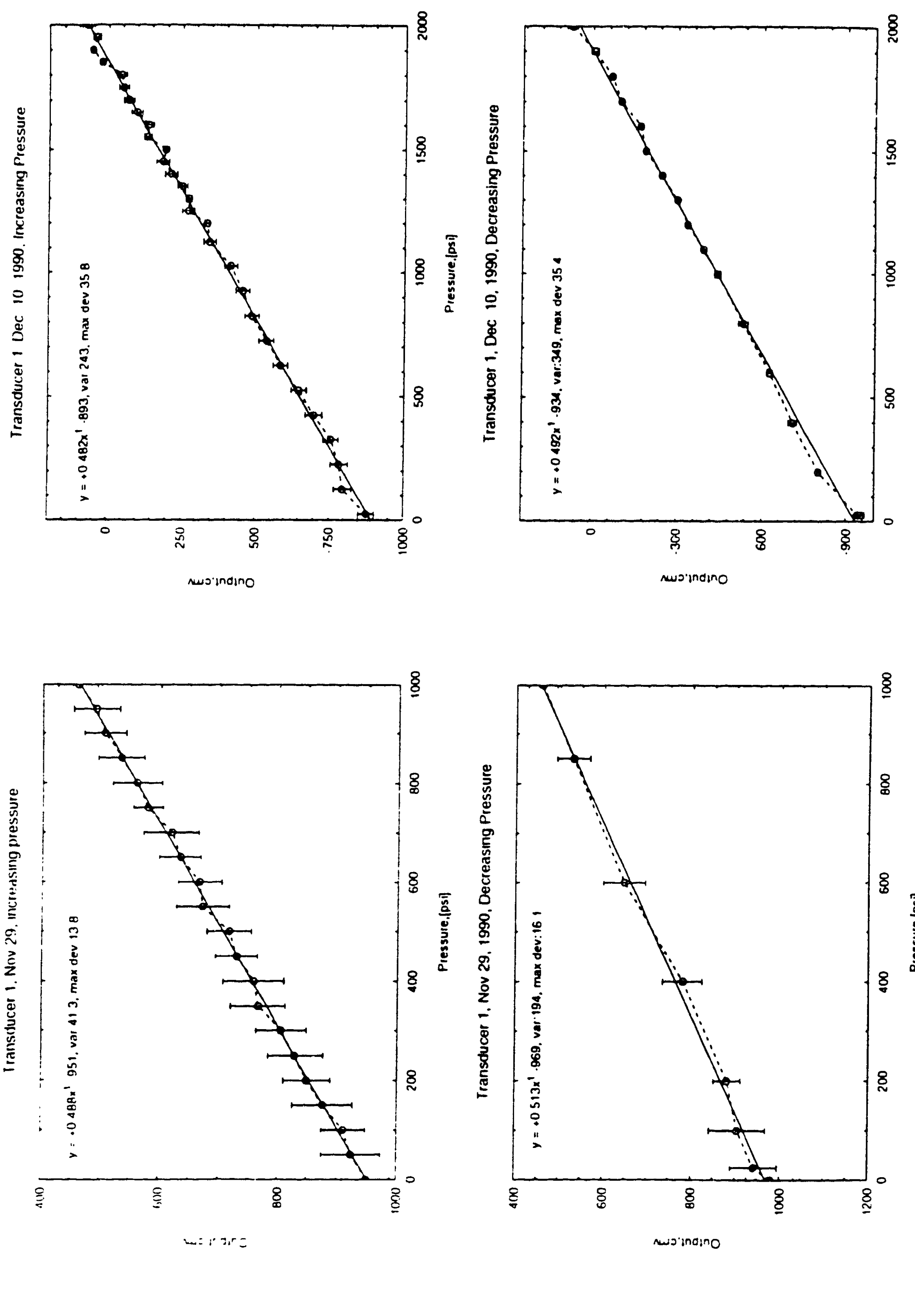


Figure B-4: Pressure Calibration of Prosthesis 33; Transducer 1

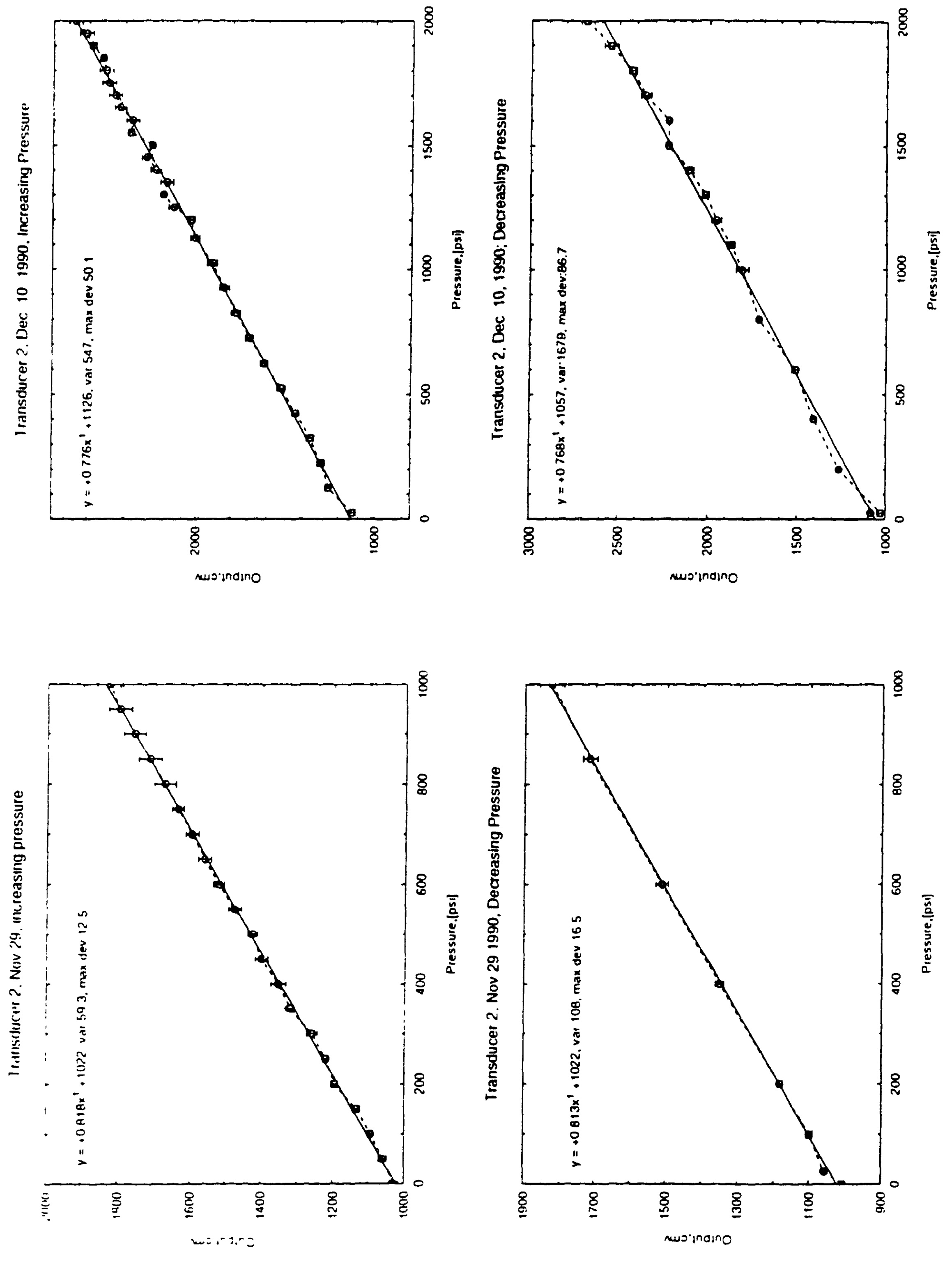


Figure B-5: Pressure Calibration of Prosthesis 33; Transducer 2

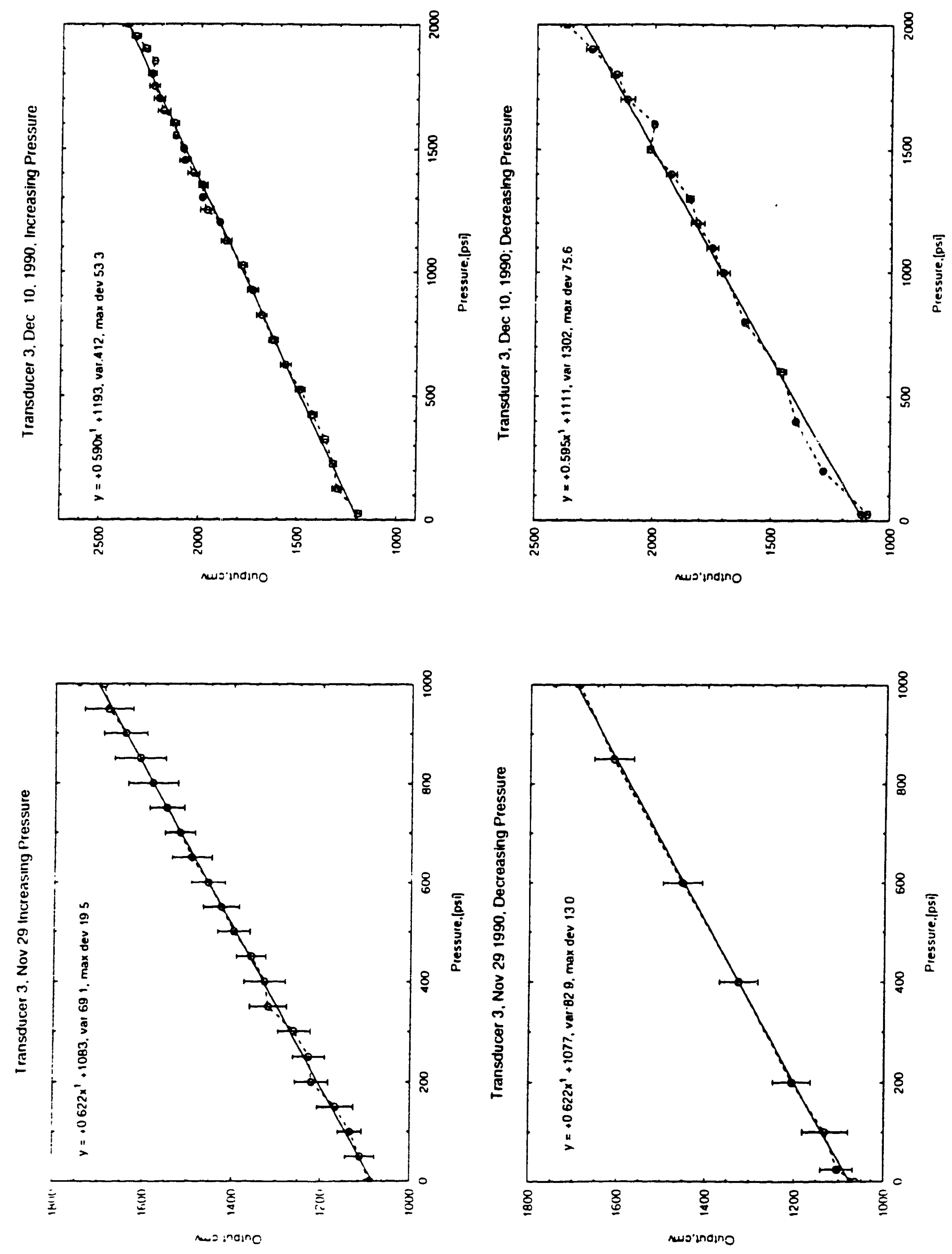


Figure B-6: Pressure Calibration of Prosthesis 33; Transducer 3

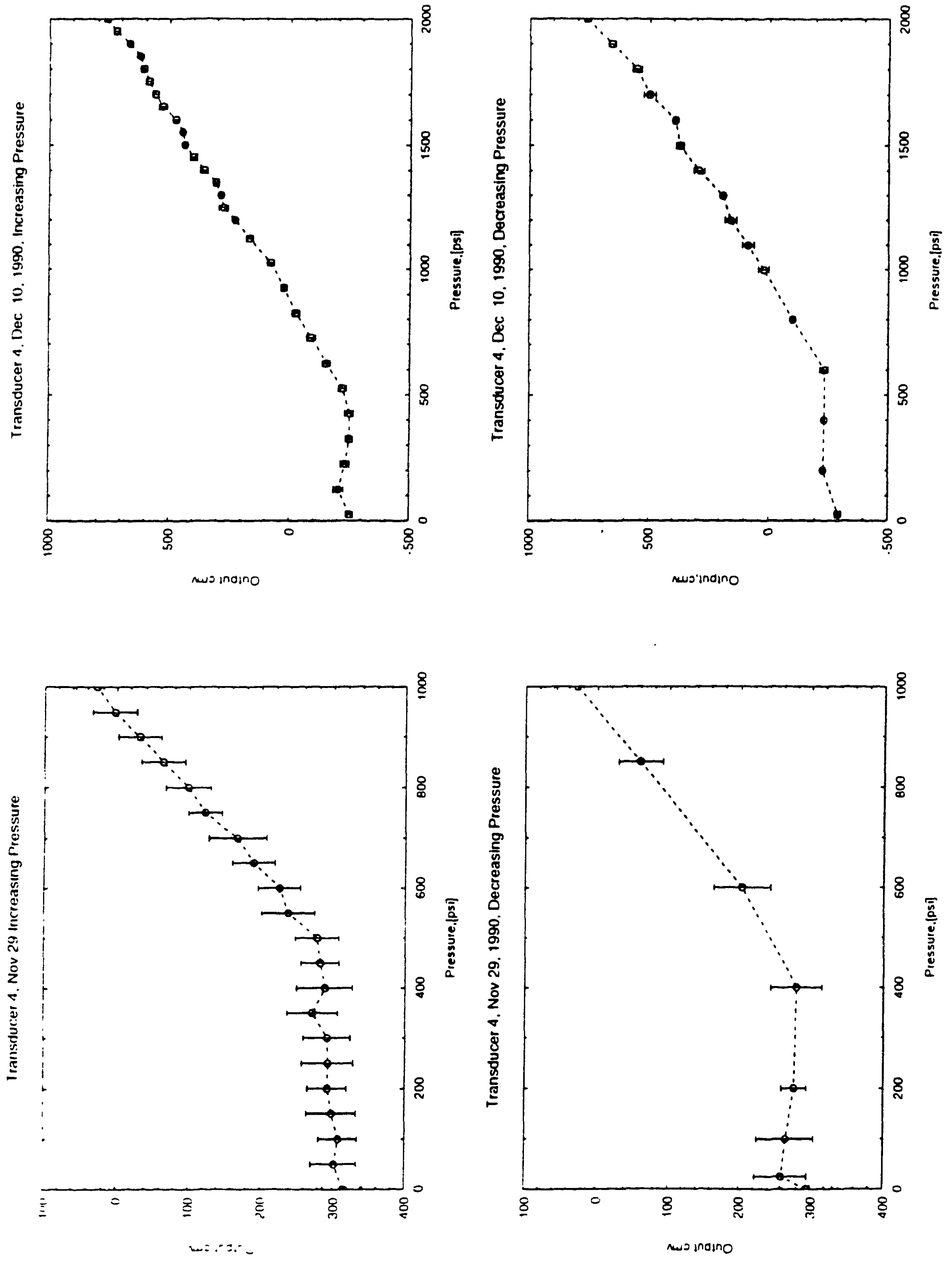


Figure B-7: Pressure Calibration of Prosthesis 33; Transducer 4

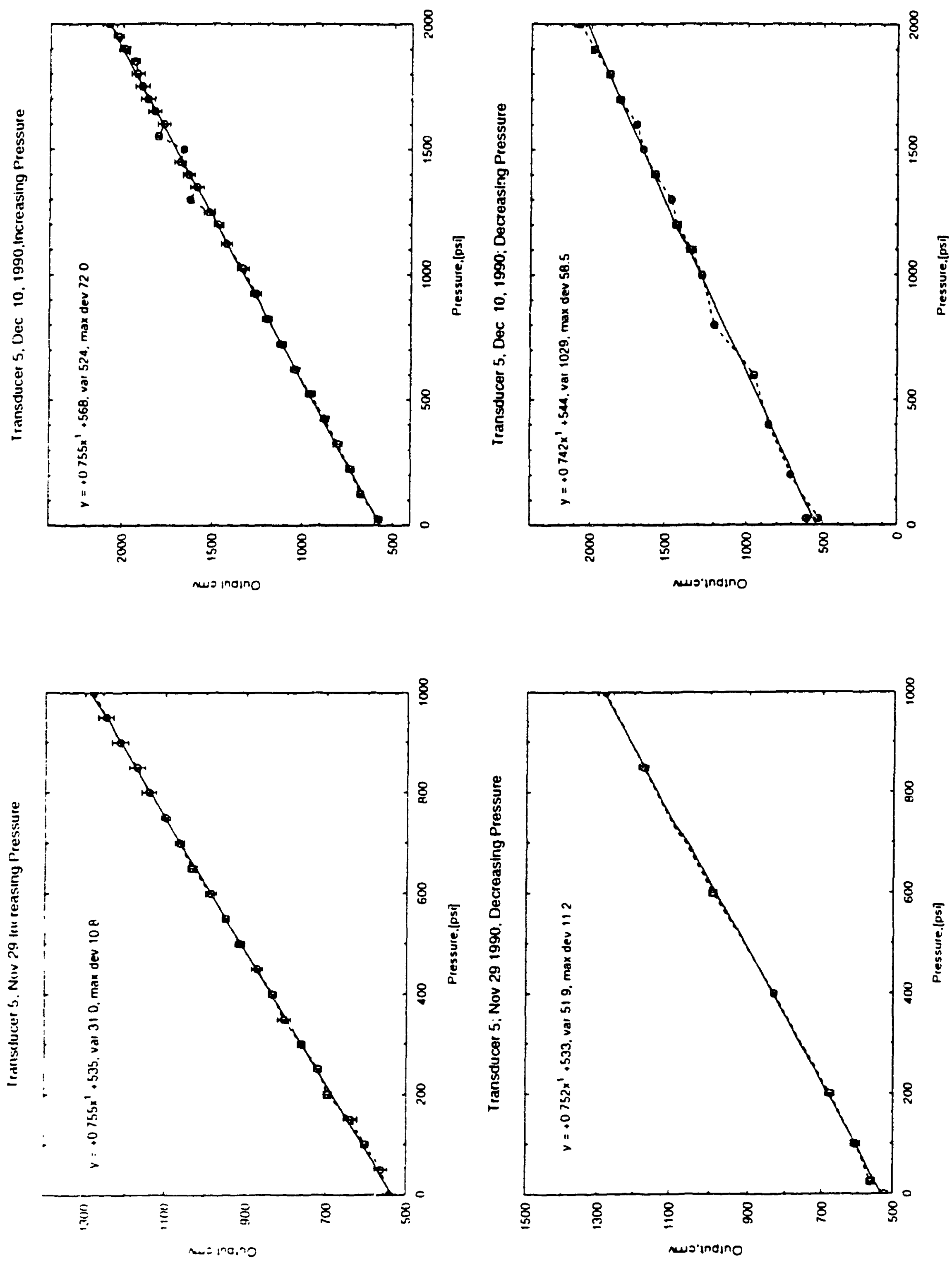


Figure B-8: Pressure Calibration of Prosthesis 33; Transducer 5

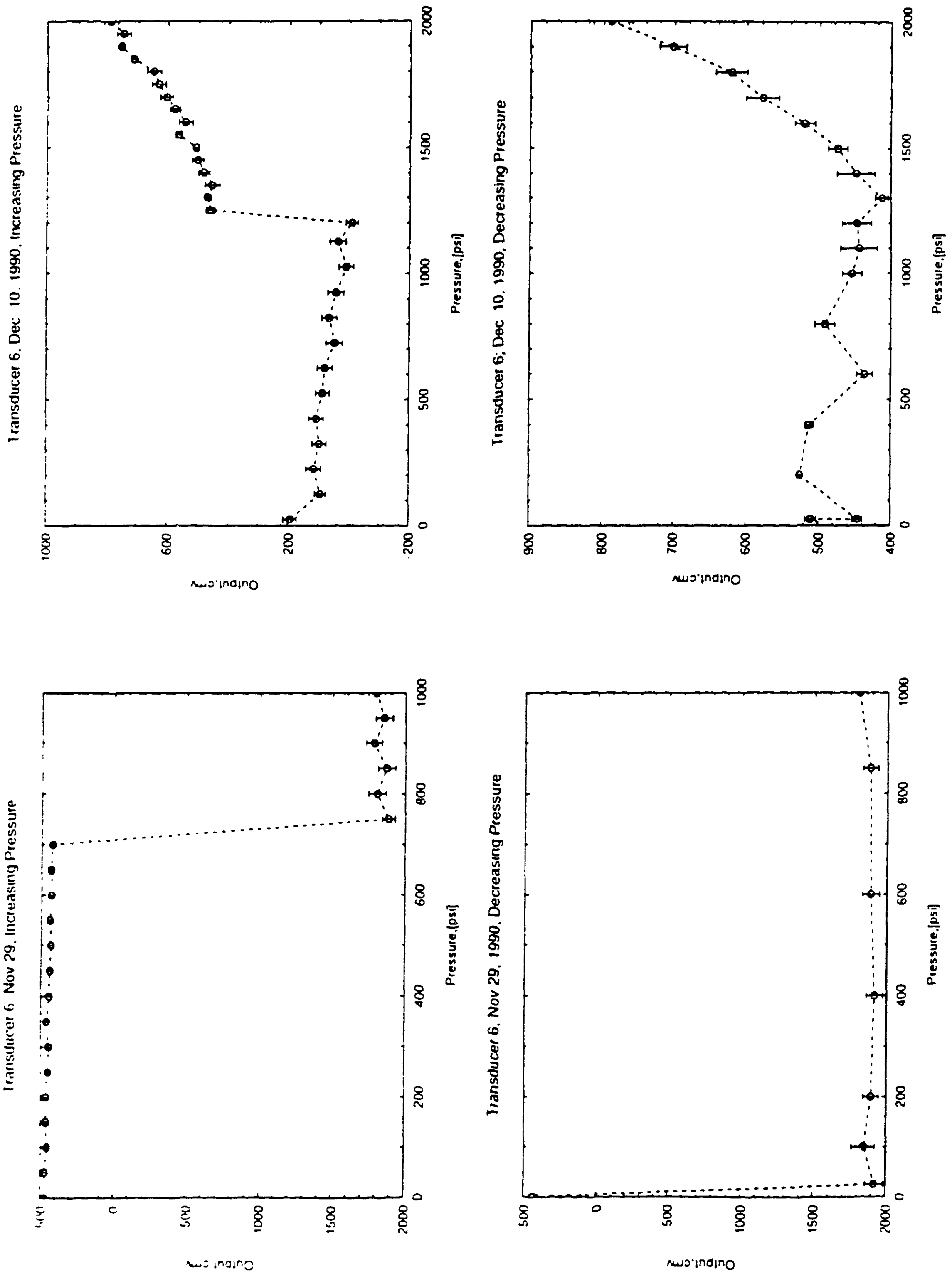


Figure B-9: Pressure Calibration of Prosthesis 33; Transducer 6

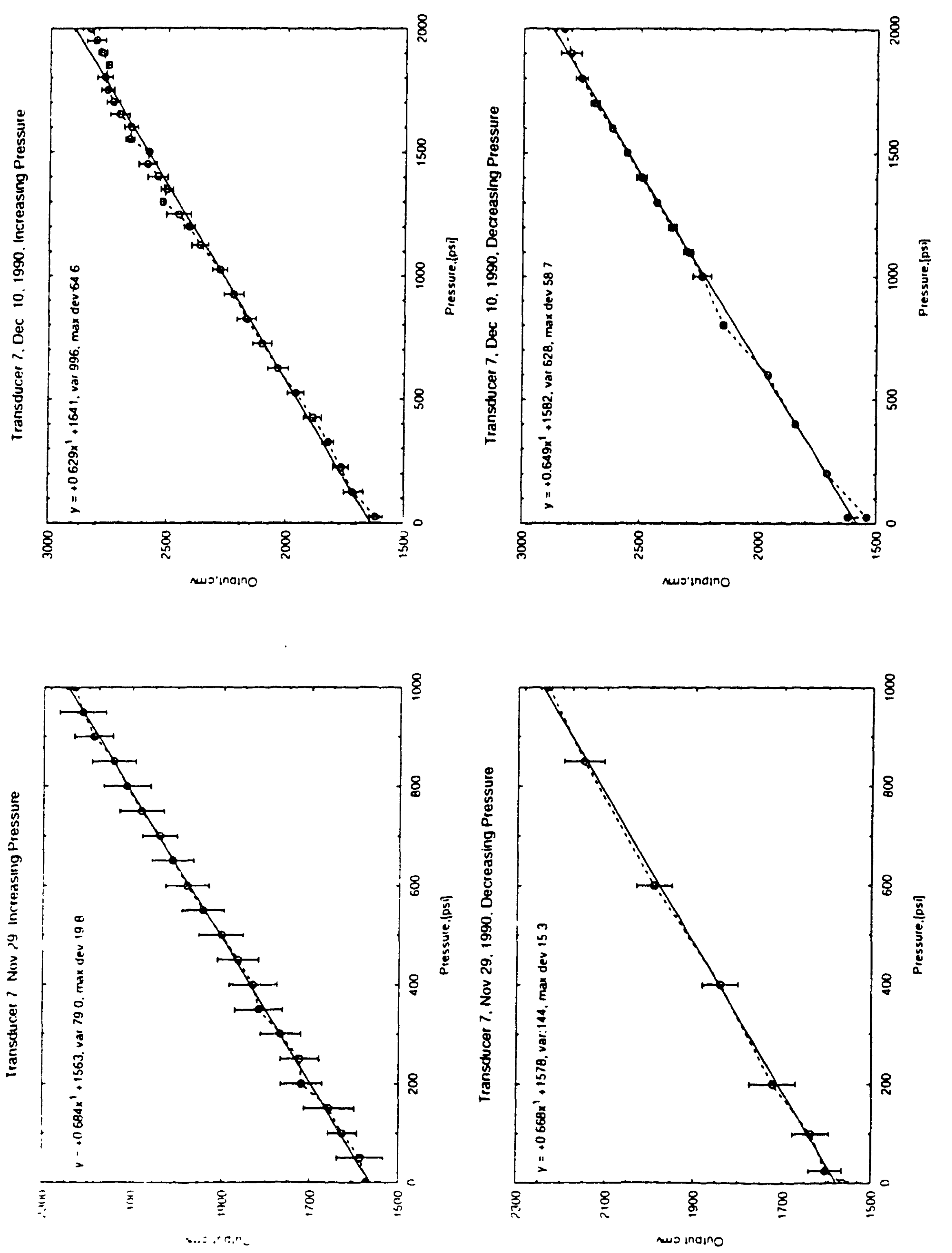


Figure B-10: Pressure Calibration of Prosthesis 33; Transducer 7

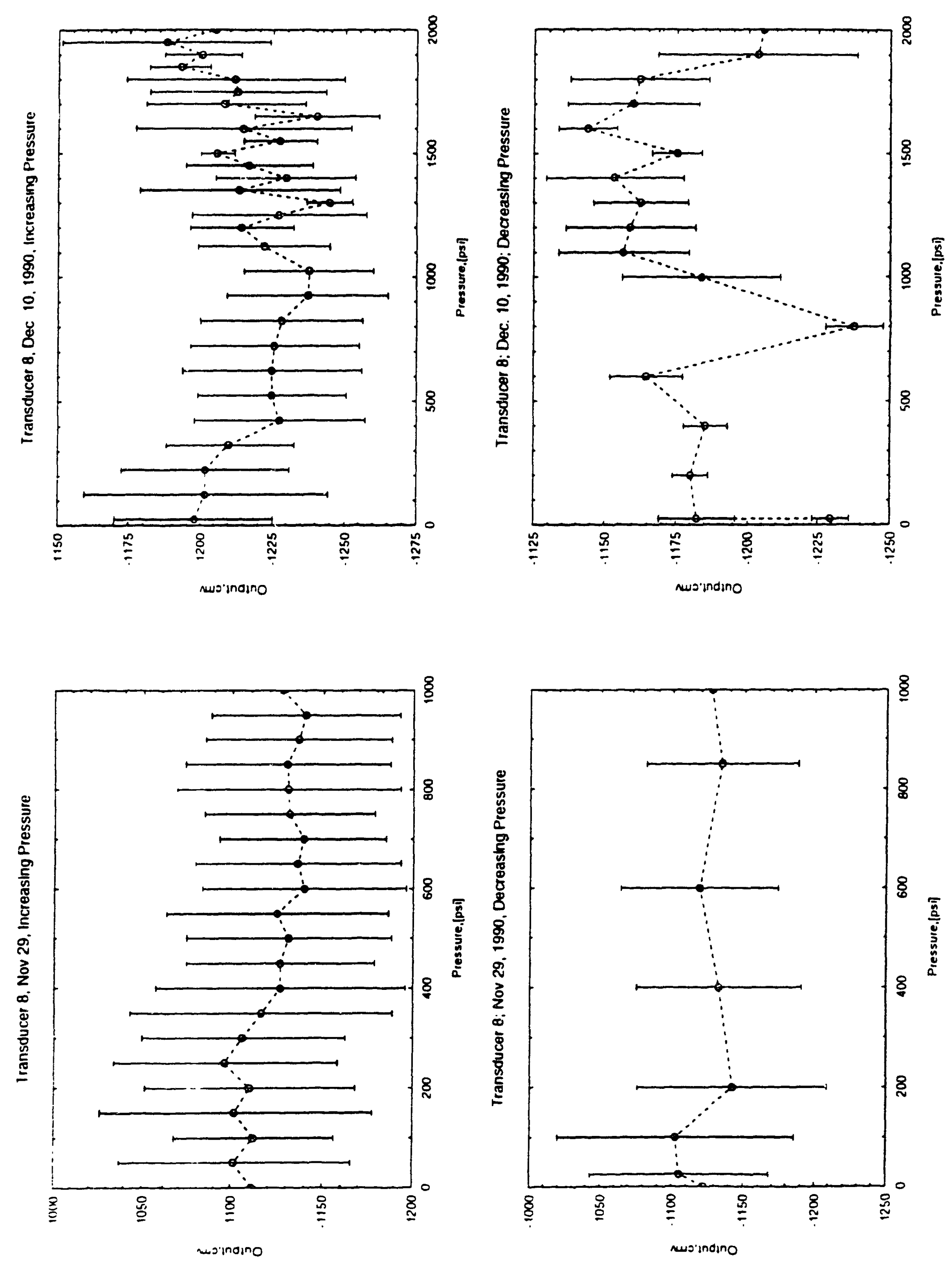


Figure B-11: Pressure Calibration of Prosthesis 33; Transducer 8

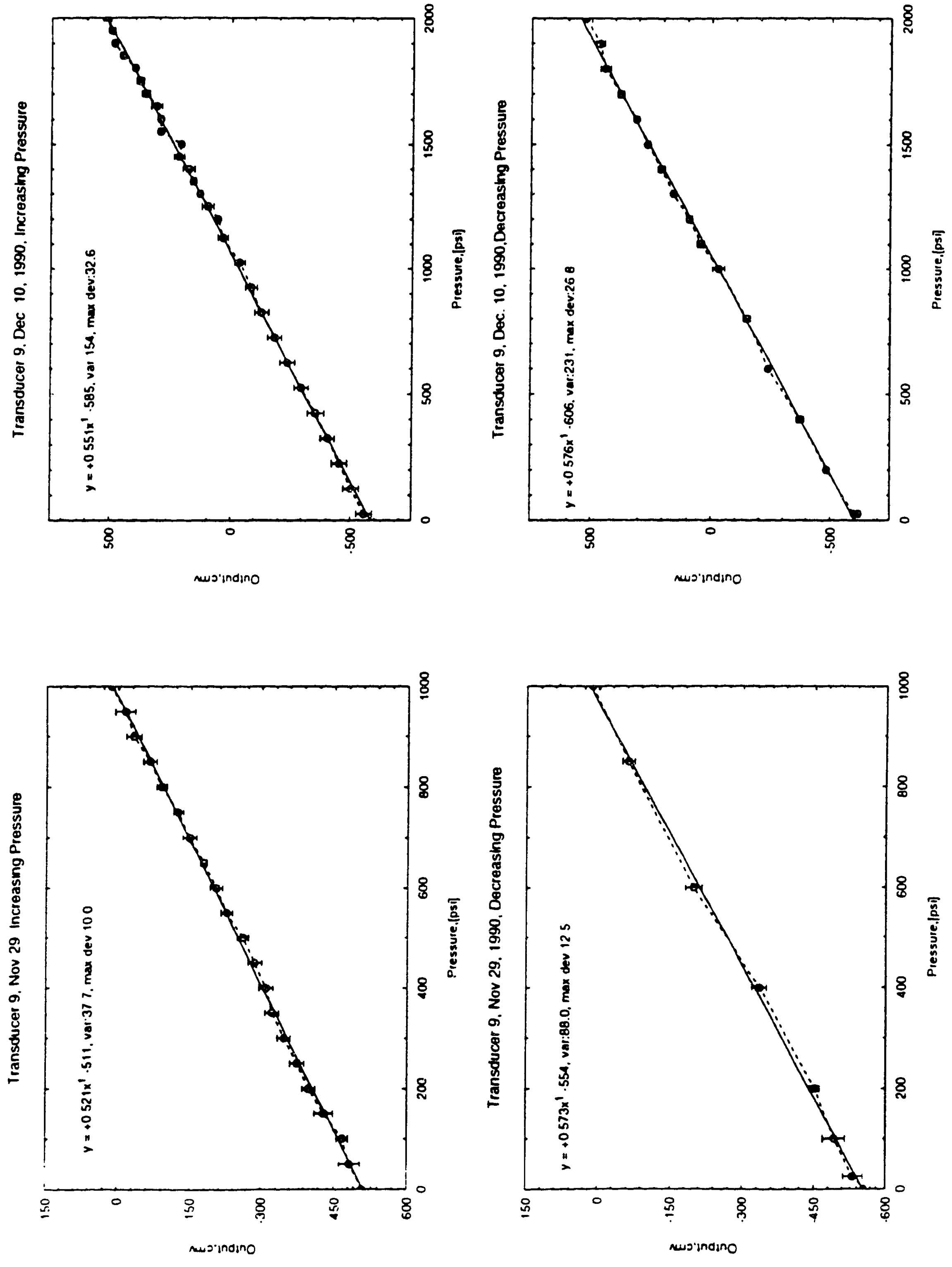


Figure B-12: Pressure Calibration of Prosthesis 33; Transducer 9

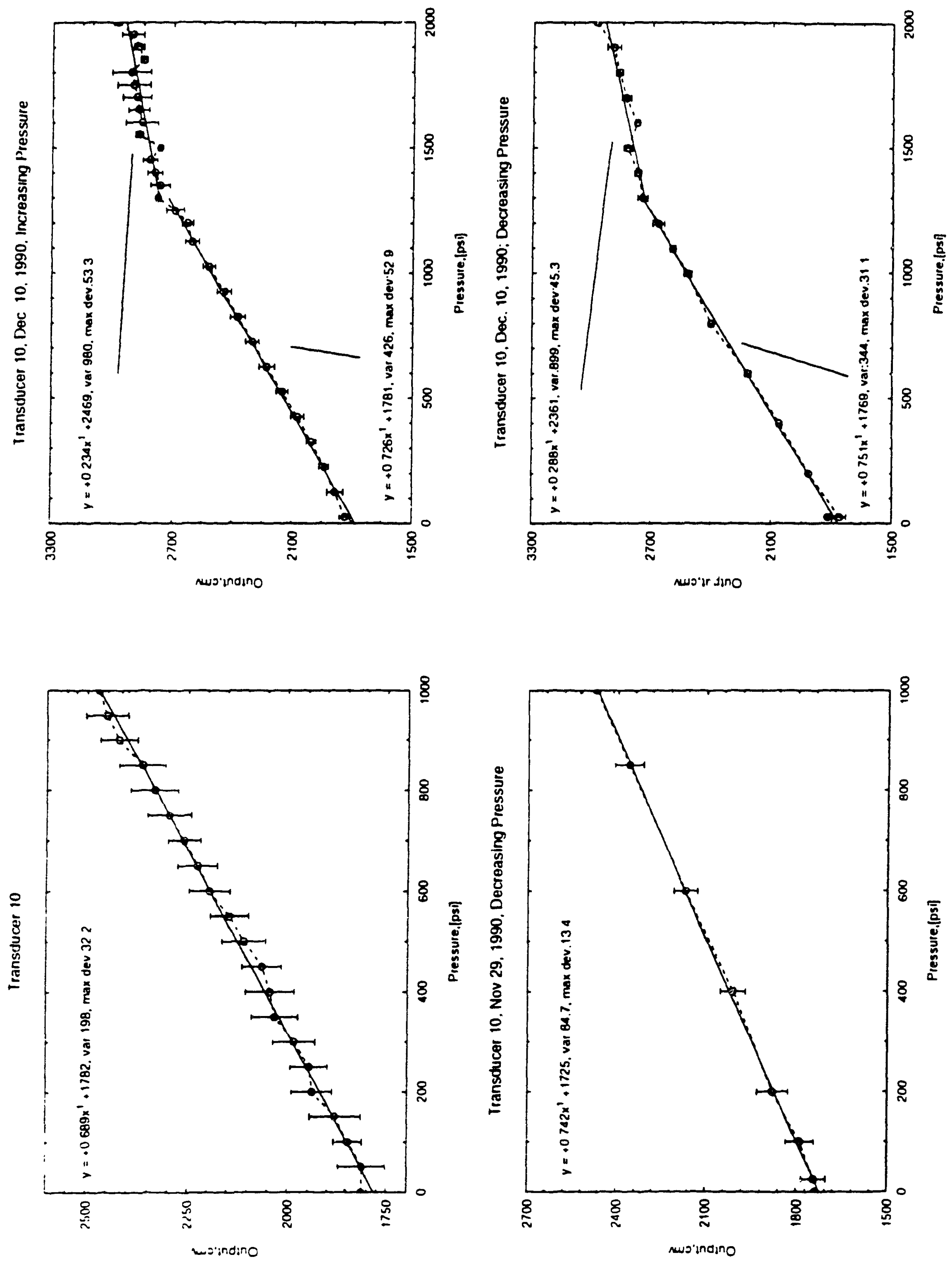


Figure B-13: Pressure Calibration of Prosthesis 33; Transducer 10

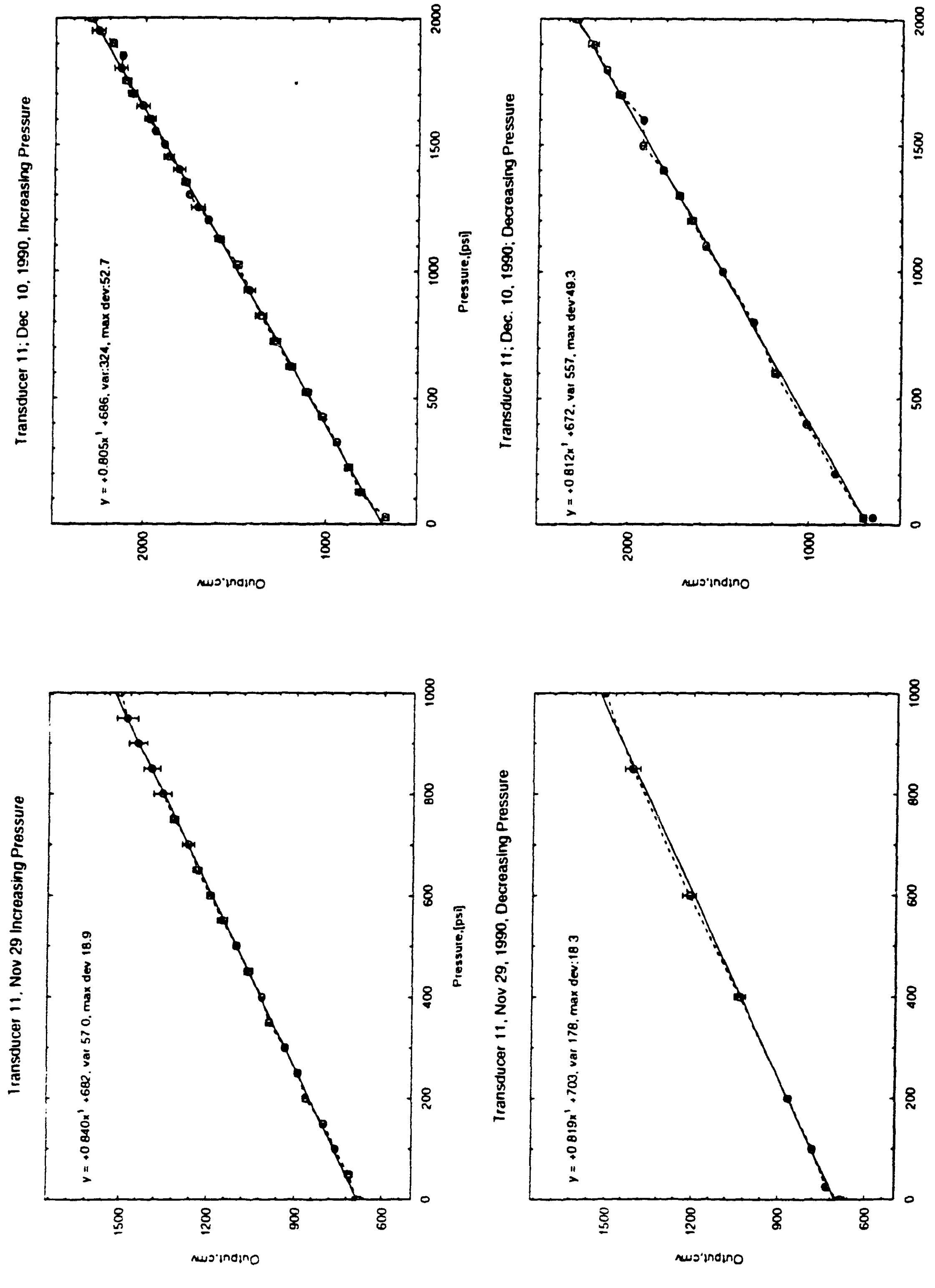


Figure B-14: Pressure Calibration of Prosthesis 33; Transducer 11

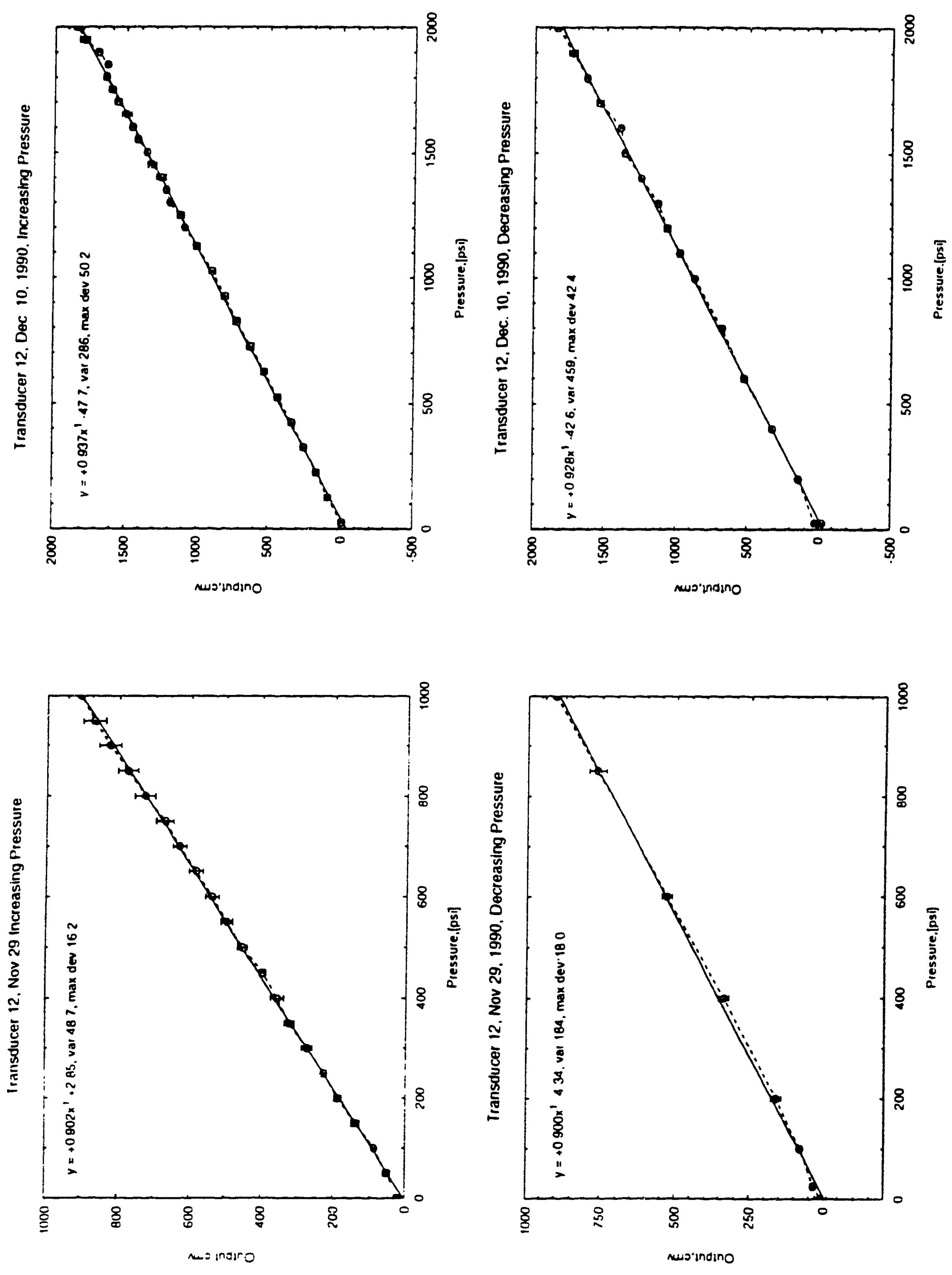


Figure B-15: Pressure Calibration of Prosthesis 33; Transducer 12

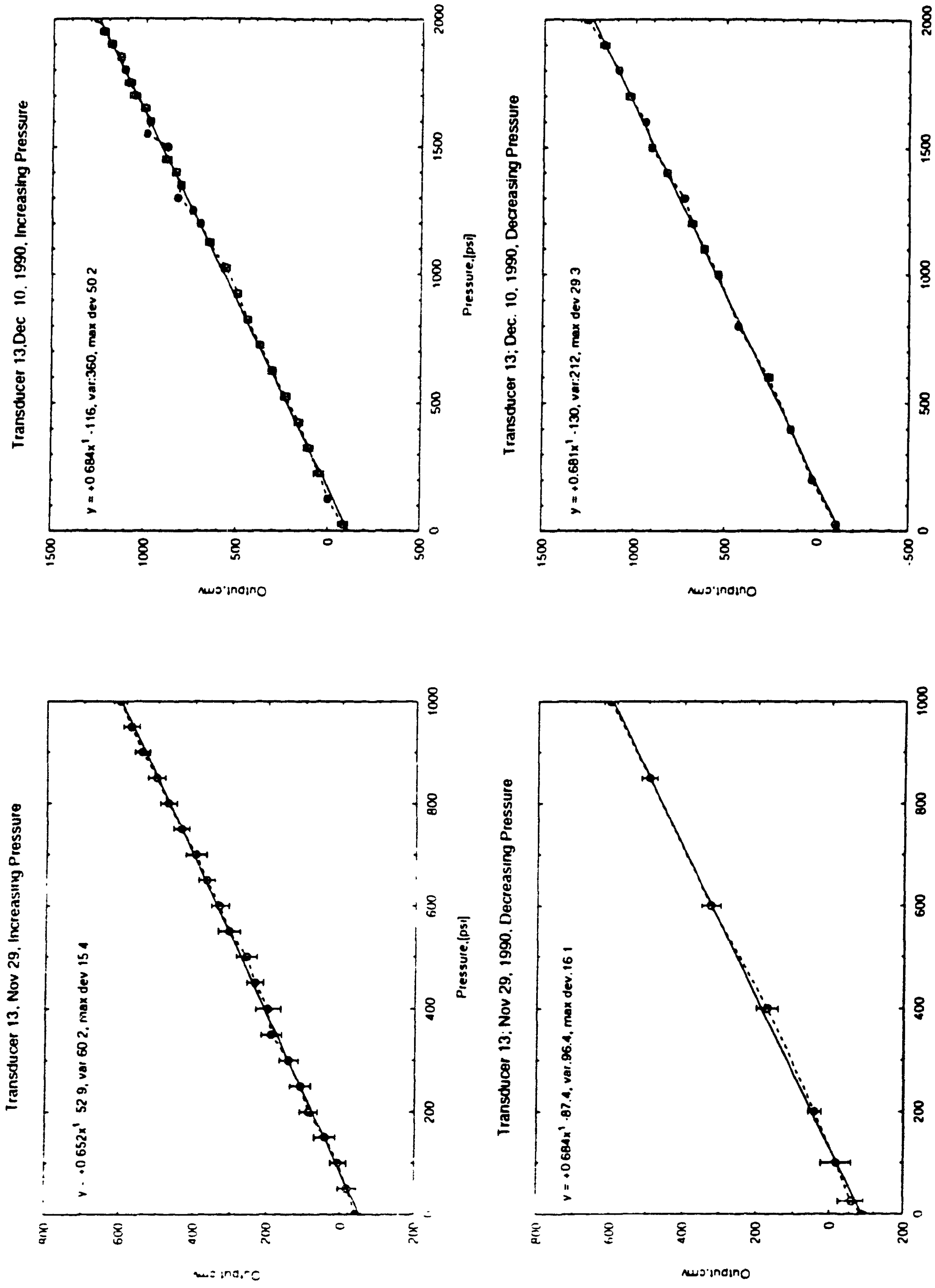


Figure B-16: Pressure Calibration of Prosthesis 33; Transducer 13

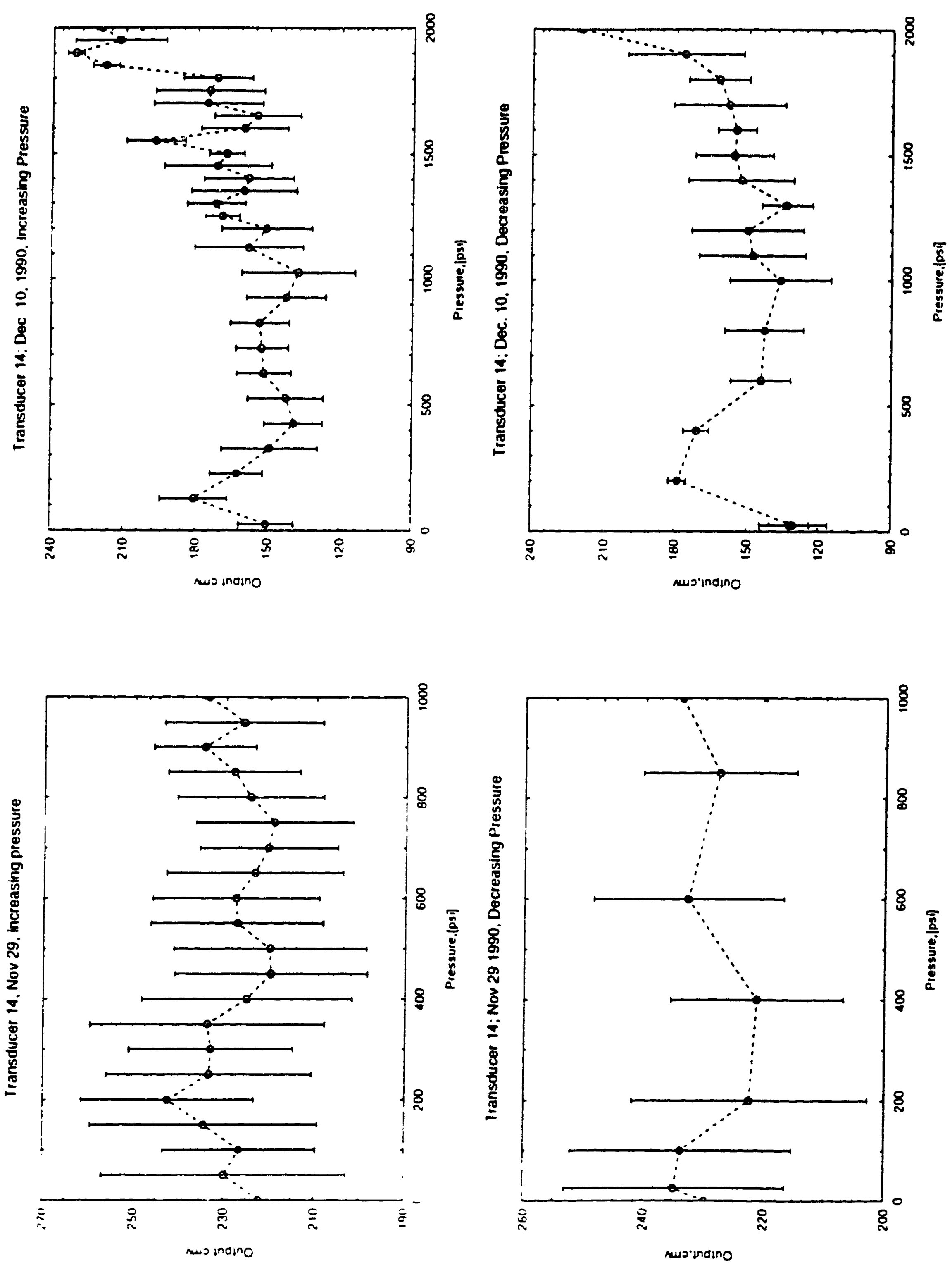


Figure B-17: Pressure Calibration of Prosthesis 33; Transducer 14

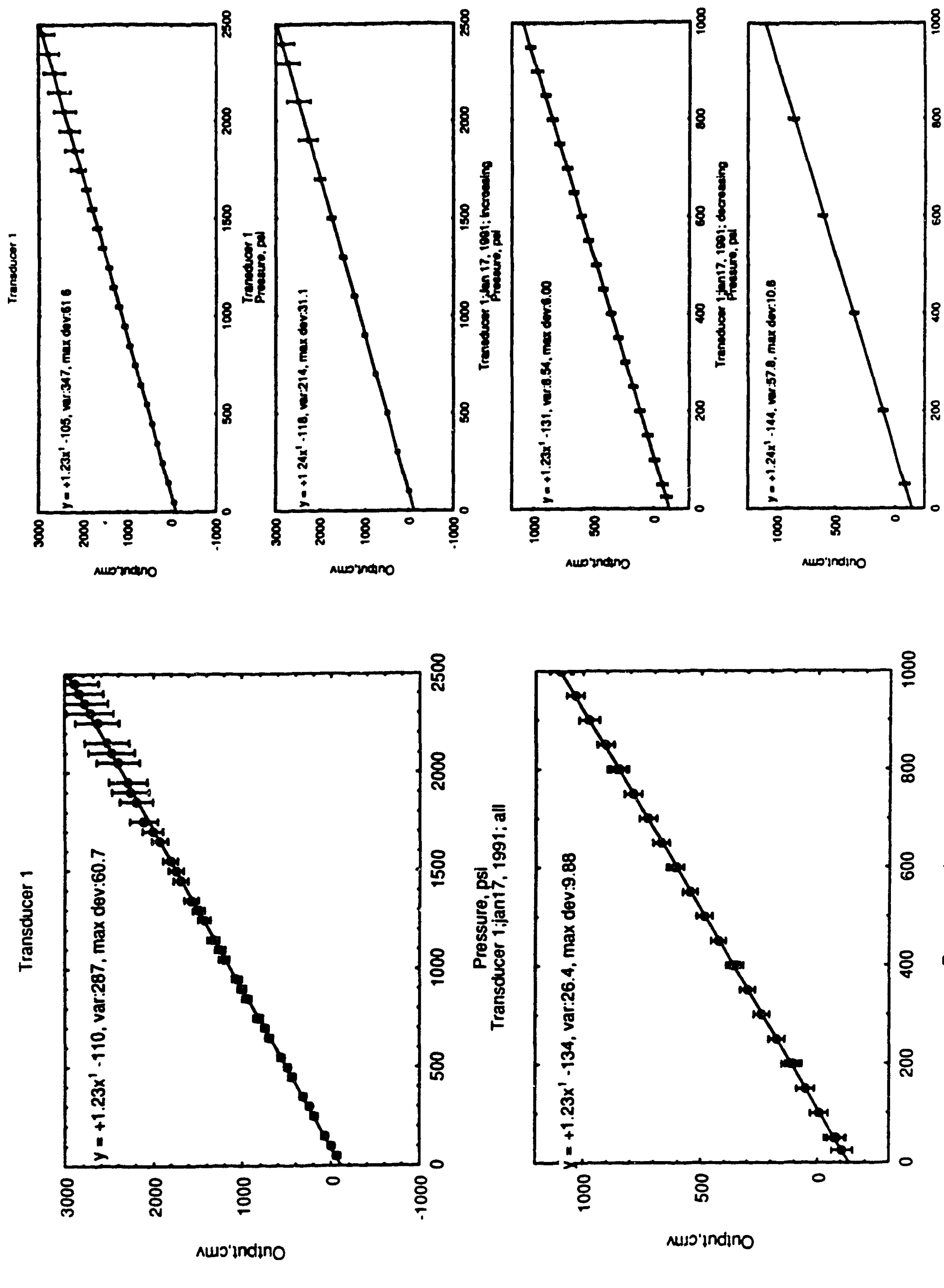
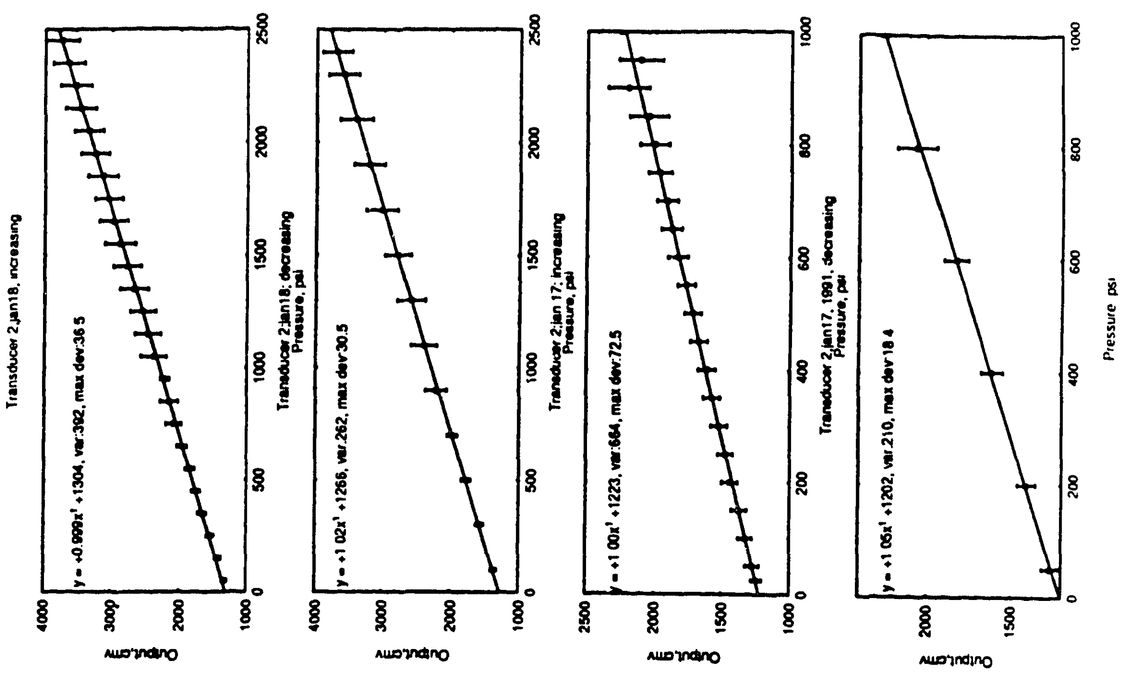


Figure B-18: Pressure Calibration of Prosthesis 43; Transducer 1



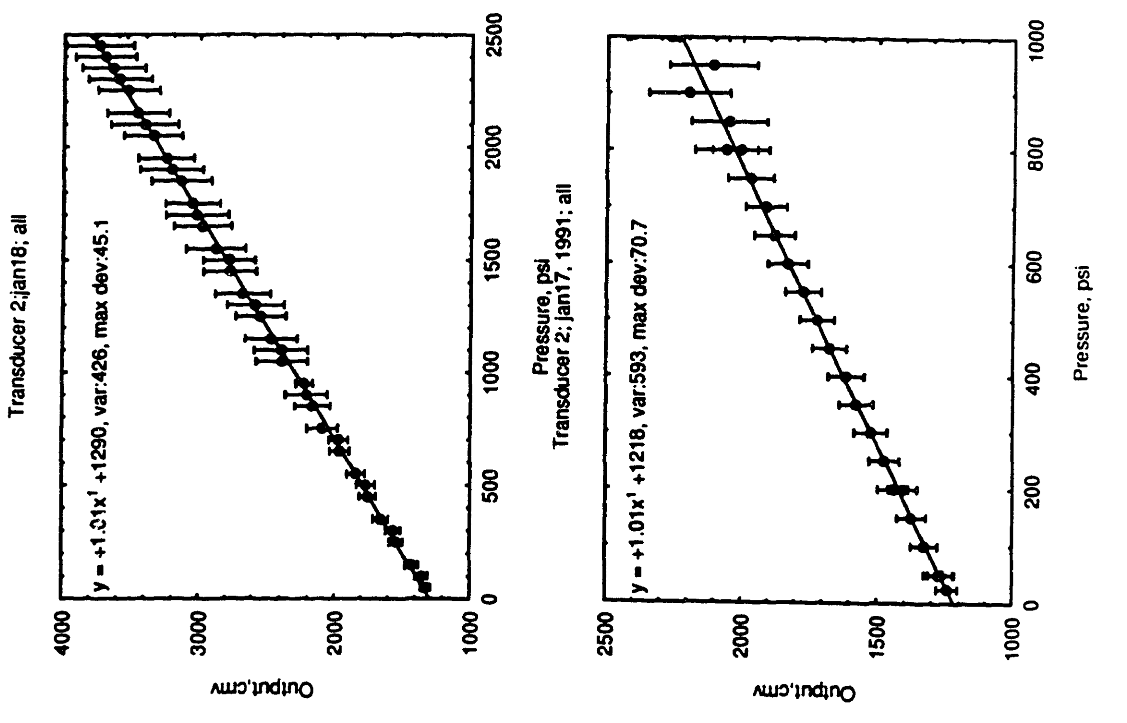


Figure B-19: Pressure Calibration of Prosthesis 43; Transducer 2

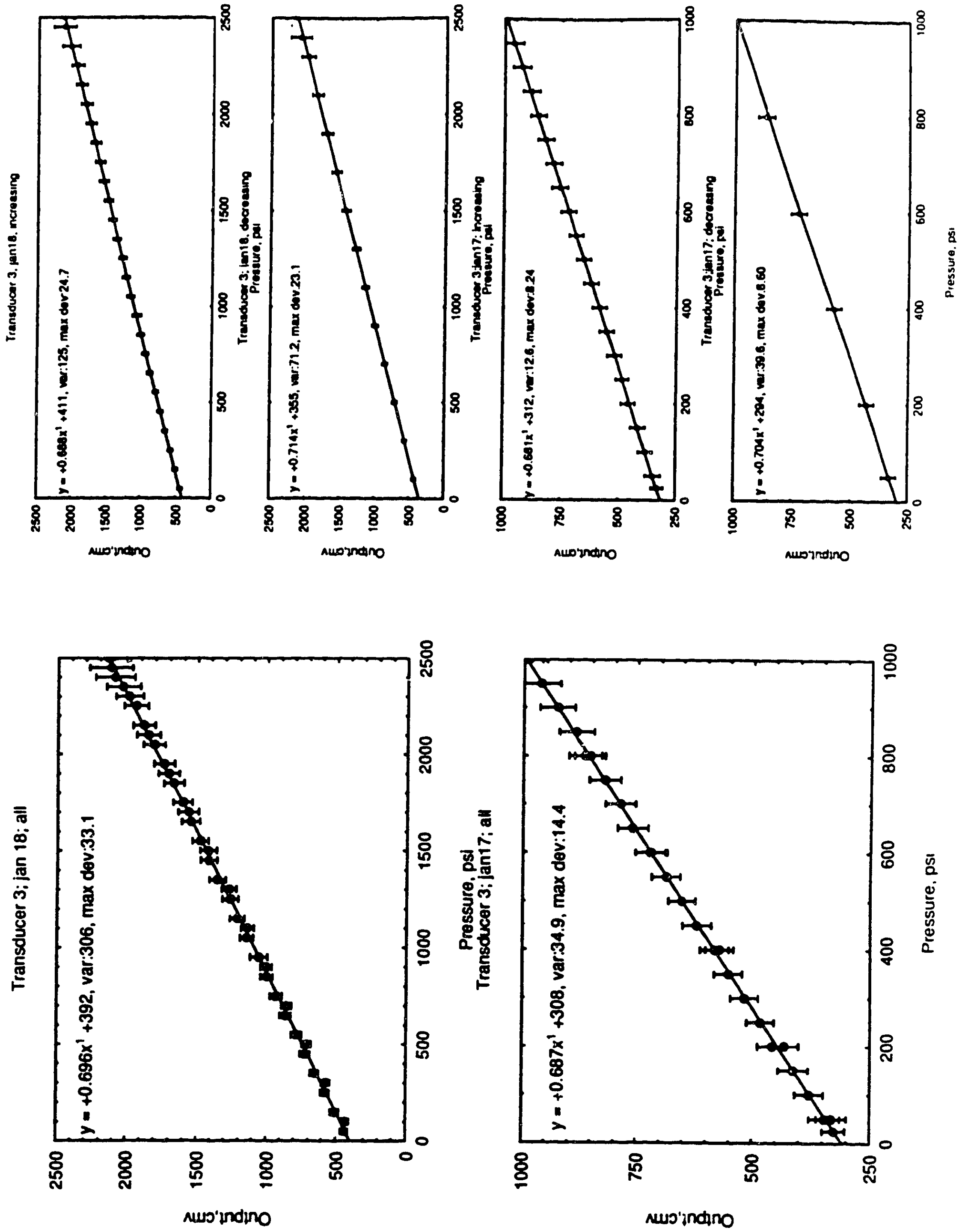


Figure B-20: Pressure Calibration of Prosthesis 43; Transducer 3

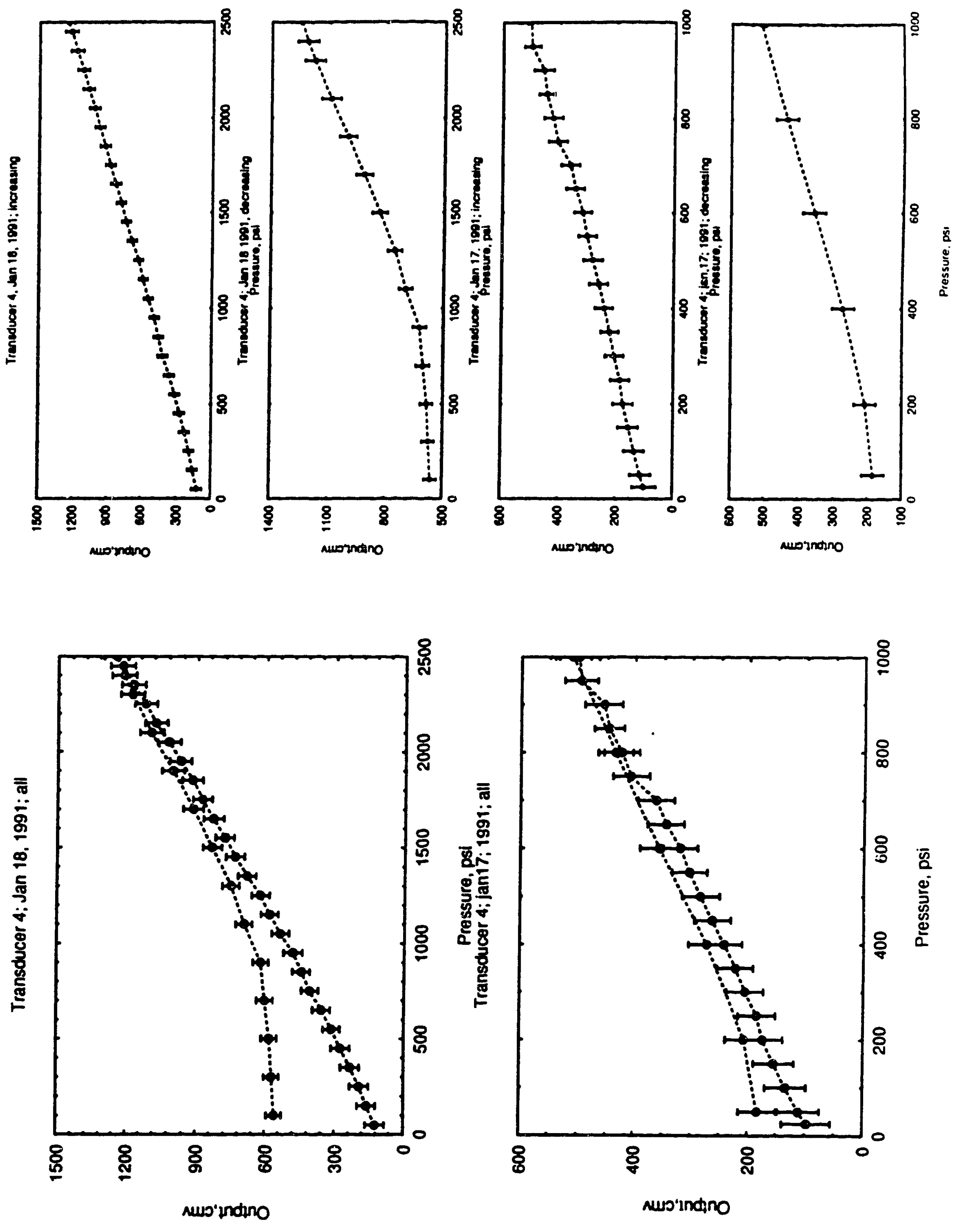


Figure B-21: Pressure Calibration of Prosthesis 43; Transducer 4

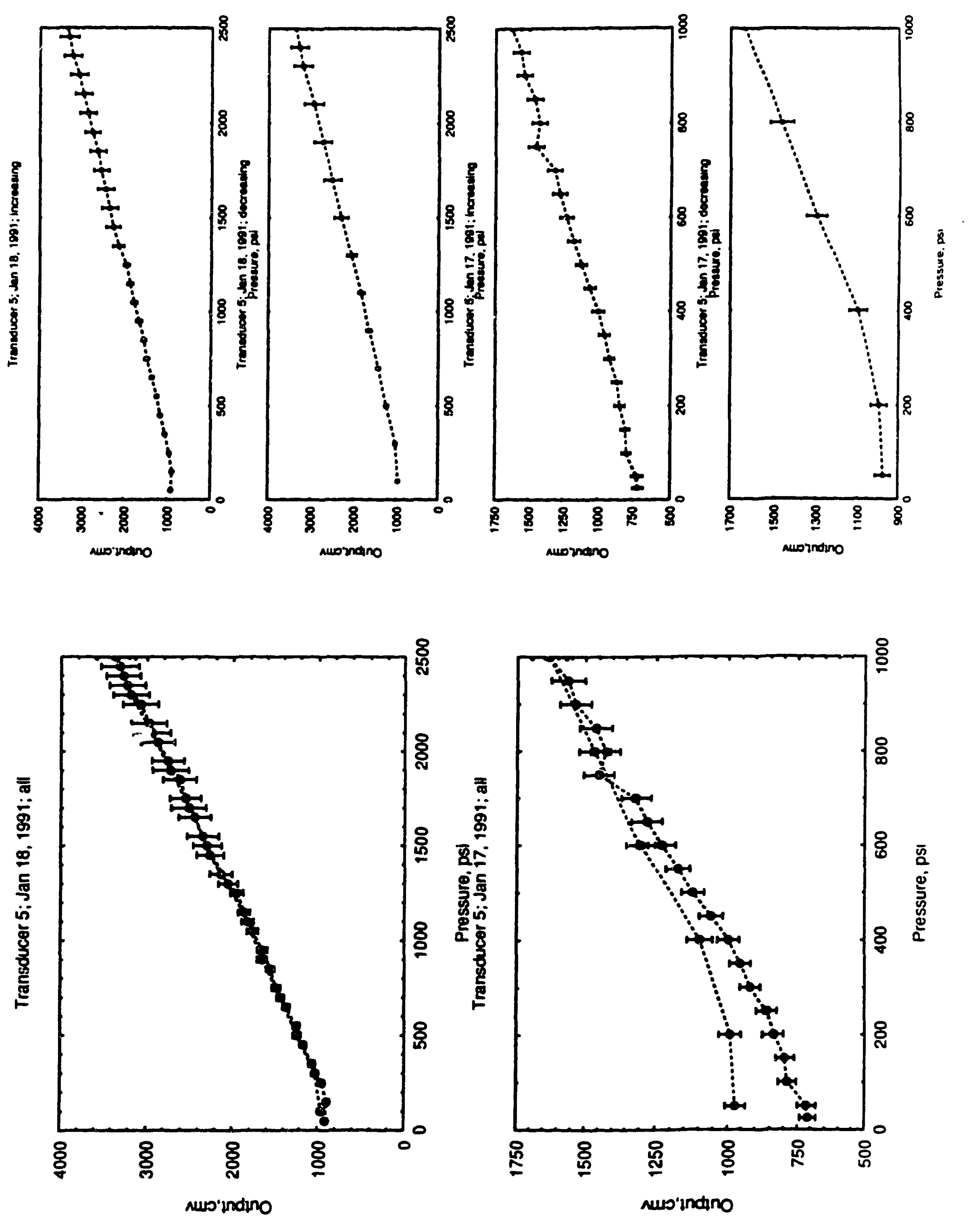


Figure B-22: Pressure Calibration of Prosthesis 43; Transducer 5

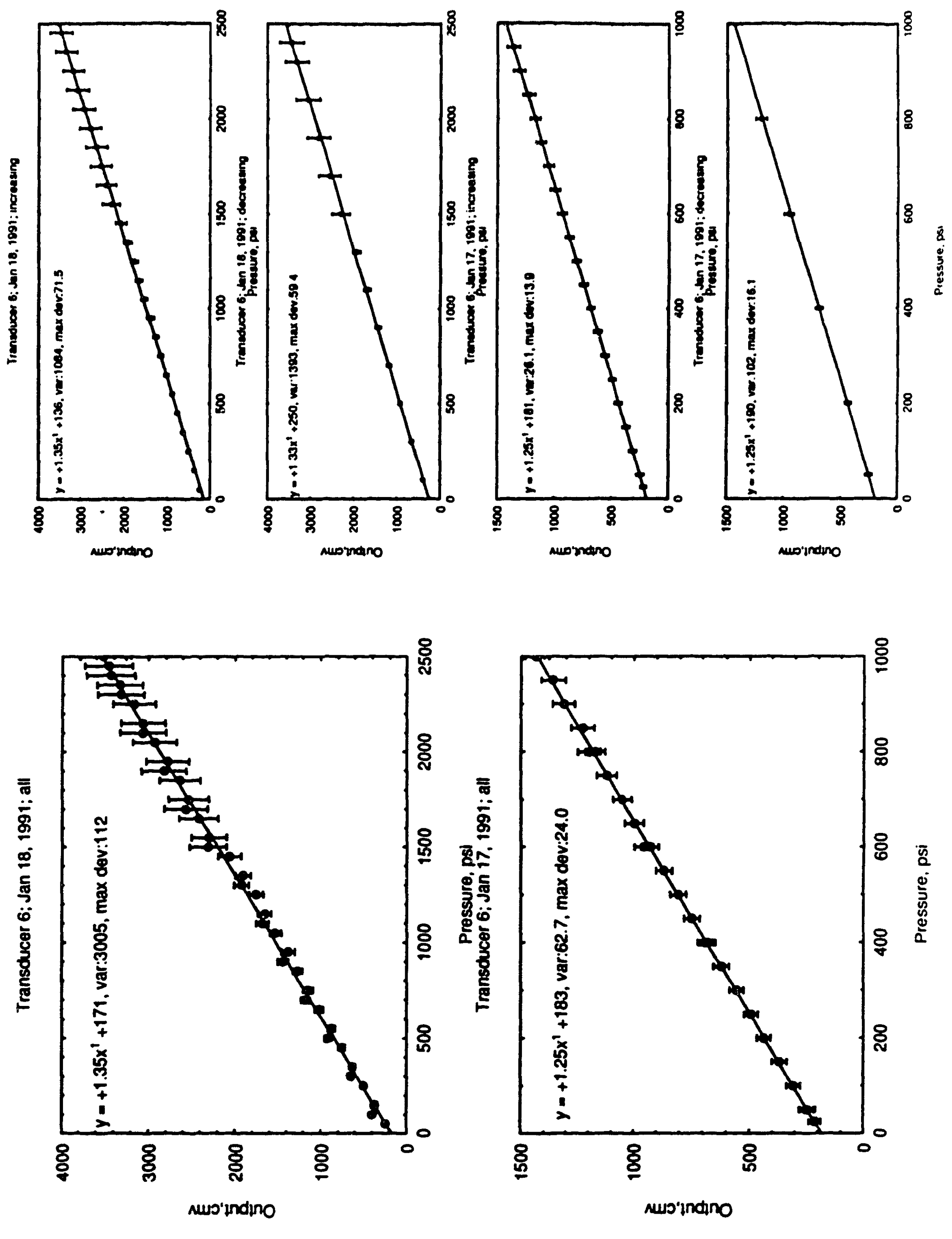


Figure B-23: Pressure Calibration of Prosthesis 43; Transducer 6

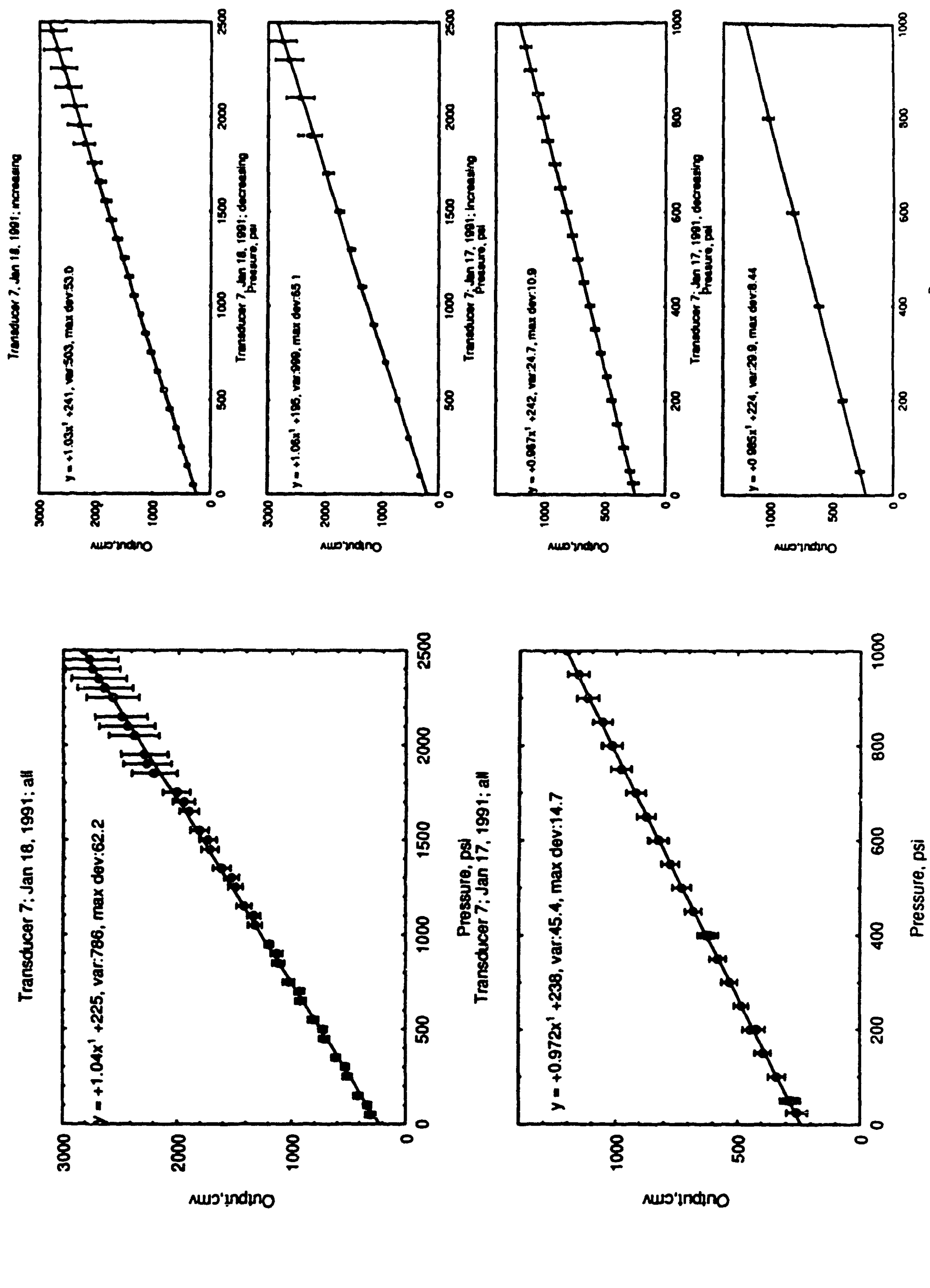


Figure B-24: Pressure Calibration of Prosthesis 43; Transducer 7

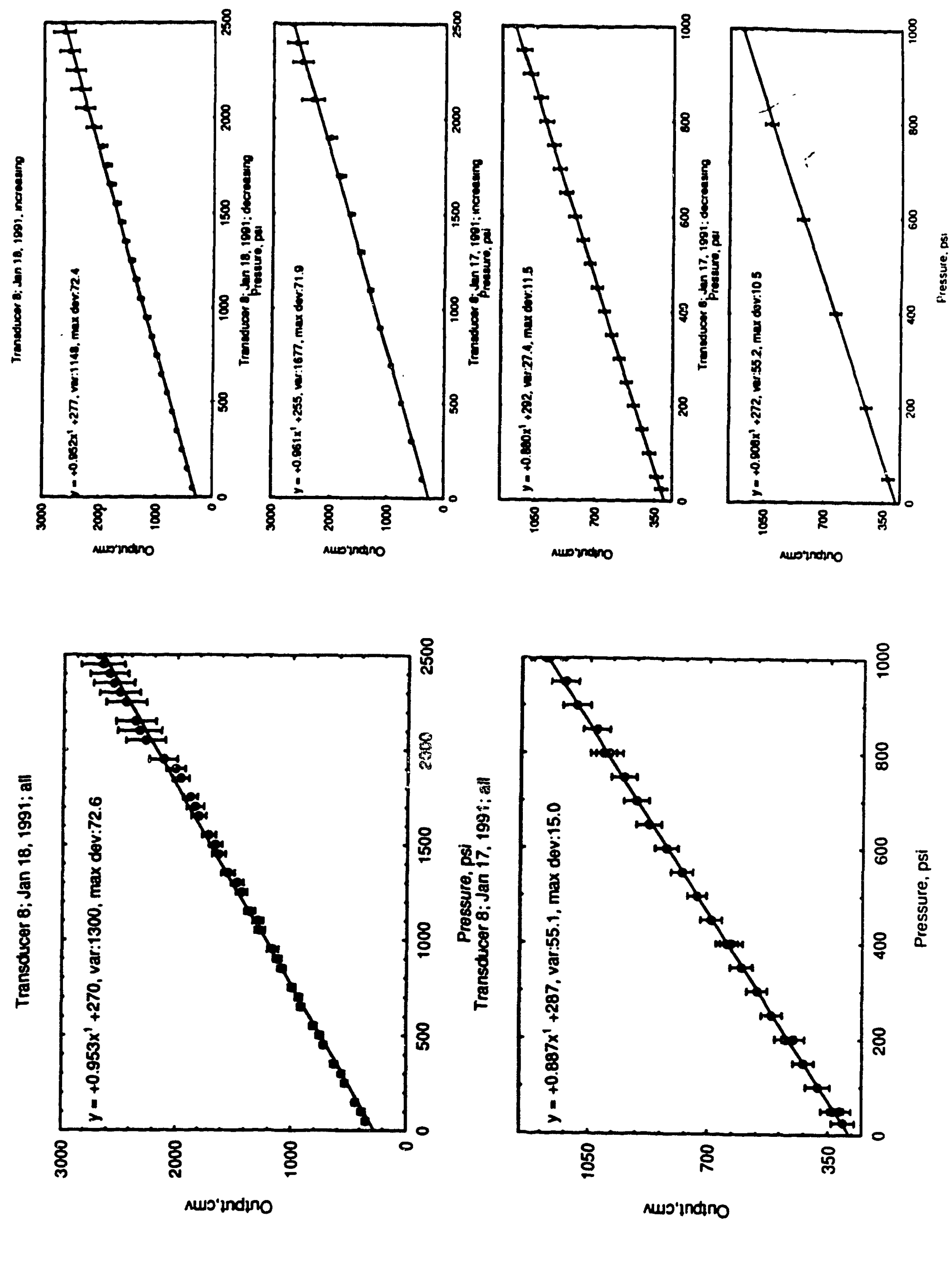


Figure B-25: Pressure Calibration of Prosthesis 43; Transducer 8

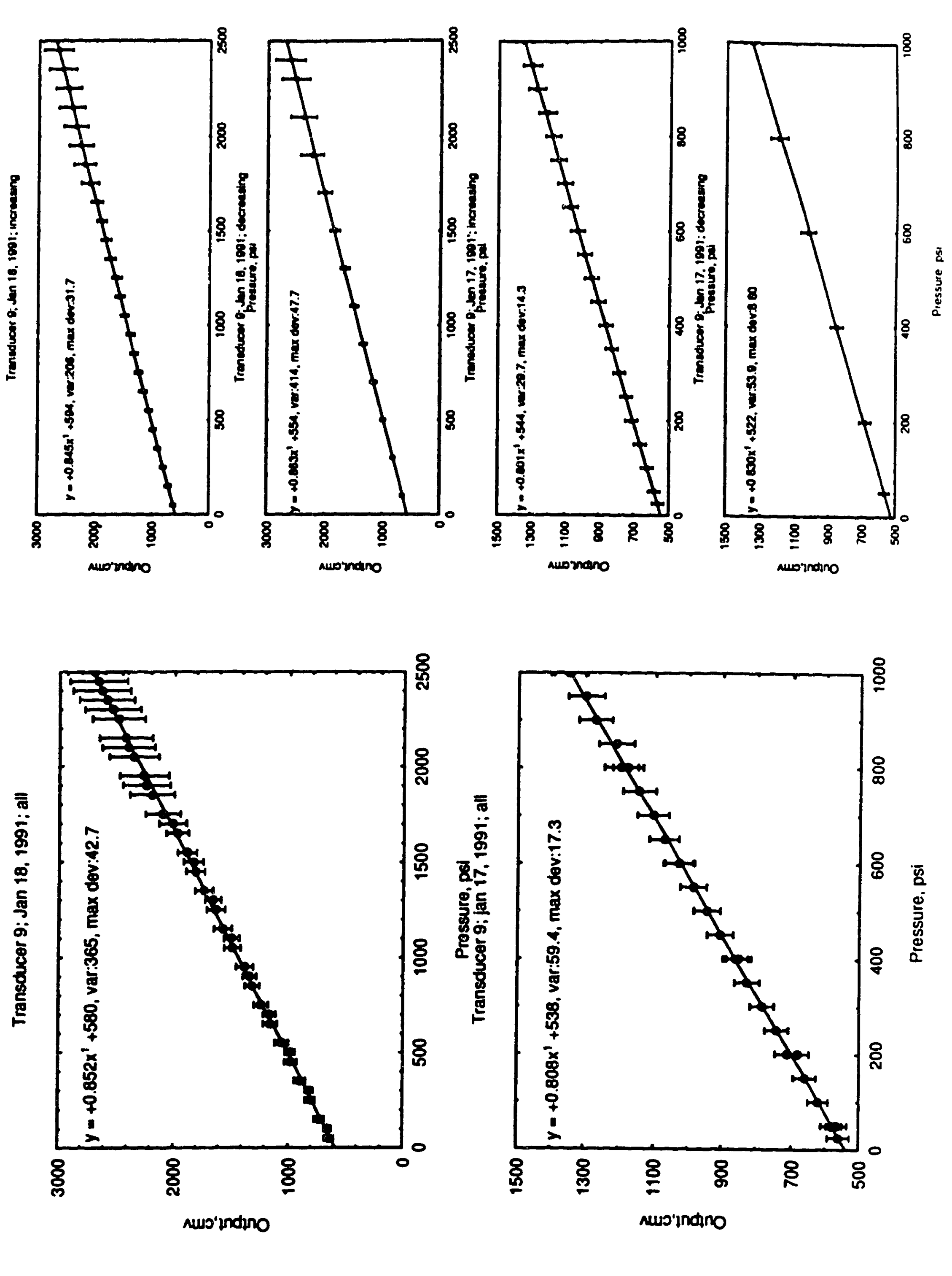


Figure B-26: Pressure Calibration of Prosthesis 43; Transducer 9

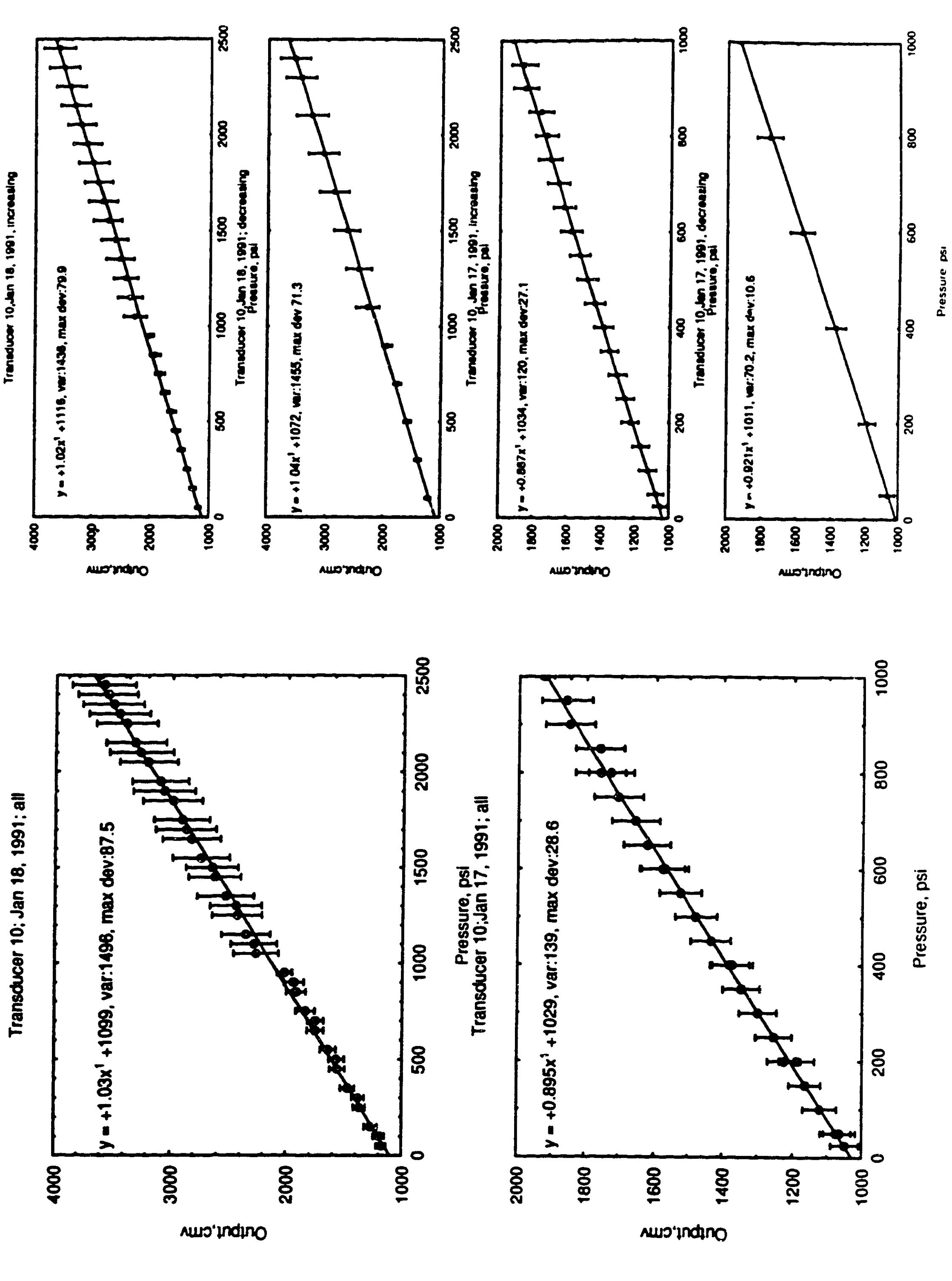


Figure B-27: Pressure Calibration of Prosthesis 43; Transducer 10

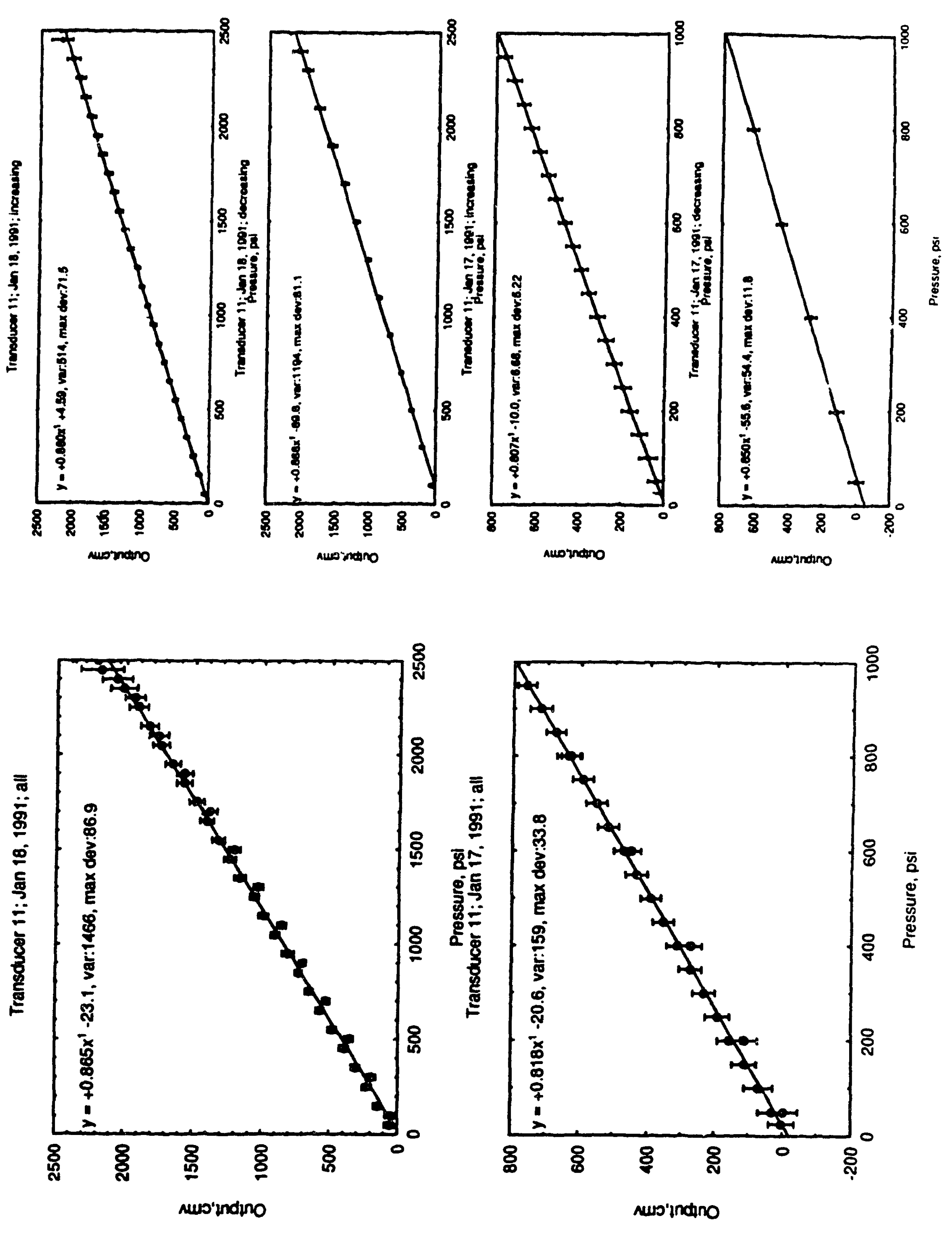


Figure B-28: Pressure Calibration of Prosthesis 43; Transducer 11

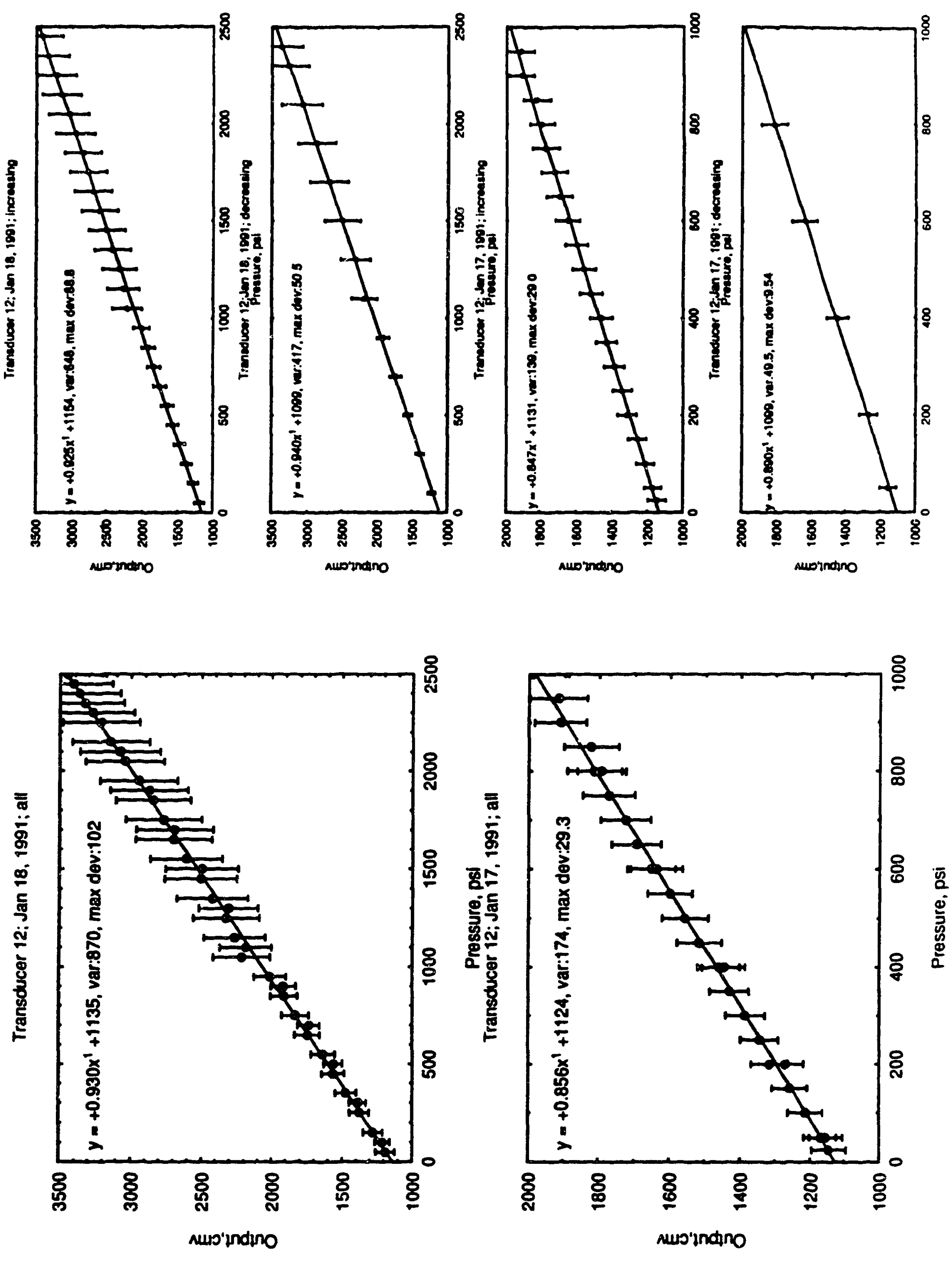


Figure B-29: Pressure Calibration of Prosthesis 43; Transducer 12

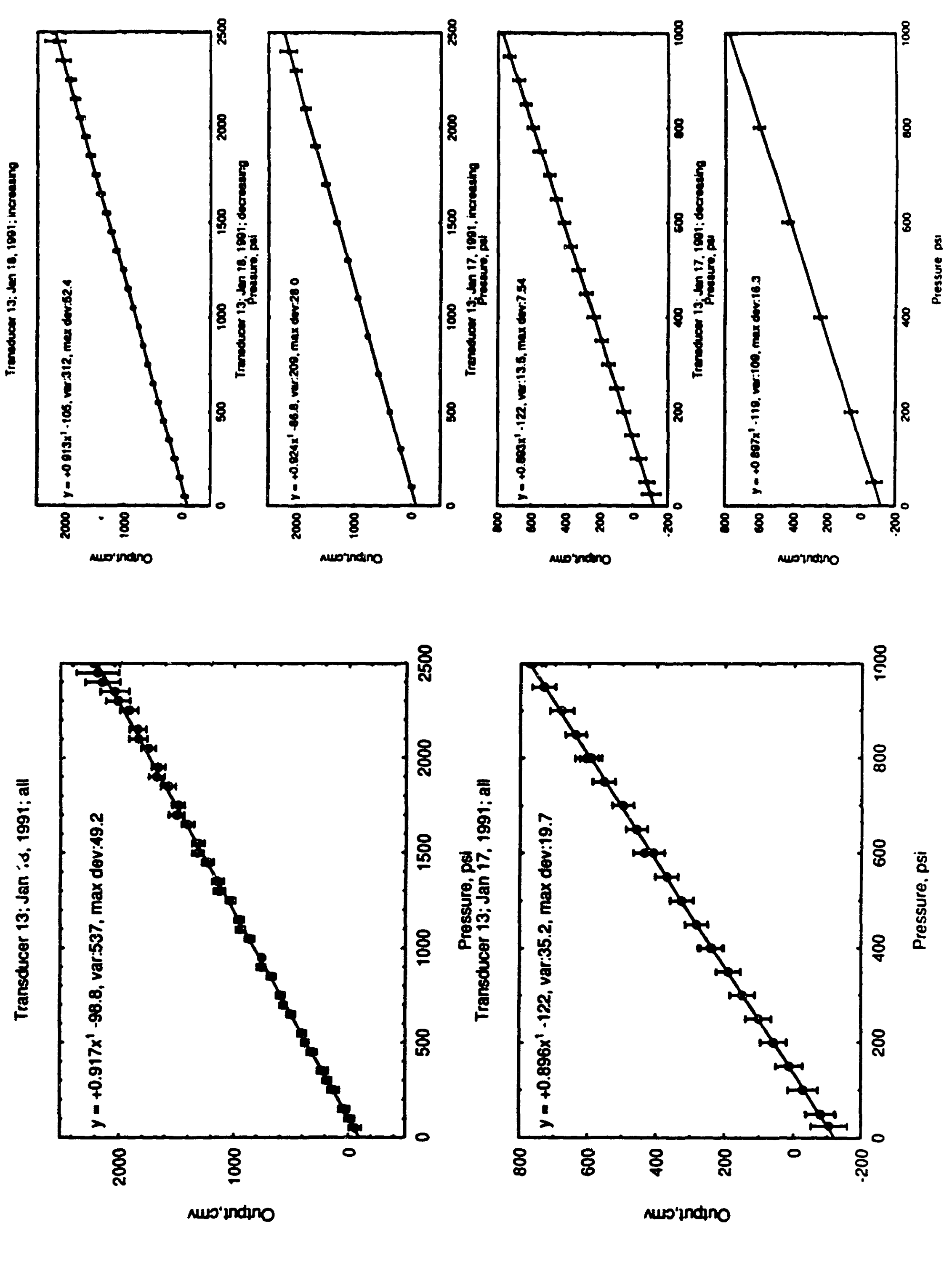


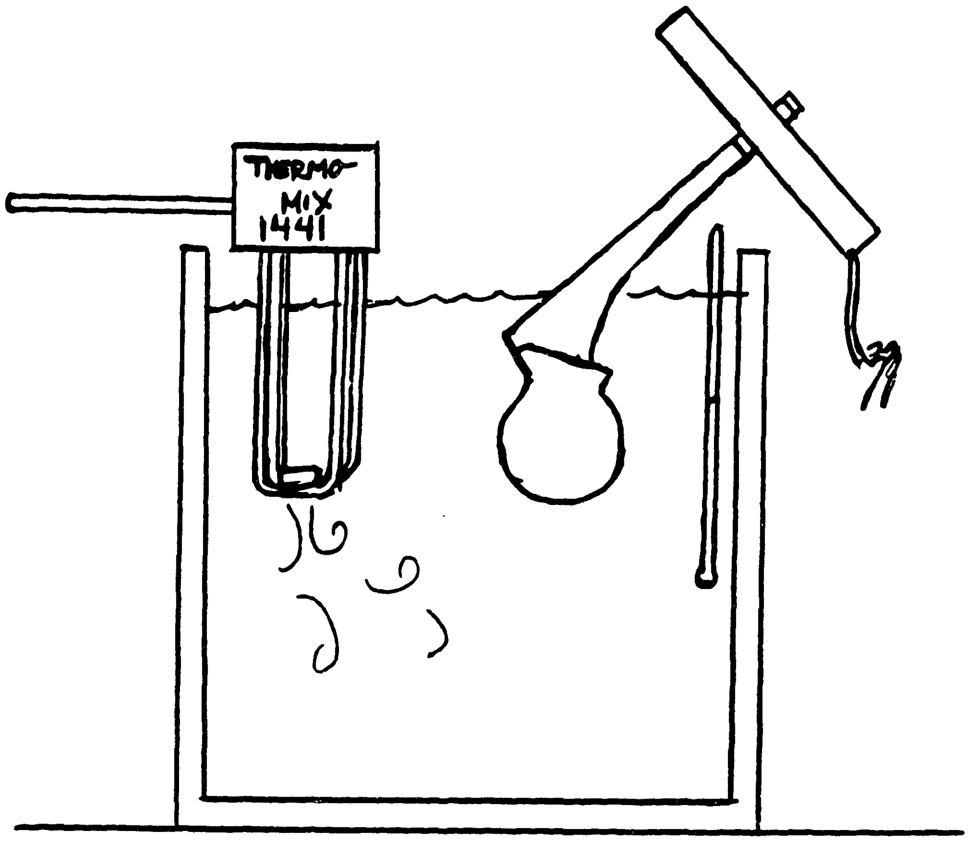
Figure B-30: Pressure Calibration of Prosthesis 43; Transducer 13

Appendix C

Temperature Calibration of Pressure Instrumented Prostheses

Calibration tests prior to the implantation of prosthesis 33 indicated that transducer output was dependant on temperature, via a dependance of the transducer offsets on temperature. Transducer gains were found to be independant of temperature. Accordingly a series of tests to calibrate the prosthesis thermally were done. These tests were carried out on both prosthesis 33 and prosthesis 43. Prosthesis 43 additionally contains a thermistor internal to the prosthesis but in a well similar to the pressure transducer wells. Thermal calibration tests also were done for the calibration of this thermistor.

Thermal calibration tests were carried out in a fluid filled bath, with the water heated and circulated by a Thermomix 1441. A diagram of the equipment used is shown in figure C-1. The fluid bath temperature was elevated and held at a value (independantly measured with an accuracy of 0.1 °C) for 20 minutes prior to data collection at each temperature. For most of the data the heater was turned off during the actual data collection; comparison with tests in which it was left on indicate that the heater status (off or on) had no effect on the data. Data was also taken while reducing the fluid bath temperature, this was done by exchanging some of the heated water for cooler water, which was difficult to control and as a result there is less information on transducer behavior during temperature reduction.



THERMAL CALIBRATION SET UP

Figure C-1: Equipment for thermal calibration of Prosthesis 43

Thermal calibration in the fluid bath was done on three days for prosthesis 33. On the first day data was only taken for increasing temperature, from 29.2 °C to 41.1 °C (84.5 °F to 106 °F). The second day on which data was taken the temperature was raised from 24.2 °C (75.6 °F) to 45.5 °C (114 °F) and lowered to 26 °C (78.8 °F). The third set of tests was done between 25.8 °C (78.4 °F) and 40 °C (104 °F). When implanted in the human body the temperature is fairly constant, it was expected to be 37 °C (98.6 °F), and has been measured in the second subject at 38 °C (100.4 °F). Figures C-2 to C-11 show the results for working transducers from all thermal calibration tests and the best fit line and quadratic through each set of means. A linear estimate was used. Table C.1 presents the slope of the line fit to each set of data and to composite sets of data, also the gain found prior to implantation. Table C.2 presents the difference in pressure estimates that would result from using *ex vivo* calibration data. These differences are independant of the pressure measured, as the temperature only affects the zero-pressure output from the pressure transducers.

Calibration of prosthesis 43 with respect ot temperature was done using the same equipment and procedure used in re-calibration of prosthesis 33. Fluid bath tests were done on two days; on the first day the temperature was raised from 72 °F to 107 °F. On the second day tests were done the temperature was raised from 73 °F to 106.7 °F and then decreased to 75 °F. Figure C-12 shows the results from both days of testing for the thermistor in prosthesis 43. These tests indicated that the thermistor output was well represented by equation C.1,

$$y = 0.3391\,F^2 - 2861. \tag{C.1}$$

which is also shown on figure C-12, in which y is the thermistor output in corrected millivolts and F is degrees Fahrenheit. Figures C-13 through C-25 show the results of the temperature calibration tests for the pressure transducers in prosthesis 43. Table C.3 lists the relationship between transducer offset and temperature that was found from these tests. Equation 4.2 indicates the use of these factors. Another set of tests was done in the fluid bath with prosthesis 43 determine how rapidly the

Transducer	**1**	**2**	**3**	**5**	**7**
Feb. 20, up	-45.4	-4.53	-24.2	6.18	13.0
Feb. 22, up	-42.7	-4.57	-22.9	5.87	12.5
Feb. 22, down	-39.2	-7.3	-25.2	5.52	11.7
Feb. 22, all	-41.0	-5.54	-22.9	5.74	11.3
Mar. 1, circ.	-43.3	-6.47	-22.7	3.07	4.59
Mar. 1, still	-42.3	-4.32	-21.3	7.24	12.1
Mar. 1, all	-42.5	-5.25	-21.8	5.21	8.41
ex vivo	-41.8	-5.38	-22.5	5.23	11.2
pre-implant[a]	-2.67	-8.45	-20.15	2.52	0.48
Transducer	**9**	**10**	**11**	**12**	**13**
Feb. 20, up	-31.3	-103.	23.5	5.68	-40.2
Feb. 22, up	-34.8	-101.	22.6	3.17	-40.7
Feb. 22, down	-30.2	-107.	24.8	4.20	-38.5
Feb. 22, all	-34.1	-102.	23.7	3.42	-39.5
Mar. 1, circ	-42.4	-100.	18.8	-0.332	-43.0
Mar. 1, still	-39.8	-96.6	24.5	2.83	-40.5
Mar. 1, all	-40.8	-97.8	21.7	1.42	-41.5
ex vivo	-36.7	-100.0	22.9	2.68	-40.4
pre-implant[b]	-32.95	-96.2	16.13	3.925	-37.3

[a]Average of gain above and below 37 °C
[b]Average of gain above and below 37 °C

Table C.1: Gains of prosthesis 33 pressure transducers offsets with temperature

Transducer	$\Delta P/^\circ C$ [MPa/°C]	$\Delta P/^\circ C$ [psi/°C]
1	0.54	78.52
2	-0.027	-3.86
3	0.026	3.81
5	-0.029	-4.17
7	-0.118	-17.08
9	0.043	6.29
10	0.033	4.85
11	-0.049	-7.15
12	0.009	1.37
13	0.029	4.18

Table C.2: Pressure estimate difference for prosthesis 33 using ex-vivo temperature calibration

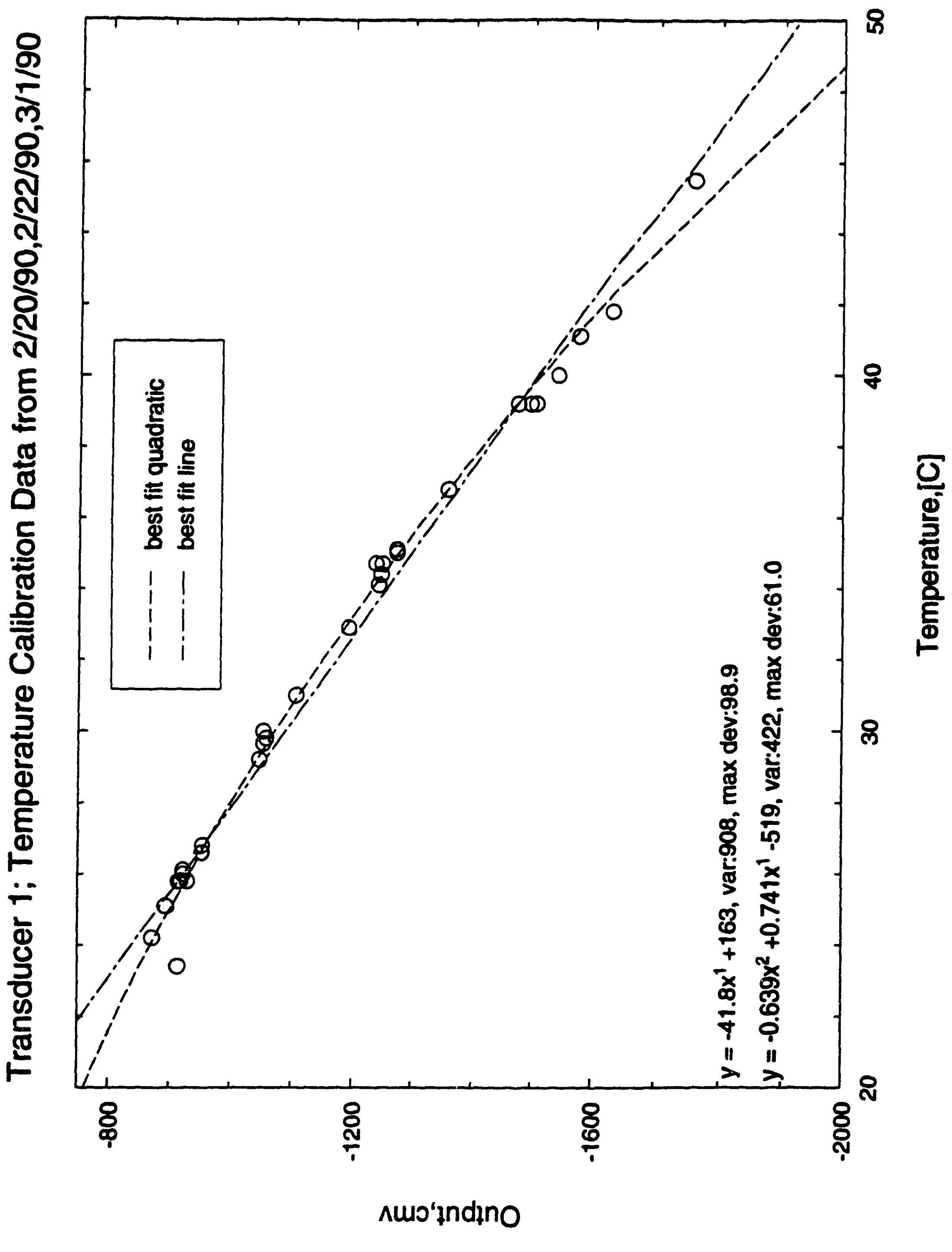


Figure C-2: Temperature Calibration of Prosthesis 33; Transducer 1

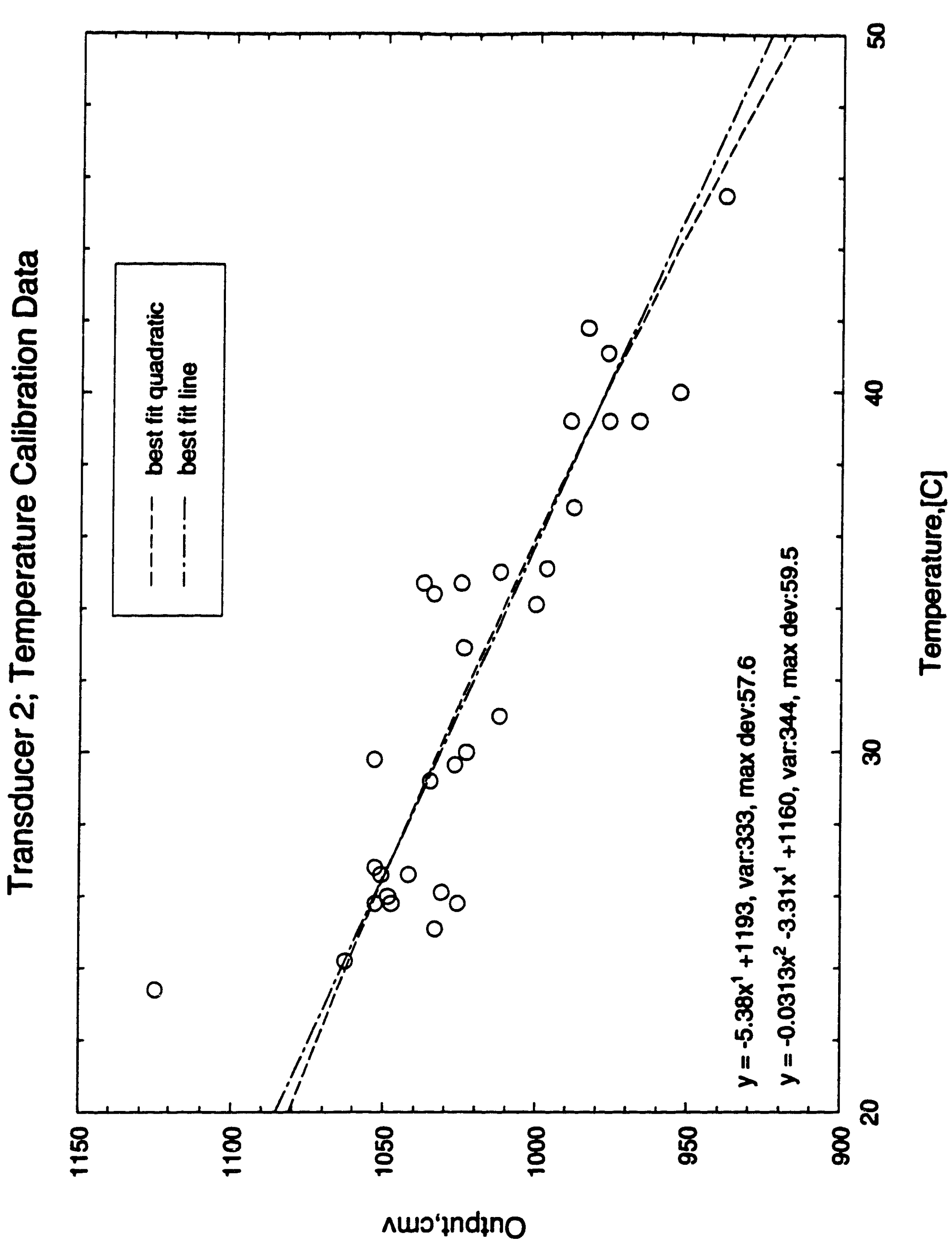


Figure C-3: Temperature Calibration of Prosthesis 33; Transducer 2

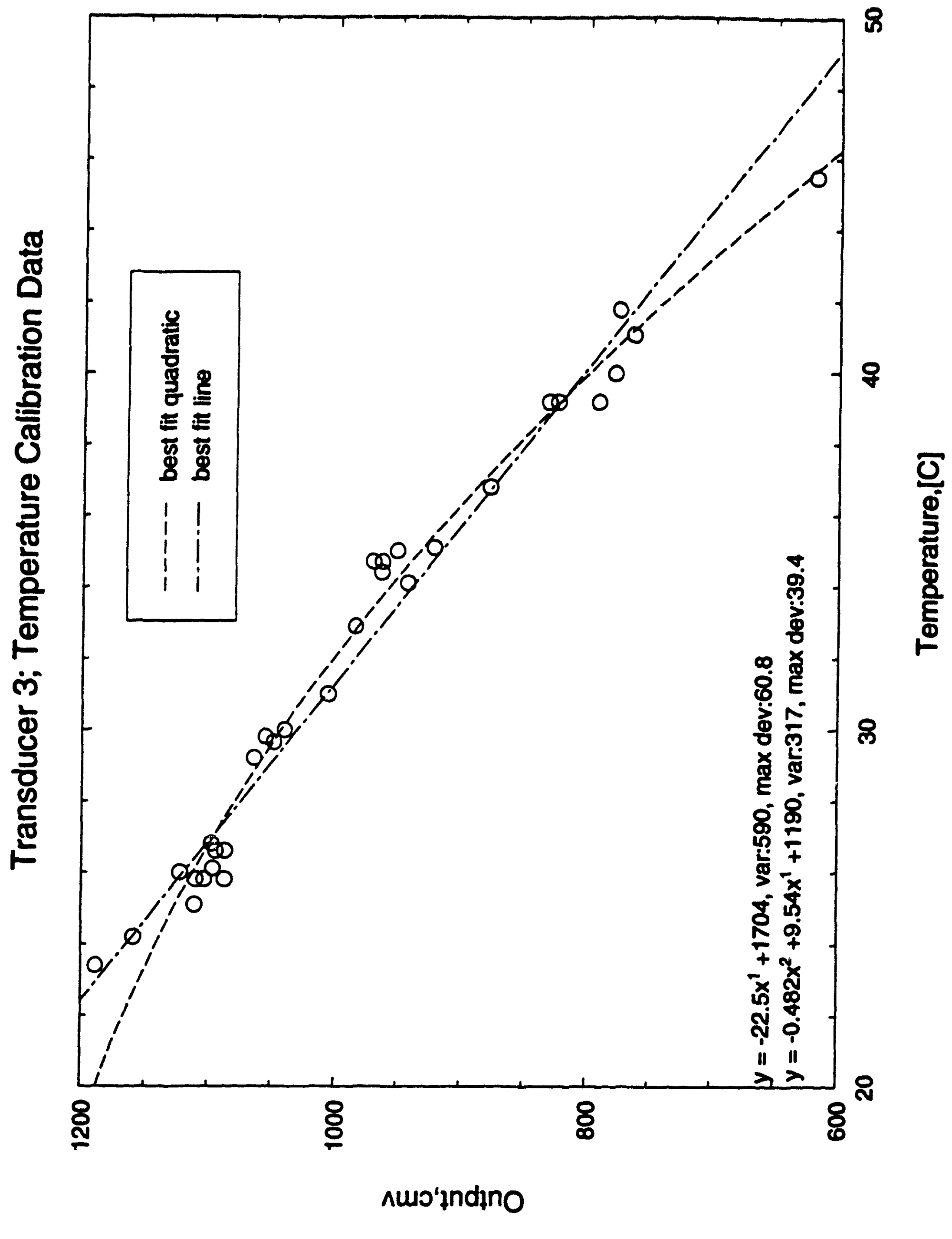


Figure C-4: Temperature Calibration of Prosthesis 33; Transducer 3

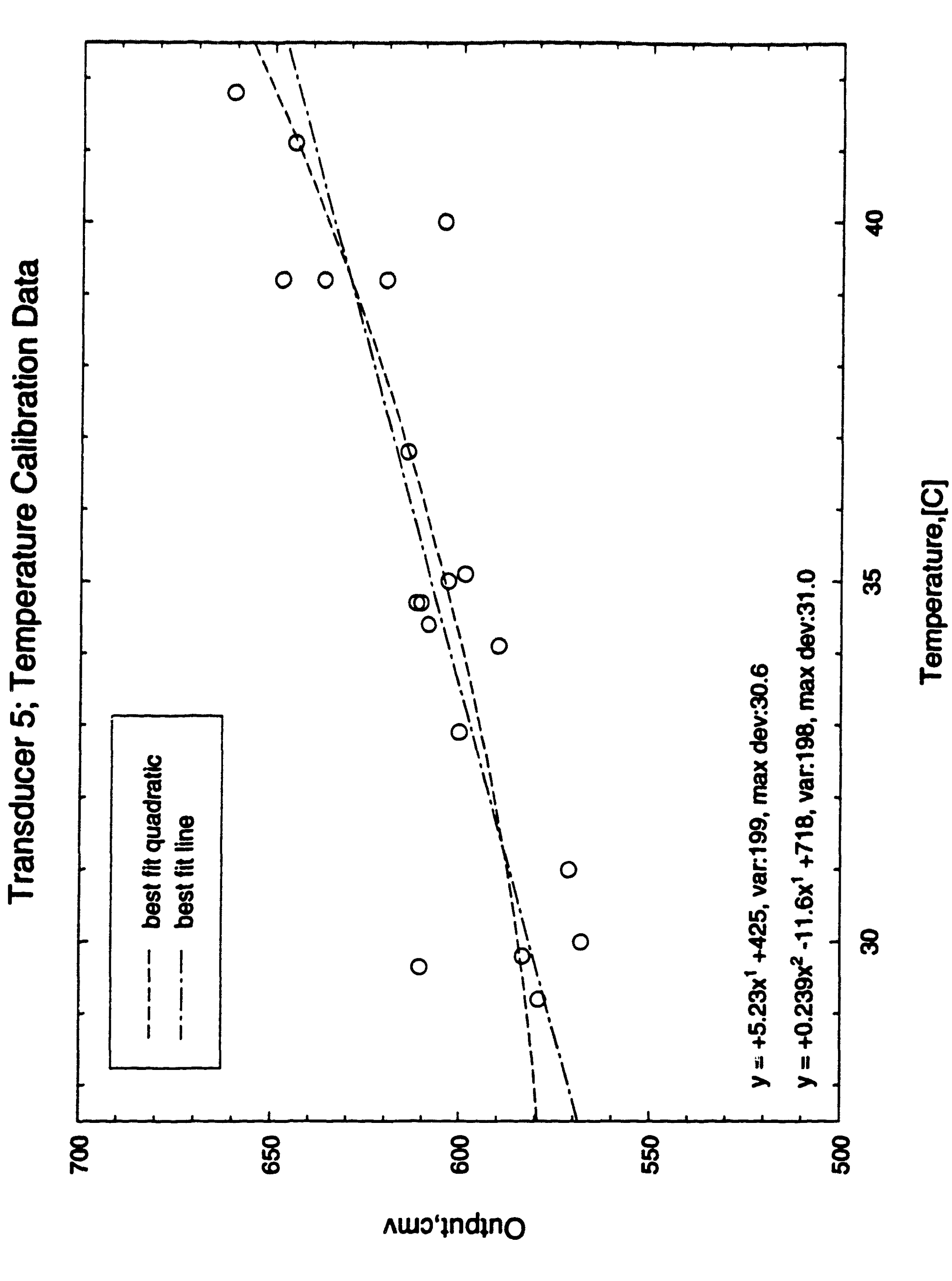


Figure C-5: Temperature Calibration of Prosthesis 33; Transducer 5

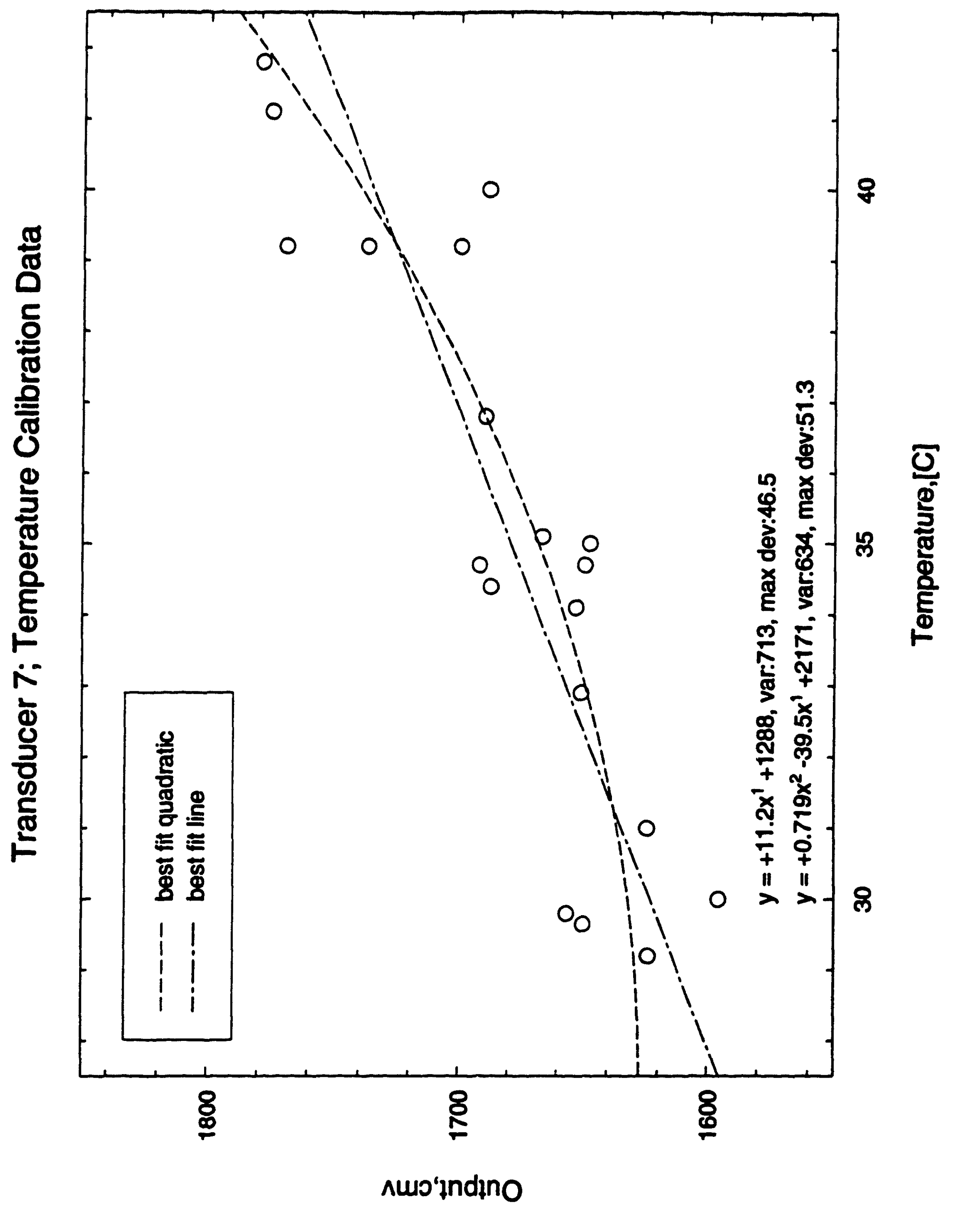


Figure C-6: Temperature Calibration of Prosthesis 33; Transducer 7

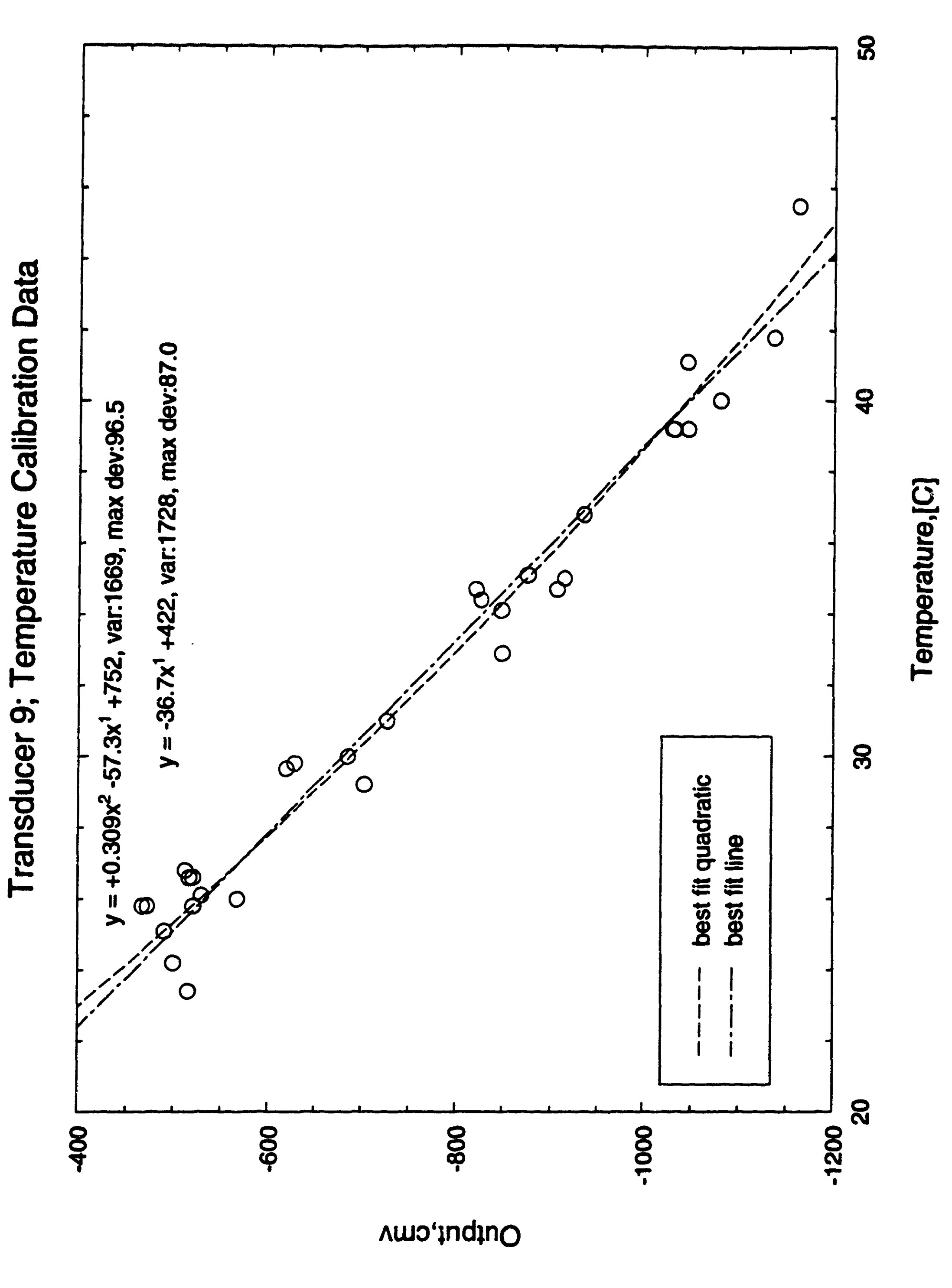


Figure C-7: Temperature Calibration of Prosthesis 33; Transducer 9

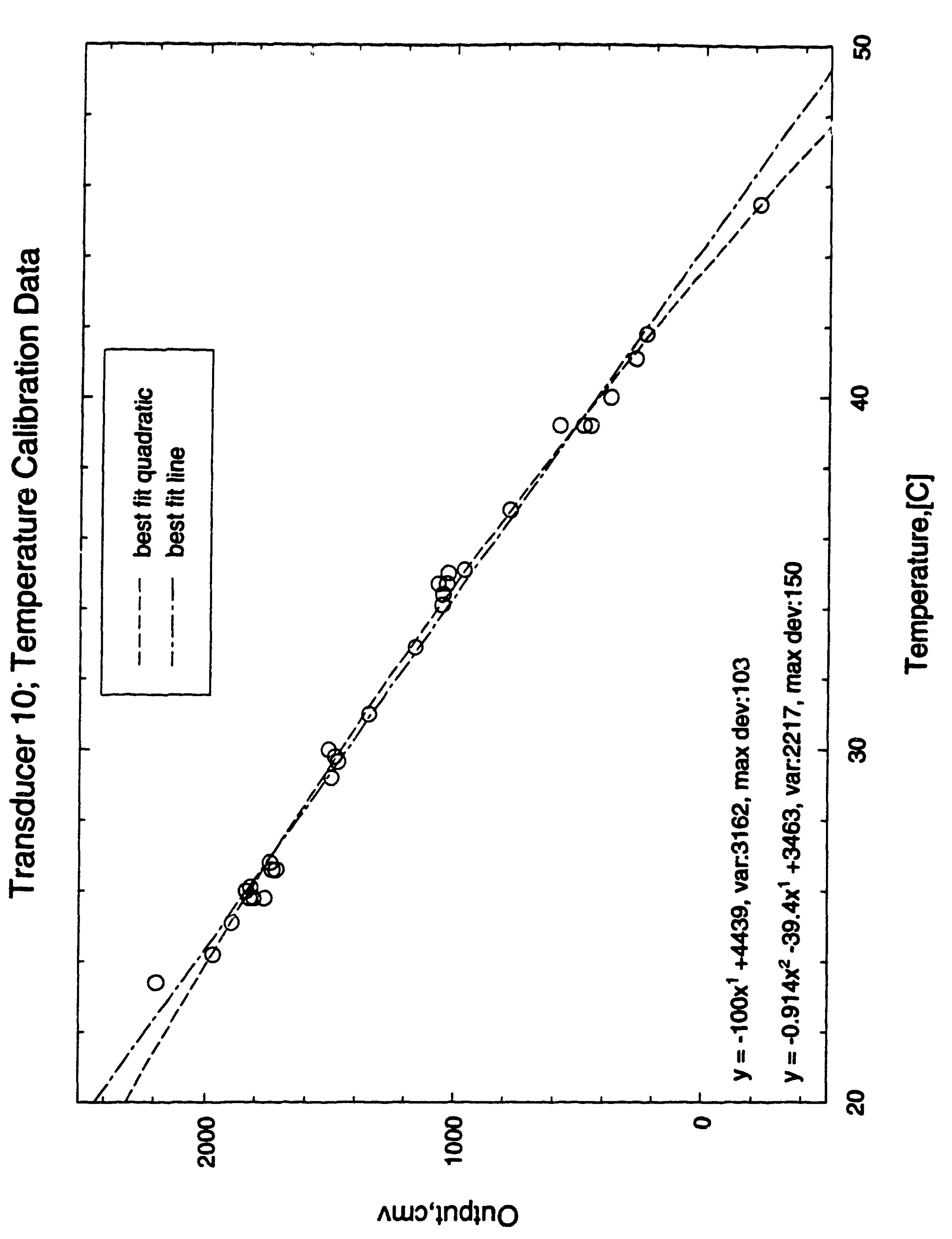


Figure C-8: Temperature Calibration of Prosthesis 33; Transducer 10

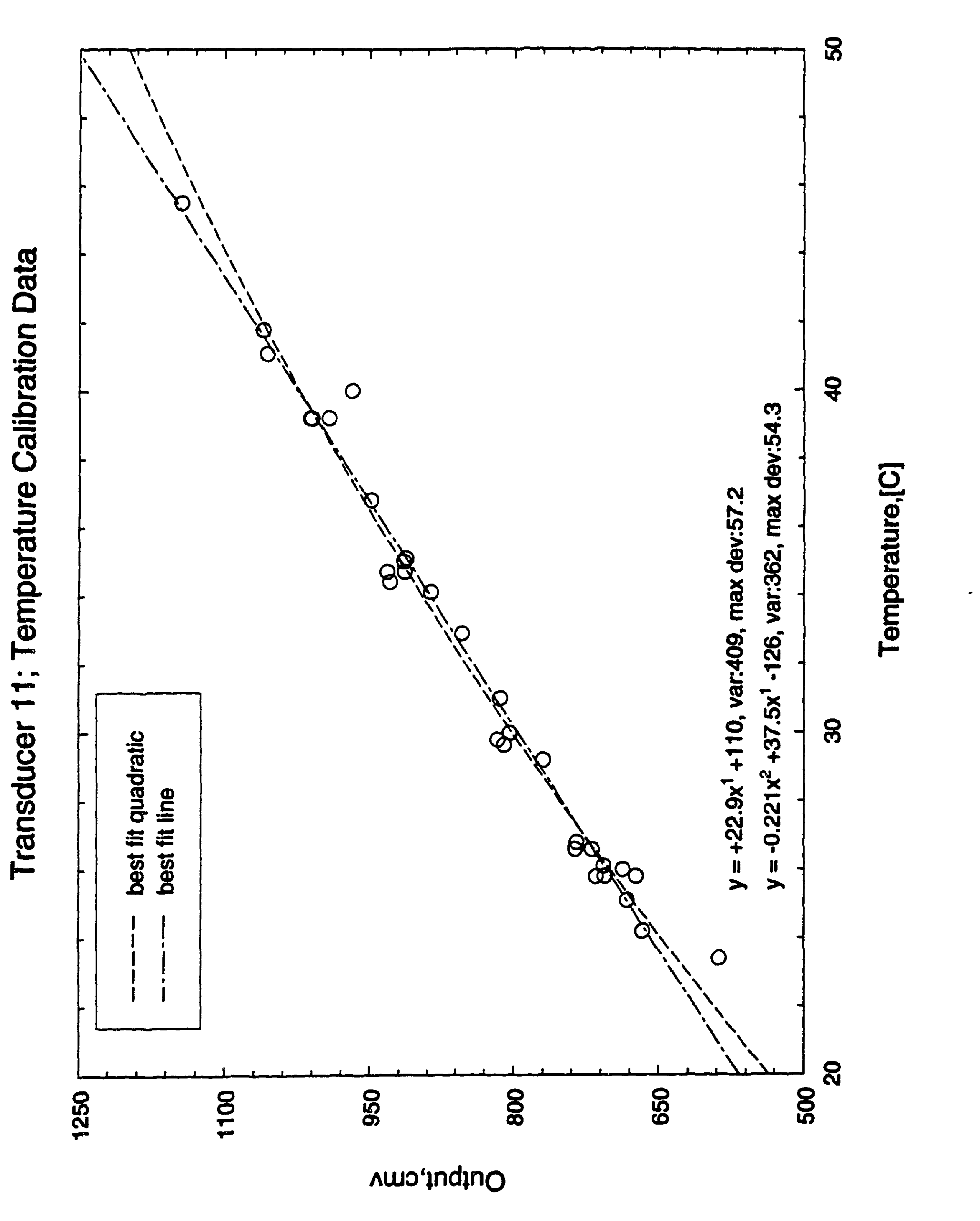


Figure C-9: Temperature Calibration of Prosthesis 33; Transducer 11

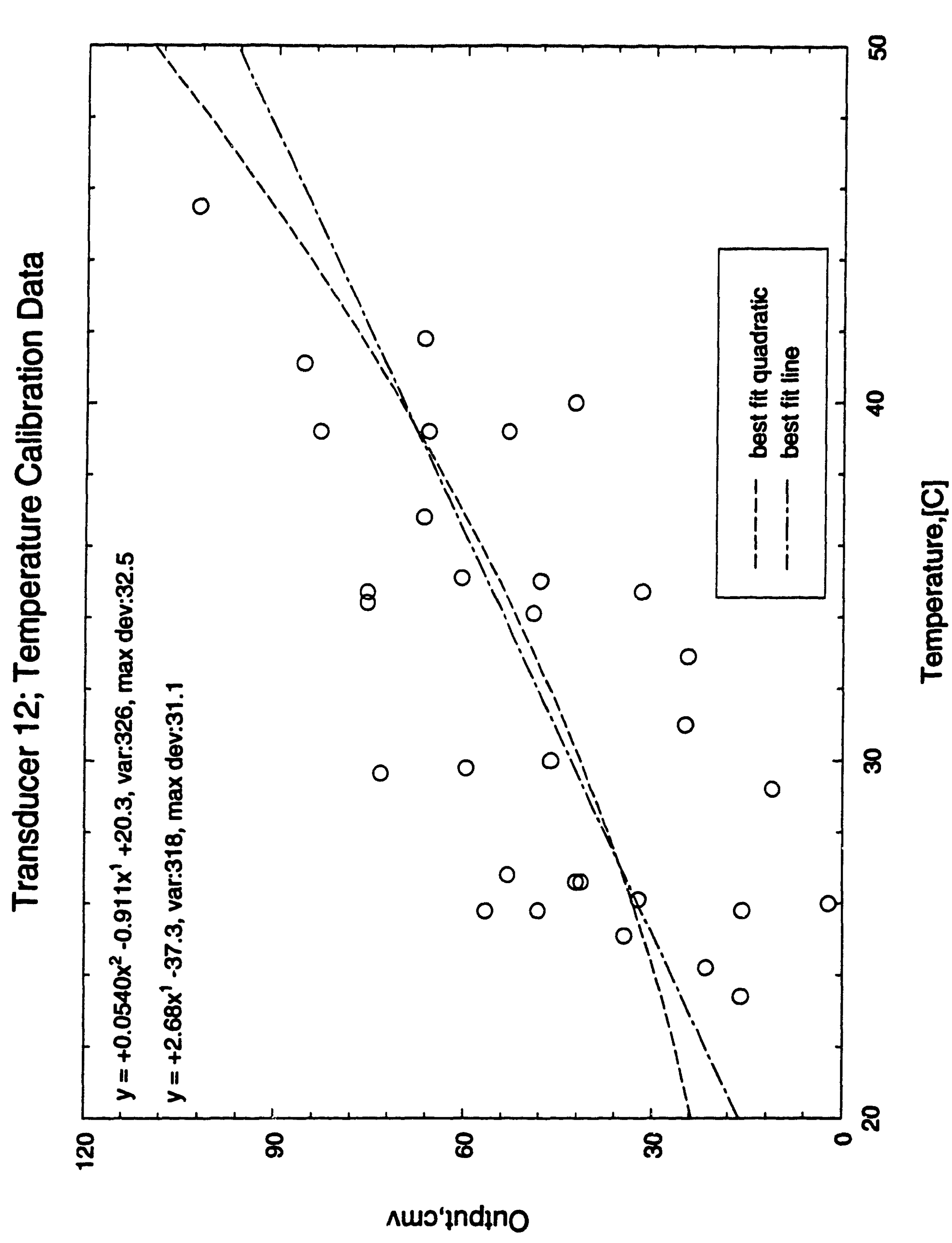


Figure C-10: Temperature Calibration of Prosthesis 33; Transducer 12

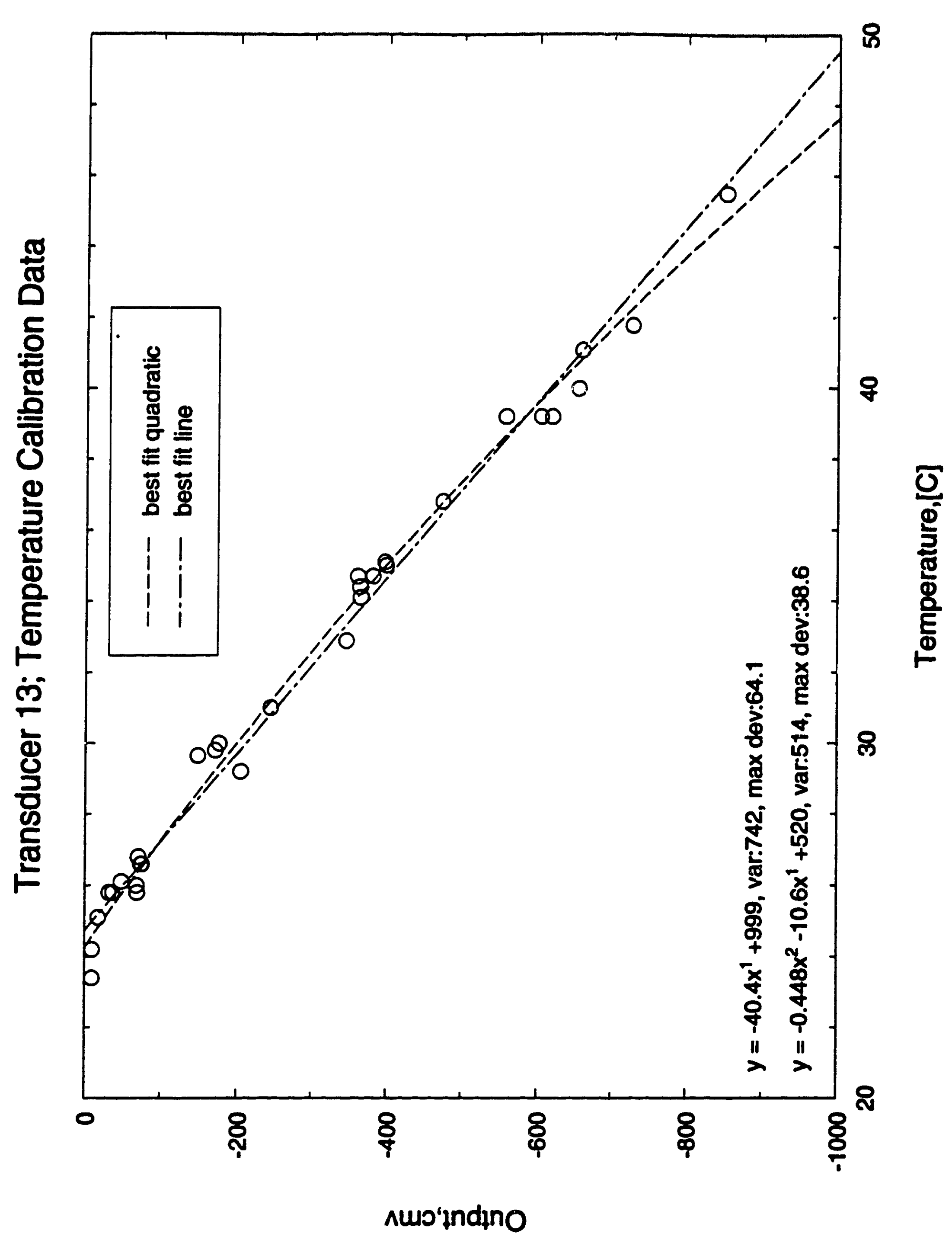


Figure C-11: Temperature Calibration of Prosthesis 33; Transducer 13

Table C.3: Thermal adjustment to transducer offsets for Prosthesis 43

Transducer	a_i	c_i
1	-0.0308	-153
2	-0.275	1009
3	-0.412	22
5	-0.150	555
6	0.201	359
7	-0.113	150
8	-0.0564	251
9	-0.2899	314
10	-0.4408	695
11	-0.177	-159
12	-0.0815	1068
13	-0.0722	-175

thermistor would respond to changes in temperature. The results of this series of tests are shown in figures C-26 and C-27.

Additionally, prosthesis 43 was subjected to pressure calibration tests at both room temperature and close to body temperature (38 C) to determine if transducer output gain with respect to pressure would be affected by temperature. The gains for the pressure transducers in this prosthesis, like those in the earlier instrumented prostheses, were found to be unaffected by temperature.

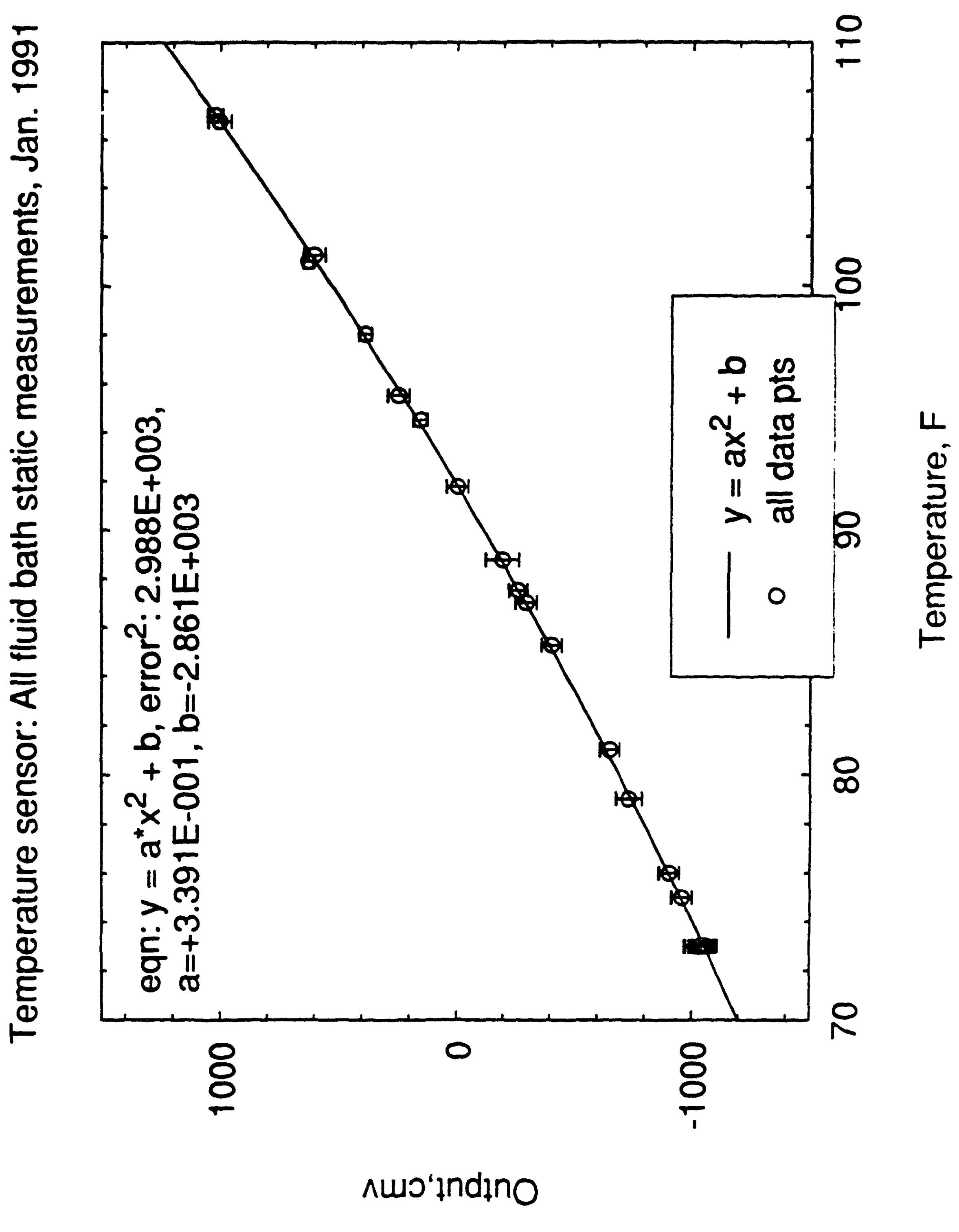


Figure C-12: Thermistor calibration

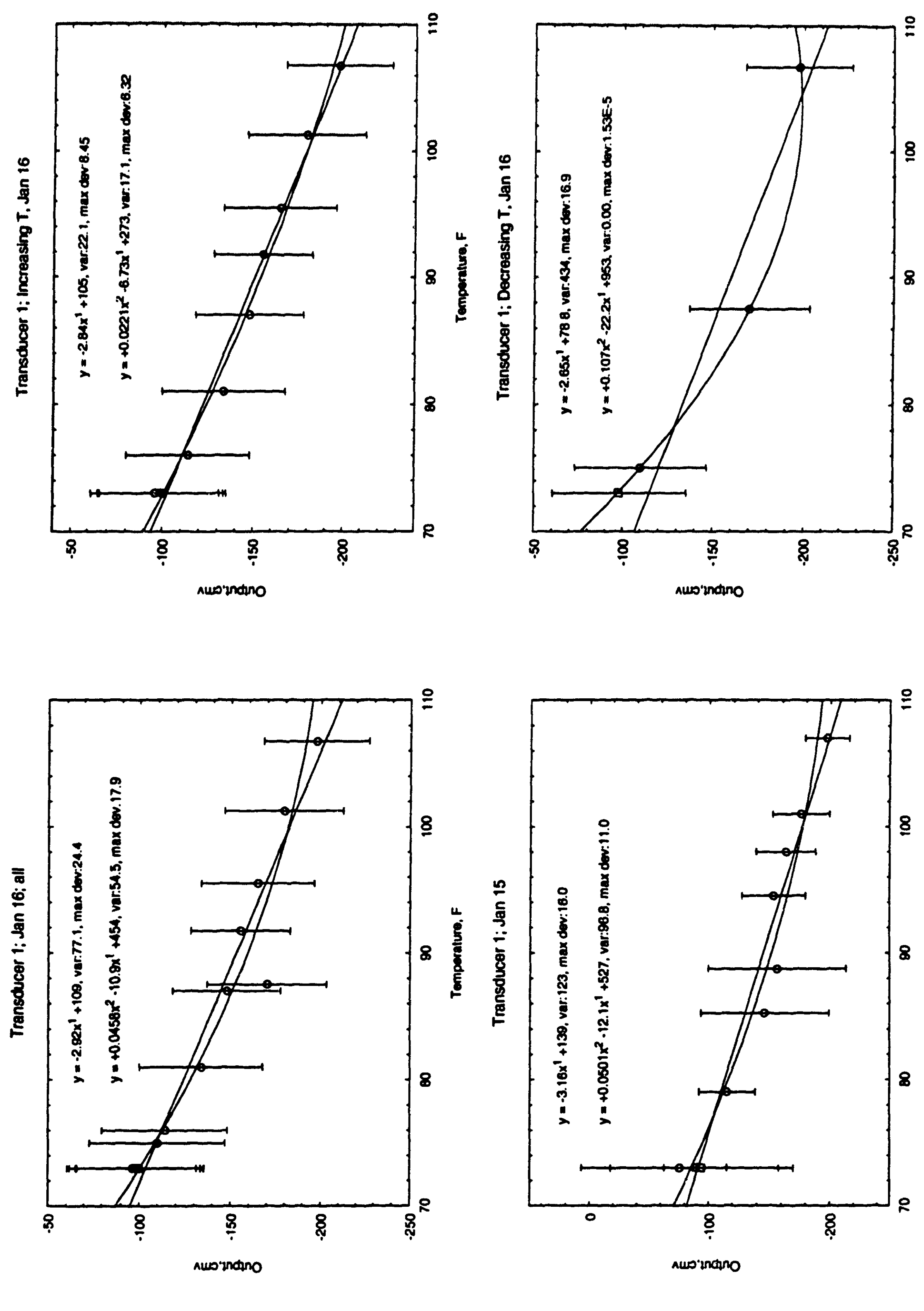


Figure C-13: Temperature Calibration of Prosthesis 43; Transducer 1

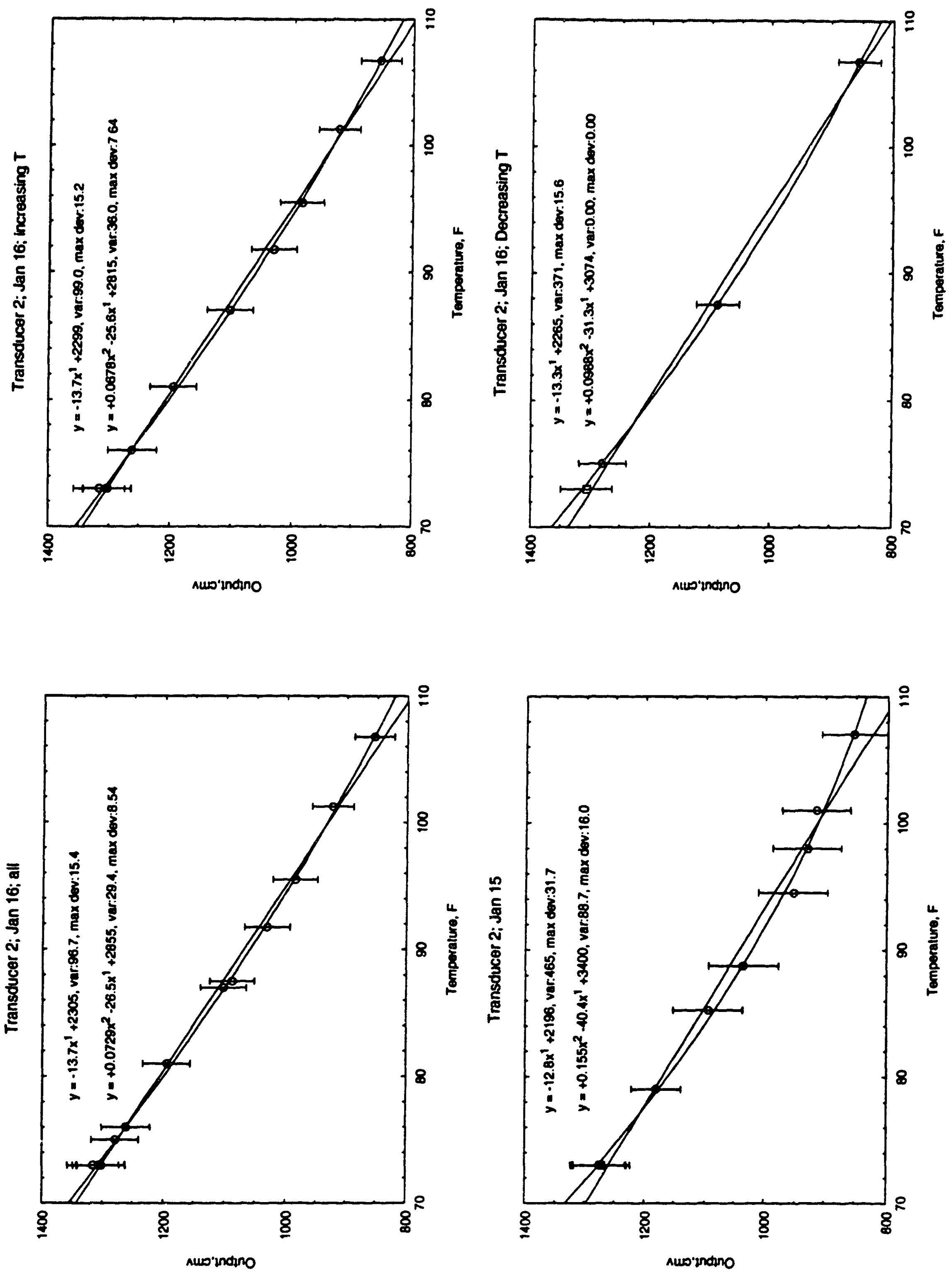


Figure C-14: Temperature Calibration of Prosthesis 43; Transducer 2

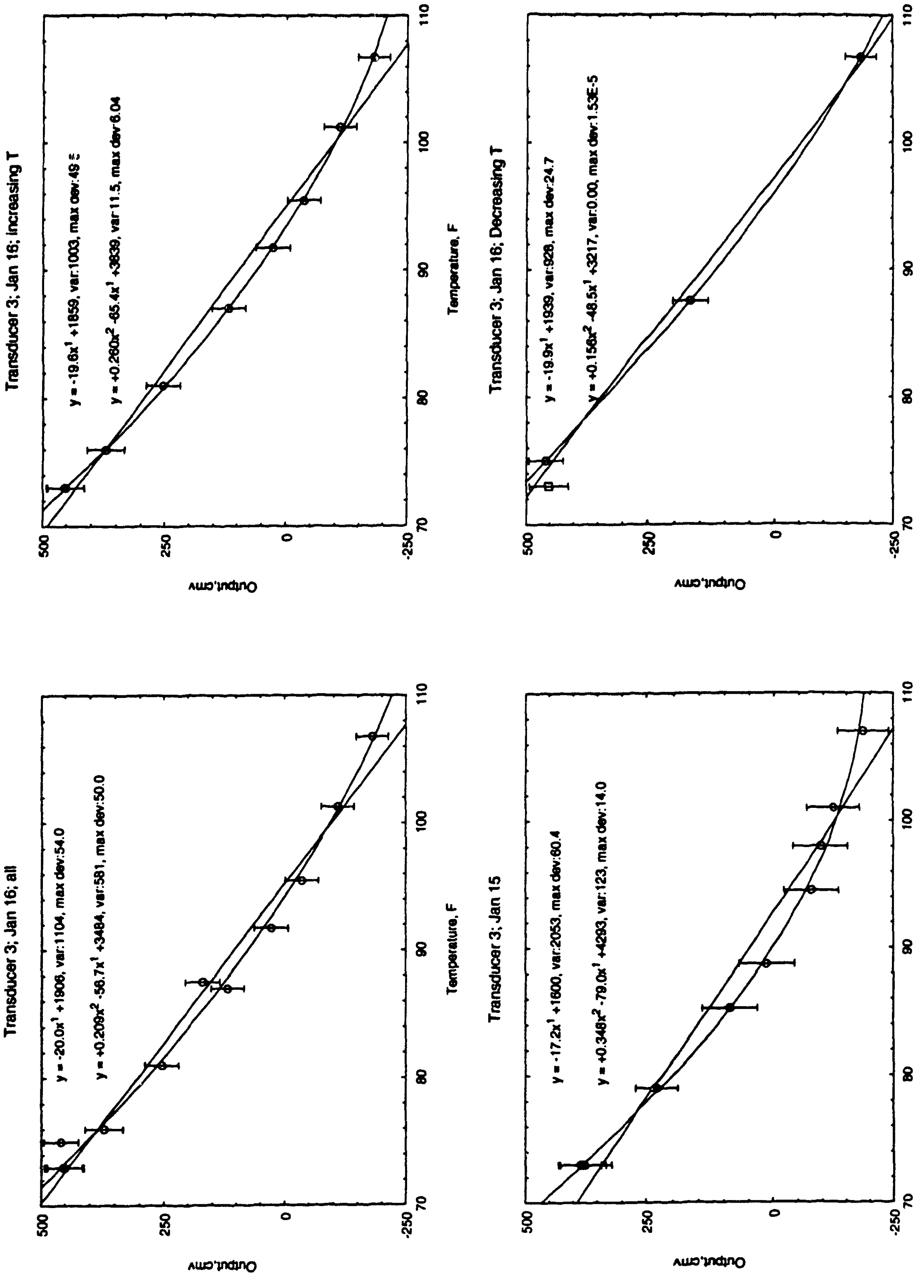


Figure C-15: Temperature Calibration of Prosthesis 43; Transducer 3

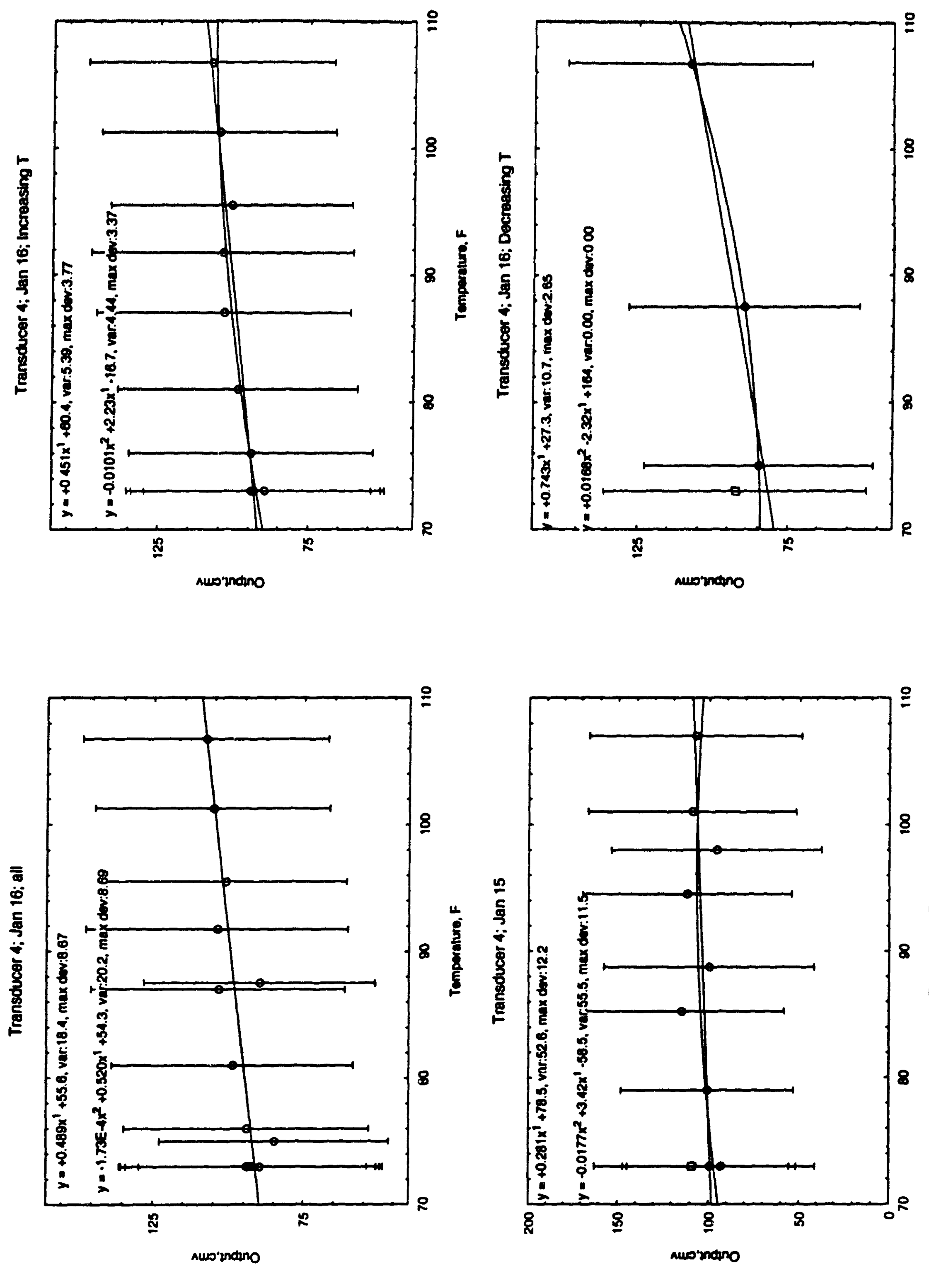


Figure C-16: Temperature Calibration of Prosthesis 43; Transducer 4

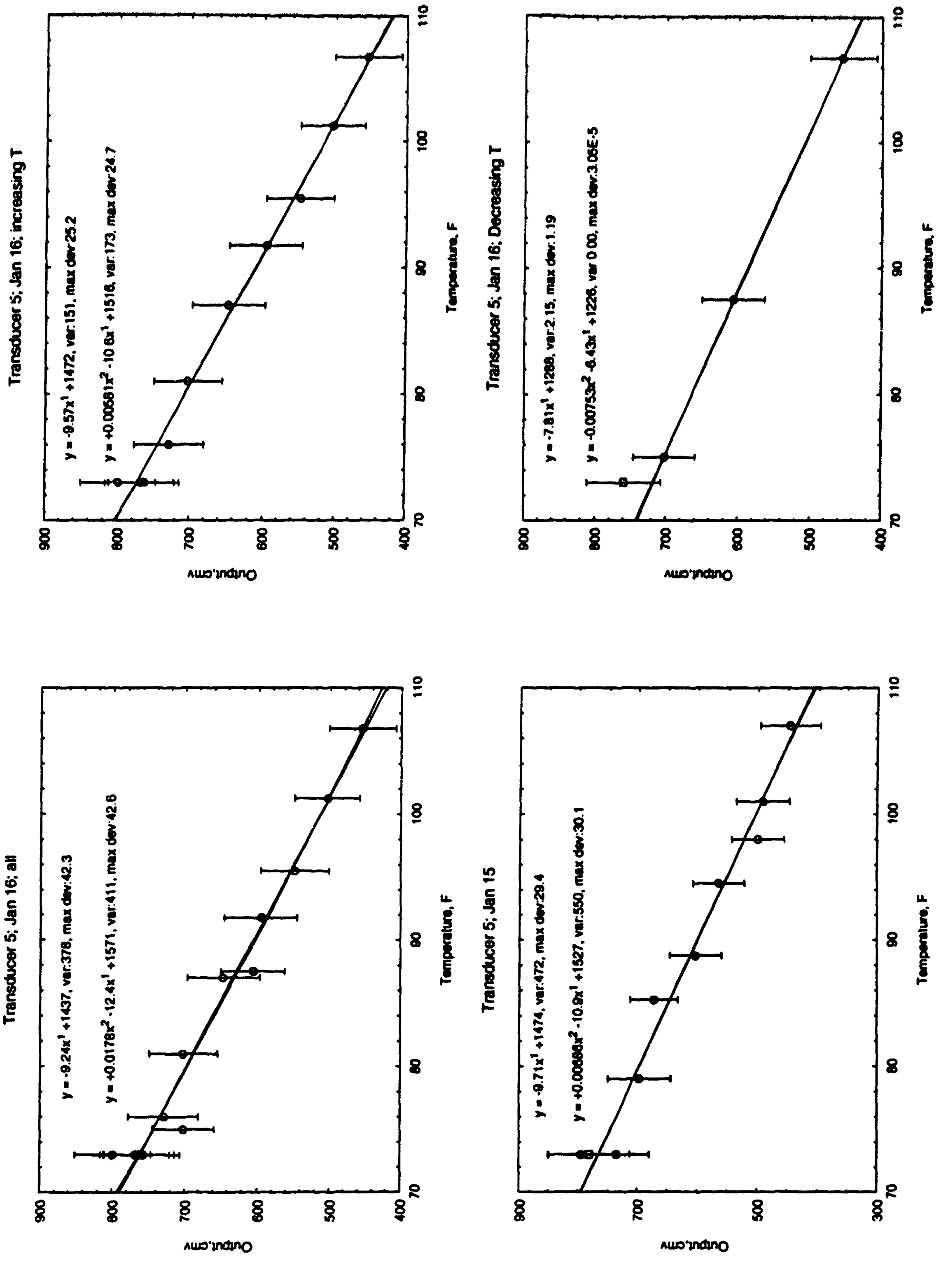


Figure C-17: Temperature Calibration of Prosthesis 43; Transducer 5

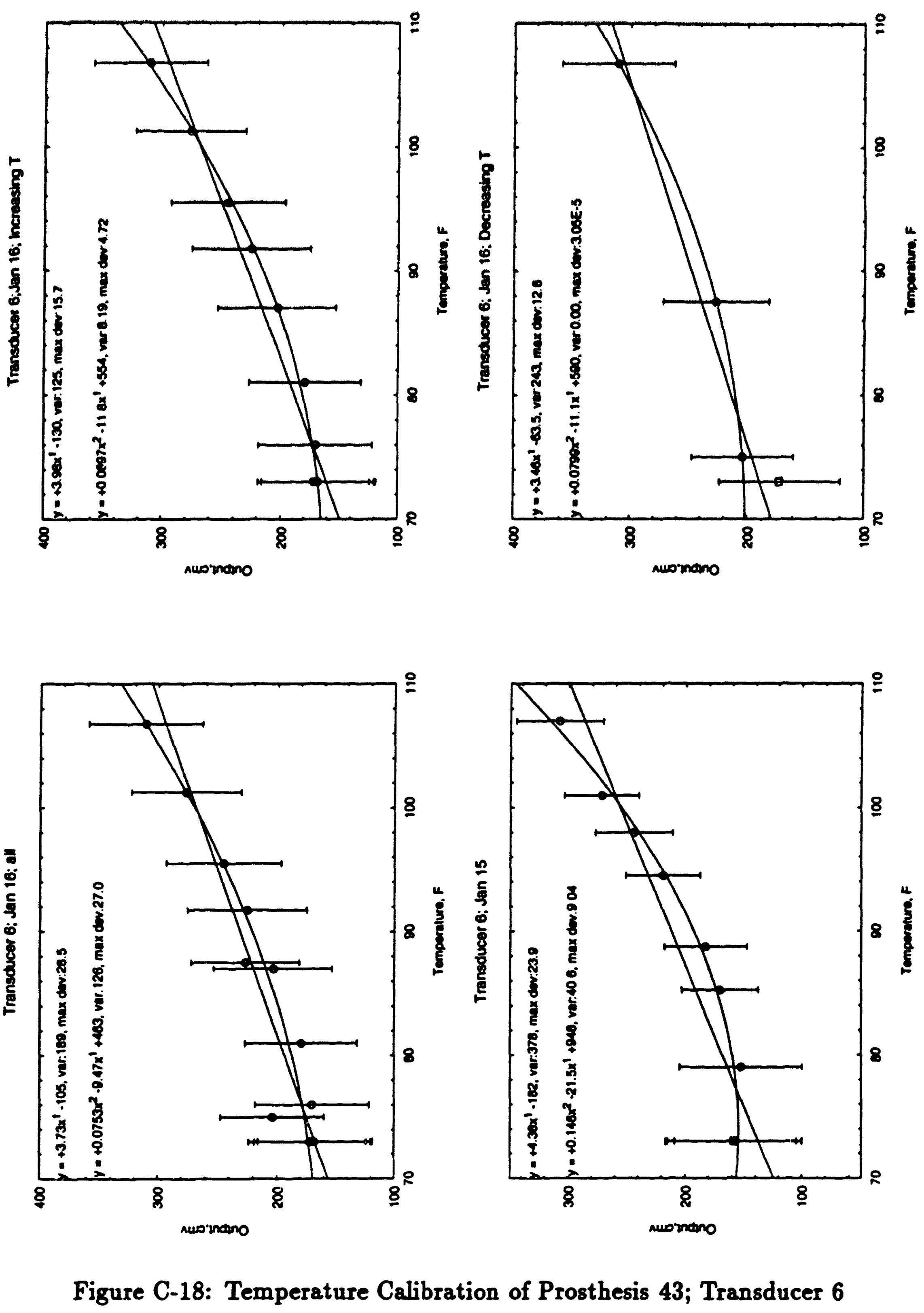


Figure C-18: Temperature Calibration of Prosthesis 43; Transducer 6

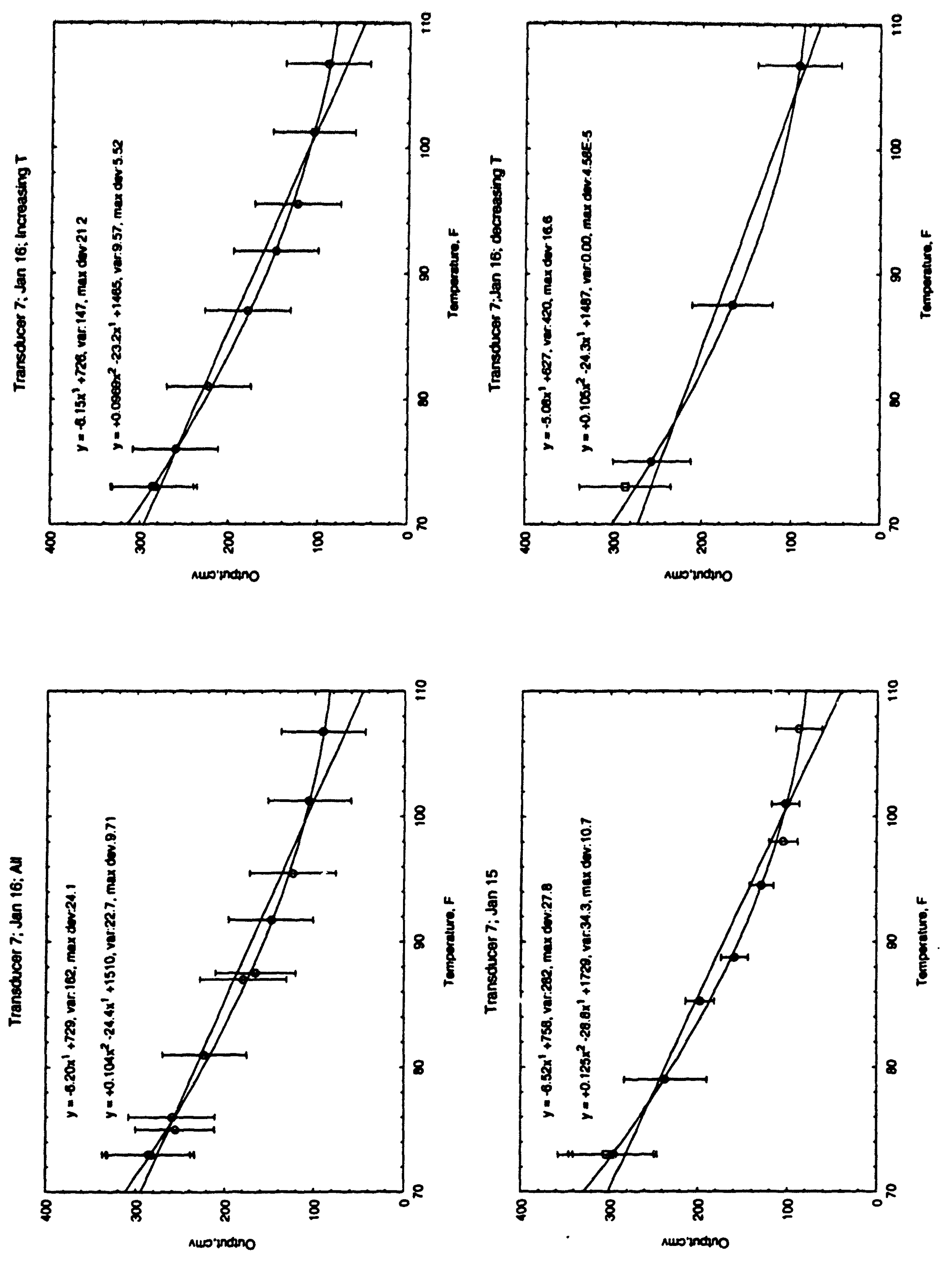


Figure C-19: Temperature Calibration of Prosthesis 43; Transducer 7

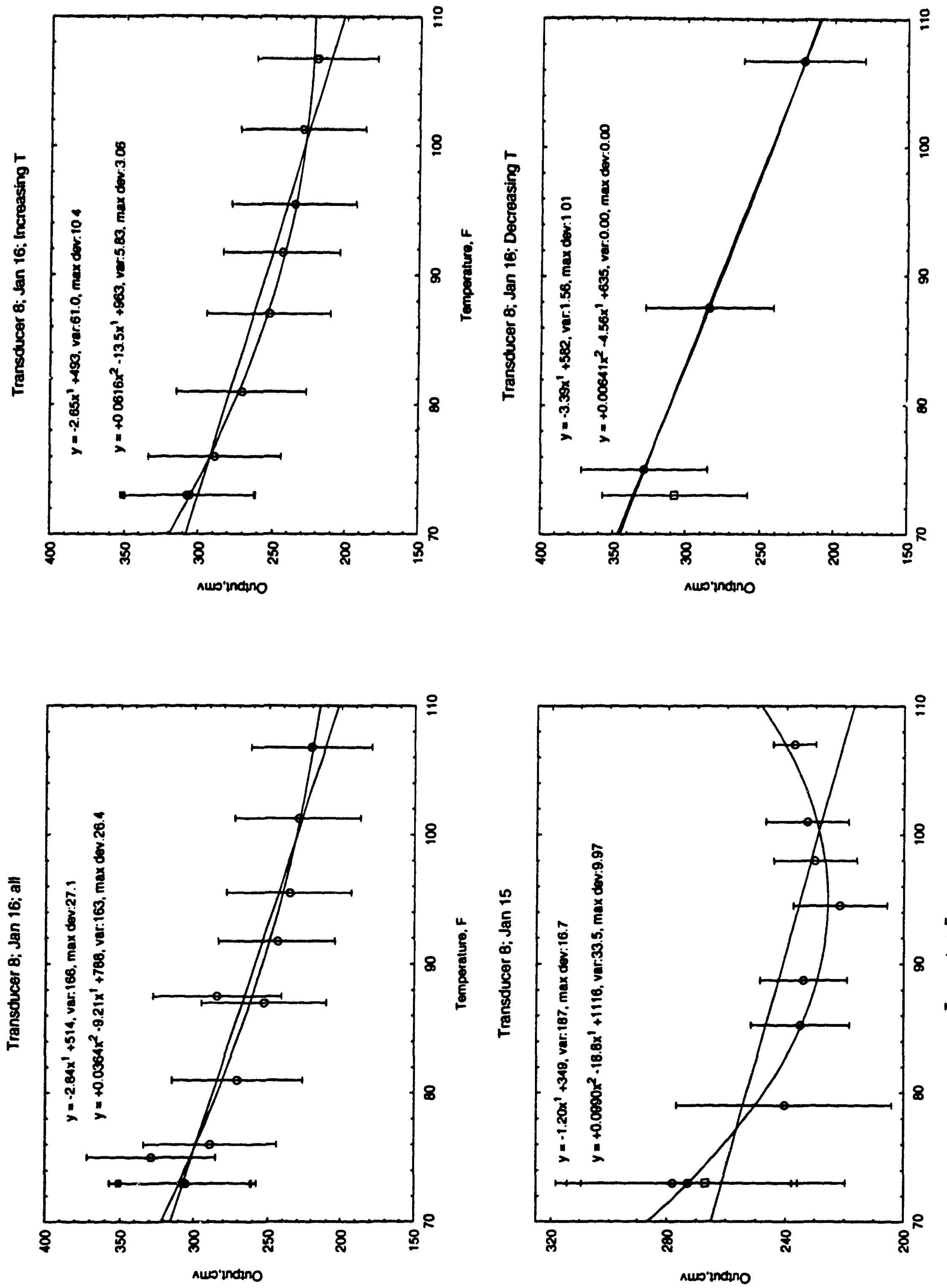


Figure C-20: Temperature Calibration of Prosthesis 43; Transducer 8

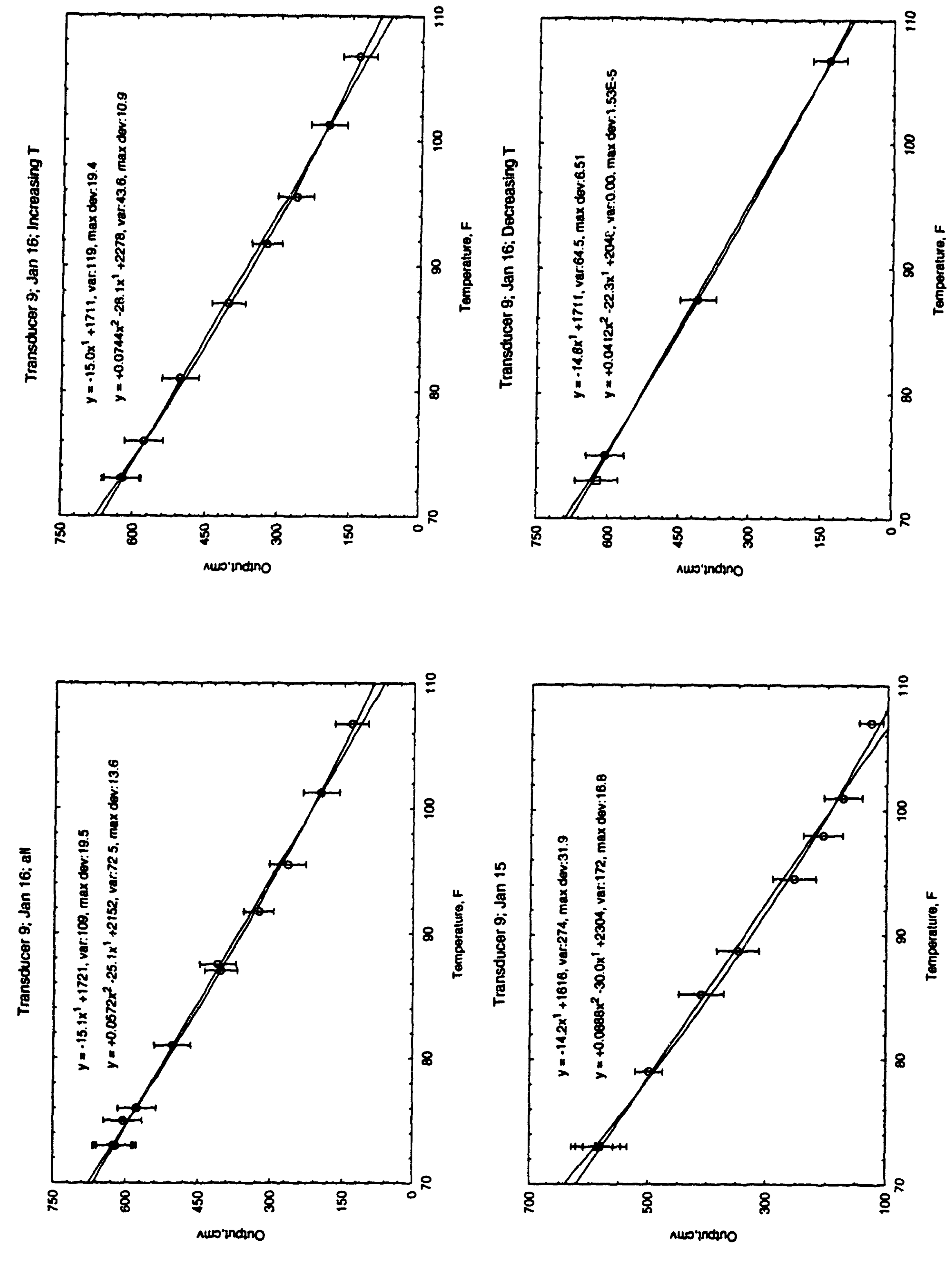


Figure C-21: Temperature Calibration of Prosthesis 43; Transducer 9

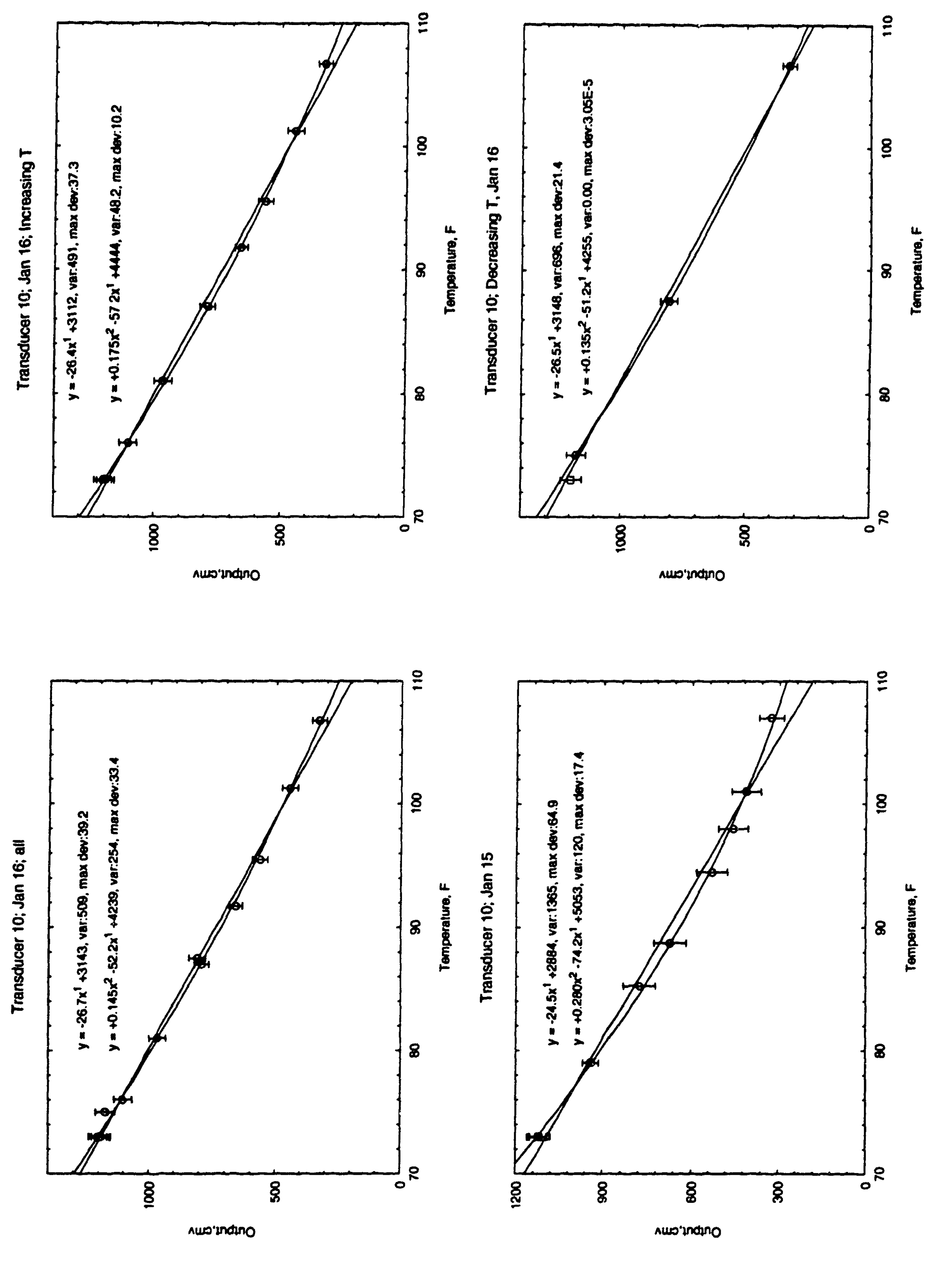


Figure C-22: Temperature Calibration of Prosthesis 43; Transducer 10

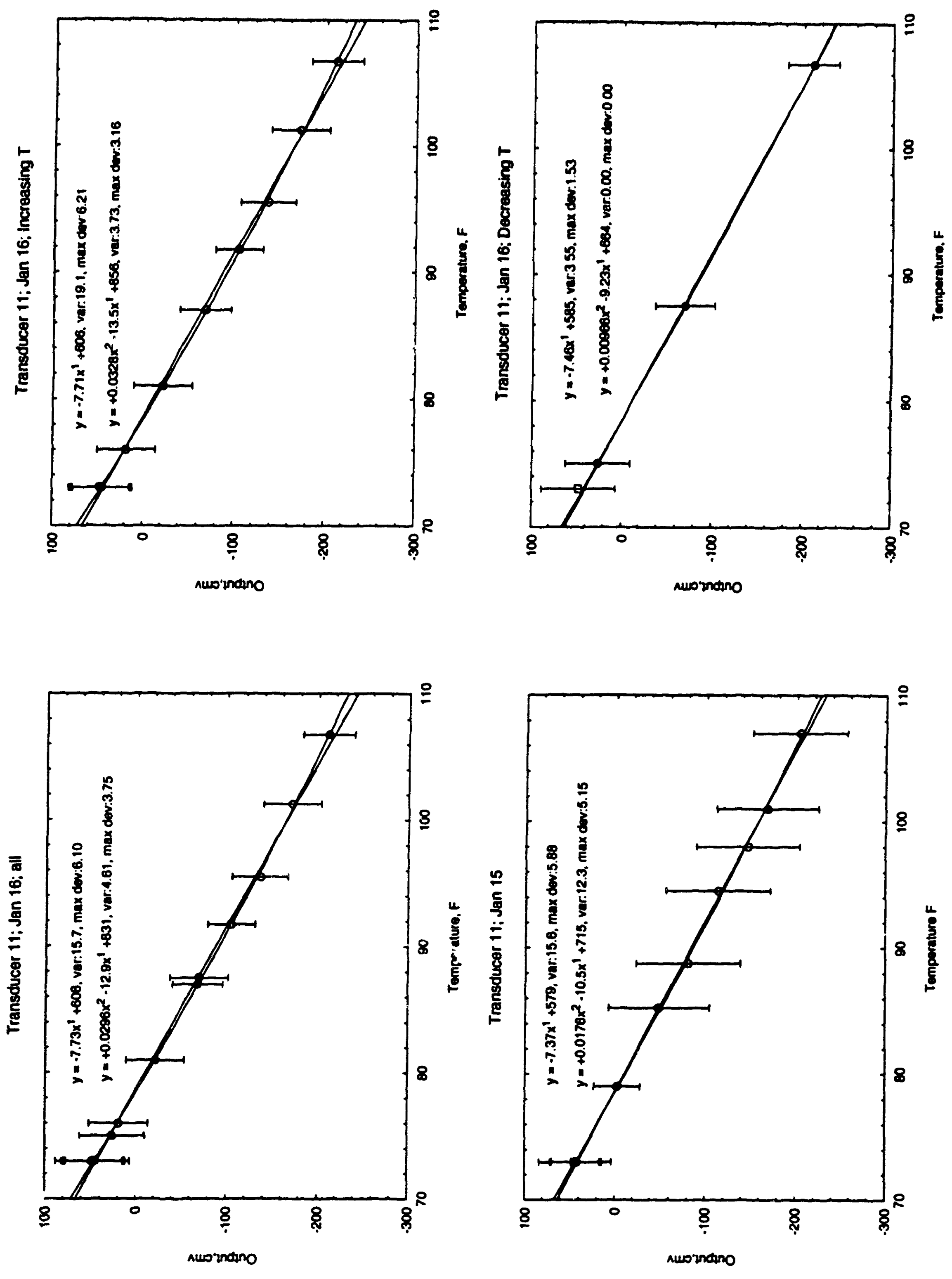


Figure C-23: Temperature Calibration of Prosthesis 43; Transducer 11

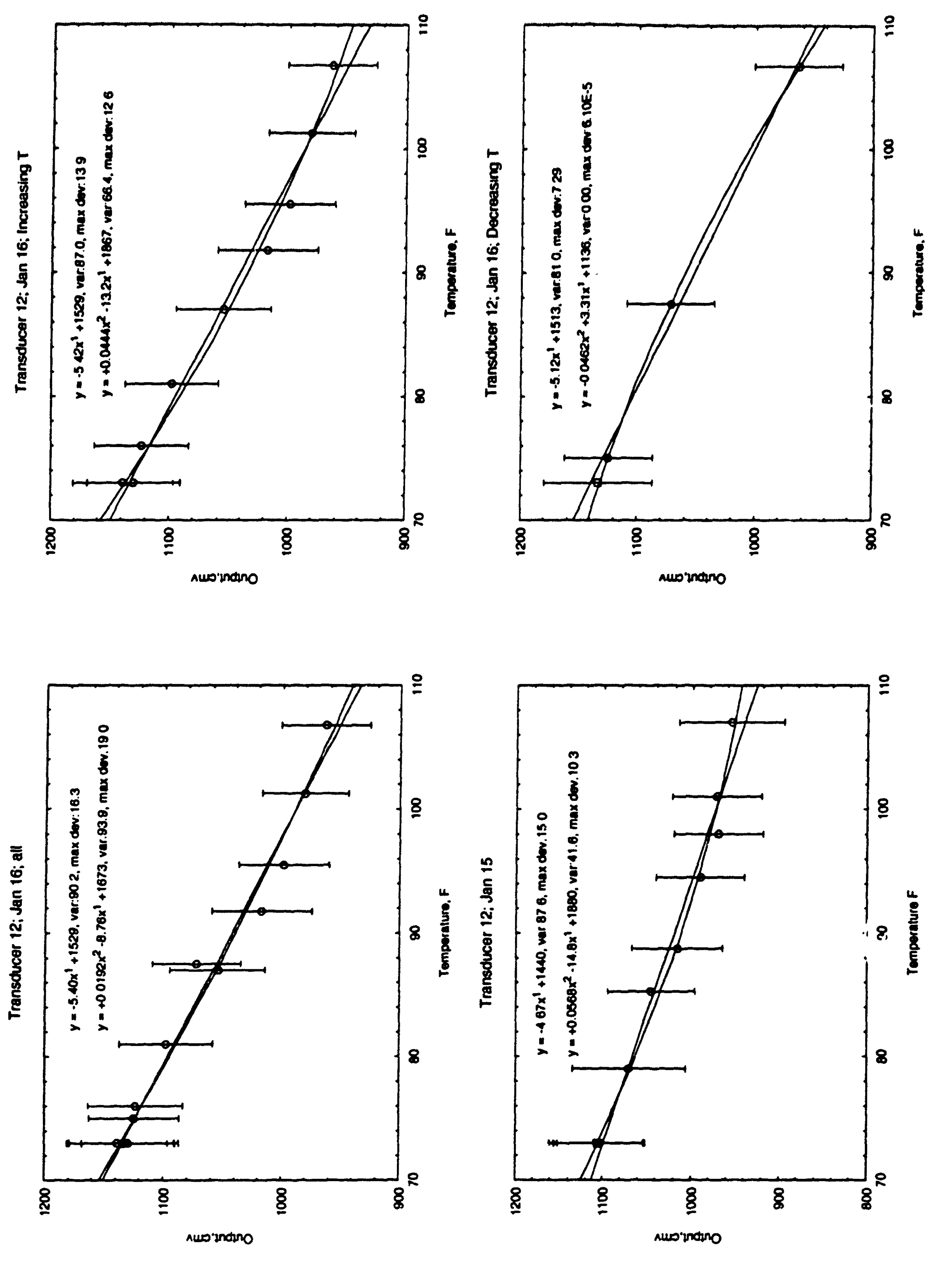


Figure C-24: Temperature Calibration of Prosthesis 43; Transducer 12

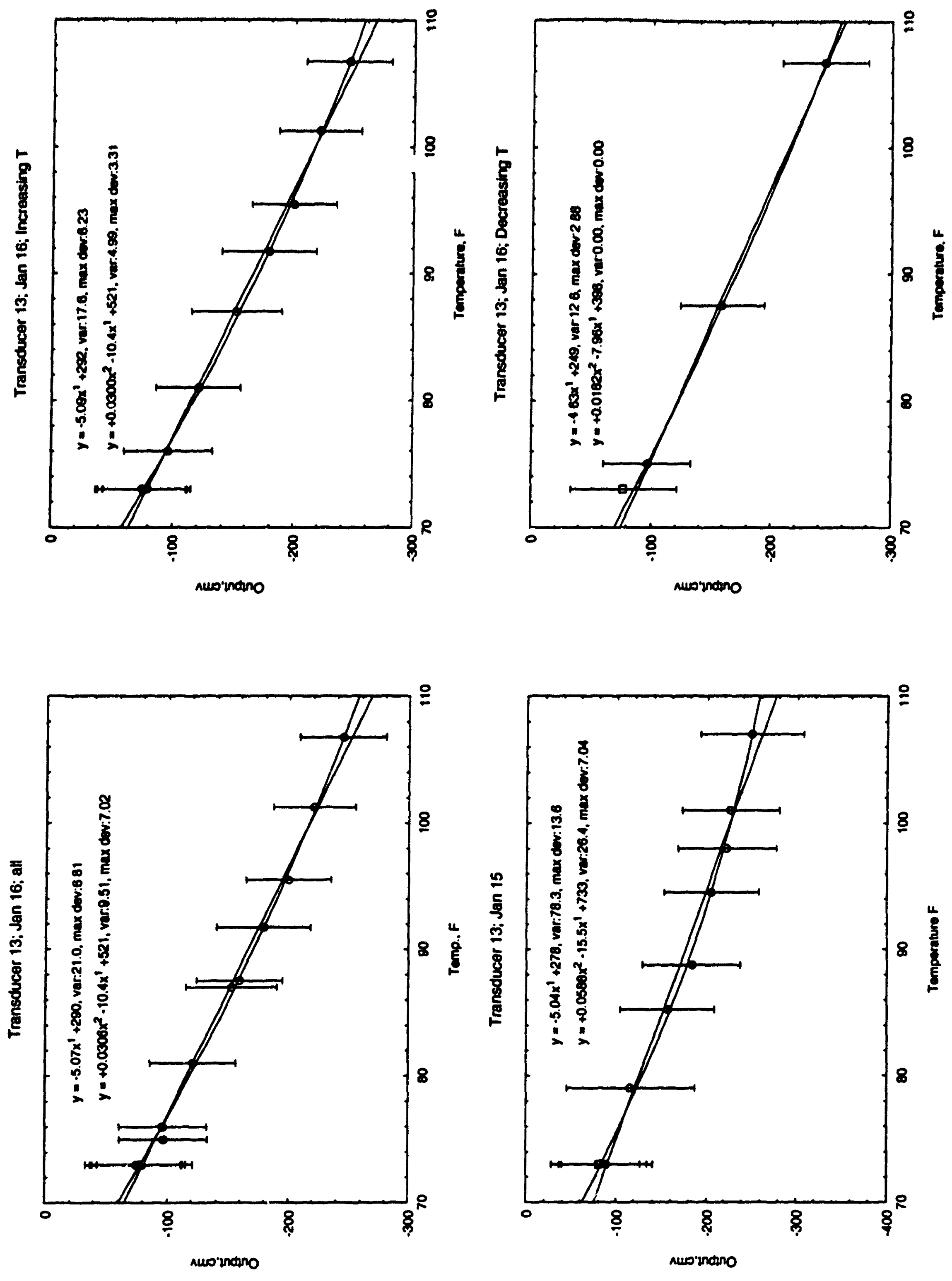


Figure C-25: Temperature Calibration of Prosthesis 43; Transducer 13

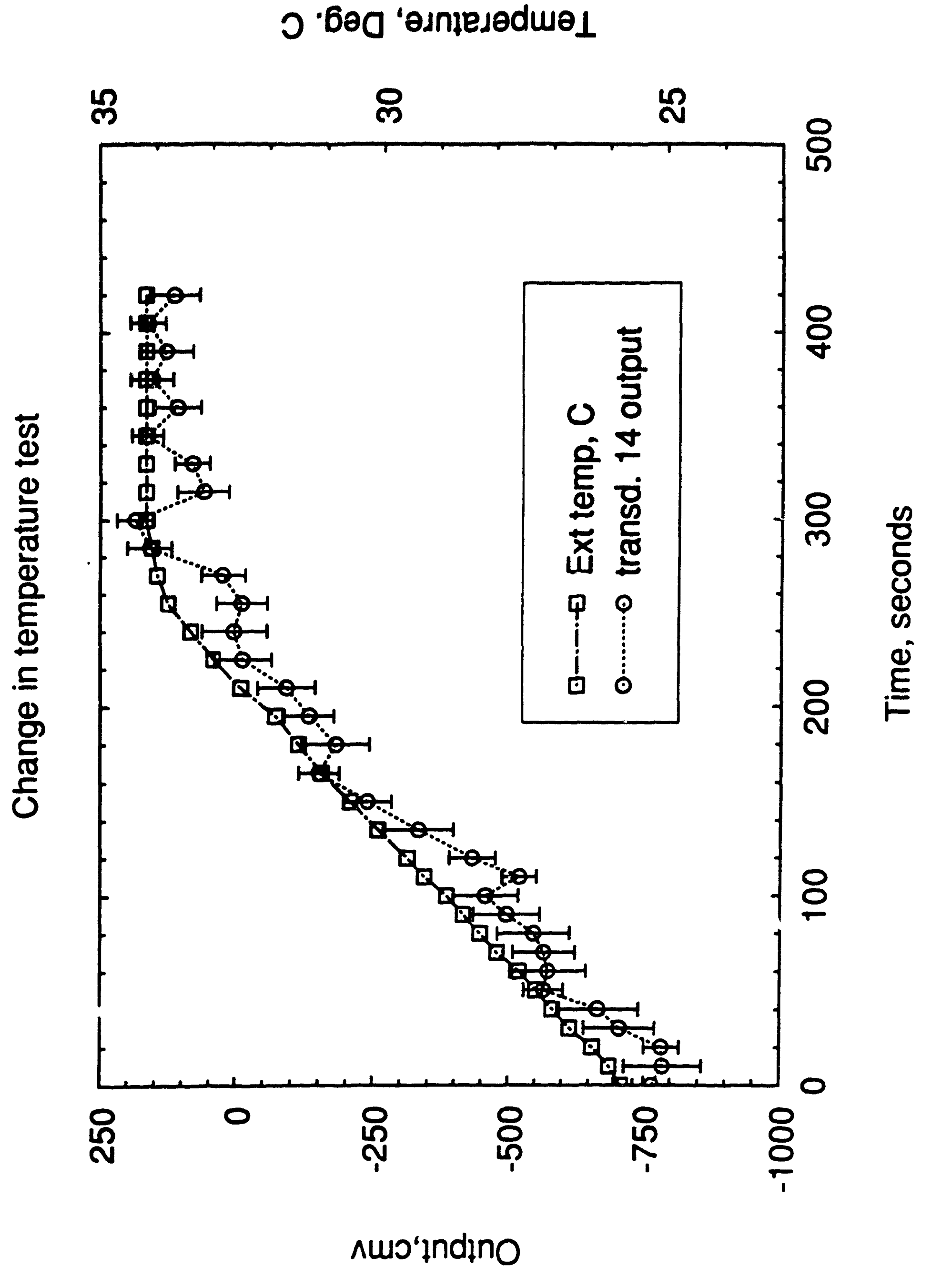


Figure C-26: Thermistor response to changes in temperature

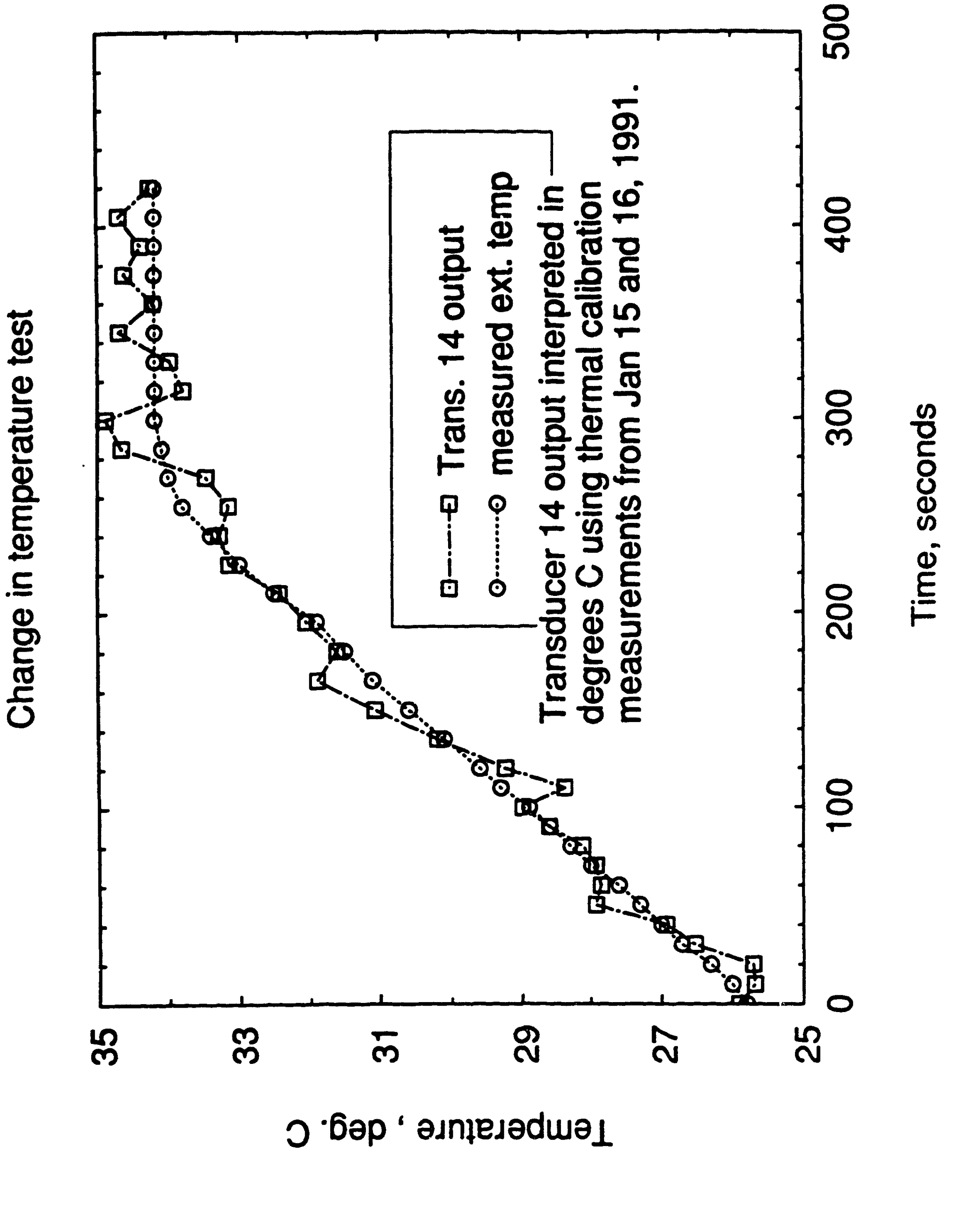


Figure C-27: Response to changes in temperature

Appendix D

Interface circuit between Prosthesis data receiver and the AT&T 6300+ PC

The interface between the receiver for the prosthesis output and the AT&T PC used to collect data *in vivo* from prosthesis 43 is straightforward for the most part. The data channel and the alarm channel are routed to A/D channels 0 and 1, auxiliary input is routed to A/D channel 2, and the strobe signal from the receiver which indicates the time at which the A/D channels should be sampled is routed to the external trigger set and external convert lines and used to trigger each scan of the channels being read. However, there is also a simple circuit that works in concert with the counter/timers on the A/D board to decrease the data collection rate from the prosthesis so that data can be continously collected for longer periods of time. Figure D-1 shows the circuit diagram for the interface box.

In order to take data for longer than 1 second (with prosthesis 43) and not overrun the DMA buffer the frequency at which data is taken needs to be slowed down. The A/D channels are sampled in sets of 32 to do this and still collect all transducers at essentially the same instant in time. Thus, the first reduction in data sampling rate is to alternate between collecting 32 samples of each A/D channel and skipping 32 times the A/D channel should be sampled. This is done by counting the channel-sample

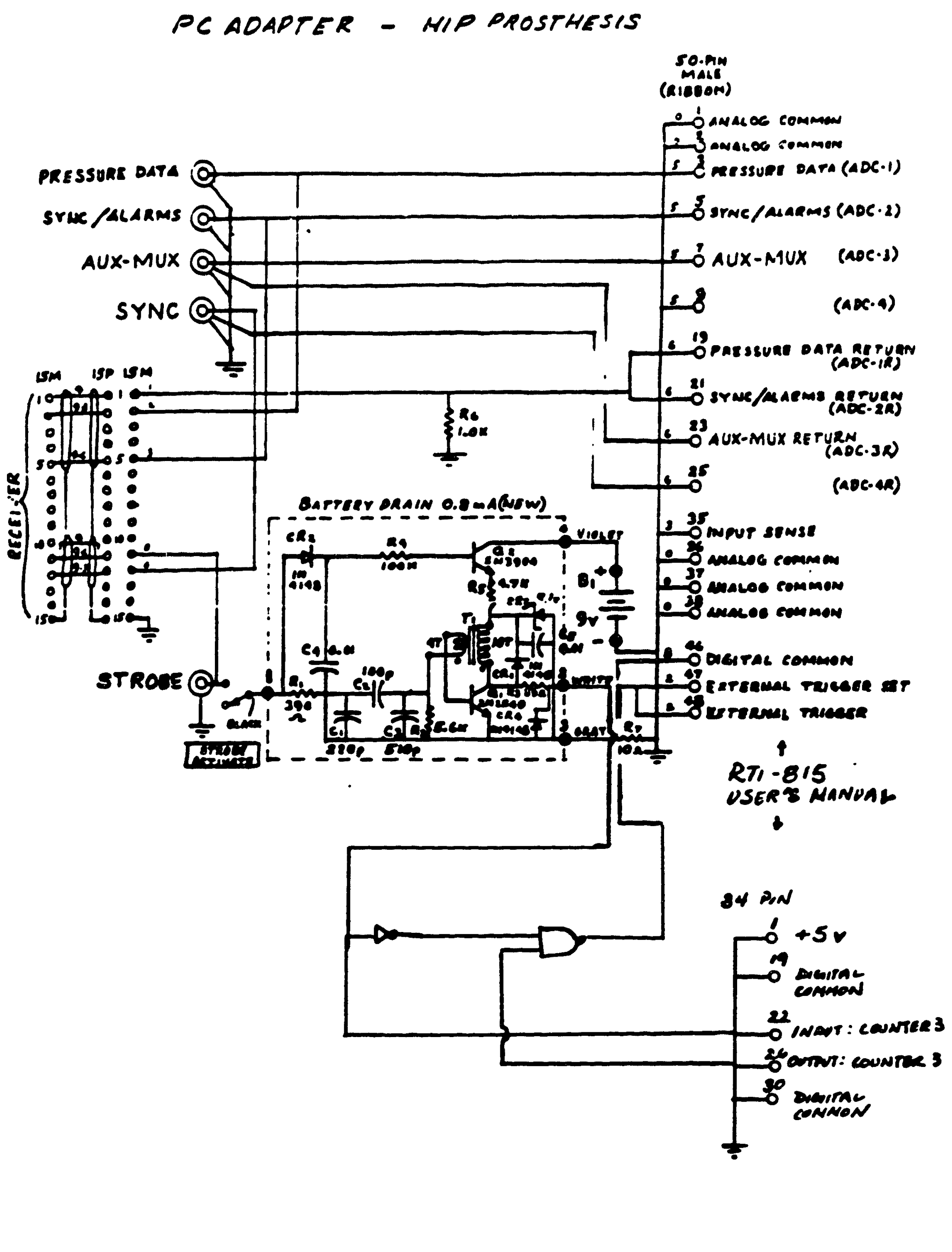


Figure D-1: Circuit Diagram for the Interface Box indicator, a strobe signal, using one of the counter/timers on the A/D board. The circuit in Figure A.3. suppresses the transmission of this strobe to the A/D sample trigger during the times when data shouldn't be collected.

Table 4.4 and 4.5 show the reduction in data sampling rate required when taking data for a given length of time.

Appendix E

Initial Processing of pressure data

The procedure to be followed to take pressure data on the PC is extremely simple. Figure E-1 shows the connections which must be made in order to correctly input data to the computer. One program is run, PCOLLECT.EXE; written in C. Figure E-2 shows a sample run through this program. The number of channels to collect is determined by whether any auxiliary data needs to be collected, such things as Selspot synch, forceplate forces, goniometer measurements. With no auxiliary data only 2 channels need to be collected, otherwise 3 channels must be collected. All data and the data directory is stored in ASCII so that it can be accessed more readily and transferred between machines more easily. The filename format that has been used includes the date data was taken and the file number, as mmddyy##.dat, where m stands for a two letter abbrev. for the month, d is the day of the month, y is the year. The programs used in data collection can be found in [8].

Processing of the data has a several steps. Three programs are used in the first stage of processing. The first of these is LONGDAT2.EXE which has as output a file with each frame of data, pressures in psi. The program AVGZERO.EXE must be used to process the zero data for the day if *in vivo* data was taken, the procedure followed is to run longdat2 on the zero data, then avgzero on the zero data, then use the results of that processing in longdat2 when processing the rest of the data for the day. Finally a program called AUX20_D.EXE is used to turn the results into a format that can immediately be plotted. Figure E-3 shows a flowchart of the

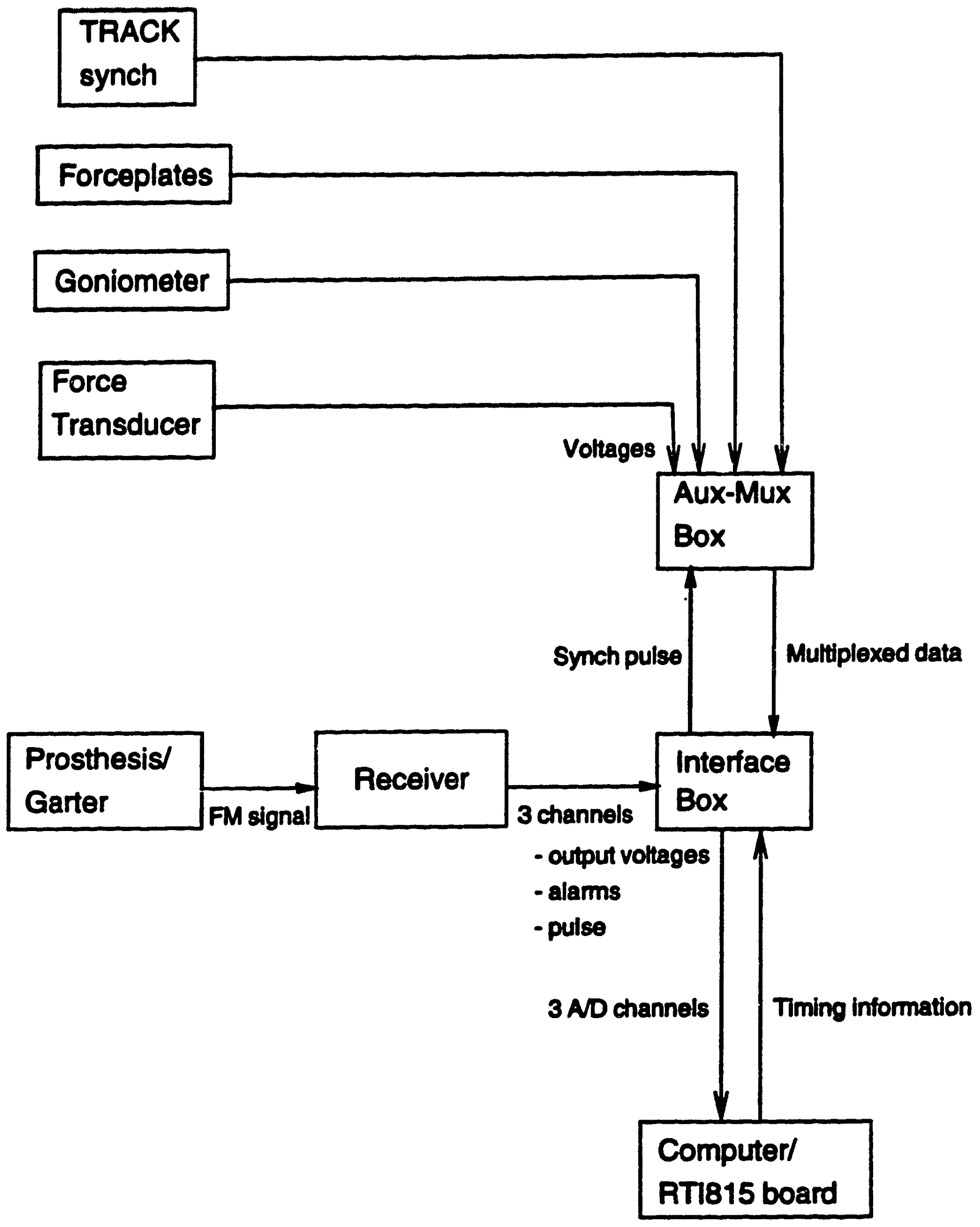


Figure E-1: Equipment Set up for Data Collection

```
E:\PRESS43>pcollect

Enter the number of channels to sample: 3

Enter the length of time to sample: 1.5

Please hit G when the prosthesis is powered up: G


(flip switch on interface box to start data collection)


Unplug the prosthesis and type C: C


(unplug prosthesis if it is prosthesis 33)


Do you want to display the data [Y:N]? N

Do you want to save this data [Y:N]: Y

Enter the prosthesis number: 43

Enter the complete filename: jn089303.dat

Enter the short filename: jn089303

Enter the data directory name: jun0893.dir

Enter the test description (30 letters or less): Free speed gait

Do you want to take more data [Y:N]? N

E:\PRESS43
```

Figure E-2: Sample run through *pcollect.exe* processing procedure. The programs used in processing are printed in [8].

The programs used to obtain acetabular locations of pressure measurements are also available in [8]. Three different programs exist, one for processing data from prosthesis 43 and two for processing data from prosthesis 33. Data files from prosthesis 33 were saved in a slightly different format from 1986 on, thus necessitating different programs to read the files. A flowchart of the programs used to obtain acetabular locations of pressure measurements is shown in figure E-4. The names of the executable programs are LOCATE43.EXE for processing data from prosthesis 43, KINE33.EXE for processing of early data from prosthesis 33, and KINE2.EXE for processing later data from prosthesis 33.

E.1 Programs for collection of pressure data

PCOLLECT.C

```
/* LONGIO.C   Data acquisition program for use with triggered */
/*            data collection.  Set up for prosthesis data.   */
/*            This version to collect longer data sets by        */
/*            ignoring some of the triggers. (Lower sampling rate)*/
/*      Mar. 8, 1991      */


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <pio.h>

void ad_setup(void);
void clk_setup_2(int, int, int, int);
int dma_setup(int *, int);
void press_display(int *, int, int);
void make_long (int *, int, int, int, float, float, int, int,\
                long,long,long, long);
```

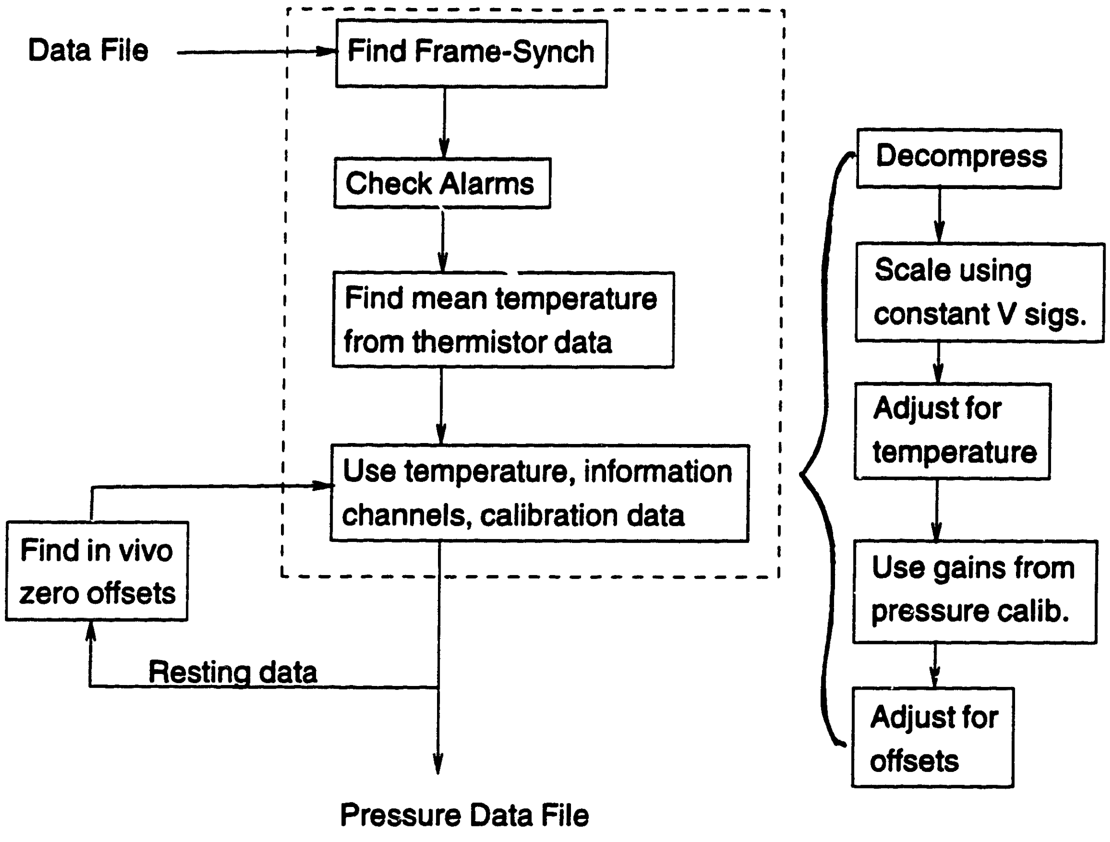

Data File
Find Frame-Synch
Check Alarms
Find mean temperature from thermistor data
Use temperature, information channels, calibration data
Find in vivo zero offsets
Resting data
Pressure Data File
Decompress
Scale using constant V sigs.
Adjust for temperature
Use gains from pressure calib.
Adjust for offsets

Figure E-3: Initial Processing Procedure

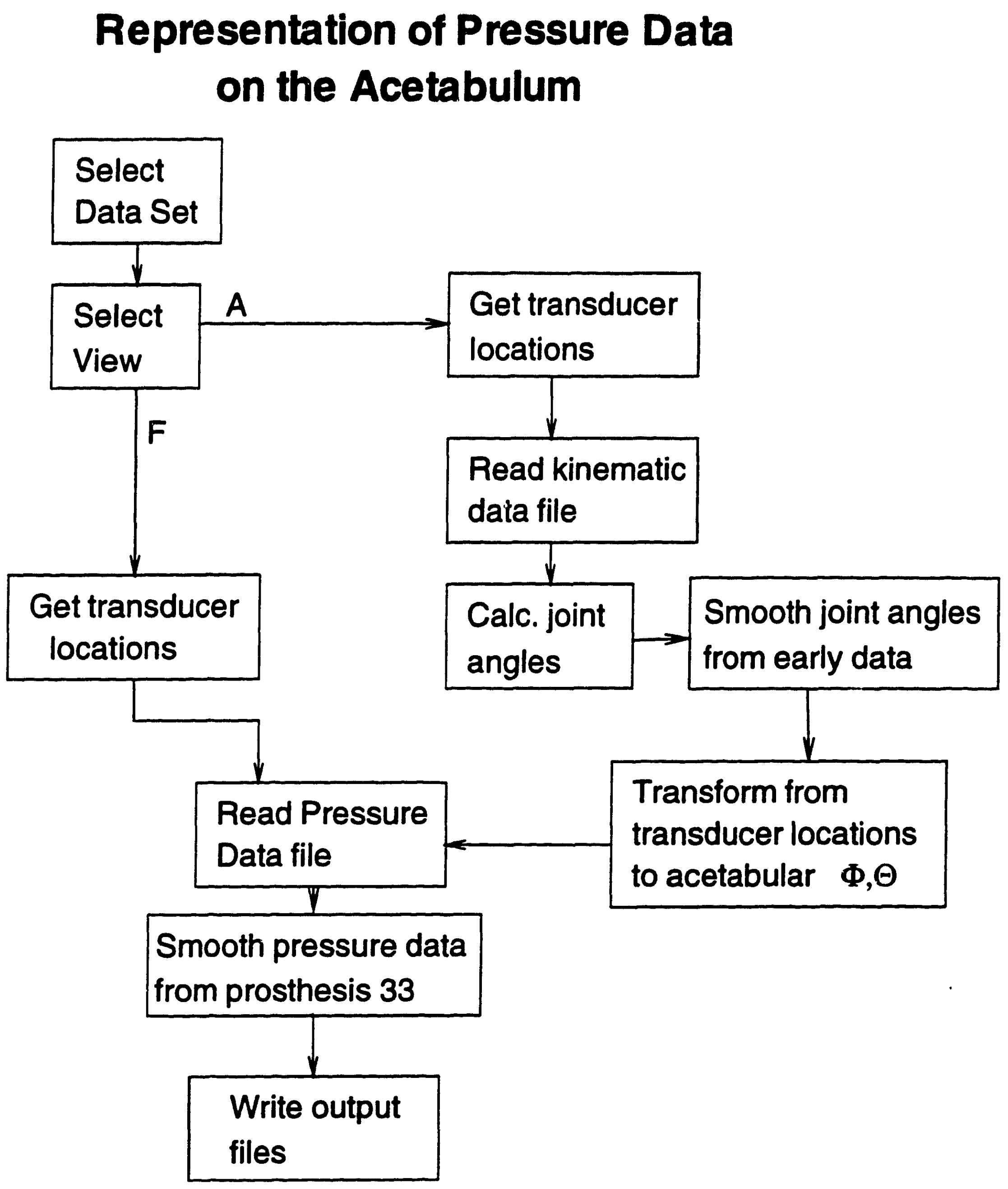

Representation of Pressure Data on the Acetabulum
Select Data Set
Select View
A
F
Get transducer locations
Read kinematic data file
Calc. joint angles
Smooth joint angles from early data
Get transducer locations
Transform from transducer locations to acetabular Φ,Θ
Read Pressure Data file
Smooth pressure data from prosthesis 33
Write output files

Figure E-4: Procedure for obtaining acetabular locations of pressure measurements

```
int var1[D_ARRSIZE];


main(){

          int *p_var1d;

          register i;
          register j;
          int idelay = 0;
          int n = 0;
       extern int var1[];

          int er;
          int count = 2;
          int nsamp = 99;
          float stime = 0.1;
          float samp,eff_freq;

          unsigned out_1;
          unsigned out_2;
          unsigned low_mux;
          int arr_times;
          int dma_stat,ad_stat,clk_stat,mux_stat;
          int cur_addr;
          int check;
          int byt_count = (count * nsamp * 2);
          int trig_count, trig_skip;
          int nrep = nsamp-1;
          long pow_time,start_time,end_time,off_time;
          char reply[5];
          char cwait[2];
          char contin[5];
          contin[0]='Y';
```

```
        for(i = 0; i < D_ARRSIZE; i++){
                var1[i]=0;
                }
        p_var1d = &var1[0];


        while(contin[0]=='Y'|| contin[0]=='y'){


/*  Get input of the number of delay cycles, # scan cycles, # channels */


                printf("Enter the number of channels to sample:  ");
                scanf("%d",&count);
                printf("Enter the length of time to sample:  ");
                scanf("%f",&stime);


                if (count > TOTCHAN){
                        count = TOTCHAN;
                        }
                low_mux = 128+count;
                samp = stime*count*P43_FREQ; /* Only figured for 43 */

/* Not working correctly here, perhaps only one elseif is allowed?
   Could use switch, would be just as convenient */

                arr_times = (samp / D_ARRSIZE);
                switch (arr_times){
                        case 0:
/*                        printf("This data set could be collected with trigio\n");
                          printf("which would result in a faster sampling rate.\n");
                          printf("Continue? [D:N] ");
                          contin[0] = 'N';
                          scanf("%s",contin);
                          if (contin[0]=='n'||contin[0]=='N') exit(0);
```

```
*/
                trig_skip = 0;
                trig_count = 32;
                nsamp = samp;
                eff_freq = P43_FREQ;
                break;
        case 1:
                trig_skip = 32;
                trig_count = 32;
                nsamp = samp/2.;
                eff_freq = P43_FREQ/4.;
                break;
        case 2:
                trig_skip = 64;
                trig_count = 32;
                nsamp = samp/3.;
                eff_freq = P43_FREQ/6.;
                break;
        case 3:
                trig_skip = 96;
                trig_count = 32;
                nsamp = samp/4.;
                eff_freq = P43_FREQ/8.;
                break;
        case 4:
                trig_skip = 128;
                trig_count = 32;
                nsamp = samp/5.;
                eff_freq = P43_FREQ/10.;
                break;
        case 5:
                trig_skip = 160;
                trig_count = 32;
                nsamp = samp/6.;
                eff_freq = P43_FREQ/12.;
                break;
        case 6:
                trig_skip = 192;
                trig_count = 32;
                nsamp = samp/7.;
                eff_freq = P43_FREQ/14.;
                break;
        case 7:
                trig_skip = 224;
                trig_count = 32;
                nsamp = samp/8.;
                eff_freq = P43_FREQ/16.;
                break;
        case 8:
                trig_skip = 256;
                trig_count = 32;
                nsamp = samp/9.;
                eff_freq = P43_FREQ/18.;
                break;
        case 9:
                trig_skip = 288;
                trig_count = 32;
                nsamp = samp/10.;
                eff_freq = P43_FREQ/20.;
                break;
        default:
         printf("Program not prepared for this length");
         printf(" data set.    Edit longio.c.\n");
         exit(0);
         break;
        }

    nrep = (nsamp/count)-1;
    byt_count = nsamp*2;
/*  printf ("%x\n", byt_count);*/
    check = 0x80 + count;
```

```
/* Ask user to wait until prosthesis is plugged in*/
                printf("Please hit G when the prosthesis is powered up:  ");
                do{
                        scanf("%s",cwait);
                }
                while( strcmp(cwait,'g' )== 0);

/* Get the time here and save it */
                time(&pow_time);

/* Initialize the board */



                INIT(&er);
                if (er!=0){
                        printf("Error during initialization %h \n",er);
                        }

/* Use the new setup routines */
/* Order: A/D
          Counter Chip
          DMA controller          */

                ad_setup();
                clk_setup_2 (MULT, count, trig_count, trig_skip);
                dma_stat = dma_setup(p_var1d, byt_count);
                i = nrep;
/*              printf("i = %d\n", i);*/
                time(&start_time);

/* Use clock status as clue to whether or not have collected a scan */
/* Found I need to set up two do...while loops to do this correctly */

                out_1 = outp (PORTCLK, 0x3c);

                while(i > 0){
```

```
                do{
                        clk_stat = inp(PORTCLK);
                }
                while ((clk_stat & 0x20)== 0);
                do{
                        clk_stat = inp(PORTCLK);
                }
                while ((clk_stat & 0x20)!= 0);

                out_1 = outp (PORTMUX, ADMUX_1);
                i --;

        } /* end the sampling loop   */

/* Make sure have last sample */

        do {
                clk_stat = inp (PORTCLK);
        }
        while ((clk_stat & 0x20) != 0);


        time(&end_time);

/* get the time here too , and also get it when prosthesis is unplugged */
        printf("Unplug the prosthesis and type C: ");

        do{
                scanf("%s",cwait);
        }
        while( strcmp(cwait,'c' )== 0);

        time(&off_time);


/* disarm counters  */
```

```
                out_1 = outp (PORTCLK, 0xDF);


/* Display the data */
                printf("Do you want to display the data?");
                scanf("%s",reply);
                if (reply[0] == 'Y' || reply[0] == 'y'){
                        press_display(p_var1d, nrep, count);
                        }

/* Here ... Need to write the data to a file, and enter the
        file name in a data directory */

                printf("Do you want to save this data?  [D:N] ");
                scanf("%s",reply);

                if (reply[0] == 'Y' || reply[0] == 'y'){

/* Need to add eff_freq (effective sampling rate) to the output file */
                        nrep = nrep +1;
                        make_long(p_var1d, nrep, count, MULT,\
                                stime, eff_freq, trig_skip, trig_count,\
                                pow_time, start_time, end_time, off_time);
                        }

                contin[0]='N';
                printf("Do you want to take more data?  [D:N] ");
                scanf("%s",contin);
        }       /* End the outside loop ... while(contin ...   */


}
```

CLK.H

```
/* This file contains definitions for the counter
          timer chip on the RTI-815 board.  */


#define GLOBAL  /* For referenceing a common quantity */
#define SEMIGLOBAL static
#define IMPORT extern

#define AD_BASE  0x350  /* Base address of A/D board */

#define PORTSB   0x350
#define COM_CLK  0x35D
#define DAT_CLK  0x35C

/* Commands to enter for counters 4 and 5 (Put in header file later) */
#define DISARM_45       0xD8     /* Disarm counters 4 & 5 */
#define DP_4_MODE       0x04     /* Data pointer to 4, mode register */
#define DP_5_MODE       0x05     /* Data ptr to 5, mode reg */
#define LOW_MODE_4      0x21     /* Low byte of counter 4 mode (MODE E) */
#define HI_MODE_4       0x8b    /* High byte of counter 4 mode (MODE E)*/
#define LOAD_4          0x48     /* Load counter 4 with load reg */
#define LOW_TOGGLE_4  0xE4  /* Clear toggle out for counter 4 */
#define LOW_MODEJ_5   0x62   /* Low byte - MODE J (repetitive counts)*/
#define HI_MODEJ_5      0x00
#define LOW_MODEL_5   0x62   /* Low byte of MODE L ... reloads */
#define HI_MODEL_5      0xF5     /* Hi byte of MODE L ... same as in SCAN */

#define LOW_LLOAD      0x01     /* Loaded into Load reg of 5 in MODE L */
#define HI_LLOAD       0x00

#define LOW_LHOLD      0x01     /* To load in hold register of 5 in MODE L*/
#define HI_LHOLD       0x00

#define LOAD_5          0x50     /* Load counter 5 with load reg */
#define HI_TOGGLE_5    0xED     /* Set toggle out high for 5 */
#define ARM_5           0x30     /* ARM counter 5 */
```

```
#define STEP_5          0xF5    /* Step counter 5 */
#define LOW_TOGGLE_5  0xE5  /* Clear toggle out for counter 5 */


/* Commands for counters 1 & 2; Used in clk4setup.c--for taking data slower */

#define DP_1_MODE       0x01    /* Set data pointer to mode of cntr 1*/
#define LOW_MODE_1      0x25    /* Mode E for counter 1; active low TC pulse*/
#define HI_MODE_1       0x81    /* Gate w/active high; Source is src 1 */
#define LOAD_1          0x41    /* Load  counter 1 ( use LLOAD defnd above) */

#define DP_2_MODE       0x02    /* Set data pointer to mode of cntr 2 */
#define LOW_MODEJ_2 0x62    /* Use Mode J w/ TC toggled */
#define HI_MODEJ_2      0x01    /* Source 2 is same as 1 (trigger/strobe) */
#define LOAD_2          0x42    /* Load counter 2, calc quantities in prog */
#define ARM_2           0x22    /* Arm 2 so can count it down */
#define LOW_TOGGLE_2 0xE2   /* Start 2 out low */
#define STEP_2          0xF2    /* Count down in program */


#define ARM_1245        0x3F    /* Arm all counters being used */
#define DISARM_1245     0xDB    /* Disarm all counters being used */
```

PIO.H

```
/*  pio.h ...      Contains some definitions used in setting
                   up the RTI-815A, the DMA controller on the
                   AT&T 6300+ and the Counter/Timer chip (AM9513)
                   on the A/D board for data acquisition of
                   pressure data.                  */


#define GLOBAL
#define SEMIGLOBAL static  /* used to define a semiglobal variable*/
#define IMPORT extern  /* used to refer to a global or semiglobal var*/
```

```
                    /* referenced elsewhere */

/* Some definitions specific to A/D board, Pressure I/O */
#define D_ARRSIZE  21752  /* As big as possible  - 21752*/
#define P43_FREQ   8000
#define P33_FREQ   4000

#define TOTCHAN     8  /* Currently RTI board configured for 8 */
                       /*   differential channels */
#define MULT  25       /* Time in microseconds between A/D channel samples */
#define PORTSB   0x350 /* Base address of RTI board - Status byte */
#define PORTCF   0x359 /* Clear flags byte */
#define PORTCLK  0x35d /* Clock control/status byte */
#define PORTMUX  0x351  /* Mux and Gain byte */
#define PORTDMSK 0x0a   /* Mask byte -- DMA controller */
#define MUXTEST  0x80   /* Mux test  */

#define AD_BASE  0x350 /* Base address for RTI-815 */
#define ADMUX_1  0x80   /* Mux entry for DMA scan, gain of 1, start at 0*/
#define ADSTAT_1 0x08   /* Get status set to go */
#define F_CLEAR  0x19  /* Any write to BASE + 9 clears the flags */



          /* DMA controller Ports, see AT&T Device Drivers */

#define DMA_COMM 0x08  /* Status(R)/command(WR) byte */
#define DMA_MASK 0x0A  /* Single Mask */
#define DMA_MODE 0x0B  /* Mode byte */
#define DMA_CFF   0x0C  /* Clear flip/flop */
#define DMA_ADD1 0x02  /* Channel 1 address(WR)/current address(R) */
                       /*  16 -bits */
#define DMA_CNT1 0x03  /* Channel 1 # of bytes to transfer */
                       /*  16 -bits        */
#define DMA_PG1  0x83  /* Channel 1 page (highest nibble of 20bit address)*/
```

AD_SETUP.C

```
/* This file contains a subroutine for setting up
   the RTI-815 for DMA transfers, gain of 1, starting at channel
   0.  The procedure follows that done in SCAN (from ADI)
   prior to setting up the clock and DMA controller          */

#include <stdio.h>
#include <dos.h>
#include <memory.h>
#include <conio.h>
#include "pio.h"

void ad_setup(void){

        int out_1;

        out_1 = outp (PORTCF, F_CLEAR);
        out_1 = outp (PORTSB, ADSTAT_1);
        out_1 = outp (PORTMUX, ADMUX_1);

}
```

DMASETUP.C

```
/* This file contains setup routines for the DMA
   controller for use with the RTI-815 board */
/* See AT&T Device Drivers section of System Programmer's Guide */
/* For the AT&T 6300+   */

#include <stdio.h>
#include <dos.h>
#include <memory.h>
#include <conio.h>
#include "pio.h"
```

```
int dma_setup (int *p_dat,
               int tot_byt_count){

     int dma_st;
     int out_1;
     int d_page,d_addr;
     int low_d_addr, hi_d_addr;
     int lo_byt_cnt, hi_byt_cnt;
     long long_addr;

/* Start by selecting channel, turning things off */

     out_1 = outp (DMA_MASK, 0x05);

/* Set the mode ... Don't change, use what was used in SCAN */
/* This is single transfer, increment address, no autoinit, */
/* write , using channel 1,                                  */

     out_1 = outp (DMA_MODE, 0x45);
     out_1 = outp (DMA_CFF, 0x45);

/* Now want to get the address set and enter it */

     long_addr = FP_SEG(p_dat);
     long_addr = long_addr << 4;
     long_addr += FP_OFF(p_dat);
     d_page = long_addr >> 16;
     d_addr = long_addr & 0x0000FFFF;

     low_d_addr = d_addr & 0x00ff;
     hi_d_addr = d_addr & 0xFF00;
     hi_d_addr = hi_d_addr >> 8;

     out_1 = outp (DMA_ADD1, low_d_addr);
```

```
        out_1 = outp (DMA_ADD1, hi_d_addr);
        out_1 = outp (DMA_PG1, d_page);
        out_1 = inpw (DMA_ADD1);
        out_1 = inp (DMA_PG1);

/* Next thing, Enter the number of bytes to transfer ( -1) */
/* Version A ... Enter the total number of bytes to transfer */

        tot_byt_count --;
        lo_byt_cnt = tot_byt_count & 0x00ff;
        hi_byt_cnt = tot_byt_count & 0xFF00;
        hi_byt_cnt = hi_byt_cnt >> 8;
        out_1 = outp (DMA_CNT1, lo_byt_cnt);
        out_1 = outp (DMA_CNT1, hi_byt_cnt);

/* Now, get status to send back, and enable transfers on chan 1 */

        dma_st = inp(DMA_COMM);
        out_1 = outp (DMA_MASK, 0x01);

        return(dma_st);
}
```

CLKS.C

```
/* This file contains setup routines for the counter
        timer chip on the RTI-815 board.
        This version allows longer data sets to be collected.
        This version uses counter 3 as well as 4 & 5 */

#include <stdio.h>
#include <dos.h>
#include <memory.h>
#include <conio.h>
#include "clk.h"
```

```
#define DISARM_ALL 0xdf
#define LOW_MODA_3 0x02
#define HI_MODA_3  0x00
#define DP_3_MODE 0x03
#define LOW_MODEE_3 0x22
#define HI_MODEE_3  0x8b
#define LOAD_3 0x44
#define LOW_TOGGLE_3 0xe3
#define ARM_3   0x24
#define HI_TOGGLE_3  0xeb
#define STEP_3   0xf3
/* This subroutine sets up the counter/timer chip
        (an AM9513A) to do scans of length specified by user
        Assumes that INIT was called prior to
        calling this routine.

   This version sometimes ignores triggers -- lower sampling rate
        but can collect longer data sets.

        See data sheet for information                  */

void clk_setup_2 (int interval,
                    int scan_count,
                    int trig_count,
                    int trig_skip){

        int out_1;
        int i;
        int low_interval,hi_interval;
        int low_sc_count,hi_sc_count;
        int low_trig_count,hi_trig_count;
        int low_trig_skip,hi_trig_skip;

/* Enter the commands  -- First disarm all counters */
```

```
        out_1 = outp (COM_CLK, DISARM_ALL);

/* Set up so can do either trigio or longio mode 12/4/91 */
/* This is new, (10/16/91) Hi toggle 3 so compatible with
        longio and circuitry.  May need to disarm 3 too */

        if (trig_skip == 0){
                out_1 = outp (COM_CLK, DP_3_MODE);
                out_1 = outp (DAT_CLK, LOW_MODA_3);
                out_1 = outp (DAT_CLK, HI_MODA_3);

                out_1 = outp (COM_CLK, HI_TOGGLE_3);
                }
        else {
/* Set up 3 to take place of cnter 1*/
                out_1 = outp (COM_CLK, DP_3_MODE);
                out_1 = outp (DAT_CLK, 0x62);
                out_1 = outp (DAT_CLK, 0x13);

/* I think want trig_skip in LOAD ( the one that gets counted down first ) */
/* and tig_count in HOLD.  Could try both ways */

                low_trig_skip = trig_skip & 0x00ff;
                hi_trig_skip = trig_skip & 0xFF00;
                hi_trig_skip = hi_trig_skip >> 8;
                out_1 = outp (DAT_CLK, low_trig_skip);/*Load counter 2 LOAD*/
                out_1 = outp (DAT_CLK, hi_trig_skip);

                low_trig_count = trig_count & 0x00ff;
                hi_trig_count = trig_count & 0xFF00;
                hi_trig_count = hi_trig_count >> 8;
                out_1 = outp (DAT_CLK, low_trig_count);/*Load counter 2 HOLD*/
                out_1 = outp (DAT_CLK, hi_trig_count);

                out_1 = outp (COM_CLK, LOAD_3);
                out_1 = outp (COM_CLK, LOW_TOGGLE_3);
        }

/* Don't count down this counter, used to, but clearly doesn't
        work.... looked at scope output from circuit */

/* Set up counter 4 to countdown continuously over interval */

        out_1 = outp (COM_CLK, DP_4_MODE);
        out_1 = outp (DAT_CLK, LOW_MODE_4);
        out_1 = outp (DAT_CLK, HI_MODE_4);

/* need to split up interval into high and low bytes */

        low_interval = interval & 0x00ff;
        hi_interval = interval & 0xFF00;
        hi_interval = hi_interval >> 8;
        out_1 = outp (DAT_CLK, low_interval);
        out_1 = outp (DAT_CLK, hi_interval);

        out_1 = outp (COM_CLK, LOAD_4);
        out_1 = outp (COM_CLK, LOW_TOGGLE_4);


/* Set up counter 5 to countdown over scan_count */

        out_1 = outp (COM_CLK, DP_5_MODE);

        out_1 = outp (DAT_CLK, 0x62);/* Mode L, alternates */
        out_1 = outp (DAT_CLK, 0xf5);/* reloads from HOLD and */
                                     /* LOAD regs           */


        low_sc_count = scan_count & 0x00ff;
        low_sc_count --;          /* Need to put number of channels-1*/
        hi_sc_count = scan_count & 0xFF00;
        hi_sc_count = hi_sc_count >> 8;
        out_1 = outp (DAT_CLK, low_sc_count); /* Load cntr 5 LOAD */
        out_1 = outp (DAT_CLK, hi_sc_count);
        out_1 = outp (DAT_CLK, LOW_LHOLD); /* Load counter 5 HOLD */
        out_1 = outp (DAT_CLK, HI_LHOLD);



        out_1 = outp (COM_CLK, LOAD_5);
        out_1 = outp (COM_CLK, ARM_5);
        out_1 = outp (COM_CLK, HI_TOGGLE_5);
                                /*Step through Load register*/
                                /*then toggle output low */
        for (i=0; i<scan_count;i++){
                out_1 = outp (COM_CLK, STEP_5);
                }

/* Counter 5 armed and ready to go, counter 4 will be armed
        when everything (DMA) is set up */

}
```

E.2 Programs for initial processing of data from Prosthesis 43

LONGDAT2.C

```
/* Data reading and looking at program */
/* For use with data collected using "longio" */
/* Jan 27 1992 */

#include <stdio.h>
#include <string.h>
```

```
#include "p_file.h"



int datarray [21752];
/* Should be 32767 */
/*Currently not multidimensioned because: number of A/D channels
 changes and can restrict the size of the data set */

/* Make datarray a multi-dimensional array: sample, a/d channel
*          and output channel as dimensions */
/* Set dimensions to maximum size  ( Set for #33 -- 16 channels )*/
/*#define SAMPLES
#define PROSCHAN 16
#define ADCHAN  4

int datarray [SAMPLES] [PROSCHAN] [ADCHAN]; */




FILE *get_outf(struct datainfo *);
int *read_lfi(struct datainfo *);
void convcut(int *, FILE *, struct datainfo *);

void main(void)
{
          char reply[3];
          int *p_arr;
          struct datainfo *p_info;
          struct datainfo thisdata;
          FILE *p_out;

          reply[0]='N';

/* Associate pointer with data information structure */
```

```
        p_info = &thisdata;
/* Allow to loop through */

          do {

/* Load the data into an array */
                    p_arr = read_lfi(p_info);

/* Put means, brief information into output file */
                    p_out = get_outf(p_info);
                    convcut(p_arr, p_out, p_info);
                    fclose(p_out);

/* Ask if want to do another file */
                    printf("Process another data file?  ");
                    scanf("%s",reply);
          }
          while (reply [0] == 'Y' || reply [0] == 'y');
          exit(0);
}

FILE *get_outf(struct datainfo *p_spec){

          char filename [NAMESIZE];
          FILE *p_out;

/* Information that need to get from user:
*         Name of File to put data into          */

          printf("Enter the output file name:  ");
          scanf("%s",filename);

/* Start filling datinfo */
          p_out = fopen(filename,"w");

          r [illegible]n(p_out);
```

```
}
```

CONVCUT2.C

```
/* Converts raw data to SORTED ATOD VALUES. Link to read_lfi.c, get43beg.c,*/
/* alarmcut.c, longdata.c */
/* DECEMBER 9, 1991*/

/*Modified in January 1992 so that temperature slope not used.*/


#include <stdio.h>
#include <math.h>

#define D_ARRSIZE 32767 /* Should be 32767 */
#define ATODVOLTS 204.8 /* Can I use a float? This is a/d units per volt */
#define ATODMV 0.2048 /*This is a/d units per millivolt */
#define FRAMESYNC -5.29 /* This is value for frame synch */
#define PCHAN 16        /* The number of output channels from prosthesis */
#define PCHAN43 18      /* Number of separate output quantities from 43 */
#define CALEVEL 165 /* nominal value for accepting cal and zero */
#define CALEVEL2 215
#define VTOMV 1000.0 /* convert to mv */
#define MINIT 10000.0

#define ALARMCHEK 0xBFFF
#define JUMPCHEK 0xDFFF
#define DISCARDHI 0x8FFF
#define SET_HI 0x7FF
#define JUMPX 0xDFFF

struct datainfo {
        char name[9];
        char id[3];
        int samp;
        int adchan;
        int mult;
        int trig;
        char time[5][12];
        char descr[32];
        int startoff;
        int frames;
        int alarmch;
        int synchch;
        double freq;
        };

/* This program is for Prosthesis 43 only. */

/* external code */
void alarmcut(int *, FILE *, struct datainfo *);

void get43_begin(int *, FILE *, struct datainfo *);
int *tempflux(int *, FILE *, float *, double, int, int, int, int);
double linefit(int *, int, int);

void convcut(int *p_arr, FILE *p_out, struct datainfo *p_spec){

        char blah;
        int chan,atodchan,scans,frames,adjust,jumpedvalue;
        int framesize,cal,zero,startoff,oldstartoff,framepair;
        int i,h,n,alarms,j,alarmch,synchch,diff,nframe,data,l;
        int lot[32][3], *begin_file,level;
        double frequency;
        float datum,datum2;
        double t14zero,t14slope,*f_t14;
        float zerovalue[14];
        FILE *zerofile;
        char zeroname[14];
/* Get information on channel to start on from get43_begin */
```

```
        get43_begin(p_arr, p_out, p_spec);
        atodchan =p_spec->adchan;
        scans = p_spec->samp;
        startoff = p_spec->startoff;
        begin_file=p_arr;
        framepair=scans/32;
        jumpedvalue=0;
        frequency = (p_spec->freq)/16;/*This is now the transducer frequency*/

/*NOW START AT STARTOFF IN ORDER TO RESORT AND CHECK DATA FOR ALARMS*/
        oldstartoff=startoff;

/*Loop through all sets of 32 values (framepairs) and reshuffle*/
        for(n=0; n<framepair; n++){

          for(i=0; startoff<(atodchan*32) && i<32; i++){
                for (h=0;h<atodchan;h++)
                        lot[i][h] = *(p_arr+h+startoff);
                /*LOOP BACK TO FIRST VALUE AT END OF FRAMEPAIR*/
                if(startoff==(atodchan*31)&&i<31) startoff-=atodchan*31;
                /*OR GO TO NEXT SET OF ATODCHANNELS*/
                else startoff+=atodchan;
                /*AND CHECK FOR BEGINNING OF LOOP*/
                if(startoff==oldstartoff) i=32;
                }



          /*set startoff back to its original value*/
          startoff = oldstartoff;

          /*replace framepair with newly shuffled framepair*/
          for(i=0;i<32;i++){
                      for(h=0;h<atodchan;h++)
                              *(p_arr+h+(i*atodchan))=lot[i][h];
                      }
```

```
            p_arr+=atodchan*32;   /*put p_arr at beginning of new framepair*/
            }

/*MUST NOW RUN THROUGH THE ALARMCUT PROGRAM BEFORE
CONTINUING CONVERSION*/
        alarmcut (begin_file, p_out, p_spec);
        atodchan-=1; /*alarmcut doesn't return the alarm values*/

/*next step is to get a linefit for the thermistor data*/
        t14zero = linefit(begin_file, framepair, atodchan);

/*set p_arr back to beginning of data*/
        p_arr = begin_file;

/*Make header for output file */

        fprintf(p_out,"%s %s\n",p_spec->name, p_spec->id);
        fprintf(p_out,"Frequency of transducers:  %.31f\n",frequency);
        fprintf(p_out,"frames:  %d\n",framepair);
        fprintf(p_out,"Channels:  %d\n",atodchan);

/*Function for getting zeros from resting data set*/

        printf("\nEnter the name of the zerofile for this day's data");
        printf("\nor 'n' if no zero values are to be used:");
        scanf("%s",zeroname);
        if(zeroname[0]=='n'){
                fprintf(p_out,"Zeros not used.\n");
                for(i=0;i<13;i++)
                        zerovalue[i]=0.0;
                        }
        else {
                zerofile=fopen(zeroname,"r");
                fprintf(p_out,"Zeros were used.\n");
                for(i=0;i<13;i++)
```

```
                    fscanf(zerofile,"%f",&zerovalue[i]);
            fclose(zerofile);
            }

/*start loop of framepairs (now with 18 values after alarmcut)*/
        for (nframe=0; nframe<framepair; nframe++){



/*at beginning of framepair, set cal, zero and level for the transducer */
/*data which is in that framepair.                                     */
            p_arr += atodchan; /*move to CAL position*/
            cal = *p_arr;
/*check limits for cal */
            if(cal<400||cal>600){
                    for(i=2;i<framepair-nframe;i++){
        if(*(p_arr+(18*i*atodchan))>400&&*(p_arr+(18*i*atodchan))<600){
                                    cal=*(p_arr+(18*i*atodchan));
                                    i=framepair-nframe;
                                    }
                            }
                    }
            p_arr += atodchan; /*move to ZERO position*/
            zero = *p_arr;
            zero = zero & DISCARDHI;
            for (l=0;l<3;l++){
                    zero = zero | ((zero & ~SET_HI) << 1);
                    }
            p_arr +=(15*atodchan); /*move to LEVEL position*/
            level = *p_arr;
/*check limits for level */
            if(level<700||level>1100){
                    for(i=2;i<framepair-nframe;i++){
        if(*(p_arr+(18*i*atodchan))>700&&*(p_arr+(18*i*atodchan))<1100){
                                    level=*(p_arr+(18*i*atodchan));
                                    i=framepair-nframe;
                                    }
```

```
                                        }
                                }

                        p_arr -= (17*atodchan); /*move back to beginning(T)*/

/*Begin with T, cal and zero before doing transducer loop*/
                        adjust = cal - zero;

        /*if alarmcharacter on T, then use next (or last) T */
                        data = *p_arr;
                        if ( (data & ~ALARMCHEK) !=0){
                                for(i=1;i<framepair-nframe;i++){
                                        if((*(p_arr+(18*i*atodchan))& ~ALARMCHEK)!=0){
                                                *p_arr=*(p_arr+(18*i*atodchan));
                                                i=framepair;
                                                }
                                        data = *p_arr;
                                        }
                                /*case where there is no more good data*/
                                if(i==framepair-nframe) nframe=framepair;
                                }
/*if JUMPED, then disregard because of slowness of temperature change*/
                        if ((data & ~JUMPCHEK)!=0){
                                data = data & DISCARDHI;
                                for (l=0;l<3;l++){
                                        data = data | ((data & ~SET_HI) << 1);
                                        }
                                }

        /*check for compression*/
                        datum = data;
                        if(datum > level){
                                        diff = datum - level;
                                        datum = level + 4*diff;
                                        }
                        datum2 = VTOMV*((float)(datum-zero))/ ((float) adjust);
```

```
/*convert datum2 to fahrenheit*/
                    datum = sqrt((datum2+2861.0)/0.3391);


/*write datum to file*/
                    fprintf(p_out," %.2f",datum);
/*write cal and zero to file*/
                    fprintf(p_out," %.2f %.2f",(float)cal,(float)zero);


/*position p_arr at 1st transducer*/
                    p_arr+=(3*atodchan);


/*Begin loop for transducers 1 through 13*/
/*Do compression in tempflux before temperature adjustment*/
p_arr=tempflux(p_arr,p_out,zerovalue,t14zero,atodchan,cal,zero,level);


/*write power and level to file*/
                    fprintf(p_out," %.2f %.2f",(float)*p_arr,(float)level);
                    p_arr+=(2*atodchan);  /*place at start of new framepair*/
/*WRITE AUXMUX DATA TO FILE AT THIS POINT  IF ATODCHAN=2*/
                    if(atodchan==2){
                              p_arr-=(18*atodchan);  /*position at T*/
                              p_arr++;   /*increment to 1st auxmux value: aux 10*/
                              for(i=0;i<16;i++){
                                        fprintf(p_out," %d",*p_arr);
                                        p_arr+=atodchan;
                                        }
                              p_arr +=(atodchan +1);   /* Position at T */
                              }
                    fprintf(p_out,"\n");

          } /*end of framepair loop*/
}
```

```
/*Linefit is used on channel 1 (thermistor) in order to do the */
/* temperature calibration on the other transducers   */

double linefit( int *y_arr, int samples, int atodchan){
        int i,l,data;
        double sum,sumy;



        sum = (double) samples;
        sumy = 0;

/*loop through samples and total them up*/
        for(i=0; i<samples; i++){
                /*if no ALARMED character*/
                if ((*y_arr & ~ALARMCHEK) == 0){
                        /*if there is a JUMPED value */
                        if ((*y_arr & ~JUMPCHEK) != 0){
                                printf ("y_arr = %d",*y_arr);
                                /* Get correct values for y_arr */
                                data = *y_arr;
                                data = data & DISCARDHI;
                                printf ("data = %d",data);
                                for (l=0;l<3;l++){
                                   data=(data | ((data & ~SET_HI)<<1));
                                   }
                                printf ("data = %d\n",data);
                                sumy += (double)data;
                                y_arr+=(18*atodchan);
                                }
                        /*case for normal values*/
                        else {
                                sumy += (double)*y_arr;
                                y_arr+=(18*atodchan);
                                }
                        }
                }
```

```
/*calculate the mean */
                return(sumy/sum);
}

/*This is the temperature flux function, used for all channels */
/*except 1(transducer 14), 2, 3(CAL, Zero, power, level) and 7 */
/*(transducer 4 not working properly) */

int *tempflux(int *p_arr, FILE *p_out, float *zerovalue, double t14zero,
        int atodchan, int cal, int zero, int level)
{
        int i,diffjumpedvalue,l, data;
        float datum;               /*intermediate variable in data conversion*/
        float b,m;            /*intercept and slope for data conversion*/
        float newvalue;
        float c[14],d[14],e[14];

/*assign values to arrays according to equations for each transducer. */
        c[1] = −.0308 ; d[1] = −153 ; e[1] = 1.22 ;
        c[2] = −.275 ; d[2] = 1009 ; e[2] = 0.987 ;
        c[3] = −.412 ; d[3] = 22 ; e[3] = .678 ;
        c[4] = 0; d[4] = 0 ; e[4]=1;  /*transducer 4 not done*/
        c[5] = −.150 ; d[5] = 555 ; e[5] = .976 ;
        c[6] = .201 ; d[6] = 359 ; e[6] = 1.23 ;
        c[7] = −.113 ; d[7] = 150 ; e[7] = .960 ;
        c[8] = −.0564 ; d[8] = 251 ; e[8] = .870 ;
        c[9] = −.2899 ; d[9] = 314 ; e[9] = .795 ;
        c[10] = −.4408 ; d[10] = 695 ; e[10] = .885 ;
        c[11] = −.177 ; d[11] = −159 ; e[11] = .814 ;
        c[12] = −.0815 ; d[12] = 1068 ; e[12] = .839 ;
        c[13] = −.0722 ; d[13] = −175 ; e[13] = .888;


/*now loop through transducers and calculate new pressure values according*/
/* to the thermistor readings */
```

```
jumpedvalue=0;
for(i = 1; i < 14; i++){
        data = *p_arr;
        /*check for ALARMED value of data*/
        if((data & ~ALARMCHEK)==0) {
                /*check for JUMPED value of data*/
                if((data & ~JUMPCHEK)!=0){
                        jumpedvalue = 10000;
                        }
                data = data & DISCARDHI;
                for (l=0;l<3;l++){
                        data = data | ((data & ~SET_HI) << 1);
                        }
          /*    }       */
                /*check for compression*/
                datum=(double) data;
                if(data > level){
                        diff = data-level;
                        datum = level + 4*diff;
                        }

        newvalue = VTOMV*((float)(datum-zero))/ ((float)(cal-zero));

                /*now convert by temperature*/
                b = c[i]*t14zero + d[i];
                m = e[i];
                datum = (newvalue - b) / m;
                newvalue = datum - *zerovalue;
        /*now write the newvalue to file*/
                if (jumpedvalue !=0){
                        fprintf(p_out," *%.2f",newvalue);
                        }
                else {
                        fprintf(p_out," %.2f",newvalue);
                        }
                p_arr+=atodchan;
```

```
                        jumpedvalue = 0;
                        }
                else {
                        fprintf(p_out," @%f", (float) data);
                        p_arr += atodchan;
                        }
                zerovalue++;  /*increment zerovalue for next transducer*/
                }
        return(p_arr); /*p_arr now points to P in the framepair*/
}
```

```
/* Function to check alarms and pare down data acquired */
/* using longio */
/* Dec. 10, 1991, rewritten Jan 27, 1992 */
/*                                        */
/* Changed Dec. 15, 1992 to deal with change in alarms */
/* that first became apparent Nov. 13, 1992.  Alarm    */
/* level during no alarms is now: 63 - 76 or so in the */
/* high field and -1000 to -965 or so in the low field */
/* Only change to program is to change the value of    */
/* "ignore" to 0x13F.  See Prosth notebook #5 near the */
/* end, under data collection 12/14/92 for calcs.      */
/*    K.C.                                  */



#include <stdio.h>
#include <math.h>
#include <p_file.h>

#define D_ARRSIZE 21752
#define ATODVOLTS 204.8
#define FRAMESYNC -5.29
#define PCHAN33 16
#define PCHAN43 18

#define ALARMED 20000
#define JUMPED 10000
#define ALARMHI  -4.9*ATODVOLTS
#define ALARMLOW -5.3*ATODVOLTS

#define DISCARDHI 0x8FFF
#define JUMPX 0xDFFF
#define ALARMX 0xBFFF

void alarmcut (int *p_arr, FILE *p_out, struct datainfo *p_spec ){
        int *p_wrt,*p_strt;
        int level, field, levelo, leveln;
        int fram,chan,atodchan, scans, frames, tframes;
        int data, startoff;
        int i, alarms, j, alarmch;
        int jump,jumped,alarmed;
        int ignore = 0x13F;     /* Changed Dec. 15, 1992 from 0x107 */

        /* Get information from datinfo */
        atodchan = p_spec->adchan;
        scans = p_spec->samp;
        alarmch = p_spec->alarmch;

/* p_arr is located at first T right now.  Want to save it. */
/* Also, want to write back to same place */

        frames = p_spec->frames;
        tframes = scans/32;
        jumped = 0;
        alarmed = 0;
        p_wrt = p_arr;
        p_strt = p_arr;

/*      for (i=0;i<64;i++){
                printf("%d ",*p_arr);
                p_arr++;
                }
        printf("\n");
*/
        p_arr = p_strt;

        for (fram=1;fram<=tframes;fram++){
                field = 2;

/* First deal with T */
                if (alarmch !=-1){
                    p_arr +=alarmch;
                    alarms = *p_arr;
                    alarms = alarms-63;
                    if ((abs (alarms) & ~ignore) > 0){
                            jump = atodchan*PCHAN33;
                            p_arr+=jump;
                            alarms = *p_arr;
                            alarms += 5.29*ATODVOLTS;
                            alarms = alarms-63;
                            jumped = 1;
                            if ((abs(alarms)& ~ignore) > 0){
                                    p_arr-=jump;
                                    alarmed = 1;
                                    jumped = 0;
                                    alarms = *p_arr;
                                    }
                            }
                    p_arr-=alarmch;
                    }
        /* Now write back T to array */
                data = *p_arr;
                data = data & DISCARDHI;
                if (jumped){
/*                      data = data+JUMPED;*/
                        data = data | ~JUMPX;
                        p_arr -= jump;
                        jumped = 0;
                        }
                if (alarmed){
                        data = data | ~ALARMX;
                        alarmed = 0;
                        }
                *p_wrt = data;
                p_arr+=2;
                p_wrt++;
        /* Write the aux-mux channel, any others */
```

```
            for (i=2;i<atodchan;i++){
                    data = *p_arr;
                    *p_wrt = data;
                    p_arr++;
                    p_wrt++;
                    }
/*      Next is CAL and ZERO just write to array */
            for (j=0;j<2;j++){
                    data = *p_arr;
                    data = data & DISCARDHI;
                    *p_wrt = data;
                    p_arr+=2;
                    p_wrt++;
                    for (i=2;i<atodchan;i++){
                            data = *p_arr;
                            *p_wrt = data;
                            p_arr ++;
                            p_wrt ++;
                            }
                    }
/*      Now need to do the transducers, 1-13 */
            field = 1;
            for (j=0;j<(PCHAN33-3);j++){
                    if (alarmch != -1){
                            p_arr+=alarmch;
                            alarms = *p_arr;
                            alarms += 5.29*ATODVOLTS;
                            alarms = alarms - 63;
                            if ((abs(alarms) & ~ignore)>0){
                                    jump = atodchan*PCHAN33;
                                    p_arr+=jump;
                                    alarms = *p_arr;
                                    alarms = alarms - 63;
                                    jumped = 1;
                                    if ((abs(alarms) & ~ignore)>0){
                                            p_arr -=jump;
```

```
                                        alarms = *p_arr;
                                        jumped = 0;
                                        alarmed = 1;
                                        }
                                   }
                          p_arr -=alarmch;
                          }
          /* Now write back to array */
                     data = *p_arr;
                     data = data & DISCARDHI;
                     if (jumped){
     /*                   data += JUMPED;*/
                          data = data | ~JUMPX;
                          p_arr -=jump;
                          jumped = 0;
                          }
                     if (alarmed){
     /*                   data += ALARMED;*/
                          data = data | ~ALARMX;
                          alarmed = 0;
                          }
                     *p_wrt = data;
                     p_arr +=2;
                     p_wrt ++;
                     for (i=2;i<atodchan;i++){
                          data = *p_arr;
                          *p_wrt = data;
                          p_arr ++;
                          p_wrt ++;
                          }
                     } /* Closes the transducer loop */
                p_arr += atodchan; /* Skip the second T recording */
          /* Now write back P and L */
                for (j=0;j<2;j++){
                     data = *p_arr;
                     data = data & DISCARDHI;
```

```
                        *p_wrt = data;
                        p_arr +=2;
                        p_wrt ++;
                        for (i=2;i<atodchan;i++){
                                data = *p_arr;
                                *p_wrt = data;
                                p_arr ++;
                                p_wrt ++;
                                }
                        } /* Close the P and L loop */
                p_arr += (atodchan * (PCHAN33-3)); /* Position at T again */
                } /* Close the tframes loop */
        p_arr = p_strt;
/*      for (i=0;i<18;i++){
                printf("%d ",*p_arr);
                data = *p_arr;
                data = data & 0x8fff;
                for (j=0;j<3;j++){
                        data = data | ((data & ~0x07FF)<<1);
                        }
                printf("%d ",data);
                p_arr++;
                }
*/
        p_arr = p_strt;
        }
```

GET43BEG.C

```
/* Function to locate the start of output from
   a prosthesis      */

/* Jan. 15, 1991 original version*/
/* April 18, 1991. Tim changed the part where the program asks for the */
/* alarm channel. It is now automatic. GET_BEGIN.C was renamed GET_BEG2.C */
/* July 15, 1991. The part of the original program that worked on #43 was */
/* modified to work with the new pressure conversion program. The new */
/*version finds and returns the first transducer 14 occurence instead of */
/* CAL and Zero. */

#include <stdio.h>
#include <math.h>
#include <p_file.h>

#define ATODVOLTS 204.8 /* Can I use a float? This is a/d units per volt */
#define ATODMV 0.2048 /*This is a/d units per millivolt */
#define FRAMESYNC -5.32  /* This is value for frame synch */
#define PCHAN33 16     /* The number of output channels from prosthesis */
#define PCHAN43 18     /* Number of separate output quantities from 43 */
#define CALEVEL 180 /* nominal value for accepting cal and zero */
#define CALEVEL2 210
#define VTOMV 1000.0 /* convert to mv */
#define MINIT 10000.0
/*
struct datainfo {
        char name[9];
        char id[3];
        int samp;
        int adchan;
        int mult;
        int trig;
        char time[5][12];
        char descr[32];
        int startoff;
        int frames;
        int alarmch;
        int synchch;
        double freq;
        };

*/

/* This one is to get start of a file written from 43 */
/* Look for first occurence of CAL and Zero, tricky because */
/* They don't occur every frame, only every other */
/* For now, have user input which A\D channel alarms are on */

void get43_begin(int *p_arr, FILE *p_out, struct datainfo *p_spec)
{
        int atodchan, scans, alarms, startoff=-1, frames, i,j;
        int chanchek [32];
        int field, start, diff, new, old;
        int alarmch = -1;
        int synchch = -1;
        int alcomp = 0x0F;
        int fivevolt = 1024;

/* Get information on data from the structure thisdata */
        atodchan = p_spec->adchan;
        scans = p_spec->samp;

        alarmch = 1;                /*This is the automatic line */

/* read the first 32 values of alarmch into an array */
        p_arr += alarmch;
        for(i=0;i<32;i++){
                chanchek [i] = *p_arr;
                p_arr+=atodchan;
                }

/*Reset p_arr to datarray[0], actually to the 4th scan start */
```

```
        p_arr -=((32*atodchan)+ alarmch);

/*Now look at alarmch and use to determine which field are in*/
        field=0;
        start=0;
        do{
                diff = chanchek[start+1]-chanchek[start];
                if ((diff & ~alcomp) == 0){
                        if (chanchek[start] < 0) field = 1;
                        if (chanchek[start] > 0) field = 2;
                        }
                else {
                        start++;
                        }
                } while (field == 0);

/* Next step, is locate an edge of the field, doesn't change fast*/
/* Looking for a 5.3 V change */
        old = chanchek[start+1];
        i = 1;
        new = chanchek[start+1+i];
        while ((abs(new-old))< fivevolt ){
                i++;
                new = chanchek[start+1+i];
                }
        if (field == 2)     startoff = ((start + i -1 ) * atodchan);
        if (field == 1)     startoff = ((start + i + 15)* atodchan);

        printf("alarm ch %d synch ch %d \n",alarmch,synchch);


                /* startoff is the number of elements of datarray before */
                /* the first full frame starts (frame alignment defined  */
                /* as starting with 14th transducer and ending with 13th */
                /* transducer.)  NOTE that datarray elements are numbered */
                /* from zero, and that first 3 scans are skipped */
```

```
    frames = (scans - startoff/atodchan)/PCHAN33;

    if (startoff == -1) {
            printf("This data may not be good," );
            printf("or alarms may not be on a/d channel 2\n");
            }

    p_spec->startoff = startoff-atodchan;
    p_spec->frames = frames;
    p_spec->alarmch = alarmch;
    p_spec->synchch = synchch;
    printf ("startoff %d frames %d\n",startoff,frames);
    }
```

READ_LFI.C

```
/* Subroutine to read the data files from "pio" */
/* currently this is called from "datalook"*/
/* Jan 4 1990 */


#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <time.h>
#include <string.h>
#include <p_file.h>


FILE *get_file (struct datainfo *);
int *pile_data (struct datainfo *, FILE *);
int *read_lfi(struct datainfo *p_spec)
{
        FILE *p_data;
        int *p_arr;

/*  Get the file in another function */
        p_data = get_file(p_spec);

/*  Now read data into array */
        p_arr = pile_data(p_spec, p_data);

/*  Close the file */
        fclose(p_data);

        return(p_arr);
}


FILE *get_file(struct datainfo *p_spec){

        char dataname [NAMESIZE];
        FILE *p_dat;

/* Information that need to get from user:
        Name of File to put data into         */

/* Open the file to put stuff in */


        printf("Enter the complete name of file to look at:  ");
        scanf("%s",dataname);

/* Start filling datinfo */
        p_dat = fopen(dataname,"r");
        fscanf(p_dat,"%s %s",p_spec->name,p_spec->id);
        return(p_dat);
}


/* Read the data from the file into an array */

int *pile_data (struct datainfo *p_spec, FILE *p_data)
        {

        IMPORT int datarray[];
        int *p_arr;

        char num[7],data[11];
/*      char s[20],t[10],f[10],e[10],c[20];
        int con1,con2;
        float sec; */
        int dat_index;
        int chan_index;
        int i,j,chan,lsamp;

/* Want to get to skip first three scans */
        p_arr = &datarray[0];

/* First read sampling parameters */
/* The first several lines contain plenty of writing as well */

        fscanf(p_data,"%d %*s %*s %*s %*s %*s",&(p_spec->samp));
        fscanf(p_data,"%d %*s %*s",&(p_spec->adchan));

        fscanf(p_data,"%*d%*s%*s%*d%*s%*s");
        fscanf(p_data,"%*s%*s%*s%*s%*s%*f");
        fscanf(p_data,"%*s%*s%*s%lf",&(p_spec->freq));

        fscanf(p_data,"%*s %*s %*s %*s %d %*s",&(p_spec->mult));
        fscanf(p_data,"%*s %*s %*s %*s %*s %*s %*s%*s %*s");
        fscanf(p_data,"%*s %*s %s %s %s %s %s",p_spec->time[1],\
                p_spec->time[2],p_spec->time[3],p_spec->time[4],\
                p_spec->time[5]);
        fscanf(p_data,"%*s %*s %*s%*s%*s%*s%*s%*s%*s %*s%*s%*s%*s%*s");

/* May also want stuff like: triggering, delays,... */


        chan_index = p_spec->adchan;
        dat_index = p_spec->samp;
        printf("Chans %d   samples %d \n",chan_index,dat_index);
        printf("Frequency:   %lf\n",p_spec->freq);
/* Read data : Note that for some reason data was written as strings */
/* Also ... would be more convenient later if this were a */
/* multidimensional array */



/* Now start storing */


        for( i=0; i< dat_index; i++)
          {
                fscanf(p_data,"%*s");

                for (j=0;j< chan_index; j++)
                {
                        fscanf ( p_data,"%s",data);

/* here need to convert the string to a number and store in datarray */
                        sscanf ( data, "%d", p_arr);
                        p_arr++;
                }
          }
        p_arr = &datarray[0];



        return(p_arr);
}
```

AUX20_D.C

```
/* Jan 23, 1992 */
/*Program uses the .int files from longdat2 and makes easyplot files.*/
/*The current version works on the four foot switch channels, the */
/*goniometer channel, and the auxiliary 11 channel for the force cane.*/
/*If other channels are needed, the program will have to be modified.*/

/* Oct. 7, 1992 */
/* Program altered from auxmux20.c --- this version allows input of */
/* which channel to read force cane signal from and can handle */
/* synch signal from TRACK */
/* K.C. */
/* Dec. 15, 1992 */
/* Program changed to handle forceplate data from TRACK also. */
/* K.C. */
/* Should make major changes - change datarray to a structure */
/* and allocate memory for it using halloc */


#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <time.h>
#include <string.h>
#include <p_file.h>

#define ADFULLSCALE 2048


/* datinfo contains the information about the dataset */
struct auxinfo {
   char name[9];
```

```
     char id[3];
     double frequency;
     int frames;
     int channels;
     };
float datarray[330][43];

FILE *get_file (struct auxinfo *);
void pile_data (struct auxinfo *, FILE *);
float fsvalue(float, int);

main ()
{
    FILE *p_data;
    char response[10];
    struct auxinfo datinfo;
    struct auxinfo *p_spec;
    p_spec = &datinfo;
    response[0] = 'y';

    while (response[0] == 'y'){

/* Get the file in another function */
      p_data = get_file(p_spec);
      printf ("%d",p_data);

/* Now read data into array */
      pile_data(p_spec, p_data);

      printf ("Do another file?  ");
      scanf ("%s",response);
    }

}
```

```
FILE *get_file(struct auxinfo *p_spec)
{
    char dataname [NAMESIZE];
    FILE *p_dat;

/* Information that need to get from user:
*    Name of File to put data into */

/* Open the file to put stuff in */


    printf("Enter the name of a data file for sorting:  ");
    scanf("%s",dataname);

/* Start filling datinfo */
    p_dat = fopen(dataname,"r");
    fscanf(p_dat,"%s %s",p_spec->name,p_spec->id);

    return(p_dat);
}



/* Read the data needed from the file and write to new file */

void pile_data (struct auxinfo *p_spec, FILE *p_data)
{


    char temp_data[10];
    char *c1,*c2;
    char filename[NAMESIZE];
    FILE *p_out;
    int check = 0;
    int cane = 0;
    int fplates = 0;
    int fpright = 0;
    int fpleft = 0;
    int leftfoot = 0;
    int rightfoot = 0;
    int gon = 0;
    int tsync = 0;  /* TRACK synch signal */
    int tsyn_ch = 9;
    int cane_ch = 10;
    int fp1 = 10;
    int fp2 = 11;
    int fp1zero, fp2zero;
    int i,j,h;
    int chan;
    int aux_chans = 41;
    char fill[8];
    int x = 0;
    int goodsamp;
    extern float datarray[330][43];
    float datum = 0.0;
    float angle;
    float conversion = 0.0;
    float timeskip = 0.0;
    float data[5];
    float fpfullscale;
    float freq;

    fill[0]='b';fill[1]='a';fill[2]='d';fill[3]='d';fill[4]='a';
    fill[5]='t';fill[6]='a';fill[7]='\0';

    *c1='@';
    *c2='*';
    data[0]=0.0;

/* First read sampling parameters */
/* The first several lines contain plenty of writing as well */

    printf("%d",p_data);
    fscanf(p_data,"%*s %*s %*s %lf",&(p_spec->frequency));
    freq = (float)p_spec->frequency;
    fscanf(p_data,"%*s %d",&(p_spec->frames));
    goodsamp=p_spec->frames;
    fscanf(p_data,"%*s %d",&(p_spec->channels));
    fscanf(p_data,"%*s %*s %*s");
    printf ("%s %s\n",p_spec->name,p_spec->id);
    printf ("%f %d %d\n",freq,goodsamp,p_spec->channels);


/* Find out what channels were used in this data set */
if(p_spec->channels>1){
    printf ("\nEnter 1 if a TRACK sync signal was taken:");
    scanf ("%d",&tsync);
    if (tsync == 1){
      printf("Enter the channel TRACK sync signal was on:");
      scanf ("%d",&tsyn_ch);
      }

    printf("\nEnter 1 if force cane was used, 0 if not used:");
    scanf("%d",&cane);
    if (cane ==1){
      printf("Enter the channel for force input:");
      scanf("%d",&cane_ch);
      }

    printf ("\nEnter the number of forceplates used, 0 if not used:  ");
    scanf ("%d",&fplates);
    if (fplates == 1){
      printf("Enter the number of the forceplate used (1 or 2):  ");
      scanf("%d",&fpright);
      if (fpright == 2){
           fpright = 0;
           fpleft = 1;
       }
      printf("Enter the channel forceplate data is on:  ");
      scanf ("%d",&fp1);
      if (fpright == 0){
        fp2 = fp1;
        fp1 = 0;
      }
    printf("Enter the zero value for this forceplate ");
    scanf ("%d",&fp1zero);
    if (fpright == 0) {
        fp2zero = fp1zero;
        fp1zero = 0;
      }
    printf("Enter the full scale value in Newtons for the forceplates:  ");
    scanf ("%f",&fpfullscale);
    }
  if (fplates ==2){
    fpright = 1;
    fpleft = 1;
    printf("Enter the channel forceplate 1 (right) is on:  ");
    scanf ("%d",&fp1);
    printf("Enter the channel forceplate 2 (left) is on:  ");
    scanf ("%d", &fp2);
    printf("Enter the zero value for forceplate 1:  ");
    scanf ("%d",&fp1zero);
    printf("Enter the zero value for forceplate 2:  ");
    scanf ("%d",&fp2zero);
    printf("Enter the full scale value in Newtons for the forceplates:  ");
    scanf ("%f",&fpfullscale);
  }


  printf("\nEnter channel of left footswitch (1-4), or 0 \n");
  printf("if footswitches were not used:");
  scanf("%d",&leftfoot);

  if (leftfoot > 0 && leftfoot < 5)  /* IF FS ARE BEING USED */
   {
       printf("Enter channel of right footswitch (1-4):");
    scanf("%d",&rightfoot);
```

```
      }

printf("\nEnter 1 if goneometer was used in this data set, 0 if not used:");
   scanf("%d",&gon);

   }
/*Loop for however many framepairs */
   for (i=0; i<p_spec->frames; i++){

   /*Loop for 18 prosthesis values*/
     for (j=0; j<18; j++) {
       fscanf(p_data,"%s",temp_data);
/*          printf("%s  ",temp_data);  */
       check=strcmp(&temp_data[0],c1);
/*CONVERT TO MPA FROM PSI HERE*/
       sscanf(temp_data,"%f",&conversion);
       if(j>2&&j<16&&j!=6){
             datarray[i][(2*j)] = 0.006894 * conversion;
             }
       else {
          datarray[i][(2*j)] = conversion;
             }
/*        printf("%.2f\n",datarray[i][2*j]);  */
       if(check!=0){  /*case when at least one point was good*/
             check=strcmp(&temp_data[0],c2);
          if(check!=0){  /*case when 1st point was used*/
             datarray[i][(2*j)+1]=timeskip;
             }
          else{   /*case when second point was used*/
             datarray[i][(2*j)+1]=(timeskip+(1/(freq*2)));
                  }
          }
       else { /*case when data point was bad*/
             datarray[i][2*j]=-10000;
          datarray[i][(2*j)+1]=timeskip;
             }
```

```
        } /*end prosthesis loop*/
        timeskip+=1/freq;

/* Will need to do this differently, check if each channel is used */

      if(p_spec->channels==2){
        for (h=36;h<43;h++){
              datarray [i][h] = 0.0;
              }
        for (j=0;j<16;j++){
              chan = j+10;
              if (chan > 16) chan = chan - 16;

                /* Footswitches are auxmux 12 - 15, if used */

              if (chan == 12 && ( leftfoot != 0 || rightfoot != 0)){

         /*put next 4 values into an array. This simplifies the process.*/

            for(h = 1; h < 5; h++){
                  fscanf(p_data,"%f",&data[h]);
                  }
                  /*NEED TO INCORPORATE FSVALUE FUNCTION INTO THESE*/
                datarray[i][37] = fsvalue(data[leftfoot],leftfoot);
                  /*assigns 0.0 if leftfoot is 0*/
                datarray[i][38] = fsvalue(data[rightfoot],rightfoot);
            j += 3;  /* increment j to keep in order */
                }
             else if (chan == tsyn_ch && tsync == 1) {
            fscanf (p_data, "%f",&datum);
                datarray [i][40] = datum;
                }
             else if (chan == cane_ch && cane == 1){
            fscanf(p_data,"%f",&datum);
                datarray[i][36] = datum*0.110;
            /* THIS CONVERTS CANE DATA INTO lbf */

           }
         else if (chan == fp1 && fpright == 1){
           fscanf (p_data,"%f",&datum);
           datarray [i][41] = (datum - fp1zero) * fpfullscale/ADFULLSCALE;
                 }
         else if (chan == fp2 && fpleft == 1){
           fscanf (p_data, "%f",&datum);
           datarray [i][42] = (datum - fp2zero) * fpfullscale/ADFULLSCALE ;
           }
        else if (gon == 1 && chan == 16){
       fscanf(p_data,"%f",&angle);
       datarray[i][39] = 3.485 * angle - 1646.1;
       /* CONVERTS ANALOG UNITS TO DEG*/
          }
        else fscanf (p_data,"%*f"); /* Else, Throw this point away */
  /* END OF FOR LOOP FOR FRAMES OF 34 */
        }
   }
  } /* close the outside loop for all framepairs*/
  /* Close the file */
  fclose(p_data);

/* FIND OUT WHERE TO PUT DATA, THEN WRITE IN DATA INFORMATION */
  printf("\nEnter the name of the output file:");
  scanf("%s",filename);

  p_out = fopen(filename,"w"); /* OPEN OUTPUT FILE */

  fprintf(p_out,"%s %s\n",p_spec->name, p_spec->id);
  fprintf(p_out,"Frequency of transducers: %lf\n",p_spec->frequency);
  fprintf(p_out,"Samples: %d\n",p_spec->frames);

  fprintf(p_out,"\n//force_pd 9.95 7.5");
```

```
    fprintf(p_out,"\n//pos 0 .75 .2 .95");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/sm OFF");
    fprintf(p_out,"\n/et x 'time (sec)'");
/*  fprintf(p_out,"\n/et y 'temperature (F)'");*/
    fprintf(p_out,"\n/et g 'Thermistor Data'");
    fprintf(p_out,"\n/or y 97 107");
    fprintf(p_out,"\n/td yx....");
    for(h=0;h<goodsamp;h++){
      if(datarray[h][0]!=-10000)
          fprintf(p_out,"\n%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                  datarray[h][0],datarray[h][1],datarray[h][36],
                  datarray[h][37],datarray[h][38],datarray[h][39]);
      else
          fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
                  datarray[h][1],datarray[h][36],datarray[h][37],
                  datarray[h][38],datarray[h][39]);
      }
    fprintf(p_out,"\n//nc");
    fprintf(p_out,"\n\n");

    fprintf(p_out,"\n//pos .2 .75 .4 .95");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'cmv'");
    fprintf(p_out,"\n/et g 'Calibration Data'");
    fprintf(p_out,"\n/or y 400 600");
    fprintf(p_out,"\n/td yx....");
    for(h=0;h<goodsamp;h++){
      if(datarray[h][2]!=-10000)
          fprintf(p_out,"\n%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                  datarray[h][2],datarray[h][3],datarray[h][36],
                  datarray[h][37],datarray[h][38],datarray[h][39]);
      else
          fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
                  datarray[h][3],datarray[h][36],datarray[h][37],
                datarray[h][38],datarray[h][39]);
            }
  fprintf(p_out,"\n//nc\n\n");
  fprintf(p_out,"\n//pos .4 .75 .6 .95");
  fprintf(p_out,"\n/ag");
  fprintf(p_out,"\n/et x 'time (sec)'");
  fprintf(p_out,"\n/et y 'cmv'");
  fprintf(p_out,"\n/et g 'Zero Data'");
  fprintf(p_out,"\n/or y -50 50");
  fprintf(p_out,"\n/td yx....");
     for(h=0;h<goodsamp;h++){
     if(datarray[h][4]!=-10000)
         fprintf(p_out,"\n%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                datarray[h][4],datarray[h][5],datarray[h][36],
                datarray[h][37],datarray[h][38],datarray[h][39]);
     else
         fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
                datarray[h][5],datarray[h][36],datarray[h][37],
                datarray[h][38],datarray[h][39]);
            }
 fprintf(p_out,"\n//nc\n\n");
 fprintf(p_out,"\n\n");
 fprintf(p_out,"\n//pos 0 .5 .2 .7");
 fprintf(p_out,"\n/ag");
 fprintf(p_out,"\n/et x 'time (sec)'");
 fprintf(p_out,"\n/et y 'pressure (MPa)'");
 fprintf(p_out,"\n/et g 'transducer 1'");
 fprintf(p_out,"\n/or y -1 7");
 fprintf(p_out,"\n/td yx.......");
 for(h=0;h<goodsamp;h++){
 x=(2*i)+4;
 if(datarray[h][6]!=-10000)
 fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f "5.2f",
         datarray[h][6],datarray[h][7],datarray[h][36],
         datarray[h][37],datarray[h][38],datarray[h][39],
         datarray[h][40],datarray[h][41],datarray[h][42]);
```

```
   else
  fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          fill,datarray[h][x+1],datarray[h][36],datarray[h][37],
          datarray[h][38],datarray[h][39],datarray[h][40],
          datarray[h][41],datarray[h][42]);
    }
  fprintf(p_out,"\n//nc\n");
/*   fclose(p_out);
  printf("\nEnter the name of the 2nd output file:");
  scanf("%s",filename);

  p_out = fopen(filename,"w");   OPEN OUTPUT FILE */



  fprintf(p_out,"\n//pos .2 .5 .4 .7");
  fprintf(p_out,"\n/ag");
  fprintf(p_out,"\n/et x 'time (sec)'");
  fprintf(p_out,"\n/et y 'pressure (MPa)'");
  fprintf(p_out,"\n/et g 'transducer 2'");
  fprintf(p_out,"\n/sm OFF");
  fprintf(p_out,"\n/or y -1 7");
  fprintf(p_out,"\n/td yx.......");
  for(h=0;h<goodsamp;h++){
  if(datarray[h][8]!=-10000)
   fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          datarray[h][8],datarray[h][9],datarray[h][36],
          datarray[h][37],datarray[h][38],datarray[h][39],
          datarray[h][40],datarray[h][41],datarray[h][42]);
   else
      fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
              datarray[h][x+1],datarray[h][36],datarray[h][37],
              datarray[h][38],datarray[h][39],datarray[h][40],
              datarray[h][41],datarray[h][42]);
    }
  fprintf(p_out,"\n//nc\n");
  fprintf(p_out,"\n//pos .4 .5 .6 .7");
```

```
   fprintf(p_out,"\n/ag");
   fprintf(p_out,"\n/et x 'time (sec)'");
   fprintf(p_out,"\n/et y 'pressure (MPa)'");
   fprintf(p_out,"\n/et g 'transducer 3'");
   fprintf(p_out,"\n/or y -1 7");
   fprintf(p_out,"\n/td yx.......");
   for(h=0;h<goodsamp;h++){
   if(datarray[h][10]!=-10000)
     fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
              datarray[h][10],datarray[h][11],datarray[h][36],
              datarray[h][37],datarray[h][38],datarray[h][39],
              datarray[h][40],datarray[h][41],datarray[h][42]);
     else
         fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
                 datarray[h][11],datarray[h][36],datarray[h][37],
                 datarray[h][38],datarray[h][39],datarray[h][40],
                 datarray[h][41],datarray[h][42]);
     }
   fprintf(p_out,"\n//nc\n");
/*  fprintf(p_out,"\n//pos .6 .5 .8 .7");
   fprintf(p_out,"\n/ag");
   fprintf(p_out,"\n/et x 'time (sec)'");
   fprintf(p_out,"\n/et y 'corrected mV'");
   fprintf(p_out,"\n/et g 'transducer 4'");
   fprintf(p_out,"\n/td yx.......");
   for(h=0;h<goodsamp;h++){
   if(datarray[h][12]!=-10000)
   fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
   datarray[h][12],datarray[h][13],datarray[h][36],
   datarray[h][37],datarray[h][38],datarray[h][39],
   datarray[h][40],datarray[h][41],datarray[h][42]);
   else
   fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
   fill,datarray[h][13],datarray[h][36],datarray[h][37],
   datarray[h][38],datarray[h][39],datarray[h][40],
   datarray[h][41],datarray[h][42]);
```

```
    }
  fprintf(p_out,"\n//nc\n");*/
  fprintf(p_out,"\n//pos .6 .5 .8 .7");
  fprintf(p_out,"\n/ag");
  fprintf(p_out,"\n/et x 'time (sec)'");
  fprintf(p_out,"\n/et y 'pressure (MPa)'");
  fprintf(p_out,"\n/et g 'transducer 5'");
  fprintf(p_out,"\n/or y -1 7");
  fprintf(p_out,"\n/td yx.......");
  for(h=0;h<goodsamp;h++){
  if(datarray[h][14]!=-10000)
   fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
           datarray[h][14],datarray[h][15],datarray[h][36],
           datarray[h][37],datarray[h][38],datarray[h][39],
           datarray[h][40],datarray[h][41],datarray[h][42]);
   else
        fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
               datarray[h][15],datarray[h][36],datarray[h][37],
               datarray[h][38],datarray[h][39],datarray[h][40],
               datarray[h][41],datarray[h][42]);
   }
  fprintf(p_out,"\n//nc\n");
/*   fclose(p_out);
  printf("Enter the name of the 3rd output file:");
  scanf("%s",filename);

  p_out = fopen(filename,"w");   OPEN OUTPUT FILE */



  fprintf(p_out,"\n//pos .8 .5 1 .7");
  fprintf(p_out,"\n/ag");
  fprintf(p_out,"\n/et x 'time (sec)'");
  fprintf(p_out,"\n/et y 'pressure (MPa)'");
  fprintf(p_out,"\n/et g 'transducer 6'");
  fprintf(p_out,"\n/sm OFF");
  fprintf(p_out,"\n/or y -1 7");
```

```
 fprintf(p_out,"\n/td yx.......");
 for(h=0;h<goodsamp;h++){
 if(datarray[h][16]!=-10000)
   fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          datarray[h][16],datarray[h][17],datarray[h][36],
          datarray[h][37],datarray[h][38],datarray[h][39],
          datarray[h][40],datarray[h][41],datarray[h][42]);
   else
        fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
               datarray[h][17],datarray[h][36],datarray[h][37],
               datarray[h][38],datarray[h][39],datarray[h][40],
               datarray[h][41],datarray[h][42]);
    }
 fprintf(p_out,"\n//nc\n");
 fprintf(p_out,"\n//pos 0 .25 .2 .45");
 fprintf(p_out,"\n/ag");
 fprintf(p_out,"\n/et x 'time (sec)'");
 fprintf(p_out,"\n/et y 'pressure (MPa)'");
 fprintf(p_out,"\n/et g 'transducer 7'");
 fprintf(p_out,"\n/or y -1 7");
 fprintf(p_out,"\n/td yx.......");
 for(h=0;h<goodsamp;h++){
 if(datarray[h][18]!=-10000)
   fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          datarray[h][18],datarray[h][19],datarray[h][36],
          datarray[h][37],datarray[h][38],datarray[h][39],
          datarray[h][40],datarray[h][41],datarray[h][42]);
   else
        fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
               datarray[h][19],datarray[h][36],datarray[h][37],
               datarray[h][38],datarray[h][39],datarray[h][40],
               datarray[h][41],datarray[h][42]);
    }
fprintf(p_out,"\n//nc\n");
fprintf(p_out,"\n//pos .2 .25 .4 .45");
fprintf(p_out,"\n/ag");
fprintf(p_out,"\n/et x 'time (sec)'");
fprintf(p_out,"\n/et y 'pressure (MPa)'");
fprintf(p_out,"\n/et g 'transducer 8'");
fprintf(p_out,"\n/or y -1 7");
fprintf(p_out,"\n/td yx.......");
for(h=0;h<goodsamp;h++){
if(datarray[h][20]!=-10000)
  fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          datarray[h][20],datarray[h][21],datarray[h][36],
          datarray[h][37],datarray[h][38],datarray[h][39],
          datarray[h][40],datarray[h][41],datarray[h][42]);
  else
       fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
              datarray[h][21],datarray[h][36],datarray[h][37],
              datarray[h][38],datarray[h][39],datarray[h][40],
              datarray[h][41],datarray[h][42]);
   }
fprintf(p_out,"\n//nc\n");
fprintf(p_out,"\n//pos .4 .25 .6 .45");
fprintf(p_out,"\n/ag");
fprintf(p_out,"\n/et x 'time (sec)'");
fprintf(p_out,"\n/et y 'pressure (MPa)'");
fprintf(p_out,"\n/et g 'transducer 9'");
fprintf(p_out,"\n/or y -1 7");
fprintf(p_out,"\n/td yx.......");
for(h=0;h<goodsamp;h++){
if(datarray[h][22]!=-10000)
  fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
          datarray[h][22],datarray[h][23],datarray[h][36],
          datarray[h][37],datarray[h][38],datarray[h][39],
          datarray[h][40],datarray[h][41],datarray[h][42]);
  else
       fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
              datarray[h][23],datarray[h][36],datarray[h][37],
              datarray[h][38],datarray[h][39],datarray[h][40],
              datarray[h][41],datarray[h][42]);
```

```
        }
     fprintf(p_out,"\n//nc\n");
/*     fclose(p_out);
     printf("Enter the name of the 4th output file:");
     scanf("%s",filename);

     p_out = fopen(filename,"w");   OPEN OUTPUT FILE */

     fprintf(p_out,"\n//pos .6 .25 .8 .45");
     fprintf(p_out,"\n/ag");
     fprintf(p_out,"\n/et x 'time (sec)'");
     fprintf(p_out,"\n/et y 'pressure (MPa)'");
     fprintf(p_out,"\n/et g 'transducer 10'");
     fprintf(p_out,"\n/sm OFF");
     fprintf(p_out,"\n/or y -1 7");
     fprintf(p_out,"\n/td yx.......");
     for(h=0;h<goodsamp;h++){
     if(datarray[h][24]!=-10000)
       fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
                  %5.2f %5.2f",datarray[h][24],datarray[h][25],
             datarray[h][36],datarray[h][37],datarray[h][38],
             datarray[h][39],datarray[h][40],datarray[h][41],
             datarray[h][42]);
       else
           fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                  fill,datarray[h][25],datarray[h][36],
                  datarray[h][37],datarray[h][38],datarray[h][39],
                  datarray[h][40],datarray[h][41],datarray[h][42]);
  }
     fprintf(p_out,"\n//nc\n");
     fprintf(p_out,"\n//pos .8 .25 1 .45");
     fprintf(p_out,"\n/ag");
     fprintf(p_out,"\n/et x 'time (sec)'");
     fprintf(p_out,"\n/et y 'pressure (MPa)'");
     fprintf(p_out,"\n/et g 'transducer 11'");
     fprintf(p_out,"\n/or y -1 7");
fprintf(p_out,"\n/td yx.......");
for(h=0;h<goodsamp;h++){
if(datarray[h][26]!=-10000)
  fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f
              %5.2f %5.2f %5.2f",datarray[h][26],datarray[h][27],
           datarray[h][36],datarray[h][37],datarray[h][38],
           datarray[h][39],datarray[h][40],datarray[h][41],
           datarray[h][42]);
  else
       fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f
                 %5.2f %5.2f %5.2f",fill,datarray[h][27],
                datarray[h][36],datarray[h][37],datarray[h][38],
                datarray[h][39],datarray[h][40],datarray[h][41],
                datarray[h][42]);
   }
fprintf(p_out,"\n//nc\n");
fprintf(p_out,"\n//pos 0 0 .2 .2");
fprintf(p_out,"\n/ag");
fprintf(p_out,"\n/et x 'time (sec)'");
fprintf(p_out,"\n/et y 'pressure (MPa)'");
fprintf(p_out,"\n/et g 'transducer 12'");
fprintf(p_out,"\n/or y -1 7");
fprintf(p_out,"\n/td yx.......");
for(h=0;h<goodsamp;h++){
if(datarray[h][28]!=-10000)
  fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
              %5.2f",datarray[h][28],datarray[h][29],datarray[h][36],
           datarray[h][37],datarray[h][38],datarray[h][39],
           datarray[h][40],datarray[h][41],datarray[h][42]);
  else
       fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
                 %5.2f",fill,datarray[h][29],datarray[h][36],
                datarray[h][37],datarray[h][38],datarray[h][39],
                datarray[h][40],datarray[h][41],datarray[h][42]);
   }
fprintf(p_out,"\n//nc\n");
```

```
    fprintf(p_out,"\n//pos .2 0 .4 .2");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'pressure (MPa)'");
    fprintf(p_out,"\n/et g 'transducer 13'");
    fprintf(p_out,"\n/or y -1 7");
    fprintf(p out,"\n/td yx.......");
    for(h=0;h<goodsamp;h++){
    if(datarray[h][30]!=-10000)
      fprintf(p_out,"\n%3.3f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
                 %5.2f",datarray[h][30],datarray[h][31],datarray[h][36],
              datarray[h][37],datarray[h][38],datarray[h][39],
              datarray[h][40],datarray[h][41],datarray[h][42]);
      else
          fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
                   %5.2f",fill,datarray[h][31],datarray[h][36],
                 datarray[h][37],datarray[h][38],datarray[h][39],
                 datarray[h][40],datarray[h][41],datarray[h][42]);
       }
    fprintf(p_out,"\n//nc\n");
/*    fclose(p_out);
    printf("Enter the name of the 5th output file:");
    scanf("%s",filename);

    p_out = fopen(filename,"w");   OPEN OUTPUT FILE */

/*Now write power and level*/
    fprintf(p_out,"\n//pos .6 .75 .8 .95");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/sm OFF");
    fprintf(p_out,"\n/or y 600 700");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'cmv'");
    fprintf(p_out,"\n/et g 'Power Data'");
    fprintf(p_out,"\n/td yx....");
        for(h=0;h<goodsamp;h++){
```

```
        if(datarray[h][32]!=-10000)
            fprintf(p_out,"\n%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                    datarray[h][32],datarray[h][33],datarray[h][36],
                    datarray[h][37],datarray[h][38],datarray[h][39]);
        else
            fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f",fill,
                    datarray[h][33],datarray[h][36],datarray[h][37],
                    datarray[h][38],datarray[h][39]);
                }
    fprintf(p_out,"\n//nc\n");
    fprintf(p_out,"\n//pos .8 .75 1 .95");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'cmv'");
    fprintf(p_out,"\n/et g 'Level Data'");
    fprintf(p_out,"\n/or y 900 1100");
    fprintf(p_out,"\n/td yx....");
        for(h=0;h<goodsamp;h++){
        if(datarray[h][34]!=-10000)
            fprintf(p_out,"\n%5.2f %5.2f %5.2f %5.2f %5.2f %5.2f",
                    datarray[h][34],datarray[h][35],datarray[h][36],
                    datarray[h][37],datarray[h][38],datarray[h][39]);
        else
            fprintf(p_out,"\n%s %5.2f %5.2f %5.2f %5.2f %5.2f",
                    fill,datarray[h][35],datarray[h][36],
                    datarray[h][37],datarray[h][38],datarray[h][39]);
                }

if(cane==1){
    fprintf(p_out,
            "\n//nc\n");
    fprintf(p_out,"\n//pos .4 0 .6 .2");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'force (lbf)'");
    fprintf(p_out,"\n/et g 'force cane'");
    fprintf(p_out,"\n/td yx");
    for(h=0;h<goodsamp;h++){
      fprintf(p_out,"\n%3.2f %5.2f",datarray[h][36],datarray[h][1]);
        }
    fprintf(p_out,"\n//nc\n");
 }
 if(leftfoot!=0){
    fprintf(p_out,"\n//nc");
    fprintf(p_out,"\n//pos .6 0 .8 .2");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et g 'left foot'");
    fprintf(p_out,"\n/or y 0 5");
    fprintf(p_out,"\n/td xy");
    for(h=0;h<goodsamp;h++){
      fprintf(p_out,"\n%5.2f %5.2f",datarray[h][1],datarray[h][37]);
         }
    fprintf(p_out,"\n//nc");
    fprintf(p_out,"\n//pos .8 0 1 .2");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et g 'right foot'");
    fprintf(p_out,"\n/or y 0 5");
    fprintf(p_out,"\n/td xy");
    for(h=0;h<goodsamp;h++){
      fprintf(p_out,"\n%5.2f %5.2f",datarray[h][1],datarray[h][38]);
        }
    fprintf(p_out,"\n//nc\n");
 }
 if(gon==1){
    fprintf(p_out,"\n//nc");
    fprintf(p_out,"\n//pos .6 0 .8 .2");
    fprintf(p_out,"\n/ag");
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'angle (deg)'");
    fprintf(p_out,"\n/et g 'goniometer'");
```

```
      fprintf(p_out,"\n/or y -140 140");
      fprintf(p_out,"\n/td yx");
      for(h=0;h<goodsamp;h++){
        fprintf(p_out,"\n%3.2f %5.2f",datarray[h][39],datarray[h][1]);
           }
      fprintf(p_out,"\n//nc\n");
  }
  if (tsync == 1){
      fprintf(p_out,"\n//nc");
      fprintf(p_out,"\n//pos .8 0 1 .2");
      fprintf(p_out,"\n/ag");
      fprintf(p_out,"\n/et x 'time (sec)'");
      fprintf(p_out,"\n/et g 'TRACK synch'");
      fprintf(p_out,"\n/td xy");
      for(h=0;h<goodsamp;h++){
        fprintf(p_out,"\n%5.2f %5.2f",datarray[h][1],datarray[h][40]);
        }
      fprintf(p_out,"\n//nc\n");
   }
  if(fpright==1){
      fprintf(p_out,"\n//nc\n");
      fprintf(p_out,"\n//pos .4 0 .6 .2");
      fprintf(p_out,"\n/ag");
      fprintf(p_out,"\n/et x 'time (sec)'");
      fprintf(p_out,"\n/et y 'Vertical force (N)'");
      fprintf(p_out,"\n/et g 'Forceplate 1 (right)'");
      fprintf(p_out,"\n/td xy");
      for(h=0;h<goodsamp;h++){
fprintf(p_out,"\n%3.2f %5.2f",datarray[h][1],datarray[h][41]);
          }
      fprintf(p_out,"\n//nc\n");
  }
  if(fpleft==1){
      fprintf(p_out,"\n//nc\n");
      fprintf(p_out,"\n//pos .6 0 .8 .2");
      fprintf(p_out,"\n/ag");
```

```
    fprintf(p_out,"\n/et x 'time (sec)'");
    fprintf(p_out,"\n/et y 'Vertical force (N)'");
    fprintf(p_out,"\n/et g 'Forceplate 2 (left)'");
    fprintf(p_out,"\n/td xy");
    for(h=0;h<goodsamp;h++){
fprintf(p_out,"\n%3.2f %5.2f",datarray[h][1],datarray[h][42]);
        }
}

    fpleft = 0;
    fpright = 0;
    cane = 0;
    fplates = 0;
    check = 0;
    leftfoot = 0;
    rightfoot = 0;
    gon = 0;
    tsync = 0;  /* TRACK synch signal */
    x = 0;
    datum = 0.0;
    conversion = 0.0;
    timeskip = 0.0;

fclose(p_out);
return;

}


/* FSVALUE14 READS IN DATA IF FS CHANNEL 1 OR 4 WAS USED, AND
RETURNS
  1 IF FOOT IS NOT TOUCHING GROUND
  2 IF ONLY HEEL IS TOUCHING GROUND
  3 IF BOTH HEEL AND TOE ARE TOUCHING GROUND
  4 IF ONLY TOE IS TOUCHING GROUND
  0 IF VALUE FALLS OUT OF RANGE (data is bad) */
```

```
float fsvalue(float datum,int channel)
{
  if(channel==0)
   return (0.0);

  else if(channel==1 || channel==4){
   if (datum > 277.0 && datum < 300.0)
     return (1.0);

   else if (datum > 240 && datum < 252)
     return (2.0);

   else if (datum < 236 )
     return (3.0);

   else if (datum > 259 && datum < 271)
     return (4.0);

   else
     return (0.0);
     } /*end channel 1 or 4 loop*/
 else {
  if (datum > 282.0 && datum < 300.0)
    return (1.0);

  else if (datum > 250.0 && datum < 262.0)
    return (2.0);

  else if (datum < 245.0)
    return (3.0);

  else if (datum > 266.0 && datum < 278.0)
    return (4.0);

  else
```

```
        return (0.0);
        } /*end channel 2 or 3 loop*/


}
```

```
/*Function for averaging resting data sets into zero values for*/
/*Prosthesis 43's thirteen pressure transducers*/
/*January 28,1992*/

#include <stdio.h>
#include <string.h>

void main (void){

int i,h,j;
int frames,adchan;
char filein[14],fileout[14],datastr[20];
FILE *p_data,*p_out;
float data[14],datum,temp;
float power,level,cal,zero;
for(i=0;i<13;i++)
          data[i]=0.0;
/*get input file*/
printf("\nEnter the name of the resting data set:");
scanf("%s",filein);
p_data=fopen(filein,"r");

/*Input header values*/
          fscanf(p_data,"%*s %*s %*s %*s %*s %*lf");
          fscanf(p_data,"%*s %d %*s %d",&frames,&adchan);
          printf("\nFrames: %d Channels: %d",frames,adchan);
          fscanf(p_data,"%*s %*s %*s");
/*Input and total up the transducer values */
for(i=0;i<frames;i++){
          fscanf(p_data,"%f %f %f",&temp,&cal,&zero);
          for(h=0;h<13;h++){
                    fscanf(p_data,"%s",datastr);     /*transducers*/
                    sscanf(datastr,"%f",&datum);
                    *(data+h)+=datum;
/*              printf("\n%.2f %.2f",datum,*(data+h));   */
                }
        fscanf(p_data,"%f %f",&power,&level);           /*power and level*/
/*      printf("\n%d\n%d",power,level);  */
        if(adchan==2){
                for(j=0;j<16;j++)
                        fscanf(p_data,"%*s");      /*aux/mux values*/
                }
        datum=0.0;
        } /*end frames loop*/
fclose(p_data);
/*now average the transducer sums*/
for(h=0;h<13;h++){
        datum=data[h]/(float)frames;
        data[h]=datum;
        }

/*get output file*/
printf("\nInput name of output file (ex:  dc16zero.dat):");
scanf("%s",fileout);
p_out=fopen(fileout,"w");

/*put averages into file*/
for(h=0;h<13;h++)
        fprintf(p_out,"%.2f ",data[h]);
fclose(p_out);
```

E.3 Programs for obtaining acetabular locations of pressure measurements

E.3.1 For data from prosthesis 43

Include files for LOCATE43.EXE

```
/**************************************************/
```

```
/*      kinemat.h                               */
/*                include file for "locate"     */
/*                                              */
/*      Dec. 5, 1992      K.C.                  */
/************************************************/

#define NTRANS  13
#define MAXSEG  10

typedef struct angles {
          double phi;
          double theta;
          } ang;

typedef struct f_angles {
       struct angles ang[NTRANS];
     } femoral_ang;

typedef struct a_trans {
          struct angles ang;
          double mpa;
          };



typedef struct a_frame {
          struct a_frame *next;             /* For doubly linked list */
          struct a_frame *prev;
          double time;
          double flexion, abduction, extrotation;/*clinical jt angs at hip*/
          struct a_trans alltrans [NTRANS]; /* phi,theta, and pressure */
          float fp1_vert, fp2_vert;    /*Scaled FP Y forces */
          } trans_a;

/*   If using femoral view: no need to duplicate phi and theta */

typedef struct f_frame {
        struct f_frame *next;  /* For creating a linked list */
        struct f_frame *prev;
        double mpa [NTRANS];
        } trans_f;

trans_a _huge *a_entry;  /* anchor for a_frame list */
trans_f _huge *f_entry;  /* anchor for f_frame list */

femoral_ang *f_ang;


typedef struct kineinfo {
        char name[12];              /* File name */
        char descr[40];             /* Description */
        char side;                  /* Right or Left */
        int nseg;                   /* Number of segments collected */
        int frames;                 /* frames of kinematic data taken */
        int synch_frames;           /* # of synched frames (w/pressure)*/
        int nfp;                    /* # of forceplates used */
        float wgt;
        double freq;                /* frequency of kinematic data */
        double prosth_to_femoral_rot [NTRANS][3];  /* Rot matrix – transducer*/
                                    /*       position in femoral coordinates */
        double pelvis_to_acet_rot [3][3]; /* Rot matrix –pelvis to acet. */
        float seg_vec [MAXSEG][3];          /* Segment vectors from kine header*/
        float seg_rot [MAXSEG][3][3];       /* Segment rot. matrix from T4header*/
        float array_pos [2][3];
        float array_rot [2][3][3];
        };

typedef struct pzeros {
        float offset[16];
        float gain[16];
        float tempcorr[16];
        float temp;
        };
```

```
/* Header file for use with programs that access prosthesis */
/* data files                Jan. 17, 1990                 */



#define GLOBAL
#define SEMIGLOBAL static /* used to define a semiglobal variable*/
#define IMPORT extern /* used to refer to a global or semiglobal var*/
                      /* referenced elsewhere */


#define SHORTSIZE 9
#define PATHSIZE 80
#define PROSTH_IDSIZE 3
#define DESCR_SIZE 32
#define DIRNAMESIZE 50
#define NAMESIZE 30


struct datainfo {
        char name[9];
        char id[3];
        int samp;
        int adchan;
        int mult;
        int trig;
        char time[5][12];
        char descr[32];
        int startoff;
        int frames;
        int alarmch;
        int synchch;
        double freq;
        int ntrans; /* number of transducers */
        int date[3];
        char hip;
        int synch_frames; /* Number of frames TSYNCH active */
        int binary;
        };

SEMIGLOBAL char short_dataname [SHORTSIZE] = "";
SEMIGLOBAL char prosth_id [PROSTH_IDSIZE] = "";
SEMIGLOBAL char description[DESCR_SIZE] = "";
SEMIGLOBAL char namdir[DIRNAMESIZE] = "";
```

```
/*--------------------------------------------*/
/*      t4i_struct.h                 */
/*      From MGH                        */
/*      11/23/92                      */
/*--------------------------------------------*/

#define DNAMESIZE    12
#define MAXDESCR     40
#define TIMESIZE     26
#define MAXLED       32
#define MAXCAM       2
#define MAXFP        2
#define KSCHAN       16        /* Really 2 * number of FP's */
#define MAXSEG       10
#define NU_SUBJ_PARS 48
#define MAX_ECHN     8
#define MAX_EVENT    24

#define MAXLINE      10   /* MADE THESE 2 UP */
#define TERMCAR       10

typedef struct {
```

```
char    file_raw[DNAMESIZE];      /* Name of original raw */
char    dir_raw[DNAMESIZE];       /* Directory of orig. raw */
char    file_segm[DNAMESIZE];     /* Segment file name */
char    description[MAXDESCR];    /* Data description */
char    sampl_time[TIMESIZE];     /* Time of data sampling*/
char    smooth_raw;               /* Raw data smoothing flag*/
char    smooth_3dp;               /* 3d data smoothing flag */
char    smooth_bcs;               /* BCS data smoothing flag */
char    filter_fpd;               /* F'plate filtering flag */
char    dum1;   /* Dummy variables to be on even word boundary*/
char    dum2;   /* ditto*/
float   sord_raw;          /*smoothing function order for raw data*/
float   sord_3dp;          /*smoothing function order for 3dp data*/
float   sord_bcs;          /* smoothing function order for track data*/
float   fltcut_fpd;        /*lowpass filter cutoff for fcplate*/
char    data_windo;        /*Processing window flag*/
char    only_3dp;          /* Compute 3D data only flag */
char    selspot_flag;      /* Collect Selspot data */
char    kistler_flag;      /* collect Kistler data */
char    analog_flag;       /* collect A/D data flag */
char    fpside_flag;       /* forceplate to associate with ?*/
char    cam1_flag;         /* first camera flat [1,2,3,4]*/
char    cam2_flag;         /* second camera flat [1,2,3,4] */
long    bad_points;        /* number of bad points */
long    nb_fplates;        /* number of forceplates used */
long    nb_frames;         /* number of frames collected */
long    nb_segment;        /* number of selspot segments used */
long    nb_channel;        /* number of selspot channels used */
long    act_frame1;        /* number of frames windowed (?) */
long    f_channels;        /* number of bad channels */
long    f_segments;        /* number of bad segments */
long    sray_badpt;        /* bad points due to skew ray error */
long    f_wframe;          /* first frame of window */
long    l_wframe;          /* last frame of windwo */
long    led_segmnb[MAXLED];       /* segment number for ea. channel?*/
float   led_xcoord[MAXLED];       /*seg x-coord for ea channel*/
float   led_ycoord[MAXLED];     /*seg y_coord for ea. channel*/
float   led_zcoord[MAXLED];     /*seg z_coord for ea. channel*/
float   campos[MAXCAM][3];      /*camera positions*/
float   fppos[MAXFP][3];        /* forceplate positions */
float   cam_rotmtx[MAXCAM][3][3];/*camera rot. matrix for ext. cal.*/
float   fp_rotmtx[MAXFP][3][3];/*fp rot matrix for ext. calib.*/
float   focal_cam1;       /*focal length camera 1*/
float   focal_cam2;       /* focal length camera 2 */
float   max_ledvar;       /* max seen inter-led error */
float   skewray_mx;       /* max seen skew ray error */
float   frequency;        /* frequency of collection */
float   worst_led;        /* worst inter-LED length */
float   globx_axis;       /* global origin relocation */
float   globy_axis;       /* global origin relocation */
float   globz_axis;       /* global origin relocation */
float   fpxscl;           /* forceplate X scale factor */
float   fpyscl;           /* forceplate Y scale factor */
float   fpzscl;           /* forceplate Z scale factor */
float   fp_zero[KSCHAN];        /* forceplate zero offsets */
short   iscal;            /* segment file scale param - MEGATEK*/
short   xseg[MAXSEG][10];/* segment descr. MEGATEK artifact */
short   yseg[MAXSEG][10];/* segment descr. MEGATEK artifact */
short   zseg[MAXSEG][10];/* segment descr. MEGATEK artifact */
char    dum3;   /* dummy var. to be on even word boundary */
char    dum4;   /* dummy var. */
float   seg_vec[MAXSEG][3];     /*array to segment vectors */
float   seg_rot[MAXSEG][3][3]; /*array to segment xform matrix */
float   fxyz[12][3];      /* fp segment description */
float   subj_params[NU_SUBJ_PARS];    /*age, ht, wt.,measurements*/
short   fs_dat[MAX_ECHN][2][MAX_EVENT]; /*event marker record */
float   rd[MAXSEG][3];  /*distal joint vector from crgeom */
float   rp[MAXSEG][3];  /*proximal joint vector from crgeom */
float   rcg[MAXSEG][3]; /* segment cg vector from crgeom */
float   segmas[MAXSEG]; /* segment mass from dinert */
float   seginert[MAXSEG][3]; /* principal moments of inertia */
float   segint_mat[MAXSEG][3][3]; /*moments of inertia rot matrix */
```

```
        } t4header;

struct t4tree {
        char    data[MAXLINE];
        char    parameter[MAXLINE];
        char    segment[MAXLINE];
        char    work[MAXLINE];
        };

struct t4terminal{
        char    line_up[TERMCAR];
        char    clear_scr[TERMCAR];
        char    clean_page[TERMCAR];
        int     nb_lines;
        int     nb_columns;
        };

typedef struct{
        char    byte1, byte2, byte3, byte4;
        } Value;
```

```
/*   Function prototypes for locate43.c */

FILE *select_data (struct datainfo *, FILE *);
char get_view(FILE *);
femoral_ang *f_locate(struct datainfo *);
struct f_frame _huge *get_mpamemory (struct datainfo *);
void fstor_prs (FILE *, struct datainfo *, trans_f _huge **);
void astor_prs (FILE *, struct datainfo *, struct kineinfo *,
        trans_a _huge **);
void xform(FILE *,trans_a _huge **, struct datainfo *, struct kineinfo *,
                femoral_ang *, double);
trans_a _huge *get_angles(struct datainfo *, struct kineinfo *,
                        FILE *, int, FILE *);
void fwrite_tec (trans_f _huge **, struct datainfo *, femoral_ang *, FILE *);
void awrite_tec (trans_a _huge **, struct datainfo *, struct kineinfo *,
                FILE *);
void freealist (trans_a _huge **);
void freeflist (trans_f _huge **);
void calc_matrices (FILE *, struct dtainfo *, struct kineinfo *,
                femoral_ang *);
void hip_xform (FILE *,double, double, double, struct kineinfo *,
                trans_a _huge *, struct datainfo *);
void get_dtl(struct kineinfo *, FILE *);
void ang_calc(struct kineinfo *, trans_ _huge **,
                struct datainfo *, FIL *, FILE *);
int find_synch (FILE *, struct dataino *, int);
trans_a _huge *get_memory(int);
void hip_ang (float *, double *, double *, double *, char);
void mgh_ang (float *, double *, double *, double *, char);
void mult_matrices(float *, float *, float *, int, int, int);
```

LOCATE43.C

```
/*********************************************************/
/*      Program to determine transducer locations        */
/*      and write locations and pressures to a file.     */
/*      File structure not yet determined.               */
/*                                                       */
/*      Dec. 2, 1992      K.C.                           */
/*                                                       */
/*      Feb 9, 1993: Changed to LOCATE43, just deal      */
/*      with data from 43.  Make another verson to deal */
/*      with 33 -- just too many options right now.      */
/*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys\types.h>
#include <sys\stat.h>
```

```
#include <string.h>
#include <math.h>
#include <errno.h>
#include <malloc.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

struct datainfo pinfo;
struct kineinfo kinfo;



void main(int argc, char *argv[] )
{
          char view[2], reply[4];
          double freq;
          int frames, syn_chan;

          struct datainfo *ppinfo;
          struct kineinfo *pkinfo;
          femoral_ang *f_a;

          char filepath[40];
          char paramfile[40];
          struct stat *buffer;

          FILE *p_file;
          FILE *outfile;
          FILE *parfile;

          if (argc < 2){
             printf("Enter file of processing parameters:  ");
             scanf ("%s", argv[1]);
             }
          strcpy(paramfile,argv[1]);
          parfile = fopen(paramfile,"r");
```

```
            /* For now just read synch-channel out */
        fscanf(parfile,"%d",&syn_chan);


        ppinfo = &pinfo;
        pkinfo = &kinfo;
        p_file = select_data (ppinfo,parfile);

#ifdef _X_DEBUG_
        outfile = fopen("debugx.dat","w");
#endif
        fscanf (parfile,"%s",view);
/*      fclose(parfile);*/

        f_ang = f_locate(ppinfo);
        switch (view[0]) {
                case 'f':
                        f_entry = get_mpamemory (ppinfo);
                        fstor_prs (p_file, ppinfo, &f_entry);
                        break;
                case 'a':
                        a_entry = get_angles(ppinfo,pkinfo,p_file,
                                                syn_chan,parfile);
                        if ((ppinfo->freq) < (pkinfo->freq))
                                        freq = ppinfo->freq;
                        else freq = pkinfo->freq;
#ifdef _KJC_PRINT_
                        printf("Freq = %lf\n",freq);
                        printf("Want to go on?");
                        scanf("%s",reply);
#endif
                        xform (outfile, &a_entry, ppinfo, pkinfo,
                                        f_ang, freq);
#ifdef _KJC_PRINT_
                        printf("Am back from xform");
#endif
                        astor_prs(p_file, ppinfo, pkinfo,
                                        &a_entry);
                            break;
                    }

#ifdef _X_DEBUG_
        fclose(outfile);
#endif
        if (f_entry == 0){
            awrite_tec(&a_entry, ppinfo, pkinfo,parfile);
            freealist (&a_entry);
          }
        else  {
            fwrite_tec(&f_entry, ppinfo, f_ang, parfile);
            freeflist (&f_entry);
          }
        printf("Memory should be freed.");
        fclose(parfile);
}

void freealist (trans_a _huge **list)
{
        trans_a _huge *hold;
        trans_a _huge *current;

        current = *list;

        while (current->prev) current = current->prev;
        while (current) {
             hold = current->next;
             hfree(current);
             current = hold;
           }
        return;
}

void freeflist (trans_f _huge **list)
```

```
{
        trans_f _huge *hold;
           trans_f _huge *current;

           while (current->prev) current = current->prev;
           while (current){
                hold = current->next;
                hfree (current);
                current = hold;
              }
          return;
}
```

SELECT_D.C

```
/*********************************************************/
/*        select_data -                                 */
/*                 function to determine data set to    */
/*                 look at.                             */
/*                                                      */
/*        Dec. 2, 1992       K.C.                       */
/*********************************************************/

#include <stdio.h>
#include <string.h>
#include "p_file.h"
#include <malloc.h>


FILE *select_data(struct datainfo *p_spec, FILE *parfile)
{
          int pfreq,i;
          char dataname [NAMESIZE];
          char shortname [7];
```

```
        char reply[3];
        char *dum;
        FILE *p_dat;

/*      printf("Enter the complete name of file to look at: "); */
        fscanf(parfile,"%s",dataname);

/* Ask what format file is in */

/*      printf ("Is this file in old MGH format? ");    */
        fscanf(parfile,"%s",reply);
if (reply[0] == 'y' || reply[0] == 'Y'){
/*              printf("Is this file in binary format? "); */
                fscanf (parfile,"%s",reply);
                if (reply[0] == 'y' || reply [0] == 'Y'){
                        p_dat = fopen (dataname,"rb");
                        dum = (char *)calloc(sizeof(char),76);
                        for (i=0;i<11;i++){
                          fread( dum, sizeof(char),76,p_dat);
                          }
                        free(dum);
                        p_spec->binary = 1;
                        strcpy(p_spec->id,"33");
                        p_spec->freq = 254;
                        }
                else {
                        p_dat = fopen (dataname, "r");
                        fscanf (p_dat, "%d %d",&p_spec->frames,&pfreq);
                        fscanf (p_dat, "%d %d %2d%*d",&p_spec->date[0],
                                  &p_spec->date[1], &p_spec->date[2]);
                        sscanf (dataname,"%6s",shortname);
                        strcpy (p_spec->name, shortname);
                        strcpy (p_spec->id,"33");
                        p_spec->freq = (double)pfreq;
                        }
                }
```

```
          else {
                    /* Start filling datainfo */
                    p_dat = fopen(dataname,"r");
                    fscanf(p_dat,"%s %s",p_spec->name,p_spec->id);
                    fscanf (p_dat,"%*s %*s %*s %lf",&(p_spec->freq));
                    fscanf (p_dat,"%*s %d",&(p_spec->frames));
#ifdef _KJC_PRINT_
                    printf (" %lf %d\n",p_spec->freq, p_spec->frames);
#endif
                    fscanf (p_dat,"%*s %d",&(p_spec->adchan));
                    fscanf (p_dat,"%*s %*s %*s");
/*        Leave p_dat at start of data            */
                    }
          if (strncmp(p_spec->id,"33",2)==0) p_spec->ntrans = 13;
          if (strncmp(p_spec->id,"43",2)==0) p_spec->ntrans = 13;
          return(p_dat);
}
```

GET_DTL.C

```
/*****************************************************************/
/*        get_dtl  ---                                           */
/*                  function to get the .DTL file.  Not sure     */
/*        yet what to read or how.....                           */
/*                                                               */
/*        Dec. 9, 1992        K.C.                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include "p_file.h"
```

```
#include "kinemat.h"
#include "t4i_stru.h"


void get_dtl( struct kineinfo *kinfo, FILE *finl)
{

   float nrec,fchn,frame,fseg,ddum;
   int index,rec,ireturn;

   t4header *lfile_header;
   int ii,jj,kk,ijk,retval;
   int  nseg_l, side;
   int nframe;
   int nfp;
   int kscal;
   float wgt, freq, cutoff;

   lfile_header = (t4header *) calloc(1,(unsigned long)sizeof(t4header));

/*                    Program was altered assuming a left leg -- but should */
/* work for either.   Need to determine side, store in kineinfo.   */


   retval =  fread(lfile_header, (unsigned long)sizeof(t4header),1,finl);
   if(retval){
         nseg_l = lfile_header->nb_segment;
         kinfo->nseg = nseg_l;
         nfp = lfile_header->nb_fplates;
         kinfo->nfp = nfp;
         strcpy(kinfo->name, lfile_header->file_raw);
         strcpy(kinfo->descr, lfile_header->description);
         printf ("This file contains data for:  %s\n",kinfo->descr);
         nframe = lfile_header->nb_frames;
#ifdef _KJC_PRINT_
         printf ("The number of kinematic frames is:  %d",nframe);
#endif
        kinfo->frames = nframe;
        kinfo->freq = (double) (lfile_header->frequency);
#ifdef _KJC_PRINT_
        printf ("%lf\n",kinfo->freq);
#endif
        for (ii =0; ii < nseg_l; ii++){
           for(jj = 0; jj < 3; jj++){
                 kinfo->seg_vec[ii][jj] = lfile_header->seg_vec[ii][jj];
                 for(kk = 0; kk < 3; kk++){
                    kinfo->seg_rot[ii][jj][kk] =
                                   lfile_header->seg_rot[ii][jj][kk];
                 }
           }
        }
  }

  wgt = lfile_header->subj_params[3];
  kinfo->wgt = wgt;
  printf(" Subject weight is:  %f\n", wgt);
  printf(" Subject height is:  %f\n", lfile_header->subj_params[2]);

  free (lfile_header);
  return;
}
```

F_LOCATE.C

```
/*****************************************/
/*      f_locate --                      */
/*            function to get femoral    */
/*            locations from file        */
/*            ID.ang   (where ID is      */
/*            prosthesis serial #        */
/*                                       */
/*      Dec. 2, 1992      K.C.           */
```

```
/******************************************/

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "p_file.h"
#include "kinemat.h"

femoral_ang *f_locate (struct datainfo *pinfo)
{
        FILE *angfile;
        char angname[20];
        int i;

        f_ang = (femoral_ang *)_fcalloc (1,  (unsigned long)sizeof(femoral_ang));

/*              Get the file name with the femoral locations */

        strcpy (angname, pinfo->id);
        strcat (angname,".ang");

/*              Read the femoral locations of transducers */
/*              phi is latitude; theta is longitude              */

        angfile = fopen (angname, "r");
        for (i=0;i<pinfo->ntrans;i++){
                fscanf (angfile,"%lf %lf",&(f_ang->ang[i].phi), \
                                        &(f_ang->ang[i].theta));
#ifdef _KJC_PRINT_
                printf("%lf, %lf\n",f_ang->ang[i].phi, f_ang->ang[i].theta);
#endif
                }
        fclose(angfile);

        return(f_ang);
}
```

FIND_SYN.C

```
/**********************************************************/
/*      find_synch  ---                                   */
/*              function to determine how many synched    */
/*      frames there are in a pressure data file (.INT) */
/*      Synch is TRACK synch signal.  May as well start   */
/*      reading the file, store the information, leave  */
/*      file pointer at the start of first synched frame*/
/*                                                        */
/*      Dec. 23, 1992     K.C.                            */
/**********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"


int find_synch (FILE *prsfile, struct datainfo *pinfo, int sync)
{
        char sdata[10];
        int syn_chan, idata, sum_frames;
        int i, j, retval, skip, frames;
        fpos_t *begin;
        fpos_t *hold;

/*              prsfile was left at start of data by "select_data" */

        sum_frames = 0;
        retval = fgetpos(prsfile, begin);
#ifdef _KJC_PRINT_
        printf("begin = %ld\n", *begin);
#endif
```

```
        if (pinfo->adchan < 2) {
                printf ("This data did not have synchronizing signal.");
                abort;
                }
        else {
                syn_chan = sync;
                syn_chan += 6;
                if (syn_chan > 15) syn_chan = syn_chan - 16;
                }

        frames = pinfo->frames;
        for (i=0; i<frames; i++){
              retval = fgetpos (prsfile, hold);
#ifdef _KJC_PRINT
                if (i<10)
                   printf("hold = %ld ", *hold);
#endif
                for (j=0; j<18; j++){
                        fscanf (prsfile,"%*s");
                        }
                if (pinfo->adchan == 2){  /* HAD BETTER BE!!!*/
                        for(j=0;j<16;j++){
                                if (j==syn_chan){
                                      fscanf(prsfile,"%s",sdata);
                                         if (sdata[0] == '*'){
                                            sdata[0] = ' ';
                                           }
                                         sscanf(sdata,"%d",&idata);
                                    }
                                else fscanf (prsfile,"%*s");
                                }

#ifdef _KJC_PRINT_
                                if (i<10) printf ("frame:  %d; %d\n",i,idata);
#endif
                        }
                if (idata < 50){
                      sum_frames++;
                         if (sum_frames == 1){
                            *begin = *hold;
#ifdef _KJC_PRINT_
                            printf("begin = %ld\n", *begin);
#endif
/* This is where to start saving data*/
                         }
                      }
/*  Reset the file pointer to start of data want to read */
           }

#ifdef _KJC_PRINT_
        printf("begin = %ld\n", *begin);
#endif
        retval = fsetpos(prsfile, begin);
        printf ("synched frames %d\n",sum_frames);
        pinfo->synch_frames = sum_frames;

        return(sum_frames);
}
```

GET_ANGL.C

```
/*******************************************************************/
/*      get_angles  ---                                           */
/*              function to get the .DTL file.  Not sure          */
/*      yet what to read or how..... will read .dtl file          */
/*      in a subfunction (cut-down version of process from        */
/*      MGH), decide on frequency to use, allocate memory         */
/*      for the trans_a structure, and calculate joint angles     */
/*      and store them in trans_a structure.                      */
/*                                                                 */
/*      Also, try to read .FPD file if it is available            */
/*                                                                 */
/*      Dec. 9, 1992      K.C.                                  */
/****************************************************************/

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "p_file.h"
#include "kinemat.h"
#include "t4i_stru.h"
#include "fnc.h"

trans_a _huge *get_angles(struct datainfo *pinfo,
        struct kineinfo *kinfo, FILE *prsfile,int sync, FILE *parfile)
{

  FILE *d_data, *f_data;
  char dataname[80],fpdname[80];
  char leg[5], side[5];
  int i, nframe, synch_frames;

/*  Get the file name */

/*      printf("Enter the name of file to look at (no extension): ");*/
        fscanf(parfile,"%s",dataname);
      printf("%s\n",dataname);
        strcpy(fpdname,dataname);
        strcat(fpdname,".fpd");
/*      printf ("Is this right or left leg data? ");*/
        fscanf (parfile,"%s",leg);
        kinfo->side = leg[0];
        if (leg[0] == 'L' || leg[0] == 'l')
           strcat(dataname,".dtl");
        if (leg[0] == 'R' || leg[0] == 'r')
           strcat(dataname,".dtr");
        d_data = fopen(dataname,"rb");
        get_dtl(kinfo, d_data);
```

```
/*      Decide whether to use TRACK or pressure frequency        */
/*      Perhaps don't want to do this here...                    */
/*      .... Need to account for non-synchronicity of TRACK      */
/*      and pressure data - need to find TSYNCH signal in .INT   */
/*      file and figure out how to synchronize the two sets      */
/*      of data.  It won't affect much here, except the memory   */
/*      allocatation. (TRACK always starts after pressure)       */
/*      Need to know - which pframe to start on, how many frames*/
/*      it is active for.                                        */

/*    Having memory troubles - need to read the TRACK data    */
/*    a frame at a time instead.                           */

        synch_frames = find_synch (prsfile, pinfo, sync);
        if (pinfo->freq < kinfo->freq){
                nframe = synch_frames;
                pinfo->synch_frames = nframe;
                }
        else {
                if ((synch_frames/pinfo->freq) <
                   (kinfo->frames/kinfo->freq)){
                        nframe = synch_frames/pinfo->freq * kinfo->freq;
                        }
                else nframe = kinfo->frames;
                kinfo->synch_frames = nframe;
                }
/*      Allocate memory for trans_a structure              */

        a_entry = get_memory(nframe);

/*      Calculate the angles                               */

        f_data = fopen(fpdname,"rb");
        ang_calc(kinfo, &a_entry, pinfo, d_data, f_data);
#ifdef _KJC_PRINT_
```

```
        printf("Back in get_angles.\n");
#endif


        fclose(d_data);
        fclose(f_data);
        return(a_entry);


}
```

ANG_CALC.C

```
/*********************************************************/
/*      ang_calc ---                                     */
/*              function to calculate the clinical       */
/*      joint angles.  Uses Grood and Suntay             */
/*      formulation, from Pat Lord.  May also do         */
/*      Tupling and ? as at MGH - for comparison         */
/*                                                       */
/*      Dec. 22, 1992   K.C.                             */
/*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

void ang_calc (struct kineinfo *kinfo,
                trans_a _huge **a_entry, struct datainfo *pinfo,
                FILE *fdata, FILE *fpdata)
{
        char side,reply[4];
        int skip, kframe, curframe, num, ntp;
        int i, j, k, l, nframe, iside, ijoint, nseg, nsynch;
        float smt[2][4][4]; /* Check dimensions ???? */
        float smt2[2][3][3];
        float seg_rot [2][3][3]; /* Store only for the two */
        float seg_vec [2][3];         /* around the joint of interest */
        float array_pos [2][3]; /* Array position */
        float array_rot [2][3][3]; /* Array rot. */
        double f,a,e;
        double *flex, *abd, *extrot;
        double bad, time, rem,freq;
        float *psmt2, *psegrot, *parrayrot, *psmt;
        float *data_hold, *data;
        float *fpd;
        float wgtnt;
        double *skipptr, kf;
        trans_a _huge *base;
        trans_a _huge *hold;
        trans_a _huge *current; /* For linked list */

        FILE *debug;

#ifdef _KJC_DEBUG_
        debug = fopen ("debugang.dat","w");
#endif

        bad = -1000.0;
        nsynch = 0;
        side = kinfo->side;
        if (side == 'L') iside = 2;
        else iside = 1;
        nfp = kinfo->nfp;
        ijoint = 3; /* Hope this is the right answer! Want the hip joint*/
        nseg = kinfo->nseg;
        base = *a_entry;              /* Set up linked list */
        current = base;
        curframe = 0;
```

```
    data = (float *) calloc (16*nseg, sizeof(float));
    data_hold = data;
    if (fpdata != NULL) fpd = (float *) calloc (6*nfp, sizeof(float));
    wgtnt = kinfo->wgt * 9.8/2.2;      /* wt. in lbs -> wt. in Newtons */
    skipptr = &kf;
    flex = &f;
    abd = &a;
    extrot = &e;

/* Set all of smt to 0.0 */
    for (j=0;j<2;j++){
      for (k=0;k<4;k++){
        for (l=0;l<4;l++){
          smt[j][k][l] = 0.0;
        }
      }
      for (k=0;k<3;k++){
        for (l=0;l<3;l++){
          array_rot[j][k][l] = 0.0;
        }
    array_pos[j][k] = 0.0;
      }
    }


/*              Fill the segment vector and rotation matrices */
/*              from kineinfo structure.                       */
/*              Assume the segments of interest (femur, pelvis) */
/*              are segments 3 and 4                           */

/*          Transpose the segment rotation matrices now in  */
/*          preparation for matrix multiplication          */

    for (k=2;k<4;k++){                /* Segment loop */
            for (i=0;i<3;i++){
                    seg_vec [k-2][i] = kinfo->seg_vec[k][i];
```

```
                        for (j=0;j<3;j++){
                                seg_rot[k-2][j][i]=kinfo->seg_rot[k][i][j];
                                }
                        }
                }

/*              Now extract the array position and rotation matrices*/
/*              from the stored data.  16 numbers per segment per frame      */
/*              Order - array position, 4 unidentified #'s, array rot.*/

/*              Deal with interpolation at this point if pfreq < kfreq */
#ifdef _X_DEBUG_

        printf("kfreq = %f\n",kinfo->freq);
        printf("pfreq = %f\n",pinfo->freq);
        printf("ksync = %d\n",kinfo->synch_frames);
        printf("psync = %d\n",pinfo->synch_frames);
#endif
        if (pinfo->freq < kinfo->freq) {
                skip = 1;
                nframe = pinfo->synch_frames;
                freq = pinfo->freq;
                }
        else {
                skip = 0;
                nframe = kinfo->synch_frames;
                freq = kinfo->freq;
                }

        for (i=0;i<nframe;i++){
                if (skip){
                        time = ((double) i)/ freq;
                        rem = modf((time * kinfo->freq),skipptr);
                        kframe = (int)(time*(kinfo->freq));
                        if (rem > 0.5) kframe++;
#ifdef _KJC_PRINT_
```

```
                         printf ("kframe = %d\n",kframe);
#endif
                         if (kframe > kinfo->frames){
                              printf("kinematic data exceeded\n");
                              scanf("%s",reply);
                            }
                         fread(data,sizeof(float),16*nseg,fdata);
                         if (fpdata != NULL)
                            fread(fpd,sizeof(float),6*nfp,fpdata);


#ifdef _KJC_PRINT_
                         printf ("Read 1");
#endif
                         for (j=curframe;j<kframe;j++){
/*        Increment pointer past the stuff don't want to use                    */
                              num =fread(data,sizeof(float),16*nseg,fdata);
                              if (fpdata != NULL)
                                      fread(fpd,sizeof(float),6*nfp,fpdata);
                             }
/*        Increment curframe now for next i-loop */
                         curframe = kframe+1;
                         }
               else {
                         fread(data,sizeof(float),16*nseg, fdata);
                         if (fpdata != NULL)
                            fread(fpd,sizeof(float),6*nfp,fpdata);
                    }
               data += 16*2; /* Increment pointer to femoral array(seg3)*/
               for (j=0;j<2;j++){          /* Segment loop  */
                         for (k=0;k<3;k++){
                                   array_pos[j][k] =  *data;
                                   ++data;
                                   }
                         data+=4;  /* Pass the 4 unidentified numbers */
                         for (k=0;k<3;k++){
                                   for (l=0;l<3;l++){
                                        array_rot[j][k][l] =  *data;
                                        data++;
                                        }
                                }
                        }
                data = data_hold;
/*      Multiply the array orientation matrix into body segment coord.*/
                for (j=0;j<2;j++){    /* Segment loop */
                        psmt2 = &smt2[j][0][0];
                        psegrot = &seg_rot[j][0][0];
                        parrayrot = &array_rot[j][0][0];

                        if (array_pos[j][0]!=bad && array_pos[j][1]!=bad
                           && array_pos[j][2]!=bad){
                                mult_matrices(psegrot,parrayrot,psmt2,3,3,3);
                                for (k=0;k<3;k++){
                                        for (l=0;l<3;l++){
                                                smt[j][k][l]=smt2[j][k][l];
/*                                              printf("%f ",smt[j][k][l]);*/
                                                }
                                        smt[j][k][3] = 0.0;
/*                                      printf("%f\n",smt[j][k][3]);*/
                                        }
                                for (l=0;l<3;l++){
                                      smt[j][3][l] = 0.0;
/*                                              printf("%f ",smt[j][3][l]);*/
                                    }
                                smt[j][3][3] = 1.0;  /* Good data */
/*                                              printf("%f\n",smt[j][3][3]);*/
                                }
                        else
                           smt[j][3][3] = 0.0;  /* Bad data */

/*                                              printf("%f\n",smt[j][3][3]);*/
                        }
```

```
/*      Do angle calculation elsewhere */

                psmt = &smt[0][0][0];
#ifdef _KJC_PRINT_
                printf ("Going to hip_ang\n");
#endif
/*              hip_ang ( psmt, flex, abd, extrot, side); */
                mgh_ang( psmt, flex, abd, extrot, side);
#ifdef _KJC_DEBUG_
                fprintf(debug,"%lf %lf %lf\n",*flex, *abd, *extrot);
#endif
                if (*flex !=bad && *abd!=bad && *extrot!=bad) {
/*      Store in a_trans */
                        current->flexion = *flex;
                        current->abduction = *abd;
                        current->extrotation = *extrot;
                        current->time = ((double) i)/freq;

/*      Scale the FP data by 100/body wt in Newtons (as at MGH) */

                        if (fpdata != NULL){
                                fpd++;
                                current->fp1_vert = -(*fpd)*100./wgtnt;
                                fpd+=6;
                                current->fp2_vert = -(*fpd)*100./wgtnt;
                                fpd-=7;
                                }
                        nsynch++;
/*      Set up the linked list and update "current" */

                        hold = (trans_a _huge *)halloc (1,
                                    ( unsigned long) sizeof(trans_a));
                        if (NULL == hold){
                                printf("Out of memory");
                                while (current->prev){
                                        hold = current->prev;
                                        hfree(current);
                                        current = hold;
                                        }
                                hfree(current);
                                free(data);
                                abort;
                                }
                        current->next = hold;
                        hold->prev = current;
                        hold->next = NULL;
                        current = hold;
                        }
                }
        free(data);
        if (fpdata != NULL) free(fpd);
        pinfo->synch_frames = nsynch;
        kinfo->synch_frames = nsynch;
#ifdef _KJC_DEBUG_
        fclose(debug);
#endif
        return;
}
```

MGH_ANG.C

```
/**********************************************************/
/*      hip_ang ---                                       */
/*              Function to get flex, abd, extrot         */
/*              given array orientation in body segmen */
/*              coordinates.                              */
/*                                                        */
/*      Dec. 22, 1992     K.C.                            */
/**********************************************************/

#include <stdio.h>
```

```
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

#define PI       3.1415927
#define BAD      -1000.0

void mgh_ang (float *psmt, double *flex, double *abd, double *extrot,
          char side)
{
          int i, j, k, l;
          float smt[2][4][4];
          float distal[3][3];
          float proximal[3][3];
          float result[3][3];
          float *dist, *prox, *res;
          double f_vect[3],numerator,denominator;
          double scale = 1.0;
          double flexion,abduc, extrotation,factor;

          if (side != 'R') scale = -1.0;
/*                  Fill smt - too lazy to keep track of pointer */
          for (i=0;i<2;i++){
                    for (j=0;j<4;j++){
                              for (k=0;k<4;k++){
                                        smt[i][j][k] = *psmt;
                                        psmt++;
                                        }
                              }
                    }
/*                  Check for bad data              */

          if ( (smt[0][3][3] == 0.0) || (smt[1][3][3] == 0.0)){
                    *flex = BAD;
```

```
                *abd = BAD;
                *extrot = BAD;
                return;
                }

/*      Transpose the proximal matrix in prep for matrix mult. */

        for (i=0;i<3;i++){
                for (j=0;j<3;j++){
                        distal[i][j] = smt[0][i][j];
                        proximal[j][i] = smt[1][i][j];
                        result[i][j] = 0.0;
                        }
                }

        dist = &distal[0][0];
        prox = &proximal[0][0];
        res = &result[0][0];

/*      Here - switch to MGH formulation of angles */
/*      Wasn't too sure of answers I was getting with */
/*      MIT formulation.           */

        mult_matrices( dist, prox, res, 3,3,3);
        abduc = asin(result[1][2]);
/*      printf("%lf %f\n",abduc, result[1][2]); */
        if (abduc != (PI/2.) && abduc!= (-PI/2.)){
           factor = (double) result[1][0]/cos(abduc);
           if (factor < -1.0) factor = -1.0;
           if (factor > 1.0) factor = 1.0;
           flexion = asin(factor) * scale;
           if (result[1][1] < 0.0) flexion = PI - flexion;
           if (flexion > PI) flexion = flexion - 2*PI;
           factor = (double) result[0][2]/cos(abduc);
           if (factor<-1.0) factor = -1.0;
           if (factor > 1.0) factor = 1.0;
```

```
            extrotation = asin(factor);
            if (result[2][2] < 0.0) extrotation = PI - extrotation;
            extrotation = extrotation * scale;
            if (extrotation > PI) extrotation = extrotation - 2*PI;
            }
        else {
            extrotation = 0.0;
            flexion = asin((double)result[0][1]);
            }
        *flex = flexion*180.0/PI;
        *abd = abduc*180.0/PI;
        *extrot = extrotation*180.0/PI;
#ifdef _KJC_PRINT_
        fprintf ("%1f %1f %1f\n", *flex, *abd, *extrot);
#endif
        return;
        }
```

XFORM.C

```
/*********************************************************/
/*      xform ---                                        */
/*              function to transform transducer         */
/*      locations into acetabular locations.             */
/*      Clinical joint angles are obtained earlier       */
/*      and are stored for each frame in the trans_a     */
/*      structure.  Joint angles are from Grood &        */
/*      Suntay defn's.                                   */
/*                                                       */
/*              Bulk of the program from TRANS(in FORTAN*/
/*      by Bob Fijan and me - but may be significant     */
/*      alterations for use with left as well as right   */
/*      hips and because am working from notes made      */
/*      a couple years ago - reworked TRANS and found    */
/*      some bugs/better ways to do things.              */
/*                                                       */
/*                                                          */
/*      Dec. 10, 1992     K.C.                              */
/************************************************************/

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

#define PI      3.1415927

void xform (FILE *outfile,trans_a _huge **a_entry, struct datainfo *pinfo,
        struct kineinfo *kinfo, femoral_ang *f_ang, double freq)
{
        char hip, reply[4];
        int i, j, frames;
        trans_a _huge *base;
        trans_a _huge *current;
        trans_a _huge *hold;
        double flex, abd, rot;
        double phi,theta;

        base = *a_entry;
        current = base;
        hip = pinfo->hip;

        if (pinfo->freq > freq ) frames = kinfo->synch_frames;
        else frames = pinfo->synch_frames;
#ifdef _KJC_PRINT_
   printf("frames = %d; ksynch_f = %d; psynch_f = %d\n",frames,
                kinfo->synch_frames,pinfo->synch_frames);
        scanf("%s",reply);
#endif
/*              First calculate the matrix products that            */
/*                  are constant for all frames.                     */
/*        pos. acetab. = [c] [c] [c] [c] [position matrix]          */
/*                          ha  fh  sf  ps            prosth         */

        calc_matrices (outfile,pinfo, kinfo, f_ang);



/*                  Convert to radians          */

        for (i=0; i<frames; i++){
                flex = current->flexion * PI/180.0;
                abd = current->abduction * PI/180.0;
                rot = current->extrotation * PI/180.0;
                hip_xform (outfile, flex, abd, rot, kinfo, current, pinfo);
#ifdef _KJC_PRINT_
                printf ("%d, flex = %lf\n",i,flex);
#endif
                hold = current->next;
                if (hold == NULL){
                    printf("gone off end of list\n");
                      return;
                  }
                else current = hold;
                }
        return;
}
```

HIP_XFOR.C

```
/***********************************************************/
/*      hip_xform ---                                       */
/*              function to transform transducer            */
/*      locations into acetabular locations.                */
/*      Clinical joint angles are obtained earlier          */
/*      and are stored for each frame in the trans_a        */
/*      structure.  Joint angles are from Grood &           */
/*      Suntay defn's.                                      */
/*                                                          */
/*              Bulk of the program from TRANS(in FORTAN*/
/*      by Bob Fijan and me - but may be significant        */
/*      alterations for use with left as well as right      */
/*      hips and because am working from notes made         */
/*      a couple years ago - reworked TRANS and found       */
/*      some bugs/better ways to do things.                 */
/*                                                          */
/*                                                          */
/*      Dec. 10, 1992     K.C.                              */
/**********************************************************/

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"

#define PI      3.1415927


void hip_xform (FILE *outfile,double flex, double abd, double rot, struct kineinfo *kinfo,
               trans_a _huge *current, struct datainfo *pinfo)
{
        char hip,reply[4];
        double u[3],c[3][3],rad,theta;
        double f[3],a[3],c2[3][3];
        int i,j;

        hip = kinfo->side;
        for (i=0;i<2;i++){
          for (j=0;j<2;j++){
            c[i][j] = 0.0;
            c2[i][j] = 0.0;
          }
          u[i] = 0.0;
          f[i] = 0.0;
          a[i] = 0.0;
        }

/*      Calculate the rotation matrix first */
/* Worked this out for a right leg in 2/90...is also different */
/* from HTRANS.  */

        c[0][0] = cos(rot)*cos(flex);
        c[0][1] = cos(rot)*sin(flex) + sin(rot)*sin(abd);
        c[0][2] = sin(rot)*cos(abd);
        if (hip == 'L') c[0][2] = -c[0][2];

        c[1][0] = -cos(abd)*sin(flex);
        c[1][1] = cos(abd)*cos(flex);
        c[1][2] = -sin(abd)*cos(flex);
        if (hip == 'L') c[1][2] = -c[1][2];

        c[2][0] = -sin(rot)*cos(flex);
        c[2][1] = -sin(flex)*sin(rot) + cos(rot)*sin(abd);
        c[2][2] = cos(rot)*cos(abd);
        if (hip == 'L'){
                c[2][0] = -c[2][0];
                c[2][1] = sin(flex)*sin(rot) - cos(rot)*sin(abd);
                }
#ifdef _X_DEBUG_
        fprintf(outfile,"Rotation matrix:  femoral to hip\n");
      for (j=0;j<2;j++){
          fprintf(outfile,"%lf, %lf, %lf\n",c[0][j],c[1][j],c[2][j]);
        }
#endif
```

```
/*      Now get c2 (pelvis_to_acet_rotation matrix) from kineinfo */

        for (j=0;j<3;j++){
                for (i=0;i<3;i++){
                        c2[j][i] = kinfo->pelvis_to_acet_rot[j][i];
                        }
                }

/* Now rotate for each transducer */

        for (j=0;j<pinfo->ntrans;j++){
                for (i=0;i<3;i++){
                        f[i] = kinfo->prosth_to_femoral_rot[j][i];
/*                      if (j==2) printf("f[%d] = %lf\n",i,f[i]);*/
                        }

/*      Now need to go into pelvis coord using the clinical */
/*      joint angles.   Assume angles in radians.           */


                u[0] =  c[0][0]* f[0] + c[1][0] * f[1] + c[2][0] * f[2];
                u[1] =  c[0][1] * f[0] + c[1][1] * f[1] + c[2][1] * f[2];
                u[2] =   c[0][2] * f[0] + c[1][2] * f[1] + c[2][2] * f[2];
#ifdef _X_DEBUG_
                fprintf(outfile,"Point %d\n",j);
                fprintf (outfile,"Pelvis coord:   %lf, %lf, %lf\n",u[0],u[1],u[2]);
#endif


/* Now change to acetabular coordinates */
/* Use the rotation matrix calculated in calc_matrices and stored */
/* in pelvis_to_acet_rot in structure kineinfo */
/* Transposed relative to the previous matrix */


                a[0] = c2[0][0] * u[0] + c2[0][1] * u[1] + c2[0][2] * u[2];
                a[1] = c2[1][0] * u[0] + c2[1][1] * u[1] + c2[1][2] * u[2];
                a[2] = c2[2][0] * u[0] + c2[2][1] * u[1] + c2[2][2] * u[2];
#ifdef _X_DEBUG_
                fprintf(outfile,"Acetabular coord:  %lf, %lf, %lf\n",a[0],a[1],a[2]);
#endif

/*      Now change [a] into an acetabular phi and theta and store as degrees */
/*      For a left hip, phi measured from pos Za axis -- so change sign there*/
                if (hip == 'L') a[2] = - a[2];
                current->alltrans[j].ang.phi = (acos(-1 * a[2])) * 180./PI;
                rad = sqrt((a[0]*a[0] + a[1]*a[1]));
                theta = acos((a[0]/rad)) * 180./PI;
                if (a[1] < 0) theta = -theta;
                current->alltrans[j].ang.theta = theta;
/*              if (a[0] > 0.0){
                    current->alltrans[j].ang.theta =
                        atan2(a[1],a[2])*180./PI;
                    }
                else {
                    current->alltrans[j].ang.theta =
                        atan2( a[0], a[1]) * 180./PI;
                    } */
#ifdef _X_DEBUG_
                fprintf(outfile,"Acetabular phi, theta:  %lf, %lf\n",\
                        current->alltrans[j].ang.phi,        \
                        current->alltrans[j].ang.theta);
#endif
                }
        return;
}
```

CALC_MAT.C

```
/*********************************************************/
/*      calc_matrices                               */
/*              Calculates the constant matrices     */
/*      and stores their product in the kineinfo              */
/*      structure.                                            */
/*                                                            */
/*              Bulk of the program from TRANS(in FORTAN*/
/*      by Bob Fijan and me - but may be significant          */
/*      alterations for use with left as well as right        */
/*      hips and because am working from notes made           */
/*      a couple years ago - reworked TRANS and found         */
/*      some bugs/better ways to do things.                   */
/*                                                            */
/*                                                            */
/*      Dec. 10, 1992     K.C.                                */
/************************************************************/

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"

#define PI        3.1415927

void calc_matrices ( FILE *outfile, struct datainfo *pinfo,
                     struct kineinfo *kinfo, femoral_ang *f_ang)
{

        char hip;
        int i, j;
        double phi, theta,dtr,rtd;
        double angle[6];                /* Need to allow to be different*/
                                    /* May want to put in datainfo */
                                    /*      Assume are in radians      */
        double u[3],s[3],c[3][3],tan1,tan2,tan3;
        double f[3],a[3];

        hip = kinfo->side;
        dtr = PI/180.;
        rtd = 180./PI;
        angle[0] = 25.0*dtr;
        angle[1] = 7.0*dtr;
        angle[2] = 10.0*dtr;
        angle[3] = 0.0*dtr;
        angle[4] = 20.0*dtr;
        angle[5] = 30.0*dtr;
        for (i=0;i<6;i++){
           if ((angle[i] == PI/2.) || (angle[i] == -PI/2.))
                 angle[i] = (angle[i] - .001);
           }

#ifdef _X_DEBUG_
        fprintf(outfile,"%c\n",hip);
#endif


/*              First calculate the matrix products that              */
/*              are constant for all frames.                          */
/*      pos. acetab. = [c] [c] [c] [c] [position matrix]              */
/*                     ha  fh  sf  ps          prosth                 */

/*              Convert to radians          */

/*      Calculate matrix for transformation into femoral coordinates          */
/*    This part majorly revised from TRANS - */
/*      found errors there, (at least I think        */
/*      they were errors                             */
/*              Uses angles that define prosthesis */
/*              orientation in femur                 */
/*    Note that Zfemoral points laterally for a right hip */
/*      but medially for a left hip.                 */
/*    Matrix notation is transpose of that usually use, just */
/*    happened to write that way when calculating elements */

      tan1 = tan(angle[1]);
```

```
        tan2 = tan(angle[2]);
        tan3 = tan(angle[3]);
        c[1][1] = 1.0/(sqrt(1.0 + tan1*tan1 + tan3*tan3));
        c[1][0] = -tan3 * c[1][1];
        c[1][2] = tan1 * c[1][1];
        if (hip == 'L') c[1][2] = -c[1][2];

        c[2][2] = 1.0 /(sqrt(1.0 + tan1*tan1 + tan2*tan2));
        c[2][0] = tan2 * c[2][2];
        c[2][1] = -tan1 * c[2][2];
        if (hip == 'L') {
                c[2][0] = - c[2][0];
                c[2][1] = - c[2][1];
                }

        c[0][0] = c[1][1] * c[2][2] - c[1][2] * c[2][1];
        c[0][1] = c[1][2] * c[2][0] - c[1][0] * c[2][2];
        c[0][2] = c[1][0] * c[2][1] - c[1][1] * c[2][0];



/*      Calculate the position matrix for transducers */

        for(j=0; j<pinfo->ntrans; j++){
                phi = f_ang->ang[j].phi * PI/180.0;
                theta = f_ang->ang[j].theta * PI/180.0;

/*              Define a vector from center of prosthesis head to */
/*              the point in question in prosthesis coordinates      */
/*                      Let theta = 0 define X direction for a R hip */
/*                      Let theta = 180 define X for a L hip        */
/*                      Let theta = 90 define Y in both cases */
/*                      u[3] ends up pointing in opp. direction */

                if (hip == 'R') {
                        u[0] = sin(phi) * cos(theta);
                u[2] = -cos(phi);
                }
        if (hip == 'L') {
                u[0] = -sin(phi) * cos(theta);
                u[2] = cos(phi);
                }
        u[1] = sin(phi) * sin(theta);

/*      Go to "stem" coordinates next                   */
/*      Use the angle[0] - of head to stem */

        s[0] = u[0];
        if (hip == 'R') {
                s[1] = u[1] * sin(angle[0]) - u[2] * cos(angle[0]);
                s[2] = u[1] * cos(angle[0]) + u[2] * sin(angle[0]);
                }
        if (hip == 'L') {
                s[1] = u[1] * sin(angle[0]) + u[2] * cos(angle[0]);
                s[2] = - u[1] * cos(angle[0]) + u[2]*sin(angle[0]);
                }

/*      Now transform into femoral coordinates          */
/*      Use matrix calculated above.                    */

        f[0] = c[0][0]* s[0] + c[1][0] * s[1] + c[2][0] * s[2];
        f[1] = c[0][1]* s[0] + c[1][1] * s[1] + c[2][1] * s[2];
        f[2] = c[0][2]* s[0] + c[1][2] * s[1] + c[2][2] * s[2];

#ifdef _X_DEBUG_
    fprintf(outfile,"Point %d\n"j);
    fprintf(outfile," Prosthesis coord:  %lf, %lf, %lf\n",u[0],u[1],u[2]);
    fprintf(outfile," Stem coord:  %lf, %lf, %lf\n",s[0],s[1],s[2]);
    fprintf(outfile," Femoral coord:  %lf, %lf, %lf\n",f[0],f[1],f[2]);
#endif

/*      Store this information in kineinfo in prosth_to_femoral_rot */
```

```
			for (i=0;i<3;i++){
				kinfo->prosth_to_femoral_rot[j][i] = f[i];
				}
			}

/*	Calculate the rotation matrix between the pelvis coordinates */
/*	and the acetabular coordinates		*/

		c[0][0] = cos(angle[4]);
		c[0][1] = 0.0;
		c[0][2] = -sin(angle[4]);

		c[2][2] = 1.0+tan(angle[4])*tan(angle[4])+tan(angle[5])*tan(angle[5]);
		c[2][2] = 1.0 / sqrt(c[2][2]);
		c[2][0] = tan(angle[4]) * c[2][2];
		c[2][1] = -tan(angle[5]) * c[2][2];
		if (hip == 'L') {
			c[2][0] = -c[2][0];
			c[2][1] = -c[2][1];
			c[0][2] = -c[0][2];
			} /* Yes, OK to do here, signs work out the same */


		c[1][0] = c[2][1] * c[0][2] - c[2][2] * c[0][1];
		c[1][1] = c[2][2] * c[0][0] - c[2][0] * c[0][2];
		c[1][2] = c[2][0] * c[0][1] - c[2][1] * c[0][0];

#ifdef _X_DEBUG_
		fprintf (outfile,"Rotation matrix:  hip to acetabular\n");
		for (i=0;i<3;i++){
			fprintf(outfile,"%lf, %lf, %lf\n",c[0][i],c[1][i],c[2][i]);
		  }
#endif


		for (i=0;i<3;i++){
```

```
		for (j=0;j<3;j++){
			kinfo->pelvis_to_acet_rot[i][j] = c[i][j];
			}
		}
	return;

}
```

MULT_MAT.C

```
/**********************************************************/
/*	mult_matrices	---				*/
/*		function to multiply two matrices.	*/
/*	Probably should make it general -- is taking	*/
/*	the place of GMPRD -- old Fortran subroutine	*/
/*							*/
/*	Dec. 23, 1992	K.C.				*/
/**********************************************************/

#include <stdio.h>
#include <math.h>



void mult_matrices (float *m1, float *m2, float *mresult,
	int row1, int col1, int col2)
{
	int i, j, k;
	float *base1, *base2, *baser;

#ifdef _KJC_PRINT_
	printf("hi.\n");
#endif
	base1 = m1;
	base2 = m2;
	baser = mresult;
```

```
        for (i=0;i<row1;i++){
                for (j=0;j<col2;j++){
                        *mresult = 0.0;
                        for (k=0;k<col1;k++){
                                *mresult = *mresult + (*m1)*(*m2);
                                m1++;
                                m2 += col2;
                                }

                        mresult++;
                        m1 -= col1;
                        m2 -= col1*col2;
                        m2++; /* set at start of next column */
                        }
                m1 +=col1; /* set m1 at start of next row */
                m2 -= col2; /* set m2 at start of first column */
                }
        m1 = base1;     /* Return pointers to start position */
        m2 = base2;
        mresult = baser;
        return;
}
```

ASTOR_PR.C

```
/*******************************************************************/
/*      astor_prs ---                                              */
/*              function to read .PRS files in the                 */
/*              new data format and transfer                       */
/*              the data to trans_a linked list.                   */
/*              Should give                                        */
/*              option of normalizing as well (w.r.t BW/H^2?)      */
/*                                                                 */
/*      Dec. 8, 1992    K.C.                                       */
/*******************************************************************/
```

```
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include "p_file.h"
#include "kinemat.h"

#define PSITOMPA  0.0068948
#define END 10000   /*Define END a number will never get more frames than*/


void astor_prs (FILE *prsfile, struct datainfo *pinfo, struct kineinfo *kinfo,
         trans_a _huge **a_entry)
{
       char sdata[10], reply[4];
          int i, j, k, skip, pframe, kframe, frames, curframe;
          trans_a _huge *base;
          trans_a _huge *hold;
          trans_a _huge *current;
          double time, rem;
          double pf, *intptr;
          double data;

          base = *a_entry;
          current = base;
          intptr = &pf;
          data = 0.0;

/*                 Read the data then convert to MPa                    */
/*                 Read .INT files -- already in psi                    */
/*                 Only deal with 43 for now!  Requires lots of         */
/*                 changes if alter later.                              */

/*                 File pointer is set at first synched frame           */
/*                 Read only the synched frames.                        */
/*              Need to deal with skipping frames here if kfreq */
/*              is less than pfreq (else in get_dtl and xform)        */

        if ( kinfo->freq < pinfo->freq){
                skip = 1;
                frames = kinfo->synch_frames;
                curframe = 0;
                }
        else {
                skip = 0;
                frames = pinfo->synch_frames;
                }
        for (i=0;i<END;i++) {
                time = current->time;
                rem = modf ((time * pinfo->freq), intptr);
                pframe = (int)(pf);
                if (rem > 0.5) pframe++;
                if (skip){
                        for (k=curframe;k<pframe;k++) {

/* Remember - this is set for prosth #43 */
                                for (j=0;j<18; j++){
                                        fscanf(prsfile,"%*f");
                                        }
                                if (pinfo->adchan == 2){
                                        for (j=0;j<16;j++){
                                                fscanf (prsfile,"%*d");
                                                }
                                        }
                                }
                        curframe = pframe+1; /*Increment curframe now*/
                        }
                for (j=0; j<18; j++){
                        if (j>2 && j<16 && j!=6){
                                fscanf(prsfile,"%s",sdata);
                                if (sdata[0] == '*') sdata[0] = ' ';
```

```
                                        sscanf(sdata,"%lf",&data);
                                        data = data * PSITOMPA;
                                        current->alltrans[j-3].mpa = data;
                                        data = 0.0;
                                        }
                                else fscanf (prsfile,"%*f");
                                }
                        if (pinfo->adchan == 2){
                                for(j=0;j<16;j++){
                                        fscanf(prsfile,"%*d");
                                        }
                                }

#ifdef _KJC_PRINT_
                        if (i<10)  printf("%d\n",i);
#endif
                        hold = current->next;
                        current = hold;
                        if (current->next == NULL){
                                i = END-1;
                          }
                        }

/* Close the file */
                fclose(prsfile);
                return;
}
```

AWRITE_T.C

```
/*****************************************************************/
/*      awrite_tec ---                                            */
/*              writes output file suitable for entry to          */
/*      Tecplot (I hope!).  This version writes acetabular info */
/*      writes: t, trans#, phi, theta, and pressure (MPa)         */
/*                                                                */
/*      Now also writes two other files -- joint angles and          */
/*      forceplate data.  Need to figure out not to write these */
/*      files if that data wasn't available.                 */
/*                                                           */
/*      Dec. 28, 1992     K.C.                                */
/*****************************************************************/

#include <stdio.h>
#include <string.h>
#include "p_file.h"
#include "kinemat.h"


void awrite_tec (trans_a _huge **a_entry, struct datainfo *pinfo,
                 struct kineinfo *kinfo, FILE *parfile)
{
        int i,j,k,nframe;
        double time, phi, theta;
        FILE *outfile, *angfile, *fpdfile;
        char filename[40],angname[40],fpdname[40];
        trans_a _huge *base;
        trans_a _huge *current;
        trans_a _huge *hold;

        nframe = pinfo->synch_frames;
        if (pinfo->synch_frames == 0){
                nframe = kinfo->synch_frames;
                }
        base = *a_entry;
        current = base;

/*      printf ("Enter the name of the output file [ext. will be .tec]: ");*/
        fscanf (parfile,"%s",filename);
        strcpy(angname,filename);
        strcat(angname,".ang");
        strcpy(fpdname,filename);
```

```
strcat(fpdname,".fp");
strcat(filename,".tec");
outfile = fopen(filename,"w");
angfile = fopen(angname,"w");
fpdfile = fopen(fpdname,"w");



fprintf(outfile,"TITLE = ""%s""\n",pinfo->name);
fprintf(outfile,"VARIABLES = t, trans, phi, theta, P\n");

fprintf(angfile,"TITLE = Angles%s\n",angname);
fprintf(angfile,"VARIABLES = t, flex, abd, extrot\n");

fprintf(fpdfile,"TITLE = Vertforce%s\n",fpdname);
fprintf(fpdfile,"VARIABLES = t, y_fp1, y_fp2\n");

for (j=0;j<(pinfo->ntrans);j++){
        fprintf(outfile,"ZONE T=Trans%d, I= %d, F= POINT\n", \
              (j+1),nframe);
        for (i=0;i<nframe;i++){
                fprintf(outfile,"%lf %lf %lf %lf %lf\n",current->time,
                        ((double) j),current->alltrans[j].ang.phi,
                        current->alltrans[j].ang.theta,
                        current->alltrans[j].mpa);
                hold = current->next;
                current = hold;
                }
        current = base;
        }
fclose(outfile);

/*      Write the file with joint angles in it */

fprintf(angfile,"ZONE T=ANGLES, I = %d, F = POINT\n",nframe);
for (i=0;i<nframe;i++){
```

```
                fprintf(angfile,"%lf %lf %lf %lf\n",current->time,\
                   current->flexion,current->abduction, current->extrotation);
                hold = current->next;
                current = hold;
                }
        current = base;
        fclose(angfile);

/*          Write the file with forceplate forces in it */

        fprintf(fpdfile,"ZONE T=FP_FORCES, I = %d, F = POINT\n",nframe);
        for (i=0;i<nframe;i++){
                fprintf(fpdfile,"%lf %f %f\n",current->time,\
                   current->fp1_vert,current->fp2_vert);
                hold = current->next;
                current = hold;
                }
        current = base;
        fclose(fpdfile);

        return;
}
```

E.3.2 For data from prosthesis 33

Include files for KINE2.EXE

```
/************************************************/
/*    kinemat.h                          */
/*          include file for "locate"    */
/*                                       */
/*    Dec. 5, 1992    K.C.               */
/************************************************/


#define NTRANS  13
```

```
#define MAXSEG  10

typedef struct angles {
          double phi;
          double theta;
          } ang;

typedef struct f_angles {
          struct angles ang[NTRANS];
     } femoral_ang;

typedef struct a_trans {
          struct angles ang;
          double mpa;
          };



typedef struct a_frame {
          struct a_frame *next;          /*  For doubly linked list */
          struct a_frame *prev;
          double time;
          double flexion, abduction, extrotation;/*clinical jt angs at hip*/
          struct a_trans alltrans [NTRANS]; /* phi,theta, and pressure */
          float fp1_vert, fp2_vert;    /*Scaled FP Y forces */
          } trans_a;

/*    If using femoral view: no need to duplicate phi and theta */

typedef struct f_frame {
          struct f_frame *next;  /* For creating a linked list */
          struct f_frame *prev;
          double mpa [NTRANS];
          } trans_f;

/* trans_a _huge *a_entry;  /* anchor for a_frame list */
/* trans_f _huge *f_entry;  /* anchor for f_frame list */
```

```
femoral_ang *f_ang;


typedef struct kineinfo {
        char name[12];          /* File name */
        char descr[40];         /* Description */
        char side;              /* Right or Left */
        int nseg;               /* Number of segments collected */
        int frames;             /* frames of kinematic data taken */
        int synch_frames;       /* # of synched frames (w/pressure)*/
        int nfp;                /* # of forceplates used */
        float wgt;
        double freq;            /* frequency of kinematic data */
        double prosth_to_femoral_rot [NTRANS][3]; /* Rot matrix - transducer*/
                                /*      position in femoral coordinates */
        double pelvis_to_acet_rot [3][3]; /* Rot matrix -pelvis to acet. */
        float seg_vec [MAXSEG][3];      /* Segment vectors from kine header*/
        float seg_rot [MAXSEG][3][3];   /* Segment rot. matrix from T4header*/
        float array_pos [2][3];
        float array_rot [2][3][3];
        };

typedef struct pzeros {
        float offset[16];
        float gain[16];
        float tempcorr[16];
        float temp;
        };
```

```
/* Header file for use with programs that access prosthesis */
/* data files              Jan. 17, 1990                    */
```

```
#define GLOBAL
#define SEMIGLOBAL static  /* used to define a semiglobal variable*/
#define IMPORT extern  /* used to refer to a global or semiglobal var*/
                       /* referenced elsewhere */


#define SHORTSIZE 9
#define PATHSIZE 80
#define PROSTH_IDSIZE 3
#define DESCR_SIZE 32
#define DIRNAMESIZE 50
#define NAMESIZE 30


struct datainfo {
        char name[9];
        char id[3];
        int samp;
        int adchan;
        int mult;
        int trig;
        char time[5][12];
        char descr[32];
        int startoff;
        int frames;
        int alarmch;
        int synchch;
        double freq;
        int ntrans;  /* number of transducers */
        int date[3];
        char hip;
        int synch_frames; /* Number of frames TSYNCH active */
        int binary;
        };
```

```
SEMIGLOBAL char short_dataname [SHORTSIZE] = "";
SEMIGLOBAL char prosth_id [PROSTH_IDSIZE] = "";
SEMIGLOBAL char description[DESCR_SIZE] = "";
SEMIGLOBAL char namdir[DIRNAMESIZE] = "";
```

```
/* function prototypes */

float floatflt( char *);
void write_stat (int );
void error(char *);
void get_3ds (struct kineinfo *, FILE *);
fpos_t *get_dat (struct kineinfo *, FILE *);
int read_dat (struct kineinfo *, FILE *,fpos_t *, int, int);
void read_fpd (struct kineinfo *, FILE *, int, trans_a *, fpos_t *, int);
int bread_prs (struct kineinfo *, struct datainfo *, FILE *,
                struct pzeros *, int, int, trans_a *, fpos_t *);
int aread_prs (struct kineinfo *, struct datainfo *, FILE *,
                struct pzeros *, int, int, trans_a *, fpos_t *);
void write_tmp(struct kineinfo *, struct datainfo *, FILE *,
                trans_a *, fpos_t *);

void raw_to_mpa (int *, double *, struct datainfo *,float *, float *,
               float *, float);
float zero33 (float *, float *, float *, struct datainfo *);
femoral_ang *f_locate (struct datainfo *);
void read_tmp(struct kineinfo *, struct datainfo *, FILE *, trans_a *, fpos_t *);

void mgh_ang (float *, trans_a *, char *);
void mult_matrices(float *, float *, float *, int, int, int);

void calc_jta (struct kineinfo *, struct datainfo *, trans_a *);

void calc_xfm (struct kineinfo *, struct datainfo *, trans_a *);
void calc_matrices(struct kineinfo *, struct datainfo *, femoral_ang *);
void awrite_tec(struct kineinfo *, struct datainfo *, trans_a *, char *,
                fpos_t *, fpos_t *, fpos_t *, FILE *, FILE *, FILE *, int);
```

KINE33.C

```
/************************************************************************/
/*      kine33                                                         */
/*          Program to determine transducer locations and pressures */
/*          for prosthesis 33 data.                                    */
/*                                                                     */
/*      2/24/93 K.C.                                                   */
/************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <string.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

int write_count = 0;
int read_count = 0;


void main( int argc, char *argv[])
{
    char paramfile[80],filename[80],prsname[80],datname[80];
    char fpdname[80],name3ds[80],angname[80],fpname[80];
    char pformat[2];

    struct datainfo pdata;
    struct datainfo *pinfo;
    struct kineinfo kdata;
    struct kineinfo *kinfo;
    struct pzeros zeros;
    struct pzeros *pzero;

    int i, curframe, next, err, frames, kframe, knext;
    trans_a data_frame, *thisframe;
    fpos_t *fpos,*fapos,*ffpos, *fopos;
    float time;

    FILE *parfile = NULL;
    FILE *pfile= NULL;
    FILE *dfile= NULL;
    FILE *ffile= NULL;
    FILE *file3= NULL;
    FILE *tmpfile= NULL;
    FILE *outfile,*angfile, *fpfile;

    if (argc < 2){
            printf("Enter the name of the file with processing parameters:  ");
           scanf("%s",argv[1]);
           }
    strcpy (paramfile, argv[1]);
    parfile = fopen(paramfile, "r");

    pinfo = &pdata;
    kinfo = &kdata;
    pzero = &zeros;

/* Get filenames */

    fscanf(parfile,"%s",filename);
    fscanf(parfile,"%s",pformat);
    fscanf(parfile,"%d %d %d",&pinfo->date[0], &pinfo->date[1],
                  &pinfo->date[2]);
    fclose(parfile);
```

```
     strcpy (prsname,filename);
     strcpy (datname,filename);
     strcpy (fpdname,filename);
     strcpy (name3ds, filename);
     strcpy (angname, filename);
     strcpy (fpname, filename);
     strcat (prsname,".prs");
     strcat (datname,".dat");
     strcat (fpdname, ".fpd");
     strcat (name3ds, ".3ds");
     strcat (angname, ".ang");
     strcat (fpname,".fp");

/*   Read .3DS file */
     if( NULL != (file3 = fopen (name3ds, "rb"))) {
          get_3ds (kinfo, file3);
          fclose (file3);
     }
     else {
          printf("\n file %s could not be opened",name3ds);
     }

/*   Read header to .DAT file */
          if( NULL != (dfile = fopen (datname, "rb"))) {
            fpos = get_dat (kinfo, dfile);
          }
     else {
          printf("\n file %s could not be opened",datname);
     }


     /* thisframe = (trans_a *) calloc(1, sizeof(trans_a));                    */
     thisframe = &data_frame;

/* Frames loop, for all kinematic frames */
```

```
 if( NULL != (ffile = fopen (fpdname, "rb"))){
  if (pformat[0] == 'b'){
         pfile = fopen (prsname,"rb");
         fseek ( pfile, 0, SEEK_SET);
         fseek (ffile, 0, SEEK_SET);
        }
        else {
          pfile = fopen (prsname,"r");
          fseek (pfile, 0, SEEK_SET);
          fseek (ffile, 0, SEEK_SET);
        }
 }
 else {
        printf("\n file %s could not be opened",fpdname);
 }
if(NULL == pfile)
 printf("\n could not open file %s ",prsname);

 if( NULL == (tmpfile = fopen("temp.dat","wb"))) {
        printf("\n file temp.dat could not be opened ");
 }
 fseek (tmpfile, 0, SEEK_SET);
 if (kinfo->freq > 254.0){
        time = (kinfo->frames)/(kinfo->freq);
        frames = (int) (254 * time);
  }
  else frames = kinfo->frames;
  curframe = 0;
  kframe = 0;

  for (i=0;i<frames;i++){
          write_stat (1);
          fseek (dfile, 0, SEEK_CUR);
          err = fgetpos(dfile, fpos);
          knext = read_dat(kinfo, dfile,fpos, kframe, i);
          write_stat (2);
        fseek (ffile, 0, SEEK_CUR);
        err = fgetpos(ffile, fpos);
        read_fpd(kinfo, ffile, i, thisframe,fpos, kframe);
        write_stat(3);
        if (pformat[0] == 'b'){
            fseek (pfile,0, SEEK_CUR);
              err = fgetpos (pfile, fpos);
              next = bread_prs(kinfo, pinfo, pfile, pzero, i,
                        curframe, thisframe,fpos);
              write_stat(4);
             }
        else {
            fseek ( pfile, 0, SEEK_CUR);
              err = fgetpos (pfile, fpos);
            next = aread_prs(kinfo, pinfo, pfile, pzero, i,
                        curframe, thisframe, fpos);
             }
        fseek (tmpfile, 0, SEEK_CUR);
        err = fgetpos (tmpfile, fpos);
        write_tmp(kinfo, pinfo, tmpfile, thisframe,fpos);
        write_stat(5);
        curframe = next;
        kframe = knext;
        }
    fclose (tmpfile);
    fclose (dfile);
    fclose (pfile);
    fclose (ffile);

    printf(" Done with read loop \n");

/*      Open the files that will be read by TECPLOT */

    if( NULL == (outfile = fopen("a.out","w")))
       printf("\n file a.out could not be opened ");
    if( NULL == (angfile = fopen(angname,"w")))
```

```
        printf("\n file angname could not be opened ");
    if( NULL == (fpfile = fopen(fpname,"w")))
        printf("\n file temp.dat could not be opened ");

    fseek (outfile, 0, SEEK_SET);
    fseek (angfile, 0, SEEK_SET);
    fseek (fpfile, 0, SEEK_SET);

/*      Having trouble with program, try closing all files *
/*      and reopening, each frame */
    fclose(outfile);
    fclose(angfile);
    fclose(fpfile);

/*    Get the transducer locations from 33.ang file, used in transfm */
    f_ang = f_locate(pinfo);

/*    Calculate the transformation matrices which remain constant */
    calc_matrices( kinfo, pinfo, f_ang);

    *fpos = 0;
    for (i=0;i<frames;i++){

        if( NULL == (tmpfile = fopen("temp.dat","rb"))) {
            printf("\n file temp.dat could not be opened ");
            }
        if (0!=(err = fseek (tmpfile, *fpos, SEEK_SET))){
            printf("\n temp.dat file pointer not set correctly");
            fclose(tmpfile);
            exit(1);
            }
        err = fgetpos(tmpfile,fpos);
        read_tmp(kinfo, pinfo, tmpfile, thisframe, fpos);
        err = fgetpos (tmpfile,fpos);
        fclose(tmpfile);
        calc_jta(kinfo, pinfo, thisframe);
        calc_xfm(kinfo, pinfo, thisframe);
        if( NULL == (outfile = fopen("a.out","a")))
                printf("\n file a.out could not be opened ");
        if( NULL == (angfile = fopen(angname,"a")))
                printf("\n file angname could not be opened ");
        if( NULL == (fpfile = fopen(fpname,"a")))
                printf("\n file temp.dat could not be opened ");
        err = fgetpos ( outfile, fopos);
        err = fgetpos (angfile, fapos);
        err = fgetpos (fpfile, ffpos);

        awrite_tec(kinfo, pinfo, thisframe, filename, fopos, fapos, ffpos,\
                   outfile, angfile, fpfile, i);


        fclose(outfile);
        fclose(angfile);
        fclose(fpfile);
        }
}
```

FLOATFLT.C

```
/*******************************************************************/
/*      floatflt.c                              */
/*      function to read 4 byte reals written in PDP 11      */
/*      binary and figure out what the number is.          */
/*                                              */
/*      Feb. 19, 1993  K.C.                         */
/*******************************************************************/

#include <stdio.h>
#include <math.h>
#include "kinemat.h"
#include "fnc.h"

float floatflt(char *dat)
```

```
{

   short int i, h1, h2, exp;
   char data[4];
   int h3;
   unsigned long int frac, div;
   float sign, fout;
   float *fpo;


/*    printf ("\n inside floatflt");*/
   fpo = &fout;

   data[0] = *dat;
   data[1] = *(dat+1);
   data[2] = *(dat+2);
   data[3] = *(dat+3);

/*  byte order swapped  */

   h3 = (((short int) data[1]) << 8);
   h1 = h3 | (((short int) data[0]) & 0x00ff);
   h3 = (((short int) data[3]) << 8);
   h2 = h3 | ( ((short int) data[2] ) & 0x00ff) ;

/*  get the exponent from bits 14-6 in the first WORD */

   exp = ((0x7f80 & h1 ) >> 7) -128;
   if(exp == -128)
     exp = 0;

/*  convert the high bytes by shifting 16 bits and add to the low bytes */
/*  set the leading bit to 1 using 0x00800000 since is implied 24-bit # */

   frac = ( 0x00800000 | ((long) (h2 & 0x0080FFFF))) |
                  (((long) h1 & 0x007f) << 16) ;
```

```
    div = 0x00ffffff;


/*  make the fraction fout from the fraction integer */

    fout = ((float) frac)/ ((float) div);

/*    printf("\n exponent & fraction calculated "); */
    if( (fout >1.0 )|| (fout <0.5)){
     printf("\n fraction not normalized in floatflt-->%f",fout);
     printf("\n frac = %lx, data--> %x %x %x %x ",
                  frac,(int)data[0],(int)data[1],(int)data[2],(int)data[3]);
     error("\n exiting");
   }
   if( (exp < -127 ) || (exp>127)) {
    printf("\n exponent out of range in floatflt -->%d", exp);
    error("\n exiting");
   }
/*  set the sign bit    */

   if ((h1 & 0x8000) != 0x0000){
         fout = -1. * fout * pow((double) 2, (double) exp);
         }
   else{
         fout = fout * pow((double) 2, (double) exp);
         }


/*    printf("\n exiting floatflt, %f,",fout); */
   return(fout);

}
```

GET_3DS.C

```
/***************************************************************/
```

```
/*          get_3ds.c --                                  */
/*          function to read the .3ds file                */
/*          from old MGH data                             */
/*                                                        */
/*          Feb 17, 1993    K.C.                          */
/**********************************************************/

#include <stdio.h>
#include <malloc.h>
#include "kinemat.h"
#include "fnc.h"

void get_3ds(struct kineinfo *kinfo, FILE *data3)
{
   char iscal,leg;
   float *pdum;
   char *pchar;
   int i,j,k;
   char pchar_array[130];

   pchar = &(pchar_array[0]);
   /*    pchar = (char *)calloc( 128, sizeof(char));    */
   fread(pchar, sizeof(char), 128, data3);
   iscal = *pchar;
   pchar++;
   leg = *pchar;
   pchar--;

/*   Each record is 128 bytes long.   Header is 12 lines */

   for (i=0;i<12;i++){
         fread(pchar,sizeof(char),128,data3);
         }

/*   Next six lines are segment position vectors, each seg on a line */
```

```
 for (i=0;i<6;i++){
        pchar = &(pchar_array[0]);
        fread(pchar,sizeof(char),128,data3);
        for (j=0;j<3;j++){
              kinfo->seg_vec[i][j] = (floatflt (pchar));
              if (i == 3 || i == 4){
                   printf("%f  ",kinfo->seg_vec[i][j]);
              }
              pchar+=4;
              }


        }

/*    Next 4 records are segment rotation matrices.  Read in with */
/*    first index (besides the segment index) incrementing first */
/*    Is really the transpose I think - doing to match MGH procedure */
/*    in angplot_sub.F, easier to transpose now than add a step I guess */

 for (i=0;i<4;i++){
        pchar = &(pchar_array[0]);
        fread(pchar,sizeof(char),128,data3);
        for (j=0;j<3;j++){
           for (k=0;k<3;k++){
                  kinfo->seg_rot[i][k][j] = floatflt (pchar)  ;
                  if (i==3 || i== 4){
                    printf("%f  ",kinfo->seg_rot[i][k][j]);
                        }
                  pchar+=4;
                  }
           }


        }
        printf ("\n");
 /* free(pchar); */
 return;
 }
```

GET_DAT.C

```
/*********************************************************************/
/*      get_dat  ---                                                */
/*              function to get the .Dat file.      Not sure        */
/*      yet what to read or how.....                                */
/*                                                                  */
/*      Feb. 16, 1993    K.C.                                       */
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"



fpos_t *get_dat( struct kineinfo *kinfo, FILE *finl)
{
   char *pchar;
   char pchar_array[260];
   float *dat;
   fpos_t *fpos;
   int i, retval, err;
   int  nseg_l;
   int nframe;
   int nfp;

/*  Read the first "record" - discard.  Get info from 2nd*/
   pchar = &pchar_array[0];
   /*pchar = (char *) calloc(256, sizeof(char)); */
```

```
    for (i=0;i<2;i++){
      retval = fread(pchar,sizeof(char), 256, finl);
      err = fgetpos(finl, fpos);
    }
    pchar+=4;
    nframe = (int)  floatflt(pchar)  ;
    pchar+=4;

    kinfo->freq = (double) floatflt(pchar) ;
    pchar +=4;
    nseg_l = (int)  floatflt(pchar);
    pchar+=12;
    nfp = (int)  floatflt(pchar) ;
    kinfo->nseg = nseg_l;
    kinfo->nfp = nfp;
    kinfo->frames = nframe;
    kinfo->wgt = 150.;
    pchar = &pchar_array[0];
    strcpy(kinfo->side,"R");

    /*        Read the rest of header */
    for (i=0;i<8;i++){
            retval = fread(pchar, sizeof(char), 256, finl);
            err = fgetpos(finl, fpos);
    }
    /* free(pchar);
    free(dat);        */

    return(fpos);
}
```

READ_DAT.C

```
/*********************************************************/
/*   read_dat                                  */
/*          file to just read a frame at a time of     */
/*          .DAT file and save info                 */
/*                                          */
/*   2/24/93     K.C.                        */
/*******************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <errno.h>
#include "kinemat.h"
#include "p_file.h"
#include "fnc.h"

int read_dat(struct kineinfo *kinfo, FILE *dfile, fpos_t *fdpos, int kframe,\
           int itime)
{
        char *pchar,*dum;
        char pchar_array[260];
        int i, j, k, l, retval;
        int err, knext, frame;
        float *data;
        float dat;
        double time, rem;
        double *intptr;

        /*  Sometimes is skipping these - I have no idea why! */

        for (j=0;j<2;j++){
          printf("%d",j);
          for (k=0;k<3;k++){
            for (l=0;l<3;l++){
              kinfo->array_rot[j][k][l] = 0.0;
            }
            kinfo->array_pos [j][k] = 0.0;
          }
        }
```

```
/*          Now extract the array position and rotation matrices*/
/*          from the stored data.  16 numbers per segment per frame */
/*          Order – array position, 4 unidentified #'s, array rot.*/

        pchar = &pchar_array[0];
        err = fsetpos (dfile, fdpos);
        if (kinfo->freq > 254.){
          time = ((double) itime)/254.;
          rem = modf ((time * (kinfo->freq)), intptr);
          frame = (int) floor(*intptr);
          if (rem > 0.5) frame++;
        }
        else frame = kframe;

        for (k=0;k<(frame-kframe+1);k++){
          if ((retval = (fread(pchar,sizeof(char),256, dfile))) != 256 ){
            printf("Not reading in read_dat.c\n");
            scanf("%c",dum);
          }
        }

        err = fgetpos (dfile, fdpos);
        for (i=0;i<64;i++){
          dat = floatflt(pchar);
          pchar+=4;
        }
        pchar=&pchar_array[0];
        pchar += 16*2*4; /* Increment pointer to femoral array(seg3)*/
        for (j=0;j<2;j++){        /* Segment loop  */
              for (k=0;k<3;k++){
                    kinfo->array_pos[j][k] = floatflt(pchar);
/*                  printf("%f ",kinfo->array_pos[j][k]);*/
                    pchar+=4;
                    }
```

```
                pchar+=(4*4);  /* Pass the 4 unidentified numbers */
                for (k=0;k<3;k++){
                        for (l=0;l<3;l++){
                                kinfo->array_rot[j][k][l] = floatflt(pchar);
/*                      printf("%f ",kinfo->array_rot[j][k][l]);*/
                                pchar+=4;
                                }
                        }
                }
        printf(" \n");
        /* free(pchar); */
        return(frame++);
}
```

READ_FPD.C

```
/*************************************************************/
/*   read_fpd                                               */
/*          File to read FPD files (in binary)              */
/*          single frame at a time                          */
/*                                                          */
/*   2/24/93        K.C.                                    */
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

void read_fpd (struct kineinfo *kinfo, FILE *fdata, int itime,
                trans_a *thisframe, fpos_t *ffpos, int kframe)
{
        char *pchar;
        int i, j, k, l, nseg, err, retval, knext,frame;
        float *fpd;
        float wgtnt;
        char pchar_array[50];
        double time, rem;
        double *intptr;

/*      printf("\n entering read_fpd");*/

    /*          if (NULL == (pchar = (char *) calloc (48, sizeof(char)))){
         error("\n null pointer in read_fpd");
        }
    */
        if (itime == 0){
           for (i=0;i<10;i++){
                write_stat (21);
                pchar = &pchar_array[0];
                err = fsetpos(fdata, ffpos);
                retval = fread(pchar,sizeof(char),6*8,fdata);
                err = fgetpos(fdata, ffpos);
               }
          }
        wgtnt = kinfo->wgt * 9.8/2.2;      /* wt. in lbs -> wt. in Newtons */
        write_stat(22);
        pchar = &pchar_array[0];
        err = fsetpos(fdata, ffpos);
        if (kinfo->freq > 254.){
          time = ((double) itime)/254.;
          rem = modf ((time * (kinfo->freq)), intptr);
          frame = (int)floor(*intptr);
          if (rem > 0.5) frame++;
        }
        else frame = kframe;
        for (k=0;k<(frame-kframe+1);k++){
          retval = fread(pchar,sizeof(char),6*8,fdata);
        }
        err = fgetpos(fdata, ffpos);

/*      Scale the FP data by 100/body wt in Newtons (as at MGH) */
        pchar+=4;
        write_stat(23);

        write_stat(24);
        thisframe->fp1_vert = -1.0 * (floatflt(pchar)) * 100. / wgtnt;
        pchar+=(6*4);
        thisframe->fp2_vert = -1.0*(floatflt(pchar))*100./wgtnt;
        /* free(pchar);  */
        write_stat(25);
/*      printf("\n exiting read fpd");*/
        return;
}
```

BREAD_PR.C

```
/*******************************************************/
/*   bread_prs                                   */
/*        subroutine to read binary .PRS files       */
/*        a frame at a time.                           */
/*        made it change to MPA too                 */
/*                                              */
/*   2/24/93     K.C.                              */
/*******************************************************/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"
```

```
int bread_prs (struct kineinfo *kinfo, struct datainfo *pinfo, FILE *prsfile,
                    struct pzeros *pzero, int itime,
                    int curframe, trans_a *thisframe, fpos_t *fppos)
{
        int i, j, k, ib[16], pframe;
        int dum[38], retval, err;
        int *pdum;
        float offset[14], gain[14], tempcorr[14], temp;
        float *poff, *pgain, *ptcorr;
        double data[16], time, rem;
        int *p_ib;
        double *p_data;
        double pf, *intptr;

        p_ib = &ib[0];
        p_data = &data[0];
        poff = &offset[0];
        pgain = &gain[0];
        ptcorr = &tempcorr[0];
        intptr = &pf;
        pdum = &dum[0];

/*          Read the data then turn into pressures          */
/*          Only have to deal with prosthesis 33            */
/*           First time through, find zeros, read header    */

        if (itime == 0){
                write_stat(11);
                temp = zero33 (poff, pgain, ptcorr, pinfo);
                write_stat(12);
                pzero->temp = temp;
                for (i=0;i<14;i++){
                        pzero->offset[i] = *poff;
                        pzero->gain[i] = *pgain;
                        pzero->tempcorr[i] = *ptcorr;
                        poff++;
                        pgain++;
                        ptcorr++;
                        }
                poff-=14;
                pgain-=14;
                ptcorr-=14;
                for (i=0;i<11;i++){
                        pdum = &dum[0];
                        err = fsetpos(prsfile, fppos);
                        retval = fread(pdum,sizeof(int),38,prsfile);
                        err = fgetpos(prsfile, fppos);
                    }
                pinfo->freq = 254.;
                strcpy(pinfo->id,"33");
                pinfo->ntrans=13;
                }
else{
            for (i=0;i<14;i++){
                        offset[i] = pzero->offset[i];
                        gain[i] = pzero->gain[i];
                        tempcorr[i] = pzero->tempcorr[i];
                        }
             temp = pzero->temp;
            }
if (kinfo->freq <=254.){
  time = ((double)itime)/ (kinfo->freq);
  rem = modf ((time * pinfo->freq), intptr);
  pframe = (int) floor(*intptr);
  if (rem > 0.5) pframe++;
}
else {
  pframe = curframe;
}
for (k=curframe; k<pframe; k++){
            pdum = &dum[0];
            err = fsetpos (prsfile, fppos);
```

```
                fread(pdum,sizeof(int),38,prsfile);
                err = fgetpos (prsfile, fppos);
                }
        pdum = &dum[0];
        err = fsetpos (prsfile, fppos);
        retval = fread(pdum,sizeof(int),38,prsfile);
        err = fgetpos(prsfile, fppos);
        for (j=1; j<17; j++){
                ib[j−1] = dum[j];
                        }
        pframe++;
        write_stat(13);
        raw_to_mpa(p_ib, p_data, pinfo, poff, pgain, ptcorr, temp);
        for (j=2; j<15; j++){
                if (j!=5&&j!=7&&j!=9){
                        thisframe−>alltrans[j−2].mpa = data[j−2];
                        }
                else {
                  thisframe−>alltrans[j−2].mpa = 0.0;
                  }
                }
        thisframe−>time = time;
        return(pframe);
}
```

AREAD_PR.C

```
/*******************************************************/
/*   aread_prs                                        */
/*         subroutine to read ASCII .PRS files        */
/*         a frame at a time.                         */
/*         made it change to MPA too                  */
/*                                                    */
/*   2/24/93      K.C.                                */
/*******************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

int aread_prs (struct kineinfo *kinfo, struct datainfo *pinfo, FILE *prsfile,
                struct pzeros *pzero, int itime,
                int curframe, trans_a *thisframe, fpos_t *fppos)

{
        int i, j, k, ib[16], pframe;
        int dum[38], pfreq, retval, err;
        int *pdum;
        float offset[14], gain[14], tempcorr[14], temp;
        float *poff, *pgain, *ptcorr;
        double data[16], time, rem;
        int *p_ib;
        double *p_data;
        double pf, *intptr;

        p_ib = &ib[0];
        p_data = &data[0];
        poff = &offset[0];
        pgain = &gain[0];
        ptcorr = &tempcorr[0];
        intptr = &pf;
        pdum = &dum[0];

/*              Read the data then turn into pressures          */
/*              Only have to deal with prosthesis 33            */
/*                First time through, find zeros, read header    */

        if (itime == 0){
            temp = zero33 (poff, pgain, ptcorr, pinfo);
            pzero->temp = temp;
            for (i=0;i<14;i++){
                    pzero->offset[i] = *poff;
                    pzero->gain[i] = *pgain;
                    pzero->tempcorr[i] = *ptcorr;
                    poff++;
                    pgain++;
                    ptcorr++;
                    }
            poff-=14;
            pgain-=14;
            ptcorr-=14;
            err = fsetpos(prsfile, fppos);
            fscanf (prsfile, "%d %d",&pinfo->frames,&pfreq);
            fscanf (prsfile, "%d %d %2d%*d",&pinfo->date[0],
                            &pinfo->date[1], &pinfo->date[2]);
            err = fgetpos(prsfile, fppos);
            strcpy (pinfo->id,"33");
            pinfo->freq = (double)pfreq;
            pinfo->ntrans = 13;
            }
    else{
            for (i=0;i<14;i++){
                    offset[i] = pzero->offset[i];
                    gain[i] = pzero->gain[i];
                    tempcorr[i] = pzero->tempcorr[i];
                    }
            temp = pzero->temp;
            }
    if (kinfo->freq <= 254.){
      time = ((double)itime)/ (kinfo->freq);
      rem = modf ((time * pinfo->freq), intptr);
      pframe = floor(*intptr);
      if (rem > 0.5) pframe++;
    }
```

```
        else {
          pframe = curframe;
        }
        err = fsetpos(prsfile, fppos);
        for (k=curframe; k<pframe; k++){
                for (j=0;j<37;j++){
                        fscanf(prsfile,"%*d");
                        }
                }
        for (j=0; j<37; j++){
                if (j>=1 && j<=16)
                        fscanf(prsfile,"%d",&ib[j-1]);
                else fscanf (prsfile,"%*d");
                        }
        err = fgetpos(prsfile, fppos);
        pframe++;
        raw_to_mpa(p_ib, p_data, pinfo, poff, pgain, ptcorr, temp);
        for (j=2; j<15; j++){
                if (j!=5&&j!=7&&j!=9){
                        thisframe->alltrans[j-2].mpa = data[j-2];
                  }
                  else{
                        thisframe->alltrans[j-2].mpa = 0.0;
                        }
                }
        thisframe->time = time;
        return(pframe);
}
```

WRITE_TM.C

```
/****************************************************************/
/*   write_tmp                                        */
/*        Writes a temporary file containing info         */
/*        for each frame of data                        */
/*                                                    */
/*          FORMAT: all floats                          */
/*          time, pressures (13, in MPA, 0 if bad), fp1_vert,*/
/*          fp2_vert, array_pos[2][3], array_rot[2][3][3]  */
/*                                                      */
/*  2/24/93     K.C.                                   */
/**********************************************************/
#include <stdio.h>
#include <errno.h>
#include "kinemat.h"
#include "p_file.h"
#include "fnc.h"

void write_tmp(struct kineinfo *kinfo, struct datainfo *pinfo,
                FILE *tmpfile, trans_a *thisframe, fpos_t *ftpos)
{
    int i,j,k, err;
    float temp[45];
    float *ptemp;
    extern int write_count;
/*      printf("\n entering write_temp");*/
    ptemp = &temp[0];

    temp[0] = (float) (thisframe->time);
    write_stat(29);
    for (i=0;i<NTRANS;i++){
/*          printf("\n i = %d, val = %lf",i, (thisframe->alltrans[i].mpa ));
*/
            temp[i+1] = (float) (thisframe->alltrans[i].mpa);
            }
    temp[14] = thisframe->fp1_vert;
    temp[15] = thisframe->fp2_vert;
    write_stat(31);
    for (i=0;i<2;i++){
            for (j=0;j<3;j++){
                    temp[i*3+j+16] = kinfo->array_pos[i][j];
            }
```

```
        }
        write_stat(32);
        for (i=0;i<2;i++){
                for (j=0;j<3;j++){
                        for (k=0;k<3;k++){
                                temp[i*9+j*3+k+22] = kinfo->array_rot[i][j][k];
                                }
                        }
                }
/*      printf("\n starting write int write_tmp ");*/
        err = fsetpos(tmpfile, ftpos);
        fwrite (ptemp, sizeof(float), 40, tmpfile);
        err = fgetpos(tmpfile, ftpos);
/*          printf ("\n %d ",write_count); */
/*      for (i=0;i<40;i++){
            printf("%f ", temp[i]);
        }
        printf("\n");
*/
        ++write_count;
        return;
}
```

RAW_TO_M.C

```
/**********************************************************************/
/*      raw_to_mpa                                      */
/*              function to get pressure in MPa at each         */
/*      transducer (for 33 only) from raw values collected      */
/*      at MGH. (Well – could have been collected elsewhere)    */
/*                                                      */
/*      Dec. 9, 1992    K J.                            */
/**********************************************************************/

#include <stdio.h>
#include <math.h>
```

```
#include "p_file.h"
#include "kinemat.h"

#define VTOMV   1000.0
#define PSITOMPA  0.0068948



void raw_to_mpa ( int *praw, double *pmpa, struct datainfo *ppinfo,
                  float *poff, float *pgain, float *ptcorr, float temp)
{
        int i, j, cal, zero, level, diff, data;
        double cmv;


/*          Extract the scaling information */
/*          If analog value is > 3*(cal-zero)*/
/*          then the part above that level has */
/*          been compressed by a factor of 4 */

        cal = *praw;
        praw++;
        zero = *praw;
        praw++;
        level = 3*(cal-zero);

/*          Need to check if Temperature should be difference */
/*          between "normal" and actual or just actual       */
#ifdef _MGH_PRINT_
        printf("praw = %d",*praw);
#endif

        for (i=0; i<ppinfo->ntrans; i++){
                data = *praw;
                if (i!=3 && i!=5 && i!=7){
                        if (*praw > level){
```

```
                        diff = *praw - level;
                        diff = diff*4;
                        *praw = level + diff;
                        data = *praw;
                        }
                cmv = ((double)(data-zero))/((double)(cal-zero)) * ((double) VTOMV) ;
                *pmpa = (cmv - (double) ( (*ptcorr)*(temp--37.0)
                                        + (*poff)) ) /
                        ( (double) (*pgain) );
                *pmpa = (*pmpa) * ((double) PSITOMPA);
                }
        else {
                cmv = 0.0;
                *pmpa = 0.0;
                }
#ifdef _MGH_PRINT_
        printf("pmpa = %lf, praw = %d\n",*pmpa, *praw);
#endif
        pmpa++;
        praw++;
        poff++;
        pgain++;
        ptcorr++;
        }
    return;
}
```

ZERO33.C

```
/*********************************************************************/
/*      zero33  ---                                                  */
/*              function to get offsets and gains and temp          */
/*      adjustment and temperature for a data set taken             */
/*      from prosthesis 33.   Reads 33.Dat file, or same format     */
/*                                                              */
/*      Dec. 9, 1992      K.C.                                  */
/****************************************************************/

#include <stdio.h>
#include "p_file.h"
#include "kinemat.h"

float zero33 (float *poff, float *pgain, float *ptcorr,
              struct datainfo *pinfo)
{
        int i, date[3];
        char zname[30];
        float temp;
        FILE *zfile;

        zfile = fopen ("33.dat","r");

        for (i=0;i<16;i++){
                fscanf (zfile,"%f",pgain);
                if (i>=2) pgain++;
                }
        for (i=0;i<16;i++){
                fscanf (zfile, "%f",ptcorr);
                if (i>=2) ptcorr++;
                }
        do {
                for (i=0;i<16;i++){
                        fscanf (zfile, "%f", poff);
                        if (i>=2) poff++;
                        }
                fscanf (zfile,"%d",&date[0]);
                if (date[0] != 13){
                        fscanf (zfile, "%d", &date[1]);
                        fscanf (zfile, "%d", &date[2]);
                        fscanf (zfile, "%f", &temp);
```

```
                    if (date[0] == pinfo->date[0] &&
                            date[1] == pinfo->date[1] &&
                            date[2] == pinfo->date[2]){
                            fclose (zfile);
                            return(temp);
                            }
                    }
            } while (date[0] !=EOF );
        fclose(zfile);
        printf("Zeros not found for this date\n");
        temp = 37.0;  /* May want to change default to 38 */
        return(temp);
}
```

READ_TMP.C

```
/**************************************************************/
/*   read_tmp                                         */
/*       Reads temp.dat file created by write_tmp.c       */
/*       Should at some point put a header on temp file  */
/*                                                    */
/*       FORMAT: all floats                           */
/*        time, pressures (13, in MPA, 0 if bad), fp1_vert,*/
/*        fp2_vert, array_pos[2][3], array_rot[2][3][3]  */
/*                                                    */
/*   2/24/93    K.C.                                  */
/**************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include "kinemat.h"
#include "p_file.h"
#include "fnc.h"

void read_tmp(struct kineinfo *kinfo, struct datainfo *pinfo, FILE *tmpfile,
                    trans_a *thisframe, fpos_t *fpos)
{
    int i,j,k, retval;
    float temp[45];
    float *ptemp;
    extern int read_count;
    printf("\n entering read_tmp");
    ptemp = &temp[0];

/* Read a frame from temp.dat file (40 floats)*/

/*     printf("\n starting read in read_tmp "); */
    if (40 != (retval = fread(ptemp, siseof(float), 40, tmpfile))){
          printf(" \n Didn't read in read_tmp retval = %d",retval);
          fclose(tmpfile);
          exit(1);
        }
/*     printf("\n  %d %f %f %f",read_count,temp[0],temp[14],temp[1]);*/

/* Copy first number to thisframe->time */

    thisframe->time = (double) temp[0];

/* Copy next NTRANS numbers to thisframe->alltrans[i].mpa */

    for (i=0;i<NTRANS;i++){
            thisframe->alltrans[i].mpa = (double) temp[i+1];
            }

/* Copy next two numbers to thisframe->fp1_vert & fp2_vert */

    thisframe->fp1_vert = temp[14];
    thisframe->fp2_vert = temp[15];

/* Next 6 numbers are kinfo->array_pos[seg][3], array position vectors, */
/*    only for 3rd (femur) and 4th (pelvis) segments, which are */
/*    locally numbered 0 and 1 */
```

```
    for (i=0;i<2;i++){
            for (j=0;j<3;j++){
                    kinfo->array_pos[i][j] = temp[i*3+j+16];
            }
    }

/* Next 18 numbers are kinfo->array_rot[seg][3][3], array rotation */
/*    matrices for femur and pelvis */

    for (i=0;i<2;i++){
            for (j=0;j<3;j++){
                    for (k=0;k<3;k++){
                            kinfo->array_rot[i][j][k] = temp[i*9+j*3+k+22];
                            }
                    }
            }

    ++read_count;
    return;
}
```

CALC_JTA.C

```
/*******************************************************/
/*      calc_jta                                     */
/*          Calculate joint angles, calls mgh_ang.c     */
/*      which uses Pierrinowsky and Tupling             */
/*      Only works on one frame at a time - called      */
/*      from kine33.c                            */
/*                                                   */
/*******************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
```

```
#include <string.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"


void calc_jta (struct kineinfo *kinfo,
            struct datainfo *pinfo, trans_a *thisframe)
{
        char *side;
        int num;
        int i, j, k, l, nframe, iside, ijoint, nseg, nsynch;
        float smt[2][4][4];
        float smt2[2][3][3];
        float seg_rot [2][3][3]; /* Store only for the two */
        float array_rot [2][3][3];
        double bad;
        float *psmt2, *psegrot, *parrayrot, *psmt;

        bad = -1000.0;
        nsynch = 0;
        strcpy(side,"R");
        if (*side == 'L') iside = 2;
        else iside = 1;
        ijoint = 3; /* Hope this is the right answer! Want the hip joint*/
        nseg = kinfo->nseg;

/* Set all of smt to 0.0 */
        for (j=0;j<2;j++){
          for (k=0;k<4;k++){
            for (l=0;l<4;l++){
              smt[j][k][l] = 0.0;
            }
          }
        }
```

```
/*                  Fill the segment vector and rotation matrices */
/*                  from kineinfo structure.                       */
/*                  Assume the segments of interest (femur, pelvis) */
/*                  are segments 3 and 4                           */
/*       Also, fill array_rot from kinfo->array_rot, just so */
/*       don't have to change mult_mat.c */

        for (k=2;k<4;k++){                 /* Segment loop */
                for (i=0;i<3;i++){
                        for (j=0;j<3;j++){
                            seg_rot[k-2][i][j]=kinfo->seg_rot[k][i][j];
                            array_rot[k-2][i][j]=kinfo->array_rot[k-2][i][j];
                            }
                        }
                }
/*       printf ("in calc_jta, before segment loop\n");*/
/*  Now do the calculations, already have the data, and are only */
/*  doing a frame at a time.  Still need to check for bad data */

/*       Multiply the array orientation matrix into body segment coord.*/

      for (j=0;j<2;j++){    /* Segment loop */
                psmt2 = &smt2[j][0][0];
                psegrot = &seg_rot[j][0][0];
                parrayrot = &array_rot[j][0][0];

                if ((kinfo->array_pos[j][0])!=bad &&
                   (kinfo->array_pos[j][1])!=bad
                   && (kinfo->array_pos[j][2])!=bad){
/*                     printf("in calc_jta, going to mult_mat\n");*/
                     mult_matrices(psegrot,parrayrot,psmt2,3,3,3);
                        for (k=0;k<3;k++){
                                for (l=0;l<3;l++){
                                        smt[j][k][l]=smt2[j][k][l];
                                        }
```

```
                              smt[j][k][3] = 0.0;
                              }
                        for (l=0;l<3;l++){
                              smt[j][3][l] = 0.0;
                            }
                        smt[j][3][3] = 1.0;  /* Good data */
                        }
                  else{
                     smt[j][3][3] = 0.0;  /* Bad data */
                    }
                  }

/*      Do angle calculation elsewhere */

     psmt = &smt[0][0][0];
/*      printf("in calc_jta, going to mgh_ang\n");*/
        mgh_ang( psmt, thisframe, side);
        return;
}
```

CALC_XFM.C

```
/*********************************************************/
/*    calc_xfm.c - based on
/*      hip_xform ---                                  */
/*              function to transform transducer       */
/*      locations into acetabular locations.           */
/*      Clinical joint angles are obtained earlier     */
/*      and are stored for each frame in the trans_a   */
/*      structure.  Joint angles are from Grood &      */
/*      Suntay defn's.                                 */
/*                                                     */
/*              Bulk of the program from TRANS(in FORTAN*/
/*      by Bob Fijan and me - but may be significant   */
/*      alterations for use with left as well as right */
/*      hips and because am working from notes made    */
/*       a couple years ago - reworked TRANS and found      */
/*       some bugs/better ways to do things.                */
/*                                                          */
/*                                                          */
/*       Dec. 10, 1992     K.C.                             */
/***********************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"

#define PI       3.1415927


void calc_xfm (struct kineinfo *kinfo,
               struct datainfo *pinfo, trans_a *thisframe)
{
        char hip[3],reply[4];
        double flex, abd, rot;
        double u[3],c[3][3],rad,theta, factor, alter;
        double f[3],a[3],c2[3][3];
        int i,j;

        strcpy(hip,"R");
        flex = (thisframe->flexion);
        rot = (thisframe->extrotation);
        abd = (thisframe->abduction);
        for (i=0;i<3;i++){
          for (j=0;j<3;j++){
            c[i][j] = 0.0;
            c2[i][j] = 0.0;
          }
          u[i] = 0.0;
        f[i] = 0.0;
        a[i] = 0.0;
       }

/*      Calculate the rotation matrix first */
/* Worked this out for a right leg in 2/90...is also different */
/* from HTRANS.  */

       c[0][0] = cos(rot)*cos(flex);
       c[0][1] = cos(rot)*sin(flex) + sin(rot)*sin(abd);
       c[0][2] = sin(rot)*cos(abd);
       if (hip[0] == 'L') c[0][2] = -c[0][2];

       c[1][0] = -cos(abd)*sin(flex);
       c[1][1] = cos(abd)*cos(flex);
       c[1][2] = -sin(abd)*cos(flex);
       if (hip[0] == 'L') c[1][2] = -c[1][2];

       c[2][0] = -sin(rot)*cos(flex);
       c[2][1] = -sin(flex)*sin(rot) + cos(rot)*sin(abd);
       c[2][2] = cos(rot)*cos(abd);
       if (hip[0] == 'L'){
              c[2][0] = -c[2][0];
              c[2][1] = sin(flex)*sin(rot) - cos(rot)*sin(abd);
              }

/*      Now get c2 (pelvis_to_acet_rotation matrix) from kineinfo */

       for (j=0;j<3;j++){
              for (i=0;i<3;i++){
                     c2[j][i] = kinfo->pelvis_to_acet_rot[j][i];
                     }
              }

/* Now rotate for each transducer */
```

```
        for (j=0;j<pinfo->ntrans;j++){
                for (i=0;i<3;i++){
                        f[i] = kinfo->prosth_to_femoral_rot[j][i];
/*                      if (j==2) printf("f[%d] = %lf\n",i,f[i]);*/
                        }


/*      Now need to go into pelvis coord using the clinical */
/*      joint angles.  Assume angles in radians.          */



                u[0] =  c[0][0]* f[0] + c[1][0] * f[1] + c[2][0] * f[2];
                u[1] =  c[0][1] * f[0] + c[1][1] * f[1] + c[2][1] * f[2];
                u[2] =    c[0][2] * f[0] + c[1][2] * f[1] + c[2][2] * f[2];



/* Now change to acetabular coordinates */
/* Use the rotation matrix calculated in calc_matrices and stored */
/* in pelvis_to_acet_rot in structure kineinfo */
/* Transposed relative to the previous matrix */



                a[0] = c2[0][0] * u[0] + c2[0][1] * u[1] + c2[0][2] * u[2];
                a[1] = c2[1][0] * u[0] + c2[1][1] * u[1] + c2[1][2] * u[2];
                a[2] = c2[2][0] * u[0] + c2[2][1] * u[1] + c2[2][2] * u[2];


/*      Now change [a] into an acetabular phi and theta and store as degrees */
/*      For a left hip, phi measured from pos Za axis -- so change sign there*/
                if (hip[0] == 'L') a[2] = - a[2];
                factor = -a[2];
                thisframe->alltrans[j].ang.phi = (acos(factor)) * 180./PI;
                rad = sqrt((a[0]*a[0] + a[1]*a[1]));
                theta = acos((a[0]/rad)) * 180./PI;
                if (a[1] < 0) theta = -theta;
                thisframe->alltrans[j].ang.theta = theta;
/*              if (a[0] > 0.0){
                        thisframe->alltrans[j].ang.theta =
                        atan2(a[1],a[2])*180./PI;
                    }
                else {
                    thisframe->alltrans[j].ang.theta =
                        atan2( a[0], a[1]) * 180./PI;
                    } */
            }
        thisframe->flexion = flex * 180./PI;
        thisframe->abduction = abd * 180./PI;
        thisframe->extrotation = rot * 180./PI;

    return;
}
```

E.3.3 Programs for data taken after 1986 from prosthesis 33

READ_DAT.C

```
/*******************************************************/
/*  read_dat                                    */
/*      file to just read a frame at a time of  */
/*      .DAT file and save info                 */
/*                                              */
/*  2/24/93     K.C.                            */
/*******************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>
#include "kinemat.h"
#include "p_file.h"
#include "fnc.h"

void read_dat(struct kineinfo *kinfo, FILE *dfile, fpos_t *fdpos)
{
 char *pchar,*dum;
 char pchar_array[260];
 int i, j, k, l, nseg, retval, err;
 float *data;
 float dat;


 nseg = kinfo->nseg;


 /* pchar = (char *) calloc (16*16, sizeof(char));        */

 for (j=0;j<2;j++){
   for (k=0;k<3;k++){
     for (l=0;l<3;l++){
       kinfo->array_rot[j][k][l] = 0.0;
     }
     kinfo->array_pos [j][k] = 0.0;
   }
 }


/*          Now extract the array position and rotation matrices*/
/*          from the stored data.  16 numbers per segment per frame */
/*          Order - array position, 4 unidentified #'s, array rot.*/

 pchar = &pchar_array[0];
 err = fsetpos (dfile, fdpos);
 if ((retval = (fread(pchar,sizeof(char),256, dfile))) != 256 ){
   printf("Not reading in read_dat.c\n");
   scanf("%c",dum);
 }
 err = fgetpos (dfile, fdpos);
 for (i=0;i<64;i++){
   dat = floatflt(pchar);
   pchar+=4;
```

```
        }
        pchar=&pchar_array[0];
        pchar += 16*2*4; /* Increment pointer to femoral array(seg3)*/
        for (j=0;j<2;j++){        /* Segment loop  */
                for (k=0;k<3;k++){
                        kinfo->array_pos[j][k] = floatflt(pchar);
/*                      printf("%f ",kinfo->array_pos[j][k]);*/
                        pchar+=4;
                        }
                pchar+=(4*4);  /* Pass the 4 unidentified numbers */
                for (k=0;k<3;k++){
                        for (l=0;l<3;l++){
                                kinfo->array_rot[j][k][l] = floatflt(pchar);
/*                      printf("%f ",kinfo->array_rot[j][k][l]);*/
                                pchar+=4;
                                }
                        }
                }
        printf(" \n");
        /* free(pchar); */
        return;
}
```

READ_FPD.C

```
/************************************************************/
/*   read_fpd                                              */
/*          File to read FPD files (in binary)             */
/*          single frame at a time                         */
/*                                                         */
/*   2/24/93      K.C.                                     */
/************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
```

```
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

void read_fpd (struct kineinfo *kinfo, FILE *fdata, int itime,
                    trans_a *thisframe, fpos_t *ffpos, int kframe)
{
         char *pchar;
         int i, j, k, l, nseg, err, retval, knext,frame;
         float *fpd;
         float wgtnt;
         char pchar_array[50];
         double time, rem;
         double *intptr;

/*       printf("\n entering read_fpd");*/

    /*            if (NULL == (pchar = (char *) calloc (48, sizeof(char)))){
           error("\n null pointer in read_fpd");
         }
     */
         if (itime == 0){
             for (i=0;i<10;i++){
                     write_stat (21);
                     pchar = &pchar_array[0];
                     err = fsetpos(fdata, ffpos);
                     retval = fread(pchar,sizeof(char),6*8,fdata);
                     err = fgetpos(fdata, ffpos);
                   }
             }
         wgtnt = kinfo->wgt * 9.8/2.2;      /* wt. in lbs -> wt. in Newtons */
         write_stat(22);
         pchar = &pchar_array[0];
         err = fsetpos(fdata, ffpos);
```

```
        if (kinfo->freq > 254.){
          time = ((double) itime)/254.;
          rem = modf ((time * (kinfo->freq)), intptr);
          frame = (int)floor(*intptr);
          if (rem > 0.5) frame++;
        }
        else frame = kframe;
        for (k=0;k<(frame-kframe+1);k++){
          retval = fread(pchar,siseof(char),6*8,fdata);
        }
        err = fgetpos(fdata, ffpos);

/*      Scale the FP data by 100/body wt in Newtons (as at MGH) */
        pchar+=4;
        write_stat(23);

        write_stat(24);
        thisframe->fp1_vert = -1.0 * (floatflt(pchar)) * 100. / wgtnt;
        pchar+=(6*4);
        thisframe->fp2_vert = -1.0*(floatflt(pchar))*100./wgtnt;
        /* free(pchar);  */
        write_stat(25);
/*      printf("\n exiting read fpd");*/
        return;
}
```

BREAD_PR.C

```
/**********************************************************/
/*  bread_prs                              */
/*       subroutine to read binary .PRS files       */
/*       a frame at a time.                  */
/*       made it change to MPA too            */
/*                                         */
/*  2/24/93    K.C.                       */
/**********************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

int bread_prs (struct kineinfo *kinfo, struct datainfo *pinfo, FILE *prsfile,
               struct pzeros *pzero, int itime,
               int curframe, trans_a *thisframe, fpos_t *fppos)
{
        int i, j, k, ib[16], pframe, iheader;
        int dum[64], retval, err;
        int *pdum;
        float offset[14], gain[14], tempcorr[14], temp;
        float *poff, *pgain, *ptcorr;
        double data[16], time, rem;
        int *p_ib;
        double *p_data;
        double pf, *intptr;
        fpos_t *fpos;

        p_ib = &ib[0];
        p_data = &data[0];
        poff = &offset[0];
        pgain = &gain[0];
        ptcorr = &tempcorr[0];
        intptr = &pf;
        pdum = &dum[0];
        fpos = fppos;

/*          Read the data then turn into pressures          */
/*          Only have to deal with prosthesis 33            */
/*            First time through, find zeros, read header   */
```

```
if (itime == 0){
        write_stat(11);
        temp = zero33 (poff, pgain, ptcorr, pinfo);
        write_stat(12);
        pzero->temp = temp;
        for (i=0;i<14;i++){
                pzero->offset[i] = *poff;
                pzero->gain[i] = *pgain;
                pzero->tempcorr[i] = *ptcorr;
                poff++;
                pgain++;
                ptcorr++;
                }
        poff-=14;
        pgain-=14;
        ptcorr-=14;
        iheader=8;
        if ( pinfo->date[2]!=86)iheader=11;
        for (i=0;i<iheader;i++){
                pdum = &dum[0];
                err = fsetpos(prsfile, fpos);
                retval = fread(pdum,sizeof(int),64,prsfile);
                err = fgetpos(prsfile, fpos);
            }

        pinfo->freq = 254.;
        strcpy(pinfo->id,"33");
        pinfo->ntrans=13;
        }
else{
        for (i=0;i<14;i++){
                offset[i] = pzero->offset[i];
                gain[i] = pzero->gain[i];
                tempcorr[i] = pzero->tempcorr[i];
                }
                temp = pzero->temp;
                }
        time = ((double)itime)/ (kinfo->freq);
        rem = modf ((time * pinfo->freq), intptr);
        pframe = (int) floor(*intptr);
        if (rem > 0.5) pframe++;
        for (k=curframe; k<pframe; k++){
                pdum = &dum[0];
                err = fsetpos (prsfile, fpos);
                fread(pdum,sizeof(int),64,prsfile);
                err = fgetpos (prsfile, fpos);
                }
        pdum = &dum[0];
        err = fsetpos (prsfile, fpos);
        retval = fread(pdum,sizeof(int),64,prsfile);
        err = fgetpos(prsfile, fpos);
        for (j=1; j<17; j++){
                ib[j-1] = dum[j];
                        }
        pframe++;
        write_stat(13);
        raw_to_mpa(p_ib, p_data, pinfo, poff, pgain, ptcorr, temp);
        for (j=2; j<15; j++){
                if (j!=5&&j!=7&&j!=9){
                        thisframe->alltrans[j-2].mpa = data[j-2];
                        }
                else {
                  thisframe->alltrans[j-2].mpa = 0.0;
                  }
                }
        thisframe->time = time;
        return(pframe);
}
```

AREAD_PR.C

```
/*****************************************************/
/*   aread_prs                                        */
/*           subroutine to read ASCII .PRS files     */
/*           a frame at a time.                       */
/*           made it change to MPA too                */
/*                                                    */
/*   2/24/93       K.C.                               */
/*****************************************************/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "p_file.h"
#include "kinemat.h"
#include "fnc.h"

int aread_prs (struct kineinfo *kinfo, struct datainfo *pinfo, FILE *prsfile,
                        struct pzeros *pzero, int itime,
                        int curframe, trans_a *thisframe, fpos_t *fppos)

{
          int i, j, k, ib[16], pframe;
          int dum[38], pfreq, retval, err;
          int *pdum;
          float offset[14], gain[14], tempcorr[14], temp;
          float *poff, *pgain, *ptcorr;
          double data[16], time, rem;
          int *p_ib;
          double *p_data;
          double pf, *intptr;

          p_ib = &ib[0];
          p_data = &data[0];
          poff = &offset[0];
          pgain = &gain[0];
```

```
        ptcorr = &tempcorr[0];
        intptr = &pf;
        pdum = &dum[0];

/*              Read the data then turn into pressures          */
/*              Only have to deal with prosthesis 33            */
/*               First time through, find zeros, read header    */

        if (itime == 0){
                temp = zero33 (poff, pgain, ptcorr, pinfo);
                pzero->temp = temp;
                for (i=0;i<14;i++){
                        pzero->offset[i] = *poff;
                        pzero->gain[i] = *pgain;
                        pzero->tempcorr[i] = *ptcorr;
                        poff++;
                        pgain++;
                        ptcorr++;
                        }
                poff-=14;
                pgain-=14;
                ptcorr-=14;
                err = fsetpos(prsfile, fppos);
                fscanf (prsfile, "%d %d",&pinfo->frames,&pfreq);
                fscanf (prsfile, "%d %d %2d%*d",&pinfo->date[0],
                                &pinfo->date[1], &pinfo->date[2]);
                err = fgetpos(prsfile, fppos);
                strcpy (pinfo->id,"33");
                pinfo->freq = (double)pfreq;
                pinfo->ntrans = 13;
                }
        else{
                for (i=0;i<14;i++){
                        offset[i] = pzero->offset[i];
                        gain[i] = pzero->gain[i];
                        tempcorr[i] = pzero->tempcorr[i];
                }
            temp = pzero->temp;
            }
    time = ((double)itime)/ (kinfo->freq);
    rem = modf ((time * pinfo->freq), intptr);
    pframe = (int) floor(*intptr);
    if (rem > 0.5) pframe++;
    err = fsetpos(prsfile, fppos);
    for (k=curframe; k<pframe; k++){
            for (j=0;j<37;j++){
                    fscanf(prsfile,"%*d");
                    }
            }
    for (j=0; j<37; j++){
            if (j>=1 && j<=16)
                    fscanf(prsfile,"%d",&ib[j-1]);
            else fscanf (prsfile,"%*d");
                    }
    err = fgetpos(prsfile, fppos);
    pframe++;
    raw_to_mpa(p_ib, p_data, pinfo, poff, pgain, ptcorr, temp);
    for (j=2; j<15; j++){
            if (j!=5&&j!=7&&j!=9){
                    thisframe->alltrans[j-2].mpa = data[j-2];
                    }
            }
    thisframe->time = time;
    return(pframe);
}
```

E.4 Programs for acetabular regional information

LOC3_STA.C

```
/*****************************************************************/
/*      loc_stat.c                                               */
/*          Program to take a file written by "locate.exe" that */
/*      contains pressure data and phi and theta and to calc.    */
/*      some basic statistics for small regions -- like max p    */
/*      in that region for that file, min p, avg p, rms pressure*/
/*                                                               */
/*      Jan 18, 1993   K.C.                                      */
/*****************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>

float pmax[36][36],pmin[36][36],pavg[36][36],prms[36][36];
int pnpts[36][36];

void cartcalc (float, float, float, float *, float *, float *, char);

void main (int argc, char *argv[])
{
        char inname[40], outname[40], batname[40];
        char title[40], side[3];
        int i,j,k,nframe, ntrans, skip, npts,n_phi, n_theta;
        int ii, jj, retval;
        float phi, theta, x[2], y[2], z[2], time;
        float p, trans, ht, wt;
        float normalize, scale, dum;
        float *px, *py, *pz;
        FILE *infile;
        FILE *outfile;
        FILE *batfile;
        fpos_t *start;
        if (argc < 2){
            printf("Supply the batch file name to process.\n");
            scanf("%s",argv[1]);
            }
/*          Automate the file accessing, numbering somewhat*/
/*          Assume that input is a batch file with files that*/
/*          Want to process.                    */
/*          Assume input file names are "name".tec; Output will*/
/*          be "name".stat and maybe "name"s.tec */

        strcpy(batname,argv[1]);
        batfile = fopen(batname,"r");
        if (batfile == NULL){
            printf("Couldn't open batch file\n");
            abort;
            }
        n_phi = 36;
        n_theta = 36;
        while ( (fscanf(batfile,"%s",inname)) != EOF ){

            strcpy(outname,inname);
            strcat(inname,".tec");
            strcat(outname,"stat3.tec");
            printf("Processing %s\n",inname);
            dum = 0.0;
            ntrans = 13;            /* Just set it for now. */

/*      Initialize the arrays          */
            for (i=0;i<n_phi;i++){
                for (j=0;j<n_theta;j++){
                    pmax[i][j] = 0.0;
                    pmin[i][j] = 0.0;
                    pavg[i][j] = 0.0;
                    prms[i][j] = 0.0;
                    pnpts[i][j] = 0.0;
                }
              }


/*        For now, don't normalize, but may want to eventually     */

          fscanf (batfile,"%f",&ht);
          fscanf (batfile,"%f", &wt);
          fscanf (batfile,"%s",side);   /* side = 'L' or 'R'*/

          px = &x[0];
          py = &y[0];
          pz = &z[0];
          infile = fopen(inname,"r");
          outfile = fopen(outname,"w");

          fscanf(infile,"%*s %*s %s",title);
          fscanf(infile,"%*s %*s %*s %*s %*s %*s %*s");

/*        Calculate number of points to do for each line              */
          fscanf(infile,"%*s %*s %*s %d, %*s %*s",&nframe);
          npts = nframe;
          retval = fgetpos(infile,start);

          for (j=0;j<ntrans;j++){
            printf("Transducer %d\n",j);
               for (i=0;i<nframe;i++){
                      fscanf(infile,"%f %f %f %f %f", &time, &trans,
                                &phi, &theta, &p);
                      if (theta < -115. && theta > -180.) theta=theta+360.;
                      ii = (int)(((phi-30.)/80.)*n_phi - 0.5);
                      jj = (int)(((theta+65.)/310.)*n_theta - 0.5);
                      if ((ii >= 0 && ii <= 35)&&(jj>=0&&jj<=35)){
                           if (pmax[ii][jj] < p) pmax[ii][jj] = p;
                           if (pmin[ii][jj] > p) pmin[ii][jj] = p;
                           pnpts[ii][jj]++;
```

```
                    pavg[ii][jj]+=p;
                    }
                }
            fscanf(infile,"%*s %*s %*s %*d, %*s %*s");
            }

        for (ii=0;ii<n_phi;ii++){
            for(jj=0;jj<n_theta;jj++){
                    if (pnpts[ii][jj] != 0){
                        pavg[ii][jj] = pavg[ii][jj]/pnpts[ii][jj];
                        }
                    }
            }
        retval = fsetpos(infile,start);

        for (j=0;j<ntrans;j++){
            for (i=0;i<nframe;i++){
                    fscanf(infile,"%f %f %f %f %f", &time, &trans,
                            &phi, &theta, &p);
                    if (theta < -115. && theta > -180.) theta=theta+360.;
                    ii = (int)(((phi-30.)/80.)*n_phi- 0.5);
                    jj = (int)(((theta+65)/310)*n_theta - 0.5);
                    if ((ii >= 0 && ii <= 35)&&(jj>=0&&jj<=35)){
                        prms[ii][jj] += (pow((p-pavg[ii][jj]),2));
                        }
                    }
            fscanf(infile,"%*s %*s %*s %*d, %*s %*s");
            }

/*  Not sure I'm calculating prms correctly --- CHECK  */

        for (ii=0;ii<n_phi;ii++){
            for(jj=0;jj<n_theta;jj++){
                    if (pnpts[ii][jj] != 0){
                        prms[ii][jj] = prms[ii][jj]/pnpts[ii][jj];
                        prms[ii][jj] =(float)sqrt((double)prms[ii][jj]);
                        }
                    }
                }

            fclose(infile);
            p = 0.0;
/*          side[0] = 'L';*/ /* Just set it for now */
/*      Write the output file: for now do as phi,theta + stats          */
            fprintf(outfile,"TITLE= ""stat%s""\n",outname);
            fprintf(outfile,"VARIABLES = x, y, z, pmax, pmin, pavg, pnpts, prms\n");
            fprintf(outfile,"ZONE T=acet, I=%d, J=%d, F=POINT\n",n_phi,n_theta);
            for (jj=0;jj<n_theta;jj++){
                theta = ((((float)jj+ 0.5)*310./n_phi)-65.);
                for (ii=0;ii<n_phi;ii++){
                    phi = ((((float)ii+ 0.5)* 80/n_phi)+30.);
                    cartcalc(phi,theta,p,px,py,pz,side[0]);
                    fprintf(outfile,"%f %f %f %f %f %f %f %f\n", \
                            x[0],y[0],z[0],\
                            pmax[ii][jj],pmin[ii][jj],pavg[ii][jj],\
                            ((float)pnpts[ii][jj]),prms[ii][jj]);
                }
            }
            fclose(outfile);
        }
}
```

CARTCALC.C

```
/*******************************************************************/
/*      cartcalc.c                                                 */
/*          Just gets Cartesian coordinates for phi, theta, P      */
/*      For representing pressure in acetabular or femoral         */
/*      coordinates as a line, length proportional to P.           */
/*                                                                 */
/*          Jan 2, 1993 K.C.                                       */
/*******************************************************************/
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#define PI   3.14159

void cartcalc (float phi, float theta, float p, float *px,
         float *py, float *pz, char *side)
{
   int i;
   double dtr;

   dtr = PI/180.0;

   *px = (float) sin(phi*dtr)*cos(theta*dtr);
   *py = (float) sin(phi*dtr)*sin(theta*dtr);
   *pz =  - (float)(cos(phi*dtr));
   if (side[0] == 'L'){
          printf("L\n");
          *pz = (float) cos(phi*dtr);
         }
   px++;
   py++;
   pz++;

   *px = (float) (p+1)*sin(phi*dtr)*cos(theta*dtr);
   *py = (float) (p+1)*sin(phi*dtr)*sin(theta*dtr);
   *pz = - (float) ((p+1)*cos(phi*dtr));
   if (side[0] == 'L') *pz = (float) ((p+1)*cos(phi*dtr));

   px--;
   py--;
   pz--;

   return;
}
```

Bibliography

[1] Eric K. Antonsson. *A Three-Dimensional Kinematic Acquisition and Intersegmental Dynamic Analysis System for Human Motion.* PhD thesis, Massachusetts Institute of Technology, 1982.

[2] G. Bergmann, F. Graichen, and A. Rohlmann. Five month in vivo measurement of hip joint forces. In *XII Int'l Congress of Biomechanics*, 1989.

[3] G. Bergmann, F. Graichen, and A. Rohlmann. Instrumentation of a hip joint prosthesis. In *Implantable Telemetry in Orthopaedics*, 1990.

[4] R.H. Brown, A.H. Burstein, and V.H. Frankel. Telemetering *in vivo* loads from nail plate implants. *J. Biomech*, 15, 1982.

[5] R.G. Burgess and R.W. Mann. A precision pam-fm multichannel implantable patient-monitor telemetry system. In *Proc., IEEE 9th Ann. Conf. of the Eng. in Med and Biol. Soc.*, volume 3, pages 1501–1502, Nov 1987.

[6] Charles E. Carlson. *An Instrumented Prosthesis for Measuring the Cartilage Surface Pressure Distribution in the Human Hip.* PhD thesis, Massachusetts Institute of Technology, June 1972.

[7] Charles E. Carlson, R.W. Mann, and Harris W.H. A radio telemetry device for monitoring cartilage surface pressures in the human hip. *IEEE Transactions on Biomedical Eng.*, BME-21:4, July 1974.

[8] Kjirste Carlson. Programs for processing pressure data. Interoffice document, 1993.

[9] Kjirste L. Carlson. Local pressures in the human hip joint *in vivo*; correlated with motion kinematics and external forces. Master's thesis, Massachusetts Institute of Technology, 1986.

[10] Tatiana Carvajal. *In Vivo* pressure distribution data from the human hip joint. Master's thesis, M.I.T., 1985.

[11] M.C. Clayton, K.L. Carlson, W.A. Hodge, and R.W. Mann. Postmortem cartilage changes due to *in vivo* hip pressures. In *38th Annual Meeting, Orthopedic Research Society*, 1992.

[12] Frank C. Conati. Real-time measurement of three-dimensional multiple rigid body motion. Master's thesis, M.I.T., June 1977.

[13] T.A. English and M. Kilvington. *In Vivo* records of hip loads using a femoral implant with telemetric output (a preliminary report). *J. Biomed. Eng.*, 1, April 1979.

[14] A.C. Hall, J.P.G. Urban, and K.A. Gehl. The effects of hydrostatic pressure on matrix synthesis in articular cartilage. *Journal of Orthopaedic Research*, 1991.

[15] W.H. Harris and C.B. Sledge. Total hip and total knee replacement (part 1). *New England Journal of Medicine*, 323, Sept. 1990.

[16] N.J. Hogan. Adaptive control of natural joint stiffness by antagonist muscles. *IEEE Trans. on Automatic Control*, 1981.

[17] Patrick J. Lord. Real-time analysis and display of human movement. Master's thesis, Massachusetts Institute of Technology, 1989.

[18] Robert W. Mann. Understanding synovial joint mechanics.

[19] Peter K. Mansfield. *A Large Volume Close-Range Photogrammetric System*. PhD thesis, Massachusetts Institute of Technology, June 1990.

[20] J.P. Paul. *Forces at the Human Hip Joint*. PhD thesis, University of Glasgow, Scotland, 1967.

[21] Paul D. Rushfeldt. *Human Hip Joint Geometry and the Resulting Pressure Distributions*. PhD thesis, Massachusetts Institute of Technology, 1978.

[22] N.W. Rydell. Forces acting on the femoral head prosthesis. 1966.

[23] Avram K. Tetewsky. Implementing a real-time computation and display algorithm for the selspot system. Master's thesis, M.I.T., May 1983.

[24] Tupling and Pierrynowski. Use of cardian angles to locate rigid bodies in three-dimensional space. *Med. and Biol. Eng. and Computing*, 25:527–532, 1987.

[25] Inc Voluntary Hospitals of America. Examining hip and knee implants, 1992.

[26] J. Wolff. Uber die bedeutung der architektur der spongiosen substanz. *Zentralblatt fur die medizinische Wisseschaft*, 1869.

[27] Quiang Xue. *Characterization of the Interarticular Gap in the Loaded Human Hip Joint*. PhD thesis, Massachusetts Institiute of Technology, January 1991.