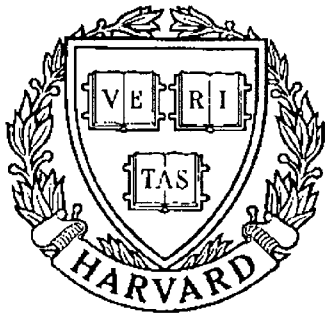
UNDERGRADUATE REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

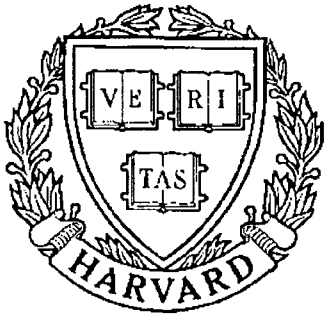
Determination of the Static Friction Present in Each Finger of a Three Fingered Modular Dextrous Hand

by J.A. Uber

U.G. 89-1
Formerly TR 89-97

UNDERGRADUATE REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Research support for this report has been provided partially by a NASA-USRA Design project grant

Determination of the Static Friction Present in Each Finger of a Three Fingered Modular Dextrous Hand

by J.A. Uber

U.G. 89-1
Formerly TR 89-97

Determination of the Static Friction Present in Each Finger of a Three Fingered Modular Dextrous Hand*

by

John A. Uber

* This work was supported in part by the National Science Foundation's Engineering Research Centers Program: NSF CDR 8803012 and by a NASA-USRA Design project grant.

Table of Contents

INTRODUCTION

The purpose of this experiment was to determine the static friction (stiction) present in each finger of a three fingered modular hand and determine the torque required to break away from a stuck position. The static friction is a function of both position and direction of movement of the motor driving the finger. Upon completion of enough trial runs from each position in each direction the results obtained will be averaged together to produce a mapping of the static friction of each finger of the gripper as a function of position and direction. This stiction data base will then be incorporated as a torque look-up table into the open loop control systems architecture of the hand. This compensation has been used in other robotics control experiments to produce a significantly more accurate open loop movement, greatly easing the operational burden of the eventual closed loop control.

MISCELLANEOUS

The ideas for this experiment were in principle borrowed from the documentation of other previous experiments. The basic testing premise was obtained from Brian S. R. Armstrong's doctoral thesis entitled "Dynamics for Robot Control" Stanford, 1988 [1]. It is intended that this paper will serve as a detailed report of all the work done and to provide a record outlining the details of the existing physical set-up of the experiment. Although this paper does outline and explain all of the testing algorithms used in the experiment it is not intended to serve as a research thesis, and is written with the intention that it will present an elementary description of the theories behind this experiment.

The physical layout of the experiment was designed to be as modular as possible, with all connecting cables being detachable at some form of connector. The pinouts of these connectors along with a diagram outlining the layout of the set-up are included for reference in appendix A. All of the code written for this experiment is fully explained in this report, and included in its entirety in thoroughly commented form in various appendices noted later. The schematic diagram of the circuit used in this experiment is included along with a flow chart describing the function of the circuit in various figures in this report.

Many thanks to Amir Sela for his help in enabling the DSP board, writing the code to interface it, and in making me "clean up" my programs so others could understand what they were doing. Also many thanks to Dr. Josip Loncaric who helped me immeasurably along the way with almost every facet of the experiment and who provided most of the mathematic algorithms used in the experiment. Thanks also to Dr. P.S. Krishnaprasad for allowing me to finish this project at my leisure and for providing the parts and equipment used in developing this experiment.

ABOUT THE HAND

The robotic hand used in this experiment was designed and constructed by Dr. Josip Loncaric and colleagues at the University of Maryland's Systems Research Center, SRC TR 89-31 [2]. It was designed as an alternative to existing anthropomorphic dextrous hands with the goal of simplifying some of the aspects of robotic hand technology. Although the Stanford/JPL and Utah/MIT hands have proven to be extremely dextrous, full complexity mechanisms, the complicated inverse kinematics resulting from their fingers' multiple joints make the grippers difficult to control. The hand built at UM-SRC has attempted to simplify the control systems architecture required by basing the design of the hand on the division of function principle.

The hand was intended to provide two independent modules; a fine manipulation stage and a "three degrees of freedom" grasping stage. This grasping stage functions as a medium complexity hand, with its grasp based on the "three points of contact with friction" idea. The grasping and manipulation of objects results from the actuation of each of the hands' three revolute finger joints. Although this gripper is nonanthropomorphic, and certainly lacks the dexterity of a human hand, it is anticipated that its mechanical simplicity, modular nature, and decoupled control strategies will make it an optimal choice for fine manipulation robotics.

The hand (see figure 1) has three "L" shaped fingers which can grasp and manipulate any arbitrarily shaped object up to eleven inches in diameter. Each finger is located five and three-sixteenths inches apart and is driven by a Pitman GM9633G102 motor with a gear down ratio of 24:1. This gear ratio is achieved through a two stage planetary gear configuration. The imprecision of this gearing linkage allows the finger to wobble back and forth, introducing an error of as much as one degree of movement from its intended position.

There is an HP HEDS-9100 optical shaft encoder precision mounted on each motor with an HEDS-5100-A03 encoder wheel which can be used to determine the speed, direction of movement, and current position of the motor shaft to within nine fiftieths of a degree. Since the finger is geared to the motor at 24:1 the position of the finger can be determined to 1/24 * 9/50 = 1/133.33 of a degree. The gripper has a set pin for each finger, which is used to reset the counter to its correct position after power up. This pin also limits the rotation of the finger to 316 degrees.

The motors are each capable of applying ten pounds of force at the fingertips normal to their direction of movement, and can withstand fifty pounds of force perpendicular to the direction of motion. A pliant spherical fingertip is mounted on the end of each finger to ease in the grasping of smooth metallic or oddly shaped objects. The entire hand weighs less than six pounds and is intended for use on the end of a General Electric GP110 robotic arm (see figure 2).

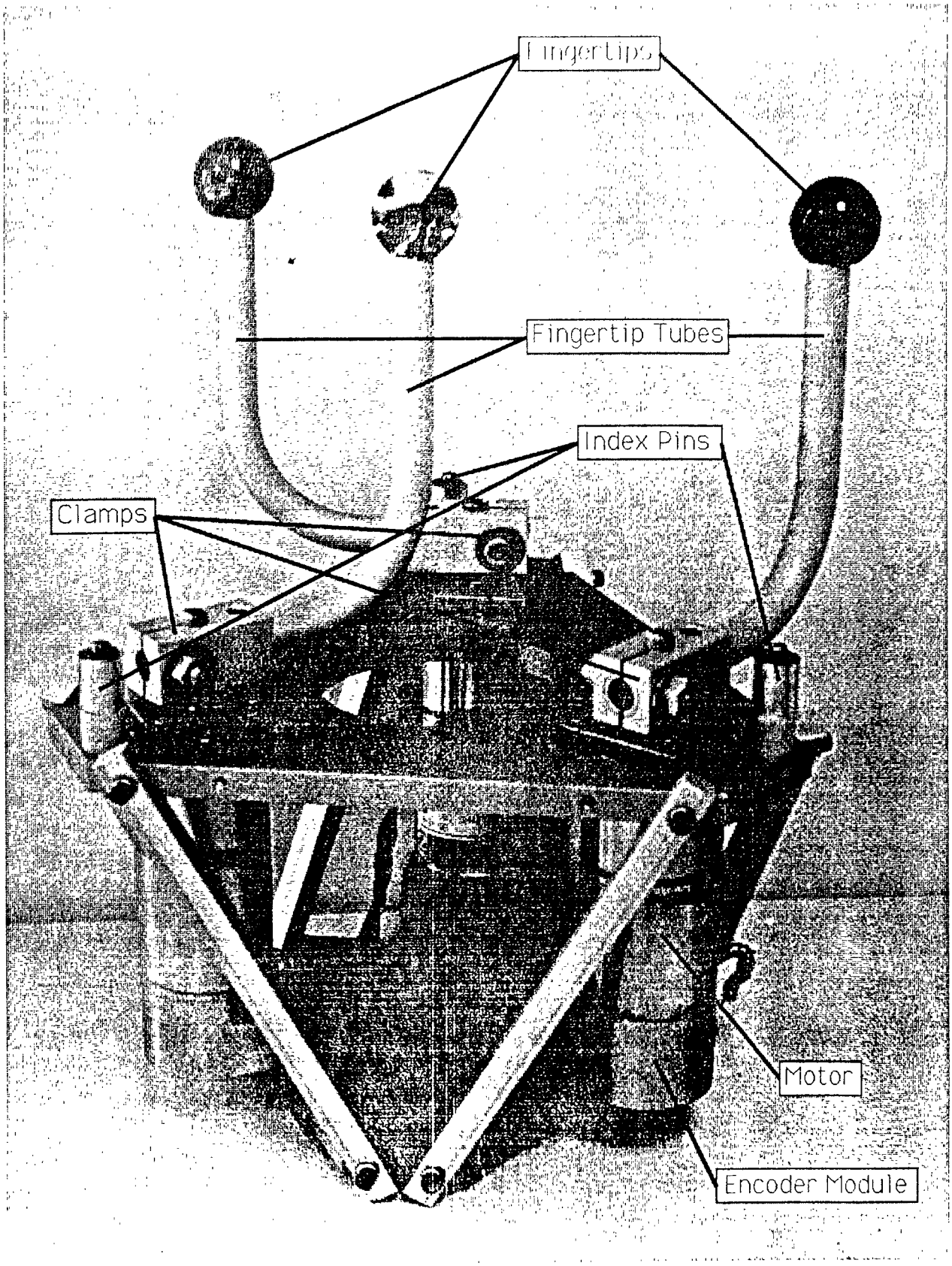

Fingertips
Fingertip Tubes
Index Pins
Clamps
Motor
Encoder Module

Figure 1

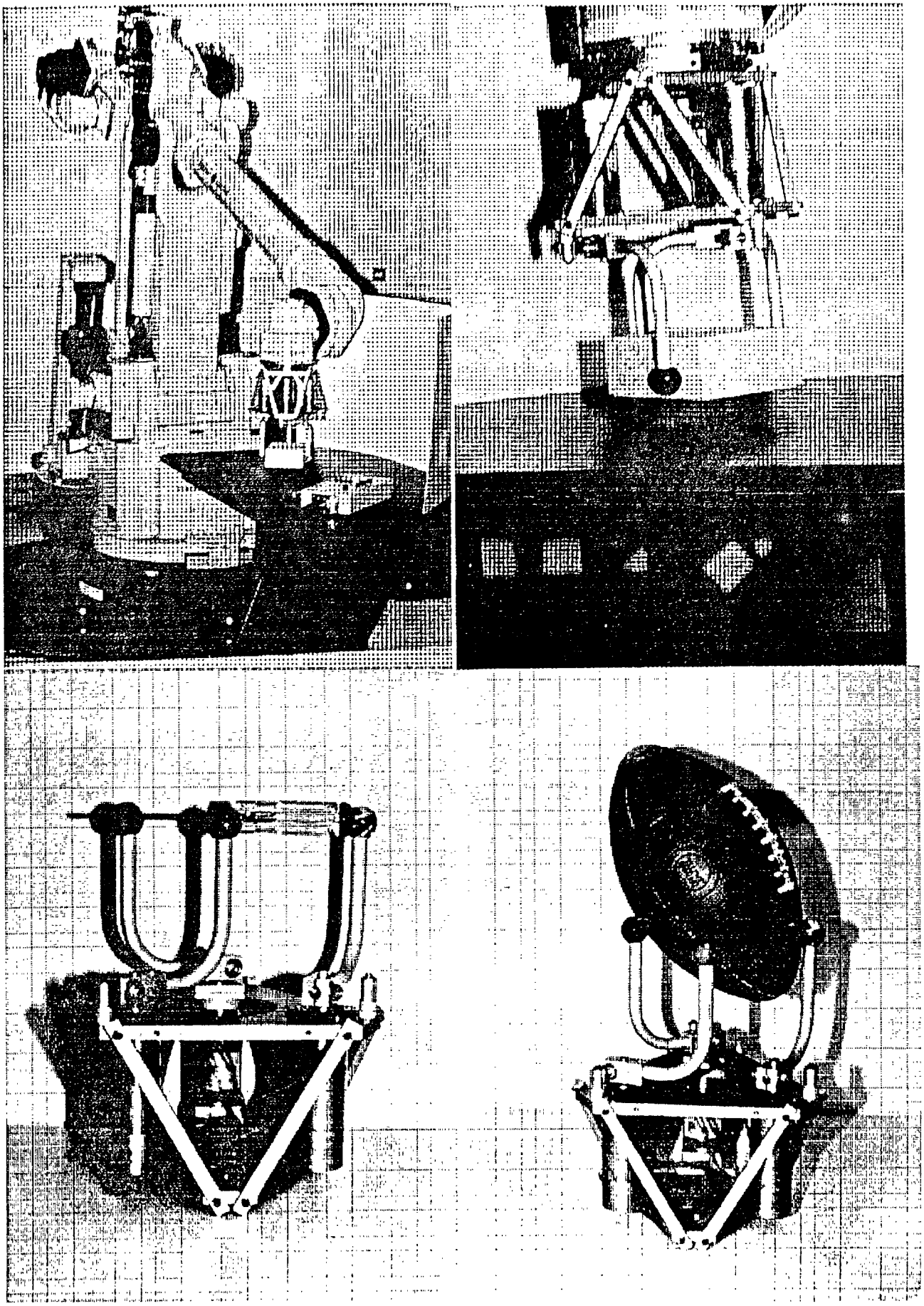


figure 2

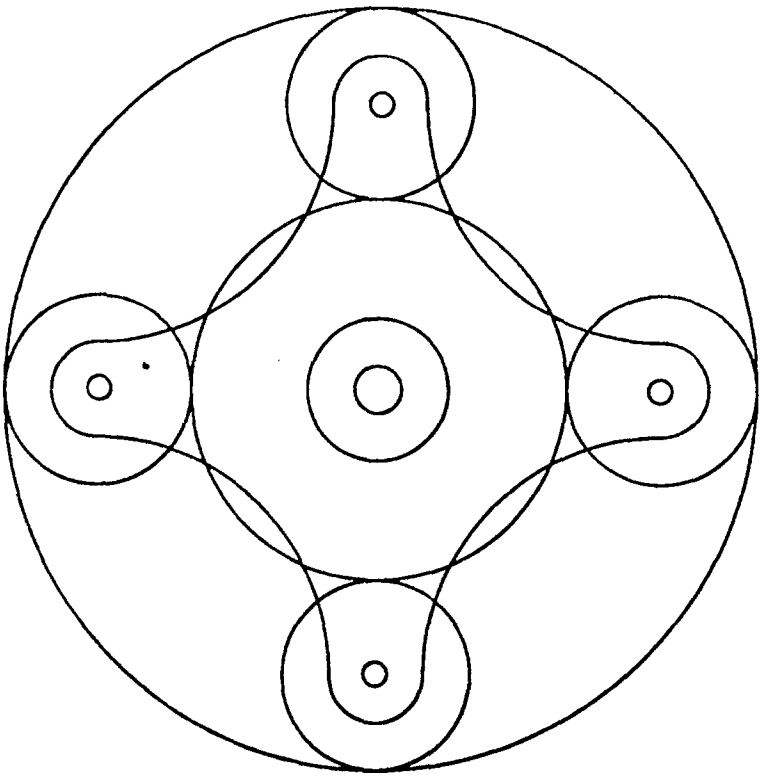
EXPERIMENTAL ALGORITHM

Essentially what the experiment should be is this: set the voltage of the motor driving the finger to zero, read the position of the encoder, increment the voltage, and read the encoder again. If the position didn't change then continue to increment the voltage and read the position until it has moved. Once it has moved, record the position it broke free from and the voltage applied when it broke free. Unfortunately this simple test procedure wouldn't work with the modular hand due to the multiple stages of stiction present in the assembly linkage between the motor and the finger.

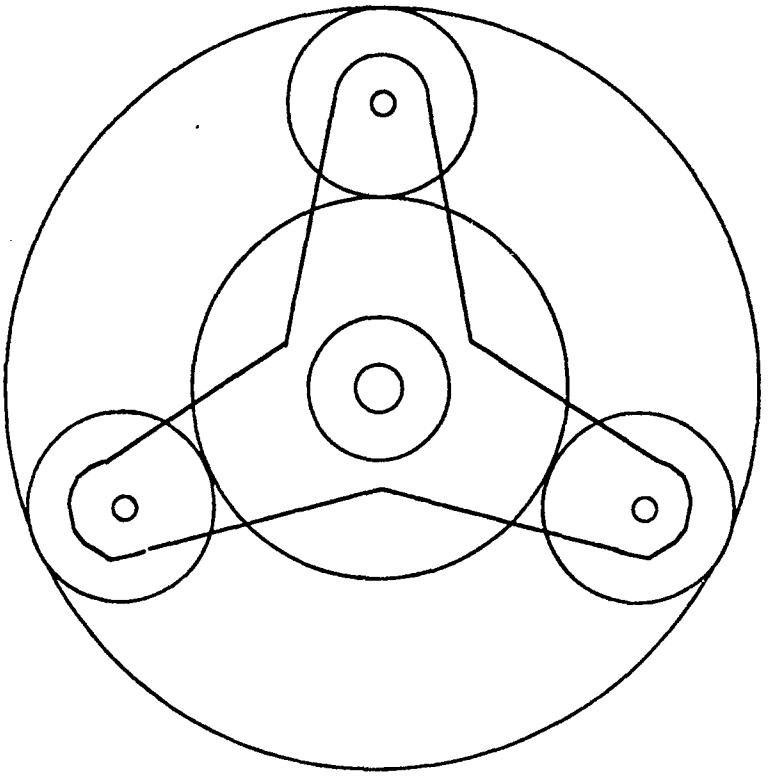
Initially the shaft of the motor won't turn due to the stiction present in the motor alone. As the incrementation of the voltage continues, the torsional force of the motor will eventually overcome the force of the friction and the shaft will turn, changing the position of the encoder. But as the shaft spins it will cause the first set of the gears to become engaged. These gears were at rest so their stiction will again stop the motor shaft. When the motor shaft stops it will redevelop the static friction within the motor housing at this new position. As the motor voltage is incremented further, the applied torque must now break the stiction of both the motor and the first gear section.

This start-stop motion will continue until finally all of the stages of the gearing and the finger have become engaged, with each individual piece stuck by its own static friction. To now break free the applied torque must overcome the resistive forces of all the individual components, the total stiction of the finger assembly. Once this is broken, the only forces acting against movement are the kinetic frictions inherent to each piece of the assembly. Since kinetic friction is always a smaller force than static friction, the finger will continue to spin once it has broken the total static friction barrier.

The total static friction of the finger consists of the combined static friction of each individual component of the finger assembly. From the configuration of the two stage planetary gear system (see figure 3) it is apparent that the map of the static friction of the finger should be comprised of five separate pieces; 1) the stiction of the finger itself due to its internal bearings and its contact with the mounting plate of the hand, 2) the stiction within the motor housing due to its own internal brushes and bearings, and 3,4,5) the stiction within the three different stages of the 24:1 gear down ratio between the motor and the finger.



Upper Stage 6:1 Planetary Gear Configuration



Lower Stage 4:1 Planetary Gear Configuration

figure 3

Simple dynamics can be used to determine that the gear ratio of a planetary gear configuration is just

$$2\left[\frac{N_{sun}}{N_{planetary}}\right] + 1$$

for each section of the assembly. Plugging in the number for the appropriate wheel of each section it turns out that the upper section operates at 6:1 and the lower section produces another 4:1.

For each rotation of the finger, the motor and the lower sun gear each turn 24 times, the lower planetary and upper sun gears turn 6 times, and the upper planetary gears and finger housing each turn once. Thus in the total friction map of the 360 degrees of motion of the finger there should be the stiction of the finger mapped once, the stiction of the motor mapped 24 times, the stiction of the first reduction set of the planetary gears mapped 6 times, and the second reduction set mapped 24 times (see figure 4).

Thus while the controlling program of the experiment must increment the voltage and watch for a change of position, it must also ignore any small movements of the encoder while all the individual pieces of the finger are winding up. The logic flow diagram of the stiction detection/measurement algorithm is shown in figure 5. The delays in the program serve two purposes. They allow the new voltage applied to the motor time to mechanically turn the motor shaft and they also allow the shaft to continue to turn after the final breakaway so that in the check for a second consecutive motion the program can detect the change in the position of the shaft.

This second time delay is set up such that the sample time between the first and second detected motions is 25 milliseconds. To ensure that only the static and not the kinetic friction is acting against the finger at breakaway it is necessary that the experiment always begin from a settled position. Also, to minimize the errant effects of the play in the gears, it is essential that the final movement of the finger before each run be in the same direction as the intended breakaway. There is an algorithm incorporated into the controlling program which checks for the correct direction of movement, and then waits for 0.2 seconds of no movement before each run of the experiment.

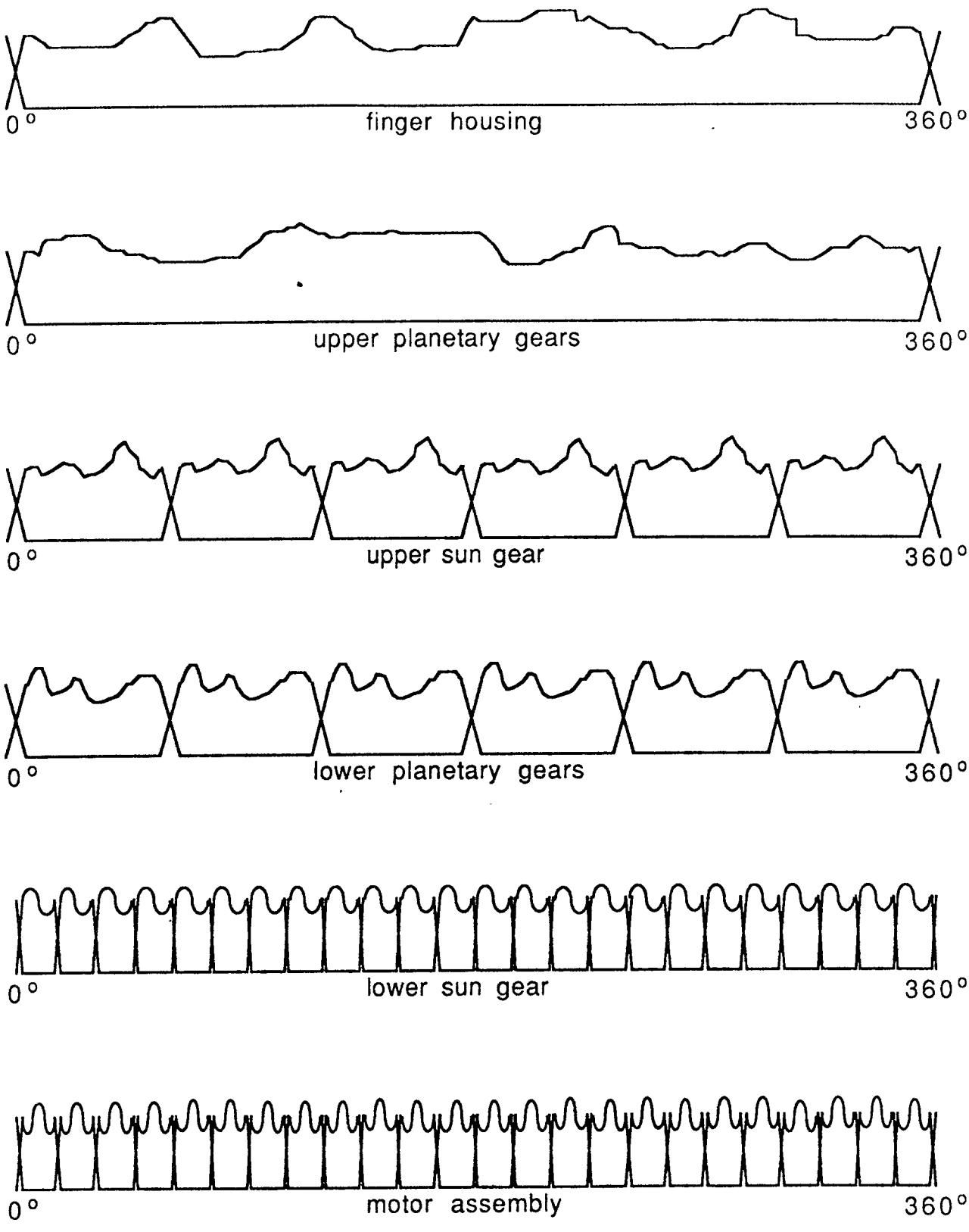

0°
finger housing
360°
0°
upper planetary gears
360°
0°
upper sun gear
360°
0°
lower planetary gears
360°
0°
lower sun gear
360°
0°
motor assembly
360°

figure 4

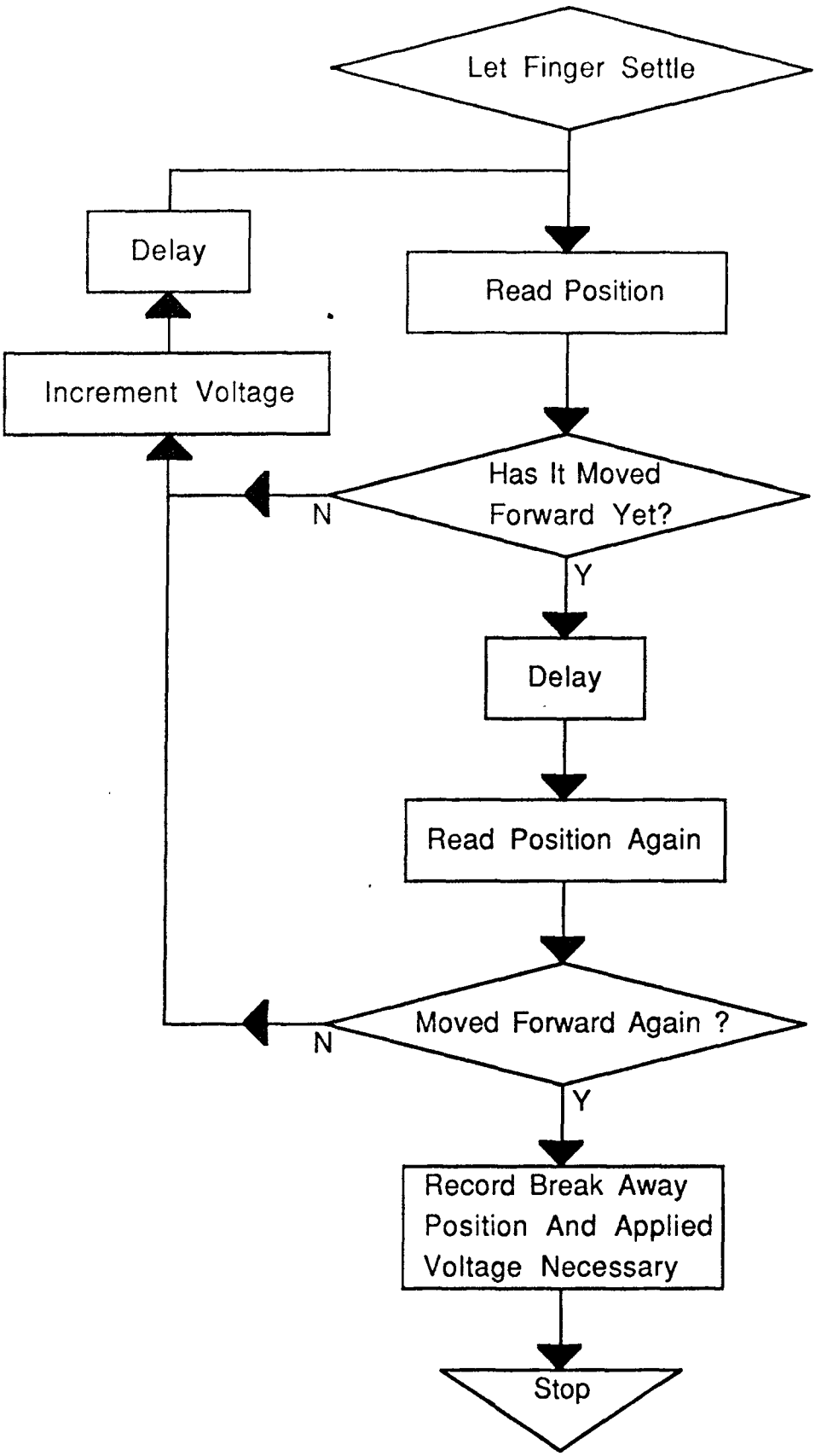

Let Finger Settle
Delay
Read Position
Increment Voltage
Has It Moved Forward Yet?
N
Y
Delay
Read Position Again
Moved Forward Again ?
N
Y
Record Break Away Position And Applied Voltage Necessary
Stop

Figure 5.

OUTLINE OF THE EXPERIMENT

Preparation for this experiment included the implementation of four main parts. The most important part of the experiment was the controlling "C" program used to run the experiment on an IBM AT PC and collect the stiction data generated. In addition to running the stiction detection algorithm noted before it must also set up all the other programs and circuitry of the experiment. It must reset the encoder after power up, enable all I/O boards, reposition the finger at the end of each run, check for execution errors, and store the accumulated data.

The second aspect of the experiment is the outputting of the motor voltage from the controlling program to the servo amplifier. This output voltage is generated by the DDA6 D/A output board of the PC and is then amplified by a bipolar operational power supply before being applied to the motor. This amplifier functions as a voltage controlled current source, using the analog voltage output of the DDA6 board to control the current applied to the motor. The board was addressed and controlled from within the controlling "C" program, and then outputs the analog voltage through D/A port #3.

The third part of the experiment was the design and construction of a digital logic circuit to decode the data from the encoder, convert the position count data from parallel to serial format, and output this data to the DSP board of the PC. This circuit ran off a clock supplied by the DSP board which it decaded, inverted, and manipulated in order to provide the correct rising and falling edges to the appropriate devices at the necessary times. These "shift" and "load" clocks were re-input to the DSP board so the board could repartition the position count from the serial input data. This circuit also required a "one shot" high strobe to enable the output latch of the decoder after power up.

The final part of the experiment was the programming of the AT&T DSP-32 board in its own machine code. This board was programmed to provide a 2 MHz clock to the circuit, and to continually read the serial input data through its serial input port and asynchronously present the data to the controlling program whenever it was requested. The DSP board must also maintain an absolute count which watches the count from the circuit and adjusts itself accordingly each time the circuit count flips over in either direction. The incorporation of this board into the set-up also gives future users the flexibility of processing the input data before presenting it to the PC, which greatly increases the speed and numerical processing power of the system. The downloaded assembly programs which enable the DSP board's external clock and continually read serial data are included in appendix B.

CONTROLLING PROGRAM

This program (BREAKAWAY.c) was used to determine the breakaway voltage and breakaway position as noted before, but must also set up each run of the experiment. Before the experiment can begin it is extremely important that the power to both the circuit and the amplifier is off. The circuit must be powered up before the amplifier is, and only after the program has begun so that it can immediately accept and generate control strobes to and from the PC. Failure to have the power to the circuitry on when requested may result in the accumulation of erroneous data and improper position initialization, but failure to have the power to the amplifier off could result in damage to the hand or an injury to the user.

To achieve an applied voltage of zero volts at the motor it is necessary for the program to output the variable "zerovolt" (~2060) to the DDA6 board. Therefore the program must output this number to the board before the amplifier is turned on or the DDA6 board will see the decimal number 0000 being output from the PC. This is converted into a negative 9.74 volts by the board which will drive the motor in the counterclockwise direction with enough force to possibly snap off the reset pin of the finger, and crush any human fingers which may be in the way. The first function of the program is to step through the possible values of "zerovolt" in order to determine which "user selected" number produces the voltage closest to zero at the output of the D/A board.

Once the program has begun, it will immediately instruct the user to properly position the other fingers of the hand to keep them from interfering in the experiment. It then systematically orders the user to power up the 5V power supply and then the amplifier once the necessary boards have been enabled and properly initialized. Next the user is told how to output enable the position decoder circuit, and then the program checks to make sure that the circuit is indeed enabled. The finger is then exercised back and forth to warm up the mechanism and reduce any overnight settling effects in the grease and bearings. Once instructed, the program will continually perform the stiction detection algorithm in each direction for as many iterative runs as are desired.

Since an optical shaft encoder loses its position when it loses power, the counter must be reset by the program after the initial power up. Upon approval of the user, the finger is rotated counterclockwise (from above), which decrements the count, until it engages the reset pin. The finger is forced against the pin by a motor current of one-half of an ampere, and the DSP board's internal position count is read. This count bias is then subtracted from all subsequent readings to compensate for the encoder's count bias and provide a relative "zero" position. This count bias will not be reinitialized until the power cycles off and on again.

Before each run, the program must make sure that the last movement of the finger is in the same direction as the finger will try to move in the experiment. It is also imperative that each successive run begin from a

bias will not be reinitialized until the power cycles off and on again.

completely stopped position. Once the finger has broken free the motor is stopped as quickly as possible in order to fit as many trials in as possible in each arc swing of the finger. Upon completion of each test run, the program must store the breakaway position and final applied torque (calculated from the applied voltage) to be later average in to the stiction value for that particular position. The actual "C" code used in the experiment is thoroughly commented and included for reference in appendix C.

The timing between successive reads was one of the more critical aspects of the experiment. Armstrong's thesis recommended sampling at 40 Hz and indeed it was determined that 25 millisecond delays generated the most accurate data. In order to ensure that the hand had time to move it was necessary to incorporate a 25 millisecond delay between successive position readings. By enabling the system clock of an existing DASH-16 I/O board in the IBM it was possible to create a series of arithmetic functions which required 0.990 milliseconds to complete. Thus the delay_msec function is just a DO-LOOP which performs as many one millisecond iterations as it is told to by the passed variable.

Once the stiction detection algorithm is complete the program sends a 10 millisecond burst of power to the motor to drive the finger a tiny distance further to prevent the finger from continually rolling back into a large stiction "valley". Once the finger reaches this new forward position the program enters a 200 millisecond delay to allow the static friction within the hand to redevelop to its full value. This ensures that each run of the experiment will only be opposed by the forces of static friction, and that dynamic friction will not be a factor in the experiment.

Since there are over 42,000 positions in each sweep of the finger it will take tens of thousands of passes before the program has recorded a number of breakaway torque values for each position. To allow for this the program has a subroutine incorporated into it which checks for the finger being stuck clockwise against the pin, and then continues the experiment back in the counterclockwise direction. When the finger again hits the reset pin in this direction it will be briefly exercised, returned to the original reset position, and then the program will continue again in clockwise direction. It it thus possible to set the experiment going and let it run all night unattended to collect tremendous amounts of data.

There are five main subroutines used by the controlling "C" program, they include:

delay_msec()	implements the () millisecond delay
get_current_pos()	reads position from DSP board
output_voltage()	outputs voltage thru D/A board
hand_settled()	checks for no movement of hand
warm_up_hand()	exercises gears and bearings in hand

All of these are thoroughly commented and included at the end of the BREAKAWAY.c program in appendix C.

D/A OUTPUT BOARD AND AMPLIFIER

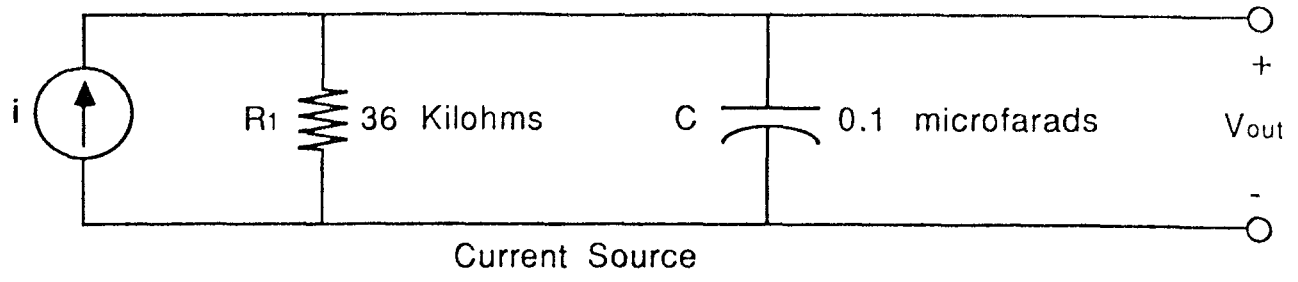
The purpose of the Metrabyte DDA6 board is to serve as a D/A output port which converts a number from within the IBM to a voltage to be applied to an external device. This voltage is then used to adjust the current being applied to the motor by the amplifier. A program which output a voltage through the port already existed from a previous experiment. The program and hard wiring was modified to make use of the open D/A port #3 (pins 12 & 13 of the DDA6 board).

The Kepco BOP 50-4M bipolar operational power supply was chosen to amplify the signal to be applied to the motors because it can be operated as a VCIS with applied voltage and current limiters. The voltage across the motor is proportional to the angular velocity of the motor, but since friction is acting as a resistive torque it is necessary to measure the applied torque required to break the static friction. The amplifier is thus used in current mode because the torque within a motor is directly proportional to the current passing through it. The voltage limiter is set to 10 volts and the current limiter is set to one-half of an amp so as to avoid high speeds and strong torques.

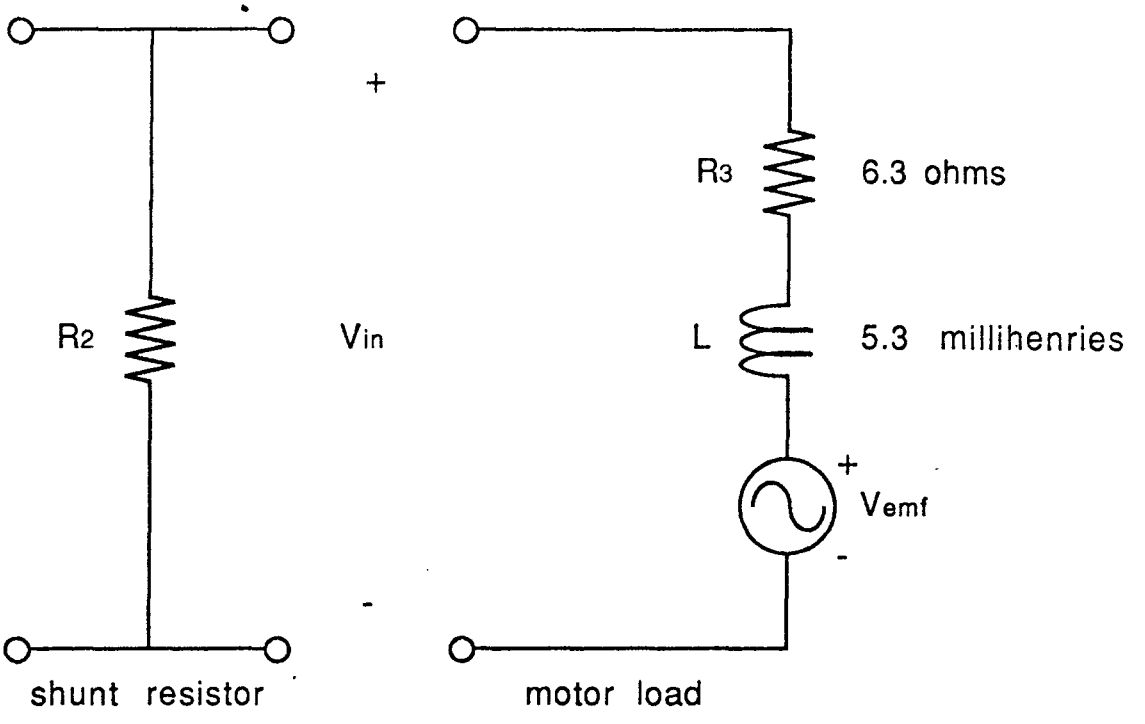
There was a simple linear relationship existing between the inputted number of the DDA6 board and the outputted voltage to the amplifier. Similarly, it was possible to determine the "applied voltage to outputted current" characteristics of the amplifier. Using these two relations along with the torque constant of the motors it was possible to determine the resistive torque of (and thus the static friction existing in) the motor by knowing only the number being sent to the DDA6 board by the PC. These relations and the algorithm used to calculate the static friction from the IBM's decimal voltage variable is included in appendix D.

When the amplifier was first connected to the motor, it began to oscillate and create a high pitched whine. The amplifier is modeled as a high shunt impedance, low capacitance current source and the motor is modeled by a low impedance, high inductance load. These models, with the specifications obtained from their manufacturers, are shown in figure 6. If a shunt resistor is placed in parallel with the load, the resulting circuit can be easily placed in state space form. The resulting state space model is observable and controllable from the resistor R_4 and is shown below.

$$\dot{\underline{Z}} = \begin{bmatrix} \frac{-1}{CR4} & \frac{-1}{C} \\ \frac{1}{C} & \frac{-R3}{L} \end{bmatrix} \underline{Z} + \begin{bmatrix} \frac{1}{C} \\ 0 \end{bmatrix} \underline{i} \qquad Y = \begin{bmatrix} 1, 0 \end{bmatrix} \underline{i}$$

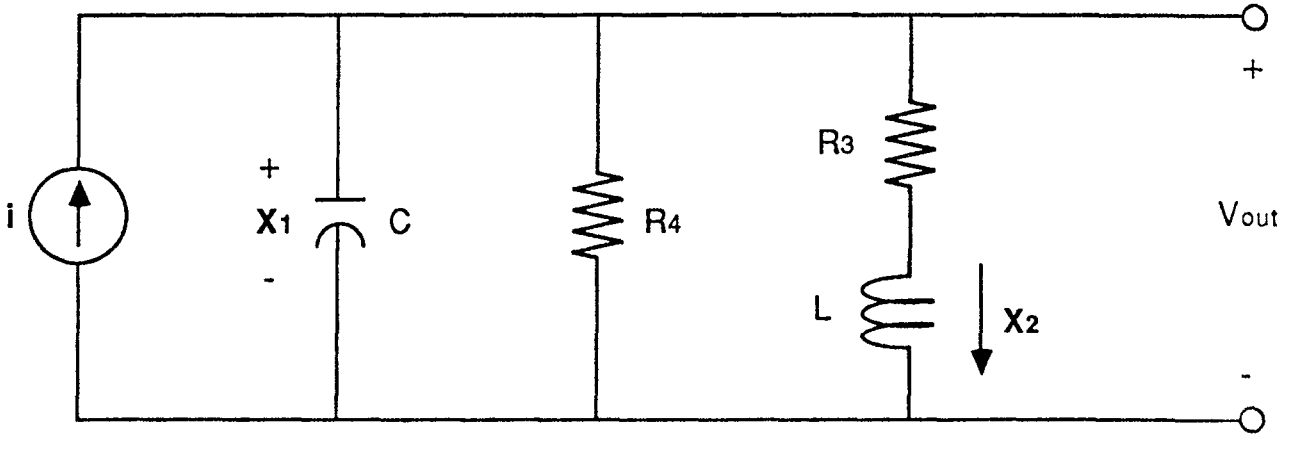


Current Source



shunt resistor

motor load



where $R_4 = R_1 \| R_2$

figure 6

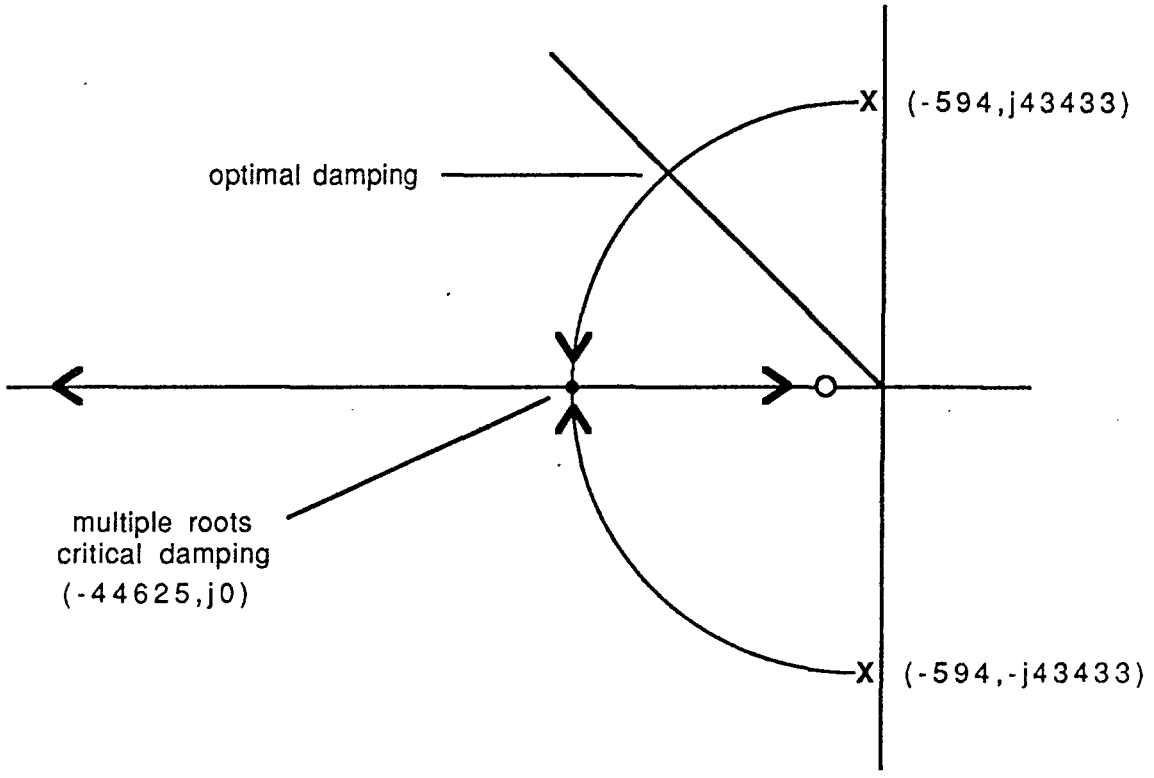
The transfer function of the circuit is easily obtained to be

$$Z = \frac{\frac{1}{C}\left(s + \frac{R_3}{L}\right)}{s^2 + s\left(\frac{R_3}{L} + \frac{1}{CR_4}\right) + \frac{R_3}{LCR_4} + \frac{1}{LC}}$$

Placing the characteristic equation of the transfer function in Root Locus form we have

$$1 + Y_4\left[\frac{\frac{1}{C}\left(s + \frac{R_3}{L}\right)}{s^2 + s\frac{R_3}{L} + \frac{1}{LC}}\right] = 0$$

Simple math shows that multiple roots occur at (-44625.922, j0). The complete locus is shown below with $Y_4=1/R_4$ as the gain parameter.



Solving for critical damping we find a gain of Y_4 = 0.0088063 which yields a value for resistor R_2 of 113.9 ohms. Similarly if we solve for an optimal damping coefficient of 0.7, we find an R_2 resistor value of 163.6 ohms. The complete math and drawings is included in appendix E.

DECODER CIRCUIT

The function of the digital logic circuitry is to take the data from the encoder, decode it, convert it from parallel to serial, introduce the proper delay time, and input it into the serial input port of the AT&T DSP-32 board. The decoder must sample the A and B channels of the encoder at 2 MHz to ensure that it never misses a state transition of the encoder wheel. In this application only the lower eight bits of the encoder's internal twelve bit counter are monitored. This count would flip over before the finger has even moved two degrees. Thus we choose to sample the decoder's output position count at a frequency of 128 KHz and update the master counter within the assembly code of the DSP board accordingly.

The logic flow diagram of this circuit is shown in figure 7, the actual hard wiring schematic is shown in figure 8. The 2.048 MHz clock was obtained from the DSP board and was used to provide a 2 MHz "shift bits" clock and then divided down to generate a 128 KHz "load word" clock. The A and B channels of the decoder are sampled on each rising edge of the 2 MHz clock, and the decoder's internal count is adjusted depending on the direction of movement of the finger.

To ensure that the count is not read during a transition, the count is loaded into the first shift register using a reconfigured 128 KHz clock. The falling edge of this clock (same as the falling edge of the 2 MHz clock) will parallel load the eight bit shift register 0.5 microseconds after the previous count update, thus ensuring its stability. These eight bits are then shifted out on the rising edge of the 2 MHz clock as serial data to the delay shift register. In contrast to the parallel to serial shift register, the delay shift register runs off of an inverted version of the 2 MHz clock and is used to delay the serial data stream by two bits for the DSP board.

The DSP board is supplied with the inverted 2 MHz clock on its ICK pin because it reads the serial line on the rising edge of this clock, and this read must occur in the middle of each bit. The nonharmonic 128 KHz clock marks the start of each word to the ILD pin of the DSP board and is thus set up to go low and trigger the start of a new word when the LSB of the next word becomes available on the line. In order to create a "+" sign bit to be read by the DSP board before the LSB is read, it was necessary to delay the serial data stream by two clock cycles. These clock signals, timing considerations, and stable data periods are shown in further detail in figure 9.

Upon power up of the board, it is necessary for the user to generate a five volt, "one shot" pulse of at least 900 nanoseconds in duration. This is accomplished by setting switch #8 ON and then OFF again. This single high strobe is sampled during the falling edge of the 2 MHz clock and signals the decoder to reset the internal inhibit logic of the chip. Once this line goes low again, the decoder's counter is output enabled on the next falling edge of the 2 MHz clock. The chip is then forever output enabled until power down as long as the /OE line is never again sent high.

the decoder to reset the internal inhibit logic of the chip. Once signals

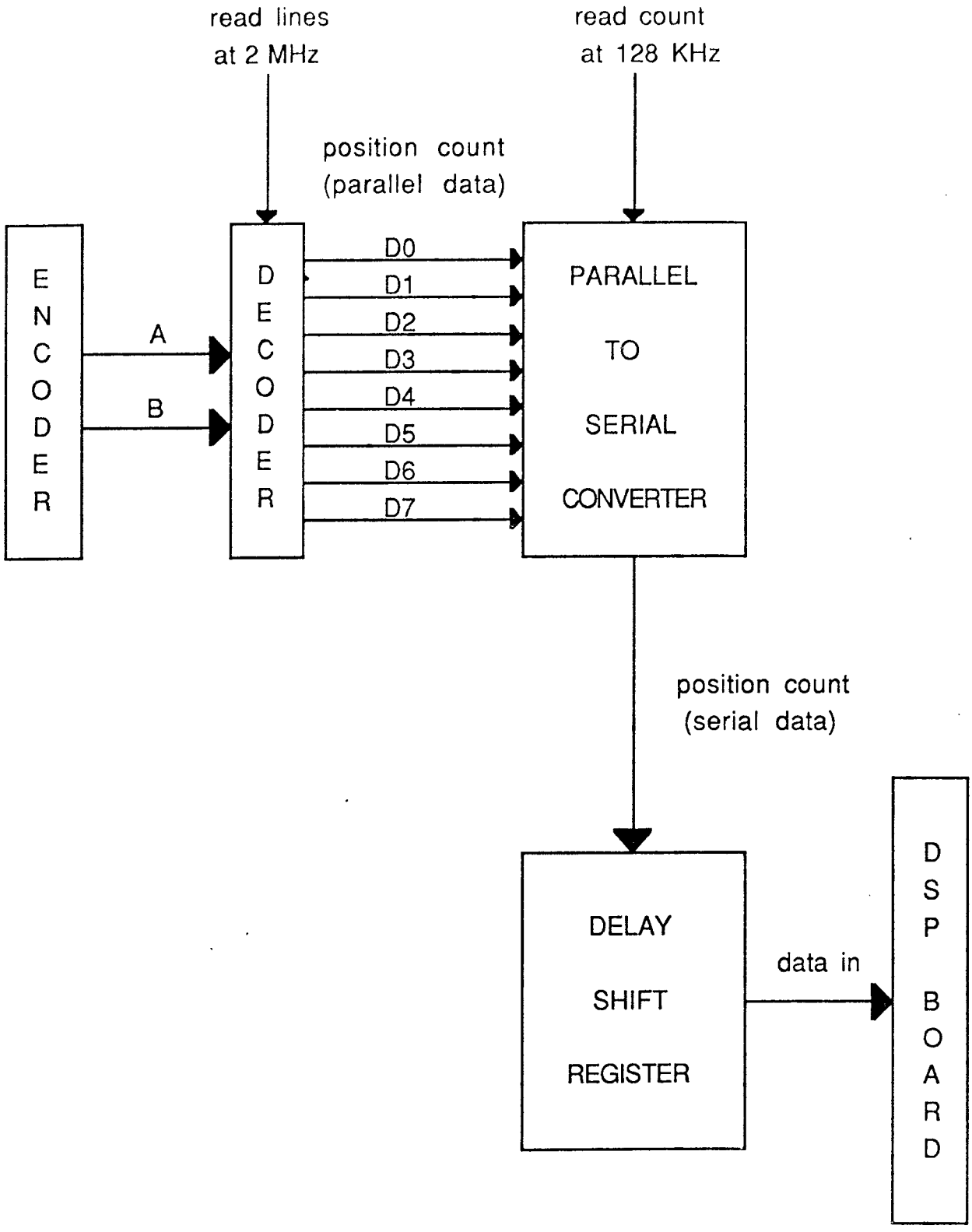

read lines
at 2 MHz
read count
at 128 KHz
position count
(parallel data)
ENCODER
A
B
DECODER
D0
D1
D2
D3
D4
D5
D6
D7
PARALLEL
TO
SERIAL
CONVERTER
position count
(serial data)
DELAY
SHIFT
REGISTER
data in
DSP
BOARD

figure 7

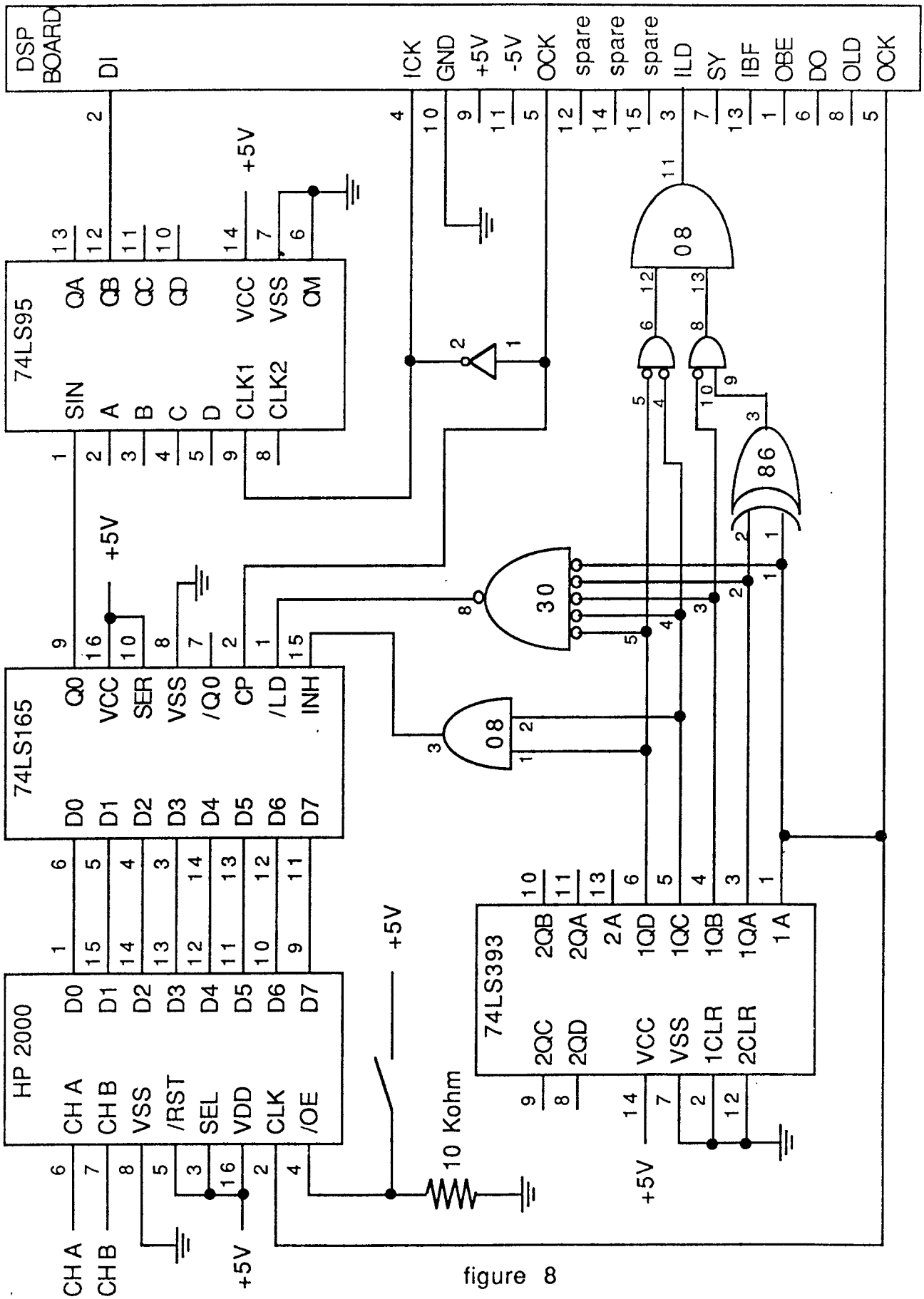

DSP BOARD
DI
ICK
GND
+5V
-5V
OCK
spare
spare
spare
ILD
SY
IBF
OBE
DO
OLD
OCK
74LS95
SIN
A
B
C
D
CLK1
CLK2
QA
QB
QC
QD
VCC
VSS
QM
74LS165
Q0
VCC
SER
VSS
/Q0
CP
/LD
INH
D0
D1
D2
D3
D4
D5
D6
D7
HP 2000
CH A
CH B
VSS
/RST
SEL
VDD
CLK
/OE
74LS393
2QC
2QD
VCC
VSS
1CLR
2CLR
2QB
2QA
2A
1QD
1QC
1QB
1QA
1A
08
30
86
10 Kohm
CH A
CH B

figure 8

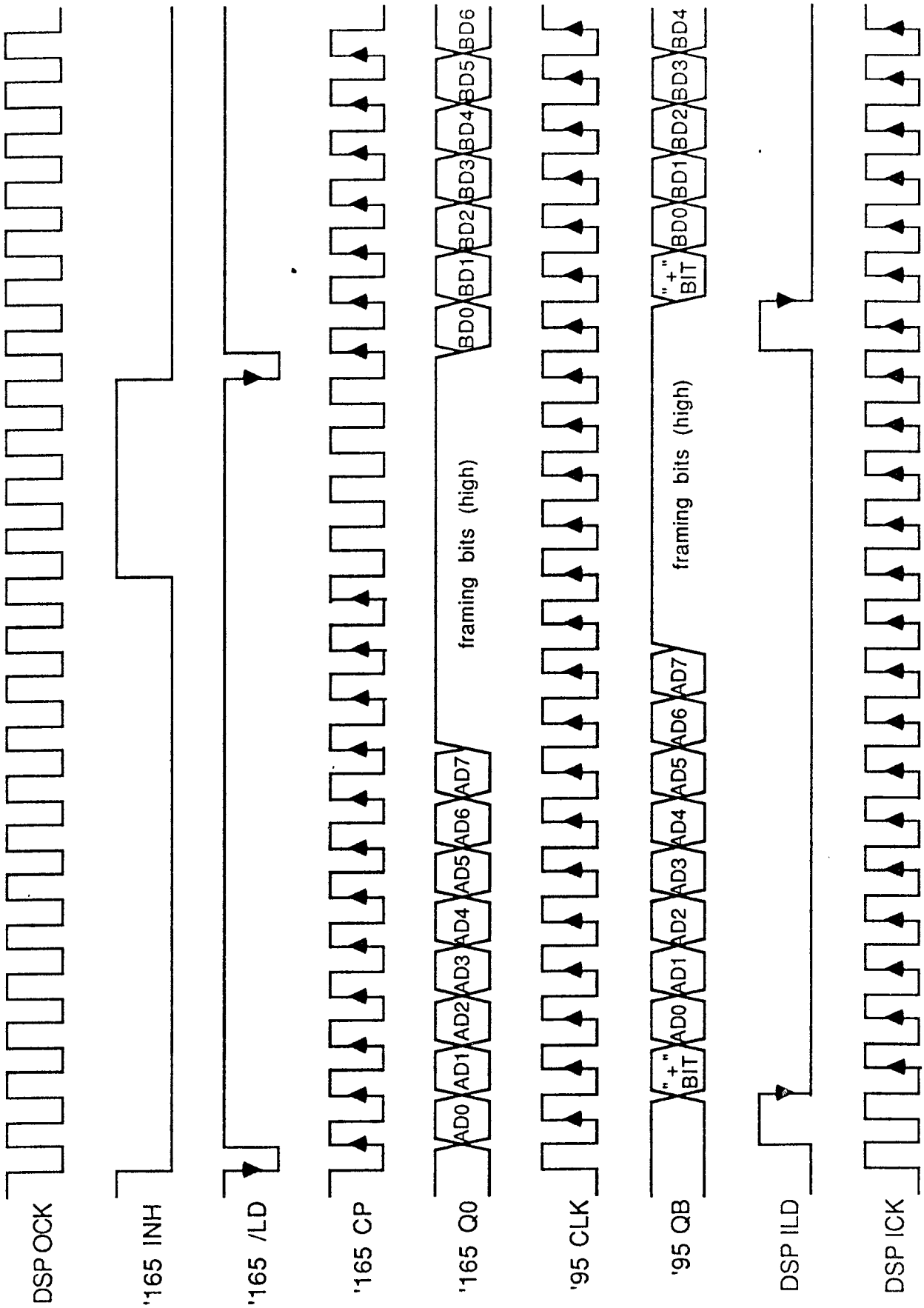

DSP OCK
'165 INH
'165 /LD
'165 CP
'165 Q0
'95 CLK
'95 QB
DSP ILD
DSP ICK
framing bits (high)
framing bits (high)
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
BD0
BD1
BD2
BD3
BD4
BD5
BD6
"+" BIT

figure 9

DIGITAL SIGNAL PROCESSOR

The DSP board was programmed to produce a 2.048 MHz clock at its OCK clock out port, input data through its serial input data port and then output this position to the IBM whenever the "C" program tried to read. The assembly code to enable the DSP board (POSITION.s) is included for reference in appendix D. The DSP input/output control register was loaded with the value 182, causing the board to read serial 16 bit input data and produce the 2 MHz clock as an output on the OCK pin. This clock was used by the circuit to generate the ICK and ILD clocking signals which were then reinput to the DSP board. The ioc register was also set to look for an external SY signal, produce no serial output data, clear the sanity, and never operate in DMA mode.

Since the eight bit data word is being transmitted in a sixteen bit word, the loaded data word is first masked in the assembly code to remove the higher order framing bits. As was noted before, this eight bit counter will flip over before the finger has even moved two degrees, thus after each read the counter checks for a flip in the circuit's eight bit count. If the count has changed by more than $2^7 = 128$ between successive readings (taken at 128 KHz) then the circuit's counter must have flipped over. The DSP board's absolute counter is then incremented or decremented accordingly, depending on the direction of the flip in the count (up or down). This scheme thus creates a one-to-one mapping of the position of the finger to the count being read in the BREAKAWAY.c program.

When BREAKAWAY.c is first begun, the finger is forced against the reset pin and the current count in the DSP board (which may even be negative) is read into the BREAKAWAY.c variable count_bias. This value is then subtracted from all subsequent readings in the "C" program to produce a relative zero position of the finger. Thus the manipulation of the circuit's position count is done in two levels, the DSP board adjusts for flips in the circuit's count and the "C" program removes the bias in the DSP count. In order to disable some of the features hardwired into the DSP board, it was necessary to remove jumper pack #2 from the board. The instruction manual detailing the DSP board is included with the literature on the DDA6 board and all other relevant handouts in appendix F.

FLEXIBLE BEAM APPARATUS

While waiting for the decoder chips to arrive from a long overdue parts order, the program was written and tested on an existing flexible beam apparatus (see figure 10). This set-up had an optical position encoder and an amplifier to drive the motor already connected to the IBM. All the assembly code for the required boards existed from a previous doctoral thesis project [3], so it was a perfect arrangement to detect programming errors. The flexible beam set-up had its own special problems however, including the inconvenience of having an absolute encoder mounted on its motor. An absolute encoder will retain its true position even if the arm is moved while the power is off, once again generating the correct position following power up.

The flexible beam also had unlimited rotation, there is no set pin to determine the relative position of the motor. Thus it was impossible to test the encoder reset portion of the program on the flexible beam. The flexible beam was direct-driven by the motor so some of the logic to compensate for the windup of the finger's gears couldn't be thoroughly tested. Also, since the motor was direct-driven, there was a significant amount of ripple torque inherent to the motor assembly. This was an added constriction on the movement of the arm in that instead of the forces opposing motion disappearing when the arm broke free, they actually increased as the position moved further forward.

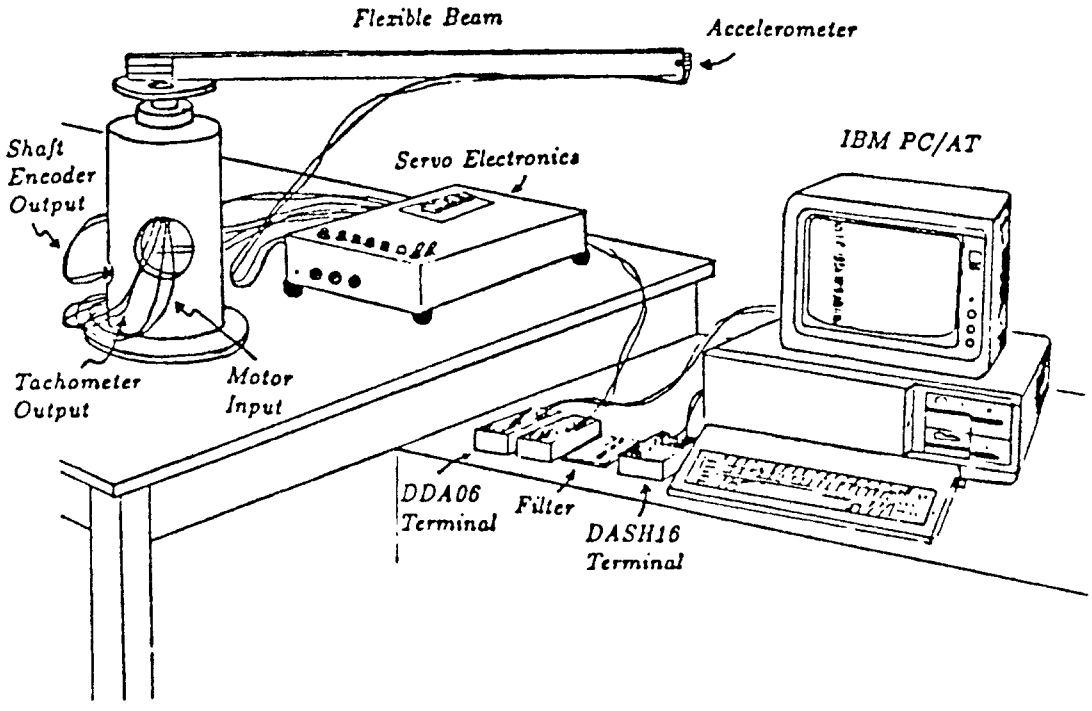


figure 10

The flexible beam experiment transmitted the encoding of the position from its decoder to the IBM PC along three foot long parallel data lines. These parallel lines, in conjunction with some extremely noisy filter and amplification circuitry, left open the opportunity for external noise to introduce a tremendous number of errors into the position reading. With the motors, power supply and other sources operating, erroneous readings were detected in almost 16% of the samples taken.

It was noticed that due to the existing hard wiring of the board, the position was loaded in the upper four bits, with a 0010 being sent continuously along the parallel lines into the lower four bits. This was used to develop a crude error detection/correction scheme. If any reading was not a multiple of 16 + 2 (n16 + 2) then it was a noise induced error and the position was immediately reread until an (n16 + 2) multiple was obtained.

PRELIMINARY OBSERVATIONS

Although a formal analysis of the data has yet to be completed, it is obvious from the accumulated data that the stiction was extremely position dependent and had a repeating pattern which oscillated approximately 21 times during each 316 degree rotation of the finger. It was also discovered that the stiction was even more direction dependent than position dependent. If it took a force F to break from position x in the clockwise direction it usually required a force very close to F to break from position x+1 or x-1 in the clockwise direction. However the required force to break from x in the counterclockwise direction was usually nowhere near F, or at least had no correlation with it.

On average, the voltage had to be incremented 33.28 times (done at 0.012846 ounce-inches per 25 milliseconds) to produce a breakaway movement in the clockwise direction. This translates to an average value of 0.4275 ounce-inches of torque to develop motion in the clockwise direction. An average incrementation of 32.69 counts results in an average breakaway torque of 0.4199 ounce-inches in the counterclockwise direction. This difference could be due to an insufficient number of data points sampled, a slight bias from zero in the current delivered by the amplifier, or simply due to the intrinsic properties of the motor assembly.

It was noted that the required breakaway torque could be as low as 0.2697 ounce-inches or as high as 0.5395 ounce-inches, and that the full range of this difference could show up in samples only a few degrees apart. It was not possible to detect some of the more subtle harmonic variations in the friction, but it is fully expected that a more thorough accumulation and analysis of data will be able to document these abnormalities as well as other as of yet undiscovered idiosyncrasies in the hand. Among these was an inability of the hand to mechanically settle into certain locations.

REFERENCES

[1] Armstrong, Brian S. R. "Dynamics for Robotic Control: Friction Modeling and Ensuring Excitation During Parameter Identification." *Doctoral Thesis*, Stanford University, May 1988.

[2] J. Loncaric, F. de Comarmond, J. Bartusek, Y. C. Pati, D. Tsakiris, R. Yang "Modular Dextrous Hand." *SRC TR 89-31*, University of Maryland, 1989.

[3] Wang, Li-Sheng "Control System Design for a Flexible Arm." *Master's Thesis*, University of Maryland, 1987.

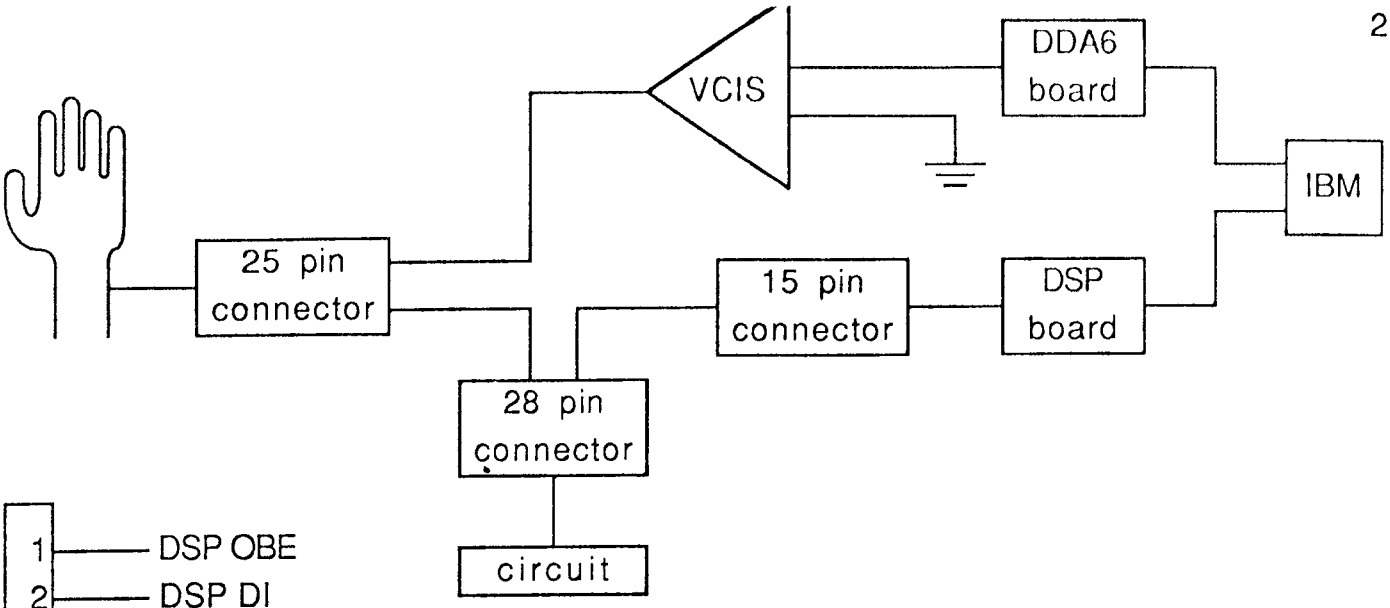
[4] Communications, Automation & Control, *D3EMU Software Emulator for AT&T's DSP32 Floating Point Digital Signal Processor and Hardware Reference Manual for the DSP32-PC Plug-In Board for the PC/XT/AT.* Release 1.7.

[5] Metrabyte Corporation, *DDA-06 Manual.* 1984.

[6] Metrabyte Corporation, *DASH-16 Manual.* 1984.

APPENDIX A

Apparatus Connector Pin Outs



1	DSP OBE
2	DSP DI
3	DSP ILD
4	DSP ICK
5	DSP OCK
6	DSP +5V
7	DSP GND
8	circuit GND
9	circuit +5V
10	spare
11	-5V
12	spare
13	IBF
14	spare
15	spare
16	/OEN
17	Red Enc. GND
18	Red Enc. CH A
19	Red Enc. +5V
20	Red Enc. CH B
21	Blue Enc. GND
22	Blue Enc. CH A
23	Blue Enc. +5V
24	Blue Enc. CH B
25	Yellow Enc. GND
26	Yellow Enc. CH A
27	Yellow Enc. +5V
28	Yellow Enc. CH B

1	DSP OBE
2	DSP DI
3	DSP ILD
4	DSP ICK
5	DSP OCK
6	DSP DO
7	DSP SY
8	DSP OLD
9	DSP +5V
10	DSP GND
11	DSP -5V
12	spare
13	DSP IBF
14	spare
15	spare

1	Red Enc. GND
2	Red
3	Red Enc. CH A
4	Red Enc. +5V
5	Red Enc. CH B
6	Blue motor GND
7	Blue motor power
8	empty
9	Yellow Enc. GND
10	Yellow
11	Yellow Enc. CH A
12	Yellow Enc. +5V
13	Yellow Enc. CH B
14	Red motor power
15	Red motor GND
16	empty
17	Blue Enc. CH B
18	Blue Enc. +5V
19	Blue Enc. CH A
20	Blue
21	Blue Enc. GND
22	empty
23	Yellow motor powe
24	Yellow motor GND
25	empty

```
.global position
.global flag
/* Position.s */
/* Program DSP chip to read data from serial port */

        ioc = 0x182     /* SY external, OCK makes SY, no SY clock
                           ICK external, ILD external, 16bit in,
                           OCK internal=CKI/8, OLD external, no output,
                           clear sanity, no DMA */
        r6 = 0          /* False */
        r7 = 1          /* True  */
        r4 = 0          /* 16 bit position register */
        r5 = position   /* 16 bit position counter */
        r8 = flag       /* Semaphore flag */
        r9 = 0          /* recent low order 8 bits */

loop:   if (ibe) goto loop
        nop
        *r8 = r6        /* Start position update by locking via flag */
        r9 = ibuf       /* read serial input */
        nop
        r9 = r9 & 0xFF  /* r9 = new low order 8 bits */
        r3 = r4         /* r3 = old count */
        nop
        r3 = r3 & 0xFF  /* r3 = old low order 8 bits */
        nop
        r3 = r3 + 0x80  /* r3 = old low bits + 128 */
        nop
        r3 = r9 - r3    /* r3 = new - old - 128 */
        nop
        if (pl) goto skipdown
        nop
        r3 = r3 + 0x100 /* r3 = new - old + 128 */
        nop
        if (mi) goto skipup
        nop
        goto newpos
        nop

skipdown:
        r4 = r4 - 0x100 /* down by 256 */
        nop
        goto newpos
        nop

skipup: r4 = r4 + 0x100         /* up by 256 */
        nop

newpos: r4 = r4 & 0xFF00        /* high order 8 bits of position */
        nop
        r4 = r4 | r9            /* splice in new low order bits */
        nop
        *r5 = r4                /* update position count in memory */
        *r8 = r7                /* unlock position */
        nop
        r3 = 0x100              /* wait loop follows */
        nop
wait:   r3 = r3 - 1
        nop
```

```
        if (p1) goto wait
 28     nop
        goto loop
        nop
position: int 1
flag:   int 1
```

APPENDIX C

BREAKAWAY.c, Controlling "C" Program

```
/*************************************************************************/
/*************************************************************************/
/*******                                                         *******/
/*******                      BREAKAWAY.C                        *******/
/*******                                                         *******/
/*******    This program computes the torque required to         *******/
/*******    break this static friction existing within the       *******/
/*******    motor assembly of the 3 fingered modular hand        *******/
/*******                                                         *******/
/*************************************************************************/
/*************************************************************************/

#include <conio.h>
#include <process.h>
#include <math.h>
#include "dsp_util.c"

extern int dda6();

int check = 0;
int trials = 0;
int stopctr, settlpos, movedone, movedtwo, totalpos;
int breakvol, base, chan, xh, x1, flag_val, i, ii, iiii;
int count_bias = 0;
int pinflag = 0;
int moved = 0;
int zerovolt = 0;
float torque;
unsigned int pos_ad;
unsigned int flag_ad;
unsigned int iii;
unsigned int startpos;

void delay_msec(loops)                  /* performs N millisecond delay */
int loops;
{
int i, j;
float value;

        for (i = 0; i < loops; i++) {
                j = abs(i);             /* abs() takes  19 microseconds */
                value = pow(i,5);       /* pow() takes 971 microseconds */
        }
return;
}

main()
{
int voltage;
```

main

```
default_addr();                                    /* Download POSITION.s to the DSP board */
if (!dsp_dl_exec("position")) exit(1);
dsp_run();
pos_ad=get_addr("position");
flag_ad=get_addr("flag");
                                                                                    60

printf("Move all fingers away from the center ");          /* Clear fingers out of the way */
printf("of the hand and enter a 1\n");
do { scanf("%d",&check); } while ( check != 1 );

printf("The DSP board has been enabled, please turn on\n");  /* Turn on the +5 volt supply */
printf("the 5V power supply and enter a 2\n");
do { scanf("%d",&check); } while ( check != 2 );

startpos = get_current_pos();                      /* Set up DSP ioc register */
output_voltage( 2055 );
printf("The D/A board has been enabled and zeroed out.\n");  /* Turn on the amplifier */
printf("Turn on the Kepco amplifier and enter a 3\n");
do { scanf("%d",&check); } while ( check != 3 );

do {                                               /* Select value of zerovolt */
        set_zero_voltage();
        printf("Is this an acceptable value of zerovolt? (1=Yes, 0=No) ");
        scanf("%d",&check);
} while (check!=1);                                                                 80


printf("Enable the circuit by moving switch #8 ON and\n");  /* Enable the position circuit */
printf("then OFF again, then enter a 5\n");
do { scanf("%d",&check); } while (check != 5 );

printf("To reset the finger position counter enter a 6\n");  /* Determine count bias offset */
do { scanf("%d",&check); } while (check != 6 );

output_voltage( zerovolt - 300 );                  /* Rewind finger */  90
hand_settled();
count_bias=get_current_pos() ;                     /* Read unbiased count */
printf("Settled reset position is %d\n",count_bias);
output_voltage( zerovolt );                        /* Relax the motor */

printf("To exercise the finger enter a 7\n");      /* Exercise the finger */
do { scanf("%d",&check); } while (check != 7);
warm_up_hand();
output_voltage( zerovolt );                        /* Kill the motor */
                                                                                   100
printf("Each run takes approximately 1.6 seconds,\n");  /* Determine number of trials */
printf("10,000 trials can be done in 275 minutes\n\n");
printf("How many iterations to be performed?   \n");
do { scanf("%d",&trials); } while ( trials == 0 );

output_voltage( zerovolt - 50 );                   /* Rewind finger until settled */
```

```
hand_settled();
startpos = get_current_pos();
printf("Current position is %u\n",startpos);                    /* Display position and bias */
printf("Count bias is %d\n",count_bias);                                                    110

printf("To begin the repetitive experiment enter an 8\n");      /* Begin the experiment */
do { scanf("%d",&check); } while (check != 8 );

printf("Friction detection algorithm has begun\n");

                                        /**** Begin iterative trials ****/

for ( ii = 1; ii <= trials; ii++) {
                                                                                            120
        do {

                printf("# %d: ", ii );
                delay_msec(25);                                 /* Set up next clockwise run */
                hand_settled();
                startpos = get_current_pos();
                moved=0;
                voltage = zerovolt;
                output_voltage( voltage );                      /* Set voltage to zero */
                                                                                            130
                do {
                        delay_msec(25);
                        movedone = get_current_pos();
                        if ( movedone > startpos ) {            /* Moved forward */
                                delay_msec(25);
                                movedtwo = get_current_pos();
                                if ( movedtwo > movedone ) {    /* Moved forward again */
                                        moved = 1;
                                        break;
                                }                                                           140
                                else {
                                        startpos = movedtwo;    /* Experiencing wind up */
                                        voltage++;
                                }
                        }
                        else {
                                startpos = movedone;            /* Still completely stuck */
                                voltage++;
                        }
                        output_voltage( voltage );              /* Apply incremented voltage */

                } while ( voltage < (zerovolt + 150));          /* While finger not against pin */

                ii++;

                if( voltage >= (zerovolt + 149)) pinflag = 1;   /* Finger is against pin */

                breakvol = voltage - zerovolt;
                torque = breakvol * 0.012846153;
```

```
                                                                    160
        if ( moved == 1) {
                output_voltage( voltage + 15 );    /* Burst of power after it broke free */
                delay_msec(15);
                output_voltage( zerovolt );                 /* Kill the motor */
                delay_msec(200);
                printf("%5u ",startpos);                    /* Output the data */
                printf("%4d ",breakvol);
                printf("%9.5f ounce-inches\n",torque);
        }
        else {                                                      170
                printf("\n");                               /* Stuck against pin, no output */
                output_voltage( zerovolt );
        }
        voltage = zerovolt;

} while ( pinflag != 1 );                                   /* Continue until against pin */


pinflag = 0;                                                /* Reset against-pin flag */
                                                                    180
do {

        printf("# %d: ", ii );
        delay_msec(25);                                     /* Set up counterclockwise run */
        hand_settled();
        startpos = get_current_pos();
        moved = 0;
        voltage = (zerovolt - 1);
        output_voltage( voltage );                          /* Set voltage to zero */
                                                                    190
        do {
                delay_msec(25);
                movedone = get_current_pos();
                if ( movedone < startpos ) {                /* Moved forward */
                        delay_msec(25);
                        movedtwo = get_current_pos();
                        if ( movedtwo < movedone ) {        /* Moved forward again */
                                moved = 1;
                                break;
                        }                                           200
                        else {
                                startpos = movedtwo;        /* Experiencing wind up */
                                voltage--;
                        }
                }
                else {
                        startpos = movedone;                /* Still completely stuck */
                        voltage--;
                }
                output_voltage( voltage );                  /* Apply incremented voltage */

        } while (voltage > (zerovolt - 150));               /* While finger not against pin */



if ( movedtwo < movedone ) {    /* Moved forward again */
```

```
                ii++;

                if( voltage <= (zerovolt - 99)) pinflag = 1;        /* Finger is against pin */

                breakvol = voltage - zerovolt;
                torque = breakvol * 0.012846153;

                                                                                    220
                if ( moved == 1 ) {
                        output_voltage( voltage - 15 );    /* Burst of power after it broke free */
                        delay_msec(15);
                        output_voltage( zerovolt );                 /* Kill the motor */
                        delay_msec(200);
                        printf("%5u ",startpos);                    /* Output the data */
                        printf("%4d",breakvol);
                        printf("%9.5f ounce-inches\n",torque);
                }
                else {                                                              230
                        printf("Hand is being exercised \n");      /* Stuck against pin, no output */
                        output_voltage( zerovolt + 250 );          /* Exercise the finger */
                        hand_forward( 36000 );
                        output_voltage( zerovolt - 250 );
                        hand_backward( 5000 );
                        output_voltage( zerovolt + 250 );
                        hand_forward( 33000 );
                        output_voltage( zerovolt - 150 );          /* Return to reset position */
                        hand_settled();
                        output_voltage( zerovolt );                                 240
                }

                voltage = zerovolt;

        } while ( pinflag != 1 );                                  /* Continue until against pin */

        pinflag = 0;                                               /* Reset against-pin flag */

    }
                                                                                    250
}

                /*********************************************************/
                /**** Subroutine to read position from the DSP board ****/
                /*********************************************************/


get_current_pos()                                                   get_current_pos
{
    while (1) {                                                                     260
        flag_val=dsp_up_int(flag_ad);
        if (flag_val == 1 ) {
            iii= dsp_up_int(pos_ad);
            dsp_dl_int(flag_ad,0);
            iii = iii - count_bias;            /* Remove the DSP board's count bias */
```

```
                        return iii;
                }
        }
}
```

270

get_addr(s) get_addr

```
char *s;
{
        unsigned int iii;
        if ((iii=find_label_name(s)) == -1){
                printf("Global label '%s' missing from DSP32 program\n",s);
                exit(1);
        }
        return iii;                                                    280
}
```

```
                        /************************************/
                        /**** Output voltage subroutine ****/
                        /************************************/
```

output_voltage(voltage) output_voltage

```
int voltage;
{                                                                      290
        base = 774;
        xh = voltage/256;
        xl = voltage - xh * 256;
        outp( base, xl);
        outp( base+1, xh );
}
```

300

```
                        /**********************************/
                        /**** Exercise hand subroutine ****/
                        /**********************************/
```

warm_up_hand() warm_up_hand

```
{
                for (iiii = 0; iiii < 10; iiii++) {          /* Perform 10 times */
                        output_voltage( zerovolt + 50 );
                        hand_settled();                                310
                        output_voltage( zerovolt - 50 );
                        hand_settled();
                        output_voltage( zerovolt + 50 );
                        hand_settled();
                        output_voltage( zerovolt - 50 );
                        hand_backward( 8000 );
                        output_voltage( zerovolt + 50 );
                        hand_forward( 38000 );
```

```
                        return iii;
                }
        }
}

                                                                        270
get_addr(s)                                                     get_addr

char *s;
{
        unsigned int iii;
        if ((iii=find_label_name(s)) == -1){
                printf("Global label '%s' missing from DSP32 program\n",s);
                exit(1);
        }
        return iii;                                                     280
}

                                /***********************************/
                                /**** Output voltage subroutine ****/
                                /***********************************/


output_voltage(voltage)                                         output_voltage
int voltage;
{                                                                       290
        base = 774;
        xh = voltage/256;
        x1 = voltage - xh * 256;
        outp( base, x1);
        outp( base+1, xh );
}


                                                                        300
                                /***********************************/
                                /**** Exercise hand subroutine ****/
                                /***********************************/


warm_up_hand()                                                  warm_up_hand
{
                        for (iiii = 0; iiii < 10; iiii++) {     /* Perform 10 times */
                                output_voltage( zerovolt + 50 );
                                hand_settled();                         310
                                output_voltage( zerovolt - 50 );
                                hand_settled();
                                output_voltage( zerovolt + 50 );
                                hand_settled();
                                output_voltage( zerovolt - 50 );
                                hand_backward( 8000 );
                                output_voltage( zerovolt + 50 );
                                hand_forward( 38000 );
```

```
                output_voltage( zerovolt - 50 );
                hand_backward( 4000 );                                  320
                output_voltage( zerovolt + 50 );
                hand_forward( 34000 );
                output_voltage( zerovolt - 64 );
                hand_backward( 18000 );
                output_voltage( zerovolt + 64 );
                hand_forward( 24000 );
                output_voltage( zerovolt - 84 );
                hand_backward( 14000 );
                output_voltage( zerovolt + 84 );
                hand_forward( 28000 );                                  330
                output_voltage( zerovolt - 150 );
                hand_backward( 10000 );
                output_voltage( zerovolt + 150 );
                hand_forward( 32000 );
                output_voltage( zerovolt - 150 );
                hand_backward( 3000 );
                output_voltage( zerovolt + 150 );
                hand_forward( 35000 );
                output_voltage( zerovolt - 150 );
                hand_backward( 21000 );                                 340
                output_voltage( zerovolt );
        }
}


hand_forward(position)          /* Move hand clockwise to (position) */     hand_forward
int position;
{                                                                       350
        do { startpos = get_current_pos(); } while ( startpos <= position );
        output_voltage( zerovolt );
}


hand_backward(position)         /* Move hand counterclockwise to (position) */hand_backward
int position;
{
        do {
                startpos = get_current_pos();                           360
        } while ( startpos >= position );
        output_voltage( zerovolt );
}


                /*****************************************/
                /**** Check to see if hand has settled ****/
                /*****************************************/

                                                                        370
hand_settled()                                                          hand_settled
```

```
{
                        startpos = get_current_pos();
                        for (stopctr = 0; stopctr < 30; stopctr++) {      /* Until 30 readings the same */
                                settlpos = get_current_pos();
                                if ( settlpos != startpos ) {             /* Hand isn't settled */
                                        stopctr = 0;
                                        startpos = settlpos;
                                }
                        }                                                                          380
}

                        /************************************************/
                        /**** Subroutine to select value of zero_volts ****/
                        /************************************************/

set_zero_voltage()                                                          set_zero_voltage
{
        printf("It is necessary to reselect 'zero_volts'.\n");          /* Choose value of zero_volts */
        printf("Please attatch a voltmeter on the millivolt scale\n");
        printf("to the lower right connectors of the amplifier.\n");
        printf("Enter a 4 to proceed with the subroutine to choose\n");
        printf("the correct value for the variable 'zero_volts'.\n");
        printf("You will need to select the number which produces the\n");
        printf("smallest non-negative voltage at the connector.\n");

        do { scanf("%d",&check); } while (check != 4);

        zerovolt = zero_out_dda6();                                                                400
        printf("Your choice for 'zero_volts' was %d\n\n", zerovolt );
}

zero_out_dda6()                                                             zero_out_dda6
{
int volt;
        for (volt = 2030; volt <= 2080; volt++) {
                printf("Zero_volts is now set to be : %d\n", volt );
                output_voltage( volt );
                printf("Enter a 0 to select this value of zero_volts. \n");                        410
                printf("Enter a 4 to increase zero_volt, \n");

                scanf("%d",&check);
                if (check == 0) break;
        }
        return volt;
}
```

APPENDIX D

Decimal Voltage To Applied Torque Conversion

	Decimal Number	D/A millivolts	Amp. milliamps
1	2000	-292	-114.7
2	2001	-287	-112.5
3	2002	-282	-110.5
4	2003	-277	-108.5
5	2004	-272	-106.6
6	2005	-267	-104.7
7	2006	-262	-102.8
8	2007	-257	-100.8
9	2008	-252	-98.8
10	2009	-247	-96.8
11	2010	-242	-94.9
12	2011	-237	-92.9
13	2012	-232	-91.1
14	2013	-227	-89.1
15	2014	-222	-87.2
16	2015	-217	-85.2
17	2016	-212	-83.4
18	2017	-208	-81.4
19	2018	-203	-79.5
20	2019	-198.3	-77.5
21	2020	-193.2	-75.6
22	2021	-188.4	-73.7
23	2022	-183.8	-71.8
24	2023	-178.6	-69.8
25	2024	-173.9	-68.0
26	2025	-169.2	-66.1
27	2026	-164.2	-64.2
28	2027	-159.2	-62.1
29	2028	-154.2	-60.2
30	2029	-149.2	-58.2
31	2030	-144.5	-56.3
32	2031	-139.2	-54.3
33	2032	-134.8	-52.6
34	2033	-129.8	-50.7
35	2034	-125.5	-49.0
36	2035	-120.6	-46.9
37	2036	-115.6	-44.9
38	2037	-110.3	-42.8
39	2038	-105.7	-41.1
40	2039	-100.8	-39.1
41	2040	-96.1	-37.2
42	2041	-90.6	-35.2
43	2042	-85.9	-33.3
44	2043	-80.9	-31.3
45	2044	-76.2	-29.5
46	2045	-70.9	-27.4
47	2046	-66.2	-25.5
48	2047	-61.1	-23.5
49	2048	-56.5	-21.7
50	2049	-51.5	-19.43
51	2050	-46.8	-17.65
52	2051	-41.8	-15.70
53	2052	-37.6	-14.11
54	2053	-32.0	-11.95
55	2054	-27.2	-10.03
56	2055	-22.3	-8.12

Decimal Number	D/A	millivolts Amp	milliamps
57	2056	-17.3	-6.30
58	2057	-12.1	-4.21
59	2058	-7.2	-2.38
60	2059	-2.1	-0.398
61	2060	2.3	1.115
62	2061	7.3	3.33
63	2062	12.1	5.15
64	2063	16.8	6.98
65	2064	21.4	8.81
66	2065	26.7	10.87
67	2066	31.6	12.71
68	2067	35.8	14.45
69	2068	40.7	16.31
70	2069	44.8	17.75
71	2070	49.7	19.91
72	2071	53.3	21.5
73	2072	57.9	23.3
74	2073	63.7	25.7
75	2074	68.5	27.6
76	2075	73.5	29.9
77	2076	79.2	31.8
78	2077	83.9	33.6
79	2078	87.6	35.1
80	2079	92.8	37.1
81	2080	95.3	38.2
82	2081	100.1	40.1
83	2082	108.8	43.4
84	2083	110.6	44.3
85	2084	115.7	46.2
86	2085	123.3	49.2
87	2086	128.5	51.1
88	2087	132.1	52.7
89	2088	138.3	55.2
90	2089	143.1	57.0
91	2090	149.2	59.1
92	2091	153.8	61.0
93	2092	158.9	63.0
94	2093	163.8	64.9
95	2094	168.1	66.7
96	2095	173.2	68.7
97	2096	177.9	70.5
98	2097	182.9	72.5
99	2098	188.5	74.8
100	2099	193.3	76.6
101	2100	198.1	78.6
102	2101	203	80.5
103	2102	207	82.4
104	2103	213	84.4
105	2104	218	86.5
106	2105	223	88.4
107	2106	227	90.1
108	2107	232	92.1
109	2108	237	94.0
110	2109	242	95.9
111	2110	247	97.7

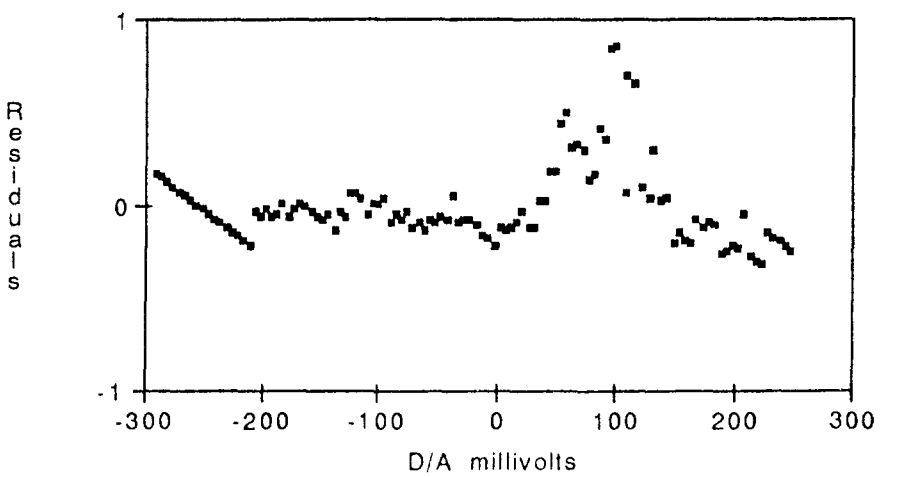
Data File: Dapper Dam Dependent Variable: Decimal Number

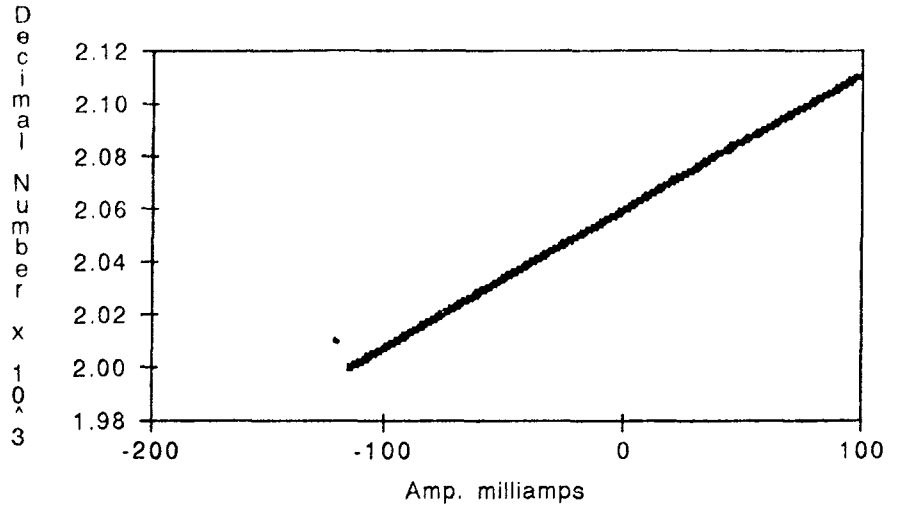
Variable Name	Coefficient	Std. Err. Estimate	t Statistic	Prob > t
Constant	2059.640	0.021	99786.133	0.000
D/A millivolts	0.205	0.000	1568.522	0.000

Data File: Dapper Dam

Source	Sum of Squares	Deg. of Freedom	Mean Squares	F-Ratio	Prob>F
Model	113954.951	1	113954.951	2460260.587	0.000
Error	5.049	109	0.046		
Total	113960.000	110			

Coefficient of Determination (R^2)	1.000
Adjusted Coefficient (R^2)	1.000
Coefficient of Correlation (R)	1.000
Standard Error of Estimate	0.215
Durbin-Watson Statistic	0.477





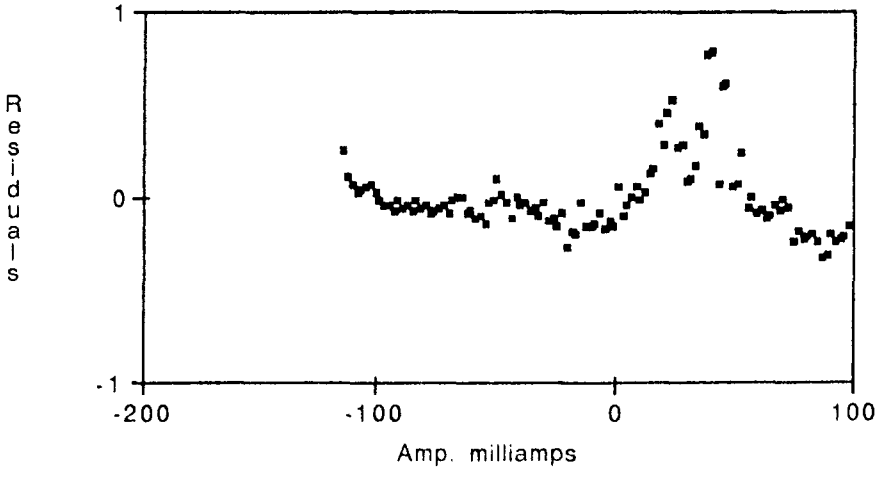
Data File: Dapper Dam Dependent Variable: Decimal Number

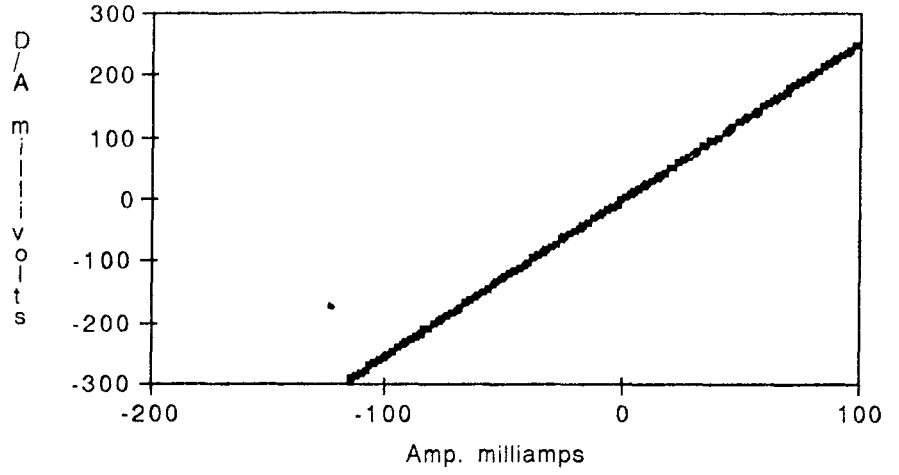
Variable Name	Coefficient	Std. Err. Estimate	t Statistic	Prob > t
Constant	2059.366	0.020	103697.562	0.000
Amp. milliamps	0.520	0.000	1628.307	0.000

Data File: Dapper Dam

Source	Sum of Squares	Deg. of Freedom	Mean Squares	F-Ratio	Prob>F
Model	113955.315	1	113955.315	2651384.249	0.000
Error	4.685	109	0.043		
Total	113960.000	110			

Coefficient of Determination (R^2)	1.000
Adjusted Coefficient (R^2)	1.000
Coefficient of Correlation (R)	1.000
Standard Error of Estimate	0.207
Durbin-Watson Statistic	0.457





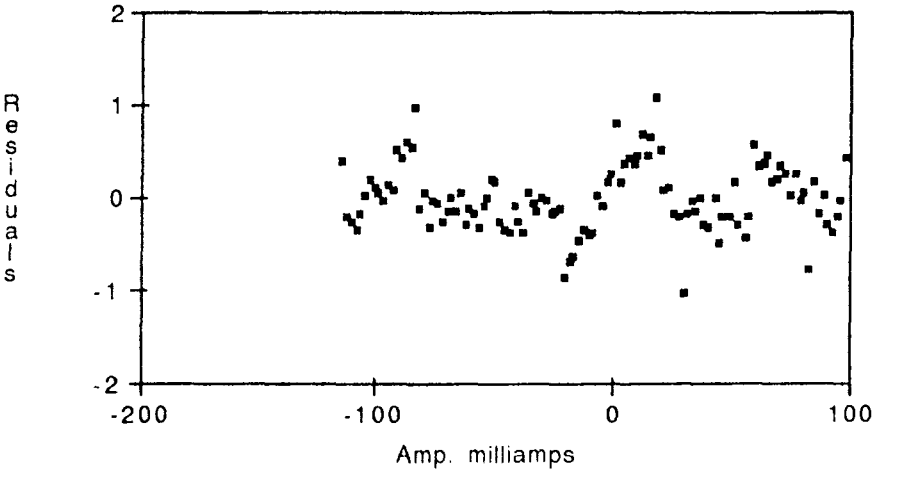
Data File: Dapper Dam Dependent Variable: D/A millivolts

Variable Name	Coefficient	Std. Err. Estimate	t Statistic	Prob > t
Constant	-1.335	0.034	-38.784	0.000
Amp. milliamps	2.538	0.001	4585.344	0.000

Data File: Dapper Dam

Source	Sum of Squares	Deg. of Freedom	Mean Squares	F-Ratio	Prob>F
Model	2715636.159	1	2715636.159	21025381.901	0.000
Error	14.078	109	0.129		
Total	2715650.237	110			

Coefficient of Determination (R^2)	1.000
Adjusted Coefficient (R^2)	1.000
Coefficient of Correlation (R)	1.000
Standard Error of Estimate	0.359
Durbin-Watson Statistic	0.808

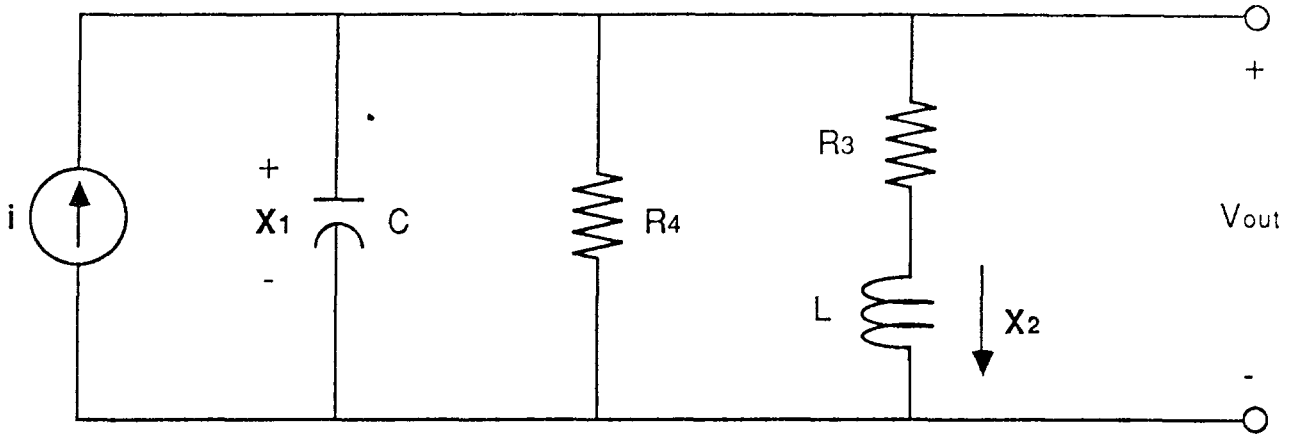


APPENDIX E

Root Locus Stability For Motor Amplifier

Determination of the value for a shunt resistor to improve the stability of the amplifier driving the finger.

Beginning with the equivalent circuit model,



where $R_4 = R_1 \| R_2$

we find from summing the currents at the top that

$$X_1 = \frac{-1}{CR_4} X_1 - \frac{1}{C} X_2 + \frac{1}{C} i$$

and from summing the voltages in the rightmost branch that

$$X_2 = \frac{-1}{L} X_1 - \frac{R_3}{L} X_2$$

and

$$y = V_{out} = X_1$$

which becomes just a 2x2 state space matrix on the next page.

$$\dot{\underline{X}} = \begin{bmatrix} \frac{-1}{CR4} & \frac{-1}{C} \\ \frac{1}{C} & \frac{-R3}{L} \end{bmatrix} \underline{X} + \begin{bmatrix} \frac{1}{C} \\ 0 \end{bmatrix} \underline{i} \qquad Y = \begin{bmatrix} 1, 0 \end{bmatrix} \underline{X}$$

now $Z = \frac{V_{out}}{i} = C\,(\,SI - A\,)^{-1}\,B$

thus $$Z = \frac{\frac{1}{C}\left(s + \frac{R_3}{L}\right)}{s^2 + s\left(\frac{R_3}{L} + \frac{1}{CR_4}\right) + \frac{R_3}{LCR_4} + \frac{1}{LC}}$$

and setting the characteristic equation equal to zero, with $Y_4 = \frac{1}{R_4}$, we have

$$s^2 + s\frac{R_3}{L} + \frac{1}{LC} + Y_4\left(s\frac{1}{C} + \frac{R_3}{LC}\right) = 0$$

which in root locus form becomes

$$1 + Y_4\left[\frac{\frac{1}{C}\left(s + \frac{R_3}{L}\right)}{s^2 + s\frac{R_3}{L} + \frac{1}{LC}}\right] = 0$$

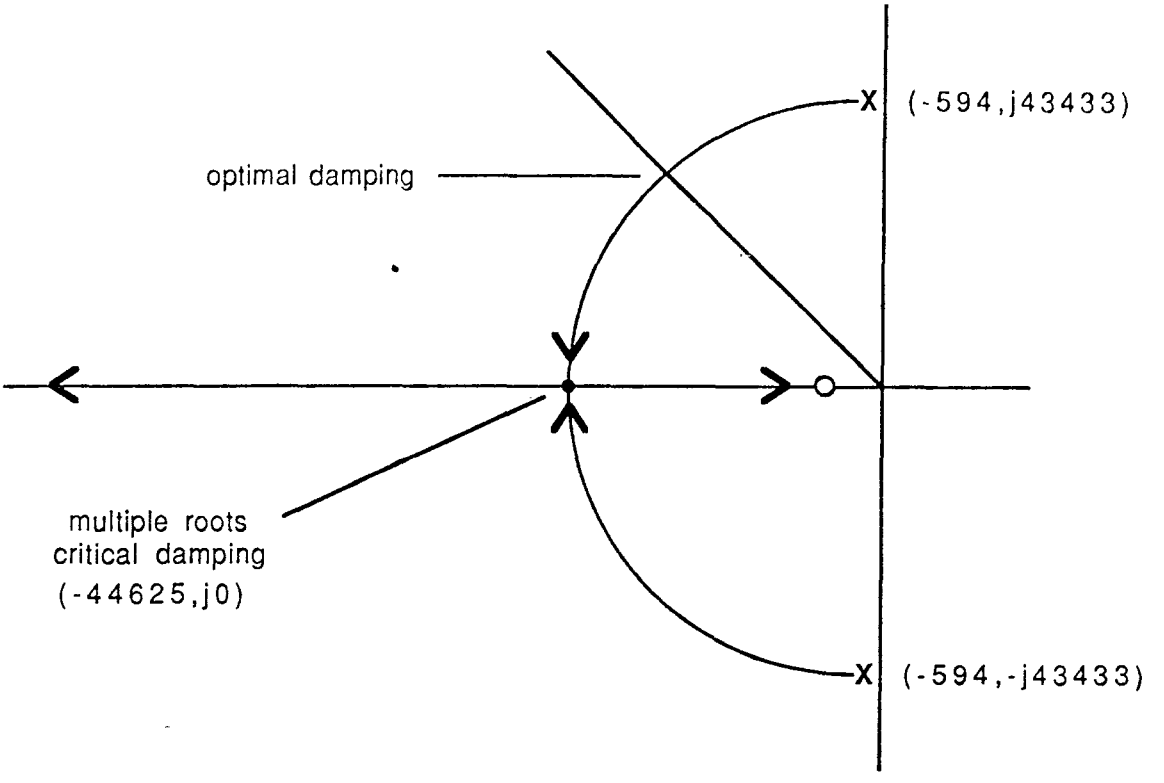
so in order to plot $1 + Y_4G(s) = 0$, we use

$$G(s) = \frac{1 \times 10^7\,(\,s + 1188.6792\,)}{s^2 + s\,1188.6792 + 1886792530}$$

which has poles at $s = -594.3396 + j43433.159$

and a zero at $s = -1188.6792$

So the entire root locus is then easily constructed.



Solving for multiple roots by setting $ba' - ab' = 0$, we find multiple roots at

$$s = 42248.564, \text{ and } -44625.922$$

where only the latter is on the root locus. So our only multiple roots occur at

$$s_o = -44625 + j0$$

Solving for critical damping at this point, s_o, we know that

$$|G(s_o)| = \frac{1 \times 10^7 \, (43437.243^2 + 0^2)^{\frac{1}{2}}}{44031.582^2 + 43433.159^2} = 113.5549$$

and so $Y_4 = \frac{1}{|G(s_o)|} = 0.0088063 = \frac{1}{R_4}$

Now since $R_4 = R_1 \parallel R_2$, where $R_1 = 36000$

simple math shows that $R_2 = 113.91422$ ohms provides critical damping.

Similarly, for optimal damping with a damping coefficient of 0.7 we have

$$s_1 = -31133.587 + j31133.587$$

which produces

$$|G(s_1)| = \frac{1 \times 10^7 \left(29944.908^2 + 31133.587^2 \right)^{\frac{1}{2}}}{\left(12299.572^2 + 30539.248^2 \right)^{\frac{1}{2}} \left(74566.746^2 + 30539.248^2 \right)^{\frac{1}{2}}}$$

$$= 162.8357 \text{ ohms}$$

and so $Y_4 = \frac{1}{|G(s_1)|} = 0.0061413 = \frac{1}{R_4}$

And we still have $R_4 = R_1 \parallel R_2$, where $R_1 = 36000$

So in solving we find $R_2 = 163.57134$ ohms provides optimal damping.

APPENDIX F

Miscellaneous Handouts and Literature

Appendix F references:

(1) Hewlett-Packard Technical Data sheet on "Two Channel Optical Incremental Encoder Module 11 mm Optical Radius (HEDS-9100 Series)", December 1986.

(2) Molex Technical Data sheet on KK-Products 2695/6471 Crimp Terminals and Housing.

(3) Communications, Automation & Control: "D3EMU Release 1.7" and "Hardware Reference Manual for the DSP32-PC" manuals.