REAL-TIME ANALYSIS AND DISPLAY OF HUMAN MOVEMENT

by

PATRICK JEAN LORD

S.B., Mechanical Engineering
Massachusetts Institute of Technology (1987)

SUBMITTED TO THE DEPARTMENT OF
MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1989

Signature of Author_____
Department of Mechanical Engineering
May 12, 1989

Certified by_____
Professor Robert W. Mann
Thesis Supervisor

Accepted by_____
Professor Ain A. Sonin
Chairman, Departmental Committee on Graduate Studies

REAL-TIME ANALYSIS AND DISPLAY OF HUMAN MOVEMENT

by

PATRICK JEAN LORD

Submitted to the Department of Mechanical Engineering
on May 12, 1989 in partial fulfillment of the
requirements for the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

This research has been directed towards the development of real-time kinematic data acquisition for the study of human locomotion in the Newman Laboratory for Biomechanics and Human Rehabilitation. The goal of this thesis was to demonstrate that real-time motion analysis is feasible without the expenses of high-speed supercomputers and show that invaluable interactions between clinicians and patients during gait experiments would result from the intrinsic swiftness of the system.

First, an interactive computer graphic display was developed to resolve the problems of interpretation of three-dimensional motion and force plate data collected, and of the presentation of these using planar projections on a two-dimensional computer screen. The implementation led to results that are now presented in a pictorial rather than numerical output, thus being an important visual aid to the ultimate users, clinicians, usually more familiar with visual analyses than numerical interpretations. This automatic three-dimensional display models each segment of the lower extremity as a geometrical prism while presenting simultaneously gait relevant information such as the clinical joint angles.

Then, a real-time multi processor software based on a pipelining architecture was designed to estimate six degree of freedom information from Selspot data as well as foot/floor force interactions through the Kistler platform interface. However, in order to accomplish such a processing scheme, a software development package for VMEbus base CPU boards had to be implemented.

Finally, both programs were combined via a complex network communication method to bring about a real-time gait analysis system. Issues involving the design of real-time algorithms and high speed data transfer between computers were identified and evaluated. In addition, the complex requirements for the man-computer interface as well as the compatibility with existing software have been addressed.

Thesis Supervisor: Robert W. Mann, Sc.D.
Title: Whitaker Professor of Biomedical Engineering
Department of Mechanical Engineering

Acknowledgements

My thanks begin with Professor Robert W. Mann, my thesis advisor, for allowing me to work with him in the Biomechanics Laboratory, providing direction, and introducing me to his attitude towards research which has resulted in a learning and working experience that have not only impressed me but also made me eager to pursue.

I offer my most sincere gratitudes to Professor Derek Rowell for his advice, support and many suggestions during the development of this project. I am indebted to him for all the constructive criticism he brought up as well as the warnings and lessons about how to interact with computers.

Many thanks to Peter Mansfield for his expertise, time and patience at teaching me everything critical about hardware, maintenance and debugging. Especially showing me how patient and calm one has to be with hardware as well as to never give up to software bugs.

Thanks to the residents of the "*South Mezzanine*", Greg Brown, Kjirste Carlson, Keita Ito, Sylvain Levesque, John Mansfield, Pete Mansfield, Crispin Miller and Mike Murphy, and to all the people of the Biomechanics Lab for helping me and making my work there such an enjoyable experience.

Much heartfelt thanks to Deborah for convincing me that I could do it and write it up: I could not have done it without you!

Last, but not least, I wish to thank my Parents who, in a way, made it all possible and kept me going with their unconditional support, encouragements, efforts and contributions.

This thesis research project was conducted in the Eric P. and Evelyn E. Newman Laboratory for Biomechanics and Human Rehabilitation in the Department of Mechanical Engineering at the Massachusetts Institute of Technology, and was funded in part by Grant No. H133E80024-89 from the National Institute on Disabilities and Rehabilitation Research of the U. S. Department of Education.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Man has been fascinated with motion analysis since ancient times. Even though one might easily say that human motion studies started as early as Aristotle, the advances achieved since the 19th century have been most decisive for this discipline. Only recently, has technology reached a level allowing for more accurate and precise studies of motion encompassing the three-dimensional aspects inherent to human gait. Moreover, the development of computer graphic capabilities has opened a completely new range of fields of application for computers. Evidence of this new trend is certainly the progress achieved in medical imaging where three-dimensional reconstruction and observation capabilities are now providing clinicians with anatomical structures formerly unobtainable without physical dissections. This computer power has also been directed to speeding up the processing of information, making it thus readily available.

In the field of motion analysis, this trend is also very apparent. The study of mobility, or human motion, has focussed on the phenomenon's complexity that most of us ignore in our daily activities. Gait involves coordination and control of balance, mechanical support and movement including the central nervous system, muscle activity (not only restricted to the lower extremity of the body), the bones and the joints. In this manner,

gait analysis requires the involvement of several disciplines which can be regrouped under kinematics, dynamics/mechanics, biology, physiology, neurology and obviously anatomy, but also, to a certain extent, chemistry and thermodynamics. To analyze human motion correctly, anything that could influence the subject's motion should be avoided. Thus, gait analysis is restricted to external experimentation, and most often to the study of kinematics, electro-myography (EMG) and external forces (foot/floor interactions). The foot/floor interactions are easily studied through the measurements provided by commercially available force platforms. On the other hand, the EMG and kinematic information are not as easily obtainable. For example, without implanting electrodes deep under the skin surface, one only records the activity of superficial muscles. Ethical reasons justify the ban of invasive means for direct motion or force measurements in humans. Thus, these restrictions make acquisition of measurements necessary for gait analysis more challenging. Certainly, one of the most challenging aspects addressed in part by this study is deriving the three dimensional kinematic information from the actual human motion.

1.2 Measurement of Human Motion

Reviewing the current mobility analysis techniques is certainly the first step in understanding the difficulties in collecting spatial kinematics. In this section, the advantages and limitations of the most prominent human motion measurement techniques will be presented.

It is possible to divide the different motion analysis techniques into four different groups: cinematography, goniometry, accelerometry, stereogrammetry.

- Cinematographic techniques record the subject's motion in two dimensions. One of the simplest motion study methods uses a film or video based camera to record the movement with play back in slow motion. Obviously, this provides the user with very limited information on the actual motion aside from a permanent recording of an external

observation. In a similar manner, X-ray techniques can be used, giving invaluable information on the internal skeleton kinematics. However, the levels of radiation and toxicity to the subject are totally unacceptable for medically fit patients. In addition, the lack of three-dimensional information makes impossible full evaluation of the complexity of human gait.

- Goniometry can be subdivided into mechanical and electro-mechanical systems. Among clinicians, mechanical goniometers (i.e.: protractors) are a standard for analyzing gait: usually a physical examination will include static measurements of joint angle range of motion. While easy and inexpensive, the technique is quite inaccurate, non dynamic and impractical for the measurement of small angles. Thus, this method is more suited for evaluating the maximum anatomical flexions. In order to gain better accuracy and resolution, some studies use electro-mechanical goniometers. These consist of an articulated exoskeleton with rotational and translational potentiometers at the joints. The linkage system is positioned on the limb segments and mounted on the external surface of the skin. Such system instrumentation then directly evaluates limb-relative measurements of joint angles. The relative frame of reference precludes including inertial frame information, such as the direction of the gravity vector. Still, the system is often cumbersome. The added weight and friction affect the dynamics of the lower extremity movement and thus can significantly influence normal motion. Also, the system assumes an alignment of the mechanical brace joints with the anatomical ones. Motion of the soft tissue where the brace is attached can produce misalignment and induce error. Furthermore, quality instrumentation requires a patient-specific brace which sometimes proves difficult to design in cases of severe pathological deformity.

- Accelerometry approaches gait analysis from a different point of view. Instead of obtaining position information of the limb segments, acceleration is measured directly. Then, double integration yields the position and velocity kinematics of the segments under study. Assuming data on an initial position and velocity (for example standing still at time

t=0) one can calculate the exact solution to the double integration. However, to calculate the acceleration of each segment, the influence of gravity has to be known. This represents the major issue with accelerometry since in order to subtract the gravity vector from the resultant total acceleration (measured by the accelerometers) one has to know the orientation in the gravity field of the segments studied. Unfortunately, this information is the result one is trying to evaluate through accelerometry.

- Stereogrammetry is a technique based on photogrammetry and employs a minimum of two cameras (stereophotography). The vertical and a horizontal position of markers are collected by each camera. Then triangulation based on the position and orientation of the cameras can reconstruct the three-dimensional location of each of the markers. Thus, by positioning markers on limb members, this technique can evaluate the position kinematics of these segments as well as their respective velocities and accelerations by using, for example, a double differentiation (data noise level can in this case become a critical issue). This method is at the core of several different systems: some employ reflective-passive markers, while others use emitting-active ones. The techniques utilizing reflective markers are most often video (VICON from Oxford Metrics, or Motion Analysis) or high-speed film based. Once the data (film) are collected each recorded frame is digitized, either manually or semi-automatically, so that the vertical and horizontal positions of the markers for each camera can be indicated. The markers are usually relatively large and the sampling frequency limited to 50 Hz for most systems and 200 Hz for the most advanced ones (stroboscopic), with positional resolution relatively low due to video resolution and digitization process errors. In addition. light requirements can be distracting to the subject.

Other systems based on reflective markers use laser beams swept through the viewing volume via glass prisms (CODA-3, Advanced Mechanical Technologies). These systems usually have a very small viewing volume and a sampling frequency of up to 600 Hz. They can prove difficult in normally lighted environments where interferences due to

prism reflections can occur. Ultrasound based systems that replace light with sound are also used. Here again viewing volumes are small, the number of markers are limited, and scattered reflections make measurements difficult and complicated. Finally, an intrinsic limitation of such systems is the speed of sound as opposed to the speed of light.

The system used in the Biomechanics Lab at MIT belongs to the class of systems using active markers. These markers, usually infra-red Light Emitting Diodes (LED) are multiplexed to the data acquisition device so that only one of the markers is on at a time. The two most popular, and very similar, systems are Watsmart (Northern Digital) and Selspot (Selspot AB). The sampling frequency can reach up to 5000 Hz and is a function of the number of markers used. The cameras use lateral photo effect diode detectors (positioned on the image plane where the film usually sits in a conventional camera) to pickup the infra-red signals which radiate below the visible light spectrum (940 nano-meters). With this method, data digitizing is automatic and can be performed in a normally lighted environment with the infra-red detection providing for background light rejection. The sensitivity and resolution of the calibrated detector, lens and electronics, provide with high system accuracy.

1.3 Problem Definition

In the Newman Laboratory for Biomechanics and Human Rehabilitation at the Massachusetts Institute of Technology (MIT) there has been a continuing effort to develop, improve and enhance a human movement acquisition and analysis system. This system is used for various research projects in the laboratory and at the Massachusetts General Hospital (MGH) Biomotion Laboratory where investigations are performed on normal and pathological human movement.

There is also in the laboratory ongoing research to develop Computer Aided Surgery Simulation (CASS). This system will use kinematic data and musculoskeletal

models coupled to computer tomography scans which provide patient-specific morphological and physiological characteristics [1]. Current technology is offering clinicians, especially surgeons, very little assistance during the process of diagnosis and evaluation of proposed procedures for correcting physical medical abnormalities. For example, gait defects arise from many variable sources and each case is very patient specific. Thus to assess this problem, the CASS software package for manipulating, modifying and measuring surface representations of the human anatomy has been developed in the Lab [2]. However, the diagnostic evaluation of a patient's gait, along with the post-operative progress and the projected result simulation can only be obtained from a gait analysis system. It thus becomes apparent that a display that could present human movement information effectively would be an important step in the future development of an integrated kinematic and CT/NMR data computer aided surgery system providing computer graphic displays of a patient's explicit geometrical and physical anatomy together with the kinematic gait information.

A complete description of the data acquisition system and the software approach can be found in Erik K. Antonsson's thesis [3]. However, we will present here a brief description of the system used in the Biomechanics Lab for those unfamiliar with it.

Data acquisition at MIT employs a set of opto-electronic Selspot II infra-red sensitive cameras used to measure the motion of markers. In our configuration the system uses two cameras to establish the location of up to 16 infra-red LED's (the maximum capacity of the Selspot II system is 128 LED's). The LED's are flashed sequentially by the Selspot interface. Then, the position of an LED at a camera is determined as two voltages representing the u and v position of the projection of the LED on the camera focal plane. These voltages are digitized as numbers between 0 and 4096 and sent to the computer. Each LED has four numbers attached to it, an u and v position for each of the two cameras. A two camera Selspot system is capable of monitoring the position of 16 LED's at up to 312 times a second (for 1 LED, a sampling speed of 5000 Hz can be achieved). The data

processing is done with the TRACK software developed in the Biomechanics Lab at MIT. TRACK, the Telemetered Real-Time Acquisition and Computation of Kinematics software, automatically acquires and processes 3-D gait movement. Using uniquely designed arrays of infra-red markers (Figure 1.1) composed of four 3 LED clusters (4 channels each), the camera signals, once processed, become kinematic information of the array measured within a translation accuracy of 1 to 1.5 millimeters and a rotation precision of 0.6 to 1.0 degrees in a 1.5 meter viewing cube [4]. During the collection process a piezoelectric Kistler platform built in the laboratory floor records the force magnitudes and directions at the floor-foot interface.

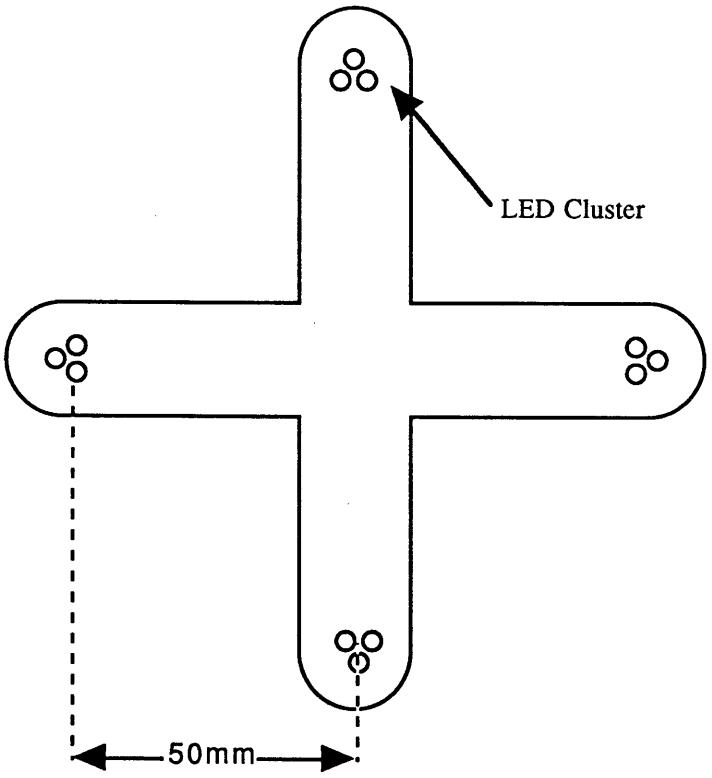

Figure 1.1: Infra-red LED Standard Array

Overall gait data acquisition with TRACK is based on the following concept: rigid arrays containing the specific geometries of the LED's are firmly attached to the patient's

lower extremity segments. Then the position and orientation of each segment can be determined as long as at least three valid LED clusters per array are recorded by the cameras. Although in principle, the array would produce valid data mounted anywhere on the segment (assuming it to be a rigid body), attention is paid to the position and orientation of each array with respect to the body segment. Each array is fixed on the patient's segment so as to maximize the prospect that the array will follow the anatomical/skeletal motion. Although this assumption can be challenged, it has been found to be appropriate; more information on the validity of this practice can be obtained in Robert Fijan's Master Thesis [5].

The data acquired through the Selspot cameras and processed with TRACK are very accurate and generate the kinematics of the anatomical segments through the simple expedient of having the patient cross the viewing range of the system.

Once the data have been collected, the particular problem of processing the information rapidly and displaying the human motion in a kinematically and dynamically clear and correct way has to be overcome:

- The complexity of the algorithms makes them conceptually difficult, and definitely time-consuming when run on a computer. This processing time precludes the use of any current motion analysis systems for real-time gait analysis. Typically, the "raw" data on a patient's locomotion are first collected, then saved, and finally recalled at a later time for processing. The processing time required can vary from a few minutes to several hours, or days, and, as a result, clinicians can seldom wait for the overall process. Two major issues present themselves: first, how can the data be checked for validity before the patient removes the markers and is sent home? Then, how relevant are the data if the clinician requesting the analysis does not associate him or herself with the experiment? Ideally, an interactive real-time system would provide clinicians with quantitative and qualitative gait analysis information while the patient "moves" in front of the cameras. Then, any

particularity or range of motion study can be requested on the spot by the clinician using his/her own judgement and expertise. The system would thus add invaluable information and be very similar to a standard physical examination.

- Next, the problem of displaying the information to the clinician has to be addressed. Although the TRACK data acquisition system stores human gait in an entirely quantitative way, the results have to be presented in a more qualitative manner since the ultimate users, medical personel, are usually more familiar with visual analysis than numerical interpretations.

The design/development phase of real-time analysis and visual aids described in this thesis involved four inherent major requirements:

- First, the display would employ standard TRACK gait data but would have to present a model of the lower extremity simple enough to visualize and yet graphically accurate in order to capture nuances of different actual gait characteristics. In addition, rapid and easy interpretation requires the compilation and output of the data in a rather straight forward manner. Particular attention was given to dynamic effects: the display of a recorded gait cycle should reflect the time frame during which the data has been sampled. The system should therefore represent the gait velocity as actually occurring while also allowing for "slow-motion" or even manual control of the overall display cycle.

- Second, the man machine interface had to be as simple and explicit as possible. How should a user, perhaps unfamiliar or even reluctant to use computers in his/her profession control such a display system? If the system is to be seen as a tool, it should not be frustrating to the user, nor should it require lengthy and complex command inputs. The problem of user selection of the information that he/she wants to see on the screen also had to be addressed.

- Third, a real-time processing scheme had to be developed. Several options were available, but cost-to-performance criteria indicated a multi-processor based system to be the best possible solution. However, this required the design and implementation of a

software development package for parallel multi-processing. Such a system had to be stable, simple, and easily modifiable to match the inevitable introduction and evolution of new hardware.

- Finally, the TRACK processing software had to be updated for a completely new computer hardware system as a first step towards Real-Time TRACK. This real-time software also had to be compatible with existing software to allow for better integration within the gait analysis tools available in the Newman Lab.

In addition, the intent of this research project was also to provide the environment for future developments, enhancements and additions to the presentation and analysis of human movement. The different aspects of the software described in this thesis should be considered as proof that real-time six degree of freedom kinematic data acquisition is now possible, thus opening the doors to "interactive" clinician/patient gait analysis. This research resulted in a software development package for multi-processing (MCC) and the Real-Time TRACK (RT-TRACK) gait analysis system. Their respective design, implementation and characteristics are presented in the following chapters.

Chapter 2

Kinematic Data Process and Display

2.1 TRACK

An important step in the development of Real-Time TRACK was that of reviewing how TRACK works. It was deemed important to understand the underlying theory so as to better grasp the algorithms in the present TRACK system as well as the complications and problems that might arise in a real-time implementation. The original work that led to what is known as the MIT TRACK system was done by Frank Conati in 1977 [6]. Originally, TRACK was a Fortran computer package aimed at investigating the nature of human mobility. Conati implemented the various algorithms, interfaced the original hardware (Selspot I) and applied the Schut rotation algorithm which had been first used by Lenox [7] in a clinical study on eye movement. Conati then linked all this theory with the infra-red optical position measuring hardware. Tetewsky [8] then worked on implementing a less general system which would be capable of completely analyzing a set of data several times a second. He also started solving ill-conditioned numerical algorithms in the software. Then, Ottenheimer [9] finished removing the last ill-conditioned states in the subroutines and worked on the first full-scale real-time TRACK. Next, Antonsson [3] brought the TRACK batch system up on a new faster computer, developed an internal camera calibration technique to take into account nonlinearities in the camera optics, and tested the system for kinematic studies. Finally, Mansfield [4] introduced and interfaced

the Selspot II hardware offering 4 times the resolution of the original system. This led to version 4 of the software, also known as TRACK IV. Thus, the TRACK software has gone through several iterative design and modification stages over the past twelve years. The software was first implemented on a PDP 11/40 and then on a faster PDP 11/60. However, throughout its revisions and updates, TRACK has remained upward compatible.

Since TRACK IV, the version of TRACK prior to this thesis, represents a milestone in the history of the MIT TRACK system, it is necessary to present here a brief description of the software so as to better highlight the differences with TRACK5[1]. The TRACK IV system consists of a series of Fortran callable subroutines which perform data analysis operations. Communication to these subroutines is through labeled common blocks. In its non real-time configuration, several main programs are executed sequentially; each reads an input work data file, calls one data analysis subroutine, and writes the resulting data into an output workfile to be passed to the next main program. Thus intensive reading and writing from the system permanent storage media, either hard disks or data cartridges, is common. This is dictated by the hardware architecture: the PDP 11/60 has a memory management system, so that a program can exist anywhere in memory from 34 to 124 kilobytes (limitations of the RSX-11M operating system) while data can fit the rest of the memory up to 256 kilobytes. Definitely, the amount of RAM available in the 11/60 is a limiting factor and a constraint in the design of the software.

The first processing step performed, after checking for the intensity of the sampled signal, is to account for the camera lens non-linearities. These non-linearities are due to the imperfections in the optics and the lateral photo effect diode field detector. Though for many experiments this camera correction only brings up minor changes and is not an absolute requirement, it becomes indispensable where more absolute precision is needed. The camera correction tables are maps of the local camera data produced by an LED

---

[1] Note the TRACK convention change with version 5: TRACK5 and not TRACK V

mounted on a Cartesian coordinate plotter and moving to a set of a discrete locations with a positional resolution of 1/1000 of an inch. From this data, a grid of correction values is established for each camera and put into two 51 by 51 arrays (horizontal and vertical directions). Since the Selspot horizontal u and vertical v camera output coordinates can vary from 0 to 4095, a simple four point interpolation is required: there are only correction points every 80 Selspot units.

Once the camera corrections have been performed, the program proceeds into the three dimensional computation for each of the LED markers. The data are reduced from a set of four coordinates (u and v for 2 cameras) to a set of three spatial coordinates x, y and z. The algorithm is based on a set of simple geometrical relationships that can be compared to a triangulation. With TRACK IV, one of the cameras is considered to be at the global space origin and is used to reference any other point in space. Since both cameras are mounted on an optical bench, it is then possible to translate the 3-D information to any world axis as long as the vertical distance from the lab floor to the camera focal points, the distance between the two cameras, and the camera rotation angles about their vertical axes are known. The spatial triangulation usually gives non intersecting virtual rays because of noisy data (ideally, virtual rays drawn from the cameras to the marker position, through the lens focal point should intersect in space: the intersection point is the 3 dimensional position of the marker). Thus a skew ray error factor is defined and accounts for the magnitude of the mutual perpendicular between the two virtual rays. The 3-D position of the marker is defined as the midpoint of this mutual perpendicular. If the error is larger than a defined threshold value (e.g.: due to reflection), the data for the specific sampling is then rejected. The maximum skew ray error is defined by the user and is expressed in Selspot units (e.g. a maximum skew ray error of 10 would represent a vertical error of about 2.5 mm when 3 meters away from the camera: $\approx 0.083\%$).

Additional error checking is introduced at this level. Since the exact geometry of the array is known, the actual inter-LED distance between the markers can be verified

against the estimated three-dimensional positions.The orientation calculation of each array is the next step. These computations are based on the Schut algorithm [10]. The result of this algorithm is the evaluation of the six degrees of freedom of the array in space. The first three degrees of freedom represent the translational position of an object while the other three define its angular orientation [11]. Thus, using the array fixed geometry and its local coordinate system, the six degrees of freedom about the array origin can be evaluated.

In addition to the processing steps emphasized above, the TRACK IV Software can filter the data using a 2-pass 6^{th} order Butterworth filter and perform numerical differentiation in order to obtain velocities and accelerations [12].

2.2 Computer Hardware

As TRACK was being revised, the computer industry was also improving hardware capabilities. After 10 years of development on Digital Equipment Corporation (DEC) minicomputers such as the PDP 11/40 or 11/60, the laboratory was faced with updating its hardware. The advent and success of the workstation made it clear that this would be the solution for the TRACK based system. However, in the computer dominated world today, there appears to be a bewildering variety of computer hardware commercially available. For example, many computer workstations with distinct processing speed performances are supported by different software, operating systems, and object code libraries. However, when the decision was made to acquire new laboratory hardware, very few companies had adopted the VMEbus specifications, now an industry standard. The VMEbus specifications define an interfacing system used to interconnect data processing, data storage, and peripheral control devices in a closely coupled hardware configuration. The VMEbus structure can be described from two points of view: its mechanical structure and its functional structure. The mechanical specification describes the physical dimensions of subracks, backplanes, front panels, plug in boards, etc... The VMEbus

functional specification describes how the bus works, what functional modules are involved in each transaction, and the rules which govern their behavior. By abiding to these specifications, hardware companies have been able to produce components that are compatible and interchangeable. The VMEbus thus offers the possibility of a multiple CPU based system where each CPU board can share memory and exchange data or program information with another board via the bus. However, the VMEbus needs a "master" computer in order to arbitrate the vast amounts of interrupts coming from the "slave" CPU's. The Sun Microsystem 3 series is based on the VMEbus and the Motorola 68020 processor coupled to 8 Mb of RAM and a 68881 floating point processing chip at a 16 MHz clock speed. The system offers a powerful base for computation intensive processes (number crunching) in a relatively small footprint (workstation) bundled with the UNIX Operating System. The operating system a computer uses is also a very important aspect of software development. Why select UNIX [13, 14] ? First, if UNIX appears to the "non-initiated" as very cryptic and less user friendly, it also offers to the programmer all the opportunities and advantages of meshing his/her own program with the operating system , thus extracting the full potential of the hardware. Moreover, well developed software running under UNIX on a specific system, will run on any other computer manufacturer hardware provided the operating system is UNIX. Proprietary operating systems unfortunately remain ..... proprietary! This certainly is the reason driving UNIX to an industry standard (and recently to a Department of Defense standard with AT&T UNIX System V).

The Newman Laboratory Sun System used for TRACK is a 3/160 with a 19 inch monochrome display and two Control Data Sabre 850 Mb hard disks for permanent data storage. A $^1/_4$ inch cartridge tape drive is also available for system backups and more permanent data filing. Via a VMEbus to VMEbus repeater from Performance Technologies (20 Megabits/sec), the Sun is connected to the Selspot interface and to a DR11W controller for the Kistler force platform as illustrated in Figure 2.1. Data collected by the interfaces

are downloaded on the system bus and processed locally by the Sun. However, though this system offers adequate numerical calculation performance, its graphics capabilities are very limited. The bitmap display conveys graphical information poorly, and the slow three-dimensional manipulations render animation nearly impossible. Thus, the Newman

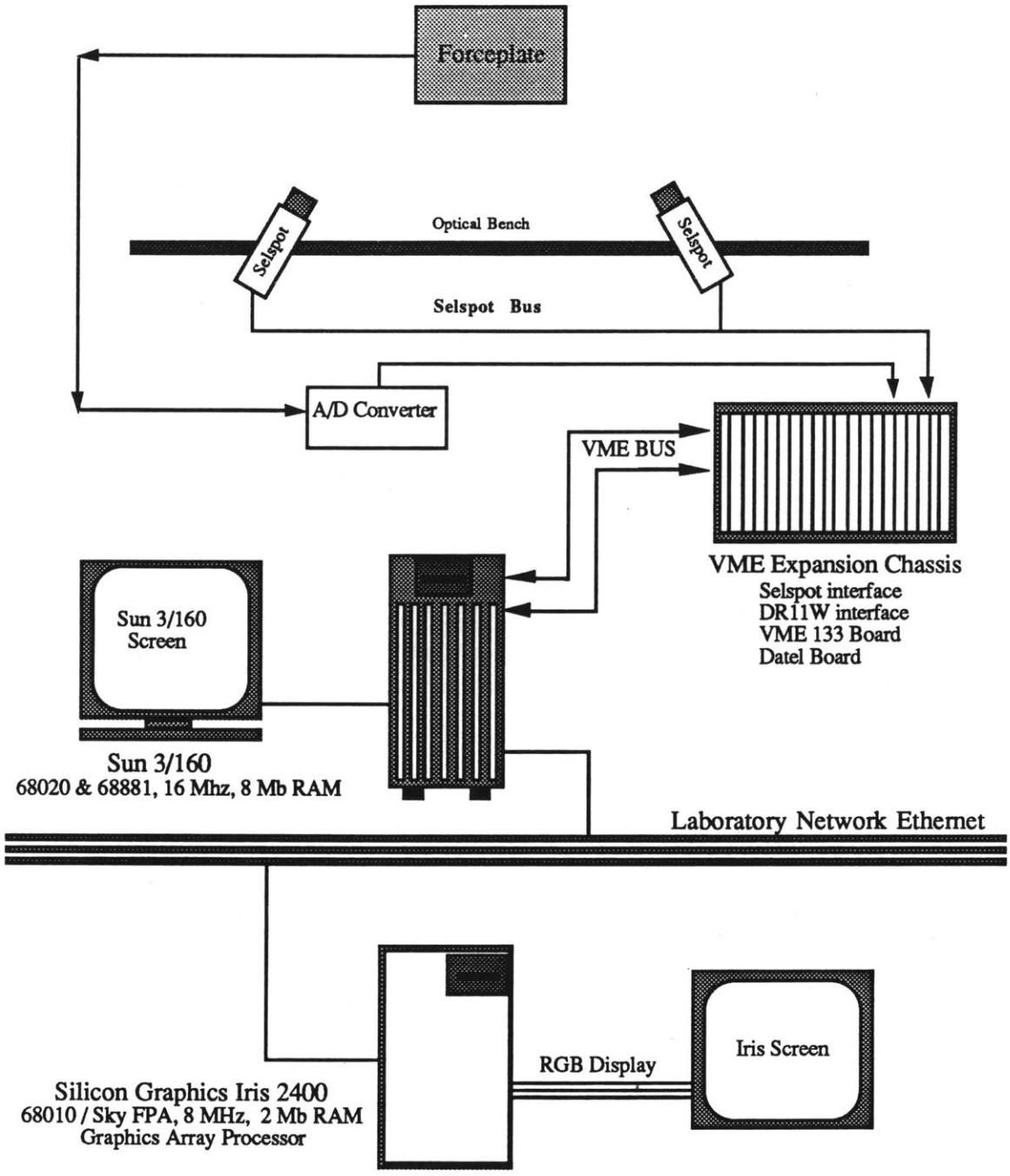

Figure 2.1: Computer Hardware Configuration

Laboratory has acquired for graphical and medical imaging purposes a Silicon Graphics IRIS 2400 workstation. The IRIS (Integrated Raster Imaging System) workstation is a high performance, high resolution color graphic system for 2-D and 3-D displays and animations. The display transformations (translations, rotations, scaling and viewing transforms), matrix manipulation, data structure management and variable computations are handled mostly by the hardware. The general purpose processor, a Motorola 68010 chip at 8 MHz coupled to 2 Mb of RAM running under Berkeley 4.3 BSD UNIX takes control of memory management, floating point calculations (with the help of a Sky floating point accelerator), and peripheral devices. Certainly, the system is not well suited for intensive number crunching, but at its heart is the proprietary "Geometry Engine" and graphics library. A pipeline of 10 Geometry Engines transforms points, vectors, polygons, characters and curves in user defined coordinate systems to screen coordinates with the use of rotations, translations, clipping and scaling in real-time. The display is a high resolution RGB color monitor offering 1024 x 768 pixels which, combined with the double buffer mode, render color animations. This is an imperative feature for an interactive gait analysis display.

2.3 TRACK5 Processing Modifications and Enhancements

The first step in this research project was to get better acquainted with TRACK IV in order to better understand the major processing components required in the design of a real time system. The Sun 3/160 recently acquired by the Newman Lab offered the formidable challenge of porting the original software to this system. Thus, this project included the software development of the TRACK processing on the Sun microsystem which led to TRACK5. Even though TRACK5 does not differ algorithmically significantly from TRACK IV, it is important to highlight some of the major structural modifications.

The procedures for solving a complex programming problem can be easily expressed in general terms, but to implement the solution on a computer, a specific programming language has to be carefully selected. Earlier versions of TRACK had been written in Fortran, more specifically in RT11 and RSX11 VAX Fortran. The software was thus not easily portable to non-DEC machines. In addition, every language has special characteristics that dictate the style of coding necessary to achieve satisfactory results. The selection of an adequate language does not seem crucial for a simple program, but as the size and complexity of the program increases, stylistic considerations become more important to the success of a project.

The C language was chosen as the optimum language for the design and implementation of TRACK5. This decision was due largely to four major considerations.

- First, since the software was to evolve in a UNIX environment, it seemed that the programming language used for UNIX would be very appropriate for TRACK. This would enhance the interaction between the computer operating system and the software by using direct system calls. In addition, the C programming language has definitely followed the success of UNIX and has thus become very popular among programmers.

-Then, the C language introduces techniques for dividing large programs into sets of functions and smaller files. This is "structured programming" using structure blocks. Other modern programming languages usually assemble a complex function in a single large unit consisting of smaller functions combined together. C offers a unique way to utilize functions as its basic building blocks. All functions are treated separately and create a set of tools which can be assembled differently to solve various programming problems. Proper usage of this UNIX block structure can increase program efficiency and flexibility.

-The third consideration was that C provides the ability to use pointer manipulation. The language itself is based on pointers and thus allows address arithmetic. A pointer is simply the location where data are stored in memory. The pointer "points" to the memory address where information is to be found. This allows the possibility of using Dynamic

Memory Allocation. Arrays and pointers share similar concepts in C; both are used to store data. An array is a predefined set of data. Its data type and maximum size are evaluated at compile time and it is always assumed that the data will not overrun the allocated space during run time. Thus, arrays can either be memory wastes if the implementation allocates too much space for the number of data elements used, or a system crash prospect if not enough space is allocated. However, proper utilization of pointers can overcome the drawbacks presented with arrays. At execution time, the amount of memory space desired can be evaluated and allocated to a pointer, thus using only the amount of memory necessary. Also, sections of dynamically allocated memory can easily be discarded in order to make space for new data structure memory segments as the program is running. This usually increases the speed performance (pointer arithmetic is faster) and efficiency (memory management) of a program.

- Finally, an important feature offered by C is the availability of user definable structures and data types. Structures are the basis for constructing the combination of data required for complex programming projects. They allow associated complex data to be handled as a single entity. While all elements in an array must share the same common data types (integer, float, character, double, etc...), the elements in a structure can consist of any combination of standard or user defined data types and structures, nested or recursive. Structures provide a convenient method for grouping logically related items. By constructing the program with structures, the programmer is able to define customized data type to suit a particular situation.

The software development of the processing section of TRACK5 in this research project fully utilized the four considerations described above. The outcome proves C's power, speed and versatility, and its ability to build upon itself [15]. TRACK5 offers a data structure for the header information which describes the data collected by Selspot. This header is considered as one entity even though it contains information such as the number of frames sampled, file character names and descriptions, sampling frequency,

processing options, number of markers, array specific geometry description, etc... Careful attention was paid in making the header compatible with future TRACK developments. A TRACK version number is included in the header as well as extra space to be used for future versions of TRACK. Thus, data collected under TRACK5 will be compatible with data under TRACK6, already under development [4]. By using this header and reading the information it contains first, the program can allocate the exact amount of RAM necessary to read in the data. The data are then read as a whole into the system memory and further manipulations and calculations are handled there. This eliminates the excessive disk drive read and write access as well as the bookkeeping dictated by the 256 Kb memory limitation of the PDP 11/60 computer.

Algorithmically most of the computations are similar but have been condensed to achieve maximum processing speed performance. However, some algorithms are different. The processing of the three dimensional markers can now be achieved either through the TRACK IV approach [3] or the newly developed external calibration method (default for TRACK5). Using this technique, a calibration frame is positioned in front of the cameras prior to data collection in order to establish the absolute position and orientation of the cameras in the laboratory space. The spatial coordinate origin is now at the origin of the calibration frame and can be translated to any point of interest [4]. The TRACK IV filtering options have been replaced by a smoother [16] in TRACK5 in order to avoid the controversial selection of a filter cutoff frequency. In addition, a spline smoother evaluates the derivatives through an analytic differentiation of the spline equation and not a numerical differentiation, thus being less sensitive to the the signal to noise ratio of the data sets.

TRACK5 was also implemented so as to run on any workstation operating under UNIX with at least 4 Mb of RAM. The software relies on direct operating system calls to perform bookkeeping, file handling, and directory structure definition. It is thus easily portable to another system with just the requirement of a recompilation. Through the use of new hardware and better software implementation, TRACK5 processing achieves an order

of magnitude (exactly 11.2) speed performance improvement over TRACK IV and an executable code reduction of 60% (processing size: from 1,000 to 320 Kb). Furthermore, the data structures and overall software scheme defined for TRACK5 processing as part of this research were used for the Selspot/Kistler data collection processes [4] and the graphic calling sequences to The National Center for Atmospheric Research (NCAR) plotting package.

2.4 3D-Gait Graphic Display

An interactive computer graphic display for the study of complex three dimensional human movement has been developed in the Biomechanics and Human Rehabilitation Laboratory as a preliminary approach to a real-time display [17]. Using data collected and processed with TRACK, the system can present off-line human gait information in a qualitative and quantitative fashion. Since the real-time display shares the same overall specifications as the off-line display, it is important to discuss here the principal features offered by this graphics software.

During normal gait all segments of the human body play important roles. Without denying the importance of the trunk, the arms and the head for balance and equilibrium during motion, this study concentrates on the analysis of the lower segments composed of the pelvis, thigh, shank, and foot. The problem of defining a graphical description that would best characterize the human leg during locomotion then had to be addressed.

Converting 2-D images into 3-D representations is achieved through stereogrammetry which has been used in a wide range of domains. Research at the University of Uppsala in Sweden produced one such display method previously used for 3-D motion pictures [18]. This technique is one possibility for presenting a three dimensional output of the collected data. The developers assert that complicated 3-D motion and force data are presented in a manner that is easy for the human brain to

interpret. Using a gait analysis source, the coordinate data are transformed in the computer into the two eye-views of an imaginary observer, with the two planar images displayed in different colors (red and cyan) on a graphic terminal as shown in Figure 2.2 [19]. An observer using a correspondingly colored pair of glasses sees the measured object as if it were moving in three dimensional space. Although this experiment reports that the interpretation of 3-D gait data is simplified by using the above technique, the graphics quality and resolution of such a method does not appear to be very high because of the crudity of the leg model (a stick figure), and the rather poor results characteristic of three dimensional motion pictures. Furthermore, the Swedish approach does not seem very well adapted to its potential users: medical personel. The users would have to tolerate too many constraints in order to use the system, e.g.: specific distances between the colored glasses and the terminal screen, and accurate and stationary positioning of the viewer to get optimal graphic effects.

Another example of stereophotography is used extensively on computers in the Chemistry and Biology Departments at MIT to visualize massive and complex three dimensional molecules. This technique requires two planar projections of the object, rotated at about 6° from each other, and a viewing apparatus (mirror mechanism marketed by numerous manufacturers)[1]. This technique renders high resolution realistic 3-D effects and produces sharp images. Furthermore, this approach is rather simple to implement. However, even with improved accuracy and resolution over the Swedish method, the man-computer interface is still very restrictive. Total, direct, focussed attention on a computer screen cannot reasonably be expected from a physician more inclined to use the system as a tool concurrent with a personal physical examination of the actual patient. These existing systems also lack a sense of spatial volume: much three dimensional information is lost by using just stick-figures.

---

[1] Nu 3-D VU Co., 71 East 28th Avenue, Eugene, OR 97405.

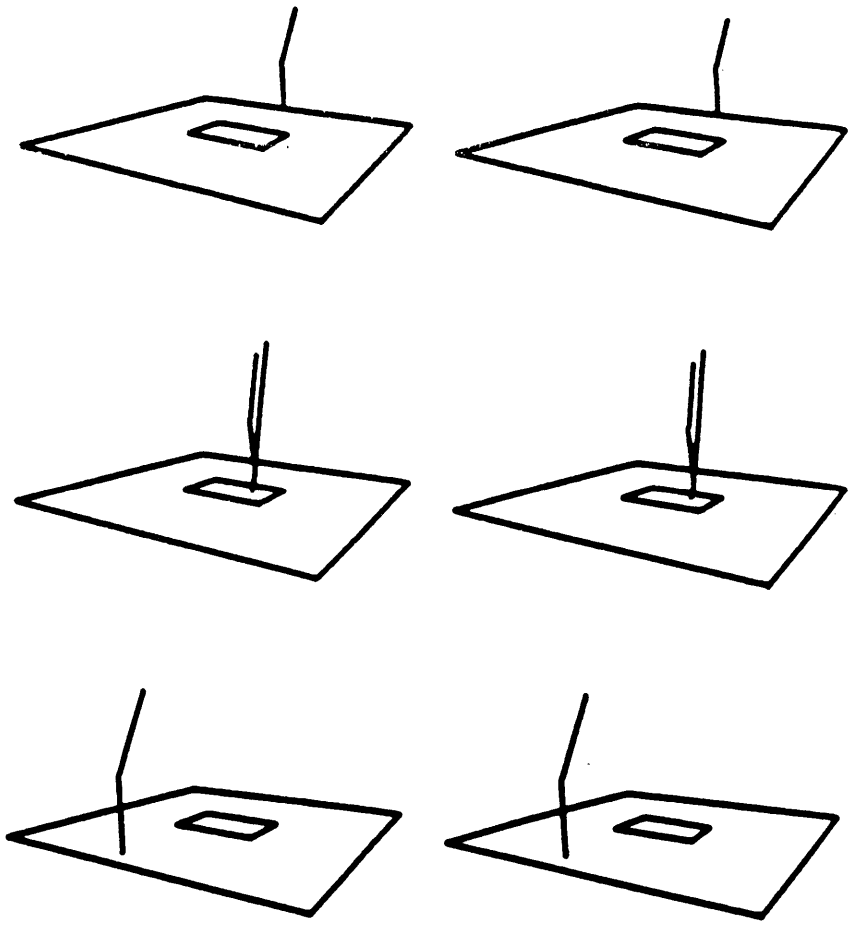

Figure 2.2: Uppsala University Gait Analysis Display [19]

The model adapted for this study involves the modelling of each segment of the lower extremity as a geometrical solid prism, and was originally developed by Fijan with a wire frame representation of the leg [5]. This technique uses planar projections on a two dimensional graphics screen using a 3-D orthographic mapping. An example of a typical screen output is shown in Figure 2.3. As can be observed, the pelvis, thigh, shank, and foot are each modeled by a cubical form which enhances the three dimensional visualization. The dimensions of each segment are proportioned to those of the patient's anatomy. The length of the foot segment is determined by the shoe size of the patient. The

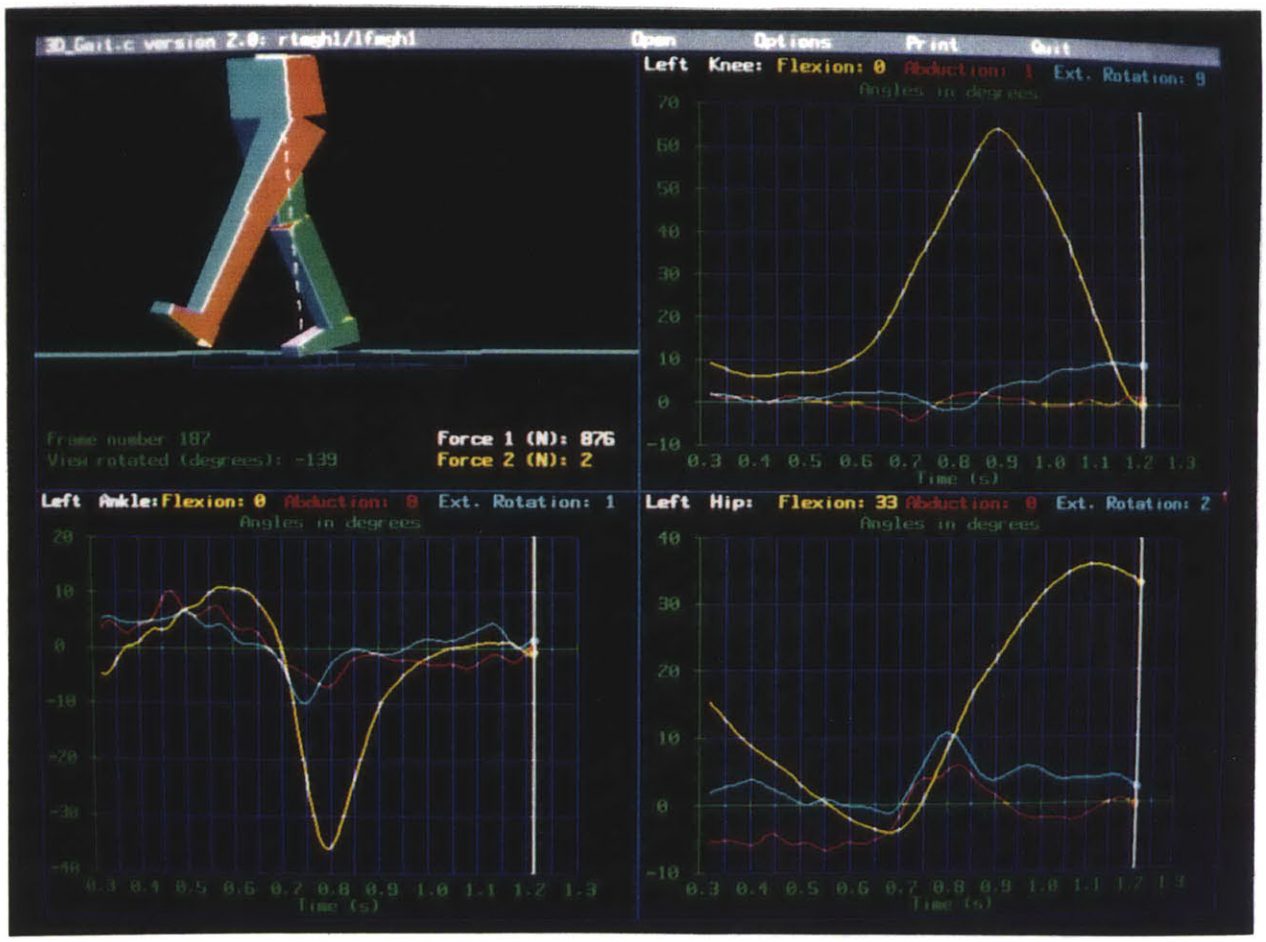

Figure 2.3: Standard 3D-Gait Analysis Screen Display

other segment measurements are performed while the patient is standing still (relaxed stance) and in the range of the gait recording system. The joint positions are quantized by using a LED equipped pointer at the ankle, the knee and the hip, respectively, the lateral malleolus, the patella, and the greater trochanter. These points are only used for length modelling and are never considered to be the actual joint rotation centers (contrary to the usual technique in many gait laboratories not using a six degree of freedom body segment approach). In addition, the circumference measurement of each segment is recorded so as to best represent the morphology of the patient.

Each panel defining one side of a segment is defined with respect to the transformation matrix of the LED array of the particular segment. The lower extremity

model consists of 17 panels for unilateral data and 34 panels for bilateral data. The leg is displayed on the screen by using a sorting algorithm. In order to achieve real-time changes of the viewing orientation of the walking legs, sorting procedures such as Z-buffering or Z-shading were eliminated. These are usually non real-time operations, computationally expensive and more suited to the enhancement in depth and perception of static images. For this dynamic display a simple technique was adopted; each panel has an associated position which corresponds to its average X, Y, Z values in the local coordinate of the body segment. By re-ordering the panels for every display frame according to the viewing conditions, the legs are constructed so that the last panel drawn on the screen is the one closest to the observer. This method offers a fast alternative to Z-buffering and produces realistic dynamic animations as illustrated in Figure 2.4. To complete this 3-D view modelling, the laboratory floor is presented including the Kistler force platform while dashed lines provide visual rendering of the magnitude and directions of the force vectors at the foot-floor interfaces.

This modeling technique has proven to be a significant improvement for clinicians attempting to analyze results of gait data collections. It offers a simple image that does not severely constrain the position of the user in front of the computer screen, and produces a realistic model of the human lower extremity. The approach described in this chapter was utilized, and confirmed, by an investigation performed at the California Institute of Technology [20]. The CalTech display offers a leg of a "*lay figure*" that has been adapted to fit unilateral TRACK gait data on an Evans and Sutherland PS300 color graphic system. In walking mode, the display creates the illusion of an anatomical leg. However, the patient specific anatomical validity of this model is not considered as forceful as the MIT prism approach. More importantly, the spherical and cylindrical geometry of the articulated leg neither accounts for geometrical data misfits at the joints, nor shows graphically the long axis rotation angles at those joints, a matter of keen clinical interest. The Newman Lab software, with the color-coded panels and prism geometry, provides a significantly

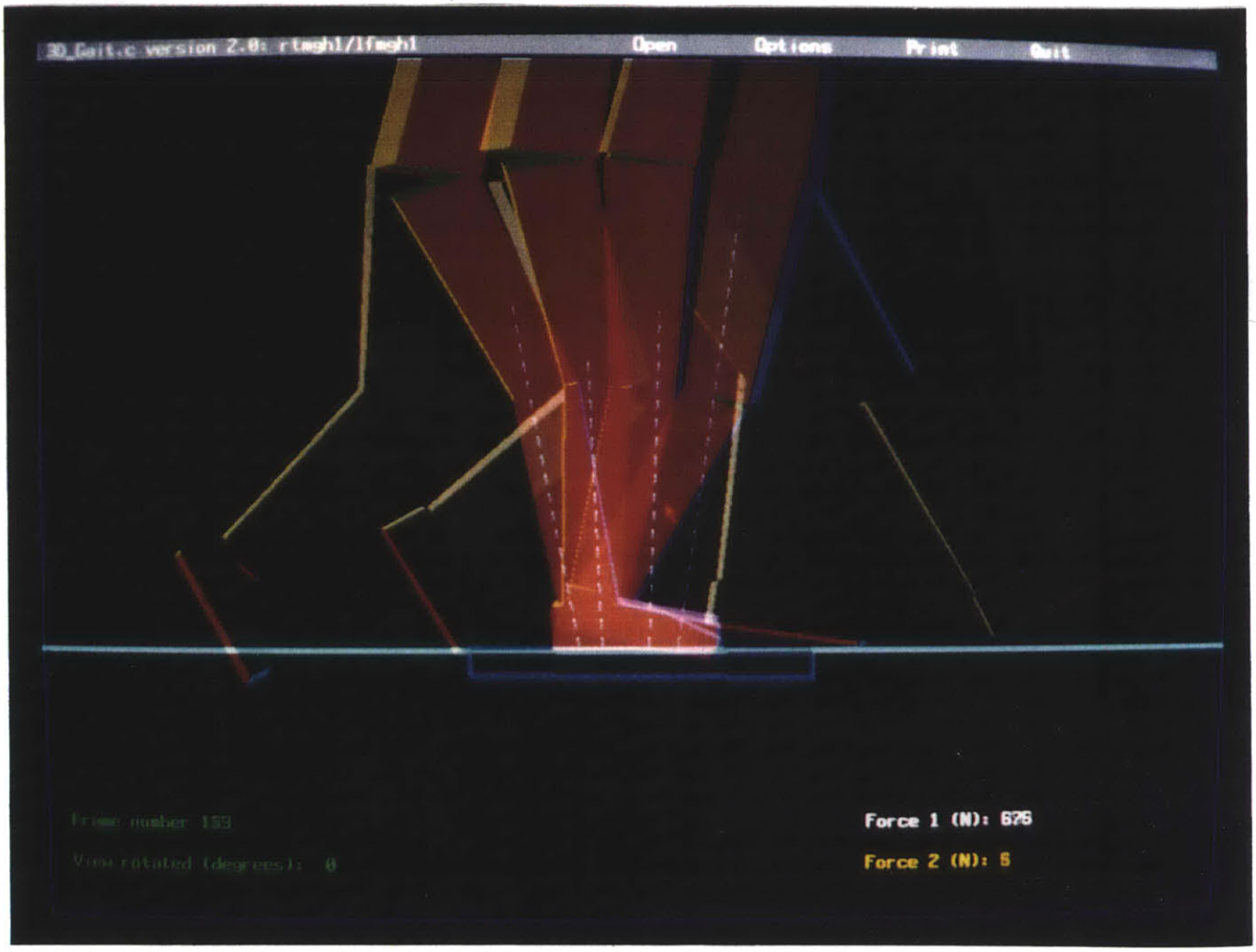

Figure 2.4: Dynamic Walking Display

better model for assessing the joint internal or external rotations. Furthermore, foot abductions and adductions are also presented effectively. This qualitative model of the human lower extremity is an important visual aid but should also be coupled with quantitative presentation of the same data. In this respect, clinicians consider the anatomical joint angles to be key elements in the analysis of human locomotion. Thus, a detailed explanation on how these joint angles are evaluated and extracted from the TRACK kinematic data will be provided in chapter 4.

Chapter 3

Multi-Processor Software

3.1 Introduction

The pipelining software development described in this chapter has also been completed at the Newman Biomechanics Laboratory on a Sun 3/160 workstation connected to a VMEbus expansion chassis. The objective of this project was to implement a software package that would let a user develop software on the Sun system, then compile, link and download them to a Motorola VME133 board located in the expansion unit. Several approaches were considered. The project was completed and the set of tools developed was used in increasing the performance of the Real-Time TRACK system. The project has also yielded a simple and efficient, although not complete, method for developing code on VME133's, as well as parallel coding schemes between several processors for various real-time applications.

In this chapter, the objectives of the project, a brief description of the hardware, a list of the advantages and limitations of the software, as well as several coding examples and performance evaluation will be presented. Finally, some conclusions and recommendations for future development and use of the pipelining software architecture will be drawn.

3.2 Objectives

The objectives for the Pipelining Software Development Package can be summarized as follows:

- The first step was to implement a downloading routine. The approach was to try to take full advantage of the VME to VMEbus repeater between the Sun and the VME133. This would then provide a fast downloader when compared to a standard serial line. This also implied that S-record format would not have to be used to load the program on the VME133 thus simplifying the coding of executable code for the off-line board.

- Then a compiler scheme had to be developed in order to compile source code on the Sun for the VME133. This also implied that the appropriate libraries, such as *stdio.h*, *math.h* would also have to be provided with the compiler. There was no need for implementing a compiler per se, since the Sun's *CC* compiler could be used with specific options.

- The third step was to automate and document the process and make it as transparent as possible to the user. The goal here was to provide a process that would be similar to the one used during the compiling process on the Sun with which the user would already be familiar. Moreover, a UNIX "Makefile" facility was provided to allow for easy distribution of the source code as well as convenient updating and debugging of the software.

- Finally, several examples were implemented to illustrate the use of the development package. Examples of "dual processor" computations were greatly emphasized, especially for the TRACK processing from a speed performance point of view.

The above objectives have been reached successfully. The project does not have the versatility of available commercial packages, but it definitely is less intricate and satisfies the laboratory hardware requirements. The limitations will be discussed further

down in this chapter, but let's first better understand the hardware configuration, the approach used, as well as why such a project is relevant for certain applications.

3.3 Design Considerations

The decision to follow the approach used at the Artificial Intelligence (AI) Laboratory at MIT with the CONDOR project was based on the design methodology presented by Siegel [21] and the similarity between the requirements of our applications and those of the Utah-MIT hand project. In addition, the observation that computations needed to acquire and process data in real-time are suitable for general purpose computer architectures made it clear that multiple single board computers would be the solution to the real-time processing of Selspot data if the algorithms involved could be divided among several processors running in parallel.

The expertise in this domain available from the AI Lab, with even the possibility of utilizing CONDOR for real-time TRACK, played a major role in our decision process. Moreover, the success of CONDOR in controlling the Utah-MIT hand suggested that the hardware setup based on the VMEbus, that had been used by the AI Lab, would represent an economical alternative to the acquisition of fast, but expensive mini-computers. Table 3.1 compares the price to performance ratio of some of the CPU's then available that were considered.

Table 3.1: Hardware Configuration Performance to Price Comparisons.

Processor Type	Instruction Speed	Cost	Remarks
DEC Microvax II	1.0 MIP	mod	Non VMEbus based
DEC Vax 11/750	1.0 MIP	high	Non VMEbus based
Intel 8086	1.0 MIP	low	PC system
Intel 80286	1.0 to 1.5 MIPS	low	PC system
Intel 80386	1.0 to 3.0 MIPS	mod	
Motorola 68000	1.0 to 1.5 MIPS	low	Lacks floating point
Motorola 68010	1.0 to 2.0 MIPS	low	
Motorola 68020	2.0 to 3.0 MIPS	low	32 bit processor

Since our system was to be designed as a research tool, it was desirable that it be highly flexible and expansible while at the same time meeting certain performance goals. The goal of flexibility and portability at the software level partially dictated our choice of C and UNIX as the basis of our development environment, and also led to a conscious effort to minimize machine-specific assembly language coding. The system is comprised of a large number of software libraries and a set of stand-alone programs. The development software is to a large extent independent of the real-time processes programs that have been written.

3.4 Background and Hardware

The Sun 3/160 microcomputer system of the Newman Biomechanics Laboratory can be schematically represented with its expansion chassis as illustrated in figure 3.1:

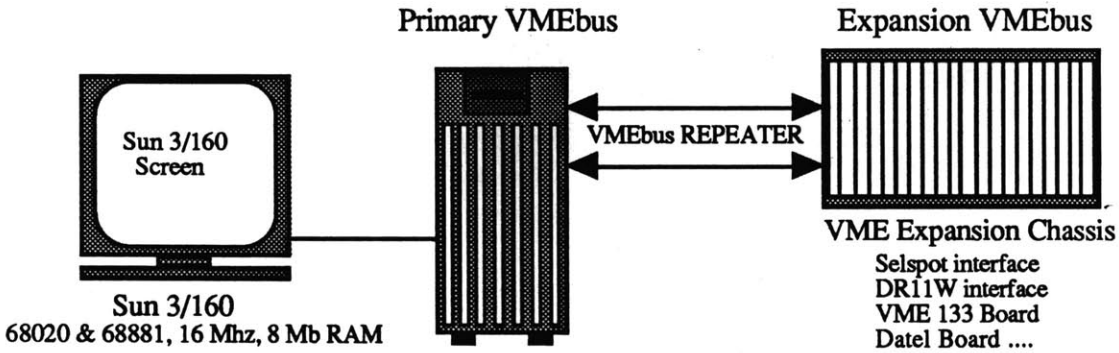

Figure 3.1: VMEbus System Configuration

The Sun 3/160 is based on a 16 MHz Motorola 68020 processor with a 68881 floating point co-processor. It has 8 mega-bytes of RAM on board in its present configuration. Data and files storage is done on two 850 Mb hard disks. The workstation screen allows for graphics as well as multiple window display. The system runs AT&T UNIX system V with Berkeley's 4.3 extensions. The Motorola VME133 is also based on a 68020 with a 68881 running at 12.5 MHz with 1 Mb of RAM on board. A Datel board

for fast Analog to Digital (A/D) and D/A data acquisition, as well as a Selspot double board composed of a Motorola MVME333 and of a Selspot MCIM board (Motorola/Camera Interface Module), can also be found in the expansion chassis along with a Creonics servo-motor controller and a DR11W DMA interface. The expansion chassis is connected to the Sun's backplane through an HVE 2000 repeater. This can be viewed as an extension of the Sun's own VMEbus. However, the expansion unit does not sit on the ninth megabyte, after the Sun's own 8 Mb of on board RAM, but rather in a virtual address space. This space is called the *24d32* address space and is part of the *32d32* memory space. This jargon is most probably most confusing to the reader not familiar with the VMEbus architecture [22]. Figure 3.2 offers a good schematic of the memory addressing of the Sun-3 computers.

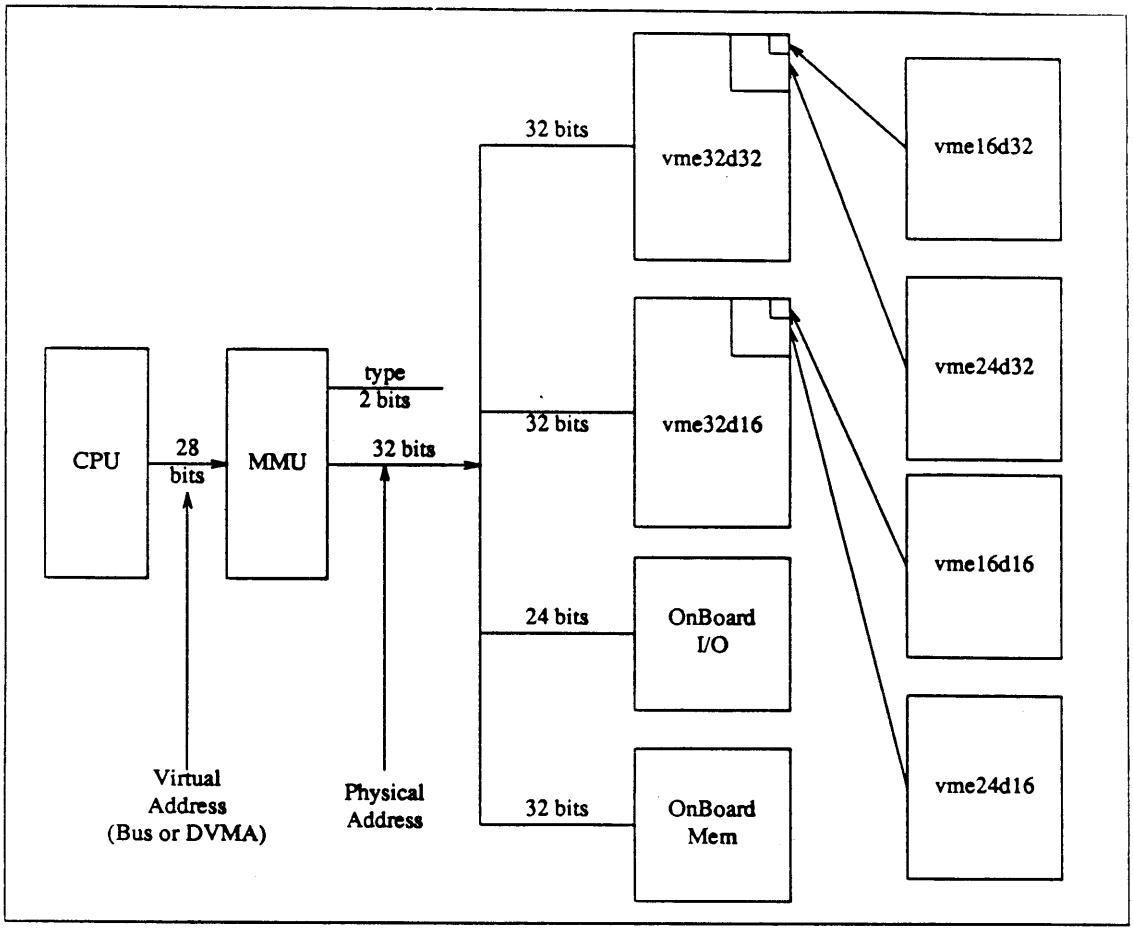

Figure 3.2: Sun 3 microsystem DVMA memory space map

In other words, *24d32* (24 address lines and 32 bit data transfers) is virtually addressed through the Memory Management Unit (MMU) by the Sun's CPU. It should thus be kept in mind that if it is easy to read and write from the Sun to the expansion chassis, the reverse is however not readily possible since no mapping of the DVMA space exists in the expansion chassis in the direction of the Sun. The Sun should therefore be considered, for the whole system, as a *master* and any board in the expansion unit as a *slave*.

With such a system, it is rather easy to set up two or more processors to work independently of each other as well as in concert in order to achieve parallel processing, and more precisely Pipelining. This was the first and foremost motivation for this part of the project: to be able to produce code for different CPU's in the expansion unit so that they could be run concurrently with the Sun. This was compelling primarily for real-time computationally-intensive data acquisition. If the computation can be shared among several processors, the sampling frequency can be significantly increased as the overall computation time is reduced.

Let's now present the core of the project: the software package. This will only be a brief description of the overall process. The avid programer will certainly find answers to all his or her questions in Appendix A where a complete listing of the software is given.

3.5 Pipelining Software Development Package

This section will review briefly the developed software. Emphasis will be on four major aspects: the downloader, the *CC* compiler scheme, the libraries and the program examples.

Let's first consider the approach used for the downloader. Using the VME to VME connection and the virtual memory address mapping, writing binary code to the VME133

can simply be reduced to first opening the DVMA space at the board's first byte memory location, and writing successively in long words (4 bytes, 32 bits) the binary executable code. The long word writing allows for fast downloading and uses the full 32 bit addressing capability of the VMEbus. In this case, a program was implemented to let a user load a program at any memory location on the board. The source code of this program can be found in Appendix A under the name startup.c. The user only has to type *startup* followed by the executable program name (he/she can also wait to be prompted for a file name by the program) to download the code on the Motorola VME133, or any other board in the expansion chassis (e.g.: in the case where several single board computers such as the VME133 are used). The user has the option of specifying the local (on board) memory location where the code should be loaded. However, a default value of 10000^{HEX} can be used automatically and is recommended since it provides ample space for the heap stack generated by the processor. This downloader has proven very effective and extremely reliable, and is now used by other students for their own software development.

The compiler scheme is called *mcc* for "Modified CC" compiler and the source code can be found in Appendix A. This program parses the input command which is very similar to the usual *cc* command (i.e: the UNIX call "cc foo.c -lm -f68881 -o foo" becomes "mcc foo.c -lm -f68881 -o foo"). However, this software will link the source program with the appropriate library for the VME133. Any valid *cc* option will be accepted by *mcc* since that option will be passed to either the UNIX *cc* compiler or to the UNIX *ld* linker and thus will be recognized. Finally, it should be pointed out that an option has been provided that lets the user specify the machine type for which the program should be compiled. This accounts for the different specifications of the numerous available single board VMEbus computers: by default the program will assume it is handling a VME133, but, for example, a "-T vme117" flag would compile the code for a Motorola VME117 board, provided the appropriate libraries have been created (in this case one would also

have to specify the 68010 flag to the *cc* compiler since the VME117 is a 68010 and 68881 cpu based board).

In order to use the above compiler scheme, several files and libraries had to be implemented. The first one is the *crt0-vme133*, and is a file containing initialization code for the VME133. This initialization code will call the C *main* program as its subroutine. It acknowledges the existence of the 68881 floating point co-processor so the appropriately compiled code can take full advantage of the enhanced performance provided by the co-processor (generally a performance factor of 4 times is observed when using the 68881 co-processor for floating point operations). This file is extremely system dependent and will vary from computer to computer, and thus requires a great understanding of how the hardware behaves since the actual file is in most instances kept secret as a trademark protection. This is usually one of the reasons why executable code will not work on different computers even though they share the same basic microprocessor. For example Sun 3's, Apple MacII's, Silicon Graphics Iris 3000's, Motorola VME133's and Ironics board all share the same basic architecture (68020/68881) but cannot share the same executable code. Another very important library that had to be created is the *libc-vme133.a*. It contains the Input/Output (I/O) information, and more specifically a subset of the standard I/O routine of the C Language. Subroutines to simulate *printf* and *scanf* have been implemented and rely on the use of the assembly language *TRAP #15* I/O calling procedure. Most of the formatting has been implemented as well, and the definitions used were based on Kernighan and Richie's *The C Programming Language* book [23]. Finally to allow for more complex computations, a math library (*libm.c*) was created. It uses the 68881 for standard trigonometric function calls (*sin, asin, cos, tan, ...*). Though not as accurate as the subroutines complying to the IEEE Standard 754 for Binary Floating Point Arithmetic, these hardcoded trigonometric functions are expeditious (by a factor of 2) and

the accuracy (up to 6 decimals) is appropriate in most applications[1]. To simplify the software, double and floating point calculations are also handled via the 68881 processor. No 68020 float computation is possible on the VME133 (anyway, an application using floating point elements should use the 68881 capabilities !!!). The source code for these libraries can be found in Appendix A.

Finally some example programs can be found in Appendix A. These programs are examples of parallel processing between the Sun and the VME133. These programs were also used to test the feasibility of pipelining software as well as benchmarks to evaluate the speed performance achievable. One of the examples used is called Savage. This is a very well known benchmark that is usually used to compare different systems in trigonometric intensive applications. The test consists of computing the result value of

$$\text{asin}\,(\,\sin\,(\,\text{acos}\,(\,\cos\,(\,\tan\,(\,\text{atan}\,(\,\sqrt{\alpha^2}\,)\,)\,)\,)\,)\,) \qquad (3.1)$$

where α is a double (8 bytes or 64 bits). The computations involved can however be divided among several processors. For example, one processor could evaluate part of the above equation:

$$\text{RESULT} = \tan\,(\,\text{atan}\,(\,\sqrt{\alpha^2}\,)\,) \qquad (3.2)$$

while another processor calculates:

$$\text{asin}\,(\,\sin\,(\,\text{acos}\,(\,\cos\,(\,\text{RESULT}\,)\,)\,)\,) \qquad (3.3)$$

The Savage benchmark took 10.4 seconds[2] on the Sun microsystem. The same benchmark on the VME133 took only 0.5 seconds more. This minimal performance difference can be explained despite the fact that the clock speed of the VME133 (12.5 MHz) is much slower than that of the Sun (16 MHz), because the VME board does not have to account for the

---

1 Draft 796 IEEE Floating Point Standard.
2 Times represent 25000 loops averaged over 10 runs.

UNIX operating system. Moreover, the Sun complies with the IEEE standard 754 for its floating-point calculations. However, the modified (Pipeline scheme) Savage benchmark using both CPU's took only 7.2 seconds, resulting in a performance improvement of 30%. A 50% performance improvement cannot be achieved due to the overhead induced by keeping both computers in *phase*.

Using the same approach, an example applied to the off-line TRACK processing software was implemented. One of the goals here is to achieve real-time Body Coordinate System (BCS) calculations of kinematic data recorded through the Selspot cameras. One approach to achieve real-time processing is to divide the different steps among the available processors. To illustrate the feasibility of such an algorithm, an example of simultaneous camera corrections on the raw data and 3 dimensional marker position calculation (2 major steps of the overall process) has been implemented. The computation time for 2 seconds of data with 16 channels sampled at 312.5 Hz is 17.5 seconds on the Sun alone and 10.5 seconds when processing is shared by the Sun and the VME133. This represents an improvement of about 40% and is thus a good indicator that with multiple processor, real-time TRACK can be attained. This also shows that speed performance improvement based on the pipelining technique relies heavily on where the division of the process is done: to obtain the best possible result, sub-processes should have about the same execution time requirement (avoid wasted time in loops used to synchronize out of phase processes).

In order to manage the development of this software package, a rather elaborate Makefile was also implemented. The distribution of the source code is also greatly enhanced by the use of such a facility.

3.6 Limitations

Even though the approach described above has been very successful and has satisfied every requirement encountered in the development of the Real-Time TRACK version, it has some limitations that we will describe below.

First, I/O to devices such as hard disks cannot be performed. The VME133 cannot read nor write to the Sun memory and thus access the different peripherals connected to the Sun. Moreover, the VME133 is unable to interrupt the Sun CPU which implies that any handshaking between the two processors has to be performed through polled I/O, a more time consuming and less flexible approach. Finally, the libraries provided with the system do not implement a complete C/UNIX environment. For example, the memory management functions such as dynamic space allocation (calloc, malloc, etc ...) and certain low level I/O operations have been neglected. However, this software package can be regarded as a tool for implementing simple code in C, Fortran and Assembly Language, for off-line single board computers like the VME133. The simplicity of the system makes it easily adaptable to different hardware and thus increases the versatility of this software package.

More sophisticated, complex and complete approaches to the goal of the Newman Laboratory for Biomechanics and Human Rehabilitation software development system are provided by several commercially available programs. Among them, CONDOR, which has been implemented at the Artificial Intelligence Laboratory at MIT, was considered for our applications. This software was one result of the Utah-MIT hand project. This project's goal was a high performance dexterous robotic end-effector, capable of human-like performance. The Utah-MIT hand is a pneumatically powered, four-fingered hand, built to investigate issues related to machine dexterity. The hand has over sixteen degrees of freedom and needs a fast computer process scheme to control it: CONDOR. Although the computational architecture was originally developed to control the hand, the architecture that has resulted is a powerful, multi-micro-processor system that is fairly general in its scope, and is oriented specifically towards real-time computation. However, the system, which really provides a complete simulation of a UNIX environment on VMEbus single board CPU's, is extremely dependent on the hardware utilized. The only VME boards supported by CONDOR are the 68020/68881 chip-based Ironics. Moreover, the

complexity of the code, the lack of comments and the number of "hacks" make it impractical to modify the software in order to use the Motorola VME133 boards. In addition, the CONDOR package comes with no software support (unless you are willing to disturb in an endless manner your contact, a graduate student working on CONDOR at the MIT AI Lab.) and the revisions and updates to the package flourish too rapidly to make it yet a stable environment for software development. These observations are in no way intended to undermine the value and power of the CONDOR software package, but simply to explain its inapplicability to the real-time TRACK gait analysis project.

Those are the reasons why an exhaustive approach provided by commercially available and third-party packages such as CONDOR was not adopted. The Newman Lab. method does not pretend to compete with these packages, but has developed a simpler software development package which can be expanded as needed. Furthermore, implementing a software development package for a VMEbus based system is an invaluable learning experience if one intends to unfold the capabilities of the VMEbus environment.

Chapter 4

Real-Time TRACK

4.1 Processing

After updating the off-line processing algorithms of the TRACK software and finalizing a first C language implementation, the research focussed on the second phase of the project: dissecting the processing sections of TRACK5 and implementing a single computer, streamlined, real-time code. Then, using this first version of a real time software, the multi-processor real-time TRACK, RT-TRACK, was implemented. The software development scrupulously followed design parameters which included the following guidelines:

- First, the design was to be modular. In other words, the software would be a set of tools or modules that could be assembled in any way by the user. For example, the program should account for any combination of markers and array geometries as well as processing sequences. The software could thus be customized to specific experiments that could include, for example, only three-dimensional marker position calculations or Kistler force computations. However, there is a conflict between real-time application software and modular structured programming. Real-time applications are usually very specific and optimized to satisfy the best speed performance. These applications, such as control loops, are often written in a non-structured programming language, for example Assembly Language, so as to minimize the running time of every single operations. However, a

complex process involving a multitude of algorithms cannot be easily implemented in Assembly language. Moreover, if the system has to adapt to various configurations, it becomes imperative to divide the code into a set of multi-purpose subroutines (e.g.: the procedure performing Kistler platform force reductions).

- Then, the high quality of the kinematic data acquired and processed with TRACK has to be preserved in the real-time implementation. It is common to sacrifice part of the accuracy of calculations when implementing a real time loop: computational speed is then the most critical factor. For example, floating point calculations are replaced by faster integer approximations. Also, the numerous error checking and data corrections available in off-line processes are often skipped in real-time applications because they are time consuming. However, the enhanced accuracy provided in TRACK by the camera lens non-linearity corrections cannot be neglected in most experiments, even for the sake of real-time. Thus, implementing a real time camera lens correction scheme is essential. However, the real-time software should also allow the user to tailor the data processing so as to satisfy his/her personal requirements: the trade-off between best possible data quality and increased sampling and processing speeds should remain the decision of the ultimate user.

- The next important feature required of a RT-TRACK software package is the restriction of being totally compatible with already existing code. Thus, the real-time program will have to share data types, structures, storage and directories with the off-line TRACK5 software. This will allow further processing, with TRACK5, of data collected with RT-TRACK to evaluate, for example, velocities and accelerations using the spline smoother. This compatibility should also extend to the common usage. Handling the real-time software should be very similar to using TRACK5. Thus, users already familiar with TRACK5 will easily, and willingly, turn to RT-TRACK without the necessity of special training.

- The data storage also has to be transparent to the user. For example, a real-time session should be able to run indefinitely with the assurance that a subset of the data corresponding the the last few frames collected will be saved. This allows for the storage and retrieval of the sampled information for re-examination at a later date. This also implies that the overall system should be as user-friendly and transparent to the user as possible.

- Finally, in order to increase the processing speed of the software, a multi-processor based system approach using "pipelining" as described in Chapter 3 was adopted. However, the complexity of such a program implementation should not have any impact on the user. The multi-processor based system should behave as the single-processor one: the only apparent difference being the higher processing frequency provided by parallel computing.

The next step in implementing the real-time software was to evaluate the time involved in each of the processing sequences of TRACK5 for each sampled frame. Table 4.1 indicates that most of the time is spent doing the camera lens non-linearity corrections,

Table 4.1: TRACK5 Processing Profile

Process	% Execution Time[1]
Camera Corrections	26.0 %
BCS Computations	20.5 %
3-D Calculations	19.1 %
Inter-LED Error Check	7.0 %
Kistler Forceplate	2.3 %
Data Windowing	2.3 %
Miscellaneous (UNIX,...)	22.8 %

followed by the BCS computations and the 3-D calculations. These represent about $\frac{2}{3}$ of the overall TRACK processing. If one also notes that the "Miscellaneous" category includes processes irrelevant to a real time application (Operating System calls, Input/Output to the screen, etc...), the real-time software can easily be divided in three

[1] Time evaluations where performed with the sampling of 1000 frames of 4 standard arrays (16 Channels)

major blocks: camera corrections, 3-D calculations and BCS orientation computation. These three processes can also be complemented, as desired, with error checking routines (inter-LED, skew ray error) as well as the Kistler platform information. Using this design approach, the single processor real-time software, or streamlined software was implemented. If the real-time processing performance of this software compared favorably with the original versions of TRACK as shown in Table 4.2, it certainly was not satisfying for a full scale gait analysis system. Only without the camera corrections does the sampling rate reach 23.5 Hz. Thus, a faster implementation scheme was required: pipelining.

Table 4.2: Processing Performance

Systems	Performance
Original TRACK (Selspot I & PDP 11/40)	0.5 Hz
TRACK IV (Selspot II & PDP 11/60)	1.1 Hz
TRACK5 (Selspot II & Sun 3/160)	13.3 Hz
Streamlined RT-TRACK	15.4 Hz

The multi-processor software is based on the pipelining architecture. Using the software development package described in Chapter 3, different processes can run simultaneously on separate CPU's and perform calculations in parallel. Figure 4.1 illustrates the theory behind pipelining. On the left hand side (a), the flow chart shows a single processing approach where a frame is sampled and then goes successively through several processes one after the other until the execution loop end is reached and another frame can be sampled. On the other hand, the flow chart on the right (b) represents a simple pipelining scheme between two processors (e.g.: the Sun 3/160 and the Motorola VME133). In this case, Processor #1 samples the information and performs process 1 (e.g.: the camera corrections). It then passes the result to Processor #2 and immediately samples a new frame on which process 1 is executed while Processor #2 performs process 2 (e.g.: 3-D calculations). If both processes take about the same amount of time, this technique can theoretically double the processing speed. However, some handshaking is

required between the two CPU's to keep both processors in phase. This can be accomplished by using mailboxes at specific memory locations where each CPU indicates to the other its processing states.

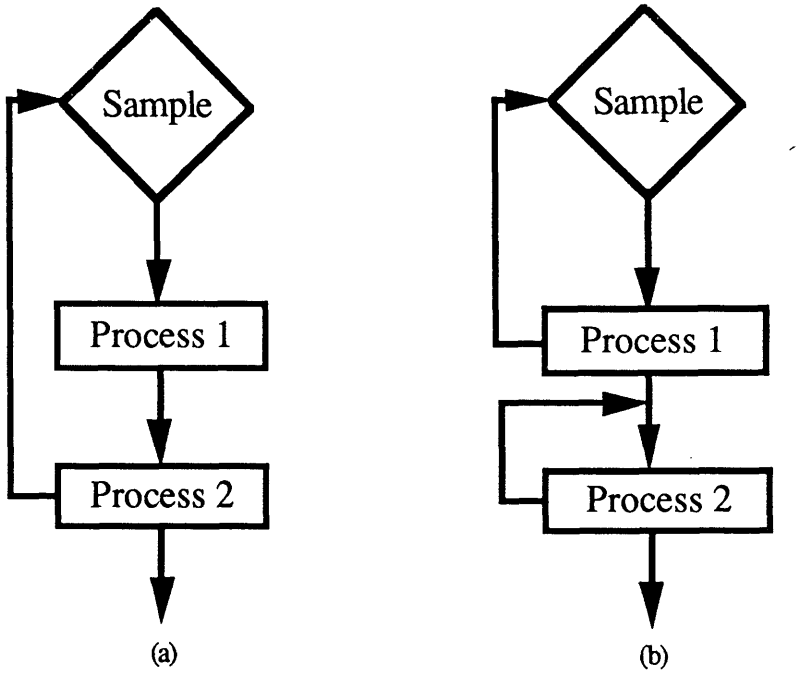

Figure 4.1: Pipelining Algorithms

The Pipelining scheme described above illustrates exactly how the RT-TRACK software has been implemented. The avid programmer and researcher will certainly find all the answers to his/her questions in Appendix B where the source code for RT-TRACK is itemized. At execution time, the main program of this software running of the Sun, loads the 3-D calculations algorithms on the VMEbus Motorola 133 board. It then starts the handshaking so that both CPU's are set for real-time sampling and processing. A ring buffer saving the last 10 seconds of data has been implemented in the main RAM of the Sun microsystem. When a real-time analysis routine is terminated by the user, the program will ask for a file name and save the data on the hard disk in a format identical to the one used in TRACK5. At this point, the software evaluates the sampling frequency achieved and updates the data file header. The graphic package provided with TRACK5 can then be

used to analyze the data and obtain hardcopies of plots on different output devices. However, the real-time system was designed so that the information can be readily used for gait analysis as the marker signals are sampled. Then, a subroutine filtering the data in real-time can be selected. Since no filter can satisfy the requirements and conditions of every experiment, the implementation of a filtering subroutine is left to the user of the real-time system. As an example, and for real-time gait analysis purposes, a real-time Finite Impulse Response filter (FIR) has been implemented [24]. This filter is only used to reduce most of the noise on the graphic display and introduces phase lag. This trade-off decision remains with the user who, by modifying this subroutine (or even implementing another filter) can design the filtering action according to personal specifications.

4.2 Display

The real-time gait analysis display software, RT-Gait listed in Appendix C is very similar to the off-line program 3D-Gait. The BCS data are provided in real time to the graphics display instead of coming from a previously stored data file. The major differences with the off-line display reside in additional features and a different layout, illustrated in Figure 4.2, producing real-time bar graphs for clinical joint angles and Kistler platform forces.

For those unfamiliar with the body segment motions and the anatomical reference movement nomenclature, we will attempt to present a brief description. For consistent identification, the movements which may be performed by body segments have been named and a definite terminology is emphasized. Although this terminology fails for certain complicated movement combinations as pointed out by David Kelley [25], it applies for walking motion with an acceptable consistency. For the human lower extremity, i.e.: the hip, the knee and the ankle joints, we are primarily concerned with five movement classes:

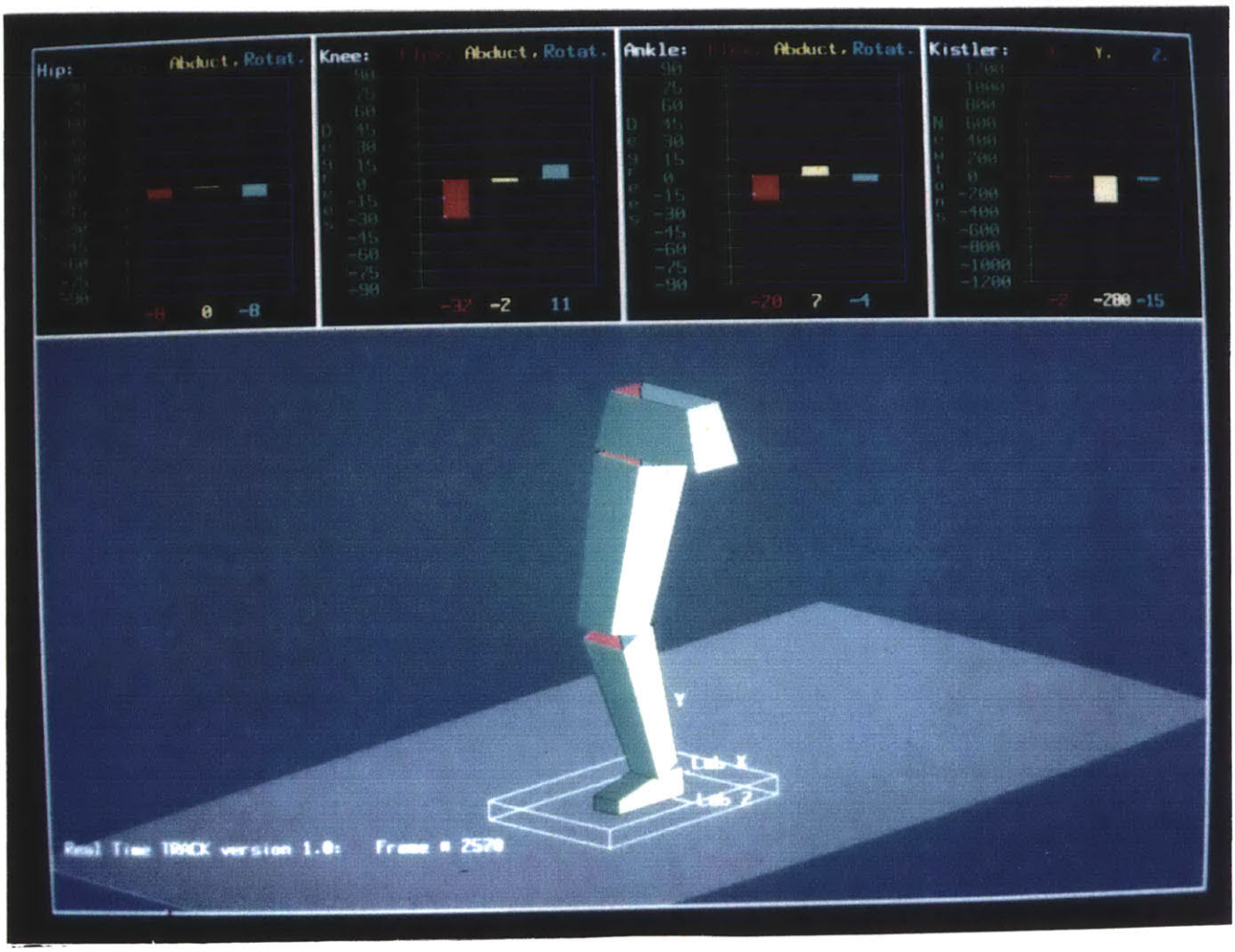

Figure 4.2: Real-Time Gait Analysis Display

- Flexion refers to the bending at a joint between two segments so that the angle magnitude between them decreases. The term flexion is commonly used for the hip and the knee while the term dorsiflexion usually refers to the ankle flexion.

- Extension refers to the stretching, or the return from the flexed position. Under extension, the angle formed between segments is increased. The term extension, while applying to the hip and the knee, is often replaced by plantar flexion at the ankle.

- Abduction refers to the movement of a body segment away from the centerline of the segment to which it is attached as well as away from the anatomical axis of the body symmetry. At the foot, abduction is usually referred to as eversion. However, as will be emphasized later, eversion is often accompanied by some rotation.

- Adduction refers to the movement of a body segment toward the body symmetry axis and toward the midline of the body segment to which it is attached. At the ankle, adduction is often referred to as inversion.

- Rotation quantifies the amount of rotation of a body segment along its longitudinal axis. Rotation can either be internal (if the result of the motion is to face toward the body midline) or external (if the result of the motion is to face away from the body midline). Rotation of the foot is rarely accomplished without some abduction/adduction (eversion/inversion). These combined joint motion are often referred to as pronation and supination respectively.

This terminology, although very adequate, was simplified in order to be implemented on the computer. At the hip, the knee and the ankle, flexion, abduction and external rotation were defined as positive (while extension, adduction and internal rotation were defined respectively as negative flexion, abduction and external rotation). The data is then presented in the form of bar graphs (Figure 4.2) in the real-time display. The graphs are composed of the three anatomical angle values for a given joint. These angles (in degrees) fluctuate as a function of time: the bars vary as the angle magnitude changes. A numerical display also lists the values for these angles so that they can be read directly by the operator. Providing a good display of the anatomical angles is crucial to the analysis of human gait. However, the complexity of the terminology as well as the wide-spread confusion over the definitions, requires a more detailed description of how the anatomical angles were calculated from RT-TRACK data and what conventions were used.

Grood and Suntay [26] offer an important insight in the understanding of joint kinematics with their description of a joint coordinate system for the clinical analysis of three dimensional motions. Although their investigation applied only to the knee, the convention they developed was adopted by this study to also compute the angles at the hip and the ankle. Figure 4.3 illustrates the geometric coordinate system used by Grood and

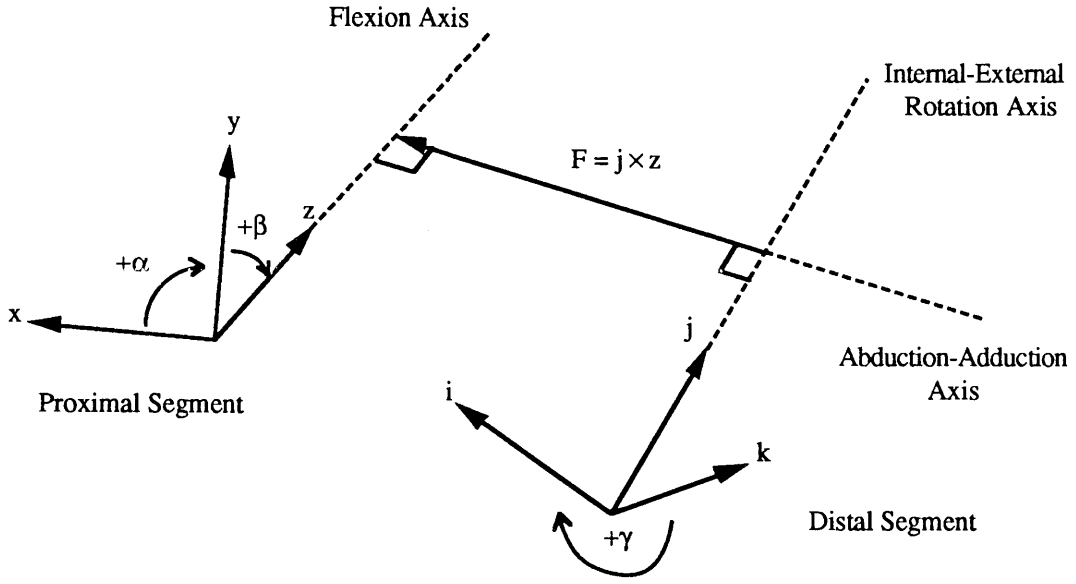

Figure 4.3: Segment Cartesian Coordinate Axes

Suntay. These Cartesian systems describe the fixed relationships of the bones. Since the local Cartesian transformations matrices computed by TRACK from the arrays attached to each body segments correspond to the positions of the respective bones, the same angle calculation convention can be applied. The method used can be described as follows. Note that i, j, k and x, y, z are vectors. First we want to define the common axis F, perpendicular to the segment fixed axes z and j:

$$F = j \times z \tag{4.1}$$

then the clinical joint angles flexion (+ α), abduction (+ β) and rotation (+ γ) can be calculated using:

$$\text{Flexion} = +\alpha = \arctan \frac{F \bullet y}{F \bullet x} \tag{4.2}$$

$$\text{Abduction} = +\beta = \arctan \frac{z \bullet j}{z \bullet k} \tag{4.3}$$

$$\text{Rotation} = +\gamma = \arctan \frac{-F \bullet k}{F \bullet i} \tag{4.4}$$

This method provides not only consistent results that are independent of the order in which the flexions, rotations and abductions are performed, but also a clear mathematical

description of the clinical terminology for joint rotations. The display of the anatomical joint angles under the format presented above has proven to be a significant improvement over numerical analyses.

The introduction into a clinical setting of a gait analysis display system, like any other computer application, will automatically lead to interactions between the operator and the hardware. Such an operator is usually more familiar with visual analyses than numerical interpretations. Furthermore, the introduction of a display as a clinical tool would certainly mean a reorganization of a habitual work situation. Reluctance to change of work patterns is often exhibited by medical personnel. This resistance is very often driven by avoidance of the hassle of understanding and using a new machine combined with memories of painful past experience. When a computer is introduced into such an environment, it does not communicate nor "think" like the humans who will be using it. Computer languages require technical skills common to most engineers but which cannot be expected from all physicians. Finally, computers are "stupid" and perform only as well as they have been programmed, which thus explains why they function within a very limited and rigid environment and why they are unable to adapt their behavior to the requirements of their potential users.

Thus, the man-machine interface has to be carefully designed and never underestimated in order to achieve an effective system. Knowing in advance the desires of future users will facilitate the implementation of a system best suited for them. In the case of gait analysis display, questions include what should be presented, what the user will want, how the user will prefer to communicate with the system, and what will the user find frustrating?

These human-computer interface requirements have been investigated by Sanblad, Lind and Schneider [27] as well as Foley and Van Dam [28]. In the light of their analysis, the man-machine interface of the real-time display (RT-Gait) was developed so as to provide flexibility and effectiveness. Recent developments on new computer workstations

have shown that the use of the keyboard should be progressively restricted while input through other devices, especially the mouse, should be favored. This approach appears consistent with the needs and style of clinicians. The mouse, being an artificial extension of the hand, is a natural way to interact with a computer. RT-Gait uses this approach by conserving the interface developed for 3D-Gait [17]. In addition to the windowing facilities, RT-Gait allows the user to perform a certain number of functions while the display is running. First, at any time the viewer can control manually the advance of each display frame. This permits a manual slow, fast and reversed motion, including a frame-freeze position that lets the user stop the display at a time of particular interest in the gait cycle to analyze and compare the quantitative data with the qualitative information. The operator can also control with the mouse the viewing direction of the walking legs with 360° of freedom. In addition, the user can customize the screen to his/her personal taste by controlling what information is to be displayed concurrently on the terminal. The user is also offered the opportunity to display the LED arrays as mounted on the human lower extremity to confirm the data acquisition process. These can be replaced by local coordinate axes to help visualize the spatial orientations defined by the TRACK rotation matrices. Finally, several options provide for printing color and black/white hardcopies of the screen. This should not be overlooked, since for medical archiving and retrieval of patient records a printout still remains the best possible information support when a computer cannot be easily accessed.

4.3 Communication

For clarity of presentation the real-time data acquisition system (RT-TRACK) and the real-time gait analysis display (RT-Gait) have been kept separate. However, real-time gait analysis requires both processes to run in parallel as well as to exchange data. Thus, one is faced with the problem of interconnecting two computers, based on different

architectures. One computer, the Sun, samples the LED signals and processes the six degree of freedom information while the other, the IRIS, uses these results as an input to a graphic display. It is apparent that the real-time computer should run freely, providing information to the graphical system only upon request; graphic operations are time consuming and should not interfere with the sampling process. If conceptually this seems rather straightforward, it is much more complicated in implementation than one might imagine. Mostly, two methods are available to let one computer converse with another: serial or networked communications.

- Serial communication is a very popular technique to transmit information over rather short distances from a computer to peripherals, terminals, or other computers. The technique involves a serial line (or phone line) connected to an RS-232 communication port (or a modem). Transfer rates are slow and can go as high as 38.4 kilobits/sec (and even higher in certain applications). Error detection is usually necessary because of the noise level encountered in the serial cables used. This slows down transmissions which then often only reach half of the published baud rates.

- Network communication is definitely a faster method. Relying on coaxial cables instead of electrical wire, and wave transmission instead of signal pulses, networking can achieve speeds in excess of 10 Megabits/sec. The Networking Standard is Ethernet [29]. Ethernet has grown from a collaboration between three American corporations - Digital Equipment, Intel and Xerox (DIX). In 1980, the consortium published a booklet outlining the specifications for a bus topology network employing the CSMA/CD technique (Carrier Sense Multiple Access with Collision Detection). The document was based on a successful experimental system developed by Xerox at their Palo Alto Research Center. The DIX specifications were designed to encourage a standardized approach for different manufacturers wishing to design equipment capable of utilizing Ethernet. The system is based on the Cambridge Ring network theory: a token ring communication. Thus, transfers can reach up to 10 Megabits/sec but the polling frequency of about 10 Hz remains

the same across each node. For example, ten transfers of 1 byte each take the same amount of time as ten transfers of 1000 bytes each. The network scheme makes extensive usage of the packet transmission approach in order to be as efficient as possible with up to 1024 Ethernet stations over a maximum length of 2.8 km without a repeater.

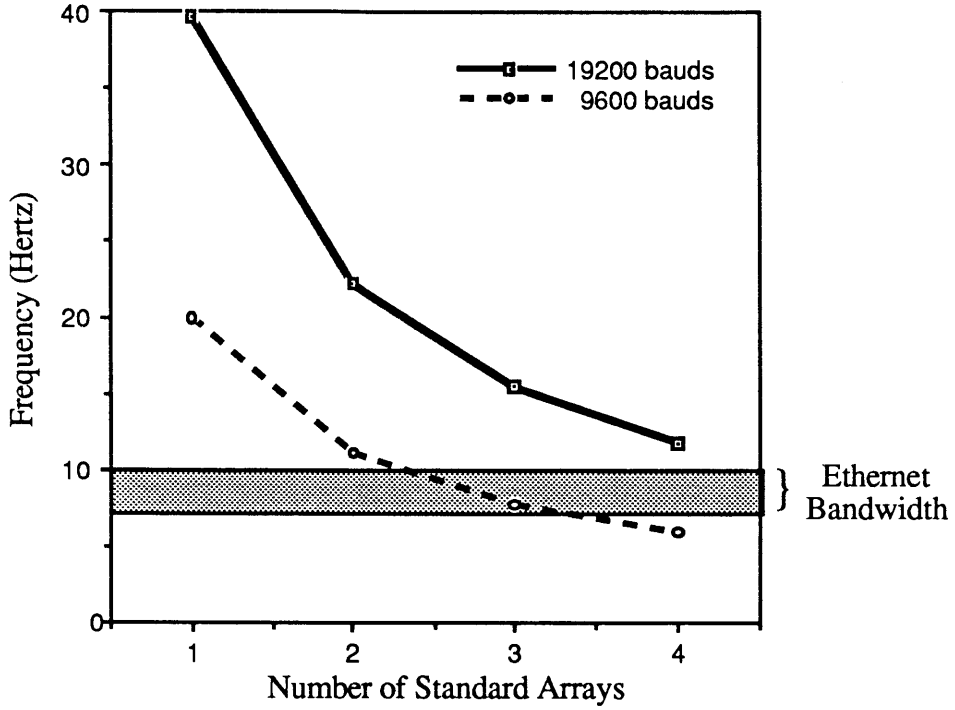

Figure 4.4: Communication Performance

The size of the BCS data generated by RT-TRACK can vary from 256 bits to 2048 bits per frame depending on the number of arrays used. Figure 4.4 shows the transfer frequencies achievable using serial communication (at 19.2 kbauds and 9600 bauds due to limitations enforced by the Silicon Graphics hardware) as well as Ethernet communication. As expected, Ethernet communication is not dependent on the amount of data transfered (since it is much less than the upper limit of 10 Megabits/sec). Note that with four standard arrays (16 channels), the performance of Ethernet falls within the range of the serial communications. Moreover, Ethernet includes error detection and is much less noisy than serial transfers. From Figure 4.4, it is clear that for gait analysis purposes Ethernet

communication is certainly the most appropriate (even more so if bilateral information is collected: 8 arrays, 32 channels, 4096 bits per frame). These reasons influenced the design of the communication software between the IRIS workstation and the Sun microsystem. Using the laboratory network, each computer opens a transmission socket on the other in order to communicate in both directions. This communication scheme provides the control of the overall system from only one computer, primarily the IRIS graphics terminal. Thus, even though the end user is controlling a parallel processing system composed of three computers and data acquisition interfaces, he/she remains totally unaware of the complexity of the overall system and interacts with only one computer.

Chapter 5

Results

5.1 Speed Performance

The speed performance of a real-time software is certainly the most significant implementation result. Figure 5.1 presents the speed performance obtained from RT-TRACK for different configurations. The solid lines represent the single processor software results while the dashed ones correspond to the performance of the pipelining multi-processor software. Each test was averaged over 1000 sampled frames using the standard LED arrays shown in Figure 1.1. These results show that the pipelining algorithm significantly increased the processing frequency over the single processor implementation. Also notice that camera corrections can be performed by the multi-processor with only a slight time penalty (handshaking overhead) when compared to the single CPU executing only the three-dimensional and BCS computations. For a single standard array (4 channels) , the system can reach 100 Hz without the camera correction calculations. This value compares very favorably with other reports in the literature. Table 5.1 indicates a normalized performance of 200 Hz for RT-TRACK. This adjusts the performance of the system developed here for comparison with the experiments of Conati [6], Tetewsky [8] and Ottenheimer [9], all of whose used a three marker array, no camera corrections, no error checking (Skew ray, Inter-LED), and partial rotation matrix

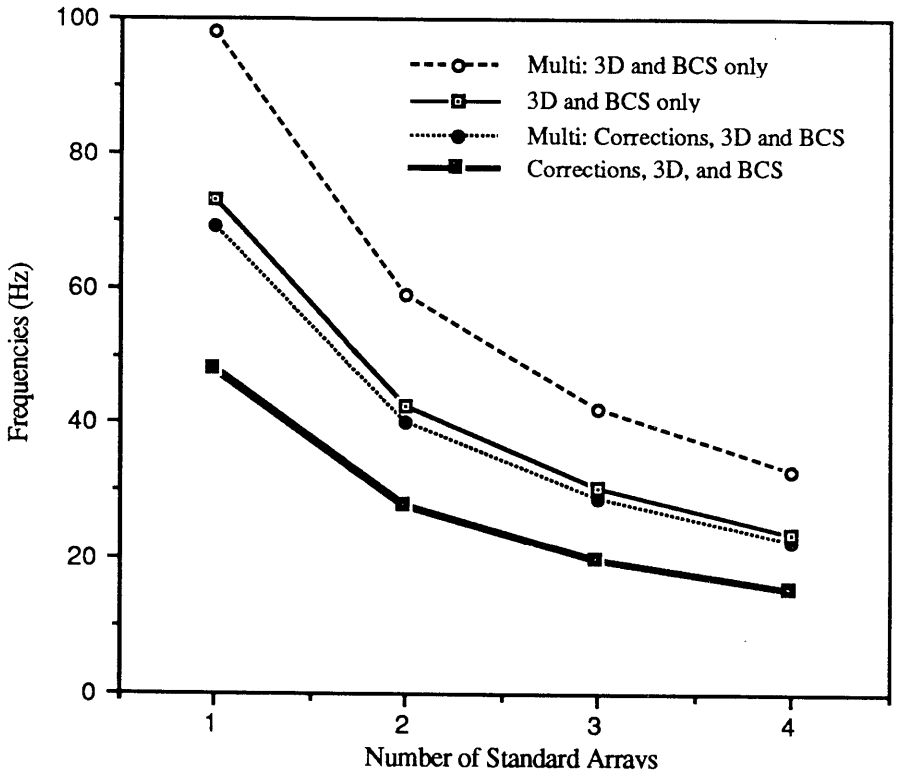

Figure 5.1: Real-Time TRACK Performance Results

calculations. In addition, they used the Selspot I system which has but a quarter of the resolution offered by the Selspot II system. They also made extensive usage of integer arithmetic (faster than floating point but less accurate) to speed up processing time since the errors introduced by the integer approximations fell within the resolution and accuracy of the Selspot I system. Ottenheimer used a special PDP 11/60 accelerator processing board for his numerical algorithms which were directly derived from Tetewsky's study. This explains the order of magnitude improvement Ottenheimer achieved over the the previous

Table 5.1: Literature Real-Time Performance Comparison

Software	Maximum Performance
RT-TRACK Normalized Performance	200.0 Hz
Conati (1977) [6]	8.7 Hz
Tetewsky (1978) [8]	11.2 Hz
Ottenheimer (1982) [9]	120.2 Hz

attempts. These earlier real-time kinematic data acquisition systems were, for the most part, developed for specific experiments and projects (auditory spatial displays to study the mobility of blind persons). The programs, written mostly in Assembly language, offer little if any flexibility. RT-TRACK, written in C, does not compromise the precision of the data with integer operations, and can easily be customized to suit an application. In addition, the system also incorporates real-time Kistler force platform sampling and processing. With Kistler alone, a real-time processing frequency over 1.1 kHz can be reached. Adding Kistler force calculations to the Selspot computations is hardly noticeable in the performance results (usually 1 Hz difference). A standard unilateral gait analysis

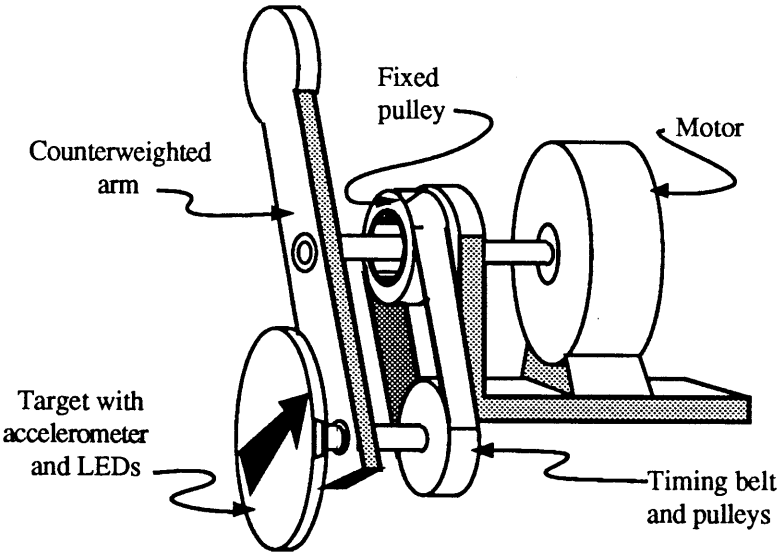

Figure 5.2: Mechanical Sine Wave Generator [30]

(with foot/floor interface forces) can thus be performed in real-time at 21.5 Hz (and 32.5 Hz without the camera lens non-linearity corrections). This is satisfactory for most gait analyses, but comes short of the minimum 40 Hz sampling frequency required to capture all the components of gait which are below a 20 Hz frequency [3]. However, the modular multi-processor pipelining programming theory can be recursively applied with the addition of other VMEbus board computers such as the Motorola VME133 ($\approx$ $1,500), and thus yield even better processing performance inexpensively. For example, only one more

VME133 dedicated to the computation of the BCS orientations would boost real-time gait analysis to about 43 Hz (≈ 65 Hz without the camera corrections).

5.2 Mechanical Experiment

To illustrate that RT-TRACK can be used for real-time experiments other than gait analysis, the following mechanically generated experiment tested the software. The device shown in Figure 5.2 was designed to evaluate the frequency response of optical positional measurement systems using either active or passive markers and has been utilized to assess the performance of the MIT TRACK/Selspot system [30]. Thus, it seemed logical to check the real-time software with this mechanical sinusoidal wave generator. Figure 5.3 illustrates the motion of the array dictated by the mechanics of the device. The 8 markers constituting the array move along a circle in the plane of the paper, with theoretically no motion along the perpendicular to the plane of the paper as well as no rotation about that axis. The real-time experiment was performed and a sampling frequency of 40 Hz achieved. The parametric motion recorded is presented in Figure 5.4 along with the vertical motion as a function of time (Figure 5.5), and the vertical velocity also as a function of time (Figure 5.6)[1].

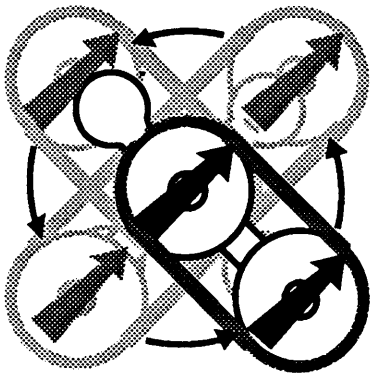

Figure 5.3: Schematic of Motion [30]

[1] The velocity calculations were performed off-line with TRACK5 using the data recorded by RT-TRACK.

Geometrical verification demonstrates that the recorded circular path fits the actual motion to within 1.2 mm. Also, the velocities evaluated by the software were verified analytically (simple sinusoidal function fit). Thus, RT-TRACK can be used for all sorts of kinematic six degree of freedom analyses with excellent accuracy. However, mechanical kinematics that approach the maximum recordable frequency of the real-time system are much more satisfactorily studied using the off-line implementation TRACK5 (up to 5000 Hz).

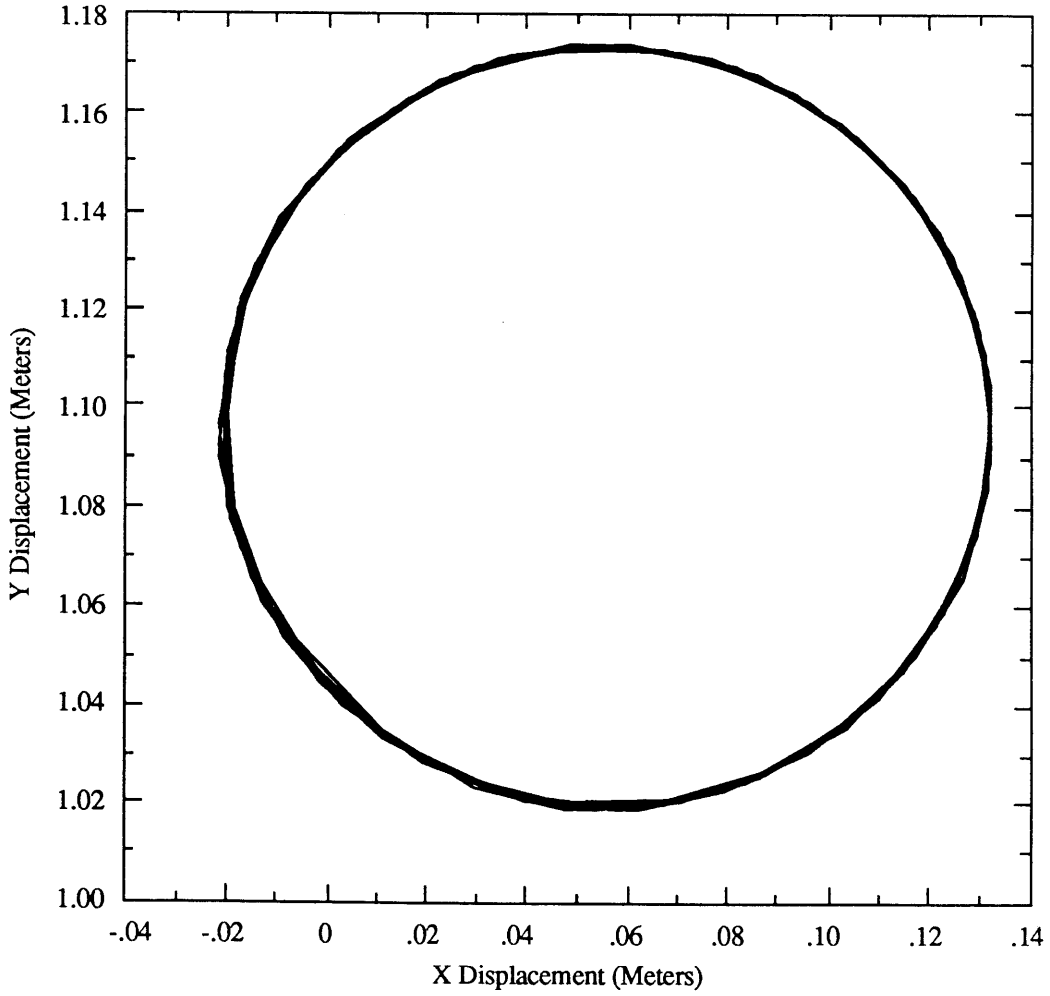

Figure 5.4: Horizontal and Vertical Parametric Motion

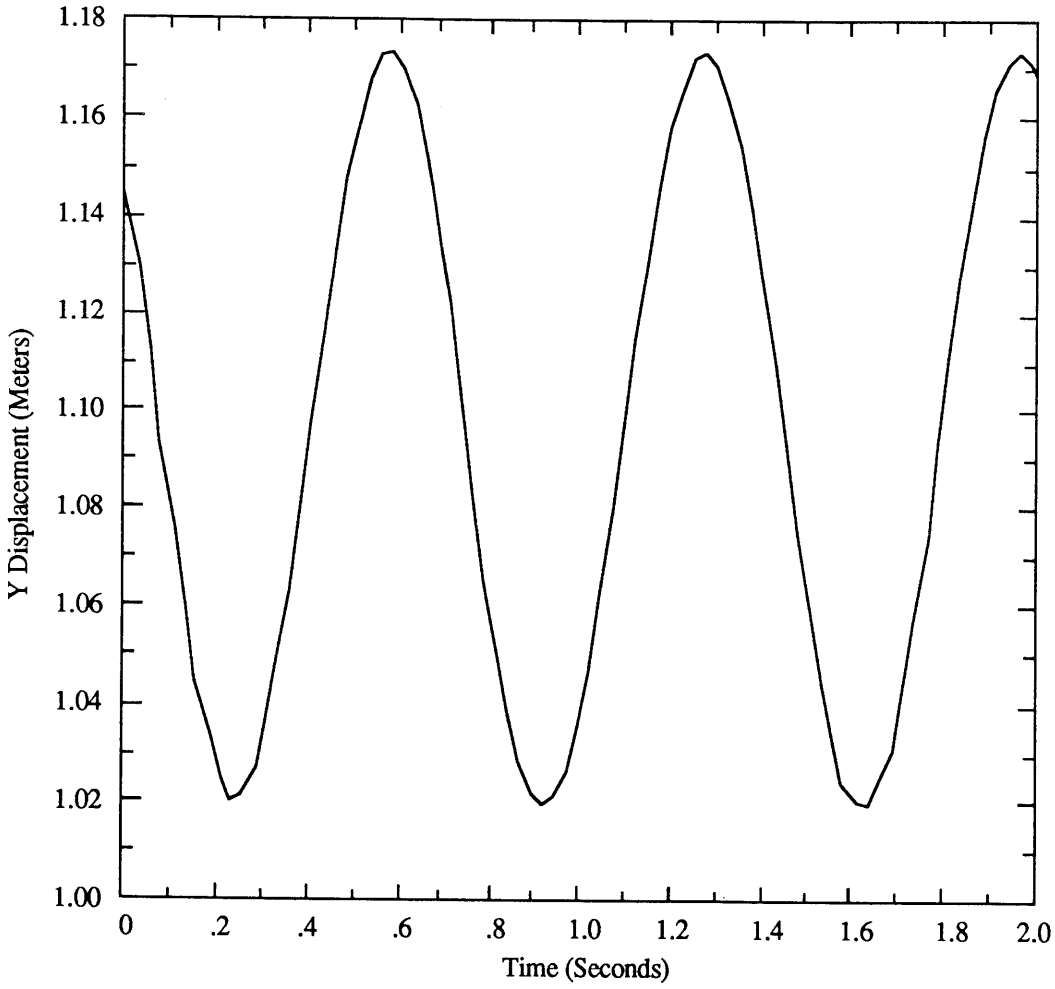

Figure 5.5: Vertical Position

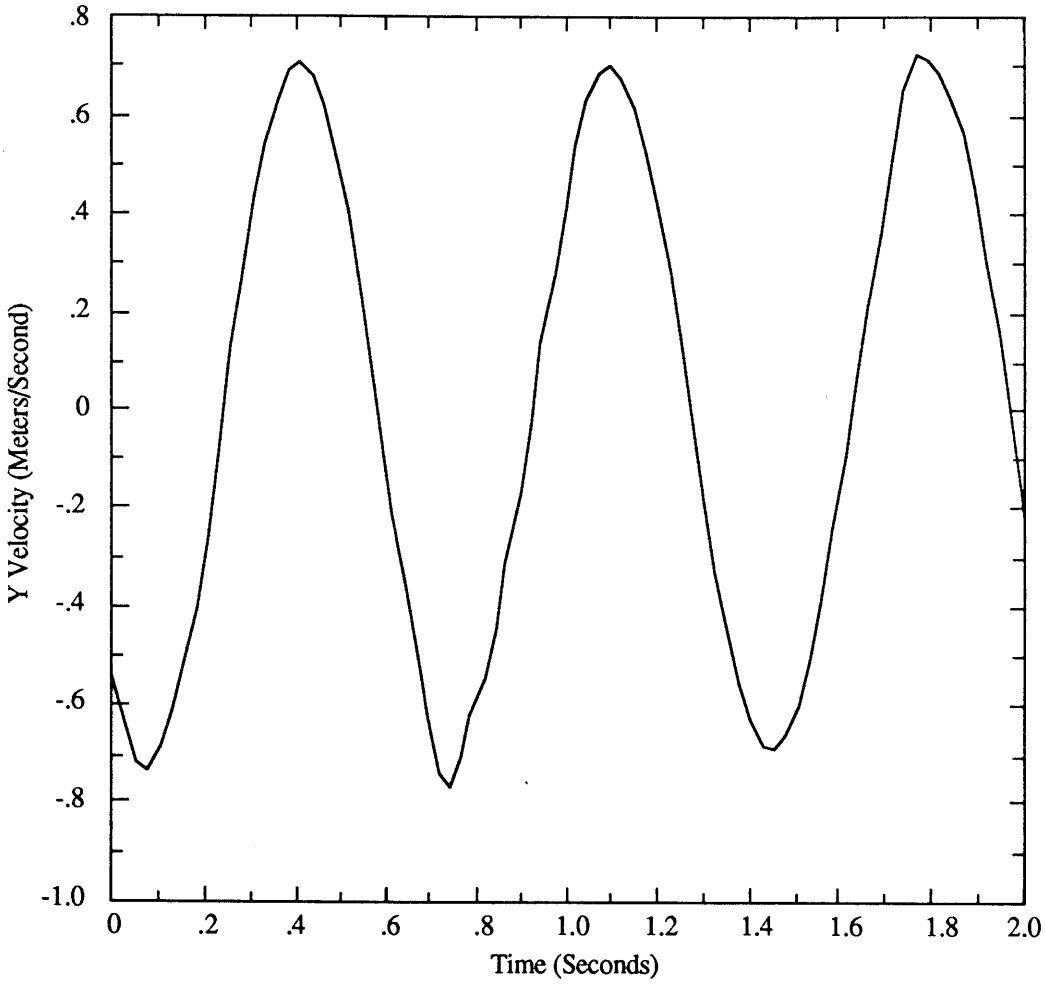

Figure 5.6: Vertical Velocity

5.3 Real-Time Gait Analysis

The real-time display presented in figure 4.2 is the result of a gait analysis experiment. A subject was fitted with the arrays as illustrated in figure 5.7. The real-time

Figure 5.7: Instrumented Subject

software RT-TRACK, running in parallel with the display software RT-Gait (Figure 5.8), was then started. After a few minutes of interactive experimenting and studying ranges of motion and frequency responses in real-time, a simple gait experiment consisting of the subject walking across the camera viewing field and stepping on the forceplate was conducted. The resulting information was acquired and analyzed in real-time. The Kistler

platform force in the vertical direction is presented in Figure 5.9. Finally, the clinical joint angles recorded at the knee are provided as example results in Figures 5.10 to 5.12.

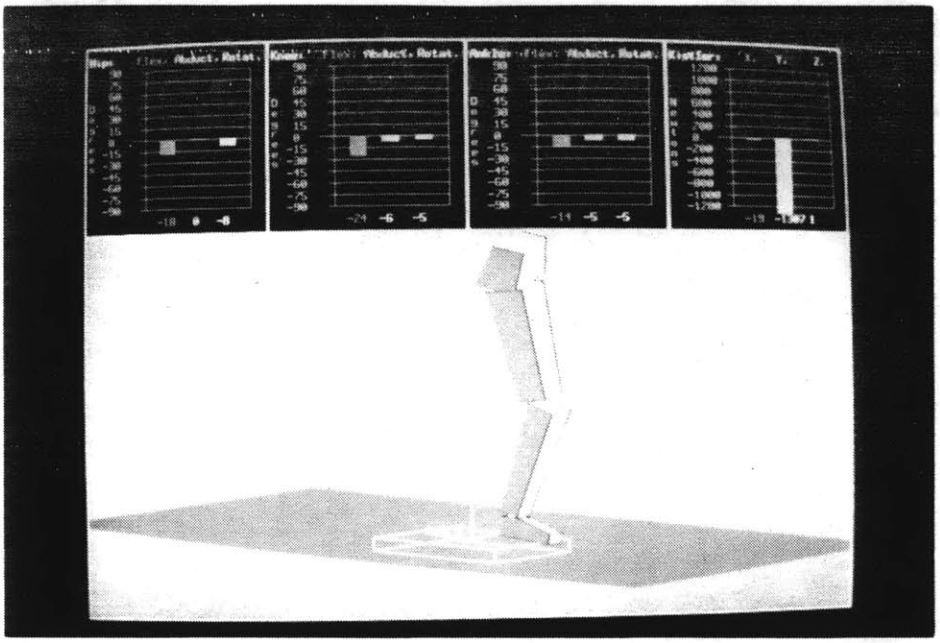

Figure 5.8: Real-time Gait Analysis Experiment

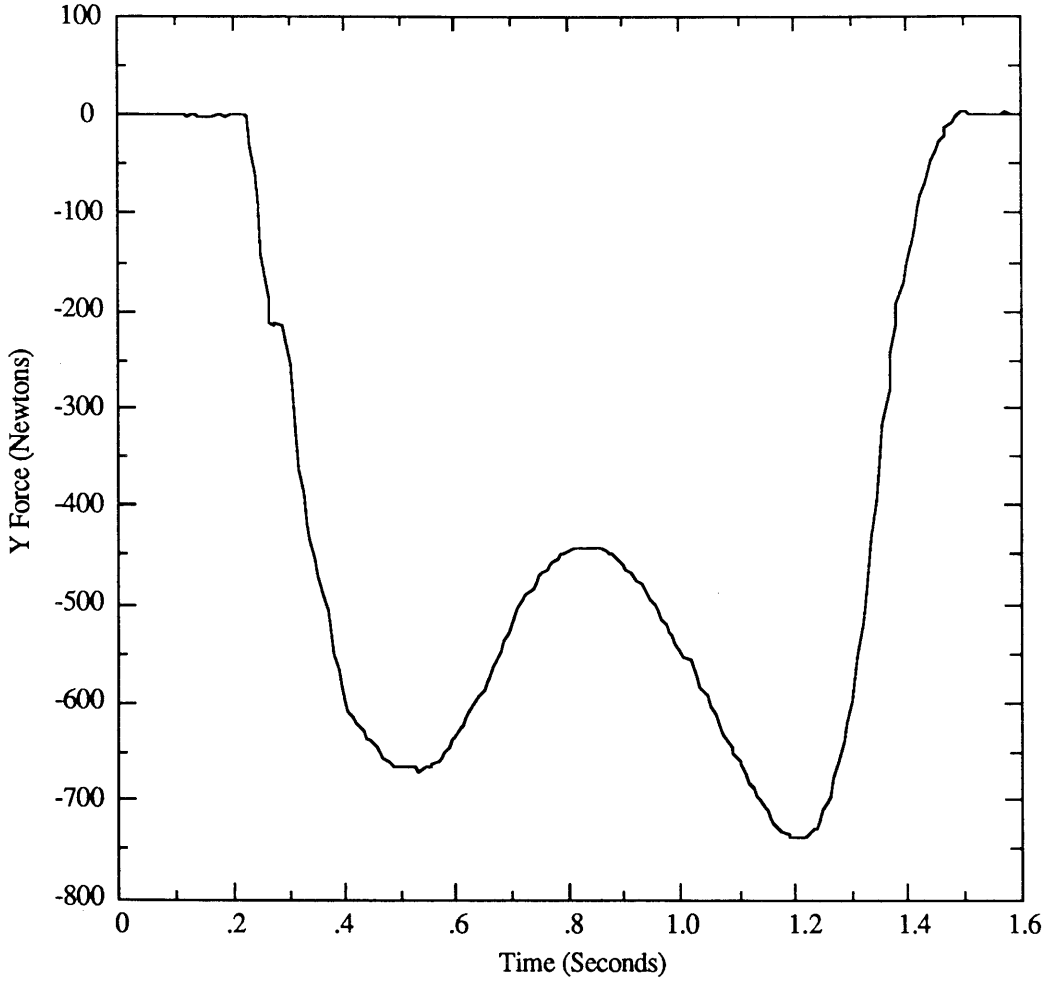

Figure 5.9: Vertical Kistler Force Measurement

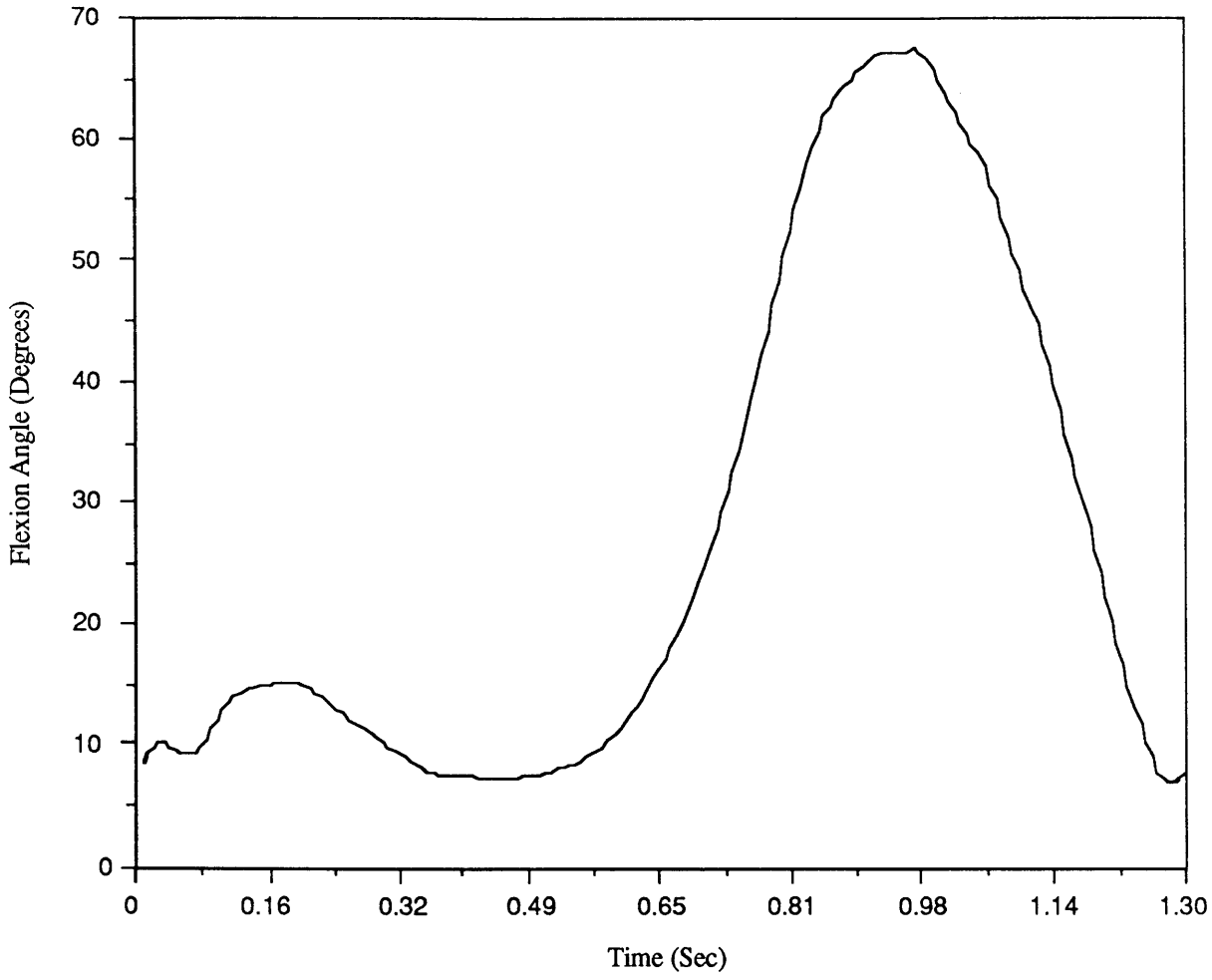

Figure 5.10: Knee Flexion

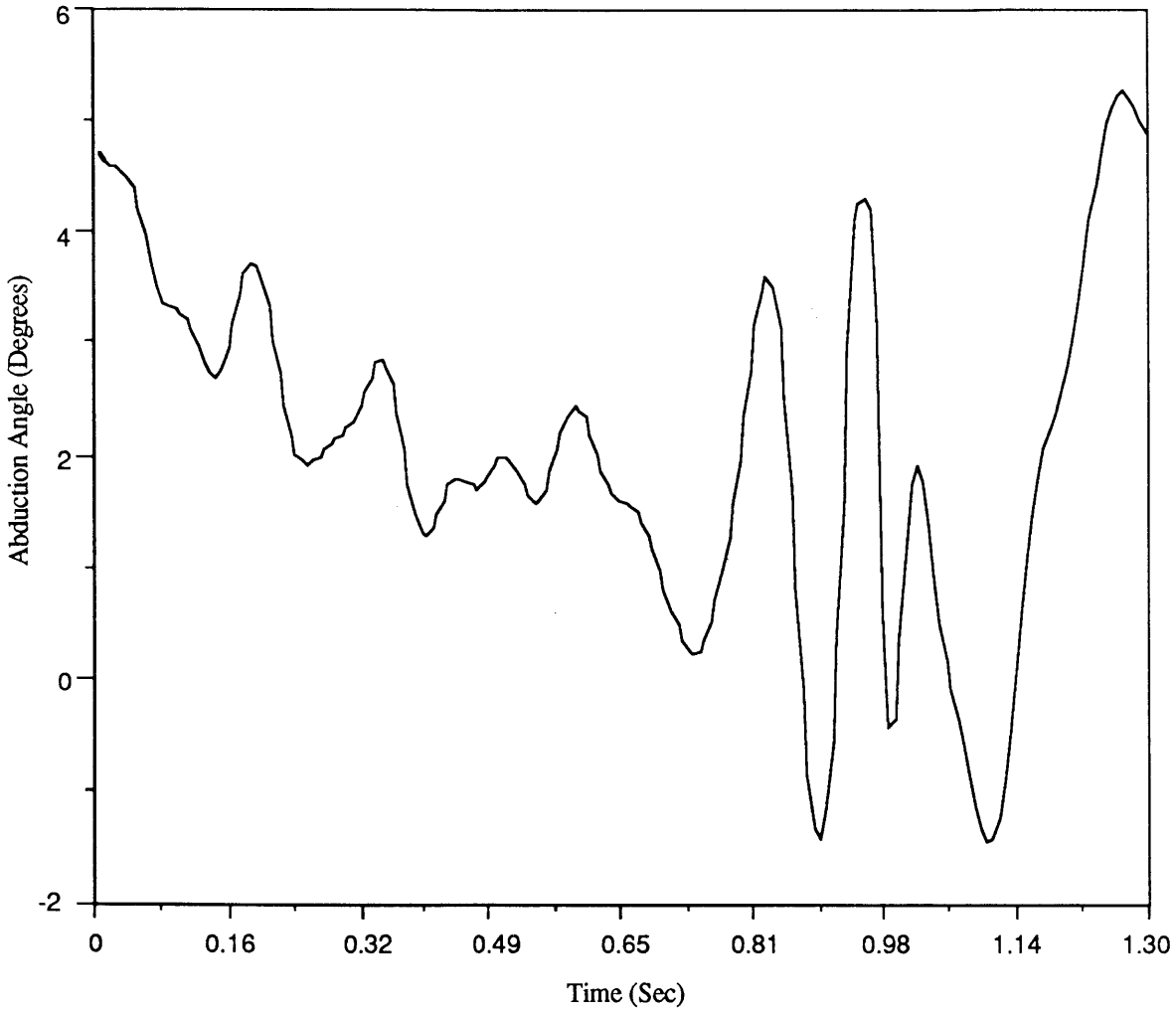

Figure 5.11: Knee Abduction

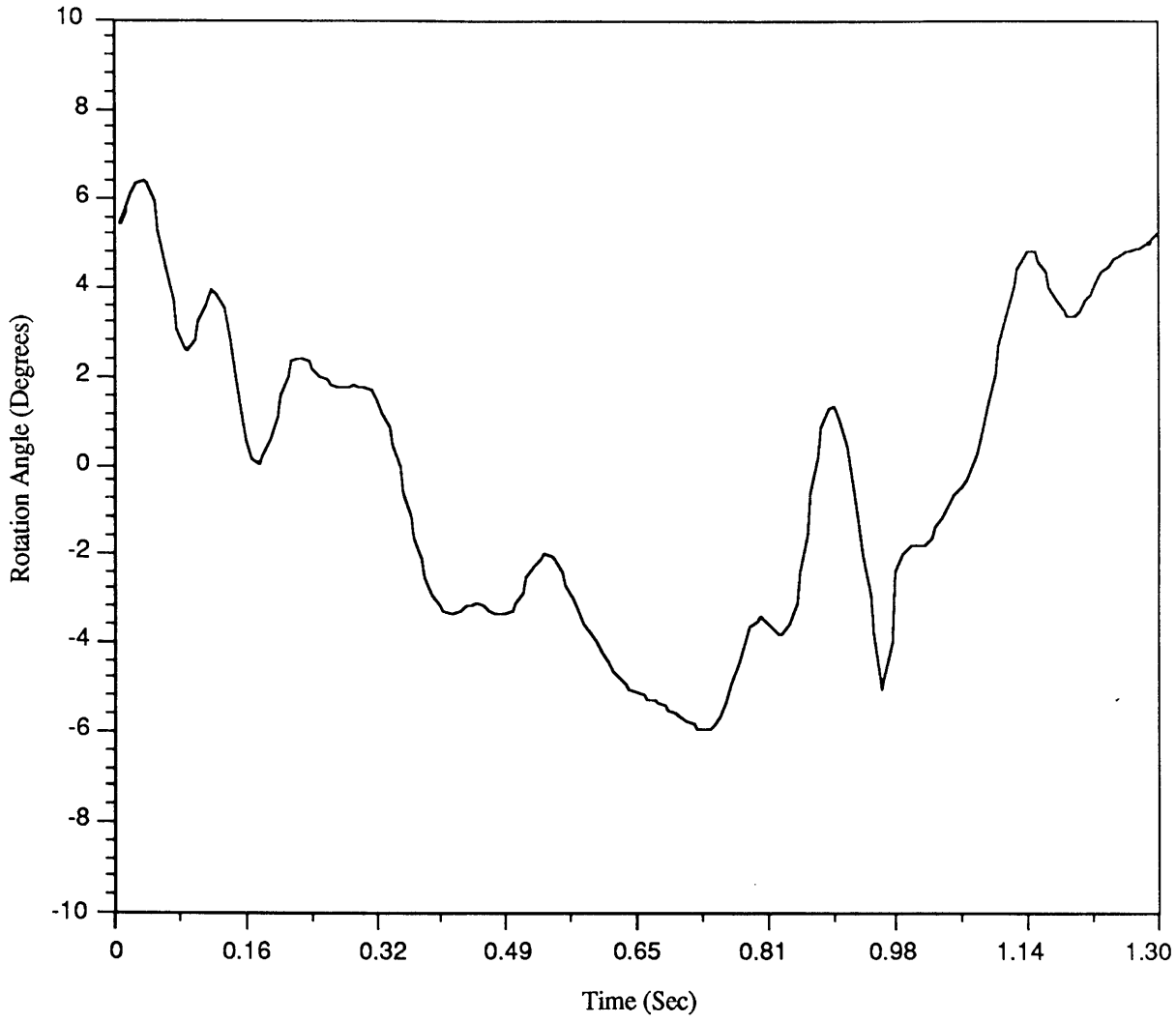

Figure 5.12: Knee Rotation

Chapter 6

Conclusion and Recommendations

6.1 Summary

A real-time interactive system for the study of complex, three dimensional human locomotion has been developed in the Newman Laboratory for Biomechanics and Human Rehabilitation at MIT. Human motion, acquired by the Selspot and Kistler systems, and processed in real-time with the multi processor software RT-TRACK, can then be transfered to a computer graphics terminal. There, the software RT-Gait produces a three dimensional model of the lower extremity together with concurrently updated data bar graphs. The system models each leg segment as a geometric solid prism thus providing realistic visualization of clinical joint angles, especially the long axis rotations, and enhanced understanding of gait cycle nuances. The window management system provides simultaneous presentation of gait relevant quantitative information along with the three-dimensional perspective view of the model. The man-machine interface has been designed to allow the operator, ultimately a clinician unfamiliar with programming languages, to customize the screen with display of the desired kind and amount of information. This interface makes the complexity of the system entirely transparent to the user who remains unaware of the multi-CPU implementation which processes and displays the information in real-time. By using this system, it is expected that clinicians can interact with the

instrumented patient and the graphic display while using the software options which have been implemented as powerful visual aids.

Several data sets have been collected in the Newman Lab at MIT, and have been analyzed using the software package to evaluate its performance as a gait analysis tool. The results obtained for walking were internally consistent and corresponded to the expected patterns. However, to validate even further the approach used, the results were compared to and confirmed by the numerical analysis performed by Krebs [31] using a different gait analysis system. It also was concluded that the real-time feedback provided by this system is invaluable, not only for the clinician-computer-subject interaction, but also to simply verify the validity of data while the patient is still wearing the infra-red markers. The system will instantly point out any problems such as malfunctions with the hardware and erroneous assumptions. In addition the compatibility of RT-TRACK with the off-line processing software TRACK5 renders the real-time kinematic analysis extremely easy to master. Although walking was emphasized as the major application justifying the development of this system, the software can accommodate all ranges of human motion recordable within the actual viewing volume of the Selspot cameras. It has been tested at MIT against users' imagination. An example of the effectiveness of the system was that while watching the display it became apparent that users requested additional specific motions from the subject being analyzed: this is the result, and in a way the goal, of real-time gait analysis. Research now under way in the Laboratory [4] will significantly increase the viewing field of the system, making TRACK then definitely the most challenging optical kinematic data acquisition thus far. It will then become even more invaluable for research concerning mobility impairment, limb replacement design, sport biomechanics and computer aided surgery simulation and evaluation.

It should also be stressed that gait analysis is not the only application of this real-time system. Any kinematics study, human or mechanical can be performed. The system can be used to acquire what would be sensory information for feedback control of input

action as in the human's control of mechanical devices such as robots. A current research project in the Newman Lab is studying the impedance and stiffness of the human arm. An air-jet powered device called the "Wrist-Rocket" is used to disturb the limb during purposeful movement. RT-TRACK is being considered as the source of control information.

Finally, the VMEbus CPU software development package implemented for this study has proven to be a useful tool for other application software. This programming method provides powerful numerical computation at a reasonable cost and has become the solution to most real-time algorithms developed in the Lab.

6.2 Recommendations

Although to this date, the software appears free of implementation "bugs", these may manifest themselves with the further development and use of this analysis system. System crashes due to bugs that still remain unknown today have happened! This further emphasizes that *gracious* error trapping will have to be implemented since it is essential to all software package design. A good program should be as user friendly as possible and provide warnings in the case of preventable obstacles. The system presented in this study includes a basic error detection program that handles most non-fatal errors. But time constraints on this thesis work defer further work and software refinements in order to make the system completely "dumb-proof".

Certainly, a prime candidate for future research is the design of a complete system on a single computer capable of handling the numerical computations and the graphical animations. This would remove the bottleneck corresponding to the rather slow network communication used between the IRIS and the Sun workstations. Such hardware is now available in midrange prices from several manufacturers. Based on past experience, this technology should become easily affordable within the next few years.

Further development that would dynamically display the human upper trunk, head and upper extremities, as well as the lower extremities of the body, would certainly enhance the gait analysis process. These improvements can be expected soon since by simply adding more markers the overall body motion could be recorded by the kinematic acquisition system. However, more computational power will be required to manipulate the added information.

Finally, this study is part of the development of a computerized surgical simulation system which integrates kinematic information with computer tomography and musculoskeletal dynamics. The goals of the system are surgical outcome prediction using pre-operative surgical simulations performed on a computer display of the patient's specific anatomy and changes in gait dictated by the alterations done. This system is several years from completion, but preliminary results indicate that data structure and modelling refinements, medical imaging improvements, kinematic information, myoelectric data, animation and processing speed will remain crucial elements of this preoperative surgical tool.

Appendices

Source Code

Newman Laboratory for Biomechanics and Human Rehabilitation

Massachusetts Institute of Technology

Department of Mechanical Engineering

Room 3-137, 77 Massachusetts Avenue

Cambridge, MA 02139.

(617) 253-8112.

Appendix A

Source Code:
MCC Software Developement Package

```
# Makefile for vme133
#
# Patrick J. Lord       December 10, 1987.

.SUFFIXES: .o  .c .s
INCLUDE= -I/usr2/pat/condor/vme133

FILES = abort.o exit.o getstring.o print_double.o print_float.o \
print_int.o printf.o putstring.o read_double.o read_float.o \
print_long.o read_long.o read_int.o scan_double.o scan_long.o scanf.o \
write_double.o write_long.o

OBJECTS = mcc.o startup.o fabs.o abs.o math.o crt0-vme133.o ${FILES}

CFLAGS= -O -f68881 -fsingle

# Generate include libraries
archive: all mcc startup
    ar cr libc-vme133.a ${FILES}
    ranlib libc-vme133.a
    ar cr libm.a math.o fabs.o abs.o
    ranlib libm.a

mcc: mcc.o
    cc mcc.o ${CFLAGS} -o mcc

startup: startup.o
    cc startup.o ${CFLAGS} -o startup

all: ${OBJECTS}

# Make .o files from the .c files
.c.o: $*.c
    @echo "> Generating" $@ because of $?
    cc -c ${CFLAGS} $*.c

# Make .o files from .s files
.s.o: $*.s
    @echo "> Generating" $@ because of $?
    as $*.s -o $*.o
|-------------------------------------------------------------
|-                                                            -
|-   abort.s                    12/10/87       Patrick J. Lord   -
|-                                                            -
|-   Motorola library 68020 vme133. Abort program and return to -
|-   debug monitor.                                           -
|-------------------------------------------------------------

| Declare functions for external use

    .data
    .text
    .globl    _abort

_abort:
    trap    #15                 | return to monitor
    .word    0x63
    rts
/*
-------------------------------------------------------------
-                                                            -
-   abs.c                      12/10/87       Patrick J. Lord   -
-                                                            -
-   This functions returns the absolute value of an integer    -
-                                                            -
-------------------------------------------------------------
*/

abs(i)
register int i;
{
    if(i < 0) return(-i);
    else return(i);
}
|-------------------------------------------------------------
|-                                                            -
|-   crt0-vme133.s              12/10/87       Patrick J. Lord   -
|-                                                            -
|-   Motorola vme133 68020/68881 C start routine.               -
|-                                                            -
|-------------------------------------------------------------

| Declare external functions and variables

    .globl  cstart          | this routine
    .globl  _main           | user's main

cstart:
    movw    #0x2000,sr      | level 0, supervisor mode
    movl    #0x10000,sp     | initialize stack pointer

    jsr     _main           | call users program

    .comm   f68881_used,4   | acknowledge existence of the 68881

    trap    #15             | return to firmware monitor
    .word   0x63            | Note that this should never get executed

/*
-------------------------------------------------------------
-                                                            -
-   exit.c                     12/10/87       Patrick J. Lord   -
-                                                            -
-   exit() is the function to be included a the end of a C     -
-   program.                                                   -
-------------------------------------------------------------
*/
#include    "stdio.h"

void exit()
    {
    abort();
    }
/*
-------------------------------------------------------------
-                                                            -
-   fabs.c                     12/10/87       Patrick J. Lord   -
-                                                            -
-   This functions returns the absolute value of a float       -
-                                                            -
-------------------------------------------------------------
*/

float fabs(num)
register float num;
{
    if(num < 0) return(-num);
    else return(num);
}
|-------------------------------------------------------------
|-                                                            -
|-   getstring.s                12/10/87       Patrick J. Lord   -
|-                                                            -
|-   Motorola library 68020 vme133. Get a string from the       -
|-   terminal.                                                  -
|-------------------------------------------------------------

| Declare functions for external use

    .data
```

```
    .text
    .globl  _getstring

_getstring:
    link    a0,#0           | Link, no local variables
    moveml  a5/a6,sp@-      | save registers

| Get the input by a firmware trap #15 function call.

    subql   #4,sp           | Allocate space for result.
    movl    a0@(8),a5       | string pointer to a5
    pea     a5@             | push pointer to destination
    trap    #15             | input from console
    .word   0x2
    movl    sp@+,a6         | Retrieve address of last character +1.
    movb    #0,a6@          | Terminate the string with a Null Pointer

| Clean up and return

    moveml  sp@+,a5/a6      | Restore registers
    unlk    a0              | Unlink
    rts                     | Return to calling program
/*
-------------------------------------------------------
-                                                     -
-   math.h                12/10/87          Patrick J. Lord   -
-                                                     -
-   This include file is to be inserted at the beginning of    -
-   a C program for the Motorola 68020 VME133.                 -
-   it generates proper function declarations for math funct.  -
-                                                     -
-------------------------------------------------------
*/
extern double sin();
extern double cos();
extern double tan();
extern double asin();
extern double acos();
extern double atan();
extern double sqrt();
extern double sincos();

extern int   abs();
extern float fabs();

/* End of function declaration */
|-----------------------------------------------------
|-                                                    -
|-   math.s                    12/10/87       Patrick J. Lord   -
|-                                                    -
|-   Motorola library 68881 vme133 math.h.  Uses doubles!!!!    -
|-                                                    -
|-----------------------------------------------------

| Declare functions for external use

    .text
    .globl _sin
_sin:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    fsinx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .text
    .globl _sincos
_sincos:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    movel   a6@(0x10),a0    | pointer to cos
    movel   a6@(0x14),a1    | pointer to sin
| This is a hack because we dont know what the sun assembler mnemonics
| are for triple operand instructions.
    .long   0xf40000b0
    fsincosx  fp0,fp0:fp1   | fp0 - cos, fp1 -sin
    fmoved  fp0,a0@         | return cos of theta
    fmoved  fp1,a1@
    unlk    a6
    rts

    .globl _cos
_cos:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    fcosx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .globl _tan
_tan:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    ftanx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .globl _sqrt
_sqrt:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    fsqrtx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .globl _asin
_asin:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    fasinx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .globl _acos
_acos:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    facosx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

    .globl _atan
_atan:
    link    a6,#0
    fmoved  a6@(0x8),fp0    | Unary operand which is a double
    fatanx   fp0,fp0
    fmoved  fp0,a6@(0x8)
    movl    a6@(0x8),d0
    movl    a6@(0xc),d1
    unlk    a6
    rts

/*
-------------------------------------------------------
-                                                     -
-   mcc.c                 12/10/87          Patrick J. Lord   -
-                                                     -
-   mcc:                                              -
-   Parses compiling and linking command for code to be loaded -
-   on the vme133.  It uses the Sun CC compiler and LD linker. -
-                                                     -
-------------------------------------------------------
*/

#include <stdio.h>
#include <sys/file.h>

char *rindex();

#ifndef START_ADDRESS
#define START_ADDRESS "10020"
#endif

#define DEFAULT_LIBS  "/usr2/pat/condor/vme133"
char libpath[512];

#define DEFAULT_MACHINE "vme133"
char machinename[80];
#define DEFAULT_OUTPUT "a.out"
char outfile[512];
char crt0[512];
char libc[512];
char libdir[512];

main(argc, argv)
int argc;
char *argv[];
    {
    int val, status, no_args, outp, i, no_ldargs;
    char **farg;
    char **ldarg;

    no_args = 0;
    no_ldargs = 0;
    outp = 0;
    farg = (char **)calloc((unsigned) argc+10, sizeof(char **));
    ldarg = (char **)calloc((unsigned) argc+10, sizeof(char **));

    farg[no_args++] = "/bin/cc";
    farg[no_args++] = "-c";

    ldarg[no_ldargs++] = "/bin/ld";

    strcpy(libpath, DEFAULT_LIBS);
    strcpy(machinename, DEFAULT_MACHINE);
    strcpy(outfile, DEFAULT_OUTPUT);

    /* check "-options" for the compiler and the linker */

    for(i=1;i<argc;i++)
        {
        if(argv[i][0] == '-')
            {
            switch(argv[i][1])
                {
                case 'L':
                if((i+1) >= argc)
                    {
```

```
                fprintf(stderr,"-L should be followed by a
directory\n");
                goto usage;
                }
            ++i;
            strcpy(libpath, argv[i]);
            break;

            case 'M':
            if((i+1) >= argc)
                {
                fprintf(stderr,"-M should be followed by a machine
name\n");
                goto usage;
                }
            ++i;
            strcpy(machinename, argv[i]);
            break;

            case 'o':
            outp++;
            if((i+1) >= argc)
                {
                fprintf(stderr, "-o should be followed by an
filename\n");
                goto usage;
                }
            ++i; strcpy(outfile, argv[i]);
            break;
            default:
            farg[no_args++] = argv[i];
            break;
            }
        }
    else
        {
        /* has to be a library or .o, .c file */
        farg[no_args++] = argv[i];
        }
    }
switch(val = vfork())
    {
    case -1:
    fprintf(stderr, "%s: No more processes?\n", argv[0]);
    exit(0);
    case 0:
    /* Execute the CC command */
    execv("/bin/cc", farg);
    break;
    default:
    while(val != wait(&status));
    if((val=(status&0377)) != 0 && val != 14)
        if(val != 2) exit(0);
    }
if(outp == 0)
    {
    printf("No linking. \n");
    exit(0);
    }

strcpy(libdir, "-L");
strcat(libdir, libpath);
ldarg[no_ldargs++] = libdir;

ldarg[no_ldargs++] = "-T";
ldarg[no_ldargs++] = START_ADDRESS;


/* make up the crt0 file */
strcpy(crt0, libpath);
strcat(crt0, "/crt0-");
strcat(crt0, machinename);
strcat(crt0, ".o");

ldarg[no_ldargs++] = crt0;

for(i=0;i<no_args;i++)
    {
    if(iscfile(farg[i]))
        ldarg[no_ldargs++] = farg[i];
    }

/* now make up the lib */
strcpy(libc, libpath);
strcat(libc, "/libc-");
strcat(libc, machinename);
strcat(libc, ".a");

ldarg[no_ldargs++] = libc;

/* add any library as specified by the user input command */

for(i=0;i<no_args;i++)
    {
    if(islibfile(farg[i]))
        ldarg[no_ldargs++] = farg[i];
    }

ldarg[no_ldargs++] = libc;
ldarg[no_ldargs++] = "-o";
ldarg[no_ldargs++] = outfile;

/* now do the linking */
switch(val = vfork())
    {
    case -1:
    fprintf(stderr, "%s: No more processes?\n", argv[0]);
    exit(0);
    case 0:
    execv("/bin/ld", ldarg);
    break;
    default:
    while(val != wait(&status));
    if((val=(status&0377)) != 0 && val != 14)
        if(val != 2) exit(0);
    }
exit(0);

usage:
    printf("mcc [-M machine] [-L libdir] [-o output file] args\n\r");
    exit(0);
    }

/* Subroutine to check if library file */
islibfile(ptr1)
char *ptr1;
    {
    char *temp;
    register char *ptr = ptr1;
    if(*ptr++ == '-' && *ptr == 'l') return(1);
    if((temp = rindex(ptr1, '.')) == NULL) return(0);
    if(strcmp(temp, ".a") == 0) return(1);
    return(0);
    }

/* Subroutine to check if C file */
iscfile(ptr)
char *ptr;
    {
    char *temp;
    if((temp = rindex(ptr, '.')) == NULL) return(0);
    if(strcmp(temp,".c") == 0) temp[1] = 'o';
    return(1);
    }
/*
-------------------------------------------------------------
-                                                           -
-   print_double.c          12/10/87      Patrick J. Lord   -
-                                                           -
-   print_double():                                         -
-   prints a double (number) to the screen (terminal).      -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void print_double(number,number_of_decimals,pos_space)
double number;
int number_of_decimals;
char pos_space;
    {
    char line[LINE_LENGTH];

    write_double(line,number,number_of_decimals,pos_space);
    putstring(line);
    }
/*
-------------------------------------------------------------
-                                                           -
-   print_float.c           12/10/87      Patrick J. Lord   -
-                                                           -
-   print_float():                                          -
-   prints a float (number) to the screen (terminal).       -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void print_float(number,number_of_decimals,pos_space)
float number;
int number_of_decimals;
char pos_space;
    {
    print_double((double)number,number_of_decimals,pos_space);
    }
/*
-------------------------------------------------------------
-                                                           -
-   print_int.c             12/10/87      Patrick J. Lord   -
-                                                           -
-   print_int():                                            -
-   prints an integer (number) to the screen (terminal).    -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void print_int(number,pos_space)
int  number;
char pos_space;
    {
    print_long((long)number,pos_space);
    }
/*
-------------------------------------------------------------
-                                                           -
-   print_long.c            12/10/87      Patrick J. Lord   -
-                                                           -
-   print_long():                                           -
-   prints a long integer (number) to the screen (terminal).-
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void print_long(number,pos_space)
```

```
long number;
char pos_space;
    {
    char line[LINE_LENGTH];

    write_long(line,number,pos_space);
    putstring(line);
    }
/*
-------------------------------------------------------------
-                                                           -
-   printf.c                  12/10/87       Patrick J. Lord  -
-                                                           -
-   printf()                                                -
-   This subroutine simulates the C printf function for the -
-   VME133 board. (NOT a full implementation!!!!!).         -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"
printf(fmt,args)
char *fmt;
    {
    char line[LINE_LENGTH],pos_space;
    int  *largs,field,index;

    largs= &args;
    while(*fmt!=NULL)
        {
        index=0;
        /* scan the string to know what to print */
        while((*fmt!='%')&&(*fmt!=NULL))
                {
                /* if '\n' do a newline and a carriage return */
                if(*fmt=='\n')
                    {
                    line[index]='\r';
                    index++;
                    }
                line[index]= *fmt;
                fmt++;
                index++;
                }
        line[index]=NULL;
        putstring(line);

        /* if character is a '%' then check what follows and act */
        if(*fmt=='%')
            {
            /* check for appropriate format */
            fmt++;
            if(*fmt==' ')
                {
                pos_space=TRUE;
                fmt++;
                }
            else pos_space=FALSE;
            if(*fmt=='.')
                {
                fmt++;
                index=0;
                while((*fmt>='0')&&(*fmt<='9'))
                    {
                    line[index]= *fmt;
                    fmt++;
                    index++;
                    }
                line[index]=NULL;
                field=(int)scan_long(line,&index);
                }
            else field=FLOAT_FIELD;
            switch(*fmt++)
                {
                /* check if string, int, long, float, or double ..... */
                case '%':
                putstring("%");
                break;
                case 'd':
                print_int(*(int*)largs,pos_space);
                largs++;
                break;
                case 'f':
                print_float(*(double*)largs,field,pos_space);
                largs+=2;
                break;
                case 'l':
                switch(*fmt++)
                    {
                    case 'd':
                    print_long(*(long*)largs,pos_space);
                    largs++;
                    break;
                    case 'f':
                    print_double(*(double*)largs,field,pos_space);
                    largs+=2;
                    break;
                    default:
                    return(-1);
                    break;
                    }
                break;
                case 's':
                putstring(*largs);
                largs++;
                break;
                default:
                return(-1);
                break;
                }
            }
        }
    }
|-----------------------------------------------------------|
|-                                                          -
|-   putstring.s               12/10/87       Patrick J. Lord  -
|-                                                          -
|-   Motorola library 68020 vme133. Print a string to the    -
|-   terminal.                                               -
|-----------------------------------------------------------|

| Declare functions for external use

    .data
    .texc
    .globl  _putstring

_putstring:
    link    a0,#0           | Link, no local variables
    moveml  a1/a2,sp@-      | save registers

    movl    a0@(8),a1       | string pointer to a1
    movl    a1,a2           | copy to a2

PARSE:
    tstb    a2@+            | check for null
    bne     PARSE           | no keep going
    tstb    a2@-            | point a1 to NULL
    moveml  a1/a2,sp@-      | load registers
    trap    #15
    .word   0x21

| Clean up and return

    moveml  sp@+,a1/a2      | Restore registers
    unlk    a0              | Unlink
    rts                     | Return to calling program
/*
-------------------------------------------------------------
-                                                           -
-   read_double.c             12/10/87       Patrick J. Lord  -
-                                                           -
-   read_double():                                          -
-   reads & returns a double from the screen (terminal).    -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void read_double(number)
double    *number;
    {
    char line[LINE_LENGTH];
    int  index;

    getstring(line);
    *number=scan_double(line,&index);
    }
/*
-------------------------------------------------------------
-                                                           -
-   read_float.c              12/10/87       Patrick J. Lord  -
-                                                           -
-   read_float():                                           -
-   reads & returns a float from the screen (terminal).     -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void read_float(number)
float    *number;
    {
    char line[LINE_LENGTH];
    int  index;

    getstring(line);
    *number=(float)scan_double(line,&index);
    }
/*
-------------------------------------------------------------
-                                                           -
-   read_int.c                12/10/87       Patrick J. Lord  -
-                                                           -
-   read_int():                                             -
-   reads & returns an integer from the screen (terminal).  -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void read_int(number)
int    *number;
    {
    char line[LINE_LENGTH];
    int  index;

    getstring(line);
    *number=(int)scan_long(line,&index);
    }
/*
-------------------------------------------------------------
-                                                           -
-   read_long.c               12/10/87       Patrick J. Lord  -
-                                                           -
-   read_long():                                            -
-   reads & returns a long integer from the screen (terminal). -
```

```
-                                                                -
-----------------------------------------------------------------
*/
#include    "stdio.h"

void read_long(number)
long *number;
    {
    char line[LINE_LENGTH];
    int  index;

    getstring(line);
    *number=scan_long(line,&index);
    }
/*
-----------------------------------------------------------------
-  Savage.c            12/04/87          Patrick J. Lord     -
-                                                              -
-  Floating point speed and accuracy test: C version.          -
-  Derived from "Byte" article July 1987.                      -
-                                                              -
-----------------------------------------------------------------
*/
#include    <stdio.h>
#include    <math.h>

#define LOOP    25000

main()
    {
    int i;
    double a[LOOP];

    printf("\nStart\007\n");
    for(i=0;i<LOOP;i++)
        a[i]=asin(sin(acos(cos(tan(atan(sqrt((double)(i*i))))))));

    printf("LAST value is a[%d]= %.14lf\n",LOOP-1,a[LOOP-1]);
    printf("Done\007\n");
    }
/*
-----------------------------------------------------------------
-                                                               -
-   scan_double.c             12/10/87       Patrick J. Lord    -
-                                                               -
-   scan_double():                                              -
-   converts a string to a double and returns a pointer to      -
-   that double value.                                          -
-                                                               -
-----------------------------------------------------------------
*/
#include    "stdio.h"

double scan_double(line,index)
char *line;
int  *index;
    {
    int i,sindex;
    double number,offset,sign;
    char string[LINE_LENGTH];

    sindex=0;
    offset=10.;
    sign=1.;

    /* Skip initial spaces or tabs */
    while((*line==' ')||(*line=='\t'))
        {
        line++;
        *index+=1;
        }

    /* if number is negative then first char is a '-' */
    if(*line=='-')
        {
        sign= -1.;
        line++;
        *index+=1;
        }
    /* Get integer part of number first */
    while((*line!='.')&&(*line!=' ')&&
          (*line!='\t')&&(*line!=NULL))
        {
        string[sindex]= *line;
        line++;
        *index+=1;
        sindex++;
        }
    /* if character is a '.' then number is 0.something  */
    if((*(line-1)<'0')||(*(line-1)>'9'))
        {
        string[sindex]='0';
        string[sindex+1]=NULL;
        }
    else
        {
        string[sindex]=NULL;
        }
    number=(double)scan_long(string,&sindex);

    /* Get decimal value of number then */
    if((*line=='.')&&(*(line+1)>='0')&&(*(line+1)<='9'))
        {
        line++;
        *index+=1;
        string[1]=NULL;
        i=0;
        while((*(line+i)>='0')&&(*(line+i)<='9') &&(i<MAX_DECIMALS))
            {
            string[0]= *(line+i);
            number+=((double)scan_long(string,&sindex))/offset;
            offset*=10.;
            i++;
            *index+=1;
            }
        }

    /* return pointer to double value */
    return(number*sign);
    }
/*
-----------------------------------------------------------------
-                                                               -
-   scan_long.c               12/10/87       Patrick J. Lord    -
-                                                               -
-   scan_long():                                                -
-   converts a string to a long integer and returns a pointer   -
-   that long integer value.                                    -
-                                                               -
-----------------------------------------------------------------
*/
#include    "stdio.h"

long scan_long(line,index)
char *line;
int  *index;
    {
    register i;
    long number,factor_of_10,digit_length,sign;
    char *first_digit,*last_digit;

    sign=1;

    /* Skip initial spaces or tabs */
    while((*line==' ')||(*line=='\t'))
        {
        line++;
        *index+=1;
        }
    /* if first character is a digit or a '-' followed by a digit
       go on and try to get integer out of string                  */
    if(((*line>='0')&&(*line<='9'))||
       ((*line=='-')&&(*(line+1)>='0')&&(*(line+1)<='9')))
        {

        /* If character is '-' then number is negative */
        if(*line=='-')
            {
            sign= -1;
            line++;
            *index+=1;
            }

        first_digit=line;

        /* Parse string for length of number */
        while((*line>='0')&&(*line<='9'))
            {
            line++;
            *index+=1;
            }

        /* If after last digit charater is not alphabetic then
           go on and calculate integer value                    */
        if((*line==' ')||(*line=='\t')||(*line==NULL))
            {
            last_digit=line-1;
            digit_length=(last_digit-first_digit)+1;
            factor_of_10=1;
            for(i=1;i<digit_length;i++) factor_of_10*=10;
            number=0;
            for(line=first_digit;line<=last_digit;line++)
                {
                number+=(*line-'0')*factor_of_10;
                factor_of_10/=10;
                }
            }

        /* If after last digit character is alphabetic: error */
        else
            {
            putstring("\007 * Error *");
            putstring(" numeric has alphabetic characters!\n\r");
            abort();
            }
        }

    /* If string is not a number: error */
    else
        {
        putstring("\007 * Error * input is not a numeric value!\n\r");
        abort();
        }

    /* return pointer to integer value */
    return(number*sign);
    }
/*
-----------------------------------------------------------------
-                                                               -
-   scanf.c                   12/10/87       Patrick J. Lord    -
-                                                               -
-   scanf()                                                     -
-   This function simulates the C scanf function for the        -
-   VME133. (NOT a full implementation!!!!!)                    -
-                                                               -
```

```
-------------------------------------------------------
*/
#include    "stdio.h"

scanf(fmt,args)
char *fmt;
    {
    int **largs,index;
    char line[LINE_LENGTH];

    largs= (int**)&args;
    index=0;
    line[index]=NULL;

    while(*fmt!=NULL)
        {
        while((*fmt!='%')&&(*fmt!=NULL)) fmt++;
        fmt++;

        while((line[index]==' ')||(line[index]=='\t')||
              (line[index]==NULL))
            {
            if(line[index]==NULL)
                {
                getstring(line);
                index=0;
                }
            else index++;
            }

        switch(*fmt++)
            {
            case 'd':
            *(int*)*largs++=(int)scan_long(&line[index],&index);
            break;
            case 'f':
            *(float*)*largs++=(float)scan_double(&line[index],&index);
            break;
            case 'l':
            switch(*fmt++)
                {
                case 'd':
                *(long*)*largs++=scan_long(&line[index],&index);
                break;
                case 'f':
                *(double*)*largs++=scan_double(&line[index],&index);
                break;
                default:
                return(-1);
                break;
                }
            break;
            case 's':
            while(line[index]!=NULL)
                {
                *((char*)*largs)++=line[index];
                index++;
                }
            *(char*)*largs++=NULL;
            break;
            default:
            return(-1);
            break;
            }
        }
    }
/*
-------------------------------------------------------
-                                                     -
-   startup.c                12/10/87      Patrick J. Lord   -
-                                                     -
-   startup:                                          -
-   This routine downloads a program to the MVME-133, opens a  -
-   virtual terminal for the 133, transfers control to the 133,-
-   and then starts another job on the sun.           -
-                                                     -
-------------------------------------------------------
*/
#include <stdio.h>
#include <sys/file.h>
#include <sys/mman.h>

#define LEN 0x100000        /*  Length of address space for all devices in
                                VME-24d32 space (in bytes)  */
#define BASE 0x100000       /*  Base address of all devices in VME-24d32
                                space  */
#define BASE133 0x0         /*  Offset of the base address of the MVME-133
                                from the base of VME-24d32 space  */
#define OFFSET133 0x10000   /*  Offset of program from the base address of
                                the MVME-133 memory  */

struct mmap_dev {           /*  Structure of memory-mapped device  */
    unsigned int *addr;     /*  Base address in virtual memory  */
    int  fd;                /*  File descriptor of device  */
};


main(argc,argv)
int  argc;
char *argv[];
    {
    int ind;
    void syserr();
    unsigned int *addr, *valloc();
    int fp, open();
    struct mmap_dev vme24d32, open_vme24d32();

    if(argc<2)
        {
        printf("\nEnter the program name: \007");
        scanf("%s",argv[0]);
        ind=0;
        }
    else ind=1;

/*  Open the VME-24d32 space  */

    vme24d32 = open_vme24d32(BASE,LEN);

/* Download program  */

    fp = open(argv[ind],00400);             /*  Open file  */
    download(fp,vme24d32,BASE133,OFFSET133);    /*  Download file to
                                                    the MVME-133  */
    close(fp);                              /*  Close file  */
    }

/*
-------------------------------------------------------
-                                                     -
-   download(input_fp,dev,base_addr,offset)           -
-                                                     -
-   This function downloads the file pointed to by input_fp   -
-   to the memory-mapped device dev.  The offset of the base  -
-   address of the device to be loaded from the base of the   -
-   device space is base_addr.  The file will be downloaded   -
-   starting from the address that is offset from base_addr   -
-   by amount offset.                                 -
-                                                     -
-------------------------------------------------------
*/

download(input_fp,dev,base_addr,offset)     /*  Download file in
input_file
                                                to memory starting at
                                                base_address  */
int input_fp;
struct mmap_dev dev;
int base_addr, offset;
    {
    long val, *point;
    int n;

    point = (long *)(dev.addr + base_addr + offset/4);  /*  Init pointer
*/
    while (n = read(input_fp,&val,4) > 0)   /*  Read in longword  */
        {
        *point = val;           /*  Write long */
        point++;
        }
    }

/*************************************************************************
*****/

/*  This function opens the VME-24d32 space.  It takes two parameters,
base and
length.  base is the (real, physical) base address of the first device in
24d32
space.  length is the length of all the memory of the devices in 24d32
space.
open_vme24d32 returns a mmap_dev value, which contains a pointer
(open_vme24d32.addr) to the base of the space in virtual memory to which
the
devices in 24d32 space are mapped.  It also contains a file descriptor for
the
24d32 devices, open_vme24d32.fd.

by Peter K. Mansfield.

*/
struct mmap_dev open_vme24d32(base,length)
int  base,length;
{
    int  prot, share, offset, len1, pagesize;
    unsigned  len;
    void  syserr();
    unsigned int *valloc();
    struct mmap_dev vme;

/*  Open the MVME-133 device  */

    if ((vme.fd = open("/dev/vme24d32",O_RDWR)) < 0)
        syserr("Can't open MVME-133");

/*  Get page size  (addresses must be multiples of page size)  */

    pagesize = getpagesize();

/*  Adjust len to multiple of page size  */

    if ((length % pagesize) == 0)
        len = length;
    else
        len = pagesize + length - (length  % pagesize);
    len1 = len;

/*  Allocate len bytes of page-aligned memory  */

    if ((vme.addr = valloc(len)) == 0)
        syserr("valloc failed");

/*  Map device memory to user space  */

    prot = PROT_READ|PROT_WRITE;
    share = MAP_SHARED;
```

```
    offset = base + BASE133;

    if (mmap(vme.addr, len1, prot, share, vme.fd, offset) != 0)
        syserr("mmap failed");

    return(vme);
}

/*************************************************************************
*****/

void syserr(msg)
    char *msg;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[];

    fprintf(stderr,"ERROR: %s (%d", msg, errno);
    if (errno > 0 && errno < sys_nerr)
        fprintf(stderr, "; %s)/n", sys_errlist[errno]);
    else
        fprintf(stderr,")0");
    exit(1);
}
/*
-------------------------------------------------------
-                                                     -
-   stdio.h              12/10/87        Patrick J. Lord   -
-                                                     -
-   This include file is to be inserted at the beginning of   -
-   a C program for the Motorola 68020 VME133.                -
-   it generates proper function declarations for some I/O.   -
-                                                     -
-------------------------------------------------------
*/
extern void   putstring();
extern void   getstring();
extern void   abort();

extern void   read_int();
extern void   print_int();

extern void   read_long();
extern long   scan_long();
extern void   print_long();
extern void   write_long();

extern void   read_float();
extern void   print_float();

extern void   read_double();
extern double scan_double();
extern void   print_double();
extern void   write_double();

extern void   exit();
extern int    scanf();
extern int    printf();

/* End of function declaration */

/* Constant declarations */

#define NULL            0
#define LINE_LENGTH     255
#define FLOAT_FIELD     6
#define MAX_DECIMALS    20
#define FALSE           0
#define TRUE            1

/* End of constant declarations */
/*
-------------------------------------------------------
-  Savage.c            12/04/87        Patrick J. Lord   -
-                                                     -
-  Floating point speed and accuracy test: C version.     -
-  Derived from "Byte" article July 1987.                 -
-  Sun section of the code: mailbox algorithm             -
-                                                     -
-------------------------------------------------------
*/
#include    <stdio.h>
#include    <math.h>


#include <sys/file.h>
#include <sys/mman.h>
#include <sys/types.h>

#define BASE 0x100000       /*  Base address of all devices in VME-24d32
                                space  */
#define LENGTH  0x60000     /* Length of memory desired */

#define OFFSET  0x50000     /* offset to data */

#define LOOP    25000

int *vme24d32_map();

main()
    {
    int i;
    double a[LOOP];
    double *value,*result,number;
    char *addr,*mailbox;

    /* open virtual memory space for VME133 */
    addr=(char *)vme24d32_map(BASE,LENGTH);
    mailbox=(char *)(addr+OFFSET);
    value=(double *)(mailbox+1);
    result=(double *)(mailbox+1);
    result++;

    printf("\nStart\007\n");
    for(i=0;i<LOOP;i++)
        {
        number=tan(atan(sqrt((double)(i*i))));
        while(*mailbox==1);
        *value=number;
        if(i>0) a[i-1]= *result;
        *mailbox=1;
        }
    while(*mailbox==1);
    a[LOOP-1]= *result;
    *mailbox=2;

    printf("LAST element is a[%d]= %.14lf\n",LOOP-1,a[LOOP-1]);
    printf("Done\007\n");
    }

/*
mvme133_map.c

    This function maps dual ported memory in vme24d32 space into a user
    process.

    vme24d32_map(base,length)

    where base is base address of memory to be mapped
          length is length of mapped segment

    Peter Mansfield
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
*/

int *vme24d32_map(base,length)
int base, length;
{
    int len1, fd, prot, share, offset, pagesize;
    unsigned len;
    int *addr;
    int *valloc();
    void syserr();

    printf("%x,%x",base,length);

/*  Open the vme24d32 device                    */

    if ((fd = open("/dev/vme24d32",O_RDWR)) < 0)
        syserr("open");

/*  Get page size (addresses must be multiples of page size)    */

    pagesize = getpagesize();

/*  Adjust len to multiple of page size                 */

    len = pagesize + length - (length % pagesize);
    len1 = len;


/*  Allocate len bytes of page-aligned memory           */

    if ((addr = valloc(len)) == 0)
        syserr("valloc failed");

    prot = PROT_READ|PROT_WRITE;
    share = MAP_SHARED;
    offset = base;

/*  Map device memory to user space                 */
    if (mmap(addr, len1, prot, share, fd, offset) != 0)
        syserr("mmap failed");

    printf("function addr=%ld\n",addr);
    return (addr);
}
/*************************************************************************/

void syserr(msg)
    char *msg;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[];

    fprintf(stderr,"ERROR: %s (%d", msg, errno);
    if (errno > 0 && errno < sys_nerr)
        fprintf(stderr, "; %s)\n", sys_errlist[errno]);
    else
        fprintf(stderr,")0");
    exit(1);
}
/*-------------------------------------------------
  -          Selspot Camera Correction Subroutine          -
  -------------------------------------------------
*/
#include "t5globdef.h"
#include <sys/file.h>
#include <sys/mman.h>

#define BASE 0x100000       /*  Base address of all devices in VME-24d32
                                space  */
#define LENGTH  0x100000     /* Length of memory desired */

#define OFFSET  0x90000     /* offset to data */
```

```
int *vme24d32_map();

float *t5camcorec(pd,start_addr,calib_dir)
struct header *pd;
float *start_addr;
char calib_dir[];
    {
    register int i,j,k;
    char  corec_table_name[2*MAXCAM][MAXLINE];
    int   binfil,x_val,y_val,ind_x,ind_y,table_size;
    float *xaddr,*yaddr,*errmat,*calloc(),*addr;
    float interp_x,interp_y,interp_xo,interp_yo,interp_coeff[4];
    float correct_x,correct_y,error_x[MAXCAM*2],error_y[MAXCAM*2];
    float *data_3dp,*data_raw,*data_3r;
    char *mailbox,*vme133;
    int *pd_offset;
    struct header *pd_addr;

    printf("Correct the camera nonlinearities.\n\n");

    table_size=MCOREC*MCOREC;
    errmat=calloc(table_size*MAXCAM*2,sizeof(float));
    sprintf(&corec_table_name[0][0],"%s/s2er1x.acc",calib_dir);
    sprintf(&corec_table_name[1][0],"%s/s2er2x.acc",calib_dir);
    sprintf(&corec_table_name[2][0],"%s/s2er1y.acc",calib_dir);
    sprintf(&corec_table_name[3][0],"%s/s2er2y.acc",calib_dir);
    addr=errmat;

    for(i=0;i<MAXCAM*2;i++)
        {
        if((binfil=open(&corec_table_name[i][0],0))==NONE)
            {
            printf("*** Warning Error ***: can't open %s\007\n",
                &corec_table_name[i][0]);
            exit(TRUE);
            }
        else
            {
            read(binfil,addr,table_size*sizeof(float));
            close(binfil);
            addr+=table_size;
            }
        }
    vme133=(char *)vme24d32_map(BASE,LENGTH);
    mailbox=(char *)(vme133+OFFSET);
    pd_offset=(int *)(mailbox+1);
    data_raw=(float *)(pd_offset+1);
    data_3r=(float *)(data_raw+4);

    while(*mailbox!=0);
    pd_addr=(struct header *)(vme133+*pd_offset);
    write_header_to_other_processor(pd,pd_addr);
    *mailbox=1;
    data_3dp=calloc(pd->nb_frames*pd->nb_channel*3,sizeof(float));
    for(j=0;j<pd->nb_frames*pd->nb_channel;j++)
        {
        for(i=0;i<MAXCAM;i++)
            {
            xaddr=start_addr+(i*2+4*j);
            yaddr=start_addr+(i*2+4*j)+1;
            if((*xaddr<=3992.)&&(*xaddr>=104.)&&
               (*yaddr<=3992.)&&(*yaddr>=104.))
                {
                ind_x=(int)((*xaddr-23.)/81.);
                x_val=ind_x+1;
                ind_y=(int)((*yaddr-23.)/81.);
                y_val=ind_y+1;

                error_x[0]= *(errmat+(y_val+i*MCOREC)*MCOREC+x_val);
                error_x[1]= *(errmat+(y_val+i*MCOREC)*MCOREC+ind_x);
                error_x[2]= *(errmat+(ind_y+i*MCOREC)*MCOREC+ind_x);
                error_x[3]= *(errmat+(ind_y+i*MCOREC)*MCOREC+x_val);

                error_y[0]= *(errmat+(y_val+(i+2)*MCOREC)*MCOREC+x_val);
                error_y[1]= *(errmat+(y_val+(i+2)*MCOREC)*MCOREC+ind_x);
                error_y[2]= *(errmat+(ind_y+(i+2)*MCOREC)*MCOREC+ind_x);
                error_y[3]= *(errmat+(ind_y+(i+2)*MCOREC)*MCOREC+x_val);

                if(error_x[0]<= -99.9){*xaddr= -9999.;*yaddr= -9999.;}
                else if(error_x[1]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_x[2]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_x[3]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_y[0]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_y[1]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_y[2]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else if(error_y[3]<= -99.9){*xaddr= -9999.;*yaddr= -
9999.;}
                else
                    {
                    interp_xo=(((float)(ind_x)*81.)+23.)+40.5;
                    interp_yo=(((float)(ind_y)*81.)+23.)+40.5;

                    interp_x=(*xaddr-interp_xo)/40.5;
                    interp_y=(*yaddr-interp_yo)/40.5;

                    interp_coeff[0]=(1.+interp_x)*(1.+interp_y)/4.;
                    interp_coeff[1]=(1.-interp_x)*(1.+interp_y)/4.;
                    interp_coeff[2]=(1.-interp_x)*(1.-interp_y)/4.;
                    interp_coeff[3]=(1.+interp_x)*(1.-interp_y)/4.;

                    correct_x=0.;
                    correct_y=0.;
                    for(k=0;k<4;k++)
                        {
                        correct_x+=interp_coeff[k]*error_x[k];
                        correct_y+=interp_coeff[k]*error_y[k];
                        }

                    *xaddr-=correct_x+2048.;
                    *yaddr-=correct_y+2048.;
                    *yaddr*= -1.;
                    }
                }
            else {*xaddr= -9999.;*yaddr= -9999.;}
            }
        while(*mailbox==1);
        *data_raw= *(xaddr-2);
        *(data_raw+1)= *(yaddr-2);
        *(data_raw+2)= *xaddr;
        *(data_raw+3)= *yaddr;
        if(j>0)
            {
            *(data_3dp+(j-1)*3)= *data_3r;
            *(data_3dp+(j-1)*3+1)= *(data_3r+1);
            *(data_3dp+(j-1)*3+2)= *(data_3r+2);
            }
        *mailbox=1;
        }
    while(*mailbox==1);
    *(data_3dp+(j-1)*3)= *data_3r;
    *(data_3dp+(j-1)*3+1)= *(data_3r+1);
    *(data_3dp+(j-1)*3+2)= *(data_3r+2);
    *mailbox=2;
    cfree(errmat);
    cfree(start_addr);
    return(data_3dp);
    }

write_header_to_other_processor(pd,pd_addr)
struct header *pd,*pd_addr;
    {
    register i;

    for(i=0;i<NAMESIZE;i++)
        {
        pd_addr->file_raw[i]=pd->file_raw[i];
        pd_addr->dir_raw[i]=pd->dir_raw[i];
        pd_addr->file_segm[i]=pd->file_segm[i];
        }
    for(i=0;i<MAXDESCR;i++) pd_addr->description[i]=pd->description[i];
    pd_addr->filter_raw=pd->filter_raw;
    pd_addr->filter_3d=pd->filter_3d;
    pd_addr->filter_bcs=pd->filter_bcs;
    pd_addr->filter_der=pd->filter_der;
    pd_addr->filter_fpl=pd->filter_fpl;
    pd_addr->interp_raw=pd->interp_raw;
    pd_addr->interp_3d=pd->interp_3d;
    pd_addr->interp_bcs=pd->interp_bcs;
    pd_addr->data_windo=pd->data_windo;
    pd_addr->velaccomp=pd->velaccomp;
    pd_addr->only_3d=pd->only_3d;
    pd_addr->bad_points=pd->bad_points;
    pd_addr->nb_frames=pd->nb_frames;
    pd_addr->nb_segment=pd->nb_segment;
    pd_addr->nb_channel=pd->nb_channel;
    pd_addr->sequence=pd->sequence;
    pd_addr->no_op=pd->no_op;
    pd_addr->act_frame1=pd->act_frame1;
    pd_addr->f_channels=pd->f_channels;
    pd_addr->f_segments=pd->f_segments;
    pd_addr->sray_badpt=pd->sray_badpt;
    pd_addr->f_wframe=pd->f_wframe;
    pd_addr->l_wframe=pd->l_wframe;
    for(i=0;i<MAXLED;i++)
        {
        pd_addr->led_segmnb[i]=pd->led_segmnb[i];
        pd_addr->led_xcoord[i]=pd->led_xcoord[i];
        pd_addr->led_ycoord[i]=pd->led_ycoord[i];
        pd_addr->led_zcoord[i]=pd->led_zcoord[i];
        }
    for(i=0;i<MAXCAM;i++)
        {
        pd_addr->cam_pos[i][0]=pd->cam_pos[i][0];
        pd_addr->cam_pos[i][1]=pd->cam_pos[i][1];
        pd_addr->cam_pos[i][2]=pd->cam_pos[i][2];
        pd_addr->cam_angles[i][0]=pd->cam_angles[i][0];
        pd_addr->cam_angles[i][1]=pd->cam_angles[i][1];
        pd_addr->cam_angles[i][2]=pd->cam_angles[i][2];
        }
    pd_addr->focal_cam1=pd->focal_cam1;
    pd_addr->focal_cam2=pd->focal_cam2;
    pd_addr->max_ledvar=pd->max_ledvar;
    pd_addr->skewray_mx=pd->skewray_mx;
    pd_addr->frequency=pd->frequency;
    pd_addr->worst_led=pd->worst_led;
    pd_addr->globx_axis=pd->globx_axis;
    pd_addr->globy_axis=pd->globy_axis;
    pd_addr->globz_axis=pd->globz_axis;
    pd_addr->fplscl=pd->fplscl;
    pd_addr->fpxscl=pd->fpxscl;
    pd_addr->fpzscl=pd->fpzscl;
    pd_addr->freq_raw=pd->freq_raw;
    pd_addr->freq_3d=pd->freq_3d;
    pd_addr->freq_bcs=pd->freq_bcs;
    pd_addr->freq_der=pd->freq_der;
    pd_addr->freq_fpl=pd->freq_fpl;
    }

/*
```

```
mvme133_map.c

    This function maps dual ported memory in vme24d32 space into a user
    process.

    vme24d32_map(base,length)

    where base is base address of memory to be mapped
          length is length of mapped segment

    Peter Mansfield
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
*/

int *vme24d32_map(base,length)
int base, length;
{
    int len1, fd, prot, share, offset, pagesize;
    unsigned len;
    int *addr;
    int *valloc();
    void syserr();

    printf("%x,%x",base,length);

/*  Open the vme24d32 device                    */

    if ((fd = open("/dev/vme24d32",O_RDWR)) < 0)
        syserr("open");

/*  Get page size (addresses must be multiples of page size)    */

    pagesize = getpagesize();

/*  Adjust len to multiple of page size                 */

    len = pagesize + length - (length % pagesize);
    len1 = len;


/*  Allocate len bytes of page-aligned memory           */

    if ((addr = valloc(len)) == 0)
        syserr("valloc failed");

    prot = PROT_READ|PROT_WRITE;
    share = MAP_SHARED;
    offset = base;

/*  Map device memory to user space                 */
    if (mmap(addr, len1, prot, share, fd, offset) != 0)
        syserr("mmap failed");

    printf("function addr=%ld\n",addr);
    return (addr);
}
/*****************************************************************/

void syserr(msg)
    char *msg;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[];

    fprintf(stderr,"ERROR: %s (%d", msg, errno);
    if (errno > 0 && errno < sys_nerr)
        fprintf(stderr, "; %s)\n", sys_errlist[errno]);
    else
        fprintf(stderr,")0");
    exit(1);
}
/*-----------------------------------------------
  -        Selspot 3-D Point Calculation Subroutine        -
  -----------------------------------------------*/
#include "st5globdef.h"

#define OFFSET  0x90000     /* offset to data */

main()
    {
    struct header pd;
    float camang_1,camang_2,camang_avg,f_matrix[2][4];
    float z_val1,z_val2,dot11,dot22,dot12,r_denom,vector_a,vector_b;
    float error_mag,error_x,error_y,error_z,ax_val,ay_val,az_val,mx_sk;
    float *xyz_3dp,x1,y1,x2,y2;

    float *data_raw;
    char *mailbox;
    int *pd_addr;

    printf("\nTRACK 3D point subroutine: waiting for Sun CPU go flag\n");

    mailbox=(char *)(OFFSET);
    pd_addr=(int *)(mailbox+1);
    data_raw=(float *)(pd_addr+1);
    xyz_3dp=(float *)(data_raw+4);

    *pd_addr=(int) &pd;
    *mailbox=0;
    while(*mailbox==0);

    printf("\nCompute the 3-D point coordinates.\n");

    camang_1=PI*pd.cam_angles[0][1]/180.;
    camang_2=PI*pd.cam_angles[1][1]/180.;
    f_matrix[0][0]=sin(camang_1);
    f_matrix[0][1]=cos(camang_1);
    f_matrix[0][2]= -pd.focal_cam1*f_matrix[0][0];
    f_matrix[0][3]=pd.focal_cam1*f_matrix[0][1];
    f_matrix[1][0]=sin(camang_2);
    f_matrix[1][1]=cos(camang_2);
    f_matrix[1][2]= -pd.focal_cam2*f_matrix[1][0];
    f_matrix[1][3]=pd.focal_cam2*f_matrix[1][1];
    camang_avg=(fabs(camang_1)+fabs(camang_2))/2.;

    mx_sk=pd.skewray_mx*pd.cam_pos[1][0]*AGDISC/(2.*sin(camang_avg));

    *mailbox=0;
    while(*mailbox!=2)
        {
    while(*mailbox==0);
    if(*mailbox==1)
        {
        x1= *data_raw;
        y1= *(data_raw+1);
        x2= *(data_raw+2);
        y2= *(data_raw+3);
        if((x1> -9998.)&&(x2> -9998.))
            {
            z_val1 = x1*f_matrix[0][0]+f_matrix[0][3];
            z_val2 = x2*f_matrix[1][0]+f_matrix[1][3];
            x1= x1*f_matrix[0][1]+f_matrix[0][2];
            x2= x2*f_matrix[1][1]+f_matrix[1][2];

            dot11= x1*x1+y1*y1+z_val1*z_val1;
            dot22= x2*x2+y2*y2+z_val2*z_val2;
            dot12= x1*x2+y1*y2+z_val1*z_val2;

            r_denom=pd.cam_pos[1][0]/(dot12*dot12-dot11*dot22);

            vector_a=(x2*dot12-x1*dot22)*r_denom;
            vector_b=(x2*dot11-x1*dot12)*r_denom;

            ax_val=vector_a*x1;
            error_x=ax_val-pd.cam_pos[1][0]-vector_b*x2;
            *xyz_3dp=ax_val-.5*error_x-pd.globx_axis;

            ay_val=vector_a*y1;
            error_y=ay_val-vector_b*y2;
            *(xyz_3dp+1)=ay_val-.5*error_y+pd.cam_pos[0][1]-pd.globy_axis;

            az_val=vector_a*z_val1;
            error_z=az_val-vector_b*z_val2;
            *(xyz_3dp+2)=az_val-.5*error_z-pd.globz_axis;


error_mag=sqrt(error_x*error_x+error_y*error_y+error_z*error_z);
            if(error_mag>=mx_sk)
                {
                pd.sray_badpt++;
                *xyz_3dp= -1000.;
                *(xyz_3dp+1)= -1000.;
                *(xyz_3dp+2)= -1000.;
                error_mag=0.;
                }
            }
        else
            {
            *xyz_3dp= -1000.;
            *(xyz_3dp+1)= -1000.;
            *(xyz_3dp+2)= -1000.;
            error_mag=0.;
            }
        *mailbox=0;
        }
        }

    printf("     The total number of bad points[frame-channels]
eliminated\n");
    printf("     by this routine due to skew ray errors was: %d out of:
%d\n\n",
            pd.sray_badpt,pd.nb_frames*pd.nb_channel);
    exit();
    }

/*
-------------------------------------------------------
-  Savage.c            12/04/87          Patrick J. Lord    -
-                                                           -
-  Floating point speed and accuracy test: C version.       -
-  Derived from "Byte" article July 1987.                   -
-  VME133 section of the code: mailbox algorithm            -
-  Sun and VME133 need to run in parallel.                  -
-                                                           -
-------------------------------------------------------
*/
#include    "vme133/stdio.h"
#include    "vme133/math.h"

#define OFFSET  0x50000     /* offset to data */

main()
    {
    char    *mailbox;
    double *value,*result,number;

    mailbox=(char *)OFFSET;
    value=(double *)(mailbox+1);
    result=(double *)(mailbox+1);
    result++;
    *mailbox=0;

    printf("\nComputing & Waiting for output\n");
    while(*mailbox!=2)
```

```
        {
        while(*mailbox==0);
        if(*mailbox==1)
            {
            number= *value;
            *result=asin(sin(acos(cos(number))));
            *mailbox=0;
            }
        }
    printf("the value is %lf\n",*result);
    exit();
    }

/*
-------------------------------------------------------------
-                                                           -
-   write_double.c            12/10/87       Patrick J. Lord  -
-                                                           -
-   write_double():                                         -
-   converts a double (number) to a string and returns a    -
-   pointer to the beginning of that string.                -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void write_double(line,number,number_of_decimals,pos_space)
char line[];
double number;
int  number_of_decimals;
char pos_space;
    {
    int i,index;

    index=0;
    if((pos_space)&&(number>=0.))
        {
        line[index]=' ';
        index++;
        }

    /* Check if negative number to print a '-' character */
    if(number<0.)
        {
        line[index]='-';
        index++;
        number*= -1.;
        }
    /* write the integer part of the number first */
    write_long(&line[index],(long)number,0);
    for(i=0;line[index]!=NULL;i++) index++;

   /* place the '.' if necessary */
    if(number_of_decimals>0)
        {
        line[index]='.';
        index++;
        }
    if(number_of_decimals>MAX_DECIMALS)
        number_of_decimals=MAX_DECIMALS;

   /* Then print the decimal values */
    for(i=0;i<number_of_decimals;i++)
        {
        number-=(long)number;
        number*=10.;
        write_long(&line[index],(long)number,0);
        index++;
        }
    line[index]=NULL;
    }
/*
-------------------------------------------------------------
-                                                           -
-   write_long.c              12/10/87       Patrick J. Lord  -
-                                                           -
-   write_long():                                           -
-   converts a long integer (number) to a string and returns a -
-   pointer to the beginning of that string.                -
-                                                           -
-------------------------------------------------------------
*/
#include    "stdio.h"

void write_long(line,number,pos_space)
char line[];
long number;
char pos_space;
    {
    register line_index;
    long  factor_of_10,number_of_digits,offset;

    line_index=0;
    if((pos_space)&&(number>=0))
        {
        line[line_index]=' ';
        line_index++;
        }

    /* Check if negative number to print a '-' character */
    if(number<0)
        {
        line[line_index]='-';
        line_index++;
        number*= -1;
        }

    factor_of_10=1;
    number_of_digits=0;

    /* Figure out the number of digits in number */
    while((long)(number/factor_of_10)>=10)
        {
        factor_of_10*=10;
        number_of_digits++;
        }

    /* Convert every digit to ASCII character and place it in string */
    while(number_of_digits>=0)
        {
        offset=(long)(number/factor_of_10);
        line[line_index]='0'+offset;
        line_index++;
        number-=offset*factor_of_10;
        factor_of_10/=10;
        number_of_digits--;
        }

    /* Terminate string with NULL character and return string pointer */
    line[line_index]=NULL;
    }
```

Appendix B

Source Code: Real-Time TRACK Software

```
# Makefile for RT-TRACK
#
# Patrick J. Lord       August 3, 1988.
#
# Modified:
#

.SUFFIXES: .o  .c .s .f

CC  = cc
MCC = /usr2/pat/condor/vme133/mcc

SELSDIR  = /usr/util/track/Source
DDTI  = /usr/util/track/include
DDSSS = /usr/util/selspot/sun/Source
DDSSI = /usr/util/selspot/sun/include
DV    = /usr/util/vme_map
UPD = /usr2/pete/dlt
UUDI = /usr/util/dr11w/include

FILES0 = ${SELSDIR}/t5s_treein.o ${SELSDIR}/t5s_typpar.o \
         ${SELSDIR}/t5s_chgpar.o ${SELSDIR}/t5s_lissgf.o \
         ${SELSDIR}/t5s_queryn.o ${SELSDIR}/t5s_termin.o \
         ${SELSDIR}/t5s_lphead.o ${SELSDIR}/t5s_getcor.o \
         ${SELSDIR}/t5s_schuta.o ${SELSDIR}/t5s_savraw.o \
         ${SELSDIR}/t5s_rawchk.o ${SELSDIR}/t5s_lisfls.o \
         ${SELSDIR}/t5s_lisdir.o

FILES1 = rts_cpt3dv.o rts_cptbcs.o server_connect.o local_accept.o \
         rts_intcor.o rts_cptfpd.o

FILES2 = ${SELSDIR}/t5s_treein.o ${SELSDIR}/t5s_termin.o \
         ${SELSDIR}/t5s_queryn.o

FILES3 = rts_mvmego.o rts_int133.o

MAPSRC =  ${DV}/vme24d32_map.o \
      ${DV}/vme16d16_map.o \
      ${DV}/shortio_map.o

SELSRC = ${DDSSS}/s2error.o \
      ${DDSSS}/s2read.o \
      ${DDSSS}/s2start.o \
      ${DDSSS}/s2sload.o \
      ${DDSSS}/s2setseq.o \
      ${DDSSS}/s2framsiz.o \
      ${DDSSS}/s2setsync.o \
      ${DDSSS}/s2lsrec.o \
      ${DDSSS}/s2init.o \
      ${DDSSS}/s2reset.o \
      ${DDSSS}/s2stop.o

SELSPOT = ${MAPSRC} ${SELSRC}

TRACK_INCLUDE = -I${DDTI}

TRACK_INC_FILES =  ${DDTI}/t5i_defini.h \
                   ${DDTI}/t5i_global.h \
                   ${DDTI}/t5i_struct.h \
                   ${DDTI}/t5i_sublis.h \
                   ${DDTI}/t5i_sunixh.h

SLSPT_INC_FILES = ${DDSSI}/slsptreg.h \
                  ${DDSSI}/mvmereg.h \
                  ${DDSSI}/config.h

DR11W_INC_FILES = ${UUDI}/drwreg.h \
                  ${UUDI}/drwcmd.h

CFLAGS= -O -f68881 -lm -ltermcap

MCFLAGS= -f68881 -lm

all: rtm_sample rtm_track rtm_loadin rts_int133 rtm_ksfram

rtm_sample: rtm_sample.o ${FILES1} ${FILES0} ${SELSPOT}
    @echo "> Linking" $@
    ${CC} rtm_sample.o -o rtm_sample ${FILES1} ${FILES0} ${SELSPOT}
${CFLAGS}

rtm_track: rtm_track.o ${FILES2}
    @echo "> Linking" $@
    ${CC} rtm_track.o ${FILES2} -o rtm_track ${CFLAGS}

rtm_loadin: rtm_loadin.o ${FILES3} ${MAPSRC} ${TRACK_INC_FILES}
    @echo "> Linking" $@
    ${CC} rtm_loadin.o -o rtm_loadin ${FILES3} ${CFLAGS} ${MAPSRC}

rts_int133: rts_int133.o rtm_vme133.o rts_cpt3dv.o ${MAPSRC}
${TRACK_INC_FILES}
    @echo "> Linking" $@
    ${MCC} -T 5020 rtm_vme133.o rts_int133.o rts_cpt3dv.o -o rts_int133
${MCFLAGS}

rtm_ksfram: rtm_ksfram.o ${MAPSRC} ${DR11W_INC_FILES}
    @echo "> Linking" $@
    ${CC} rtm_ksfram.o rts_cptfpd.o -o rtm_ksfram ${CFLAGS} ${MAPSRC}

rtm_track.o:  ${SLSPT_INC_FILES}
rtm-sample.o: ${SLSPT_INC_FILES}
rtm_loadin.o: ${SLSPT_INC_FILES}
rtm_ksfram.o: ${DR11W_INC_FILES}

${FILES0}: ${TRACK_INC_FILES}
${FILES1}: ${TRACK_INC_FILES}
${FILES2}: ${TRACK_INC_FILES}

${MAPSRC}:
    (cd ${DV}; ${MAKE} all)

${SELSRC}: ${SLSPT_INC_FILES}
    (cd ${DDSSS}; ${MAKE} all)

# Make .o files from the .c files
.c.o: $*.c
    @echo "> Compiling" $@ because of change to $?
    ${CC) -c ${CFLAGS} ${TRACK_INCLUDE} $*.c

# Make .o files from the .f files
.f.o: $*.f
    @echo "> Compiling" $@ because of change to $?
    ${CC) -c ${CFLAGS} ${TRACK_INCLUDE} $*.f

# Make .o files from .s files
.s.o: $*.s
    @echo "> Assembling" $@ because of change to $?
    as $*.s -o $*.o

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  local_accept.c
 *
 *  Description:
 *
 *  Definitions: (example call, variable types, return types,
 *                limitations, special environments required)
 *
 *  Created by: Patrick J. Lord     1-Feb-88
 *
```

```
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    <t5i_global.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int local_accept(server_socket)
int server_socket;
    {
    int sock,length;
    struct sockaddr_in sin;
    int msgsock;
    char local_name[10];

    /* create a socket */
    if ((sock = socket (AF_INET,SOCK_STREAM,0)) < 0)
        {
        perror("opening stream socket");
        exit(0);
        }

    /* initialize socket data structure */
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = 0;
    /*
    sin.sin_port = htons(IPPORT_RESERVED + 1);
    */

    /* bind socket data structure to this socket */
    if (bind (sock,&sin,sizeof(sin)))
        {
        perror("binding stream socket");
        }

    /* getsockname fills in the socket structure with information, such as
       the port number assigned to this socket */

    length = sizeof(sin);
    if (getsockname (sock,&sin,&length))
        {
        perror("getting socket name");
        exit(0);
        }
    printf ("Socket has port# %d\n",htons(sin.sin_port));

    /* prepare socket queue for connection requests and accept connections
*/
    sprintf(local_name,"busch");
    write(server_socket,local_name,10*sizeof(char));

    listen(sock,5);
    /*
    if ((msgsock = accept(sock,0,0)) <= 0) {
    */
    if ((sock = accept(sock,0,0)) <= 0) {
        perror("accept on socket");
        exit(0);
        }
    /*
    return(msgsock);
    */
    return(sock);
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_defini.h
 *
 *  Description: This file contains all the global definitions
 *               (including macro definitions) used by TRACK5.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#define     TRACK_DIR     "track"
#define     TRACK_PATH    "/usr/util/track"
#define     CALIB_DIR     "/usr/util/track/calibration"
#define     NONE         -1
#define     FALSE         0
#define     TRUE          1
#define     MAXLED       32       /* maximum number of channels */
#define     MAXSEG       10       /* maximum number of segments */
#define     MAXCAM        2       /* maximum number of cameras  */
#define     NUMCAM        2       /* actual number of cameras   */
#define     MAXPTS      144       /* max. points in calibration */
#define     KSCHAN        8       /* channels per Kistler frame */
#define     KSATOD        0.005   /* forceplate A/D volts/units */
#define     KSA           0.12    /* Kistler dimension A (m)    */
#define     KSB           0.2     /* Kistler dimension B (m)    */
#define     KSAZ         -0.054   /* Kistler dimension AZ (m)   */
#define     KS_F_MIN      5       /* fp minimum force for center*/
                                  /* of press. computations (N) */
#define     NAMESIZE     10       /* length of file name        */
#define     MAXLINE      80       /* maximum length of a line   */
#define     MAXDESCR     60       /* max description length     */
#define     MCOREC       51       /* size of calibration table  */
#define     TERMCAR      6        /* Terminal I/O char size     */
#define     PI           3.14159  /* definition of pi           */
#define     AGDISC       1.278E-4 /* sin of ang. discretization */
#define     FTM          0.3048   /* Conversion from feet to m. */
#define     LF           10       /* decimal value for linefeed */
#define     TIMESIZE     26       /* length of time/data array  */
#define     FL_HEADSIZE 2048      /* size of header in UNIX file*/

#define MAX(A, B)   ((A) > (B) ? (A) : (B))
#define MIN(A, B)   ((A) < (B) ? (A) : (B))
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_struct.h
 *
 *  Description: This file contains all the data structure necessary to
 *               TRACK5.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

struct t5header {
    char  track_version[NAMESIZE];      /* The version of TRACK used */
    char  file_raw[NAMESIZE];           /* Name of original raw data */
    char  dir_raw[NAMESIZE];            /* Directory of orig raw file*/
    char  file_segm[NAMESIZE];          /* Segment file name         */
    char  description[MAXDESCR];        /* Data description          */
    char  sampl_time[TIMESIZE];         /* Time of data sampling     */
    char  calib_time[TIMESIZE];         /* Time of external calib    */
    char  smooth_raw;                   /* Raw data smoothing flag   */
    char  smooth_3dp;                   /* 3D data smoothing flag    */
    char  smooth_bcs;                   /* BCS data smoothing flag   */
    char  smooth_fpd;                   /* F'plate smoothing flag    */
    char  interp_raw;                   /* raw dat interpolation flag*/
    char  interp_3dp;                   /* 3D data interpolation flag*/
    char  interp_bcs;                   /* BCS dat interpolation flag*/
    char  data_windo;                   /* processing window flag    */
    char  velaccomp;                    /* compute vel and acc flag  */
    char  only_3dp;                     /* compute 3D data only flag */
    char  ext_cal;                      /* use external calib flag   */
    char  selspot_flag;                 /* collect Selspot data flag */
    char  kistler_flag;                 /* collect Kistler data flag */
    int   bad_points;                   /* number of bad points found*/
    int   nb_frames;                    /* number of frames collected*/
    int   nb_segment;                   /* number of selspot segments*/
    int   nb_channel;                   /* number of selspot channels*/
    int   act_frame1;                   /* number of frames windowed */
    int   f_channels;                   /* number of bad channels    */
    int   f_segments;                   /* number of bad segments    */
    int   sray_badpt;                   /* bad points due to skew ray*/
    int   f_wframe;                     /* first frame of window     */
    int   l_wframe;                     /* last frame of window      */
    int   led_segmnb[MAXLED];           /* segment no. for ea. chan. */
    float led_xcoord[MAXLED];           /* seg x-coord for ea. chan. */
    float led_ycoord[MAXLED];           /* seg y-coord for ea. chan. */
    float led_zcoord[MAXLED];           /* seg z-coord for ea. chan. */
    float cam_pos[MAXCAM][3];           /* explicit cam pos. vector  */
    float cam_angles[MAXCAM][3];        /* explicit cam rot. angles  */
    float ext_rotmtx[MAXCAM][9];        /* rot matrix for ext calib  */
    float ext_trnvec[MAXCAM][3];        /* vectors for external 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 lenght err*/
    float globx_axis;                   /* global origin relocation X*/
    float globy_axis;                   /* global origin relocation Y*/
    float globz_axis;                   /* global origin relocation Z*/
    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   */
    };

struct t5tree {
    char  data[MAXLINE];
    char  parameter[MAXLINE];
    char  segment[MAXLINE];
    char  work[MAXLINE];
    };

struct t5terminal {
    char  line_up[TERMCAR];
    char  clear_scr[TERMCAR];
    char  clean_page[TERMCAR];
    int   nb_lines;
    int   nb_columns;
    };
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_global.h
 *
 *  Description: This file is the overall "include" file for all TRACK5
 *               subroutines and main programs.  Definitions and
 *               global variable & structure declarations are done
 *               within this file.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                     Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    "rti_defini.h"
#include    "rti_struct.h"
/*
 * ks_frame.c - Collect and display one frame of Kistler forceplate data
 *
 *          This routine repeatedly collects and displays one frame
 *          of Kistler forceplate data.  Exit this program with <ctrl> C.
 *
 *          Peter K. Mansfield      12-Jul-88
 *
 */

#include    <t5i_global.h>

#define     FPCHAN      0x08
#define     MAXZ        1000

extern int *vme24d32_map();
extern short *shortio_map();

main()
{
    char *mvaddr, *draddr;
    register struct mvme_reg *mvme_reg;
    register union drw_reg *drw_reg;
    short *data;
    float *data_fpr, *data_fpd, *calloc();
    int padd;
    int i, j, errcode, counter;
    char esc = (char)033;
    char chan[4];
    struct t5header file_header;

    file_header.fpxscl = 10.;
    file_header.fpyscl = 100.;
    file_header.fpzscl = 10.;
    data_fpr = calloc(8,sizeof(float));
    data_fpd = calloc(3,sizeof(float));
    for(i=0;i<KSCHAN;i++) file_header.fp_zero[i] = 0.;

    mvaddr = (char *)vme24d32_map(MBASE+MSGBUF, (S2BUF-MSGBUF));
    draddr = (char *)shortio_map();

    mvme_reg = (struct mvme_reg *)mvaddr;
    drw_reg = (union drw_reg *)(draddr+KBOFF);

    mvme_reg->mvme_drflg = 0; /* Clear DMA interrupt flag */

    mvme_reg->mvme_drerr = 0; /* Clear error interrupt flag */
    /*
    data = (short *)vme24d32_map(MBASE+KSBUF,KSLEN);
    */
    data = (short *)(mvaddr + (S2BUF-MSGBUF)+S2LEN);
    padd = MBASE + KSBUF;

    /* Check for Kistler power on and panel switch position */

    if(!(drw_bcr & STATB)) /* check Status B */
        {
        printf("Please set DR11W Select Switch on panel to FP\n");
        while(!(drw_bcr & STATB));
        }
    if(!(drw_bcr & STATC)) /* check Status C */
        {
        printf("Kistler power is OFF!\n");
        printf("Hit <CR> when power has been turned ON.");
        getchar();
        }

    /* Zero the Charge Amps (put in RESET mode) */
    drw_bcr &= ~(FNCT2|LED);    /* Clear FNCT2, LED on while resetting */
    for(i=0;i<100000;i++);  /* Delay for RESET to take effect */
    drw_bcr |= (FNCT2|LED); /* Enable Kistler Forceplate, LED off */

    /* Clear channel operation complete flag */
    drw_csr = drw_csr;

    while(!(mvme_reg->mvme_drerr))
        {
        /* Load BIM (68153) registers */
        drw_aicr = AICR;
        drw_dicr = DICR;
        drw_barx = (padd >> 16) & 0xFF; /* address bits 16-24 */
        drw_bar  = (MBASE+KSBUF) & 0xFFFF;/* address bits  0-15 */

        /* Load DMAI (68430) registers */
        drw_divr= DIVR; /* DMA interrupt vector */
        drw_ocr = OCR;  /* dev to mem, word transfer */
        drw_dcr = DCR;  /* Cycle steal mode */
        drw_wcr = FPCHAN; /* set number of words to transfer */
        drw_ccr = CCR;  /* Start, enable DMA interrupt */

        /* Load address modifier register */
        drw_amr = AMR;

        /* Set GO, frame start (FNCT1) */
        drw_bcr |= (GO|FNCT1);

        /* Wait for interrupt flag */
        /*
        printf("Waiting for DR11W DMA interrupt\n");
        fflush(stdout);
        */
        while(!(mvme_reg->mvme_drflg))
            {
            if(mvme_reg->mvme_drerr)
                {
                printf("DR11W ERROR INTERRUPT RECEIVED/n");
                exit(0);
                }
            }

        /* Clear interrupt flag */
        mvme_reg->mvme_drflg = 0;

        /* Clear channel operation complete flag */
        drw_csr = drw_csr;

        /* Clear GO, frame start (FNCT1) */
        drw_bcr &= ~(GO|FNCT1);

        if(counter<MAXZ)
            {
            for(i=0;i<KSCHAN;i++) file_header.fp_zero[i] += (float)
*(data+i);
            counter++;
            if(counter==MAXZ)
                for(i=0;i<KSCHAN;i++) file_header.fp_zero[i] /= MAXZ.;
            }
        else
            {
            for(i=0;i<KSCHAN;i++) *(data_fpr+i) = (float) *(data+i);
            rts_cptfpd(&file_header, data_fpr, data_fpd);

            /* Display Data */
            /*
            for(i=0; i<KSCHAN; i++)
                printf("Chan %2d %8d %6.1f\n", i, *(data+i), *(data_fpr+i));
            for(i=0; i<KSCHAN; i++) printf("%c[1A", esc);
            */
            printf("Weight = %5.1f lbs\r",
            sqrt(*(data_fpd)* *(data_fpd)+ *(data_fpd+1)* *(data_fpd+1)+
            *(data_fpd+2)* *(data_fpd+2))/4.448);
            fflush(stdout);
            }
        }
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rtm_loadin.c
 *
 *  Description:  This program loads the interrupt routines
 *                onto the mvme133 processor board.  Dual ported
 *                memory is mapped and the code is loaded to the
 *                mapped region.  The code is then executed to
```

```
 *               load the vector addresses and put the mvme133
 *               in a state for handling external interrupts.
 *               The code loaded is in the program rts_int133.s
 *
 *  Definitions: (example call, variable types, return types,
 *               limitations, special environments required)
 *
 *  Created by: Patrick J. Lord     08-Mar-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                        Copyright (C) 1988
 *                Massachusetts Institute of Technology
 *                      Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    <t5i_global.h>

main()
{
    char *program =
"/usr2/pat/track/development/rt_track/VME133/rts_int133";
    int fp, open();
    char *point;
    char *mvaddr;
    register struct mvme_reg *mvme_reg;

/*  Open vme24d32 space  */

    point = (char *)vme24d32_map(MBASE,S2BUF) + PROGBUF;

/* Abort the program if it is currently running */

    mvaddr = (char *)vme24d32_map(MBASE+MSGBUF, (S2BUF-MSGBUF)+S2LEN);
    mvme_reg = (struct mvme_reg *)mvaddr;
    mvme_reg->mvme_abo133 = 1;

/* Download program  */

    if ((fp = open(program,0)) == NONE)
        {
        printf("%s cannot be opened\n",program);
        exit(0);
        }

    while (read(fp,point,1) > 0) point++;

    close(fp);                                  /*  Close file  */

/* Start program */

    mvme_reg->mvme_abo133 = 0;  /* clear abort flag */
    rts_mvmego(PROGBUF);

}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rtm_sample.c
 *
 *  Description: This is the main sampling program of RT-TRACK.
 *               The procedure initializes the interface, allows
 *               for modification of sampling and processing
 *               parameters, and collects Selspot and/or forceplate
 *               data and writes the information to disk.
 *
 *  Created by:         Patrick J. Lord     01-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs: None
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                        Copyright (C) 1988
 *                Massachusetts Institute of Technology
 *                      Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */
#include    <t5i_global.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>

#define BASE    0x100000    /*  Base address of all devices in VME-24d32 */
#define LENGTH  0x0FFFFF    /* Length of memory desired */
#define OFFSET  0x90000     /* offset to data */
#define FPCHAN  0x08

#define MAXZ    10
#define NBSTOR  500

extern int *vme24d32_map();
extern short *shortio_map();

main()
    {
    struct t5header file_header, *pd_addr;
    struct t5tree t5_tree;
    struct timeval tp;
    register struct mvme_reg *mvme_reg;
    register struct slspt_reg *slspt_reg;
    register union drw_reg *drw_reg;
    char go,quit,*host="schlitz",filename[MAXLINE],p_dist_file[MAXLINE];
    char
ans=TRUE,*mailbox,*vme133,*mvaddr,*svaddr,*draddr,*extra_space,*new;
    char buffer, toggle;
    short *data_s, *data_k;
    long start_time, finish_time,start_utime, finish_utime;
    int i, j, errcode, padd, samp_period, samp_period_min, offset,
*pd_offset;
    int n, server_socket, local_socket, binfil, s2bytes, ksbytes, elements;
    int counter, rew_count;
    float *data_fpr, *data_fpd, *addr_raw, *errmat, *addr_3d, *addr_bcs;
    float *start_raw, *start_fpr, *addr_NET, *calloc(), *data_raw, *data_3r;
    float interval, *temp;
    FILE *fopen(), *p_dist;


    t5s_treein(&t5_tree);

    /* set up the header */
    sprintf(filename,"%s/t5_param",t5_tree.parameter);
    if((binfil=open(filename,0))==NONE)
        {
        printf("*** Warning Error ***: can't open %s\n",filename);
        if(!t5s_queryn("Do you want to create a new parameter file"))
exit(0);
        }
    else
        {
        read(binfil,&file_header,sizeof(struct t5header));
        close(binfil);
        }

    /* Display and Modify Parameters */

    while(ans)
      {
      t5s_typpar(&file_header,stdout);
      ans = t5s_queryn("Do you wish to make any changes");
      if(ans) t5s_chgpar(&file_header, &t5_tree);
      }

    sprintf(p_dist_file,"%s/p_dist",CALIB_DIR);
    if((p_dist=fopen(p_dist_file, "r"))==NULL)
        {
        printf("*** Warning Error **: can't open %s\007\n",p_dist_file);
        exit(TRUE);
        }
    else
        {
        fscanf(p_dist,"%f",&file_header.focal_cam1);
        fscanf(p_dist,"%f",&file_header.focal_cam2);
        fclose(p_dist);
        }

    /* Save Changed Parameters */

    binfil = creat(filename,'\644');
    write(binfil, &file_header, sizeof(struct t5header));
    close(binfil);

    if(t5s_queryn( "Do you want a hardcopy of the parameter file"))
           t5s_lphead(&file_header,&t5_tree);


    /* Setup VME addresses */
    s2bytes = file_header.nb_channel*NUMCAM*2 * sizeof(short);
    ksbytes = KSCHAN * sizeof(short);
    mvaddr = (char *)vme24d32_map(MBASE+MSGBUF, (S2BUF-MSGBUF));
    svaddr = (char *)vme16d16_map(SBASE,SLENGTH);
    draddr = (char *)shortio_map();

    mvme_reg = (struct mvme_reg *)mvaddr;
    slspt_reg = (struct slspt_reg *)svaddr;
    drw_reg = (union drw_reg *)(draddr+KBOFF);

    data_s = (short *)vme24d32_map(MBASE+S2BUF,s2bytes);
    data_k = (short *)vme24d32_map(MBASE+KSBUF,ksbytes);
    /* Allocate memory space for communication */
    addr_NET=calloc(file_header.nb_segment*15+3,sizeof(float));

    /* Setup Motorola MVME133 addresses */
    vme133=(char *)vme24d32_map(BASE,LENGTH);
    mailbox=(char *)(vme133+OFFSET);
    pd_offset=(int *)(mailbox+1);
    data_raw=(float *)(pd_offset+1);
    data_3r=(float *)(data_raw+(4*file_header.nb_channel));

    if(file_header.selspot_flag)
        {
        /* Allocate & read correction tables */
        errmat    = t5s_getcor();
        start_raw = calloc(file_header.nb_channel*4*NBSTOR,sizeof(float));
        addr_raw  = calloc(file_header.nb_channel*4,sizeof(float));
```

```
        addr_3d   = calloc(file_header.nb_channel*3,sizeof(float));
        addr_bcs  = addr_NET;

        /* Load the sequence */
        samp_period_min = file_header.nb_channel*NUMCAM*100;
        samp_period = (int)(1.E6*(1/file_header.frequency));
        if(samp_period < samp_period_min)
            {
            samp_period = samp_period_min;
            file_header.frequency = 1.E6*(1/(float)samp_period);
            printf("Warning: Sampling frequency has been ");
            printf("adjusted to %f Hertz\n",file_header.frequency);
            }

        if(s2sload(mvme_reg,
slspt_reg,file_header.nb_channel,samp_period)!=0)
            exit(s2error(mvme_reg));

        /* Get Loaded framesize */
        if(s2framsiz(mvme_reg, slspt_reg) != 0)
            exit(s2error(mvme_reg));
        if(mvme_reg->mvme_framsiz != file_header.nb_channel*NUMCAM*4)
            {
            printf("Sequence not properly loaded\n");
            exit();
            }

        while(*mailbox!='\000');
        pd_addr=(struct t5header *)(vme133+ *pd_offset);
        write_header_to_other_processor(&file_header,pd_addr);
        *mailbox='\001';

        mvme_reg->mvme_ccd = 0x0;
        mvme_reg->mvme_frames = 1;

        /* Check panel switch status for Automatic Triggering */
        if(!(drw_bcr & STATA)) /* check Status A */
            {
            printf("\nPlease set TRIGGER Switch on panel to AUTO
TRIG\007\n");
            while(!(drw_bcr & STATA));
            }
        if(s2setseq(mvme_reg, slspt_reg) != 0) exit(s2error(mvme_reg));
        }

    if(file_header.kistler_flag)
        {
        mvme_reg->mvme_drflg = 0; /* Clear DMA interrupt flag */
        mvme_reg->mvme_drerr = 0; /* Clear error interrupt flag */

        padd = MBASE + KSBUF;

        /* Check for Kistler power on and panel switch position */
        if(!(drw_bcr & STATB)) /* check Status B */
            {
            printf("Please set DR11W Select Switch on panel to FP\007\n");
            while(!(drw_bcr & STATB));
            }
        if(!(drw_bcr & STATC)) /* check Status C */
            {
            printf("Kistler power is OFF!\007\n");
            printf("Hit <CR> when power has been turned ON.");
            getchar();
            }

        /* Zero the Charge Amps (put in RESET mode) */
        drw_bcr &= ~(FNCT2|LED);     /* Clear FNCT2, LED on while resetting
*/
        for(i=0;i<100000;i++);       /* Delay for RESET to take effect */
        drw_bcr |= (FNCT2|LED);      /* Enable Kistler Forceplate, LED off */

        /* Clear channel operation complete flag */
        drw_csr = drw_csr;

        start_fpr = calloc(KSCHAN*NBSTOR,sizeof(float));
        data_fpr  = calloc(KSCHAN,sizeof(float));
        data_fpd  = addr_NET+file_header.nb_segment*15;
        for(i=0;i<KSCHAN;i++) file_header.fp_zero[i] = 0.;
        }

    quit=FALSE;
    buffer=FALSE;
    toggle =FALSE;
    counter = 0;
    /* Setup communication to graphics terminal */
    if(t5s_queryn("Do you want to use the IRIS graphics terminal"))
        {
        server_socket = server_connect(host);
        local_socket = local_accept(server_socket);
        printf("\nEthernet Network Communication Started\n");
fflush(stdout);
        if((n=read(local_socket,&go,sizeof(char))) > 0)
            {
            if (write(server_socket,&file_header,sizeof(struct t5header)) <
0)
                {
                perror("writing on stream socket");
                exit(0);
                }
            }
        if((n=read(local_socket,&go,sizeof(char))) <= 0)
            {
            perror("reading on stream socket");
            exit(0);
            }
        fcntl(local_socket,F_SETFL,O_NDELAY);
        netw = TRUE;
        }
    else netw = FALSE;

    printf("\nReal-Time Process Started\n\n"); fflush(stdout);
    if(gettimeofday(&tp,NULL)!=0)
        printf("Error from gettimeofday = %d\n",errno);
    else
        {
        start_time = tp.tv_sec;
        start_utime = tp.tv_usec;
        }

    /* Loop unless error condition or <ctrl>C. */
    while(mvme_reg->mvme_ccd == 0x0 && !(mvme_reg->mvme_drerr) && !quit)
        {
        if(file_header.selspot_flag)
            {
            if(buffer)
                for(i=0;i<file_header.nb_channel*4;i++) *(addr_raw+i) =
*(start_raw+(rew_count%NBSTOR)*file_header.nb_channel*4+i);
            else
                {
                if(s2start(mvme_reg, slspt_reg) != 0)
exit(s2error(mvme_reg));
                if(s2read(mvme_reg, slspt_reg,1) !=0)
                    {
                    errcode = s2error(mvme_reg);
                    s2stop(mvme_reg, slspt_reg);
                    exit(errcode);
                    }

                for(i=0;i<file_header.nb_channel*4;i++)
                    {
                    /* Take one's complement of data */
                    *(addr_raw+i) = (float) (~*(data_s+i));
                    /* Save Selspot raw data in ring buffer */
                    *(start_raw+(counter%NBSTOR)*file_header.nb_channel*4+i)
=
                        *(addr_raw+i);
                    }
                }

            /* Process data */
            for(i=0;i<file_header.nb_channel;i++)
                {
                offset=4*i;
                /* Check intensity level */
                if((((int)*(addr_raw+offset) & 0100000)==0)
                    || (((int)*(addr_raw+offset+2) & 0100000)==0))
                    {
                    *(addr_raw+offset)   = -9999.;
                    *(addr_raw+offset+1)= -9999.;
                    *(addr_raw+offset+2)= -9999.;
                    *(addr_raw+offset+3)= -9999.;
                    }
                else
                    {
                    *(addr_raw+offset)  =
                        (float)((int)*(addr_raw+offset) & 07777);
                    *(addr_raw+offset+1)=
                        (float)((int)*(addr_raw+offset+1) & 07777);
                    *(addr_raw+offset+2)=
                        (float)((int)*(addr_raw+offset+2) & 07777);
                    *(addr_raw+offset+3)=
                        (float)((int)*(addr_raw+offset+3) & 07777);

                    /* Correct data */
                    rts_intcor(0,errmat,addr_raw+offset,addr_raw+offset+1);
                    *(addr_raw+offset+1) = -*(addr_raw+offset+1);

rts_intcor(1,errmat,addr_raw+offset+2,addr_raw+offset+3);
                    *(addr_raw+offset+3) = -*(addr_raw+offset+3);
                    }
                }
            while(*mailbox=='\001');
            for(i=0;i<file_header.nb_channel;i++)
                {
                *(data_raw+i*4)= *(addr_raw+i*4);
                *(data_raw+i*4+1)= *(addr_raw+i*4+1);
                *(data_raw+i*4+2)= *(addr_raw+i*4+2);
                *(data_raw+i*4+3)= *(addr_raw+i*4+3);
                }
            /* Compute 3d data */
            if(counter>0)
                for(i=0;i<file_header.nb_channel;i++)
                    {
                    *(addr_3d+i*3)= *(data_3r+i*3);
                    *(addr_3d+i*3+1)= *(data_3r+i*3+1);
                    *(addr_3d+i*3+2)= *(data_3r+i*3+2);
                    }
            *mailbox='\001';

            /* Compute bcs data */
            rts_cptbcs(&file_header,addr_3d,addr_bcs);
            }

        if(file_header.kistler_flag)
            {
            if(buffer)
                for(i=0;i<KSCHAN;i++) *(data_fpr+i) =
                    *(start_fpr+(rew_count%NBSTOR)*KSCHAN+i);
            else
                {
                /* Load BIM (68153) registers */
                drw_aicr = AICR;
                drw_dicr = DICR;
                drw_barx = (padd >> 16) & 0xFF; /* address bits 16-24 */
                drw_bar  = (MBASE+KSBUF) & 0xFFFF;/* address bits  0-15 */
```

```
            /* Load DMAI (68430) registers */
            drw_wcr = FPCHAN; /* set number of words to transfer */
            drw_dcr = DCR;  /* Cycle steal mode */
            drw_ocr = OCR;  /* dev to mem, word transfer */
            drw_divr= DIVR; /* DMA interrupt vector */
            drw_ccr = CCR;  /* Start, enable DMA interrupt */

            /* Load address modifier register */
            drw_amr = AMR;

            /* Set GO,frame start (FNCT1) */
            drw_bcr |= (GO|FNCT1);

            /* Wait for interrupt flag */
            while(!(mvme_reg->mvme_drflg))
                {
                if(mvme_reg->mvme_drerr)
                    {
                    printf("DR11W ERROR INTERRUPT RECEIVED/n");
                    exit(0);
                    }
                }

            /* Clear interrupt flag */
            mvme_reg->mvme_drflg = 0;

            /* Clear channel operation complete flag */
            drw_csr = drw_csr;

            /* Clear GO, frame start (FNCT1) */
            drw_bcr &= ~(GO|FNCT1);

            if(counter<MAXZ)
                for(i=0;i<KSCHAN;i++)
                    file_header.fp_zero[i] +=
(float)(*(data_k+i))/MAXZ.;

            for(i=0;i<KSCHAN;i++)
                {
                *(data_fpr+i) = (float) *(data_k+i);
                /* Save Kistler raw data in ring buffer */
                *(start_fpr+(counter%NBSTOR)*KSCHAN+i) = *(data_fpr+i);
                }
            }
        rts_cptfpd(&file_header, data_fpr, data_fpd);
        /*
        printf("Weight = %5.1f lbs\r",
        sqrt(*(data_fpd)* *(data_fpd)+ *(data_fpd+1)* *(data_fpd+1)+
        *(data_fpd+2)* *(data_fpd+2))/4.448); fflush(stdout);
        */
        }

    /* Display Data & Send Input to Remote Host */
    if(!buffer) counter++;
    if(netw)
        {
        if((n=read(local_socket,&go,sizeof(char))) > 0)
            {
            if(go!=0)
                {
                while(go==1) n=read(local_socket,&go,sizeof(char));
                if(!toggle) rew_count = counter;
                if(go==2)
                    {
                    if(!toggle) toggle=TRUE;
                    rew_count-=2;
                    if(rew_count<0) counter=NBSTOR-1;
                    buffer = TRUE;
                    }
                if(go==3)
                    {
                    if(!toggle) toggle=TRUE;
                    rew_count+=2;
                    buffer = TRUE;
                    }
                }
            else if(toggle)
                {
                counter = 0;
                if(gettimeofday(&tp,NULL)!=0)
                    printf("Error from gettimeofday = %d\n",errno);
                else
                    {
                    start_time = tp.tv_sec;
                    start_utime = tp.tv_usec;
                    }
                toggle = FALSE;
                buffer = FALSE;
                }
            if (write(server_socket,addr_bcs,
                    (file_header.nb_segment*15+3)*sizeof(float)) < 0)
                {
                perror("writing on stream socket");
                exit(0);
                }
            }
        else if(n==0) quit=TRUE;
        }
    else if(counter==1000) quit=TRUE;
    }
*mailbox='\002';
/* Get time */
if(gettimeofday(&tp,NULL)!=0)
    printf("Error from gettimeofday = %d\n",errno);
else
    {
    finish_time = tp.tv_sec;
    finish_utime = tp.tv_usec;
    }
interval=(float)(finish_time-start_time)+
        (float)(finish_utime-start_utime)/1E6;
file_header.frequency = (float)counter/interval;
if(counter>NBSTOR)
    {
    /* Clean up Ring Buffers */
    file_header.nb_frames = NBSTOR;
    offset = counter%NBSTOR;

    /* Selspot Ring Buffer */
    if(file_header.selspot_flag)
        {
        temp = calloc(file_header.nb_channel*4*offset,sizeof(float));
        for(j=0;j<offset; j++)
            for(i=0;i<file_header.nb_channel*4;i++)
                *(temp+j*file_header.nb_channel*4+i) =
                    *(start_raw+j*file_header.nb_channel*4+i);
        for(j=offset;j<NBSTOR; j++)
            for(i=0;i<file_header.nb_channel*4;i++)
                *(start_raw+(j-offset)*file_header.nb_channel*4+i) =
                    *(start_raw+j*file_header.nb_channel*4+i);
        for(j=0;j<offset; j++)
            for(i=0;i<file_header.nb_channel*4;i++)
              *(start_raw+(j+NBSTOR-offset)*file_header.nb_channel*4+i)=
                    *(temp+j*file_header.nb_channel*4+i);
        cfree(temp);
        }

    /* Kistler Ring Buffer */
    if(file_header.kistler_flag)
        {
        temp = calloc(KSCHAN*offset,sizeof(float));
        for(j=0;j<offset; j++)
            for(i=0;i<KSCHAN;i++)
                *(temp+j*KSCHAN+i) = *(start_fpr+j*KSCHAN+i);
        for(j=offset;j<NBSTOR; j++)
            for(i=0;i<KSCHAN;i++)
                *(start_fpr+(j-offset)*KSCHAN+i) =
*(start_fpr+j*KSCHAN+i);
        for(j=0;j<offset; j++)
            for(i=0;i<KSCHAN;i++)
                *(start_fpr+(j+NBSTOR-offset)*KSCHAN+i) =
                    *(temp+j*KSCHAN+i);
        cfree(temp);
        }
    }
else file_header.nb_frames = counter;
printf("Final number of frames sampled is: %d\n",counter);
printf("Time interval: %.2f sec.  The frequency is %.2f Hz.\007\n",
        interval,file_header.frequency);

if(file_header.selspot_flag)
    {
    /* Check raw Selspot data */
    t5s_rawchk(start_raw,&file_header);

    /* write Selspot data to workfile */
    s2bytes =
sizeof(float)*file_header.nb_frames*file_header.nb_channel*NUMCAM*2;

    /* assemble filename path */
    sprintf(filename,"%s/t5w_raw",t5_tree.work);
    if((binfil=creat(filename,0777))== NONE) /* create file */
        {
        printf("*** Warning Error ***: can't open %s\n",filename);
        exit(TRUE);
        }
    write(binfil, &file_header, sizeof(struct t5header));
    elements=FL_HEADSIZE - sizeof(struct t5header);
    extra_space=(char *)calloc(elements,sizeof(char));
    write(binfil,extra_space,elements);
    cfree(extra_space);
    write(binfil, start_raw, s2bytes);
    close(binfil);
    }

if(file_header.kistler_flag)
    {
    /* write Kistler data to workfile */
    ksbytes = sizeof(float)*KSCHAN*file_header.nb_frames;

    /* assemble filename path */
    sprintf(filename,"%s/t5w_fpr",t5_tree.work);
    if((binfil=creat(filename,0777))== NONE) /* create file */
        {
        printf("*** Warning Error ***: can't open %s\n",filename);
        exit(TRUE);
        }
    write(binfil, &file_header, sizeof(struct t5header));
    elements=FL_HEADSIZE - sizeof(struct t5header);
    extra_space=(char *)calloc(elements,sizeof(char));
    write(binfil,extra_space,elements);
    cfree(extra_space);
    write(binfil, start_fpr, ksbytes);
    close(binfil);
    }

t5s_savraw(&file_header,&t5_tree);

if(netw)
    {
    /* Shutdown Network Communication */
    shutdown(server_socket,2);
    close(server_socket);
    shutdown(local_socket,2);
```

```
        close(local_socket);
        }

    /* Free allocated memory space */
    cfree(data_fpr);
    cfree(start_fpr);
    cfree(addr_3d);
    cfree(addr_raw);
    cfree(start_raw);
    cfree(errmat);
    cfree(addr_NET);
    exit();
    }


write_header_to_other_processor(pd,pd_addr)
struct t5header *pd,*pd_addr;
    {
    register i;
    char *pdc, *pd_addrc;

    pdc = (char *)pd;
    pd_addrc = (char *)pd_addr;

    for(i=0;i<sizeof(struct t5header);i++) *(pd_addrc+i) = *(pdc+i);
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rtm_track.c
 *
 *  Description: This is the master main program controlling RT_TRACK.
 *               Execution calls to other programs are made for
 *               real time sampling
 *
 *  Definitions: It is accessed by typing "rtm_track" at the terminal.
 *
 *  Created by:  Patrick J. Lord        1-Aug-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <t5i_global.h>

main()
    {
    int i;
    struct t5tree t5_tree;
    struct t5terminal infterm;
    char command[MAXLINE];
    char *ln1 = "Real-Time-TRACK version 1.0";
    char *ln2 = "Copyright (c) Massachusetts Institute of Technology";
    char *ln3 = "Newman Laboratory for Biomechanics and Human
Rehabilitation";
    char *t5m_loadin =
"/usr2/pat/track/development/rt_track/VME133/rtm_loadin";
    char *t5m_loadss = "/usr/util/track/t5m_loadss";
    char *t5m_csetup = "/usr/util/track/t5m_csetup";
    char *t5m_calibr = "/usr/util/track/t5m_calibr";
    char *rt_sample = "rtm_sample";
    char *t5m_proces = "/usr/util/track/t5m_proces";
    char *t5m_graphs = "/usr/util/track/t5m_graphs";
    char *selspot_host = "busch";
    char quit,sample_selspot,reset,host[7];

    t5s_treein(&t5_tree);
    gethostname(host,7);
    if(strcmp(host,selspot_host)==0) sample_selspot=TRUE;
    else sample_selspot=FALSE;
    reset=FALSE;

    quit=FALSE;
    while(!quit)
        {
        t5s_termin(&infterm);
        printf("%s\n",infterm.clear_scr);
        for(i=0;i<(infterm.nb_columns-strlen(ln1))/2;i++) printf(" ");
        printf("%s\n\n\n",ln1);
        for(i=0;i<(infterm.nb_columns-strlen(ln2))/2;i++) printf(" ");
        printf("%s\n\n",ln2);
        for(i=0;i<(infterm.nb_columns-strlen(ln3))/2;i++) printf(" ");
        printf("%s\n\n\n\n",ln3);

        if(sample_selspot)
            {
            while(t5s_queryn("\nDo you wish to calibrate the camera setup"))
                {
                if(!reset)
                    {
                    printf("\n");
                    printf("Loading Interrupt Service Routines\n");
                    sprintf(command,"%s",t5m_loadin);
                    system(command);
                    printf("\n");
                    printf("Initializing Selspot Interface\n");
                    sprintf(command,"%s",t5m_loadss);
                    system(command);
                    printf("\n");
                    reset=TRUE;
                    }
                sprintf(command,"%s 16",t5m_csetup);
                system(command);
                sprintf(command,"%s",t5m_calibr);
                system(command);
                }
            while(t5s_queryn("\nDo you wish to start real time sampling"))
                {
                if(!reset)
                    {
                    printf("\n");
                    printf("Loading Interrupt Service Routines\n");
                    sprintf(command,"%s",t5m_loadin);
                    system(command);
                    printf("\n");
                    printf("Initializing Selspot Interface\n");
                    sprintf(command,"%s",t5m_loadss);
                    system(command);
                    printf("\n");
                    reset=TRUE;
                    }
                sprintf(command,"%s",rt_sample);
                system(command);
                }
            }
        while(t5s_queryn("\nDo you wish to process data"))
            {
            sprintf(command,"%s",t5m_proces);
            system(command);
            }
        while(t5s_queryn("\nDo you wish to graph data"))
            {
            sprintf(command,"%s",t5m_graphs);
            system(command);
            }
        if(!t5s_queryn("\nDo you wish to run Real-Time-TRACK again"))
quit=TRUE;
        }
    printf("\n");
    exit();
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rtm_vme133.c
 *
 *  Description: This is the program of RT-TRACK that sits on the VME133.
 *               The procedure initializes the interface, allows
 *               for modification of sampling and processing
 *               parameters, and collects Selspot and/or forceplate
 *               data and writes the information to disk.
 *
 *  Created by:         Patrick J. Lord     08-Mar-89
 *
 *  Modified by:
 *
 *  Known Bugs: None
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    <t5i_global.h>

#define OFFSET  0x90000     /* offset to data */

#define WLOOP   10000

void rts_int133();

main()
    {
    struct t5header pd;
    int i, *pd_addr;
    char *abo133, *mailbox;
    float *addr_3d, *data_raw;

    rts_int133();

    abo133 = (char *)(0x004000 + 0x00001D);
    *abo133 = '\000';

    mailbox=(char *)(OFFSET);
    pd_addr=(int *)(mailbox+1);
    *pd_addr=(int) &pd;
    data_raw=(float *)(pd_addr+1);

    *mailbox='\000';
    while(*mailbox=='\000');
    addr_3d=(float *)(data_raw+(4*pd.nb_channel));
```

```
    while(*abol33 != '\001')
        {
        if(*mailbox=='\002')
            {
            mailbox=(char *)(OFFSET);
            pd_addr=(int *)(mailbox+1);
            *pd_addr=(int) &pd;
            data_raw=(float *)(pd_addr+1);
            *mailbox='\000';
            while(*mailbox=='\000');
            addr_3d=(float *)(data_raw+(4*pd.nb_channel));
            }
        while(*mailbox=='\000');
        if(*mailbox=='\001')
            {
            rts_cpt3dv(&pd,data_raw,addr_3d);
            *mailbox='\000';
            }
        }
    exit();
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rts_cpt3dv.c
 *
 *  Description: This subroutine computes from the raw data, the 3-D
 *               xyz coordinates for every channels.
 *
 *  Definitions: It is defined and accessed by:
 *
 *                    float *rts_cpt3dv(pd,data_raw,work_dir)
 *                    struct t5header *pd;
 *                    float *data_raw;
 *                    char  work_dir[];
 *
 *  Created by:  Patrick J. Lord        2-Aug-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                    Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <t5i_global.h>

void rts_cpt3dv(pd,data_raw,data_3dv)
struct t5header *pd;
float *data_raw, *data_3dv;
    {
    register i;
    int   offset;
    float max_skew, temp;
    float error_mag,error_x,error_y,error_z,ax_val,ay_val,az_val,mx_sk;
    float *x1addr,*y1addr,*x2addr,*y2addr;
    float p1[2], p2[2], m[6], t[3], mtm[4], mtt[2];
    float x1, y1, z1, x2, y2, z2, zc[2], dum;

    max_skew=0.;
    x1addr=data_raw;
    y1addr=data_raw+1;
    x2addr=data_raw+2;
    y2addr=data_raw+3;

    /* trnvec is stored as vector in global coordinates from global
       origin to camera origin */
    t[0] = pd->ext_trnvec[1][0] - pd->ext_trnvec[0][0];
    t[1] = pd->ext_trnvec[1][1] - pd->ext_trnvec[0][1];
    t[2] = pd->ext_trnvec[1][2] - pd->ext_trnvec[0][2];

    for(i=0;i<pd->nb_channel;i++)
        {
        offset=3*i;
        if((*x1addr> -9998.)&&(*x2addr> -9998.))
            {
            p1[0]= *x1addr/pd->focal_cam1;
            p1[1]= *y1addr/pd->focal_cam1;
            p2[0]= *x2addr/pd->focal_cam2;
            p2[1]= *y2addr/pd->focal_cam2;

            m[0]= pd->ext_rotmtx[0][0]*p1[0] +
                  pd->ext_rotmtx[0][3]*p1[1] +
                  pd->ext_rotmtx[0][6];

            m[2]= pd->ext_rotmtx[0][1]*p1[0] +
                  pd->ext_rotmtx[0][4]*p1[1] +
                  pd->ext_rotmtx[0][7];

            m[4]= pd->ext_rotmtx[0][2]*p1[0] +
                  pd->ext_rotmtx[0][5]*p1[1] +
                  pd->ext_rotmtx[0][8];

            m[1]= -pd->ext_rotmtx[1][0]*p2[0] -
                   pd->ext_rotmtx[1][3]*p2[1] -
                   pd->ext_rotmtx[1][6];

            m[3]= -pd->ext_rotmtx[1][1]*p2[0] -
                   pd->ext_rotmtx[1][4]*p2[1] -
                   pd->ext_rotmtx[1][7];

            m[5]= -pd->ext_rotmtx[1][2]*p2[0] -
                   pd->ext_rotmtx[1][5]*p2[1] -
                   pd->ext_rotmtx[1][8];

            /* mult m by pseudo inverse of m to get mtm*/
            mtm[0] = m[0]*m[0] + m[2]*m[2] + m[4]*m[4];
            mtm[1] = m[0]*m[1] + m[2]*m[3] + m[4]*m[5];
            mtm[2] = mtm[1];
            mtm[3] = m[1]*m[1] + m[3]*m[3] + m[5]*m[5];

            /* mult t by pseudo inverse of m to get mtt */
            mtt[0] = m[0]*t[0] + m[2]*t[1] + m[4]*t[2];
            mtt[1] = m[1]*t[0] + m[3]*t[1] + m[5]*t[2];

            dum = mtm[2]/mtm[0];
            /* zc is best fit to each camera's z axis */
            zc[1] = (mtt[1] - dum*mtt[0])/(mtm[3] - dum*mtm[1]);
            zc[0] = (mtt[0] - mtm[1]*zc[1])/mtm[0];

            /* Since zc is in meters the average zc gives an indication
            of the spatial resolution attainable for this point.
            The average ration of zc to the camera principal distance
            is the spatial resolution in meters/selspot unit. This
            value multiplied by the selected skew ray cutoff gives
            the skew ray cutoff in meters.*/

            mx_sk = pd->skewray_mx *
                    (zc[0]+zc[1])/(pd->focal_cam1+pd->focal_cam2);


            x1 = zc[0]*m[0] + pd->ext_trnvec[0][0];
            y1 = zc[0]*m[2] + pd->ext_trnvec[0][1];
            z1 = zc[0]*m[4] + pd->ext_trnvec[0][2];
            x2 = zc[1]*(-m[1]) + pd->ext_trnvec[1][0];
            y2 = zc[1]*(-m[3]) + pd->ext_trnvec[1][1];
            z2 = zc[1]*(-m[5]) + pd->ext_trnvec[1][2];

            error_x = x1-x2;
            error_y = y1-y2;
            error_z = z1-z2;
            error_mag=sqrt(error_x*error_x+error_y*error_y+error_z*error_z);
            temp= error_mag/((zc[0]+zc[1])/(pd->focal_cam1+pd->focal_cam2));
            max_skew=MAX(max_skew,temp);
            if(error_mag>=mx_sk)
                {
                pd->sray_badpt++;
                *(data_3dv+offset)= -1000.;
                *(data_3dv+offset+1)= -1000.;
                *(data_3dv+offset+2)= -1000.;
                error_mag=0.;
                }
            else
                {
                ax_val = (x1+x2)*0.5;
                ay_val = (y1+y2)*0.5;
                az_val = (z1+z2)*0.5;
                *(data_3dv+offset)=ax_val-pd->globx_axis;
                *(data_3dv+offset+1)=ay_val-pd->globy_axis;
                *(data_3dv+offset+2)=az_val-pd->globz_axis;
                /*
                printf("%f %f %f\n",ax_val,ay_val,az_val);
                fflush(stdout);
                */
                }
            }
        else
            {
            *(data_3dv+offset)= -1000.;
            *(data_3dv+offset+1)= -1000.;
            *(data_3dv+offset+2)= -1000.;
            error_mag=0.;
            }
        x1addr+=4;
        x2addr+=4;
        y1addr+=4;
        y2addr+=4;
        }
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rts_cptbcs.c
 *
 *  Description: This subroutine computes the BCS coordinates using the
 *               3-D data.  A memory space for the BCS data is allocated
 *               via a calloc call.  At the end of the computation, the
 *               3-D memory space is free, the BCS data saved to a
 *               workfile (t5w_bcs) in the workfile directory, and a
 *               pointer to the first element in the BCS space is
 *               returned.
 *
 *  Definitions: It is defined and accessed by:
 *
 *                    float *rts_cptbcs(pd,data_3dp,work_dir)
 *                    struct t5header *pd;
 *                    float *data_3dp;
 *                    char  work_dir[];
 *
 *  Created by:  Patrick J. Lord        3-Aug-88
 *
 *  Modified by:
 *
```

```
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                        Copyright (C) 1988
 *                Massachusetts Institute of Technology
 *                      Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <t5i_global.h>

#define RAD_TO_DEG  57.29577951

void rts_cptbcs(pd,data_3dp,data_bcs)
struct t5header *pd;
float *data_3dp, *data_bcs;
    {
    register i,j,l;
    char  seg_skip[MAXSEG],chn_skip[MAXLED];
    int   first_chan,last_chan,binfil,elements;
    int   i3, j15;
    float denom;
    float ab2,ac2,ad2,bc2,bd2,cd2;
    float chn_sum,sumhx,sumhy,sumhz,sumx,sumy,sumz,rmhatx[MAXSEG];
    float rmhaty[MAXSEG],rmhatz[MAXSEG];
    float xh[MAXLED],yh[MAXLED],zh[MAXLED],s[4][4],res_abcd[4],*calloc();
    float g0,g1,g2,g3,rmpqh,rmpq,aa,bb,cc,dd,abc,abcd,sabc,costh,theta;

    /*
    l=pd->nb_channel*3;
    */
    for(j=0;j<pd->nb_channel;j++) chn_skip[j]=FALSE;
    for(j=0;j<pd->nb_segment;j++)  /* j is the segment number*/
        {
        j15 = j*15;
        seg_skip[j]=FALSE;
        chn_sum=0.;
        sumhx=0.;
        sumhy=0.;
        sumhz=0.;
        sumx=0.;
        sumy=0.;
        sumz=0.;
        first_chan=MAXLED;
        last_chan=0;
        for(i=0;i<pd->nb_channel;i++) /* i is the channel number */
            {
            if(pd->led_segmnb[i]==j+1) /* for all chan in this seg */
                {
                i3 = i*3;
                first_chan=MIN(i,first_chan);
                last_chan=MAX(i,last_chan);
                if((*(data_3dp+i3)== -1000.)&& /* check for bad 3dp data */
                   (*(data_3dp+i3+1)== -1000.)&&
                   (*(data_3dp+i3+2)== -1000.)) chn_skip[i]=TRUE;
                else
                    {
                    chn_sum++; /* number of valid chan for this seg */
                    sumhx+=pd->led_xcoord[i]; /* sum the segment */
                    sumhy+=pd->led_ycoord[i]; /* marker coordinates */
                    sumhz+=pd->led_zcoord[i];
                    sumx+= *(data_3dp+i3); /* sum the global marker */
                    sumy+= *(data_3dp+i3+1); /* coordinates */
                    sumz+= *(data_3dp+i3+2);
                    }
                }
            }
        if(chn_sum<3.)   /* If <3 valid chan found in this seg */
            {
            seg_skip[j]=TRUE; /* set segment skip flag */
            *(data_bcs+j15)=0.;
            *(data_bcs+j15+1)=0.;
            *(data_bcs+j15+2)=0.;
            rmhatx[j]=0.;
            rmhaty[j]=0.;
            rmhatz[j]=0.;
            }
        else
            {
            rmhatx[j]=sumhx/chn_sum;     /* compute mean of markers */
            rmhaty[j]=sumhy/chn_sum;     /* in segment (bcs) coords */
            rmhatz[j]=sumhz/chn_sum;
            *(data_bcs+j15)=sumx/chn_sum; /* compute mean of markers */
            *(data_bcs+j15+1)=sumy/chn_sum; /* in global coords */
            *(data_bcs+j15+2)=sumz/chn_sum;
            }
        for(i=first_chan;i<=last_chan;i++)
            {
            i3 = i*3;
            if(chn_skip[i])
                {
                xh[i]=0.;
                yh[i]=0.;
                zh[i]=0.;
                *(data_3dp+i3)=0.;
                *(data_3dp+i3+1)=0.;
                *(data_3dp+i3+2)=0.;
                }
            else
                {
                xh[i]=pd->led_xcoord[i]-rmhatx[j]; /* find position vec */
                yh[i]=pd->led_ycoord[i]-rmhaty[j]; /* from mean to each */
                zh[i]=pd->led_zcoord[i]-rmhatz[j]; /* marker in bcs */
                *(data_3dp+i3)-= *(data_bcs+j15); /* find pos vec from */
                *(data_3dp+i3+1)-= *(data_bcs+j15+1); /*mean to global */
                *(data_3dp+i3+2)-= *(data_bcs+j15+2); /* marker */
                rmpqh=sqrt(xh[i]*xh[i]+yh[i]*yh[i]+zh[i]*zh[i]); /*length*/
                rmpq=sqrt(*(data_3dp+i3)**(data_3dp+i3)+ /*length*/
                          *(data_3dp+i3+1)**(data_3dp+i3+1)+
                          *(data_3dp+i3+2)**(data_3dp+i3+2));
                if((rmpq!=0.)&&(rmpqh!=0.))
                    {
                    xh[i]/=rmpqh; /* compute unit vectors */
                    yh[i]/=rmpqh;
                    zh[i]/=rmpqh;
                    *(data_3dp+i3)/=rmpq; /* compute unit vectors */
                    *(data_3dp+i3+1)/=rmpq;
                    *(data_3dp+i3+2)/=rmpq;
                    }
                else
                    {
                    xh[i]=0.;
                    yh[i]=0.;
                    zh[i]=0.;
                    *(data_3dp+i3)=0.;
                    *(data_3dp+i3+1)=0.;
                    *(data_3dp+i3+2)=0.;
                    }
                }
            }
        if(seg_skip[j]) for(i=0;i<15;i++) *(data_bcs+j15+i)= -1000.;
        else
            {
            chn_sum=0.;
            for(i=0;i<4;i++)
                for(l=0;l<4;l++) s[i][l]=0.;
            g0=0.;
            g1=0.;
            g2=0.;
            g3=0.;
            for(i=first_chan;i<=last_chan;i++) /* set up input to */
                if(!chn_skip[i])              /* Schut estimation */
                    {
                    i3 = i*3;
                    chn_sum++;
                    g0+= *(data_3dp+i3+2)*yh[i];
                    g1+= *(data_3dp+i3)*zh[i];
                    g2+= *(data_3dp+i3+1)*xh[i];
                    g3+=(xh[i]+*(data_3dp+i3))*(xh[i]+*(data_3dp+i3))+
                    (yh[i]+*(data_3dp+i3+1))*(yh[i]+*(data_3dp+i3+1))+
                    (zh[i]+*(data_3dp+i3+2))*(zh[i]+*(data_3dp+i3+2));
                    s[0][0]+=xh[i]**(data_3dp+i3);
                    s[1][1]+=yh[i]**(data_3dp+i3+1);
                    s[2][2]+=zh[i]**(data_3dp+i3+2);
                    s[0][3]+=zh[i]**(data_3dp+i3+1);
                    s[1][3]+=xh[i]**(data_3dp+i3+2);
                    s[2][3]+=yh[i]**(data_3dp+i3);
                    }
            s[3][3]=g3-4.*(s[0][0]+s[1][1]+s[2][2]);
            s[0][0]=g3-4.*s[0][0];
            s[1][1]=g3-4.*s[1][1];
            s[2][2]=g3-4.*s[2][2];
            s[0][1]= -2.*(g2+s[2][3]);
            s[0][2]= -2.*(g1+s[1][3]);
            s[1][2]= -2.*(g0+s[0][3]);
            s[0][3]=  2.*(g0-s[0][3]);
            s[1][3]=  2.*(g1-s[1][3]);
            s[2][3]=  2.*(g2-s[2][3]);

            s[1][0]=s[0][1];
            s[2][0]=s[0][2];
            s[3][0]=s[0][3];
            s[2][1]=s[1][2];
            s[3][1]=s[1][3];
            s[3][2]=s[2][3];

            t5s_schuta(s,res_abcd);

            aa=res_abcd[0]*res_abcd[0];
            bb=res_abcd[1]*res_abcd[1];
            cc=res_abcd[2]*res_abcd[2];
            dd=res_abcd[3]*res_abcd[3];
            ab2=2.*res_abcd[0]*res_abcd[1];
            ac2=2.*res_abcd[0]*res_abcd[2];
            ad2=2.*res_abcd[0]*res_abcd[3];
            bc2=2.*res_abcd[1]*res_abcd[2];
            bd2=2.*res_abcd[1]*res_abcd[3];
            cd2=2.*res_abcd[2]*res_abcd[3];
            abc=aa+bb+cc;
            abcd=abc+dd;
            *(data_bcs+j15+6)=(dd+aa-bb-cc)/abcd;
            *(data_bcs+j15+7)=(ab2-cd2)/abcd;
            *(data_bcs+j15+8)=(ac2+bd2)/abcd;
            *(data_bcs+j15+9)=(ab2+cd2)/abcd;
            *(data_bcs+j15+10)=(dd-aa+bb-cc)/abcd;
            *(data_bcs+j15+11)=(bc2-ad2)/abcd;
            *(data_bcs+j15+12)=(ac2-bd2)/abcd;
            *(data_bcs+j15+13)=(bc2+ad2)/abcd;
            *(data_bcs+j15+14)=(dd-aa-bb+cc)/abcd;

            /* compute Euler angles */
            /*
            *(data_bcs+j15+3) =
                      atan2(-*(data_bcs+j15+9), *(data_bcs+j15+10));
            denom = sqrt(*(data_bcs+j15+9)**(data_bcs+j15+9) +
                         *(data_bcs+j15+10)**(data_bcs+j15+10));
            *(data_bcs+j15+4) = atan2(*(data_bcs+j15+11), denom);
            *(data_bcs+j15+5) =
```

```
                    atan2(-*(data_bcs+j15+8), *(data_bcs+j15+14));
            *(data_bcs+j15+3)*= RAD_TO_DEG;
            *(data_bcs+j15+4)*= RAD_TO_DEG;
            *(data_bcs+j15+5)*= RAD_TO_DEG;
            */

            *(data_bcs+j15)-= *(data_bcs+j15+6)*rmhatx[j]+
                              *(data_bcs+j15+9)*rmhaty[j]+
                              *(data_bcs+j15+12)*rmhatz[j];
            *(data_bcs+j15+1)-= *(data_bcs+j15+7)*rmhatx[j]+
                                *(data_bcs+j15+10)*rmhaty[j]+
                                *(data_bcs+j15+13)*rmhatz[j];
            *(data_bcs+j15+2)-= *(data_bcs+j15+8)*rmhatx[j]+
                                *(data_bcs+j15+11)*rmhaty[j]+
                                *(data_bcs+j15+14)*rmhatz[j];
            }
        }
    }

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rts_cptfpd.c
 *
 *  Description: This program reduces the Kistler Forceplate raw data
 *               into 3 forces and 3 moments about the principal axes.
 *
 *  Definitions: It is called and accessed by
 *
 *                      float *rts_cptfpd(pd,data_fpr,work_dir)
 *                      struct t5header *pd;
 *                      float *data_fpr;
 *                      char  work_dir[];
 *
 *  Created by:   Patrick J. Lord           03-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */
#include <t5i_global.h>

rts_cptfpd(pd,data_fpr,data_fpd)
struct t5header *pd;
float *data_fpr, *data_fpd;
    {
    int i;
    float newton_conversion[KSCHAN], temp;
    /*
    float ax, ay, az, rmfz;
    */

    newton_conversion[0] = pd->fpzscl * KSATOD;
    newton_conversion[1] = pd->fpzscl * KSATOD;
    newton_conversion[2] = pd->fpxscl * KSATOD;
    newton_conversion[3] = pd->fpxscl * KSATOD;
    newton_conversion[4] = pd->fpyscl * KSATOD;
    newton_conversion[5] = pd->fpyscl * KSATOD;
    newton_conversion[6] = pd->fpyscl * KSATOD;
    newton_conversion[7] = pd->fpyscl * KSATOD;

    /* Substract zero offsets & convert to Newtons*/
    for(i=0;i<KSCHAN;i++) *(data_fpr+i) =
        (*(data_fpr+i) - pd->fp_zero[i]) * newton_conversion[i];

    *(data_fpd)   = *(data_fpr) + *(data_fpr+1);
    *(data_fpd+1) = *(data_fpr+2) + *(data_fpr+3);
    *(data_fpd+2) = *(data_fpr+4) + *(data_fpr+5) +
                    *(data_fpr+6) + *(data_fpr+7);
    /*
    *(data_fpd+3) = (*(data_fpr+4) + *(data_fpr+5) -
                     *(data_fpr+6) - *(data_fpr+7)) * KSB;
    *(data_fpd+4) = (*(data_fpr+5) - *(data_fpr+4) +
                     *(data_fpr+6) - *(data_fpr+7)) * KSA;
    *(data_fpd+5) = ((*(data_fpr+1) - *(data_fpr)) * KSB) +
                    ((*(data_fpr+2) - *(data_fpr+3)) * KSA);
    ax=ay=0.;
    az=KSAZ;
    if(fabs(*(data_fpd+2)) >= KS_F_MIN)
        {
        ax=((*(data_fpd)*az) - *(data_fpd+4)) / *(data_fpd+2);
        ay=((*(data_fpd+1)*az) + *(data_fpd+3)) / *(data_fpd+2);
        }
    rmfz= *(data_fpd+5) - *(data_fpd+1) * ax + *(data_fpd) * ay;
    */

    /* The Kistler forceplate coordinate system and the TRACK
       coordinate system are not parallel:
                    TRACK           Kistler
                      x               -y
                      y               -z
                      z                x
    */

    temp = *(data_fpd);
    *(data_fpd)   =  - *(data_fpd+1);
    *(data_fpd+1) =  - *(data_fpd+2);
    *(data_fpd+2) =  temp;

    /*
    *(data_fpd+3) =  - ay;
    *(data_fpd+4) =  - rmfz;
    *(data_fpd+5) =  ax;
    */
    }
```

```
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*  rts_int133.s
#*
#*  Description:    This program is designed to be loaded
#*                  on the mvme133 processor and executed
#*                  there.  It serves to field interrupts
#*                  from the various devices associated with
#*                  TRACK.
#*
#*  Definitions: (example call, variable types, return types,
#*                limitations, special environments required)
#*
#*  Created by: Peter K. Mansfield              18-Jul-88
#*
#*  Modified by: Patrick J. Lord                08-Mar-89
#*
#*  Known Bugs:
#*
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*
#*                      Copyright (C) 1988
#*              Massachusetts Institute of Technology
#*                    Cambridge, Massachusetts
#*
#*  This software is subject to change without notice and should not
#*  be construed as a commitment by MIT.  MIT assumes no responsibility
#*  for the use or reliability of its software.
#*
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*/

    MBASE   =   0x100000
    MSGBUF  =   0x004000
    S2FLG   =   MSGBUF + 0x00001A
    DRFLG   =   MSGBUF + 0x00001B
    ABO133  =   MSGBUF + 0x00001D
    S2INT   =   0xCA
    S2INTADD=   S2INT*4
    SYNCINT =   0xCB
    SYNCADD =   SYNCINT*4
    DRINT   =   0xCC
    DRINTADD=   DRINT*4
    DRERR   =   0xCD
    DRERRINT=   DRERR*4
    DRWBCR  =   0xff0161

# Initialization statements
        .text
        .globl  _rts_int133
_rts_int133:
        andw    #0xF8FF,sr          | set processor priority to 0
        movl    #s2isr,S2INTADD     | load Selspot read int vector
        movl    #syncisr,SYNCADD    | load Selspot sync int vector
        movl    #drisr,DRINTADD     | load DR11W DMA interrupt vector
        movl    #errint,DRERRINT    | load DR11W error int vector
        rts

# Selspot read interrupt routine
        .text
        .globl  s2isr
s2isr:  addqb   #1,S2FLG
        rte

# DR11W DMA interrupt routine
        .text
        .globl  drisr
drisr:  addqb   #1,DRFLG
        rte

# Selspot synchronization interrupt routine
        .text
        .globl  syncisr
syncisr:bset    #5,DRWBCR
        movl    #2,d0
delay2: subql   #1,d0
        cmpl    #0,d0
        bne     delay2
        bclr    #5,DRWBCR
        rte

# DR11W error interrupt routine
        .text
        .globl  errint
errint: movl    errstr,a0
        movl    errend,a1
        moveml  a0/a1,sp@-
        trap    #15
        .word   0x21
        rte

errstr: .ascii  "DR11W ERROR!\007"
errend: .byte   0
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*  rts_intcor.c
*
*  Description: This subroutine performs the camera corrections for
*               a given camera, a U & a V coordinates, by using a
*               pointer to the first element of the correction table.
*               The corrected U & V are returned via the use of float
*               pointers passed to the subroutine.
*
*  Definitions: It is defined as:
*
*                     void rts_intcor(camera,errmat,u,v)
*                     int   camera;
*                     float *errmat, *u, *v;
*
*  Created by:  Patrick J. Lord        06-Dec-88
*
*  Modified by:
*
*  Known Bugs:
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*
*                     Copyright (C) 1988
*               Massachusetts Institute of Technology
*                     Cambridge, Massachusetts
*
*  This software is subject to change without notice and should not
*  be construed as a commitment by MIT.  MIT assumes no responsibility
*  for the use or reliability of its software.
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

#include <t5i_global.h>

void rts_intcor(camera,errmat,u,v)
int   camera;
float *errmat, *u, *v;
    {
    register int k;
    int   x_val,y_val,ind_x,ind_y;
    float interp_x,interp_y,interp_xo,interp_yo,interp_coeff[4];
    float correct_x,correct_y,error_x[MAXCAM*2],error_y[MAXCAM*2];

    if((*u<=3992.)&&(*u>=104.)&& (*v<=3992.)&&(*v>=104.))
        {
        ind_x=((int)*u-23)/81;
        x_val=ind_x+1;
        ind_y=((int)*v-23)/81;
        y_val=ind_y+1;

        error_x[0]= *(errmat+(y_val+camera*MCOREC)*MCOREC+x_val);
        error_x[1]= *(errmat+(y_val+camera*MCOREC)*MCOREC+ind_x);
        error_x[2]= *(errmat+(ind_y+camera*MCOREC)*MCOREC+ind_x);
        error_x[3]= *(errmat+(ind_y+camera*MCOREC)*MCOREC+x_val);

        error_y[0]= *(errmat+(y_val+(camera+2)*MCOREC)*MCOREC+x_val);
        error_y[1]= *(errmat+(y_val+(camera+2)*MCOREC)*MCOREC+ind_x);
        error_y[2]= *(errmat+(ind_y+(camera+2)*MCOREC)*MCOREC+ind_x);
        error_y[3]= *(errmat+(ind_y+(camera+2)*MCOREC)*MCOREC+x_val);

        if(error_x[0]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_x[1]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_x[2]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_x[3]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_y[0]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_y[1]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_y[2]<= -99.9){*u= -9999.;*v= -9999.;}
        else if(error_y[3]<= -99.9){*u= -9999.;*v= -9999.;}
        else
            {
            interp_xo=(float)(ind_x*81+23)+40.5;
            interp_yo=(float)(ind_y*81+23)+40.5;

            interp_x=(*u-interp_xo)/40.5;
            interp_y=(*v-interp_yo)/40.5;

            interp_coeff[0]=(1.+interp_x)*(1.+interp_y)/4.;
            interp_coeff[1]=(1.-interp_x)*(1.+interp_y)/4.;
            interp_coeff[2]=(1.-interp_x)*(1.-interp_y)/4.;
            interp_coeff[3]=(1.+interp_x)*(1.-interp_y)/4.;

            correct_x=0.;
            correct_y=0.;
            for(k=0;k<4;k++)
                {
                correct_x+=interp_coeff[k]*error_x[k];
                correct_y+=interp_coeff[k]*error_y[k];
                }

            *u-=correct_x+2048.;
            *v-=correct_y+2048.;
            }
        /*
        *u-=2048.;
        *v-=2048.;
        */
        }
    else {*u= -9999.;*v= -9999.;}
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*  rts_mvmego.c
*
*  Description: This subroutine initiates execution of a program
*               on the mvme133 processor board, running the
*               133bug monitor.  The command "go" is issued
*               to the monitor with a specified address in
*               the processor's local memory map.
*
*  Definitions: (example call, variable types, return types,
*                limitations, special environments required)
*
*  Created by:  Patrick J. Lord                08-Mar-89
*
*  Modified by:
*
*  Known Bugs:
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*
*                     Copyright (C) 1988
*               Massachusetts Institute of Technology
*                     Cambridge, Massachusetts
*
*  This software is subject to change without notice and should not
*  be construed as a commitment by MIT.  MIT assumes no responsibility
*  for the use or reliability of its software.
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
#include    <sgtty.h>

void rts_mvmego(addr)
int   addr;
{
    struct sgttyb ttya;
    char *input = "go 5000";
    char crlf[2];
    int fd;

    crlf[0] = 015;
    crlf[1] = 012;

    fd = open("/dev/ttya",2);

    ioctl(fd,TIOCGETP,&ttya);   /* Get terminal characteristics */
    ttya.sg_flags |= RAW;       /* Put ttya in RAW mode */
    ttya.sg_flags &= ~ECHO;     /* Disable ECHO */
    ioctl(fd,TIOCSETP,&ttya);   /* Set new terminal chars */

    write(fd,crlf,2);       /* write leading <cr><lf> */
    write(fd,input,7);      /* write go command to debug monitor */
    write(fd,crlf,2);       /* write trailing <cr><lf> */

    close(fd);
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*  server_connect.c
*
*  Description:
*
*  Definitions: (example call, variable types, return types,
*                limitations, special environments required)
*
*  Created by: Patrick J. Lord     1-Feb-88
*
*  Modified by:
*
*  Known Bugs:
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*
*                     Copyright (C) 1988
*               Massachusetts Institute of Technology
*                     Cambridge, Massachusetts
*
*  This software is subject to change without notice and should not
*  be construed as a commitment by MIT.  MIT assumes no responsibility
*  for the use or reliability of its software.
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
#include    <t5i_global.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int server_connect(server_name)
char server_name[];
    {
    int sock,length, port;
    struct sockaddr_in sin;
    struct hostent *hp, *gethostbyname();

    /* open socket */
    if ((sock = socket (AF_INET,SOCK_STREAM,0)) < 0)
        {
        perror("opening stream socket");
        exit(0);
        }

    /* initialize socket data structure */
    sin.sin_family = AF_INET;
```

```
hp = gethostbyname(server_name);      /* to get host address */
bcopy (hp->h_addr, &(sin.sin_addr.s_addr), hp->h_length);
printf("Enter port number of socket on %s: ",server_name);
scanf("%d",&port);
sin.sin_port = htons(port);
/*
sin.sin_port = htons(IPPORT_RESERVED+1);
*/

/* connect to remote host */
if (connect(sock,&sin,sizeof(sin)) < 0)
    {
    close(sock);
    perror("connection streams socket");
    exit(0);
    }
return(sock);
}
```

Appendix C

Source Code:
Real-Time Gait Graphic Software

```
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*  Makefile for rt_gait
#*
#*  Description:
#*
#*  Definitions: It is defined by:
#*
#*  Created by:  Patrick J. Lord        11-Apr-89
#*
#*  Modified by:
#*
#*  Known Bugs:
#*
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*
#*                      Copyright (C) 1989
#*               Massachusetts Institute of Technology
#*                    Cambridge, Massachusetts
#*
#*  This software is subject to change without notice and should not
#*  be construed as a commitment by MIT.  MIT assumes no responsibility
#*  for the use or reliability of its software.
#*
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#*

FILES  = rt-track.o local_accept.o serve_connec.o draw_pannels.o \
     genr_pannels.o sort_pannels.o

FILES1 = draw_pannels.o genr_pannels.o sort_pannels.o leg_model.o

FILES2 = rt-screen.o local_accept.o serve_connec.o draw_pannels.o \
     genr_pannels.o sort_pannels.o getjoint.o

FILES3 = rt_gait.o rt_subrou.o local_accept.o serve_connec.o draw_pannels.o
\
     t5s_termin.o t5s_typpar.o genr_pannels.o sort_pannels.o getjoint.o

CFLAGS = -ltermcap -lbsd -ldbm -Zg -Zf -O

all:        rt-track leg_model rt-screen rt_gait

rt-track:   ${FILES}
        cc ${FILES} ${CFLAGS} -o rt-track

leg_model:  ${FILES1}
        cc ${FILES1} ${CFLAGS} -o leg_model

rt-screen:  ${FILES2}
        cc ${FILES2} ${CFLAGS} -o rt-screen

rt_gait:    ${FILES3}
        cc ${FILES3} ${CFLAGS} -o rt_gait

# Make .o files from the .c files
.c.o: $*.c
    @echo "> Compiling" $@ because of change to $?
    $(CC) -c ${CFLAGS} $*.c
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  draw_pannels.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <gl.h>

struct pannel {
    int nb_points;
    int seg_num;
    float points[5][3];
    Object object;
    };

struct entry    {
    float zvalue;
    int order;
    };

struct point    {
    float x,y,z;
    };

void draw_pannels( pannels, xyz_average, zsort, seg_mat, nb_pannels)
struct pannel pannels[];
struct point xyz_average[];
struct entry zsort[];
float seg_mat[4][4][4];
int nb_pannels;
    {
    int i;

    sort_pannels(zsort, xyz_average, nb_pannels);
    for(i=0;i<nb_pannels;i++)
        {
        pushmatrix();
        multmatrix(&seg_mat[pannels[zsort[i].order].seg_num][0][0]);
        callobj(pannels[zsort[i].order].object);
        popmatrix();
        }
    }
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  genr_pannels.c
 *
 *  Description:
 *
```

```
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <math.h>
#include <gl.h>

#define PI          3.141593
#define SEGMENT     4

struct pannel {
    int nb_points;
    int seg_num;
    float points[5][3];
    Object object;
    };

struct entry    {
    float zvalue;
    int order;
    };

struct point    {
    float x,y,z;
    };

void genr_pannels(array_xyz, floor_xyz, joint_xyz,
                  joint_circum, foot_size, pannels, xyz_average)
float array_xyz[SEGMENT][3], floor_xyz[], joint_xyz[3][3];
float joint_circum[], foot_size;
struct pannel pannels[];
struct point xyz_average[];
    {
    int i, j;
    float leg_pts[11][3], joint_radius[3];

    joint_radius[0]= joint_circum[0]/(2. * PI);
    joint_radius[1]= joint_circum[1]/(2. * PI);
    joint_radius[2]= joint_circum[2]/(2. * PI);

    leg_pts[0][0] = joint_xyz[0][0] + joint_radius[0] / 2.;
    leg_pts[0][1] = joint_xyz[0][1] + joint_radius[0] * 2.;
    leg_pts[0][2] = joint_xyz[0][2] + joint_radius[0] / 2.;

    leg_pts[1][0] = joint_xyz[0][0] - joint_radius[0];
    leg_pts[1][1] = joint_xyz[0][1] + joint_radius[0] * 2.;
    leg_pts[1][2] = joint_xyz[0][2] + joint_radius[0] / 2.;

    leg_pts[2][0] = joint_xyz[0][0] - joint_radius[0];
    leg_pts[2][1] = joint_xyz[0][1];
    leg_pts[2][2] = joint_xyz[0][2];

    leg_pts[3][0] = joint_xyz[0][0] + joint_radius[0];
    leg_pts[3][1] = joint_xyz[0][1];
    leg_pts[3][2] = joint_xyz[0][2];

    leg_pts[4][0] = joint_xyz[1][0] + joint_radius[1];
    leg_pts[4][1] = joint_xyz[1][1];
    leg_pts[4][2] = joint_xyz[1][2];

    leg_pts[5][0] = joint_xyz[1][0] - joint_radius[1];
    leg_pts[5][1] = joint_xyz[1][1];
    leg_pts[5][2] = joint_xyz[1][2];

    leg_pts[6][0] = joint_xyz[2][0] - joint_radius[2];
    leg_pts[6][1] = joint_xyz[2][1];
    leg_pts[6][2] = joint_xyz[2][2];

    leg_pts[7][0] = joint_xyz[2][0] + joint_radius[2];
    leg_pts[7][1] = joint_xyz[2][1];
    leg_pts[7][2] = joint_xyz[2][2];

    leg_pts[8][0] = joint_xyz[2][0] + joint_radius[2];
    leg_pts[8][1] = floor_xyz[1];
    leg_pts[8][2] = joint_xyz[2][2];

    leg_pts[9][0] = joint_xyz[2][0] + joint_radius[2] - foot_size;
    leg_pts[9][1] = floor_xyz[1];
    leg_pts[9][2] = joint_xyz[2][2];

    leg_pts[10][0] = joint_xyz[2][0] + joint_radius[2] - foot_size;
    leg_pts[10][1] = floor_xyz[1] + .0381;
    leg_pts[10][2] = joint_xyz[2][2];

    /* Generate the pannel points from this information */
    /* Outside Pelvis  */
    pannels[0].nb_points = 4;
    pannels[0].seg_num = 0;
    for(i=0;i<pannels[0].nb_points;i++)
        for(j=0;j<3;j++)
            pannels[0].points[i][j] = leg_pts[i][j] - array_xyz[0][j];
    /* Front Pelvis */
    pannels[1].nb_points = 4;
    pannels[1].seg_num = 0;
    for(j=0;j<3;j++) pannels[1].points[0][j]=leg_pts[1][j]-array_xyz[0][j];
    for(j=0;j<3;j++) pannels[1].points[1][j]=leg_pts[2][j]-array_xyz[0][j];
    for(j=0;j<3;j++)
        {
        pannels[1].points[2][j]=leg_pts[2][j]-array_xyz[0][j];
        if(j==2) pannels[1].points[2][j] += 4. * joint_radius[0];
        }
    for(j=0;j<3;j++)
        {
        pannels[1].points[3][j]=leg_pts[1][j]-array_xyz[0][j];
        if(j==2) pannels[1].points[3][j] += 3. * joint_radius[0];
        }
    /* Inside Pelvis  */
    pannels[2].nb_points = 4;
    pannels[2].seg_num = 0;
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            {
            pannels[2].points[i][j] = leg_pts[i][j] - array_xyz[0][j];
            if(j==2) pannels[2].points[i][j] += 3. * joint_radius[0];
            }
    for(i=2;i<4;i++)
        for(j=0;j<3;j++)
            {
            pannels[2].points[i][j] = leg_pts[i][j] - array_xyz[0][j];
            if(j==2) pannels[2].points[i][j] += 4. * joint_radius[0];
            }
    /* Back Pelvis  */
    pannels[3].nb_points = 4;
    pannels[3].seg_num = 0;
    for(j=0;j<3;j++) pannels[3].points[0][j]=leg_pts[0][j]-array_xyz[0][j];
    for(j=0;j<3;j++) pannels[3].points[1][j]=leg_pts[3][j]-array_xyz[0][j];
    for(j=0;j<3;j++)
        {
        pannels[3].points[2][j]=leg_pts[3][j]-array_xyz[0][j];
        if(j==2) pannels[3].points[2][j] += 4. * joint_radius[0];
        }
    for(j=0;j<3;j++)
        {
        pannels[3].points[3][j]=leg_pts[0][j]-array_xyz[0][j];
        if(j==2) pannels[3].points[3][j] += 3. * joint_radius[0];
        }

    /* Outside Thigh  */
    pannels[4].nb_points = 4;
    pannels[4].seg_num = 1;
    for(i=0;i<4;i++)
        for(j=0;j<3;j++)
            pannels[4].points[i][j] = leg_pts[i+2][j] - array_xyz[1][j];
    /* Front Thigh */
    pannels[5].nb_points = 4;
    pannels[5].seg_num = 1;
    for(j=0;j<3;j++) pannels[5].points[0][j]=leg_pts[2][j]-array_xyz[1][j];
    for(j=0;j<3;j++) pannels[5].points[1][j]=leg_pts[5][j]-array_xyz[1][j];
    for(j=0;j<3;j++)
        {
        pannels[5].points[2][j]=leg_pts[5][j]-array_xyz[1][j];
        if(j==2) pannels[5].points[2][j] += 2. * joint_radius[1];
        }
    for(j=0;j<3;j++)
        {
        pannels[5].points[3][j]=leg_pts[2][j]-array_xyz[1][j];
        if(j==2) pannels[5].points[3][j] += 2. * joint_radius[0];
        }
    /* Inside Thigh  */
    pannels[6].nb_points = 4;
    pannels[6].seg_num = 1;
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            {
            pannels[6].points[i][j] = leg_pts[i+2][j] - array_xyz[1][j];
            if(j==2) pannels[6].points[i][j] += 2. * joint_radius[0];
            }
    for(i=2;i<4;i++)
        for(j=0;j<3;j++)
            {
            pannels[6].points[i][j] = leg_pts[i+2][j] - array_xyz[1][j];
            if(j==2) pannels[6].points[i][j] += 2. * joint_radius[1];
            }
    /* Back Thigh  */
    pannels[7].nb_points = 4;
    pannels[7].seg_num = 1;
    for(j=0;j<3;j++) pannels[7].points[0][j]=leg_pts[3][j]-array_xyz[1][j];
    for(j=0;j<3;j++) pannels[7].points[1][j]=leg_pts[4][j]-array_xyz[1][j];
    for(j=0;j<3;j++)
        {
        pannels[7].points[2][j]=leg_pts[4][j]-array_xyz[1][j];
        if(j==2) pannels[7].points[2][j] += 2. * joint_radius[1];
        }
    for(j=0;j<3;j++)
        {
        pannels[7].points[3][j]=leg_pts[3][j]-array_xyz[1][j];
        if(j==2) pannels[7].points[3][j] += 2. * joint_radius[0];
        }

    /* Outside Shank  */
    pannels[8].nb_points = 4;
    pannels[8].seg_num = 2;
    for(i=0;i<4;i++)
        for(j=0;j<3;j++)
            pannels[8].points[i][j] = leg_pts[i+4][j] - array_xyz[2][j];
    /* Front Shank */
    pannels[9].nb_points = 4;
```

```
    pannels[9].seg_num = 2;
    for(j=0;j<3;j++) pannels[9].points[0][j]=leg_pts[5][j]-array_xyz[2][j];
    for(j=0;j<3;j++) pannels[9].points[1][j]=leg_pts[6][j]-array_xyz[2][j];
    for(j=0;j<3;j++)
        {
        pannels[9].points[2][j]=leg_pts[6][j]-array_xyz[2][j];
        if(j==2) pannels[9].points[2][j] += 2. * joint_radius[2];
        }
    for(j=0;j<3;j++)
        {
        pannels[9].points[3][j]=leg_pts[5][j]-array_xyz[2][j];
        if(j==2) pannels[9].points[3][j] += 2. * joint_radius[1];
        }
    /* Inside Shank  */
    pannels[10].nb_points = 4;
    pannels[10].seg_num = 2;
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            {
            pannels[10].points[i][j] = leg_pts[i+4][j] - array_xyz[2][j];
            if(j==2) pannels[10].points[i][j] += 2. * joint_radius[1];
            }
    for(i=2;i<4;i++)
        for(j=0;j<3;j++)
            {
            pannels[10].points[i][j] = leg_pts[i+4][j] - array_xyz[2][j];
            if(j==2) pannels[10].points[i][j] += 2. * joint_radius[2];
            }
    /* Back Shank  */
    pannels[11].nb_points = 4;
    pannels[11].seg_num = 2;
    for(j=0;j<3;j++) pannels[11].points[0][j]=leg_pts[4][j]-array_xyz[2][j];
    for(j=0;j<3;j++) pannels[11].points[1][j]=leg_pts[7][j]-array_xyz[2][j];
    for(j=0;j<3;j++)
        {
        pannels[11].points[2][j]=leg_pts[7][j]-array_xyz[2][j];
        if(j==2) pannels[11].points[2][j] += 2. * joint_radius[2];
        }
    for(j=0;j<3;j++)
        {
        pannels[11].points[3][j]=leg_pts[4][j]-array_xyz[2][j];
        if(j==2) pannels[11].points[3][j] += 2. * joint_radius[1];
        }

    /* Outside Foot  */
    pannels[12].nb_points = 5;
    pannels[12].seg_num = 3;
    for(i=0;i<5;i++)
        for(j=0;j<3;j++)
            pannels[12].points[i][j] = leg_pts[i+6][j] - array_xyz[3][j];
    /* Front Foot */
    pannels[13].nb_points = 4;
    pannels[13].seg_num = 3;
    for(j=0;j<3;j++) pannels[13].points[0][j]=leg_pts[10][j]-
array_xyz[3][j];
    for(j=0;j<3;j++) pannels[13].points[1][j]=leg_pts[9][j]-array_xyz[3][j];
    for(j=0;j<3;j++)
        {
        pannels[13].points[2][j]=leg_pts[9][j]-array_xyz[3][j];
        if(j==2) pannels[13].points[2][j] += 2. * joint_radius[2];
        }
    for(j=0;j<3;j++)
        {
        pannels[13].points[3][j]=leg_pts[10][j]-array_xyz[3][j];
        if(j==2) pannels[13].points[3][j] += 2. * joint_radius[2];
        }
    /* Inside Foot  */
    pannels[14].nb_points = 5;
    pannels[14].seg_num = 3;
    for(i=0;i<2;i++)
        for(j=0;j<3;j++)
            {
            pannels[14].points[i][j] = leg_pts[i+6][j] - array_xyz[3][j];
            if(j==2) pannels[14].points[i][j] += 2. * joint_radius[2];
            }
    for(i=2;i<5;i++)
        for(j=0;j<3;j++)
            {
            pannels[14].points[i][j] = leg_pts[i+6][j] - array_xyz[3][j];
            if(j==2) pannels[14].points[i][j] += 2. * joint_radius[2];
            }
    /* Back Foot  */
    pannels[15].nb_points = 4;
    pannels[15].seg_num = 3;
    for(j=0;j<3;j++) pannels[15].points[0][j]=leg_pts[7][j]-array_xyz[3][j];
    for(j=0;j<3;j++) pannels[15].points[1][j]=leg_pts[8][j]-array_xyz[3][j];
    for(j=0;j<3;j++)
        {
        pannels[15].points[2][j]=leg_pts[8][j]-array_xyz[3][j];
        if(j==2) pannels[15].points[2][j] += 2. * joint_radius[2];
        }
    for(j=0;j<3;j++)
        {
        pannels[15].points[3][j]=leg_pts[7][j]-array_xyz[3][j];
        if(j==2) pannels[15].points[3][j] += 2. * joint_radius[2];
        }
    /* Top Foot  */
    pannels[16].nb_points = 4;
    pannels[16].seg_num = 3;
    for(j=0;j<3;j++) pannels[16].points[0][j]=leg_pts[6][j]-array_xyz[3][j];
    for(j=0;j<3;j++) pannels[16].points[1][j]=leg_pts[10][j]-
array_xyz[3][j];
    for(j=0;j<3;j++)
        {
        pannels[16].points[2][j]=leg_pts[10][j]-array_xyz[3][j];
        if(j==2) pannels[16].points[2][j] += 2. * joint_radius[2];
        }
    for(j=0;j<3;j++)
        {
        pannels[16].points[3][j]=leg_pts[6][j]-array_xyz[3][j];
        if(j==2) pannels[16].points[3][j] += 2. * joint_radius[2];
        }

    for(i=0;i<17;i++)
        {
        /* Compute average positions */
        xyz_average[i].x = 0.;
        xyz_average[i].y = 0.;
        xyz_average[i].z = 0.;
        for(j=0;j<pannels[i].nb_points;j++)
            {
            xyz_average[i].x += pannels[i].points[j][0];
            xyz_average[i].y += pannels[i].points[j][1];
            xyz_average[i].z += pannels[i].points[j][2];
            }
        xyz_average[i].x /= (float) pannels[i].nb_points;
        xyz_average[i].y /= (float) pannels[i].nb_points;
        xyz_average[i].z /= (float) pannels[i].nb_points;

        /* Make pannel Objects */
        delobj(pannels[i].object);
        makeobj(pannels[i].object = genobj());
        if((i % 4) == 0)
            {
            if(i == 16) color(GREEN);
            else color(RED);
            }
        else if((i % 4) == 1) color(GREEN);
        else if((i % 4) == 2) color(YELLOW);
        else if((i % 4) == 3) color(CYAN);
        polf(pannels[i].nb_points, pannels[i].points);
        color(BLACK);
        poly(pannels[i].nb_points, pannels[i].points);
        closeobj();
        }
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  getjoint.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                        Copyright (C) 1989
 *                Massachusetts Institute of Technology
 *                      Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <math.h>

#define PI  3.141592

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    getjoint
    is a subroutine to calculate the flexion, rotation,
    and abduction at a given joint.
*/
getjoint(seg_mat, joint_angles, joint)
float seg_mat[4][4][4], joint_angles[3][3];
int joint;
    {
    int i, j, angle_scale;
    float distal[3][3], proximal[3][3];
    float F_vector[3], numerator, denominator;

    angle_scale = 1;

    for(i=0;i<=2;i++)
        for(j=0;j<=2;j++)
            {
            distal[i][j] = seg_mat[joint][j][i];
            proximal[i][j] = seg_mat[joint+1][j][i];
            }
    F_vector[0] = distal[1][1] * proximal[2][2] - proximal[1][2] *
distal[2][1];
    F_vector[1] = distal[2][1] * proximal[0][2] - proximal[2][2] *
distal[0][1];
    F_vector[2] = distal[0][1] * proximal[1][2] - proximal[0][2] *
distal[1][1];

    numerator=   F_vector[0]*proximal[0][1] + F_vector[1]*proximal[1][1] +
                 F_vector[2]*proximal[2][1];
    denominator= F_vector[0]*proximal[0][0] + F_vector[1]*proximal[1][0] +
                 F_vector[2]*proximal[2][0];
    joint_angles[joint][0] =
atan2(numerator,denominator)*180.0/PI*angle_scale;
    if(joint != 1) joint_angles[joint][0] *= -1;
```

```
    numerator=   proximal[0][2]*distal[0][1] + proximal[1][2]*distal[1][1] +
                 proximal[2][2]*distal[2][1];
    denominator= proximal[0][2]*distal[0][2] + proximal[1][2]*distal[1][2] +
                 proximal[2][2]*distal[2][2];
    joint_angles[joint][1] = atan2(numerator,denominator)*180.0/PI;

    numerator=  -F_vector[0]*distal[0][2] - F_vector[1]*distal[1][2] -
                 F_vector[2]*distal[2][2];
    denominator= F_vector[0]*distal[0][0] + F_vector[1]*distal[1][0] +
                 F_vector[2]*distal[2][0];
    joint_angles[joint][2] =
atan2(numerator,denominator)*180.0/PI*angle_scale;
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  local_accept.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <bsd/netdb.h>
#include <stdio.h>
#include <math.h>

int local_accept()
    {
    int sock,length;
    struct sockaddr_in sin;
    int msgsock;

    /* create a socket */
    if ((sock = socket (AF_INET,SOCK_STREAM,0)) < 0)
        {
        perror("opening stream socket");
        exit(0);
        }

    /* initialize socket data structure */
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = 0;
    /*
    sin.sin_port = htons(IPPORT_RESERVED + 1);
    /*

    /* bind socket data structure to this socket */
    if (bind (sock,&sin,sizeof(sin)))
        {
        perror("binding stream socket");
        }

    /* getsockname fills in the socket structure with information, such as
       the port number assigned to this socket */

    length = sizeof(sin);
    if (getsockname (sock,&sin,&length))
        {
        perror("getting socket name");
        exit(0);
        }
    printf ("Socket has port# %d\n",htons(sin.sin_port));

    /* prepare socket queue for connection requests and accept
       connections */
    listen(sock,5);
    if ((msgsock = accept(sock,0,0)) <= 0) {
        perror("accept on socket");
        exit(0);
        }
    return(msgsock);
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rt_gait.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Feb-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include "t5i_global.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <bsd/netdb.h>

#include <gl.h>
#include <device.h>


#define CUBE_SIZE   .05
#define GREY    8

struct pannel {
    int nb_points;
    int seg_num;
    float points[5][3];
    Object object;
    };

struct entry    {
        float zvalue;
        int order;
        };

struct point    {
    float x,y,z;
    };

struct rotation    {
        Angle rot[2];
        char axis, update;
        };
Object GRID;

main()
    {
    int server_socket, local_socket, counter;
    float *addr_bcs, side;
    char draw_objt, draw_strd, draw_axes, draw_limb, zero_matr, hip, knee;
    char ankle, kistler;
    Object global_axes, cube, std_array, local_axes, floor, force_plate;
    float seg_mat[4][4][4], z_seg_mat[4][3][3], joint_angles[3][3],
center[2];
    struct rotation rot_array[3];
    struct pannel pannels[17];
    struct point xyz_average[17];
    struct entry zsort[17];
    int segment;
    short qchar;
    struct t5header file_header;
    char go;
    int i;

    for(i=0;i<3;i++)
        {
        rot_array[i].rot[0] = 0;
        rot_array[i].rot[1] = 0;
        rot_array[i].update = FALSE;
        }
    rot_array[0].axis = 'x';
    rot_array[1].axis = 'y';
    rot_array[2].axis = 'z';

    draw_objt = FALSE;
    draw_axes = FALSE;
    draw_strd = TRUE;
    draw_limb = FALSE;
    zero_matr = FALSE;
    hip = knee = ankle = kistler = FALSE;
    counter=0;
    side = 1.5;
    center[0] = 0.;
    center[1] = 0.;

    initialize_network(&local_socket, &server_socket);
    if(write(server_socket,&go,sizeof(char)) < 0)
        {
        perror("writing on stream socket");
        exit(0);
        }
    if(read(local_socket,&file_header,sizeof(struct t5header)) < 0)
        {
        perror("reading on stream socket");
        exit(0);
        }
    t5s_typpar(&file_header,stdout);

    printf("Check Parameters, then type Enter to start process: ");
    fflush(stdout);
```

```
    fflush(stdin);
    scanf("%c",&go);
    go = 0;

    if(write(server_socket,&go,sizeof(char)) < 0)
        {
        perror("writing on stream socket");
        exit(0);
        }

    segment = file_header.nb_segment;
    addr_bcs = (float *)calloc(segment*15+3,sizeof(float));
    initialize_graphics();
    make_std_array(&std_array);
    make_cube(&cube);
    make_local_axes(&local_axes);
    make_global_axes(&global_axes);
    make_force_plate(&force_plate);
    make_floor(&floor);
    makeobj(GRID=genobj());
    color(BLUE);
    rect( 0., -90., 35., 90.);
    for(i=0;i<13;i++)
        {
        move2(  0., -90.+(float)i*15);
        draw2(  35., -90.+(float)i*15);
        }
    color(GREEN);
    move2( 0., -90.);
    draw2( 0.,  90.);
    move2( -5., 0.);
    draw2(  35., 0.);
    for(i=0;i<13;i++)
        {
        move2( -2., -90.+(float)i*15);
        draw2(  2., -90.+(float)i*15);
        }
    closeobj();
    qchar = NULL;
    while (qchar != '\r')
        {
        get_track_information(go,zero_matr, local_socket, server_socket,
                              seg_mat, z_seg_mat, addr_bcs, segment);
        clean_screen();
        make_window(0,1023,0,767,BLUE,WHITE);
        ortho( center[0]+side, center[0]-side,
               center[1]-(side/5.), center[1]+side*1.3, -side, side);
        draw_space(global_axes, cube, std_array, local_axes, floor,
                   force_plate, draw_objt, draw_strd, draw_axes, draw_limb,
                   seg_mat, rot_array, pannels, xyz_average, segment);
        draw_text(counter, go);
        if(hip)
            {
            make_window(0,255,512,767,BLACK,WHITE);
            draw_graph(seg_mat, joint_angles, 0,"Hip:","Degrees",
                   "Flex,","Abduct,","Rotat.",90.);
            }
        if(knee)
            {
            make_window(256,511,512,767,BLACK,WHITE);
            draw_graph(seg_mat, joint_angles, 1,"Knee:","Degrees",
                   "Flex,","Abduct,","Rotat.",90.);
            }
        if(ankle)
            {
            make_window(512,767,512,767,BLACK,WHITE);
            draw_graph(seg_mat, joint_angles, 2,"Ankle:","Degrees",
                   "Flex,","Abduct,","Rotat.",90.);
            }
        if(kistler)
            {
            make_window(768,1023,512,767,BLACK,WHITE);
            bar_graph(addr_bcs+segment*15, "Kistler:", "Newtons",
                   "    X,","    Y,","  Z.",1200.);
            }
        swapbuffers();
        counter++;
        processinput(&qchar, &zero_matr, &draw_objt, &draw_strd, &draw_axes,
                     &draw_limb, &hip, &knee, &ankle, &kistler, &go, &side,
                     seg_mat, z_seg_mat, center, rot_array, pannels,
                     xyz_average, segment);
        }
    close_graphics();
    close_network(addr_bcs, local_socket, server_socket);
    }

processinput(qchar, zero_matr, draw_objt, draw_strd, draw_axes, draw_limb,
             hip, knee, ankle, kistler, go, side, seg_mat, z_seg_mat,
center,
             rot_array, pannels, xyz_average, segment)
short *qchar;
char *zero_matr, *draw_objt, *draw_strd, *draw_axes, *draw_limb, *hip;
char *knee, *ankle, *kistler, *go;
float *side, seg_mat[4][4][4], z_seg_mat[4][3][3], center[];
struct rotation rot_array[];
struct pannel pannels[];
struct point xyz_average[];
int segment;
    {
    int foo, i,j,k;
    float joint_xyz[3][3], foot_size, array_xyz[4][3], floor_xyz[3];
    float joint_circum[3];
    struct rotation temp_rot;
    char rot_axis, junk;

    if((foo=qtest()) == KEYBD)
        {
        qread(qchar);
        if(*qchar=='o') *draw_objt = !*draw_objt;
        else if(*qchar=='s') *draw_strd = !*draw_strd;
        else if(*qchar=='a') *draw_axes = !*draw_axes;
        else if(*qchar=='i') center[1]+=.1* *side;
        else if(*qchar=='m') center[1]-=.1* *side;
        else if(*qchar=='j') center[0]-=.1* *side;
        else if(*qchar=='k') center[0]+=.1* *side;
        else if(*qchar=='w')
            {
            if(*go==0) *go = 1;
            else *go = 0;
            }
        else if(*qchar==',') *go = 2;
        else if(*qchar=='.') *go = 3;
        else if(*qchar=='1') *hip = !*hip;
        else if(*qchar=='2') *knee = !*knee;
        else if(*qchar=='3') *ankle = !*ankle;
        else if(*qchar=='4') *kistler = !*kistler;
        else if(*qchar=='f')
            {
            *hip = !*hip;
            *knee = !*knee;
            *ankle = !*ankle;
            *kistler = !*kistler;
            }
        else if(*qchar=='0')
            {
            *zero_matr = !*zero_matr;
            for(i=0;i<segment;i++)
                for(j=0;j<3;j++)
                    for(k=0;k<3;k++)
                        z_seg_mat[i][j][k] = seg_mat[i][k][j];
            }
        else if(*qchar=='x') *side *= 0.9;
        else if(*qchar=='z') *side *= 1.1;
        else if(*qchar=='r')
            {
            *side = 1.5;
            center[0] = 0.;
            center[1] = 0.;
            for(i=0;i<3;i++)
                {
                rot_array[i].rot[0] = 0;
                rot_array[i].rot[1] = 0;
                rot_array[i].update = FALSE;
                }
            rot_array[0].axis = 'x';
            rot_array[1].axis = 'y';
            rot_array[2].axis = 'z';
            }
        else if(*qchar=='l')
            {
            *draw_limb = !*draw_limb;

            /* get initial array xyz */
            for(i=0;i<segment;i++)
                for(j=0;j<3;j++) array_xyz[i][j] = seg_mat[i][3][j];

            /* get initial floor_xyz */
            floor_xyz[0] =  0.;
            floor_xyz[1] =  0.;
            floor_xyz[2] =  0.;

            /* get initial joint_xyz */
            joint_xyz[0][0] =  0.;
            joint_xyz[0][1] = .9906;
            joint_xyz[0][2] =  0.;
            joint_xyz[1][0] = -0.020;
            joint_xyz[1][1] =  .5334;
            joint_xyz[1][2] =  .0300;
            joint_xyz[2][0] =  0.;
            joint_xyz[2][1] =  .0762;
            joint_xyz[2][2] =  .0508;

            /* get joint radii */
            joint_circum[0] = .5080;
            joint_circum[1] = .3810;
            joint_circum[2] = .2540;

            /* get foot_size */
            foot_size = .2286;

            genr_pannels(array_xyz, floor_xyz, joint_xyz,
                         joint_circum, foot_size, pannels, xyz_average);
            }
        }
    if(foo) qreset();

    if(getbutton(LEFTMOUSE)) rot_axis = 'x';
    else if(getbutton(MIDDLEMOUSE)) rot_axis = 'y';
    else if(getbutton(RIGHTMOUSE)) rot_axis = 'z';
    else rot_axis = NULL;
    i=0;
    if(rot_axis != NULL)
        {
        while(rot_array[i].axis != rot_axis) i++;
        if(!rot_array[i].update)
            {
            setvaluator( MOUSEX, rot_array[i].rot[1], -3600, 3600);
            rot_array[i].update = !rot_array[i].update;
            }
        rot_array[i].rot[0] = getvaluator(MOUSEX);
        }
    else
        {
        while(!rot_array[i].update && i<3) i++;
        if(i<3)
            {
```

```
            rot_array[i].update = !rot_array[i].update;
            rot_array[i].rot[1] = rot_array[i].rot[0];
            }
        }
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  rt_subrou.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <bsd/netdb.h>
#include <stdio.h>
#include <math.h>

#include <gl.h>
#include <device.h>


#define PI  3.141592
#define CUBE_SIZE   .05
#define GREY    8

struct pannel {
    int nb_points;
    int seg_num;
    float points[5][3];
    Object object;
    };

struct entry    {
        float zvalue;
        int order;
        };

struct point    {
    float x,y,z;
    };

struct rotation    {
        Angle rot[2];
        char axis, update;
        };
Object GRID;

make_window( xmin, xmax, ymin, ymax, back_color, frame_color)
int xmin, xmax, ymin, ymax;
Colorindex back_color, frame_color;
    {
    viewport( xmin, xmax, ymin, ymax);
    color(frame_color);
    clear();
    viewport( xmin+2, xmax-2, ymin+2, ymax-2);
    color(back_color);
    clear();
    }

clean_screen()
    {
    viewport(0,1023,0,767);
    color(BLACK);
    clear();
    }

close_network(addr_bcs, local_socket, server_socket)
float *addr_bcs;
int local_socket, server_socket;
    {
    cfree(addr_bcs);
    shutdown(server_socket,2);
    close(server_socket);
    shutdown(local_socket,2);
    close(local_socket);
    }

close_graphics()
    {
    greset();
    qexit();
    exit();
    }

initialize_network(local_socket, server_socket)
int *local_socket, *server_socket;
    {
    *local_socket = local_accept();
    *server_socket = server_connect(*local_socket);
    printf("\nBusch - Schlitz communication active\n\n"); fflush(stdout);
    }

get_track_information(go, zero_matr, local_socket, server_socket,
                      seg_mat,z_seg_mat, addr_bcs, segment)
char go, zero_matr;
int local_socket, server_socket;
float seg_mat[4][4][4], z_seg_mat[4][3][3], *addr_bcs;
int segment;
    {
    int i, j, k, offset;
    float temp_mat[3][3];

    if(write(server_socket,&go,sizeof(char)) < 0)
        {
        perror("writing on stream socket");
        exit(0);
        }
    if(go!=1)
        {
        if(read(local_socket,addr_bcs,(segment*15+3)*sizeof(float)) < 0)
            {
            perror("reading on stream socket");
            exit(0);
            }
        }
    for(i=0;i<segment;i++)
        {
        offset=15*i;
        /*
        if( *(addr_bcs+offset)!= -1000.)
        */
            {

            seg_mat[i][0][3] = 0.;
            seg_mat[i][1][3] = 0.;
            seg_mat[i][2][3] = 0.;
            seg_mat[i][3][3] = 1.;

            seg_mat[i][3][0] =  *(addr_bcs + offset);
            seg_mat[i][3][1] =  *(addr_bcs + offset + 1);
            seg_mat[i][3][2] =  *(addr_bcs + offset + 2);

            if(zero_matr)
                {
                mult_mat(&z_seg_mat[i][0][0],
                         addr_bcs + offset + 6, temp_mat, 3, 3, 3);
                for(j=0;j<3;j++)
                    for(k=0;k<3;k++)
                        seg_mat[i][j][k] = temp_mat[j][k];
                }
            else
                {
                seg_mat[i][0][0] = *(addr_bcs + offset + 6);
                seg_mat[i][0][1] = *(addr_bcs + offset + 7);
                seg_mat[i][0][2] = *(addr_bcs + offset + 8);
                seg_mat[i][1][0] = *(addr_bcs + offset + 9);
                seg_mat[i][1][1] = *(addr_bcs + offset + 10);
                seg_mat[i][1][2] = *(addr_bcs + offset + 11);
                seg_mat[i][2][0] = *(addr_bcs + offset + 12);
                seg_mat[i][2][1] = *(addr_bcs + offset + 13);
                seg_mat[i][2][2] = *(addr_bcs + offset + 14);
                }
            }
        }
    }

initialize_graphics()
    {
    ginit();
    greset();
    mapcolor(GREY,127,127,127);
    doublebuffer();
    gconfig();

    qdevice(KEYBD);
    }

draw_space(global_axes, cube, std_array, local_axes,
           floor, force_plate, draw_objt, draw_strd, draw_axes,
           draw_limb, seg_mat, rot_array, pannels, xyz_average, segment)
Object global_axes, cube, std_array, local_axes, floor, force_plate;
char draw_objt, draw_strd, draw_axes, draw_limb;
float seg_mat[4][4][4];
struct rotation rot_array[];
struct pannel pannels[];
struct point xyz_average[];
int segment;
    {
    int i,j;
    struct entry zsort[17];

    pushmatrix();
    for(i=0;i<3;i++) rotate( rot_array[i].rot[0], rot_array[i].axis);
    callobj(floor);
    callobj(force_plate);
    callobj(global_axes);
    if(draw_limb) draw_pannels( pannels, xyz_average, zsort, seg_mat,  17);
    for(i=0;i<segment;i++)
    if(seg_mat[i][0][0]!= -1000.)
        {
        pushmatrix();
```

```
        multmatrix(&seg_mat[i][0][0]);
        if(draw_objt)
            {
            backface(1);
            callobj(cube);
            backface(0);
            }
        if(draw_strd) callobj(std_array);
        if(draw_axes) callobj(local_axes);
        popmatrix();
        }
    popmatrix();
    }

draw_text(counter, go)
int counter;
char go;
    {
    char line[80];

    ortho2( 0., 69., 0., 15.);
    sprintf(line,"Real Time TRACK version 1.0:");
    color(WHITE);
    cmov2i(1,1);
    charstr(line);
    if(go==2)
        {
        sprintf(line,"REWIND MODE");
        color(RED);
        }
    else if (go==3)
        {
        sprintf(line,"FORWARD MODE");
        color(GREEN);
        }
    else
        {
        sprintf(line,"Frame # %d", counter);
        color(WHITE);
        }
    cmov2i(20,1);
    charstr(line);
    }

draw_graph(seg_mat, joint_angles, joint, title, value_type, x1, x2, x3,
maxy)
float seg_mat[4][4][4], joint_angles[3][3];
int joint;
char title[], value_type[], x1[], x2[], x3[];
float maxy;
    {
    getjoint(seg_mat, joint_angles, joint);
    bar_graph(&joint_angles[joint][0], title, value_type, x1, x2, x3, maxy);
    }

make_std_array(std_array)
Object *std_array;
    {
    /* generate standard array object */
    makeobj(*std_array=genobj());
    color(BLACK);
    rectf( -.05, -.009, .05, .009);
    rectf( -.009, -.05, .009, .05);
    move( 0., 0., 0.);
    draw( 0., 0., 0.05);
    arcf( -.05, 0., .009, 900, 2700);
    arcf(  .05, 0., .009, 2700, 900);
    arcf( 0., -0.05, .009, 1800, 0);
    arcf( 0.,  0.05, .009, 0, 1800);

    color(YELLOW);
    circf( -0.0532,  0., .0016);
    circf( -0.0484, -0.0027, .0016);
    circf( -0.0484,  0.0027, .0016);

    circf(  0.0532,  0., .0016);
    circf(  0.0484, -0.0027, .0016);
    circf(  0.0484,  0.0027, .0016);

    circf(  0., 0.0532, .0016);
    circf( -0.0027,  0.0484, .0016);
    circf(  0.0027,  0.0484, .0016);

    circf(  0., -0.0532, .0016);
    circf( -0.0027, -0.0484, .0016);
    circf(  0.0027, -0.0484, .0016);
    closeobj();
    }

make_cube(cube)
Object *cube;
    {
    makeobj(*cube = genobj());
    /* back face */
    pushmatrix();
    translate(0.0, 0.0, 2*CUBE_SIZE);
    color(WHITE);
    rectf(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();

    /* right face */
    pushmatrix();
    translate(CUBE_SIZE, 0.0, 0.0);
    rotate(900, 'y');
    color(GREEN);
    rectf(-2*CUBE_SIZE,-CUBE_SIZE,0.,CUBE_SIZE);
    popmatrix();

    /* front face */
    pushmatrix();
    translate(0.0, 0.0, 0.0);
    rotate(1800, 'y');
    color(RED);
    rectf(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();

    /* left face */
    pushmatrix();
    translate(-CUBE_SIZE, 0.0, 0.0);
    rotate(-900, 'y');
    color(CYAN);
    rectf(0.,-CUBE_SIZE, 2*CUBE_SIZE,CUBE_SIZE);
    popmatrix();

    /* top face */
    pushmatrix();
    translate(0.0, CUBE_SIZE, 0.0);
    rotate(-900, 'x');
    color(MAGENTA);
    rectf(-CUBE_SIZE,-2*CUBE_SIZE,CUBE_SIZE,0.);
    popmatrix();

    /* bottom face */
    pushmatrix();
    translate(0.0, -CUBE_SIZE, 0.0);
    rotate(900, 'x');
    color(YELLOW);
    rectf(-CUBE_SIZE,0.,CUBE_SIZE,2*CUBE_SIZE);
    popmatrix();
    closeobj();
    }

make_local_axes(local_axes)
Object *local_axes;
    {
    makeobj(*local_axes=genobj());
    color(WHITE);
    move( 0., 0., 0.);
    draw( 0.05, 0., 0.);
    cmov( 0.07, 0., 0.);
    charst("x",1);
    move( 0., 0., 0.);
    draw( 0., 0.05, 0.);
    cmov( 0., 0.07, 0.);
    charst("y",1);
    move( 0., 0., 0.);
    draw( 0., 0., 0.05);
    cmov( 0., 0., 0.07);
    charst("z",1);
    closeobj();
    }

make_global_axes(global_axes)
Object *global_axes;
    {
    makeobj(*global_axes=genobj());
    color(WHITE);
    move( 0., 0., 0.);
    draw( 0.2, 0., 0.);
    cmov( 0.22, 0., 0.);
    charst("Lab X",5);
    move( 0., 0., 0.);
    draw( 0., 0.2, 0.);
    cmov( 0., 0.22, 0.);
    charst("Lab Y",5);
    move( 0., 0., 0.);
    draw( 0., 0., 0.2);
    cmov( 0., 0., 0.22);
    charst("Lab Z",5);
    closeobj();
    }

make_floor(floor)
Object *floor;
    {
    makeobj(*floor=genobj());
    rotate(900, 'x');
    color(GREY);
    rectf( -1.5, -.75, 1.5, .75);
    rotate(-900, 'x');
    closeobj();
    }

make_force_plate(force_plate)
Object *force_plate;
    {
    makeobj(*force_plate=genobj());
    color(WHITE);
    move( -.3, -.05, -.2);
    draw( -.3, -.05,  .2);
    draw( -.3,  0.,  .2);
    move( -.3, -.05,  .2);
    draw(  .3, -.05,  .2);
    draw(  .3,  0.,  .2);
    move(  .3, -.05,  .2);
    draw(  .3, -.05, -.2);
    draw(  .3,  0., -.2);
    move(  .3, -.05, -.2);
    draw( -.3, -.05, -.2);
    draw( -.3,  0., -.2);

    move( -.3, 0., -.2);
    draw( -.3, 0.,  .2);
    draw(  .3, 0.,  .2);
    draw(  .3, 0., -.2);
    draw( -.3, 0., -.2);
```

```
    closeobj();
    }

bar_graph(data, title, value_type, x1, x2, x3, maxy)
float data[3];
char title[], value_type[], x1[], x2[], x3[];
float maxy;
    {
    int i;
    float xpos;
    char line[10];

    ortho2( -21., 40., -121., 121.);
    color(WHITE);
    cmov2( -20., 100.);
    charstr(title);
    color(RED);
    cmov2( -3., 100.);
    charstr(x1);
    color(YELLOW);
    cmov2( 10., 100.);
    charstr(x2);
    color(CYAN);
    cmov2( 26., 100.);
    charstr(x3);
    callobj(GRID);
    xpos = 5.;
    for(i=0;i<3;i++)
        {
        if(i == 0) color(RED);
        else if(i == 1) color(YELLOW);
        else color(CYAN);

        if(data[i] > 0.) rectf( xpos, 0., xpos+5., data[i]*90./maxy);
        else rectf( xpos, data[i]*90./maxy, xpos+5., 0.);
        sprintf(line,"% .0f", data[i]);
        cmov2( xpos-1., -110.);
        charstr(line);
        xpos+= 10.;
        }

    color(GREEN);
    for(i=0;i<7;i++)
        {
        sprintf(line,"%c", value_type[i]);
        cmov2( -20., 40.-(float)i*13);
        charstr(line);
        }
    for(i=0;i<13;i++)
        {
        sprintf(line,"% .0f", maxy - (float)i*(maxy/6.));
        cmov2( -15., 85.-(float)i*15.);
        charstr(line);
        }
    }

mult_mat(a,b,r,n,m,l)
float *a,*b,*r;
int n,m,l;
    {
    int i,j,k;
    for (i=0;i<n;i++)
        for (j=0;j<l;j++)
            {
            *(r+(i*l+j))=0;
            for (k=0;k<m;k++) *(r+(i*l+j)) += *(a+(i*m+k)) * *(b+(k*l+j));
            }
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  serve_connec.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <bsd/netdb.h>
#include <stdio.h>
#include <math.h>

int server_connect(local_socket)
int local_socket;
    {
    int sock,length,port;
    struct sockaddr_in sin;
    struct hostent *hp, *gethostbyname();
    char server_name[10];

    read(local_socket,server_name,10*sizeof(char));

    /* open socket */
    if ((sock = socket (AF_INET,SOCK_STREAM,0)) < 0)
        {
        perror("opening stream socket");
        exit(0);
        }

    /* initialize socket data structure */
    sin.sin_family = AF_INET;
    hp = gethostbyname(server_name);     /* to get host address */
    bcopy (hp->h_addr, &(sin.sin_addr.s_addr), hp->h_length);
    printf("Enter port number of socket on %s: ",server_name);
    scanf("%d",&port);
    sin.sin_port = htons(port);
    /*
    sin.sin_port = htons(IPPORT_RESERVED+1);
    */

    /* connect to remote host */
    if (connect(sock,&sin,sizeof(sin)) < 0)
        {
        close(sock);
        perror("connection streams socket");
        exit(0);
        }
    return(sock);
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  sort_pannels.c
 *
 *  Description:
 *
 *  Definitions: It is defined by:
 *
 *  Created by:  Patrick J. Lord        11-Apr-89
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1989
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    subroutine: sort_pannels
    sorts the pannels in order to find the pannel to be
    drawn first (simulates fast 'Z-buffering').
*/

#include    <gl.h>

struct entry    {
        float zvalue;
        int order;
        };
struct point    {
        float x,y,z;
        };

void sort_pannels(zvalue_order,xyzvalues,n)
struct entry zvalue_order[];
struct point xyzvalues[];
int  n;
    {
    int i,j,itemp;
    float temp;
    Matrix pannels;
    int comp_function();

    for(i=0;i<(n-n%4);i+=4)
        {
        pannels[i%4][0]=(xyzvalues[i].x);
        pannels[i%4][1]=(xyzvalues[i].y);
        pannels[i%4][2]=(xyzvalues[i].z);
        pannels[i%4][3]=1.0;
        pannels[(i+1)%4][0]=(xyzvalues[i+1].x);
        pannels[(i+1)%4][1]=(xyzvalues[i+1].y);
        pannels[(i+1)%4][2]=(xyzvalues[i+1].z);
        pannels[(i+1)%4][3]=1.0;
        pannels[(i+2)%4][0]=(xyzvalues[i+2].x);
        pannels[(i+2)%4][1]=(xyzvalues[i+2].y);
        pannels[(i+2)%4][2]=(xyzvalues[i+2].z);
        pannels[(i+2)%4][3]=1.0;
        pannels[(i+3)%4][0]=(xyzvalues[i+3].x);
        pannels[(i+3)%4][1]=(xyzvalues[i+3].y);
        pannels[(i+3)%4][2]=(xyzvalues[i+3].z);
        pannels[(i+3)%4][3]=1.0;
```

```
        pushmatrix();
        multmatrix(pannels);
        getmatrix(pannels);
        popmatrix();
        zvalue_order[i].order=i;
        zvalue_order[i].zvalue=pannels[i%4][2];
        zvalue_order[i+1].order=i+1;
        zvalue_order[i+1].zvalue=pannels[(i+1)%4][2];
        zvalue_order[i+2].order=i+2;
        zvalue_order[i+2].zvalue=pannels[(i+2)%4][2];
        zvalue_order[i+3].order=i+3;
        zvalue_order[i+3].zvalue=pannels[(i+3)%4][2];
        }
    for(i=(n-n%4);i<n;i++)
        {
        pannels[i%4][0]=(xyzvalues[i].x);
        pannels[i%4][1]=(xyzvalues[i].y);
        pannels[i%4][2]=(xyzvalues[i].z);
        pannels[i%4][3]=1.0;
        }
    pushmatrix();
    multmatrix(pannels);
    getmatrix(pannels);
    popmatrix();
    for(i=(n-n%4);i<n;i++)
        {
        zvalue_order[i].order=i;
        zvalue_order[i].zvalue=pannels[i%4][2];
        }
    qsort((float *)zvalue_order,n,sizeof(zvalue_order[0]),comp_function);
    }

    comp_function(ep,ep2)
    struct entry *ep,*ep2;
        {
        if(ep->zvalue < ep2->zvalue) return( -1);
        else if(ep->zvalue == ep2->zvalue) return(0);
        else return(1);
        }
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_defini.h
 *
 *  Description: This file contains all the global definitions
 *               (including macro definitions) used by TRACK5.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                    Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#define     TRACK_DIR    "track"
#define     TRACK_PATH   "/usr/util/track"
#define     CALIB_DIR    "/usr/util/track/calibration"
#define     NONE          -1
#define     FALSE          0
#define     TRUE           1
#define     MAXLED        32       /* maximum number of channels */
#define     MAXSEG        10       /* maximum number of segments */
#define     MAXCAM         2       /* maximum number of cameras  */
#define     NUMCAM         2       /* actual number of cameras   */
#define     MAXPTS       144       /* max. points in calibration */
#define     KSCHAN         8       /* channels per Kistler frame */
#define     KSATOD     0.005       /* forceplate A/D volts/units */
#define     KSA         0.12       /* Kistler dimension A (m)    */
#define     KSB          0.2       /* Kistler dimension B (m)    */
#define     KSAZ      -0.054       /* Kistler dimension AZ (m)   */
#define     KS_F_MIN       5       /* fp minimum force for center*/
                                   /* of press. computations (N) */
#define     NAMESIZE      10       /* length of file name        */
#define     MAXLINE       80       /* maximum length of a line   */
#define     MAXDESCR      60       /* max description length     */
#define     MCOREC        51       /* size of calibration table  */
#define     TERMCAR        6       /* Terminal I/O char size     */
#define     PI       3.14159       /* definition of pi           */
#define     AGDISC   1.278E-4       /* sin of ang. discretization */
#define     FTM      0.3048         /* Conversion from feet to m. */
#define     LF            10       /* decimal value for linefeed */
#define     TIMESIZE      26       /* length of time/data array  */
#define     FL_HEADSIZE 2048       /* size of header in UNIX file*/

#define MAX(A, B)   ((A) > (B) ? (A) : (B))
#define MIN(A, B)   ((A) < (B) ? (A) : (B))
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_global.h
 *
```

```
 *  Description: This file is the overall "include" file for all TRACK5
 *               subroutines and main programs.  Definitions and
 *               global variable & structure declarations are done
 *               within this file.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                    Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    "t5i_sunixh.h"
#include    "t5i_defini.h"
#include    "t5i_struct.h"
#include    "t5i_sublis.h"

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_struct.h
 *
 *  Description: This file contains all the data structure necessary to
 *               TRACK5.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                    Copyright (C) 1988
 *              Massachusetts Institute of Technology
 *                   Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

struct t5header {
    char  track_version[NAMESIZE];      /* The version of TRACK used */
    char  file_raw[NAMESIZE];           /* Name of original raw data */
    char  dir_raw[NAMESIZE];            /* Directory of orig raw file*/
    char  file_segm[NAMESIZE];          /* Segment file name         */
    char  description[MAXDESCR];        /* Data description          */
    char  sampl_time[TIMESIZE];         /* Time of data sampling     */
    char  calib_time[TIMESIZE];         /* Time of external calib    */
    char  smooth_raw;                   /* Raw data smoothing flag   */
    char  smooth_3dp;                   /* 3D data smoothing flag    */
    char  smooth_bcs;                   /* BCS data smoothing flag   */
    char  smooth_fpd;                   /* F'plate smoothing flag    */
    char  interp_raw;                   /* raw dat interpolation flag*/
    char  interp_3dp;                   /* 3D data interpolation flag*/
    char  interp_bcs;                   /* BCS dat interpolation flag*/
    char  data_wIndo;                   /* processing window flag    */
    char  velaccomp;                    /* compute vel and acc flag  */
    char  only_3dp;                     /* compute 3D data only flag */
    char  ext_cal;                      /* use external calib flag   */
    char  selspot_flag;                 /* collect Selspot data flag */
    char  kistler_flag;                 /* collect Kistler data flag */
    int   bad_points;                   /* number of bad points found*/
    int   nb_frames;                    /* number of frames collected*/
    int   nb_segment;                   /* number of selspot segments*/
    int   nb_channel;                   /* number of selspot channels*/
    int   act_frame1;                   /* number of frames windowed */
    int   f_channels;                   /* number of bad channels    */
    int   f_segments;                   /* number of bad segments    */
    int   sray_badpt;                   /* bad points due to skew ray*/
    int   f_wframe;                     /* first frame of window     */
    int   l_wframe;                     /* last frame of window      */
    int   led_segmnb[MAXLED];           /* segment no. for ea. chan. */
    float led_xcoord[MAXLED];           /* seg x-coord for ea. chan. */
    float led_ycoord[MAXLED];           /* seg y-coord for ea. chan. */
    float led_zcoord[MAXLED];           /* seg z-coord for ea. chan. */
    float cam_pos[MAXCAM][3];           /* explicit cam pos. vector  */
    float cam_angles[MAXCAM][3];        /* explicit cam rot. angles  */
    float ext_rotmtx[MAXCAM][9];        /* rot matrix for ext calib  */
    float ext_trnvec[MAXCAM][3];        /* vectors for external 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 lenght err*/
    float globx_axis;                   /* global origin relocation X*/
    float globy_axis;                   /* global origin relocation Y*/
    float globz_axis;                   /* global origin relocation Z*/
    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   */
    };

struct t5tree {
    char  data[MAXLINE];
    char  parameter[MAXLINE];
    char  segment[MAXLINE];
    char  work[MAXLINE];
    };

struct t5terminal {
    char  line_up[TERMCAR];
    char  clear_scr[TERMCAR];
    char  clean_page[TERMCAR];
    int   nb_lines;
    int   nb_columns;
    };
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_sublis.h
 *
 *  Description: This file contains the declaration for all the
 *               subroutines of TRACK5.  The list is in alphabetical
 *               order for easy verification against ../Source/*.
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by: Peter K. Mansfield     17-Jul-88
 *                          - verified against ../Source/*.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

/* This list is in alphabetical order for easy verification against
 * ../Source/*.  Please keep it that way.
 */

extern void   t5s_3dpplt();
extern void   t5s_3dpsmo();
extern void   t5s_3ledck();
extern void   t5s_aclplt();
extern void   t5s_badfrm();
extern void   t5s_bcsplt();
extern void   t5s_bcssmo();
extern void   t5s_bdelim();
extern void   t5s_caldat();
extern void   t5s_caltrn();
extern void   t5s_chgpar();
extern void   t5s_corect();
extern float *t5s_cpt3dp();
extern float *t5s_cpt3dv();
extern float *t5s_cptbcs();
extern float *t5s_cptfpd();
extern void   t5s_fpdplt();
extern void   t5s_fpdsmo();
extern void   t5s_fprplt();
extern float *t5s_getcor();
extern int    t5s_getnam();
extern void   t5s_iledck();
extern void   t5s_intcor();
extern void   t5s_intpol();
extern float *t5s_ksdata();
extern void   t5s_lisdir();
extern void   t5s_lisfls();
extern void   t5s_lissgf();
extern void   t5s_lphead();
extern void   t5s_ncargp();
extern void   t5s_plotit();
extern int    t5s_queryn();
extern void   t5s_rawchk();
extern void   t5s_rawplt();
extern void   t5s_rawsmo();
extern float *t5s_readfp();
extern float *t5s_getdat();
extern float *t5s_s2data();
extern float *t5s_s2ksdt();
extern void   t5s_savraw();
extern void   t5s_savres();
extern void   t5s_schuta();
extern void   t5s_termin();
extern void   t5s_treein();
extern void   t5s_typpar();
extern void   t5s_window();

extern int   *vme24d32_map();
extern int   *vme16d16_map();
extern short *shortio_map();
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5i_sunixh.h
 *
 *  Description: This file includes all the UNIX include file necessary
 *               for TRACK5.
 *
 *  Definitions:
 *
 *  Created by:  Patrick J. Lord        24-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    <stdio.h>
#include    <math.h>
#include    <pwd.h>
#include    <sys/types.h>
#include    <sys/dir.h>
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5s_termin.c
 *
 *  Description: This subroutine gets the terminal information
 *               by using the UNIX Environment variables and the
 *               termcap file.
 *
 *  Definitions: It is defined by:
 *
 *                      void t5s_termin(&terminfo);
 *                      struct t5terminal terminfo;
 *
 *  Created by:  Patrick J. Lord        11-Feb-88
 *
 *  Modified by:
 *
 *  Known Bugs:
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                      Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include "t5i_global.h"

void t5s_termin(pd)
struct t5terminal *pd;
    {
    register i,j;
    short ospeed;
    char dummy[1024];
    char PC,*BC,*UP,*bp,**area,*calloc(),string[TERMCAR];

    /*
    bp=calloc(1024,sizeof(char));
    */
    bp = dummy;
    if(tgetent(bp,getenv("TERM"))!=1||
       strcmp(getenv("TERM"),"unknown")==0)
        {
        printf("*** Warning ***: Terminal type is 'unknown'!\007\n");
        pd->clean_page[0]=NULL;
        pd->clear_scr[0]=NULL;
        pd->line_up[0]=NULL;
        pd->nb_lines=24;
        pd->nb_columns=80;
        }
    else
        {
        pd->nb_lines=tgetnum("li");
        pd->nb_columns=tgetnum("co");
        area= &bp;
        sprintf(string,"%s",tgetstr("up",area));
        j=0;
        for(i=0;string[i]!=NULL;i++)
            if(string[i]>'9'||string[i]<'0')
                {
                pd->line_up[j]=string[i];
```

```
            j++;
            pd->line_up[j]=NULL;

            }
        tgetent(bp,getenv("TERM"));
        area= &bp;
        sprintf(string,"%s",tgetstr("cl",area));
        j=0;
        for(i=0;string[i]!=NULL;i++)
            if(string[i]>'9'||string[i]<'0')
                {
                pd->clear_scr[j]=string[i];
                j++;
                pd->clear_scr[j]=NULL;
                }
        tgetent(bp,getenv("TERM"));
        area= &bp;
        sprintf(string,"%s",tgetstr("cd",area));
        j=0;
        for(i=0;string[i]!=NULL;i++)
            if(string[i]>'9'||string[i]<'0')
                {
                pd->clean_page[j]=string[i];
                j++;
                pd->clean_page[j]=NULL;
                }
    }
    /*
    cfree(bp);
    */
    }
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  t5s_typpar.c
 *
 *  Description: This subroutine formats and types a standard TRACK5
 *               header to the screen.
 *
 *  Definitions: The only argumet to t5s_typpar is the address of
 *               (pointer to) a memory structure containing a
 *               standard TRACK5 header.
 *               example:
 *                         t5s_typpar(&filehdr,ptr);
 *
 *               where headptr has been declared as
 *
 *                         struct  t5header  filehdr;
 *
 *               and ptr as (defines the output pointer)
 *
 *                         FILE *ptr;
 *
 *  Created by:          Peter K. Mansfield      1-Feb-88
 *
 *  Modified by:         Patrick J. Lord        26-Jul-88
 *                       added pointer to define output device.
 *
 *  Known Bugs: None
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *                     Copyright (C) 1988
 *               Massachusetts Institute of Technology
 *                    Cambridge, Massachusetts
 *
 *  This software is subject to change without notice and should not
 *  be construed as a commitment by MIT.  MIT assumes no responsibility
 *  for the use or reliability of its software.
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 */

#include    "t5i_global.h"

void t5s_typpar(pd,ptr)
struct t5header *pd;
FILE *ptr;
    {
    struct t5terminal infterm;
    register i;

    t5s_termin(&infterm);
    if(ptr==stdout) fprintf(ptr,"%s\n",infterm.clear_scr);

    fprintf(ptr,"EXPERIMENTAL TRACK5 PARAMETERS:\n\n");
    fprintf(ptr,"SAMPLING PARAMETERS:\n");

    fprintf(ptr,"     Devices Selected: ");
    if(pd->selspot_flag)
        {
        fprintf(ptr," SELSPOT ");
        if(pd->only_3dp) fprintf(ptr,"(3D Data ONLY)  ");
        }
    if(pd->kistler_flag) fprintf(ptr," KISTLER ");
    fprintf(ptr,"\n");

    fprintf(ptr,"     Sampling Frequency = %.2f Hertz\n", pd->frequency);
    fprintf(ptr,"     Total number of Frames = %d : %.2f seconds of data\n",
          pd->nb_frames, (float)pd->nb_frames/pd->frequency);
    if(pd->selspot_flag)
        {
        fprintf(ptr,"     Total number of channels = %d\n",pd->nb_channel);
        if (!pd->only_3dp)
            {
            fprintf(ptr,"     Total number of segments = %d\n",pd->
>nb_segment);
            fprintf(ptr,"     Segment file name is: %s\n",pd->file_segm);
            }

        if(pd->ext_cal)
            {
            fprintf(ptr,"     External Calibration: ");
            if(pd->calib_time[0]==NULL)
                fprintf(ptr,"  NO EXTERNAL CALIBRATION TRANSFORMATIONS\n");
            else
                fprintf(ptr," last calibrated %s",pd->calib_time);
            }
        else
            {
            fprintf(ptr,"     Camera positions: X       Y      Z [meters]
");
            fprintf(ptr,"AX       AY      AZ [degrees]\n");
            for(i=0;i<NUMCAM;i++)
                {
                fprintf(ptr,
                    "     Camera %d: %10.3f %7.3f %6.3f %12.3f %8.3f
%7.3f\n",
                        i+1, pd->cam_pos[i][0], pd->cam_pos[i][1],
                        pd->cam_pos[i][2], pd->cam_angles[i][0],
                        pd->cam_angles[i][1], pd->cam_angles[i][2]);
                }
            }
        }

    fprintf(ptr,"\nPROCESSING PARAMETERS:\n");

    if(pd->kistler_flag)
        {
        fprintf(ptr,"     Kistler X,Y,Z Scale Factors: %4.0f %4.0f %4.0f\n",
                pd->fpxscl,pd->fpyscl,pd->fpzscl);
        }

    if(pd->selspot_flag)
        {
        fprintf(ptr,"     Skew ray max error=%3d [Selspot Units]  ",
                (int)pd->skewray_mx);
        fprintf(ptr,"   Errmax=%7.2f %%\n", pd->max_ledvar);
        }

    fprintf(ptr,"     Data Smoothing:    Interpolation     Smoother\n");
    if(pd->selspot_flag)
        {
        fprintf(ptr,"     Raw Selspot       ");
        if(pd->interp_raw)
            fprintf(ptr,"    Yes          ");
        else
            fprintf(ptr,"    None         ");
        if(pd->smooth_raw)
            fprintf(ptr,"     Yes\n");
        else
            fprintf(ptr,"     No\n");

        fprintf(ptr,"     3-D Selspot       ");
        if(pd->interp_3dp)
            fprintf(ptr,"    Yes          ");
        else
            fprintf(ptr,"    None         ");
        if(pd->smooth_3dp)
            fprintf(ptr,"     Yes\n");
        else
            fprintf(ptr,"     No\n");

        if(!pd->only_3dp)
            {
            fprintf(ptr,"     BCS Selspot       ");
            if(pd->interp_bcs)
                fprintf(ptr,"    Yes          ");
            else
                fprintf(ptr,"    None         ");
            if(pd->smooth_bcs)
                fprintf(ptr,"     Yes\n");
            else
                fprintf(ptr,"     No\n");
            }
        }

    if(pd->kistler_flag)
        {
        fprintf(ptr,"     Kistler data       None Available ");
        if(pd->smooth_fpd)
            fprintf(ptr,"     Yes\n");
        else
            fprintf(ptr,"     No\n");
        }

    if(pd->selspot_flag)
        {
        fprintf(ptr,"     BCS Derivative Computation :   ");
        if(pd->velaccomp) fprintf(ptr,"  Yes\n");
        else fprintf(ptr,"  No\n");

        fprintf(ptr,"     Global axes origin: ");
        fprintf(ptr,"(%6.3f, %6.3f, %6.3f) [meters]\n",
            pd->globx_axis, pd->globy_axis, pd->globz_axis);
        }
    fprintf(ptr,"\n");
    }
```

Bibliography

[1] Mann, Robert W., Biennial Report Department of Mechanical Engineering, MIT, 1983-84 and 1984-85.

Mann, Robert W., *Computer Aided Surgery*, Rehabilitation Engineering Society of North America (RESNA), 8th Annual Conference, Memphis, Tennessee, 1985.

[2] Chang, Grace Hwa-Pei, *Computer-Aided Surgery: an Interactive Simulation System for Intertrochanteric Osteotomy*, Master Thesis, MIT, August 1987.

[3] Antonsson, Erik K., *A Three-Dimensional Kinematic Data Acquisition and Intersegmental Dynamic Analysis System for Human Motion*, Ph.D. Thesis, MIT, June 1982.

[4] Mansfield, Peter K., *A Large Volume Kinematic Data Acquisition and Reduction System*, Ph.D. Thesis, MIT, Expected September 1989.

[5] Fijan, Robert S., *Axes of Rotations of the Joints of the Human Lower Extremity*, Master Thesis, MIT, June 1985.

[6] Conati, Frank C., *Real-Time Measurement of Three-Dimensional Multiple Rigid Body Motion*, Master Thesis, MIT, June 1977.

[7] Lenox, J. B., *Six Degrees of Freedom Human Eyeball Movement Analysis Involving Stereometric Techniques*, Ph.D. Thesis, Stanford University, 1976.

[8] Tetewsky, Avram K., *Implementing a Real-Time Computation and Display Algorithm for the Selspot System*, Master Thesis, MIT, May 1978

[9] Ottenheimer, Daniel E., *Dynamic Simulation of Auditory Spatial Displays*, Master Thesis, MIT, January 1982.

[10] Schut, G. H., On Exact Linear Measurement for the Computation of the Rotational Elements of Absolute Orientation, *Photogrammetria*, 17(1), 1960.

[11] Crandall, Stephen H., Karnopp, Dean C., Kurtz, Edward F., and Pridmore-Brown, David C., *Dynamics of Mechanical and Electromechanical Systems*, Krieger Publishing Co., Malabar, FL, 1982.

[12] Stearns, Samuel D., *Digital Signal Analysis*, Hayden Book Company Inc., Rochelle Park, New Jersey, 1975.

[13] Kernighan, Brian W., and Pike, Rob, *The UNIX Programming Environment*, Prentice-Hall, Englewood Cliffs, N.J., 1984.

[14] Rochkind, Marc J., *Advanced UNIX Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1985.

[15] Kernighan, B. W., and Plauger, P. J., *Software Tools*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1976.

[16] Dohrmann, C. R., Busby, H. R., and Trujillo, D. M., *Smoothing Noisy Data Using Dynamic Programming and Generalized Cross-Validation*, Journal of Biomechanical Engineering, Volume 110 (37), February 1988.

[17] Lord, Patrick J., *3D_Gait: A Three Dimensional Computer Graphics Display for Human Motion Analysis from TRACK Gait Data*, Bachelor Thesis, MIT, June 1987.

[18] Lanshamar, Håkan, Activity Report 1984/1985, Uppsala University Biomechanics Research Laboratory (UUBL).

[19] Lanshamar, Håkan, and Lindroth, Tom, *Assessment of Tibial Osteotomy Using Gait Analysis: Part 1, Methods and Genuine Three-Dimensional Results Presentation*, International Series on Biomechanics, Biomechanics X-A, Human Kinetics Publishers, Champaign, Illinois, 1987.

[20] Antonsson, Erik K., California Institute of Technology, Department of Mechanical Engineering, Personal Communication, October 1986.

[21] Siegel, David Mark, *Contact Sensors for Dexterous Robotic Hands*, Master of Science Thesis, MIT, June 1986.

[22] Motorola Inc., *The VMEbus Specification*, Series in Solid-State Electronics, Revision C.1, October 1985 (Second Printing).

[23] Kernighan, B. W., and Richie, D. M., *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

[24] Oppenheim, Alan V., and Schaffer, Ronald W., *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

[25] Kelley, David, *Kinesiology Fundamentals of Motion Description*, Prentice-Hall, Englewood Cliffs, N.J., 1971.

[26] Grood, E. S., and Suntay, W. J., *A Joint Coordinate System for the Clinical Description of Three-Dimensional Motions: Applications to the Knee*, Journal of Biomechanical Engineering, Volume 105, Transaction of the ASME, May 1983.

[27] Sanblad, B., Lind, M., and Schneider, W., *Requirements for the Human Computer Interfaces in Health Care, Human-Computer Communication in Health Care*, Uppsala University Data Center, Uppsala, Sweden, Elsevier Science Publishers B. V. (North Holland), 1986.

[28] Foley, James D., and Van Dam, Andries, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[29] Ferguson, John, *Microprocessor Systems Engineering*, Small Computer Series, Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.

[30] Ladin, Zvi, Flowers, Woodie C., and Meisner, William, *A Quantitative Comparison of a Position Measurement System and Accelerometry*, Approved for the Journal of Biomechanics, Publication expected in 1989.

[31] Krebs, David E., *Effect of Variation in Residuum Environment and Walking Rate on Residual Limb Muscle Activity of Selected Above-Knee Amputees*, Ph.D. Thesis, New York University, School of Education, Health, Nursing and Arts Professions, 1986.