NASA-CR-194248

ROLLER-GEAR DRIVES FOR ROBOTIC MANIPULATORS
DESIGN, FABRICATION AND TEST

FINAL REPORT - CONTRACT NAS 3-25803

SBIR- 05.03-1391A By Release 12-10-92

WILLIAM J. ANDERSON and WILLIAM SHIPITALO

September, 1991

NASTEC, INC. - Cleveland, Ohio

SBIR 1987 Phase II

NASA-CR-194248

ROLLER-GEAR DRIVES FOR ROBOTIC MANIPULATORS

DESIGN, FABRICATION AND TEST

FINAL REPORT

for

NASA LEWIS RESEARCH CENTER

CONTRACT NAS 3-25803

by

WILLIAM J. ANDERSON

and

WILLIAM SHIPITALO

September, 1991

SBIR 05-03-1391A Release 12-10-92

SBIR 1987 P-II

CORPORATE OFFICES:
1700 Ohio Savings Plaza
1801 East Ninth Street
Cleveland, Ohio 44114
216-696-5157

ENGINEERING & DESIGN CENTER
5310 West 161st Street
Suite G
Brook Park, Ohio 44142
216-433-1555

(NASA-CR-194248) ROLLER-GEAR DRIVES FOR ROBOTIC MANIPULATORS DESIGN, FABRICATION AND TEST Final Report (NASTEC) 250 p

N94-12802

Unclas

G3/37 0183135

TABLE OF CONTENTS

ACKNOWLEDGMENT

The authors would like to acknowledge the invaluable contribution of Mr. Richard Glassell of Glassell Enterprises, and of Professor Wyatt Newman, Department of Electrical Engineering and Applied Physics, Case-Western-Reserve University.

Mr. Glassell designed the software and control system for the Pitch-Yaw Joint drive. Professor Newman conducted the experimental evaluation of the Wrist-Roll and Hinge-Joint drives.

SUMMARY

Two single axis planetary roller-gear drives and a two axis roller-gear drive with dual inputs were designed for use as robotic transmissions. Each of the single axis drives is a two planet row, four planet arrangement with spur gears and compressively loaded cylindrical rollers acting in parallel. The two axis drive employs bevel gears and cone rollers acting in parallel.

The rollers serve a dual function: they remove backlash from the system, and they transmit torque when the gears are not fully engaged.

Specifications for the single axis drives, named the Wrist-Roll and Hinge-Joint were:

	WRIST-ROLL	HINGE-JOINT
Ratio	23.23:1	29.23:1
Input torque	1.12 Nm (9.9 in lb)	28.94 Nm (256 in lb)
Output torque	26 Nm (230 in lb)	845.8 Nm (7483 in lb)

Specifications for the two axis drive, named the Pitch/Yaw Joint Drive, were:

Ratio	3.4286:1
Dual input torques	27.1 Nm (240 in lb)
Output torque	186.5 Nm (1650 in lb)

Two Wrist-Roll Drives, one all steel and one containing alternate sets of steel and plastic rollers and gears, and one Hinge-Joint Drive, all steel, were fabricated, assembled and performance tested. Drive linearity, friction, inertia, backlash, stiffness and efficiency were determined with experimental setups employing high resolution optical encoders, precision torque meters, and dc servomotors with either proportional-plus-integral velocity or torque feedback control.

For all three single axis drives, linearity was good. Ratio variations of the two steel drives were typically within 0.5%. Ratio variations of the plastic drive were as high as 1.5%. No backlash was observed in either of the steel drives (within the 0.001 degree accuracy of the encoders). The plastic drive exhibited a small hysteresis, on the order of the measurement accuracy. Dynamic efficiencies of up to 97% were measured for the steel Wrist-Roll Drive; notably, it was more efficient as a speed increaser than as a speed reducer. Efficiency measurements of the plastic Wrist-Roll Drive were influenced by two failures that occurred during testing. Only quasi-static measurements of efficiency of the Hinge-Joint Drive were made, with a projected efficiency of 96% at full torque.

A drive and control system was designed and constructed for use in conjunction with the Pitch/Yaw Joint. The control system is PC based, and was designed to drive the joint, input specific commands to perform single or multiple defined movements, measure input and output speeds and angular positions, measure input torques, and record all of these data as functions of time for use in analyzing performance of the joint. To accomplish these tasks a system of drive motors, tachometers, torque meters and input and output resolvers was assembled.

Control system response was smooth and lag times between command and execution were typically 0.1 seconds. Execution times for 20^{o} to 30^{o} moves in pitch or yaw were typically 0.5 to 0.7 seconds, which translates to a pitch or yaw axis angular velocity of approximately 6.5 rpm.

Static torsional stiffness measurements were made of the Pitch/Yaw Joint. These revealed stiffnesses of only a few percent of the theoretical stiffness of the roller-gear train, indicating that structural elements are quite soft. A heavier structure could significantly increase torsional stiffness. Some difficulties in analyzing data were encountered because of inertia in the torque loading system. Problems with torque loading system instability limited input torques to 75% of full load rating. Efficiencies of up to 98% were measured. The joint was backlash free. Fluctuations in output axis angular velocities (either pitch or yaw) of $\pm 2\%$ were determined from differentiation of pitch and yaw axis position data, but limited resolver accuracy makes this data somewhat questionable. There may also have been some feedback from the dynamics of the torque loading system. A more meaningful assessment of joint linearity would require further work with a low (or preferably zero) inertia, damped loading system and more precise resolvers or encoders.

SYMBOLS

C	compliance rad/in lb (rad/Nm)
C_n	constants as defined
D	deflection mm(in)
E	modulus of elasticity psi, (GPa)
F	force, N (lb.)
I	bevel gear geometry factor
J	spur gear tooth geometry factor
K	stiffness in lb/rad (Nm/rad)
M	measurement over pins, in.
m_G	ratio of N_G/N_P
N	number of gear teeth normal load, N, (lbs.)
P	gear diametral pitch
P.D.	gear pitch diameter, mm (in.)
R	drive ratio roller radius
S	stress, GPa (psi)
T	tooth thickness, in. torque in lbs(Nm)
T_G	gear radial reaction force, N (lb.)
T_P	gear axial reaction force, N (lb.)
Y	Tooth deflection Lewis factor
W	load, N,(lbs)
BC	base circle diameter, in.
BP	basepitch

CD	center distance, in.
OD	outside diameter, mm (in.)
SAP	start of active profile
a	contact ellipse semi-major axis, mm (in.)
b	contact ellipse semi-minor axis, mm (in.)
a, x_1, x_2, y_1, c	roller or gear radii, mm (in.)
d	diameter, mm (in.)
f	gear face width, mm (in.)
r	radius, mm (in.)
α	bevel gear angle, deg.
η	efficiency
ω	angular velocity, rpm, rad/sec
α, γ, θ	roller or gear cluster angles, deg.
θ	angular deflection, rad
μ	traction coefficient
ϕ	gear tooth pressure angle

SUBSCRIPTS

A	axial load arm A, input A
B	bending load arm B, input B

C	compressive
a, x1, x2, y2, c	roller or gears
G	gear
N	normal
P	pinion; pitch
R	radial
S	separating; system
T	tangential
Y	yaw
ax1, y1x2, x2c	contact pair

INTRODUCTION

Traction drives offer advantages of high efficiency at high ratios, backlash free operation, low noise, no gear cogging, smooth torque transfer and back drivability. However, the maximum tangential force that can be transmitted through a roller contact is a function of the normal load and the available traction coefficient. Structural rigidity, materials, and lubrication mode limit normal loads and thus drive torque. Where space is at a premium and the torque to be transmitted exceeds the capability of a traction drive, alternative designs are required.

Roller-gear drives offer an alternative approach; they utilize torque transfer paths in which rollers and gears function in parallel. The gears enhance torque capacity. In planetary arrangements, retention of rollers acting in parallel with gears has two functions:

1. The rollers provide positioning for the gears and allow the use of multiple planet rows. In gear units without rollers, only single planet row arrangements are feasible;

2. Compressively loaded rollers remove backlash from the torque path, attenuate gear cogging and torque ripple, and support the radial component of gear tooth forces.

In non-planetary arrangements of bevel gears and bevel rollers the rollers supplement the torque capacity of the gears, remove the backlash from the torque path, attenuate gear cogging and torque ripple, and support the non-tangential components of gear tooth forces.

As in ordinary single row planetaries, in two row planetaries, the low torque high speed input is supplied to the sun roller-gear and the high torque, low speed output is taken from the ring gear. Roller diameters are nominally equal to gear pitch diameters with a preload set at assembly which exceeds the gear separating force. This insures that the rollers will not separate and cause backlash or chatter. The gear teeth will then transmit purely tangential forces.

Three row roller-gear drives can be designed in a fashion similar to two row drives where higher ratios are required. Bearings are necessary in only one of the planet roller-gear rows, usually the outermost row. Other planet roller-gears are spatially located by the rollers. Preloading of a roller-gear drive can be accomplished by employing ring rollers to compressively load the roller cluster, or by employing eccentric mounts for the planet rollers which are located on bearings.

Preliminary designs of three roller-gear drives were completed and reported in ref. 1 to assess their feasibility for use in a Laboratory Telerobotic Manipulator. These include a two axis drive which utilizes cone rollers and zerol bevel gears, and two single axis drives which utilize cylindrical rollers and spur gears.

Technical objectives of this effort were:

1. Complete detailed designs and manufacturing drawings for:

a. A nominal 25:1 ratio single axis roller-gear drive for the Wrist-Roll Joint. Design two versions of the drive - one incorporating steel gears and rollers and one incorporating combinations of plastic and steel gears and rollers.

b. A nominal 30:1 ratio high torque capacity single axis Hinge Joint roller-gear drive.

c. A 3.43:1 ratio two axis cone roller-bevel gear drive for the Large Pitch/Yaw Joint.

2. Seek manufacturing bids and have one complete assembly of each of the three drives fabricated. Fabrication will include a spare set of rollers for each drive and, for the Wrist-Roll Joint drive, two sets of gears and rollers - one steel and one plastic.

3. Assemble and check out drives for test.

4. Design test facilities for experimentally evaluating the three drives. The test facility for each drive will consist of an input drive system, a torque applicator on the output side, torque meters for monitoring input and output torques, a magnetic clutch and brake for initiating and stopping drive motion and encoders for measuring angular displacement. Appropriate shafting, couplings and a base to interconnect the components will complete each test facility. Complete manufacturing drawings for test facility components that are not standard commercial items. Include instrumentation for measurement and readout of input and output speeds, torques and angular positions.

5. Seek bids for and have fabricated non-commercial items for the test facilities.

6. Purchase all commercial items for the test facilities.

7. Assemble and check out test facility components with the drive prototypes.

8. Conduct an experimental evaluation of each of the roller-gear drives. Monitor input speed and/or angular position, and monitor and control output torque. Monitor input torque, output speed and input and output angular positions. These data will be taken over a range of input speeds and output torques. From these data drive torsional stiffness (in a locked mode), efficiency and torque ripple characteristics will be determined.

DRIVE DESIGNS

WRIST-ROLL JOINT ROLLER-GEAR DRIVE

DESIGN

Specifications for the Wrist-Roll drive were:

Nominal ratio	25:1
Output torque	26Nm (230 in. lb.)
Output speed	120 rpm
Maximum diameter	152.4 mm (6 inches)

Cluster and Gear Geometry

A cluster geometry was initially established in the Phase I study (ref. 1). When carried forward into detail design, it was found to present some assembly difficulties so a slightly modified design (Ratio = 23.23) was chosen from the solutions available for two planet row, four planet arrangements (ref. 1).

Two drive assemblies - one all steel, and one containing alternate steel and plastic rollers and gears are to be designed and fabricated.

Detail calculations of cluster geometry, gear stresses, gear geometry and measurement data are given in Appendix A. In the section Cluster Geometry, gear diametral pitch, pitch diameters, and numbers of teeth, together with nominal roller diameters are calculated. In the section Gear Stresses, gear face widths necessary to keep tooth bending and compressive stresses acceptable are calculated for both steel and plastic materials. In the section Gear Geometry and Measurement Data, tooth geometries necessary for manufacture and data required for inspection of the gears are developed. TABLE 1 summarizes gear geometry and inspection data for the WRIST-ROLL DRIVE.

Materials

Stress conditions in the steel gears are such that an alloy steel of Rc 30-38 is adequate. Special gear steels that are nitrided or carburized for operation at higher stress levels are not required. Sun, planet and ring rollers in the all steel drive are made of AMS-6490 M-50 tool steel. M-50 steel was chosen because the shrink fits required for roller mounting on the sun and first planet roller-gear assemblies require heating to 600° F. Fully hardened rollers (Rc 60) are required for the normal loads and Hertz stresses in the all steel drive. Bending stresses in the ring roller are nominal so that a carburized steel with a soft core is not required.

For the steel-plastic drive, Delrin 100 acetal resin polymer was chosen for the plastic gears because of extensive design and use experience in a wide array of gear applications. Procedures for designing Delrin 100 gears are given in reference 2. As in the all steel drive, the steel gears are an alloy steel of Rc 30-38, Torlon 4203 poly (amide-imide) resin was chosen for the plastic rollers. It has high strength over a wide temperature range (ref. 3). Steel rollers are of AISI 4340 alloy steel, Rc 39-42. Use of fully hardened rollers mating with poly (amide-imide) resin rollers is not necessary. Gear and roller material combinations for the steel-plastic Wrist-Roll Drive are shown in TABLE 2. TABLE 3 shows the gear and roller material contact combinations for the steel-plastic drive.

Roller Sizing

A number of iterative calculations of roller torque fraction, normal load and Hertzian contact were made to determine a feasible level of torque that could be carried through the rollers without extending roller widths much beyond .125 inches. A torque fraction of .20 was settled upon as reasonable. A traction coefficient of .06 was assumed for the Steel vs. Steel rollers (assuming traction oil or traction grease lubrication) and one of .15 was assumed for the Steel vs. Torlon rollers (assuming dry operation). Steel vs. Steel rollers were configured as straight cylinders with line contact except for the ring roller which was crowned for ease of assembly. Steel vs. Torlon rollers were configured as straight vs. crowned cylinders with elliptic contacts to eliminate edge effects with the torlon rollers.

Roller	Diameter, mm(in)	Load,N(lbs)	Width, mm(in)	Max Hertz Stress,GPa,(KSI)
a	14.2 (.558)	129 (29	3.18 (.125)	.52 (75.6)
x_1	30.1 (1.185)			
y	9.98 (.393)	703 (158)	3.30 (.130)	1.38 (201)
x_2	39.73 (1.564)			
y_2	40.28 (1.586)	1081 (243)	3.18 (.125)	0.57 (82)
c	110.1 (4.335)			

The Steel vs. Torlon roller data, for 20% torque transfer and u = .15 are as follows:

Roller	Diameter, mm(in)	Load,N(lbs)	Cross Radius, mm(in)	Contact Width mm(in)	Max Hertz Stress GPa(KSI)
a	14.2(.558)	26(5.85)		2.24(.088)	.08 (11.7)
x_1	30.1(1.185)		127(5)		
y	9.98(.393)	141(31.7)	76.2(3)	3.2(.126)	.17 (25)
x_2	39.73(1.564)				
y_2	40.28(1.586)	218(48.9)		2.69(.106)	.09 (12.8)
c	110.1(4.335)		63.5(2.5)		

Because of the low modulus of Torlon (4.83 GPa [700 KSI]) approximately 97 percent of the contact deformation occurs in the Torlon. Hertzian deformations in the roller contacts are as follows:

Roller	Steel vs. Steel, mm(in)	Steel vs. Torlon, mm(in)
a x_1	$7.87\text{x}10^{-4}$ ($3.1\text{x}10^{-5}$)	$6.86\text{x}10^{-3}$ ($2.7\text{x}10^{-4}$)
y_1 x_2	$3.35\text{x}10^{-3}$ ($1.32\text{x}10^{-4}$)	$2.44\text{x}10^{-2}$ ($9.6\text{x}10^{-4}$)
y_2c	$6.35\text{x}10^{-3}$ ($2.5\text{x}10^{-4}$)	$2.54\text{x}10^{-2}$ ($1\text{x}10^{-3}$)

Roller effective radii were calculated assuming all of the deformation to occur in the Torlon rollers for the Steel-Torlon drive.

The radial bending deflection of the ring roller was determined to be $2.79\text{x}10^{-3}$ mm (.00011 in) for the Steel-Torlon drive and $1.39\text{x}10^{-2}$ mm (.00055 in) for the Steel-Steel drive (ref. 4, p. 224). The operating diameter of the ring roller, after Hertzian deformation and bending deflection, should be the operating pitch diameter of gear c (110.14 mm (4.3361 in), APPENDIX A). Nominal diameters of the ring rollers were set at 110.11 mm (4.3351 in) for the Steel-Steel drive and 110.13 mm (4.3359 in) for the Steel-Torlon drive.

The theoretical diameter over the second row planet rollers before mounting the ring rollers is 110.157 mm (4.3369 in) for the Steel-Steel drive and 110.254 mm (4.3407 in) for the Steel-Torlon drive. Thus the interference fits to be overcome in mounting the ring rollers are

.0457 mm (.0018 in) for the Steel-Steel drive, and
.1219 mm (.0048 in) for the Steel-Torlon drive,

with a 63.5 mm (2.5 in) crown radius on the ring roller inside diameter, the crown drop (.03 mm (.0012 in)) is sufficient for assembly of the Steel-Steel drive. For the Steel-Torlon drive, the ring roller should be preheated to approximately 225° F.

Roller diameters and drawing sources for both the Steel-Steel and Steel-Torlon drives are given in TABLE 4.

Drawings

Design of the Steel-Steel drive is defined by assembly drawings NAS-140A and NAS-140B and detail drawings NAS-141 through NAS-181 inclusive. Commercial parts required for assembly are defined on these detail drawings. Design of the steel-plastic drive is defined by assembly drawings NAS-140AP and NAS-140B and the following detail drawings:

NAS-142P	NAS-148P	NAS-153
NAS-144	NAS-149P	NAS-154P
NAS-145	NAS-150	NAS-155
NAS-146P	NAS-151	NAS-156P
NAS-147	NAS-152P	

and NAS-157 through NAS-184 inclusive. Commercial parts required for assembly are defined on these detail drawings.

The assembly fixture required for assembly of both drives is shown on drawings NAS-185 and NAS-186.

WRIST ROLL ROLLER-GEAR DRIVE ASSEMBLY PROCEDURE
ALL STEEL DRIVE - ASSEMBLY DRAWING NAS-140A

1. Lay the Assembly Fixture, NAS-186, flat on a table.

2. Place the Sun Roller Gear Assembly, NAS-143, on the center post of the Assembly Fixture with the shaft extension inserted into the center post hole.

3. Using Dykem marking compound or equivalent, mark four equally spaced tooth spaces on the Sun Gear. Orient these four spaces to face the four equally spaced holes in the Assembly Fixture.

4. Highlight the indexed tooth on the large central gear of each of four 1st Row Roller Gear Assemblies, NAS-149.

5. Radially assemble a 1st Row Roller Gear Assembly with the Sun Roller Gear Assembly so that the marked indexed tooth goes into a marked tooth space. Index the Assembly Fixture on the table so that the shaft extension on the 1st Row Roller Gear Assembly can be inserted into an Assembly Pin, NAS-185.

6. Repeat steps for the remaining three 1st Row Roller Gear Assemblies. At this time the Assembly Fixture should be resting on the four Assembly Pin heads.

7. Press fit needle Roller Bearings, NAS-151, into each of four 2nd Row Gears, NAS-152.

8. Assemble one 2nd Row Roller, NAS-154, onto each 2nd Row Gear with Screws, NAS-153.

9. Assemble a key, NAS-170, two Bearing Races, NAS-164, and a Planet Shaft Spacer, NAS-163, onto each of four Planet Shafts, NAS-162.

10. Insert the Planet Shaft assemblies into the 2nd Row Gears and assemble these into each of four slots in the Assembly Fixture. Make certain that proper gear mesh and roller contact is made. Planet Shaft keys must properly fit into the key slots. The previously assembled 2nd Row Rollers should be laying on the side nearest the Assembly Fixture plate. It may be necessary to hold the Planet Shafts in the Assembly Fixture plate with nuts.

11. Assemble a Ring Support Washer, NAS-167, onto the top of each Planet Shaft assembly.

12. Assemble the Planet Shaft Ring Support, NAS-165, to the Planet Shafts and tighten with Ring Support Nuts, NAS-168.

13. Assemble two Ring Gear Dowels, NAS-157, and two Ring Gear Dowels, NAS-161, to the Ring Gear, NAS-155.

14. Assemble the Ring Gear to the four 2nd Row Gears from the top. This requires patience and deft maneuvering to obtain the proper mesh. The side with the shorter dowels should face down. With the Ring Gear in place, remove the Planet Shaft Ring Support.

15. The remaining four 2nd Row Rollers can now be assembled to the 2nd Row Gears. Reassemble the Planet Shaft Ring Support.

16. Assemble two Ring Rollers, NAS-156, to the 2nd Row Rollers and Ring Gear taking care to properly align dowels with dowel holes.

17. Assemble two Output Shaft Dowels, NAS-173, to the Output Shaft Adapter, NAS-171.

18. The Output Shaft Adapter, can now be assembled to the Ring Roller. Bolt the Adapter in place with Ring Gear Bolts, NAS-158, and Ring Gear Nuts, NAS-160.

19. Assemble the Output Ball Bearing, NAS-159, into the Main Housing, NAS-175. Clamp the bearing in place with Bearing Retainer Plate, NAS-179, and Flat Head Screws, NAS-174.

20. Remove the Assembly Fixture plate from the roller gear assembly by sliding it axially from the unclamped Planet Shaft extensions.

21. Slide the Main Housing and Bearing assembly over the Output Shaft Adapter, making sure that the bearing is properly seated on the Adapter. Clamp in place with Output Shaft, NAS-172, and Flat Head Screws, NAS-174.

22. Assemble Housing End Plate, NAS-176, and Main Housing Gasket, NAS-180, to Main Housing with Main Housing Screws, NAS-177, Planet Shaft Washers, NAS-166, and Planet Shaft Nuts, NAS-169.

23. Insert the Input Shaft Seal, NAS-178, into the Housing End Plate.

WRIST ROLL ROLLER-GEAR DRIVE ASSEMBLY PROCEDURE
STEEL-PLASTIC DRIVE - ASSEMBLY DRAWING NAS-140AP

1. Lay the Assembly Fixture, NAS-186, flat on a table.

2. Place the Sun Roller Gear Assembly, NAS-184, on the center post of the Assembly Fixture with the shaft extension inserted into the center post hole.

3. Using Dykem marking compound or equivalent, mark four equally spaced tooth spaces on the Sun Gear. Orient these four spaces to face the four equally spaced holes in the Assembly Fixture.

4. Highlight the indexed tooth on the large central gear of each of four 1st Row Roller Gear Assemblies, NAS-149P.

5. Radially assemble a 1st Row Roller Gear Assembly with the Sun Roller Gear Assembly so that the marked indexed tooth goes into a marked tooth space. Index the Assembly Fixture on the table so that the shaft extension on the 1st Row Roller Gear Assembly can be inserted into an Assembly Pin, NAS-185.

6. Repeat steps for the remaining three 1st Row Roller Gear Assemblies. At this time the Assembly Fixture should be resting on the four Assembly Pin heads.

7. Press fit needle Roller Bearings, NAS-151, into each of four 2nd Row Gears, NAS-152P. Check the .8752/.8747 diameter hubs for growth due to the bearing press and machine to size if necessary.

8. Assemble a key, NAS-170, two Bearing Races, NAS-164, and a Planet Shaft Spacer, NAS-163, onto each of four Planet Shafts, NAS-162.

9. Insert the Planet Shaft assemblies into the 2nd Row Gears and assemble these into each of four slots in the Assembly Fixture. Make certain that proper gear mesh and roller contact is made. Planet Shaft keys must properly fit into the key slots. Hold the 2nd Row Gears in place with an O-Ring or heavy rubber band stretched over their center recesses. It may be necessary to hold the Planet Shafts in the Assembly Fixture plate with nuts.

10. Assemble a Ring Support Washer, NAS-167, onto the top of each Planet Shaft assembly.

11. Assemble the Planet Shaft Ring Support, NAS-165, to the Planet Shafts and tighten with Ring Support Nuts, NAS-168.

12. Assemble two Ring Gear Dowels, NAS-157, and two Ring

Gear Dowels, NAS-161, to the Ring Gear, NAS-155.

13. Assemble the Ring Gear to the four 2nd Row Gears from the top. This requires patience and deft maneuvering to obtain the proper mesh. The side with the shorter dowels should face down. With the Ring Gear in place, remove the Planet Shaft Ring Support.

14. Four 2nd Row Rollers, NAS-154P, can now be assembled on the output side. Inward pressure on the roller-gear cluster by hand or with a heavy rubber band to compress the Sun-1st Row Planet Roller contacts may be necessary in order to be able to put the 2nd Row Rollers in place. The Torlon 4203 rollers are larger in diameter than their steel counterparts so the rollers must be compressed even before the Ring Rollers are put in place. Reassemble the Planet Shaft Ring Support.

15. Remove the Assembly Fixture plate from the roller gear assembly by sliding it axially from the unclamped Planet Shaft extensions. Assemble four 2nd Row Rollers on the input side. Apply inward pressure to the roller cluster, if necessary, to put the rollers in place.

16. Assemble two Ring Rollers, NAS-156P, to the 2nd Row Rollers and Ring Gear taking care to properly align dowels with dowel holes.

17. Assemble two Output Shaft Dowels, NAS-173, to the Output Shaft Adapter, NAS-171.

18. The Output Shaft Adapter, can now be assembled to the Ring Roller. Bolt the Adapter in place with Ring Gear Bolts, NAS-158, and Ring Gear Nuts, NAS-160.

19. Assemble the Output Ball Bearing, NAS-159, into the Main Housing, NAS-175. Clamp the bearing in place with Bearing Retainer Plate, NAS-179, and Flat Head Screws, NAS-174.

20. Slide the Main Housing and Bearing assembly over the Output Shaft Adapter, making sure that the bearing is properly seated on the Adapter. Clamp in place with Output Shaft, NAS-172, and Flat Head Screws, NAS-174.

21. Assemble Housing End Plate, NAS-176, and Main Housing Gasket, NAS-180, to Main Housing with Main Housing Screws, NAS-177, Planet Shaft Washers, NAS-166, and Planet Shaft Nuts, NAS-169.

22. Insert the Input Shaft Seal, NAS-178, into the Housing End Plate.

HINGE JOINT ROLLER-GEAR DRIVE

DESIGN

Specifications for the Hinge-Joint drive were:

Nominal ratio	30:1
Input torque	28.9 Nm (256 in. lb.)
Output speed	120 rpm
Maximum diameter	304.8 mm (12 in.)

Cluster and Gear Geometry

A cluster geometry with a somewhat larger toggle angle than that used in the Phase I study (ref.1) was chosen in an effort to meet the target diameter of 304.8 mm (12 in.). Detail calculations of cluster geometry, gear stresses, gear geometry and measurement data are given in Appendix B. In the section Cluster Geometry, gear diametral pitch, pitch diameters, and numbers of teeth, together with nominal roller diameters are calculated. In the section Gear Stresses, gear face widths necessary to keep tooth bending and compressive stresses acceptable are calculated. In the section Gear Geometry and Measurement Data, tooth geometries necessary for manufacture, and data required for inspection of the gears are developed.

TABLE 5 summarizes gear geometry and inspection data for the HINGE-JOINT DRIVE.

Materials

Stress conditions in the gears are such that nitrided or carburized steels are required for all of the external gears. An unhardened alloy steel is adequate for the ring gear. Nitralloy 135M with a .005/.008 in. case of Rc60-63 was chosen for all four external gears. SAE 4340 steel is adequate for the ring gear. Gear material data are summarized in TABLE 6. All rollers except the ring rollers are made of AMS-6490 vacuum melted M-50 steel through hardened to Rc 61-63. The ring rollers are made of AMS-6245 vacuum melted 9310 steel case carburized with a .030/.045 in. case of R_{15N} 89.5-91 hardness. Bending stresses in the ring rollers dictate that a carburized steel with a soft core be used rather than a through hardened steel. For 20% torque transfer through the rollers (discussed later in the section ROLLER SIZING) the maximum bending stress (at the inner fiber) is .15 GPa (22,100 psi), (ref.4, p.210). This level of reversed stress would make use of a through hardened steel risky.

The most critical aspect of gear fabrication is achievement of accurate timing of the x_1 and y_1 gears. Nitrided gears are not normally machined after nitriding because the process does not cause appreciable tooth distortions. The risk of small distortions, however, which might make assembly difficult or impossible, necessitated revisions in the fabrication procedure for the x_1 gear. The y_1 gear was finish cut and nitrided before mounting the x_1 gear blank. The x_1 gear was then rough cut with the teeth timed to the y_1 gear. Approximately 1 mil of stock was left on tooth faces. The x_1 gear was then nitrided with the y_1 gear masked off during nitriding. After nitriding, the x_1 gear was finish cut, again with teeth timed to the y_1 gear. Timing accuracies achieved are discussed in the section ASSEMBLY.

Roller Sizing

As with the WRIST-ROLL DRIVE a number of iterative calculations of roller torque fraction, normal load, and Hertzian contact were made to determine a feasible level of torque that could be carried through the rollers without extending roller widths much beyond .187 inches. At a torque fraction of .20 with an assumed traction coefficient of .06, Hertz stresses are acceptable with line contacts at ax_1 and y_1 x_2 and point contacts at y_2c. A cross radius of 30.48 mm (1.2 in.) is used on the ring roller inside diameter to produce a crown which assists in assembly. For 20% torque transfer through the rollers, loads, stresses and deformations are as follows:

Roller	Diameter mm(in.)	Width mm(in.)	Load N(lbs)	Max Hertz[1] Stress GPa(KSI)	Normal Approach[1] mm(in.)
a	21.97(.8649)	3.23(.127)			
			1099(247)	1.28(187)	.0122(.00048)
x_1	48.64(1.9151)	3.18(.125)			
y_1	18.33(.7217)	4.93(.173)			
			1776(399)	1.38(200)	.013(.00051)
x_2	91.65(3.6083)	4.82(.190)			
y_2	90.20(3.5511)	4.45(.175)			
			3164(711)	1.58(230)	.0213(.00084)
c	238.12(9.3749)	5.59(.220)			

(1) Ref. 4, p. 516.

Roller radii for manufacturing were determined by adding half the normal approach of a contact pair to each roller. The radial bending deflection of the ring roller was determined to be .0686 mm (.0027 in.) so its manufactured radius was corrected by that amount plus half the Hertzian deformation. Manufacturing diameters and drawing numbers for the rollers are shown in TABLE 7.

With each roller manufactured to its nominal diameter, the theoretical diameter over the second row planet rollers before assembly of the ring rollers is 238.19 mm (9.3774 in.) with the ring roller manufactured to 237.97 mm (9.3687 in.) the nominal radial interference to be overcome at assembly is .11 mm (.0044 in.). The crown resulting from a 30.48 mm (1.2 in.) cross radius on the ring roller is .13 mm (.0051 in.) so the ring rollers should be assemblable with little or no preheating.

Drawings

The Hinge-Joint Drive is defined by assembly drawing NAS-300A, and detail drawings NAS-301 through NAS-341 inclusive. These detail drawings include commercial parts required for assembly. The Assembly Fixture is defined by drawings NAS-342 and NAS-343.

ASSEMBLY NOTES

The first row planet gear and pinion timing errors received from the manufacturer were:

Part	Error, in.
S/N 1	.0005
S/N 2	.0005
S/N 3	.0035
S/N 4	.0005

Second row planet rollers were assembled with Nos. 1, 3, 5 and 7 on the input side and Nos. 2, 4, 6 and 8 on the output side.

No problems were encountered at my stage of assembly. The ring gear engaged the second row gears without forcing. The measurements over the second row rollers averaged 238.20 mm (9.3780 in.). Ring roller IDs were measured as 237.98 mm (9.3692 in.) resulting in an initial diametral interference of .22 mm (.0088 in.). Ring rollers were heated to approximately 250° F for ease of assembly.

HINGE-JOINT ROLLER-GEAR DRIVE ASSEMBLY PROCEDURE
ASSEMBLY DRAWING NAS-300A

1. Lay the Assembly Fixture, NAS-343, flat on a table.

2. Place the Sun Roller Gear Assembly, NAS-301, on the center post of the Assembly Fixture with the shaft extension inserted into the center post hole.

3. Using Dykem marking compound or equivalent, mark four equally spaced tooth spaces on the Sun Gear. Orient these four spaces to face the four equally spaced holes in the Assembly Fixture.

4. Highlight the indexed tooth on the large central gear of each of four 1st Row Roller Gear Assemblies, NAS-304.

5. Radially assemble a 1st Row Roller Gear Assembly with the Sun Roller Gear Assembly so that the marked indexed tooth goes into a marked tooth space. Index the Assembly Fixture on the table so that the shaft extension on the 1st Row Roller Gear Assembly can be inserted into an Assembly Pin, NAS-342.

6. Repeat steps for the remaining three 1st Row Roller Gear Assemblies. At this time the Assembly Fixture should be resting on the four Assembly Pin heads.

7. Press fit needle Roller Bearings, NAS-325, into each of four 2nd Row Gears, NAS-311.

8. Assemble one 2nd Row Roller, NAS-312, onto each 2nd Row Gear with Hex Head Screws #1, NAS-339.

9. Assemble a key, NAS-324, two Bearing Races, NAS-322, and a Planet Shaft Spacer, NAS-321, onto each of four Planet Shafts, NAS-319.

10. Insert the Planet Shaft assemblies into the 2nd Row Gears and assemble these into each of four slots in the Assembly Fixture. Make certain that proper gear mesh and roller contact is made. Planet Shaft keys must properly fit into the key slots. The previously assembled 2nd Row Rollers should be laying on the side nearest the Assembly Fixture plate. It may be necessary to hold the Planet Shafts in the Assembly Fixture plate with nuts.

11. Assemble a Ring Support Washer, NAS-323, onto the top of each Planet Shaft assembly.

12. Assemble the Planet Shaft Ring Support, NAS-327, to the Planet Shafts and tighten with Planet Shaft Nuts, NAS-326.

13. Assemble two Ring Gear Dowels #1, NAS-315, and two Ring Gear Dowels #2, NAS-316, to the Ring Gear, NAS-313.

14. Assemble the Ring Gear, to the four 2nd Row Gears from the top. This requires patience and deft maneuvering to obtain the proper mesh. The side with the shorter dowels should face down. With the Ring Gear in place, remove the Planet Shaft Ring Support.

15. The remaining four 2nd Row Rollers can now be assembled to the 2nd Row Gears. Reassemble the Planet Shaft Ring Support.

16. Assemble two Ring Rollers, NAS-314, to the 2nd Row Rollers and Ring Gear taking care to properly align dowels with dowel holes. It may be necessary to heat the Ring Rollers to be able to press them over the 2nd Row Rollers.

17. Assemble two Output Shaft Dowels, NAS-330, to the Output Shaft Adapter, NAS-328.

18. The Output Shaft Adapter can now be assembled to the Ring Roller. Bolt the Adapter in place with Ring Gear Bolts, NAS-317, and Ring Gear Nuts, NAS-318.

19. Assemble the Output Ball Bearing, NAS-331, into the Main Housing, NAS-334. Clamp the bearing in place with Bearing Retainer Plate, NAS-332, and Hex Head Screws, NAS-338.

20. Remove the Assembly Fixture plate from the roller gear assembly by sliding it axially from the unclamped Planet Shaft extensions.

21. Assemble "O" Ring #1, NAS-333, into Output Shaft, NAS-329.

22. Slide the Main Housing and Bearing assembly over the Output Shaft Adapter, making sure that the bearing is properly seated on the Adapter. Clamp in place with Output Shaft and Hex Head Screws #1, NAS-339.

23. Assemble Housing End Plate, NAS-335, and "O" Ring #2, NAS-337, to Main Housing with Hex Head Screws, #2, NAS-340, Planet Shaft Washers, NAS-320, Planet Shaft Nuts, NAS-326, "O" Rings #3, NAS-341, and Planet Shaft Keys, NAS-324.

24. Insert the Input Shaft Seal, NAS-336, into the Housing End Plate.

PITCH YAW JOINT ROLLER-GEAR DRIVE

DESIGN

Specifications for the Pitch Yaw Joint drive were:

Nominal ratio	3.43:1
Output torque	186.5 Nm (1650 in. lb.)
Dual input torques	27.1 Nm (240 in. lb.)
Maximum size	no larger than ORNL LTM Large P/Y unit

The Pitch Yaw Joint Roller-Gear Drive preliminary design was reported in ref. 1. Size constraints dictated that it fit into the same 15.24 x 15.24 cm (6 x 6 in.) cross section as the ORNL LTM Large P/Y unit.

Gear Geometry

Gear calculations were made to establish viable tooth design. They are presented in Appendix C with slight modifications of geometry factor J as obtained from the Gleason Corp. computer run. Gear dimensions, tooth stresses and reaction forces are shown in Table 8. Detailed geometries of the Zerol bevel gears obtained from the Gleason computer run are shown in Table 9.

Roller Geometry

Spatial constraints limited the width of the input cone roller face to approximately 9.525 mm (.375 in.) and the transversing bevel roller face to approximately 11.227 mm (.442 in.). It was decided to set roller loads so that they could transmit 20 percent of the torque with a design traction coefficient of 0.06.

For the input roller set:

Input torque to each cone roller bevel pinion = 27.1 Nm (240 in. lb.)

Cone roller pitch radius = 18.445 mm (.726 in.)

Cone half angle = 16.26°

Roller torque = $.726\ \mu\ F_N$

For 20 percent torque through rollers and $\mu = 0.06$, the normal force F_N is $F_N = (240)(.20)/(.726)(.06) = 1101$ lb. (4899N).

The required axial force, F_A, is then $F_A = 1101 \sin 16.26^{o} = 308$ lb. (1,371N).

For the transversing or output cone roller set:

Total output torque = 186 Nm (1646 in. lb.)

Torque through each roller-gear contact = 93 Nm (823 in. lb.)

Cone roller pitch radius = 46.83 mm (1.8438 in.)

Cone half angle = 45^{o}

Roller torque = $1.8438 \mu F_N$

For 20 percent torque through rollers and $\mu = 0.06$, the normal force F_N is

$F_N = (823)(.20)/(1.8438)(.06)$

$F_N = 1488$ lb. (6,621N)

The required axial force, F_A, on the pitch/yaw bevel roller is then

$F_A = (2)(1488) \sin 45^{o} = 2104$ lb. (9,364N)

Table 10 lists roller dimensions, loads, contact stresses, normal approach and drawing sources for the bevel rollers.

Materials

Stress conditions in the bevel gears dictate that the teeth be carburized and hardened, so AMS 6265F (AISI 9310) steel was chosen. All gears are carburized to have an effective case depth of .030 -.045 in. after finishing. The vacuum processed 9310 steel has excellent resistance to fatigue pitting, and with a specified core hardness of Rc 34-40, high strength in bending as well. Because of bending stresses as well as compressive stresses in the Hertzian contacts, AMS 6265F was also chosen for the bevel rollers. The specification for surface hardness, case depth and core hardness is the same as for the gears.

Drawing sources for the gears are given on Table 8 and for rollers on Table 10.

Drawings

Design of the Pitch/Yaw Roller-Gear Drive is defined by assembly drawing NAS-200, and detail drawings NAS-201 through NAS-240 inclusive. Commercial parts required for assembly are indicated by callouts 51 through 83 inclusive on drawing NAS-200.

Drawings NAS-241 and NAS-242 define the Output Bracket and Torque Arms which are used for loading the output.

PITCH/YAW JOINT ROLLER-GEAR DRIVE ASSEMBLY PROCEDURE
ASSEMBLY DRAWING NAS-200

As part of the assembly procedure it will be necessary to group several parts as "A" or "B" parts, with all "A" parts and all "B" parts used together. All such parts are to be clearly marked. The following parts will require coding as "A" or "B":

DESIGNATION ON DRAWING NAS-200	PART NO.	DESIGNATION ON DRAWING NAS-200	PART NO.
28	NAS-205	30	NAS-222
6	NAS-206	16	NAS-232
25	NAS-207	17	NAS-230
7	NAS-208	18	NAS-231
13	NAS-209	19	NAS-234
14	NAS-210	21	NAS-229
8	NAS-223	22	NAS-235
27	NAS-236	74	Needle Roller Thrust Washer
37	NAS-237	76	Pitch Shaft Ball Bearings
73	Intermediate Gear Ball Bearing	70	Input Gear Ball Bearing

on NAS-201 mark 2.250 Dia bores as "A" and "B"

It will be necessary to measure and record values of several dimensions on a number of parts as follows:

PART NO.	DIMENSION	VALUE A-SIDE		VALUE B-SIDE
NAS-201	6.000		6.004	
	3.000 REF.	3.004		
	2.875		2.8743	
	1.4375 REF.	1.4365		
	5.704 REF.		5.7042	
NAS-205A,B	1.624	1.622		1.622
	.375	.374		.374

	.336 .722 .593	.335 .722 .593	.3365 .722 .5943
NAS-209A,B	5.812	5.812 (.004B/L)	5.812 (.004B/L)
NAS-208A,B	2.3125	1A 2.3127 2A 2.3126	1B 2.3111 2B 2.3116
NAS-210A,B	5.690	1A 5.6883 2A 5.6876	1B 5.6866 2B 5.6875 3B 5.6878
NAS-206A,B	1.0310	1A 1.0268 2A 1.0281	1B 1.0300 2B 1.0300 3B 1.0303
NAS-207A,B	2.4375	2.4415	2.4425
NAS-203	3.034 1.562	3.034(.004B/L) 1.5615	
NAS-204	1.844	1.8429	1.8429
NAS-211	5.620	5.621	
NAS-238	5.745 .055 .477	5.746 .0548 .4768	
NAS-229	.935	.932	.932
NAS-220	.094	.094	
NAS-226	.094	.094	
NAS-202	1.094	1.095	
NAS-212	.100	.1015	
NAS-235	.625	.625	.625

1. Grind two Fitted Shims, NAS-236, to suit assembly of two sets of one each NAS-205 and NAS-207. Each shim thickness is determined as follows:

   Thickness = Value of 1.624 dim. on NAS-205 plus value of 1.4375 REF for A side or value of 2.875 dim. minus 1.4375 REF for B side on NAS-201, minus value of .593 dim. on NAS-205, minus value of 2.4375 dim. on NAS-207. Code mark each set as A and B.

2. Make two complete assemblies, A & B, of parts NAS-205, NAS-206, NAS-207, NAS-208, NAS-236 from step 1, Socket Head Cap Screws (part 62), dowels (parts 57 and 58) and bearings (part 73).

3. Determine proper distance D3 between needle thrust bearing faces on parts NAS-205A and B at assembly:

D3 = Value of 2.875 dim. on NAS-201 plus value of 1.624 dim. minus value of .336 dim. on NAS-205A, plus value of 1.624 dim. minus value of .336 dim. on NAS-205B. (Nominal value is 5.451")

4. Grind Thrust Washer, NAS-237 (part 37, Torrington TRC-3244) to the following thickness T4:

T4 = Value of 5.745 dim. on NAS-238 minus D3 minus part 74 thickness minus .1562".

5. Grind Spacers, NAS-223 (parts 8) to allow proper axial spacing of parts NAS-205. Determine S5A and S5B as follows:

Measure widths of Inner and Outer Rings of four parts 76 (543TA bearings). Nominal values are .281" for inner rings and .250" for outer rings. Calculate average IR width and average OR width.

S5A,B = 1.624 dim. minus .336 dim. minus .722 dim. for NAS-205A and B.

Each spacer width should be

W5A = 1/2 value of 5.620 dim. on NAS-211 minus value of 1.4375 dim. on NAS-201 minus S5A minus 3/2 $IR_{AV.}$ minus 1/2 $OR_{AV.}$

W5B = 1/2 value of 5.620 dim. on NAS-211 minus value of 2.875 dim. on NAS-201 plus value of 1.4375 dim. on NAS-201 minus S5B minus 3/2 $IR_{AV.}$ minus 1/2 $OR_{AV.}$

(Nominal value of W5 is .260")

6. Determine thickness of Shims, NAS-222 for proper axial spacing of assemblies from Step 2 as follows:

Measure axial distance from inner race face to outer race face on parts 73 (Kaydon bearings) as follows:

Place each bearing horizontally, resting on inner race face. Measure distance to outer race face under light thrust load. Each measurement should be approximately .375" less the bearing axial clearance. Mark bearings A and B. Calculate S6 = 1.624 dim. minus .375 dim. for NAS-205A and B.

Assemble Shims NAS-222 to thicknesses T6 as follows:

T6A = Value of 3.000 REF dim. on NAS-201 plus value of .094 dim. on NAS-226 minus bearing A width minus S6A minus 1.4375 REF dim. on NAS-201.

T6B = Value of 6.000 dim. minus 3.000 REF dim. on NAS-201 plus value of .094 dim. on NAS-220 minus bearing B width minus S6B minus 2.875 dim. plus 1.4375 REF dim. on NAS-201.

(Nominal value of T6 is .032")

7. The complete assembly of parts on the pitch axis can now be made, including the Yaw Shaft, NAS-202 and its Set Screw, NAS-224. Before the assemblies from Step 2 can be solidly mated with the Pitch Shaft, NAS-211 and Bearing Retainer, NAS-225, however, the Input Bevel Pinions, NAS-209, and Input Bevel Rollers, NAS-210, must be in place. Otherwise parts NAS-205 will interfere with entry of parts NAS-210.

Assembly on the Pitch Axis will consist of the following parts:

NAS-238	NAS-205	2 pcs	Parts 62	Socket Head Screws
	NAS-206	2 pcs	Parts 57	Dowels
	NAS-207	2 pcs	Parts 58	Dowels
	NAS-208	2 pcs	Part 74	Thrust Washer
	NAS-211	1 pc	Parts 75	Needle Thrust Bearings
	NAS-225	1 pc	Part 37	Thrust Washer
	NAS-202	1 pc	Parts 73	2 Kaydon bearings
	NAS-224	1 pc	Parts 76	4 Barden bearings
	NAS-236	2 pcs	Parts 56	Flat Head Screws
	NAS-223	2 pcs	Parts 59	Flat Head Screws
	NAS-222	2 pcs	Parts 78	Transducer
	NAS-220	1 pc	Part 61	Nut
	NAS-229	1 pc		
	NAS-226	1 pc		
	NAS-227	1 pc		
	NAS-228	1 pc		

8. Install four Parts 72, Needle Roller Bearings, in place in NAS-201.

9. Install Parts 70 (B 545 ball bearings) on NAS-209A and B. Put Parts 69 (Snap Rings) in place and measure the gap between bearing inner races and snap ring face. Grind NAS-231, 2 pcs, for a no-shake fit of the bearing inner ring on NAS-209A and B.

10. Grind two Spacers, NAS-232 to .093". Install two Parts 71 (Wave Springs) and parts NAS-232 in NAS-201.

11. Place NAS-210A and B in NAS-209A and B and install in

NAS-201.

12. After the Pitch Axis Assembly is solidly in place (Step 7), allow parts NAS-209A and B to mesh with NAS-205A and B. Without installing parts NAS-233, NAS-234 and Parts 68 (Belleville Springs) assemble parts NAS-235 with Parts 62 (socket head cap screws) and Parts 57 (dowels). Put Parts 67 (thrust washers) and Parts 66 (needle roller bearings) in place. Fit parts NAS-229 over thrust washers and measure the gap between parts NAS-229 and NAS-201 for both A and B sides. If pinions NAS-209 are solidly meshed with gears NAS-205, the gap should represent the zero backlash position. Axial withdrawal of pinion NAS-209 should result in backlash at the rate of .102 mils for each mil of withdrawal ($(\sin 16.26^{\circ}) \times (\tan 20^{\circ}) = .102$). Withdraw pinion NAS-209 to provide the level of backlash to which the teeth were cut as measured by pinion gaging (should be between .002 and .004 in.) Grind parts NAS-230 to the measured gaps plus the amount of withdrawal. Install parts NAS-230, NAS-239, and assemble parts NAS-235 with Parts 82 (socket head cap screws). Measure backlash between parts NAS-209 and NAS-205 and make appropriate adjustments. Input assembly is complete except for installation of parts NAS-233, NAS-234, Parts 68 (Belleville Springs) and Parts 65 (shoulder screws) which will load parts NAS-210 against NAS-206. When these parts are loaded, as per the schedule on drawing NAS-234, at a later assembly step, it may increase backlash between NAS-209 and NAS-205. It may then be necessary to adjust the thickness of NAS-230.

13. Assemble Parts 77 (duplex angular contact bearings) and parts NAS-214 and NAS-215 into NAS-203. Measure distance S13 from back face of NAS-203 to the face of the bearing. Grind NAS-217 to thickness T13 as follows:

T13 = S13 minus .100 dim. on NAS-212 minus .002 to .004

(This will allow .002/.004 clamping gap for the duplex bearing pair.)

14. Complete assembly of NAS-212, NAS-217, Parts 57 (dowels) and Parts 51 (socket head cap screws) to NAS-203.

15. Place the assembly from Step 14 with the face of NAS-212 supported on a horizontal surface. (Gear teeth on NAS-203 are up). Gently press down on the inner race of the upper bearing while turning it until the axial play is taken out of the bearing. Measure the distance, S-15, from the face of the bearing inner race to the back face of NAS-203. (This should be coincident with the upper face of NAS-212).

16. Grind NAS-216 to the following width W16P.

W16P = 3.034 dim. on NAS-203 minus S-15 minus 1.094

dim. on NAS-202.

Place NAS-216 on NAS-202 and follow with the assembly from Step 14. It may be necessary to freeze NAS-202 with dry ice and to warm the assembly from Step 14. Assemble Part 35 (locknut) onto NAS-202.

Measure the backlash between NAS-203 and NAS-207A and B with NAS-203 withdrawn from the mesh as much as possible by tugging on NAS-212 while the measurement is made. The backlash should be set at the level to which the teeth were cut as measured by gear gaging (should be between .002 and .004 in.). If the backlash needs to be reduced, for each mil reduction of backlash remove 3.89 mils from NAS-216 $(1/(\sin 45^{o})\ (\tan 20^{o}) = 3.89)$

W16F = W16P - 3.89 x reduction in backlash

Reassemble and recheck backlash between NAS-203 and NAS-207A and B. Make appropriate adjustments to NAS-216 to achieve desired backlash.

17. Assemble Part 78 (transducer) into NAS-240. It is assembled to NAS-212 with Parts 59 (flat head socket head cap screws).

Yaw assembly is now complete except for installation of parts NAS-218, NAS-219, Parts 54 (Belleville Springs) and Parts 53 (shoulder screws) which will load part NAS-204 against NAS-208A and B. When these parts are loaded, as per the schedule on drawing NAS-219 it may increase the backlash between NAS-203 and NAS-207A and B. It may then be necessary to adjust the width of NAS-216.

18. Remove NAS-212 from the assembly in Step 15.

19. The input rollers NAS-210A and B and the pitch/yaw roller NAS-204 can now be loaded against their mating rollers. For NAS-210A and B follow the load schedule on drawing NAS-234 for grinding that part for use in conjunction with NAS-233 and Parts 68 (Belleville Springs). For NAS-204 follow the load schedule on drawing NAS-219 for grinding that part for use in conjunction with NAS-218 and Parts 54 (Belleville Springs).

ASSEMBLY OF PITCH/YAW JOINT DRIVE

The following dimensional details were accumulated at assembly:

Step 1:

A side thickness = 1.622 + 1.4365 - .593 - 2.4415 = .024
B side thickness = 1.622 + 1.4378 - .5943 - 2.4415 = .024

Step 3:

D3 = 2.8743 + 1.622 - .335 + 1.622 - .3365
D3 = 5.4468

Step 4:

T4 = 5.746 - 5.4468 - .0615 - .1562 = .0815

Step 5:

Average inner ring width = .2804"
Average outer ring width = .2497"

W5A = (5.621/2) - (1.4365 + 1.622 - .722 - .335)
- 3/2 (.2804) - 1/2 (.2497)
W5A = .264
W5B = (5.621/2) - (2.8743 - 1.4365 + 1.622 - .7213 - .3365)
-3/2 (.2804) - 1/2 (.2497)
W5B = .2645

Spacers NAS-233 were made to .260/.255

Step 6:

Measured dimensions of Kaydon bearings

Bearing A: .3726"
Bearing B: .3713"

S6A = 1.622 - .374 = 1.248
S6B = 1.622 - .374 = 1.248
T6A = 3.004 + .094 - .3726 - 1.248 - 1.4365 = .0409"
T6B = 3.000 + .094 - .3713 - 1.248 + 1.4365 -2.8743 = .0369"

Step 8:

Before installing needle roller bearings, parts 70, into NAS-201, it was noted that the free O.D. of the bearings was .005" greater than the bore diameter. A check with Torrington revealed that up to .007" interference is permissible for this size bearing. The bearings pressed easily into place. The thin wall drawn cup does not behave as would a rigid shell. According to Torrington, if the radial thickness of the housing (even if it is aluminum) is at least equal to that of the bearing, most of the yield will take place in the drawn cup.

<u>Step 9:</u>

Spacers NAS-231 were ground to approximately .062" to provide no shake fits between bearings, parts 70, and retaining rings, parts 69, on NAS-209.

<u>Step 12:</u>

It proved to be impractical to try to measure the gap between parts NAS-229 and NAS-201 so individual dimensions in the stack up were measured to arrive at trial thicknesses S 12A, B for shims NAS-230. S 12A,B = 5.812 Dim on NAS-209 + .625 Dim on NAS-235 - 5.704 Dim on NAS-201 - .935 Dim on NAS-229 + Thrust Needle Bearing Stack (part 66 + 2 parts 67). Both the A and B side dimension sets were the same. The .935 Dim in NAS-229 less the bearing stack was measured as .666 on both A and B.

Therefore

S 12A,B = 5.812 + .625 - 5.704 - .666 = .067.

The A side was assembled with a .067" shim. The input pinion and gear mesh felt smooth. For the B side a .067" shim resulted in very tight mesh with obvious tooth ratcheting. It was gradually increased to .070". At that thickness both gear meshes had a similar feel.

<u>Step 13:</u>

Distance S 13 was measured to be .1605"
T 13 = .161 - .101 - .002
T 13 = .058
NAS-217 was ground to .058"

<u>Step 15:</u>

S 15 was measured to be 1.555"

<u>Step 16:</u>

W16P = 3.034 - 1.095 - 1.555 = .384"
NAS-216 was ground to .384"

The assembly from Step 14 went into place without any cooling or heating. The mesh of NAS-203 with the two gears NAS-207 was smooth. No further adjustment of NAS-216 was made.

Step 17:

Washers NAS-219 (8 total) were ground to .064 in. to produce 29% compression in Belleville Springs (parts 58 on NAS-200) and a total axial force on NAS-204 of 1037 pounds. Load vs. compression is given on drawing NAS-219.

Step 19:

Washers NAS-234 were ground to .045 in. to produce 35% compression in Belleville Springs (parts 68 on NAS-200) and a total axial force of 325 pounds on each input roller NAS-210. Load vs. compression is given on drawing NAS-234.

DRIVE AND CONTROL SYSTEM FOR PITCH/YAW JOINT

General Description

The objective was to design, construct and assemble a system which would: (1) drive the joint, (2) input specific commands to perform single or multiple defined movements, (3) measure relevant input speeds and angular positions, (4) measure input torques, (5) measure output angular positions, and (6) record all of these data as functions of time for use in and analyzing performance of the joint.

The system measures and records, at 4 millisecond intervals (250 HZ), the following variables:

(1) input speeds, TACH A and TACH B, rpm
(2) input torques, TORQUE A and TORQUE B, ft lbsf
(3) motor commands, A and B, volts
(4) Input Resolver positions, A and B, degrees
(5) Input Resolver Revolutions, A and B
(6) Output Pitch Position, degrees
(7) Output Yaw Position, degrees
(8) Output Pitch Velocity, rpm
(9) Output Yaw Velocity, rpm
(10) Output Desire Pitch position, degrees
(11) Output Desire Yaw position, degrees.

The control system is an IBM compatible PC based system. Other systems, such as the Macintosh, were considered, but the availability of the resolver decoders plus the simplicity of the MS-DOS operating system (important for real time operation) narrowed the choice to an IBM compatible. The system is flexible and can be re-programmed for future projects.

The hardware system is centered around three pieces of equipment: the computer, the joint, and an electronics board. The computer contains a resolver decoder card and a D/A-A/D I/O converter card. The electronics board contains the resolver reference signals, the torque sensors' reference signal, the torque output signal amplifiers, the motor tachometers' voltage divider circuit, and the PWM amplifiers for the motor drives.

The electronics board is a custom board that interfaces some of the sensors to the computer. The board is contained in a box that is set near the joint. Circuits are patterned after those that were proven on the LTM and the ORNL LeRC projects. The board has the following functions:

Motor tachometer maximum output voltage is +63 volts at maximum rpms. A voltage divider network is needed to reduce the voltage to +-10 volts for the A/D input.

The torque sensors require a +5 and -5 vdc signal for their reference signal. The circuit used is modeled after the circuit used by the LeRC electronics.

The torque sensors' output signal is very small, in the millivolt range. For the signal to be useful for the A/D's the signal needs to be amplified. Again, this circuit is modeled after the amplifiers used on the LeRC project.

The input and the output resolvers require reference drive signal for the primary winding of the resolvers. The resolvers operate at two different frequencies, therefore two separate circuits, one for the input resolver drive and one circuit for the output resolver drive, are provided.

The PWM amplifiers are mounted on the circuit board. Their control and outputs are wired to the board.

A +-15 or +-12 vdc power supply to drive the board and a 50v, 10 amp unregulated power supply for the PWM amplifiers are included.

Drawings

General arrangement of parts comprising the drive and control system is shown on assembly drawing NAS-200A. Design of machined parts is defined on detail drawings NAS-251 through NAS-266 inclusive. Commercial hardware required for assembly are defined by callouts 51 through 78 inclusive on drawing NAS-200A.

Electro-Mechanical System Components

The commercial components of the electro-mechanical system are as follows:

Input Side

1. Gear motors - Electrocraft (Robbins Myers)
   E652-MGHD, Part No. 652-006-205,
   complete with tachometer function.

2. Position indicators - Clifton Precision Products
   Brushless Resolver
   Model JSSBH-21-K-1

3. Torque meters - GSE Inc. Torque Sensor Model 8040.

Range, ± 50 ft lbf.

4. Brake - Magtrol electromagnetic friction brake, Model FB-130-101. De-energized torque, 0; energized torque, 1.27 Nm (180 oz. in.)

Output Side

1. Position indicators - Vernitron Corp. Transducer VRP 20-2

Computer and Electronics Board

Microlab 386 SX system with:

SIM Ram Module 256K
Intel CP 80387-SX Co-Processor
Toshiba 1.2 MB 5 1/4" Floppy
Western Digital 1:1 H/F Controller
386 SX Main Board
AT I/O Card
Seagate 40MB 65 ms Hard Disk
Focus 101 AT Keyboard
Star XR-1000 Printer

Electronics board with:

ILC Data Device Corporation SDC-36015 resolver decoder board (6 channel resolver to digital converter)
Metrabyte DAS-16 A/D-D/A card (16 A/D channels, two D/A channel plus eight I/O digital bits)
Advanced Motion Controls AMC-500 PWM motor drive amplifiers (used in current output mode)
ILC Data Device Corporation R/D Modules (7 units)

Wiring schematics for the various electronics board circuits are given on Figures 1-11:

1. Figure 1 - Power and Power Control
2. Figure 2 - Motor Power and Motor Brake Power
3. Figure 3 - Motor Tachometer Signals
4. Figure 4 - Torque Sensor Reference Signal
5. Figure 5 - Torque Sensor Signal
6. Figure 6 - Pitch and Yaw Resolver Reference Signal
7. Figure 7 - Motor Shaft Resolver Reference Signal
8. Figure 8 - Motor Shaft Resolver Position Signal
9. Figure 9 - Pitch and Yaw Resolver Position Signal

10. Figure 10 - Pitch, Yaw and Motor Shaft Resolver Velocity Signals
11. Figure 11 - Pitch, Yaw and Motor Resolver Signal Power Amplifiers

Software

Software for the system is written in Turbo C. The basic functions are to accept a commanded input then drive the joint to that location, collect and store joint data, monitor the joint condition, and shutdown the joint on an error. It is designed for easy operation interface and tuning of joint parameters. Files NASA 1.C through NASA 8.C inclusive, NASADATA.H, EXTDATA.H and NASADEF.H comprise the system. Hard copies of these files are in Appendix D.

Operating the Joint

The operation of the computer control panel is by menus and keystrokes. The control of the arm joint is completely done from the computer. The following is a guide to the operation and control to the computer and the control system. Power up:

1. Turn the computer on and allow the computer to perform self-diagnostics. This will take about 90 seconds. The printer (if connected) will also self-start. The computer is ready when the prompt C> appears.

2. At the prompt type cd tr . Press enter.

3. At the prompt type tr , the control screen will appear after a few seconds.

4. Turn on the power switch, the black toggle switch. The green light will come on. If the green light is dark, check that the power cord is plugged in, the fuses are good, or the cable from the switch box to the power box is plugged in.

5. Turn on the motor drive power by pushing the black button. The orange light will come on. To turn off the motor drive power, push the red button.

System Menu

"R"	Restart	- will initialize the control system.
"S"	Single Command	-to move the joint in a single motion. -computer will ask for the following parameters -Time time for motion -Pitch desired pitch angle -Yaw desired yaw angle
"M"	Multiple Command	-to move the joint in a path of multiple motions. -computer will ask for the following parameters -input motions can be entered by hand or from a disk file (the disk file is created from hand input). -motions the number of motions (0-10) in the path for each motion enter the following -Time time for that motion -Pitch desired pitch angle that motion -Yaw desired yaw angle that motion -save after the path is entered, the operator can save the path to disk file for later use (need a file name).

After the "S" and "M" commands have been completed, the path data can be saved to disk for later use. The data is collected automatically during the motion. If the answer to the question about saving the data is N (no), the data is lost. A file name is required.

"J"	Joystick Command	-joint operation with the joystick. -the joystick has a deadman and a return button. -Deadman releasing the deadman will lock the motor brakes, holding the deadman will release the brakes. -Return pushing the return button (at the rear of the stick)

will return control to the main menu.
-Control pitch and yaw motions with the joystick.

"N" Neutral Position

-will move the joint to neutral position.
-neutral position is determined from the setpoint data file.

"P" Parameters

-will display system data, system setpoints, and system gains.
-will display a short menu.
-"D" displays system data and allows the control of the motor brakes.
-"S" displays system setpoints and allows for the change of the setpoints. The setpoints determine the neutral position, high and low position setpoints.
-"G" displays system control loop gains and allows for the change of the gain values. The control loop only uses the motor velocity (motor tachs), position (pitch and yaw resolvers), and output velocity (resolver velocity). The other gains listed are not used but the variables have been defined.

When exiting the main control program and if any gain or setpoint values have been changed, the computer will ask if the gains and setpoints need to be saved. If the operator answers yes, the new values are saved. If the operator answers no, the new values are lost. During the program startup, the gains and setpoints are loaded from a disk file.

"E"

-exit the control program.
-note the motor drive power should be off before exiting the program.

Display

The program will display the pitch and yaw angles after a completed motion. The brake status is displayed. The condition window displays the operation, warning, or alarm conditions.

Data Processing

Data files are stored in the TR directory and can be hardcopied with a print command. Data is printed out in a four column format. Values of all sixteen variables are stored at 4 millisecond intervals so the volume of data from even a fractional second run is formidable.

In order to facilitate processing the data into convenient plots, MATLAB has been included in the software. The following procedure is used to make plots. At the prompts

```
>     cd         c:\matlab\wja  (changes directory)

>     copy       c:\tr\"data file name"
                 (copies data file into matlab\wja)

>     translat "data file name"  (The program translat
                 reconfigures the data file so matlab
                 can read it)

>     matlab     (into matlab)

>     makeplot   (matlab command to plot)

>     quit       (transfer to DOS directory)

>     hardcopy or finecopy torq   )  (DOS commands
                           tach   )       to
                           motor  )   print out
                           resolv )     plots)
                           py     )
                           pyv    )
```

TEST RESULTS

WRIST-ROLL AND HINGE-JOINT DRIVES

LINEARITY

Torque ripple is introduced by transmissions through an equivalent variable transmission ratio. For geared drives, the average ratio is fixed by the relative numbers of gear teeth. While this ratio must be a predictable constant, over the course of a cycle the apparent ratio can vary about this mean due to machining and assembly imperfections. Such variations introduce cyclic accelerations of the input and output, producing an apparent "torque ripple". In fact, the effect is a position ripple, and the resulting torque ripple depends on the dynamics of the drive as well as the dynamics of the loads attached to both the input and output. Thus, position ripple should be minimized, or, equivalently, drive linearity should be maximized to minimize resulting torque disturbances. Drive linearity measurements are reported here.

Experimental Procedure

To measure drive linearity and backlash, high-resolution optical incremental encoders were coupled to the input and output shafts of each of the drives. Corresponding input vs output rotations were recorded.

The shaft sensors used were BEI series 143 optical incremental encoders, which provided 360,000 counts per revolution. Each drive was tested with an encoder at the input and at the output. A computer program was written to sample the encoder angles and compute a ratio of input to output position increments for each 1 degree (1000 counts) of output (low-speed) rotation. Drive input was rotated slowly by hand during sampling in one direction, thus avoiding effects of backlash, mechanical dynamics, and sampling rate limitations. Input rotations were applied at minimum torque to minimize the influence of drive friction and stiffness on the position linearity tests.

The experimental procedure was evaluated using an "ideal" drive with unity ratio, i.e. a solid shaft. The two encoders were mounted identically to that of an actual drive measurement, and the same drive and sampling procedures were used. The equivalent incremental drive ratio was measured over each 1 degree of rotation, and the ratio proved to be unity within 0.1%, corresponding to plus or minus one encoder count. Thus, the measurement technique is believed to be accurate to 0.1%.

Data

Figure 12 shows the input angle vs output angle of the Hinge-Joint drive, measured over one rotation of the output.

The drive appears perfectly linear, at least within the resolution of the plot. To observe the small deviations from linear, the slope of input vs output, or the incremental drive ratio, was computed over each one degree of output and plotted vs output angle. Since the input vs output angles appear perfectly linear, only incremental drive ratios are displayed for the Steel Wrist-Roll and Plastic Wrist-Roll Drives.

Figure 13 shows the incremental drive ratio of the Steel Wrist-Roll drive measured over each 1 degree of output rotation, displayed over one full revolution of the output. The average drive ratio is 23.23, and the data shows incremental ratio variations as large as 0.5%. The experimental accuracy is 0.1%.

Figure 14 shows the same measurement for the Plastic Wrist-Roll drive. The average drive ratio was 23.23. Here, the incremental transmission ratio variations are 1.5% with a clear periodicity of approximately 33 degrees of output rotation. This data was taken before the two failures. Linearity after repairs was noticeably worse.

Figure 15 shows the incremental drive ratio of the Hinge-Joint drive. Its average ratio was 29.23. Typically, the drive ratio varied by less than 0.3%. However, there were four regions within the single revolution of the output in which the transmission ratio changed significantly, particularly in the region around 100 degrees output. Figure 16 displays a zoom-in of this worst-case region. The variation includes multiple consistent datapoints, which suggests that the variation is not attributable to measurement noise.

Analysis

For all three drives, linearity was good. The two Wrist-Roll drives had a ratio of 23.23, and the ratio of the Hinge-Joint drive was 29.23. The plastic Wrist-Roll drive had the greatest nonlinearity, with the ratio variations occurring periodically at about 1.5% every 33 degrees of output. Ratio variations of the two steel drives were typically within 0.5%, though the Hinge-Joint drive showed a spike of about 1.7% variation over about 3 degrees of output rotation. This variation may be attributable to a slip of a roller upon gear-tooth contact due to accumulated roller creep.

FRICTION

In addition to low backlash and high linearity, another potential advantage of roller/gear drives is low friction. Torque of the three drives was measured as both an unloaded speed reducer and as an unloaded speed increaser. For the Steel Wrist-Roll drive, tests were conducted running dry, then again after adding traction fluid.

For the plastic Wrist-Roll drive, the friction decreased noticeably with time (about 30%); the friction data reported was taken after approximately one hour of unloaded run-in near the rated speed. During this time, the input torque was monitored, and appeared to have stabilized at the values shown. Data shown was taken before the drive incurred a failure which necessitated repair.

The Hinge-Joint drive was driven both as a speed reducer and a speed increaser using the Steel Wrist-Roll drive to increase the motor drive torque. Torque measurements, however, were taken directly at the shaft of the Hinge-Joint drive. Use of the Wrist-Roll drive to increase motor torque, however, reduced the measured range of input velocities by a factor of 23. The Hinge-Joint drive thus did not experience as many cycles of run-in time as the Wrist-Roll drives. Further, all Hinge-Joint drive data was collected "dry".

Experimental Procedure

For all friction measurements, shaft torques were measured using Himmelstein model MCRT 2402T non-contact rotating torque transducers with rated 0.1% linearity. Two meters were used: a 50 in-lb range meter and a 350 in-lb range meter. Himmelstein strain-gauge amplifiers were used to produce analog torque signals. The two meters were calibrated relative to each other by coupling them in series, performing a complete friction test collecting data from both sensors, and computing the scale factor which produced correspondence of the two outputs. This calibration process thus included calibration of the analog-to-digital converters and the strain-gauge amplifiers as well as the torque meters.

For each datapoint of the friction measurements, the input was driven by a dc servomotor with proportional plus integral velocity control. For a complete dataset, the motor was commanded to successive velocities in increments of 20 rad/sec from 0 to 180 rad/sec, then back down in increments of -20 rad/sec, through zero to -180 rad/sec, then in increments of 20 rad/sec back to 0. At each new speed, the data was permitted to settle for 10 seconds, then data was sampled continuously over 15 seconds and averaged.

For friction measurements, the strain-guage amplifier low-pass filter frequency was set to 1Hz. Filtered analog data was sampled by 12-bit analog-to-digital converters at a rate of 1 kHz, and averaged over 15,000 samples for each datapoint.

Each resulting datapoint is highlighted on the graphs, with connecting lines illustrating the history of data collection.

Data

Figure 17 shows the steady-state torque required to drive the "input" (high-speed shaft) of the Steel Wrist-Roll drive with no load. This data was taken before adding traction fluid to the drive. Similarly, Figure 18 is the friction driving the Steel Wrist-Roll drive as an unloaded speed increaser. The data shows a significant Coulomb friction and a lesser viscous (speed-dependent) friction. Slopes of the friction curve about the origin should not be interpreted as an apparent saturating viscous effect. The slopes here are merely a graphical consequence of connecting discrete datapoints. In fact, the Coulomb friction effect extends down to zero velocity.

Friction tests on the Steel Wrist-Roll drive were repeated after adding traction fluid. The lubricated friction data is shown in Figure 19 and Figure 20 as a reducer and increaser, respectively.

Identical tests were performed on the Plastic Wrist-Roll drive, after a suitable run-in time. Friction data as a speed reducer and as a speed increaser are shown in Figure 21 and Figure 22, respectively.

Friction data for the Hinge-Joint drive was only taken over a speed range of -8 to +8 rad/sec, since the Steel Wrist-Roll drive was used as a speed reducer to drive it. Friction data as a reducer and an increaser is given in Figure 23 and Figure 24, respectively. Note that the range of input velocities as a speed reducer is severely limited, though the same range as a speed increaser corresponds to about 60% of the rated top speed.

Analysis

The drives exhibit a large Coulomb friction component. The Hinge-Joint and Plastic Wrist-Roll drives show virtually no viscous effect. The Steel Wrist-Roll drive does show a viscous effect. The Plastic Wrist-Roll drive showed friction levels comparable to the Steel Wrist-Roll drive, though they differ in friction vs speed dependence. The reducer friction and increaser friction are roughly the same ratio as the speed ratio, though not precisely so. Friction/speed hysteresis (friction depending on history of velocity excitation) is noticeable in the steel drives, though negligible in the plastic drive. The speed hysteresis may be due to variations in internal loading due to inertial transients when changing between successive steady state velocities.

INERTIA

Inertia was measured for the Steel Wrist-Roll drive only. Input (high-speed shaft) and output (low-speed) shaft inertia were measured separately, though accuracy of the input inertia measurement was poor.

Experimental Procedure

The Steel Wrist-Roll drive (test drive) was coupled to a dc servomotor through a 50 in-lb reactionless torque meter. The servomotor was controlled with acceleration feedforward and proportional-plus-derivative feedback to produce a smooth, specified sinusoidal motion. Over each sinusoid, the torque meter and the drive angle were sampled at 1kHz. One hundred of these sinusoids were sampled synchronously and averaged together to reduce noise. The 100-sample average, though, was hardly different from that of a single-pass, implying that the noise content was low.

The torque meter was low-pass filtered in analog at 100Hz before sampling. To correct the measurements for inertia of the couplings and the torque meter itself, the same data acquisition routine was run on the system with the test drive removed. The inertia of the measurement system was thus derived.

The test drive was driven both as an unloaded speed reducer to obtain input inertia and as an unloaded speed increaser to obtain output inertia.

Data

The measurement system alone (motor, torque meter and couplings) was driven sinusoidally with a magnitude of 10 radians and a frequency of 2 Hz. The apparent inertia seen by the torque meter (one coupling and 1/2 of the meter's own inertia) was measured to be 0.000255 Kg - m^2. The same amplitude and frequency were imposed with the high-speed shaft of the test drive coupled to the system. The apparent inertia increased by less than 25%. Thus, the measurement of input inertia had poor accuracy, since the inertial load was dominated by the inherent inertia of the measurement system itself. Nonetheless, the input inertia of the test drive could be estimated at 0.00006 Kg - m^2 (600 gm -cm^2).

Measurement of the output inertia was more accurate. The low-speed shaft was driven sinusoidally with an amplitude of 1.0 radians at a frequency of 2.5Hz, and the position and drive torque were sampled at 1kHz over 100 cycles. The averaged data is shown in Figure 25.

Analysis

The data shown in Figure 25 was curve-fit to pure sinusoids, resulting in a torque signal of 4.4 N-m amplitude, 2.5Hz, leading the position signal by 140 degrees. If the torque signal were due entirely to inertial effects, then it would have led the position signal by 180 degrees. The actual phase shift implies that the inertial effect has a magnitude of 3.35 N-m. In addition, there is an apparent viscous effect with a magnitude of 2.85 N-m in-phase with the velocity. The chosen excitation produces an angular acceleration of 247 rad/sec^2. Thus, the inertial torque of 3.35N-m implies an apparent test drive output inertia of 0.0135 Kg - m^2. The influence of the measurement system inertia compared to the true load inertia is less than 2%.

Ideally, the output inertia should be equal to the input inertia times the drive ratio squared. The measured input inertia of 0.00006 Kg - m^2 times the drive ratio of 23.23 squared is 0.033 Kg - m^2, which is roughly 2.5 times greater than the measured output inertia. The discrepancy is primarily due to the poor resolution of the input inertia measurement. Based on the more trustworthy output inertia measurement, the equivalent input inertia should be 0.000025 Kg - m^2, or 250 gm - cm^2.

Although the input inertia measurement had a relatively low accuracy, it should have been more consistent with the output inertia. It is possible that nonlinear friction terms, which cannot be accounted for in terms of phase-shifted sinusoids, influenced the measurements more than expected. Thus, while the output inertia has been measured more accurately than the input inertia, the derived value of the output inertia should not be considered precise.

BACKLASH

An unusual feature of roller/gear drives is that, ideally, they exhibit no backlash. Since the rollers always maintain rolling contact, there should be no deadzone between input and output angles.

Experimental Procedure

This measurement used the same experimental setup as used for the tests for LINEARITY. High-resolution encoders were coupled to both the input and the output of each transmission. The input shaft was rotated by hand forward, reverse, forward, and reverse again through an angle corresponding to 0.5 degrees of output rotation. The encoders were sampled at a high rate, and input and output angles were recorded for each 0.050 degrees of output rotation. Using the prescribed cyclic input, backlash in the drives would appear as a hysteresis in the input vs. output angle plots.

Data

Input vs. output angle of the Steel Wrist-Roll drive is shown in Figure 26. The negative slope is due to the arrangement of the encoders in opposition. The measured datapoints retrace themselves almost identically; any possible hysteresis is smaller then the measurement accuracy of 0.001 degrees of output angle. Similar plots are given in Figure 27 and Figure 28 for the Plastic Wrist-Roll drive and the Hinge-Joint drive, respectively.

Analysis

At least within the accuracy limitations of the high-resolution encoders (0.001 degrees), no backlash could be observed in either of the steel drives. The plastic drives might have exhibited a small hysteresis, but the effect is hardly larger than the measurement accuracy. This experiment was performed on the drive after it was damaged and re-worked.

STIFFNESS

Stiffness measurements were made only of the Steel Wrist-Roll drive. Stiffness measurements on the plastic Wrist-Roll drive were not performed prior to the failure, and the stiffness of the Hinge-Joint drive exceeded the stiffness of the couplings connecting the transmission input to the torque meter.

Experimental Procedure

The high-torque shaft of the Steel Wrist-Roll drive was held stationary (secured to ground) while the input was torqued through the 50 in-lbf reactionless torque meter. Torques were applied manually, held steady by observing the torque-meter output. The input torque and twist angle were sampled in the steady state (static torque balance) at 1kHz over 15 seconds, and the mean values were recorded. Eleven datapoints were recorded using input torque values of 0, 0.5 and 1.0 N-m, imposed alternately.

The same test was repeated on the torque meter and coupling alone, with the test drive removed, to isolate the influence of the measurement system stiffness.

Data

The measurement system exhibited a nonlinear stiffness, which increased from a value of 190 N-m/rad at low torques, to about 300 N-m/rad at torque levels near the transmission's rated maximum input torque, to nearly 400 N-m/rad at the maximum torque input of the meter.

With the measurement system in series with the test drive, the measured input stiffness was also nonlinear. An average stiffness up to 50% of maximum rated torque was approximately 30 N-m/rad. A representative (incremental) input stiffness for loadings from 50% to 100% of the rated torque was 40 N-m/rad.

A similar experiment on the Hinge-Joint drive yielded data which only demonstrated that the transmission stiffness was significantly larger than the stiffness of the measurement system.

Analysis

Since the measured stiffness with the Steel Wrist-Roll drive was an order of magnitude lower than the stiffness of the measurement system, the experiment should be valid. Thus, the Steel Wrist-Roll drive has an input stiffness of 30 to 40 N-m/rad (roughly 300 in-lbf/rad). Its input stiffness varies with the applied torque, behaving like a stiffening spring.

The stiffness of the Hinge-Joint drive could not be measured. It could only be proven that this transmission was significantly stiffer then the measurement system, which was approximately 200 N-m/rad.

EFFICIENCY

Efficiency as a function of speed and torque was measured on the two Wrist-Roll drives. For the Steel Wrist-Roll drive, efficiencies were measured both dry and with traction fluid added. Efficiencies were measured up to the rated torque, though only up to 60% of the rated speed due to equipment limitations. For the Hinge-Joint drive, only static measurements were performed.

Experimental Procedure

To measure efficiencies of the Wrist-Roll drives, the two drives were coupled together at their high-torque shafts through a rotating, 350 in-lb range torque meter. The high-speed shafts of each drive were driven by dc servomotors. The drive under test was coupled to its drive motor through a rotating, 50 in-lb range torque meter. All devices were coupled using Thomas miniature flexible disc couplings, which provide relatively high torsional stiffness along the shaft axis and compliance along all five remaining degrees of freedom. Figure 29 shows the test setup.

The high-speed shaft of the drive under test was controlled to run at specified speeds from -180 to 180 rad/sec. As in the friction experiments, each datapoint was obtained by controlling the drive speed through a proportional-plus-integral speed controller, waiting 10 seconds for settling after each new speed command, and sampling torque and speed data for 15 seconds during steady-state conditions. Both torque meters were low-pass filtered in analog at 1Hz and digitally sampled at 1kHz to obtain a 15,000-point average for each datapoint.

To produce a desired load torque, the second drive and servomotor were controlled in a proportional-plus-integral torque feedback loop, based on the measured output torque of the drive under test. The controlled torque source achieved accurate steady-state torque loads at all input speeds.

Each dataset was obtained at a fixed "output" (low-speed) torque while the "input" (high-speed) velocity was incremented through a range of values. As in the friction experiments, the data was collected starting from rest, incrementing in steps of 20 rad/sec up to 180 rad/sec, then retracing the positive-velocity measurements in speed decrements of 20 rad/sec, then continuing through zero down to -180 rad/sec, and finally retracing the negative velocity measurements in increments of 20 rad/sec back to rest. Individual datapoints are highlighted, showing repeatability and/or hysteresis in the measurements.

Load torques were held constant in each dataset at six values in equal increments from 0 to 25 N-m. Note that a positive "output load" in combination with a negative "input velocity" corresponds to power flow in the "reverse" direction. That is, at negative velocities the torque source at the low-speed shaft acts as a power source, and the velocity controller at the high-speed side acts as a controlled brake which sinks power. In this regime, the drive may be thought of as acting as a speed increaser rather than a speed reducer. This type of measurement is somewhat unusual, since drives are seldom measured with active torque sources. Additional measurements were performed to validate the procedure, described further below.

Figure 30 shows the measured output torque vs input velocity for each of the datapoints obtained. This plot documents the effectiveness of the controlled velocity and torque sources. Each line includes 37 datapoints, each consisting of 15,000 torque and velocity samples at steady states. Typically, only 19 datapoints per line can be observed, since the controlled states retrace themselves almost identically (with some exceptions at zero velocity). Figure 30 is representative of the states sampled for each of the Wrist-Roll drive cases: Steel running dry; Steel with traction fluid and Plastic.

Figure 31 shows the output power (positive for power out of the low-speed shaft, negative for power into the low-speed shaft) corresponding to the states of Figure 30. Since these controlled power flows are typical for all Wrist-Roll drive measurements, the plots are not repeated for each case.

Dynamic efficiency of the Hinge-Joint drive was not measured. Instead, the output (high) torque was measured using a 10,000 in-lb Himmelstein reaction torque meter, model 2060. The input (low) torque was excited using the Steel Wrist-Roll drive coupled through the 350 in-lb Himmelstein reactionless torque meter. The input to the Steel Wrist-Roll drive was driven by a dc servomotor which was controlled in a proportional-plus-integral torque feedback loop with respect to the measured input torque of the Hinge-Joint drive. Thus, the Steel Wrist-Roll drive acted as a torque source to the Hinge-Joint drive. The test setup is shown in Figure 32.

The controlled torque source was stepped through three cycles of increasing and decreasing input torques, ranging from -25 N-m to +25 N-m in steps of 5 N-m. For each step, the torque source was slowly ramped between successive torque commands in order to avoid torque "overshoot". Thus, the torque data was collected with a known history of loading to preserve hysteresis information. In the three cycles of torque loading, 61 datapoints were recorded. As a quasi-static torque source, the control over input torque was not as precise as under dynamic conditions. Nonetheless, the loading was smooth and reasonably reproducible, as demonstrated in the data plots.

The torque controller was given 15 seconds to settle between successive torque commands, then data was sampled at 1kHz over 10 seconds and averaged. The 10,000 in-lb reaction torque meter at the output was analog low-pass filtered at 1Hz. The 350 in-lb reactionless torque meter at the input was analog low-pass filtered at 100Hz. This filter frequency was required to obtain adequate bandwidth of the torque source feedback controller. The strain-guage amplifier used was not matched (recalibrated) to the 10,000 in-lb reaction torque transducer. Thus, the absolute accuracy of the output torque data is not guaranteed to be as precise as the input torque data. However, the 0.1% linearity specification still applies, so differential measurements can be compared with high accuracy.

Data

Each of the dataplots for the Wrist-Roll drive cases include 222 datapoints, each of which consists of a 15,000-sample average under steady-state conditions. Each datapoint is marked, and lines are drawn between successive samples, illustrating the history of the data collection and possible hysteresis. Measurements at zero speed are accurate, though they show less reproducibility due to static friction. Efficiency plots do not include the zero-velocity datapoints, as efficiency is undefined at zero speed.

Since both speed and output torque are controlled (as shown in Figure 30), characterization is completed by a measurement of the corresponding input torques. Figure 33 shows the input torques measured for the Steel Wrist-Roll drive running dry. Two datapoints on this plot, corresponding to a load of 17.3 N-m and speeds of -160 and -180 rad/sec, were inadvertently recorded at load torques slightly lower than the desired 17.3 N-m. The efficiencies corresponding to these states have been corrected for the true loading conditions. The input power corresponding to Figure 33 is shown in Figure 34. Note that the direction of power flow at negative velocities is negative, corresponding to the velocity source acting as a brake. Efficiency for the Steel Wrist-Roll drive, running dry, is shown in Figure 35. In this and subsequent efficiency plots, the efficiency is computed as the ratio of power extracted from the transmission to the power delivered to the transmission. For positive velocities, the efficiency is the low-speed power over the high-speed power, while for negative velocities the efficiency is the high-speed power over the low-speed power. The slope of the efficiency curves through zero velocity is a graphical consequence of connecting successive datapoints, and it should not be interpreted as a valid efficiency derivative near zero velocity.

Efficiency measurements were repeated for the Steel Wrist-Roll drive after adding traction fluid. The resulting input (high-speed) torque, input power, and efficiency are shown in Figure 36, Figure 37 and Figure 38, respectively.

During efficiency measurements on the Plastic Wrist-Roll drive, two failures occurred. At higher torques, the "sun" gear broke its Loctite bond on the input (high-speed) drive shaft. The gear was subsequently pinned in place. On the second efficiency measurement attempt, an internal locknut came loose, damaging gear teeth and rollers. Repairs were made with available replacement parts, but a noticeable distortion remained in the re-worked drive. All efficiency measurements reported on the Plastic Wrist-Roll drive were obtained from the re-worked unit.

Input torques and input power for the Plastic Wrist-Roll drive are shown in Figure 39 and Figure 40, respectively. Note from Figure 40 it can be seen that the lowest two output torques (4.2 and 8.3 N-m) were not sufficient to operate the drive as a speed increaser. To obtain measurements under these loading conditions, the controlled velocity source at the high-speed shaft had to supply power to the drive in addition to the power supplied by the torque source at the low-speed shaft. Thus, efficiencies as a speed increaser at these torques are actually negative, since the drive absorbs power from both ports.

Efficiency of the Plastic Wrist-Roll drive is shown in Figure 41. Data corresponding to the lowest 2 loading conditions has not been displayed, since these loads include negative efficiencies.

Dynamic efficiency of the Hinge-Joint drive was not measured. Only reaction torques were obtained. Figure 42 shows the set of input torques applied to the Hinge-Joint drive. This data is displayed vs the angle of the drive motor at the input of the Steel Wrist-Roll drive, which was used to amplify torque inputs to drive the Hinge-Joint input. In applying plus and minus 25 N-m of torque to the Hinge-Joint drive, the Steel Wrist-Roll drive required an input rotation in excess of one full revolution. Stiffness measurements showed that this wind-up is primarily due to the reflected compliance of the mechanical couplings driving the Hinge-Joint input shaft. Figure 43 shows the Hinge-Joint drive output torques corresponding to the inputs of Figure 42. Note in both cases that 61 datapoints taken over three cycles of loading are highly reproducible, except for the initial loading application. The clear hysteresis is not attributable to the instrumentation, nor to any friction effects outside of the Hinge-Joint drive.

Input torque vs output torque is plotted in Figure 44. The plot superimposes all three cycles of loading, which are virtually identical.

Efficiency Measurement Validation

A surprising result of the Steel Wrist-Roll drive efficiency measurements is that this efficiency is actually higher using the transmission as a speed increaser (power flow into the low-speed shaft) than as a speed reducer (power flow into the high-speed shaft). A potential cause of the unexpectedly high reverse efficiency is the unusual measurement technique, in which the low-speed shaft is driven by an active torque source. Ordinarily, the "load" torque is exclusively passive, such as a friction brake. In the present system, a controlled torque is maintained either as a power source or as a power sink.

To test the validity of the measurement technique, the procedure was inverted: the low-speed shaft was driven with a velocity source, and the high speed shaft was driven with a torque source. A complete dataset was obtained for the lubricated Steel Wrist-Roll drive in this reverse test mode.

Figure 45 shows the measured input torques at velocities ranging from -160 rad/sec to +160 rad/sec. The corresponding output torques are shown in Figure 46. Figure 47 and Figure 48 show the input and output power, respectively. Figure 49 shows the resulting efficiency plot.

Figure 49 should not be identical to Figure 38, since the individual datasets were taken under different loading conditions (controlled high-speed torques vs controlled low-speed torques, respectively). Nonetheless, the two graphs are quite similar, and clearly support the evidence that the Steel Wrist-Roll drive is more efficient as a speed increaser.

Analysis

The most notable feature of the efficiency measurements is that the Steel Wrist-Roll drive was more efficient as a speed increaser than as a speed reducer. More precisely, power transfer was more efficient when power flowed from the low-speed port to the high-speed port. The Steel Wrist-Roll drive was most efficient at the highest torque loads and the lowest speeds. Decrease in efficiency with increasing speeds is consistent with the observed speed-dependent frictional losses. High-efficiency as a speed increaser was observed in both the lubricated and unlubricated state, whether excited by a torque source or by a velocity source.

Another interesting feature of the efficiency measurements of the Steel Wrist-Roll drive is that it exhibited higher efficiencies when lubricated than when run dry. This indicates that lubrication was effective and that viscous losses were not significant.

Efficiency measurements on the Plastic Wrist-Roll drive were affected by the two failures which occurred during testing. Nonetheless, the efficiencies at higher loads appears flat as a function of velocity. This behavior is consistent with the input and output friction measurements performed prior to the failures, in which the friction appeared entirely Coulomb-like, with no observable speed dependence.

Dynamic efficiency measurements were not performed on the Hinge-Joint drive. The quasi-static measurements showed a torque hysteresis of approximately 30 N-m of output torque. This torque loss is higher than the measured output friction (driven as an unloaded speed increaser) and higher than the reflected measured input friction (driven as an unloaded speed increaser). However, the torque loss is less than the combined output friction plus reflected input friction.

Since the measured friction curves of the Hinge-Joint drive are essentially flat (Coulomb-like), the efficiency of this drive may be expected to be flat vs velocity. At the rated full torque of 820 N-m, a loss of 30 N-m would result in a projected efficiency of 96%.

PITCH/YAW JOINT DRIVE

STIFFNESS

Experimental Procedure

Static stiffness tests were conducted by locking the inputs and measuring deflections of the output load arms under various loads. Figure 50 shows the test setup. Dual load arms were used to apply torques about the pitch axis, and a single load arm was used to apply torques about the yaw axis.

The inputs were locked by first removing the torque meters. A length of 3/8 inch square (9.52 mm square) steel rod with a coupling silver soldered to one end was substituted for each torque meter. When in place no backlash could be detected in either rod between it and its connection to the drive. The two rods were then locked to each other by placing two lengths of heavy angle iron at right angles to the axes of the rods. Sturdy C-clamps locked the angle irons to the rods.

A .0001 in/division (.00254 mm/div) dial indicator was clamped to the frame of the drive with the indicator point contacting the load arm at 4.687 in (119mm) from the drive axis. For torque about the pitch axis, each load arm weighs 6.56 pounds (2.98 Kg) with a center of gravity 16.5 inches (419.1 mm) from the mounting axis. The crossbar weighs 1.156 pounds (.525 Kg) and acts at 33 inches (838.2 mm) from the axis. The torque about the pitch axis is therefore

$$T_p = [\ 2(16.5)(6.563) + 33\ (W+1.156)\ ]\ \cos\Theta$$

where Θ is the angle to the horizontal made by the load arms. For these tests $\Theta_P = 4.66^o$.

$$T_p = 253.89 + 32.89\ W \quad \text{in lbs (W in lbs)}$$
$$T_p = 28.69 + 0.835\ W \quad \text{Nm (W in N)}$$

The tare torque at zero deflection (arms in place) is 253.89 in lbs (28.69 Nm).

The angular deflection about the pitch axis is

$$\Theta_P = \frac{Dp}{4.687 \cos 4.66^o}$$

$$\Theta_P = \frac{Dp}{4.672} \quad \text{rad}$$

where Dp is the dial indicator reading.

The torsional stiffness about the pitch axis is

$$K_{\Theta P} = \frac{T_P - 253.89}{\Theta_P}$$

$$K_{\Theta P} = \frac{32.89}{\Theta_P}\ W$$

In a similar fashion, use of a single load arm to apply torque about the yaw axis results in

$$T_Y = [\ (16.5)(6.653) + (W + .969)(33)\ ]\ \cos\phi\ \cos\Theta$$

where ϕ is the angle between the yaw axis and a true horizontal. For these tests $\phi = 5.87^o$ and $\Theta = 5.82^o$

Therefore

$$T_Y = 138.8 + 32.659\ W \quad \text{in lbs (W in lbs)}$$
$$T_Y = 15.689 + .830\ W \quad \text{Nm (W in N)}$$

The tare torque at zero deflection is 138.8 in lbs (15.69 Nm).

In applying torques about the yaw axis there is also a torque about the pitch axis

$$T_P' = (6.563 + W + .969)(3.625)\ \cos\Theta$$
$$T_P' = 27.163 + 3.606\ W \quad \text{in lbs (W in lbs)}$$
$$T_P' = 3.07 + .0916\ W \quad \text{Nm (W in N)}$$

The tare torque at zero deflection is 27.163 in lbs (3.07 Nm).

The previously derived pitch axis stiffness, together with the torque about the pitch axis, must be taken account of in calculating stiffness about the yaw axis.

$$\theta_Y = \frac{D_Y - \theta_P\ (3.625 \cos 5.82^{o})}{4.687 \cos 5.87^{o}}$$

$$\theta_Y = \frac{D_Y - 3.810\ (3.606\ W/K_{\theta P}}{4.662}$$

$$K_{\theta Y} = T_Y - 138.8$$

or

$$K_{\theta Y} = \frac{32.659\ W}{\theta_Y}$$

Data

Measured deflections at various loads about the pitch axis are shown in TABLE 11, along with calculated values of θ_P and $K_{\theta P}$. Measured deflections at various loads about the yaw axis are shown in TABLE 12, along with calculated values of θ_Y and $K_{\theta Y}$. A value of $K_{\theta P}$ = 137,450 as determined from the pitch axis data was used calculate $K_{\theta Y}$.

Analysis

Data of load vs. angular deflection are shown plotted in Figure 51. The pitch axis data are fairly linear over the load range investigated and indicate a torsional stiffness of approximately 137,000 in lb/rad (15,490 Nm/rad). The yaw axis data are irregular, indicating decreasing stiffness with increasing load. This is not likely real, so the methodology of accounting for deflections about the pitch axis probably has deficiencies. The pitch axis data can probably be accepted as a reasonable value for the joint about either the pitch or yaw axis.

It is interesting to compare these results with the theoretical stiffness of the gear train. At the rated input torques of 240 in lb (27.2 Nm) we have the following:

1. First Stage
   Mean P.D. of pinion = 1.6567 in (42.08 mm)
   Mean P.D. of gear = 5.68 in (144.3 mm)
   F_T = 289.8 lbs (1290N) tangential
   F_N = 308 lbs (1371N) normal
   F_S = 105.5 lbs (469.5N) separating

2. Second Stage

Mean P.D. of gears = 2.88 in (73.15 mm)

F_T = 571.5 lbs (2543N) tangential

F_N = 608.8 lbs (2709N) normal

F_S = 208 lbs (926N) separating

From ref. 5,

Tooth deflection

$$Y_T = \frac{4.85\ F_N}{Ef}$$

$$2Y_T = \frac{9.70\ F_N}{Ef}$$

Compliance

$$C = \frac{2Y_T}{F_T r^2} \qquad \text{rad/in lb}$$

First Stage:

$$C_1 = \frac{(9.70)(308)}{(30\text{x}10^6)(.31)(289.8)(1.6567/2)^2}$$

$$C_1 = 8.0776\text{x}10^{-7} \quad \text{rad/in lb}$$

Second Stage:

$$C_2 = \frac{(9.70)(608)}{(30\text{x}10^6)(.5)(571.5)(2.88/2)^2}$$

$$C_2 = 3.3177\text{x}10^{-7} \quad \text{rad/in lb}$$

Overall:

$$C_T = \frac{C_1}{(3.4286)^2} + C_2$$

$$C_T = 6.871\text{x}10^{-8} + 3.3177\text{x}10^{-7}$$

$$C_T = 4.0048\text{x}10^{-7} \quad \text{rad/in lb}$$

$$K_T = 2.5\text{x}10^6 \text{ in lb/rad}$$

Since there are two inputs in parallel the theoretical stiffness of the entire gear set should be

$$2K_T = 5\text{x}10^6 \text{ in lb/rad}$$

The roller train would be expected to have a torsional stiffness comparable to that of the gear train. Since the rollers act in parallel with the gears, the stiffness of the roller-gear train should be greater than that of the gears alone.

Comparison of the theoretical gear set stiffness with the measured system stiffness shows that about 97 percent of the compliance arises from the multitude of spring elements, other than rollers and gears, that comprise the system. The sequence of spring elements from input to output is

1. Torque meter shaft
2. Key connection with NAS-235
3. Dowel pin and bolt connection to input pinion NAS-209
4. Input pinion needle bearings
5. Housing NAS-201
6. Input pinion
7. Input pinion to gear NAS-205 tooth connection
8. Gear NAS-205 Kaydon ball bearing
9. Housing NAS-201
10. Gear NAS-205
11. Dowel pin and bolt connection to gear NAS-207
12. Barden torque tube bearings
13. Pitch shaft NAS-238
14. Gear NAS-207 to gear NAS-203 tooth connection
15. Gear NAS-203 angular contact bearings
16. Yaw shaft NAS-202

The "soft" elements in this sequence are the frame, NAS-201, and the various bearings. The interaction of the elements is complex and not amenable to analysis. To appreciably increase torsional stiffness the housing and various bearings would have to be made significantly heavier.

CONTROL SYSTEM AND JOINT PERFORMANCE

Experimental Procedure

Generally, single motion commands to execute a simple change in pitch or yaw were inputted to the system to evaluate system performance and response of the joint. Figure 52 illustrates schematically the loading system which consists of a yoke and two load arms. The load arms can be bolted to the yoke in either the forward or rearward position. The joint was operated at four load levels ranging from 0 (yoke only, arms off) to 100.73 ft lbs output torque (about 75% of full load rating). Inertia effects became very troublesome at higher loads, so data was not taken at full rating. Table 13 is a tabulation of the data files; the type of maneuver, initial and final pitch and yaw positions, and load data are listed.

As stated earlier, the data system operates at 250 HZ, recording values of fourteen variables at .004 second intervals. Data files can be printed in tabular form but, because of the huge volume of data generated in runs of only a fraction of a second, they are not readily amenable to analysis without a plotter routine. To facilitate data analysis MATLAB was incorporated into the software system. Data files are translated into a format readable by MATLAB and then plotted exactly as read, or filtered to smooth out noise and irregularities arising from limits in instrument accuracy. Unless otherwise stated on a figure a value of 11 was used to smooth data. This means that each data point represents the arithmetic average of the five previous points, the point in question and the five succeeding points. Motor Command and Tach (motor speed) data were filtered to remove electronic noise. Data from Input Resolvers and Output Pitch and Yaw Position are presented unfiltered because the readouts are smooth within the accuracy of the instruments (20 minutes of arc for the input resolvers and 75 seconds of arc for the pitch and yaw resolvers). Torque data are filtered because of a ripple in the A side input (to be discussed later). Pitch and Yaw Velocity data are filtered to reduce irregularities which are magnified when slopes are calculated from position data at .004 second intervals. Changes in position at successive samples are on the order of 10 minutes of arc while resolver accuracy is on the order of 1.25 minutes of arc or 12.5 percent of the measurement difference.

Data and Analysis

Figures 53-61 illustrate data from a 20° pitch lift (Data 76), a 20° pitch drop (Data 77) and a 30° yaw swing (Data 78), each at essentially zero load. Motor Commands for the two pitch moves are shown in Figure 53. On each of the data plots of two variables, the first is represented by the solid line and the second by the dashed line. On four variable plots (Desire and Actual Pitch and Yaw, for example) the hierarchy is:

Desire Pitch - Solid line
Actual Pitch - Dashed line
Desire Yaw - Dot-dash line
Actual Yaw - Dotted line.

Full voltage for the Motor Command is ± 10 volts. For the pitch moves the system response time to reach full voltage (Figure 53) was .08 seconds and for the yaw move (Figure 59a) was .057 seconds. Duration of the full Motor Command voltage was 0.4 seconds for the 20° pitch moves and 0.65 seconds for the yaw move. Figures 54 and 59b illustrate the Tachs (motor speed patterns) for the pitch moves and the yaw move, respectively. Time to reach maximum motor speed was 0.16 seconds. Duration of maximum motor speed was 0.4 seconds for the pitch moves and 0.58 seconds for the yaw move. Note, for the pitch moves, the opposite sense of the Motor Commands and the Tachs (Figures 53 and 54). For a pure pitch move input rotations are in the opposite sense. In contrast inputs are in the same sense for a pure yaw move (Figure 59). For all three moves maximum motor speed approaches 2000 rpm. This was found to be true for all operations of the joint except when the output torque approaches full rating.

Figure 55 illustrates input shaft positions for the two pitch moves, as measured by the input resolvers. The absolute values of resolver angles, and whether or not the two curves approach or diverge for a particular move, have no meaning. Input resolver positions at the beginning of a move are random. For a pitch move they must approach or diverge, however. For these two cases each resolver moves approximately $(20)(3.4286)=68.57^{\circ}$, where 3.4286 is the reduction ratio of the joint. In contrast, for a yaw move the input resolvers move together, as shown in Figure 60a. For a 30° yaw move, each resolver should move approximately $(30)(3.4286)=102.9^{\circ}$. Close examination of Figure 60a shows that resolver A moved 113° and resolver B 92°, indicating some asymmetry in control for a pure yaw move. For all three moves, the moves are smooth. Any high frequency variations in motion are less than the accuracy of the resolvers.

Figure 56 shows the Desired and Actual Pitch and Yaw for the two pitch moves and Figure 60b the same data for the yaw move. "Desire Pitch" and "Desire Yaw" are representative of an ideal system with zero response time operating at maximum speed. Actually the curves of Figure 56 are compatible with the system's capabilities but Figure 60b is overly optimistic. At a maximum achievable motor speed of 2000 rpm the angular speed of either the pitch axis or yaw axis would be

$$\left(\frac{2000}{90}\right)\left(\frac{1}{3.4286}\right) = 6.48 \text{ rpm}$$

(90 is the gear box ratio, and 3.4286 is the joint ratio)

Then a 20° pitch move should be executable in

$$\left(\frac{20}{360}\right)\left(\frac{1}{6.48}\right)\left(\frac{60}{1}\right) = 0.514 \text{ seconds}$$

Figure 56 indicates this and the system response is excellent, with a lag time of 0.1 seconds.

For a 30° yaw move a realistic execution time would be 0.771 seconds. The "Desire Yaw" curve on Figure 60b is too optimistic. The joint executes the yaw move in 0.88 seconds which is what can be expected with a servo lag time of 0.1 seconds.

Pitch and Yaw Velocities as calculated from slopes of the position curves are shown in Figure 57 for the two pitch moves, and on Figure 61b for the yaw move. All three curve sets indicate an angular velocity of 6.5 to 6.7 rpm for the axis in motion, with a ripple of approximately ±2.2 percent. The frequency of the velocity ripple is approximately 37.5 cycles/sec or 2250 cycles/minute. This is not related to any mechanical frequency in the joint itself, but it is fairly close to the number of data samples (27) per revolution of the moving axis. Considering that the pitch and yaw resolver inaccuracies can be as high as 12 percent of the magnitude of the movement from point to point, much of the ripple may be noise.

Torque data are shown for the two pitch moves on Figure 58 and for the yaw move on Figure 61a. The B input torque curve is smooth but the A input curve shows a definite ripple. The frequency for the two pitch moves on Figure 58 is 75 cycles/sec and that for the yaw move on Figure 61a is 37 cycles/sec. Again, neither of these frequencies is relatable to any mechanical frequency in the joint. When the A side input ripple was first noted the joint was partially disassembled to check the input pinion-gear setting. The A side shim set was originally .067 in. and the B side .070 inches. A .004 in. shim was added to the A side assembly. This increased the backlash approximately .001 in. but made no difference in the pattern or magnitude of the ripple. The significance of this will be discussed more in the discussion of efficiency.

The high values of torque at startup when the joint is accelerating up to speed are the result of inertia. Even the relatively low inertia of the joint components and the yoke without the load arms or any weights attached (Figure 52) requires initial torques on the order of 12 to 16 ft lbs (Figures 58 and 61a) to accelerate the mass up to speed. In other runs at high levels of output torque load, input torques of up to 35 ft lbs were recorded. With inertia present in the joint components and loading system, only the torque values in the constant speed region of each run are amenable to analysis.

Note the relative differences and magnitudes of the two input torques in the region between sample numbers 50 and 130 on Figure 58 a and b. For the pitch lift (Figure 58a) the joint is doing work and the tare torques are approximately ±1.1 ft lbs. For the pitch drop (Figure 58b) the small torque of the yoke does work on the joint and the tare torques resisting the drop are ±0.16 ft lbs.

BACKLASH AND LINEARITY

When the static stiffness setup was in place, backlash measurements were made with dial indicators placed against output members. The inputs were effectively locked with the arrangement discussed in the section on STIFFNESS. No backlash could be discerned through the roller-gear trains from the two inputs to the output.

To investigate linearity a series of multiple runs was made (data files 79-83, TABLE 13). These consisted of three dual moves in pitch (a pitch drop followed by a pitch lift) at three load levels, a dual move in yaw (a swing followed by a return) and a quadrille move (a yaw swing followed by a pitch drop followed by a yaw swing back and a pitch lift). Data from these moves are plotted in Figures 62 through 74.

In the regions of constant motor speed, for all of these moves, the filtered velocity of the moving axis (whether pitch or yaw) indicates an approximate cyclic variation of $\pm$ 2 percent (Figures 63b, 65b, 67b, 69b and 71b. A pitch or yaw shaft angular velocity of 6.5 rpm represents a change in position of approximately 9.4 minutes of arc each .004 seconds (one sampling interval). The accuracy of the pitch and yaw resolvers is 1.25 minutes of arc or 12.5 percent of measurement magnitude. Possible inaccuracies in measurement, therefore, exceed the measurement magnitude so the cyclic fluctuation may or may not be real. Its frequency (37 cycles/sec) cannot be related to any obvious physical characteristic of the P/Y joint.

In regions of rapid acceleration or deceleration (at the beginning, center portion and end of each dual move) inertia effects predominate. This is more and more apparent as output torque (and thus load system inertia) increases. Figures 63b (zero load), 65b (17.96 ft lbs torque) and 67b (60.44 ft lbs torque) show an increasing irregularity in velocity profile. The fluctuations in velocity may be a feedback from the torque loading system. A relatively undamped high inertia loading system could account for, at least in part, for the observed velocity profiles. The increasing irregularity of input torques with increasing load for the three dual pitch moves, Figures 72a, 73a and 73b, indicate graphically that the loading system characteristics influence behavior of the joint.

Figures 70 and 71 indicate that the joint operates smoothly through a four part maneuver. Velocity fluctuations are very similar in the pitch and yaw changes (Figure 71b). For this quadrille maneuver with zero load, tare torque is greatest for the pitch lift move, midway for the two yaw swings and least for the pitch drop move (Figure 74).

It is apparent from these results that a more meaningful assessment of joint linearity would require further work with a low (or preferably zero) inertia damped loading system and more precise resolvers or encoders.

EFFICIENCY

For the Pitch/Yaw Joint efficiency can be defined as

$$\eta = \left[\frac{\text{Input Power - Output Power}}{\text{Input Power}} \right] 100$$

Input Power = $T_A \omega_A + T_B \omega_B$
where T_A and T_B are input torques and
ω_A and ω_B are input angular velocities

From the kinematics of the joint

$$\omega_{PITCH} = \omega_P = \frac{\omega_A - \omega_B}{2R}$$

$$\omega_{YAW} = \omega_Y = \frac{\omega_A + \omega_B}{2R}$$

R, the reduction ratio, is 3.4286

when torques T_P and T_Y are applied to the output of the joint, then

$$\text{Output Power} = T_P \left(\frac{\omega_A - \omega_B}{2R}\right) + T_Y \left(\frac{\omega_A + \omega_B}{2R}\right)$$

These relationships are used to calculate efficiency from known values of T_P, T_Y and R, and measured values of T_A and T_B. During a pitch lift the imposed torque load is the output. During a pitch drop the imposed torque load is the input with the normal "input" torques opposing the motion and acting as "outputs".

The loading system makes it impossible to apply a torque about the yaw axis without also applying a torque about the pitch axis. For simplicity torque data were taken only in pitch moves (pitch lifts and drops) of 20 to 30° with the load arms close to horizontal. The change in torque due to small rotations from the horizontal was neglected. The joint was operated at four load levels: 0, 17.96, 60.44 and 100.73 ft lbs output torque about the pitch axis. Figures 50a and 52 illustrate the loading arrangement. TABLE 14 summarizes the data files from which torque data are obtained and the figures on which the torque plots can be found.

The frictional loss at zero load can be calculated from the torque data on Figures 58a, 58b and 72a. The presence of the yoke (Figure 52) applies a small but non-zero torque at the output during pitch lifts and drops. Its effect can be cancelled by averaging the input torques for the lift and drop moves. The input torques during pitch lifts (Figures 58a and 72a) average ± 1.07 ft lbs. The input torques during pitch drops average ± 0.24 ft lbs. The average of these, ± 0.66 ft lbs, is probably a reasonably good representation of tare loss in the joint. This represents about 0.5 percent of the full load rating. The torque ripple in the A side input, which was present for all of the runs made, did not result in any statistical difference in absolute magnitudes between the A and B side torques. The algebraically positive input torque (A side with load arms forward, and B side with the load arms rearward) was slightly greater in magnitude than the algebraically negative torque for all of the pitch moves.

Efficiency increases with increasing load. At 60 ft lbs torque (slightly less than 50 percent of full load rating) efficiency values between 88 and 98 percent were obtained. Figure 79 is a plot of all of the torque data tabulated on TABLE 14. Efficiencies of 98 percent or better can be expected when the joint is loaded at 70 percent or greater of full load rating.

Figures 75 through 78 show the torque plots at load torques of 17.96, 60.44 and 100.73 ft lbs. Input torques become increasingly irregular at higher values of load torque because of inertia. Torques were averaged over the constant speed region of the move for the efficiency calculations. The relatively short duration of a typical move (less than one second) requires an initial acceleration up to speed, followed by a constant speed region, and a deceleration to a stop. The duration of the constant speed region is approximately 0.35 seconds. During acceleration and deceleration input torques can be high and irregular due to bouncing of the weights at the end of the load arms. As shown on Figures 75 through 78, input torques are irregular and asymmetric during the acceleration and deceleration portions of the move, reaching values as high as 35 ft lbs. This compares with steady state values of $\pm$14.4 ft lbs at the highest torque load (Figure 78).

It is apparent that for more precise torque and efficiency analysis a low inertia, damped load system would have to be employed.

REFERENCES

1. Anderson, William J.: "Roller-Gear Drives For Robotic Manipulators". Final Report for NASA Lewis Research Center, Contract NAS 3-25282, August, 1988.

2. duPont Company: "Design Handbook for duPont Engineering Plastics", Section 9, Gear Design, E. I. duPont de Nemours & Co., Inc. Polymer Products Department, Wilmington, Delaware, 19898.

3. Amoco Performance Products, Inc.: "Torlon Engineering Polymers/Design Manual". Booklet F-49893. Amoco Performance Products, Inc., 38C Grove Street, Ridgefield, Ct., 06877.

4. Roark, Raymond J., and Young, Warren C.: "Formulas for Stress and Strain, 5th Edition". McGraw-Hill Book Company, New York, 1982.

5. Cornell, R. W.: "Compliance and Stress Sensitivity of Spur Gear Teeth". Journal of Mechanical Design, Vol. 103, No. 2, April, 1981.

6. Buckingham, E.: "Manual of Gear Design". The Industrial Press, New York.

TABLE 1 - GEAR DATA FOR THE
WRIST-ROLL JOINT ROLLER-GEAR DRIVE

GEAR	a	x_1	y_1	x_2	c
Number of teeth	24	51	15	60	164
Diametral pitch	43	43	38	38	38
Pitch dia.,in.	.5581	1.1860	.3947	1.5789	4.3158
Pressure ang.deg.	20	20	20	20	20
Base Circle dia., in.	.5245	1.1145	.3709	1.4837	4.0555
Circular tooth thickness, in.	.0392 .0387	.0319 .0314	.0440 .0435	.0315 .0310	.0454 .0444
Outside or Major dia., in.	.6126 .6120	1.220 1.219	.4552 .4547	1.600 1.599	4.3626 4.3679
Root or Minor dia., in.	.5121 .5056	1.1199 1.1134	.3415 .3341	1.4913 1.4839	4.284 4.285
True involute form dia. (max)	.5318	1.1462	.371	1.5255	4.354
Msrg.pin dia.,in.	.040	.040	.050	.050	.050
Measurement over (between) pins	.6194 .6182	1.2293 1.2279	.4744 .4735	1.6344 1.6331	4.2201 4.2231
Operating Center Distance (ref.1)	.870	.870	.9775		1.3749
Contact Ratio	1.66		1.66		
				1.30	
Backlash at Operating Center Distance,(ref.1)	.0015 .0015	.0005 .0015	.0007 .0017		.001 .0025
Operating Pitch Dia., (ref.1)	.5568	1.1832	.3910	1.564 1.5864	4.3361
Mating Roller Dia.	.5568	1.1832	.3910	1.564 1.5864	4.3344
Dwg.No. (Steel)	NAS-141	NAS-147	NAS-144	NAS-152	NAS-155
Dwg.No. (Plastic)	NAS-182	NAS-147	NAS-144	NAS-152P	NAS-155

TABLE 2. - ROLLER AND GEAR MATERIALS FOR THE WRIST-ROLL DRIVES

	STEEL DRIVE (NAS-140A)	PLASTIC DRIVE (NAS-140AP)
SUN		
ROLLER a	AMS 6490 M-50	AISI 4340
GEAR a	[1]ETD 150	DELRIN 100
FIRST PLANET		
ROLLER x_1	AMS 6490 M-50	TORLON 4203
GEAR x_1	[1]ETD 150	[1]ETD 150
ROLLER y_1	AMS 6490 M-50	AISI 4340
GEAR y_1	[1]ETD 150	[1]ETD 150
SECOND PLANET		
ROLLER x_2	AMS 6490 M-50	TORLON 4203
GEAR x_2	[1]ETD 150	DELRIN 100
RING		
ROLLER c	AMS 6490 M-50	AISI 4340
GEAR c	[1]ETD 150	[1]ETD 150

[1]LA SALLE ETD 150

TABLE 3. - GEAR AND ROLLER MATERIALS FOR THE STEEL-PLASTIC WRIST-ROLL DRIVE

	GEAR	ROLLER
SUN	DELRIN	[1]STEEL
1st ROW		
GEAR	[1]STEEL	TORLON
PINION	[1]STEEL	[1]STEEL
2nd ROW		
1st	DELRIN	TORLON
RING	DELRIN	TORLON
RING	[2]STEEL	[2]STEEL

[1]LA SALLE ETD 150 STEEL

[2]AISI 4340 STEEL

TABLE 4. - ROLLER DIAMETERS AND DRAWING SOURCES FOR WRIST-ROLL DRIVES

ROLLER	STEEL DRIVE	DRAWING NO.	PLASTIC DRIVE	DRAWING NO.
a. Dim.	$\frac{.5569}{.5567}$	NAS-143	$\frac{.5570}{.5568}$	NAS-142P, NAS-183
Material	M-50 Steel (AMS-6490)	NAS-146	4340 Steel	NAS-146P, NAS-183
x_1 Dim.	$\frac{1.1833}{1.1831}$	NAS-149	$\frac{1.1840}{1.1836}$	NAS-149P
Material	M-50 Steel (AMS-6490)	NAS-148	Torlon 4203	NAS-148P
y_1 Dim.	$\frac{.3911}{.3909}$	NAS-149	$\frac{.3912}{.3908}$	NAS-149P
Material	M-50 Steel (AMS-6490)	NAS-146	4340 Steel	NAS-146P
x_2 Dim.	$\frac{1.5643}{1.5641}$	NAS-154	$\frac{1.5662}{1.5658}$	NAS-154P
Material	M-50 Steel (AMS-6490)	NAS-154	Torlon 4203	NAS-154P
y_2 Dim.	$\frac{1.5871}{1.5869}$	NAS-154	$\frac{1.5886}{1.5882}$	NAS-154P
Material	M-50 Steel (AMS-6490)	NAS-154	Torlon 4203	NAS-154P
c Dim.	$\frac{4.3356}{4.3346}$	NAS-156	$\frac{4.3365}{4.3355}$	NAS-156P
Material	M-50 Steel (AMS-6490)	NAS-156	4340 Stee	NAS-156P

TABLE 5 - GEAR DATA FOR THE HINGE-JOINT ROLLER-GEAR DRIVE

GEAR	a	x_1	y_1	x_2	c
Number of teeth	28	62	20	100	264
Diametral pitch	32	32	28	28	28
Pitch dia.,in.	.875	1.9375	.7143	3.5714	9.4286
Pressure ang.deg.	20	20	20	20	20
Base Circle dia., in.	.8222	1.8207	.6712	3.3560	8.8600
Circular tooth thickness, in.	.0519 .0514	.0338 .0333	.0663 .0658	.0614 .0609	.0617 .0607
Outside or Major dia., in.	.9461 .9455	1.9529 1.9519	.8145 .8135	3.6583 3.6573	9.4929 9.5000
Root or Minor dia., in.	.8111 .8023	1.8239 1.8151	.6602 .6502	3.5040 3.4940	9.370 9.372
True involute form dia. (max)	.8369	1.8629	.6846	3.5410	9.476
Msrg.pin dia.,in.	.060	.060	.06857	.06857	.06857
Measurement over (between) pins	.9768 .9757	1.9961 1.9947	.8423 .8414	3.6975 3.6963	9.2967 9.2996
Operating Center Distance, in.	1.390		2.1650	6.4630	
Contact ratio	1.69		1.55	1.48	
Backlash at Operating Center Distance, in.	.001 .002		.0013 .0023		.001 .002
Operating Pitch Diameter, in.	.8649	1.9151	.7217	3.6083 3.5511	9.3749
Mating Roller Dia., in.	.8649	1.9151	.7217	3.6083 3.5511	9.3749
Drawing No.	NAS-302	NAS-305 NAS-307	NAS-305 NAS-306	NAS-311	MAS-313

TABLE 6. - ROLLER AND GEAR MATERIALS FOR THE HINGE-JOINT DRIVE

	MATERIAL (Steel Alloys)	DRAWING NO.
SUN		
Roller a	AMS-6490 Rc 61-63	NAS-303
Gear a	Nitralloy 135 M .005/.008 in. CASE, Rc 60-63	NAS-302
FIRST PLANET		
Roller x_1	AMS-6490 Rc 61-63	NAS-310
Gear x_1	Nitralloy 135M .005/.008 in. CASE, Rc 60-63	NAS-307 NAS-305
Roller y_1	AMS-6490 Rc 61-63	NAS-309
Gear y_1	Nitralloy 135M .005/.008 in. CASE, Rc 60-63	NAS-306 NAS-305
SECOND PLANET		
Roller x_2 y_2	AMS-6490 Rc 61-63	NAS-312
Gear x_2	Nitralloy 135M .005/.008 in. CASE, Rc 60-63	NAS-311
RING		
Roller c	AMS-6265(AISI 9310) .030/.045 in. CASE R_{15N} 89.5-91	NAS-314
Gear c	SAE 4340 Rc 33-38	NAS-313

TABLE 7. - ROLLER DIAMETERS AND DRAWING SOURCES FOR HINGE-JOINT DRIVE

	DIAMETER, in.	DRAWING NO.
SUN		
a	$\frac{.8655}{.8653}$	NAS-301
FIRST PLANET		
x_1	$\frac{1.9157}{1.9155}$	NAS-304
y_1	$\frac{.7223}{.7221}$	NAS-304
SECOND PLANET		
x_2	$\frac{3.6089}{3.6087}$	NAS-312
y_2	$\frac{3.5520}{3.5518}$	NAS-312
RING		
c	$\frac{9.3685}{9.3689}$	NAS-314

TABLE 8 - ZEROL BEVEL GEAR STRESSES AND REACTION FORCES FOR PITCH/YAW JOINT ROLLER-GEAR DRIVE
Standard Involute 20° P.A.

	GEAR			
	Input Pinion	Intermediate Fixed Gear	Intermediate Transversing Gear	Pitch-Yaw Gear
Cone Half Angle, deg.	73.74	16.26	45	45
Diametral Pitch	16	16	12	12
Pitch Dia. mm (in.)	44.45 (1.75)	152.4 (6.00)	82.55 (3.25)	82.55 (3.25)
No. of Teeth	28	96	39	39
Face Width, mm (in.)	7.87 (.31)	7.87 (.31)	12.7 (.50)	12.7 (.50)
Tooth Bending Stress, GPa (psi)	.181 (26,300)	.181 (26,300)	.192 (27,800)	.192 (27,800)
Tooth Comp. Stress, GPa (ksi)	1.414 (205)		1.352 (196)	
Separating Force, N(lb.)	472 (106)		926 (208)	
Radial Force, N(lb.)	449 (101)		654 (147)	
Axial Force, N(lb.)	131 (29.5)		654 (147)	
Drawing	NAS-209	NAS-205	NAS-207	NAS-203

TABLE 9 - ZEROL BEVEL GEAR GEOMETRY DATA

(a) Input Pinion and Gear

ZEROL BEVEL GEAR DIMENSIONS NO. 2014788 FORM M-2 DATE 10/ 4/89 TIME 11:33

TRANSMISSION RESEARCH-CLEVELAND

	PINION	GEAR
NUMBER OF TEETH	28	96
PART NUMBER		
DIAMETRAL PITCH		16.000
FACE WIDTH	0.310"	0.310"
PRESSURE ANGLE	20D 0M	
SHAFT ANGLE	90D 0M	
TRANSVERSE CONTACT RATIO . . .		1.601
FACE CONTACT RATIO		0.000
MODIFIED CONTACT RATIO		1.601
OUTER CONE DISTANCE		3.125"
MEAN CONE DISTANCE		2.970"
PITCH DIAMETER	1.750"	6.000"
CIRCULAR PITCH	0.196"	
WORKING DEPTH	0.125"	
WHOLE DEPTH	0.137"	0.137"
CLEARANCE	0.012"	0.012"
ADDENDUM	0.089"	0.036"
DEDENDUM	0.048"	0.101"
OUTSIDE DIAMETER	1.921"	6.020"
FACE ANGLE JUNCTION DIAMETER		
THEORETICAL CUTTER RADIUS .		
CUTTER RADIUS	3.000"	
CALC. GEAR FINISH. PT. WIDTH		0.044"
GEAR FINISHING POINT WIDTH .		0.045"
ROUGHING POINT WIDTH	0.035"	0.035"
OUTER SLOT WIDTH	0.045"	0.045"
MEAN SLOT WIDTH	0.044"	0.045"
INNER SLOT WIDTH	0.043"	0.045"
FINISHING CUTTER BLADE POINT	0.025"	0.025"
STOCK ALLOWANCE	0.009"	0.010"
MAX. RADIUS-CUTTER BLADES .	0.023"	0.024"
MAX. RADIUS-MUTILATION . . .	0.047"	0.050"
MAX. RADIUS-INTERFERENCE . .	0.010"	0.031"
CUTTER EDGE RADIUS	0.010"	0.020"
CALC. CUTTER NUMBER	0	0
MAX. NO. BLADES IN CUTTER .		60.723
CUTTER BLADES REQUIRED . . . STD	DEPTH	STD DEPTH
GEAR ANGULAR FACE - CONCAVE		5D 47M
GEAR ANGULAR FACE - CONVEX .		6D 3M
GEAR ANGULAR FACE - TOTAL .		5D 56M

		PINION		GEAR
PITCH APEX TO CROWN		2.975"		0.840"
FACE ANG JUNCT TO PITCH APEX				
MEAN CIRCULAR THICKNESS . .		0.108"		0.075"
OUTER NORMAL TOP LAND . . .		0.034"		0.049"
MEAN NORMAL TOP LAND		0.037"		0.051"
INNER NORMAL TOP LAND . . .		0.039"		0.048"
PITCH ANGLE		16D 16M		73D 44M
FACE ANGLE OF BLANK		19D 29M		75D 3M
INNER FACE ANGLE OF BLANK .				
ROOT ANGLE		14D 57M		70D 31M
DEDENDUM ANGLE		1D 19M		3D 13M
OUTER SPIRAL ANGLE				2D 53M
MEAN SPIRAL ANGLE				0D 0M
INNER SPIRAL ANGLE				- 3D 3M
HAND OF SPIRAL		LH		RH
DRIVING MEMBER		PIN		
DIRECTION OF ROTATION-DRIVER		REV		
OUTER NORMAL BACKLASH . . .	MIN	0.002"	MAX	0.004"
DEPTHWISE TOOTH TAPER . . .		DPLX		
GEAR TYPE				GENERATED
FACE IN PERCENT OF CONE DIST				9.920
DEPTH FACTOR - K				
ADDENDUM FACTOR - C1				
GEOMETRY FACTOR-STRENGTH-J .		0.3185		0.3175
STRENGTH FACTOR-Q		92.587		27.094
EDGE RADIUS USED IN STRENGTH		0.010"		0.020"
CUTTER RADIUS FACTOR - KX .				
FACTOR	KI	1.2494		
STRENGTH BALANCE DESIRED . .		STRS		
STRENGTH BALANCE OBTAINED .	STRS			0.003
GEOMETRY FACTOR-DURABILITY-I		0.1112		
DURABILITY FACTOR-Z		12313.51		4650.06
GEOMETRY FACTOR-SCORING-G .		0.002767		
SCORING FACTOR - X		0.6134		
ROOT LINE FACE WIDTH		0.310"		0.310"
PROFILE SLIDING FACTOR . . .		0.00169		0.00294
RATIO OF INVOLUTE/OUTER CONE				1.657
RATIO OF INVOLUTE/MEAN CONE				1.744
AXIAL FACTOR-DRIVER CW . . .	OUT	0.123	OUT	0.123
AXIAL FACTOR-DRIVER CCW . .	OUT	0.123	OUT	0.123
SEPARATING FACTOR-DRIVER CW	SEP	0.420	SEP	0.036
SEPARATING FACTOR-DRIVER CCW	SEP	0.420	SEP	0.036
DUPLEX SUM OF DEDENDUM ANG .		0D 0M		
ROUGHING RADIAL		4.221"		
INPUT DATA . . . . KTT		F		
INPUT DATA . . . . CUTM				1
BLADE GROUPS				
EFFECTIVE CUTTER RADIUS. . .				

RELEASED BY - PC

ZEROL IS A TRADEMARK OF THE GLEASON WORKS

TABLE 9 - ZEROL BEVEL GEAR GEOMETRY DATA

(b) Transversing and Pitch/Yaw Gears

ZEROL BEVEL GEAR DIMENSIONS NO. 2014788 FORM M-2 DATE 10/ 4/89 TIME 11:33

TRANSMISSION RESEARCH-CLEVELAND

	PINION	GEAR
NUMBER OF TEETH	28	96
PART NUMBER		
DIAMETRAL PITCH		16.000
FACE WIDTH	0.310"	0.310"
PRESSURE ANGLE	20D 0M	
SHAFT ANGLE	90D 0M	
TRANSVERSE CONTACT RATIO . .		1.601
FACE CONTACT RATIO		0.000
MODIFIED CONTACT RATIO . . .		1.601
OUTER CONE DISTANCE		3.125"
MEAN CONE DISTANCE		2.970"
PITCH DIAMETER	1.750"	6.000"
CIRCULAR PITCH	0.196"	
WORKING DEPTH	0.125"	
WHOLE DEPTH	0.137"	0.137"
CLEARANCE	0.012"	0.012"
ADDENDUM	0.089"	0.036"
DEDENDUM	0.048"	0.101"
OUTSIDE DIAMETER	1.921"	6.020"
FACE ANGLE JUNCTION DIAMETER		
THEORETICAL CUTTER RADIUS .		
CUTTER RADIUS	3.000"	
CALC. GEAR FINISH. PT. WIDTH		0.044"
GEAR FINISHING POINT WIDTH .		0.045"
ROUGHING POINT WIDTH	0.035"	0.035"
OUTER SLOT WIDTH	0.045"	0.045"
MEAN SLOT WIDTH	0.044"	0.045"
INNER SLOT WIDTH	0.043"	0.045"
FINISHING CUTTER BLADE POINT	0.025"	0.025"
STOCK ALLOWANCE	0.009"	0.010"
MAX. RADIUS-CUTTER BLADES .	0.023"	0.024"
MAX. RADIUS-MUTILATION . . .	0.047"	0.050"
MAX. RADIUS-INTERFERENCE . .	0.010"	0.031"
CUTTER EDGE RADIUS	0.010"	0.020"
CALC. CUTTER NUMBER	0	0
MAX. NO. BLADES IN CUTTER .		60.723
CUTTER BLADES REQUIRED . . .	STD DEPTH	STD DEPTH
GEAR ANGULAR FACE - CONCAVE		5D 47M
GEAR ANGULAR FACE - CONVEX .		6D 3M
GEAR ANGULAR FACE - TOTAL .		5D 56M

		PINION		GEAR
PITCH APEX TO CROWN		2.975"		0.840"
FACE ANG JUNCT TO PITCH APEX				
MEAN CIRCULAR THICKNESS . .		0.108"		0.075"
OUTER NORMAL TOP LAND . . .		0.034"		0.049"
MEAN NORMAL TOP LAND		0.037"		0.051"
INNER NORMAL TOP LAND . . .		0.039"		0.048"
PITCH ANGLE		16D 16M		73D 44M
FACE ANGLE OF BLANK		19D 29M		75D 3M
INNER FACE ANGLE OF BLANK .				
ROOT ANGLE		14D 57M		70D 31M
DEDENDUM ANGLE		1D 19M		3D 13M
OUTER SPIRAL ANGLE				2D 53M
MEAN SPIRAL ANGLE				0D 0M
INNER SPIRAL ANGLE				- 3D 3M
HAND OF SPIRAL		LH		RH
DRIVING MEMBER		PIN		
DIRECTION OF ROTATION-DRIVER		REV		
OUTER NORMAL BACKLASH . . .	MIN	0.002"	MAX	0.004"
DEPTHWISE TOOTH TAPER . . .		DPLX		
GEAR TYPE				GENERATED
FACE IN PERCENT OF CONE DIST				9.920
DEPTH FACTOR - K				
ADDENDUM FACTOR - C1				
GEOMETRY FACTOR-STRENGTH-J .		0.3185		0.3175
STRENGTH FACTOR-Q		92.587		27.094
EDGE RADIUS USED IN STRENGTH		0.010"		0.020"
CUTTER RADIUS FACTOR - KX .				
FACTOR	KI	1.2494		
STRENGTH BALANCE DESIRED . .		STRS		
STRENGTH BALANCE OBTAINED .	STRS			0.003
GEOMETRY FACTOR-DURABILITY-I		0.1112		
DURABILITY FACTOR-Z		12313.51		6650.06
GEOMETRY FACTOR-SCORING-G .		0.002767		
SCORING FACTOR - X		0.6134		
ROOT LINE FACE WIDTH		0.310"		0.310"
PROFILE SLIDING FACTOR . . .		0.00169		0.00294
RATIO OF INVOLUTE/OUTER CONE				1.657
RATIO OF INVOLUTE/MEAN CONE				1.744
AXIAL FACTOR-DRIVER CW . . .	OUT	0.123	OUT	0.123
AXIAL FACTOR-DRIVER CCW . .	OUT	0.123	OUT	0.123
SEPARATING FACTOR-DRIVER CW	SEP	0.420	SEP	0.036
SEPARATING FACTOR-DRIVER CCW	SEP	0.420	SEP	0.036
DUPLEX SUM OF DEDENDUM ANG .		0D 0M		
ROUGHING RADIAL		4.221"		
INPUT DATA . . . . KTT		F		
INPUT DATA . . . . CUTM				1
BLADE GROUPS				
EFFECTIVE CUTTER RADIUS. . .				

RELEASED BY - PC

ZEROL IS A TRADEMARK OF THE GLEASON WORKS

TABLE 10 - BEVEL ROLLER GEOMETRY, LOADS AND STRESSES FOR PITCH/YAW JOINT ROLLER-GEAR DRIVE

	ROLLER			
	Input	Intermediate Fixed	Intermediate Transversing	Pitch/ Yaw
Cone Half Angle, deg.	16.26°	73.74°	45°	45°
Pitch Rad. mm (in.)	18.445 (.726)	63.246 (2.490)	46.83 (1.8438)	46.83 (1.8438)
Face Width, mm (in.)	9.525 (.375)	9.525 (.375)	11.226 (.442)	11.226 (.442)
Cross Rad., mm (in.)	∞	381 (15)	381 (15)	∞
Normal Load,N (lb.)	4,899 (1,101)		6,621 (1,488)	
Contact Ellipse, 2a mm (in.) 2b mm (in.)	6.528 (.257) 0.940 (.037)		6.782 (.267) 1.448 (.057)	
Maximum Hertz Stress GPa (ksi)	1.53 (222)		1.28 (186)	
Normal Approach, mm (in.)	0.020 (.0008)		0.023 (.0009)	
Drawing	NAS-210	NAS-206	NAS-208	NAS-204

TABLE 11 - PITCH/YAW JOINT DRIVE PITCH AND TORSIONAL STIFFNESS DATA

LOAD W, lbs (N)	AVERAGE DEFLECTION, $\frac{D_{PA} + D_{PB}}{2}$ in (mm)	θ_P, rad	$K_{\theta P}$ $\frac{\text{in lb}}{\text{rad}}$ $\left(\frac{\text{Nn}}{\text{rad}}\right)$
2.5 (11.13)	.0028 (.0711)	.00060	137,000 (15,485)
5.06 (22.52)	.0049 (.1245)	.00105	158,500 (17,915)
10. (44.5)	.0108 (.2743)	.00231	142,400 (16,096)
15.72 (69.95)	.0181 (.4597)	.00387	133,600 (15,100)

TABLE 12 - PITCH/YAW JOINT DRIVE YAW AXIS TORSIONAL STIFFNESS DATA

LOAD W, lbs (N)	AVERAGE DEFLECTION, Dy, in(mm)	$\theta_P = \frac{3.606W}{K_{\theta P}}$, rad $K_{\theta P}$=137,450 (Figure 52)	θy, rad	Key $\frac{\text{in lb}}{\text{rad}}$ ($\frac{\text{Nm}}{\text{rad}}$)
2	.0011	.000052	.000193	338,000(38,204)
4	.0028	.000105	.000515	253,700(28,676)
6.56	.0058	.000172	.01104	194,000(21,927)
10.	.0122	.000262	.002403	135,900(15,361)
15	.0235	.000394	.00472	103,800(11,732)

TABLE 13 - DATA FILES USED FOR PITCH/YAW JOINT DRIVE ANALYSIS

FILE	MANEUVER	PITCH, deg. INITIAL	PITCH, deg. FINAL	YAW, deg. INITIAL	YAW, deg. FINAL	TORQUE ARM POSITION	LOAD TORQUE, ft lbs
DATA 48	PITCH LIFT	-20.3	20.1	10.07	10.8	REARWARD	60.44
DATA 49	PITCH DROP	20.1	-21.86	10.8	11.2	REARWARD	60.44
DATA 63	PITCH DROP	-0.44	-20.76	10.07	11.22	REARWARD	100.73
DATA 64	PITCH LIFT	-20.76	-0.11	11.22	10.93	REARWARD	100.73
DATA 67	PITCH DROP	-0.46	20.26	10.78	10.69	FORWARD	17.96
DATA 68	PITCH LIFT	20.26	0.26	10.69	11.24	FORWARD	17.96
DATA 71	PITCH DROP	0.57	20.65	10.65	10.89	FORWARD	60.44
DATA 72	PITCH LIFT	20.65	0.07	10.89	11.09	FORWARD	60.44
DATA 76	PITCH LIFT	20.58	-0.75	11.02	11.11	ARMS OFF YOKE ONLY	0
DATA 77	PITCH DROP	-0.75	20.21	11.11	10.93	ARMS OFF YOKE ONLY	0
DATA 78	YAW SWING	0.07	0.07	-20.14	10.56	ARMS OFF YOKE ONLY	0
DATA 79	PITCH DROP PITCH LIFT	0 20	20 0	11 11	11 11	ARMS OFF YOKE ONLY	0
DATA 80 Dual move	CCW YAW SWING CW YAW SWING	0 0	0 0	11 30	30 11	ARMS OFF YOKE ONLY	0
DATA 81 Quadrille move	CCW YAW SWING PITCH DROP CW YAW SWING PITCH LIFT	0 0 20 20	0 20 20 0	10 30 30 10	30 30 10 10	ARMS OFF YOKE ONLY	0
DATA 82 Dual move	PITCH DROP PITCH LIFT	0 20	20 0	11 11	11 11	FORWARD	17.96
DATA 83 Dual move	PITCH DROP PITCH LIFT	0 20	20 0	11 11	11 11	FORWARD	60.44

TABLE 14 - PITCH/YAW JOINT DRIVE TORQUE DATA

FILE	FIGURE	MANEUVER	INPUT TORQUES, ft lbs	LOAD TORQUE, ft lbs	EFFICIENCY,
Data					
76	58a	Pitch Lift	±1.1	0	---
79	72a	Pitch Lift	±1.04		
68	75a	Pitch Lift	±3.71	17.96	70.6
82	73a	Pitch Lift	±3.54		73.8
48	76a	Pitch Lift	±9.	60.44	97.9
72	77a	Pitch Lift	±10.		88.1
64	78a	Pitch Lift	±15.	100.73	97.9
77	58b	Pitch Drop	±0.16	0	---
79	72a	Pitch Drop	±0.32		
67	75b	Pitch Drop	±2.58	17.96	98.5
82	73a	Pitch Drop	±2.9		> 100
49	76b	Pitch Drop	±8.0	60.44	94.0
71	77b	Pitch Drop	±8.51		96.5
63	78b	Pitch Drop	±14.4	100.73	98.1

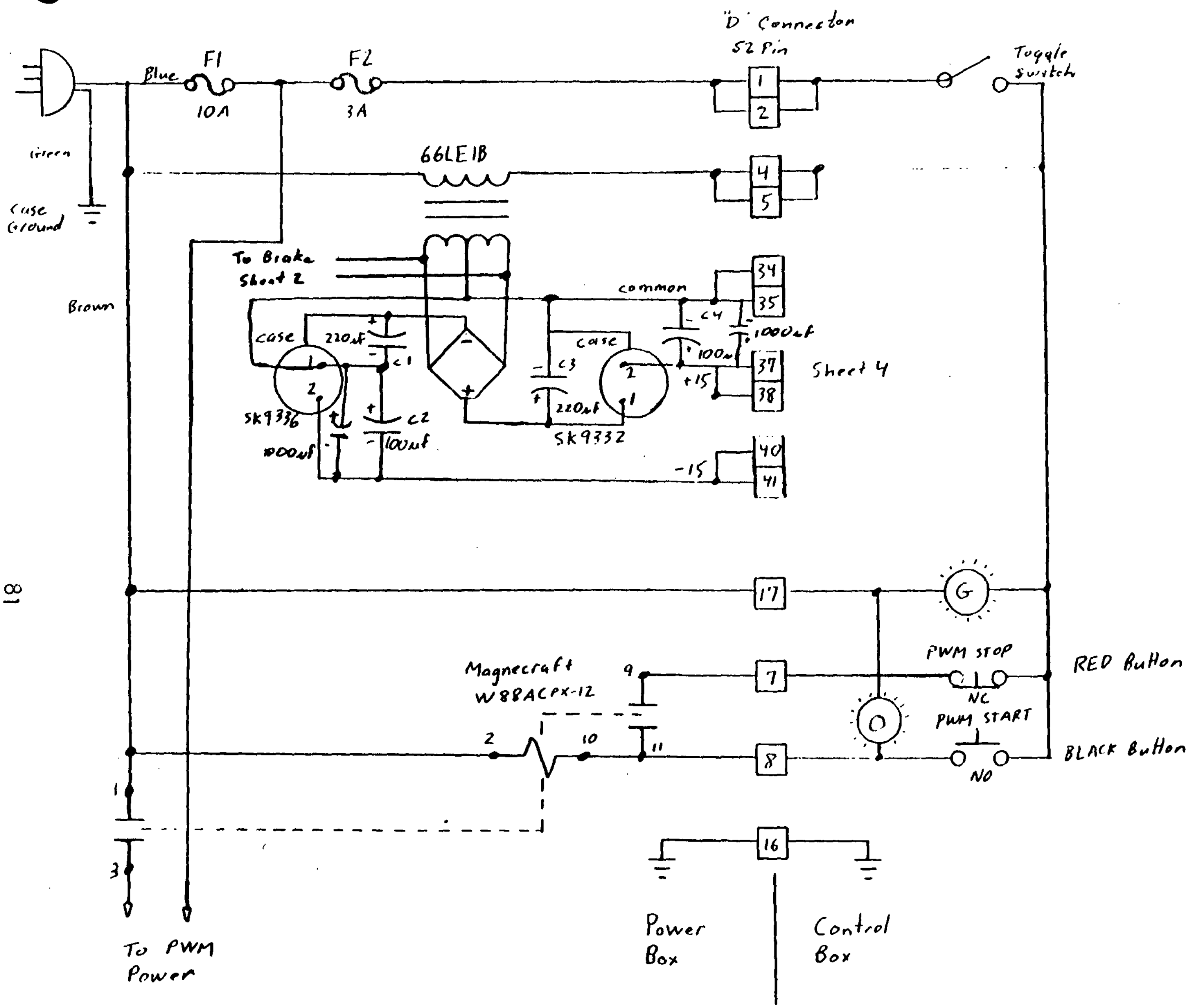


FIGURE 1 - POWER AND POWER CONTROL

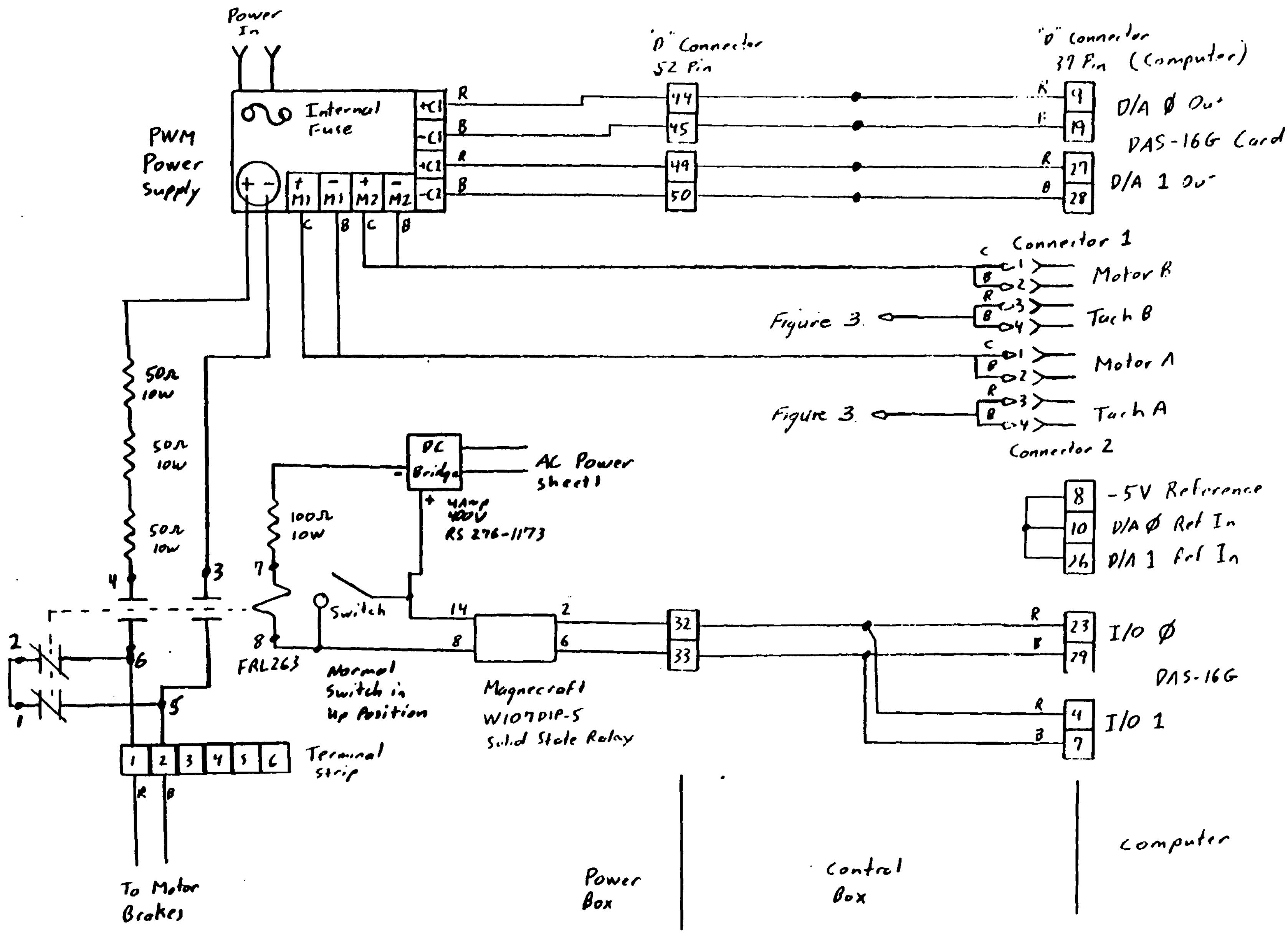


FIGURE 2 - MOTOR POWER AND MOTOR BRAKE POWER

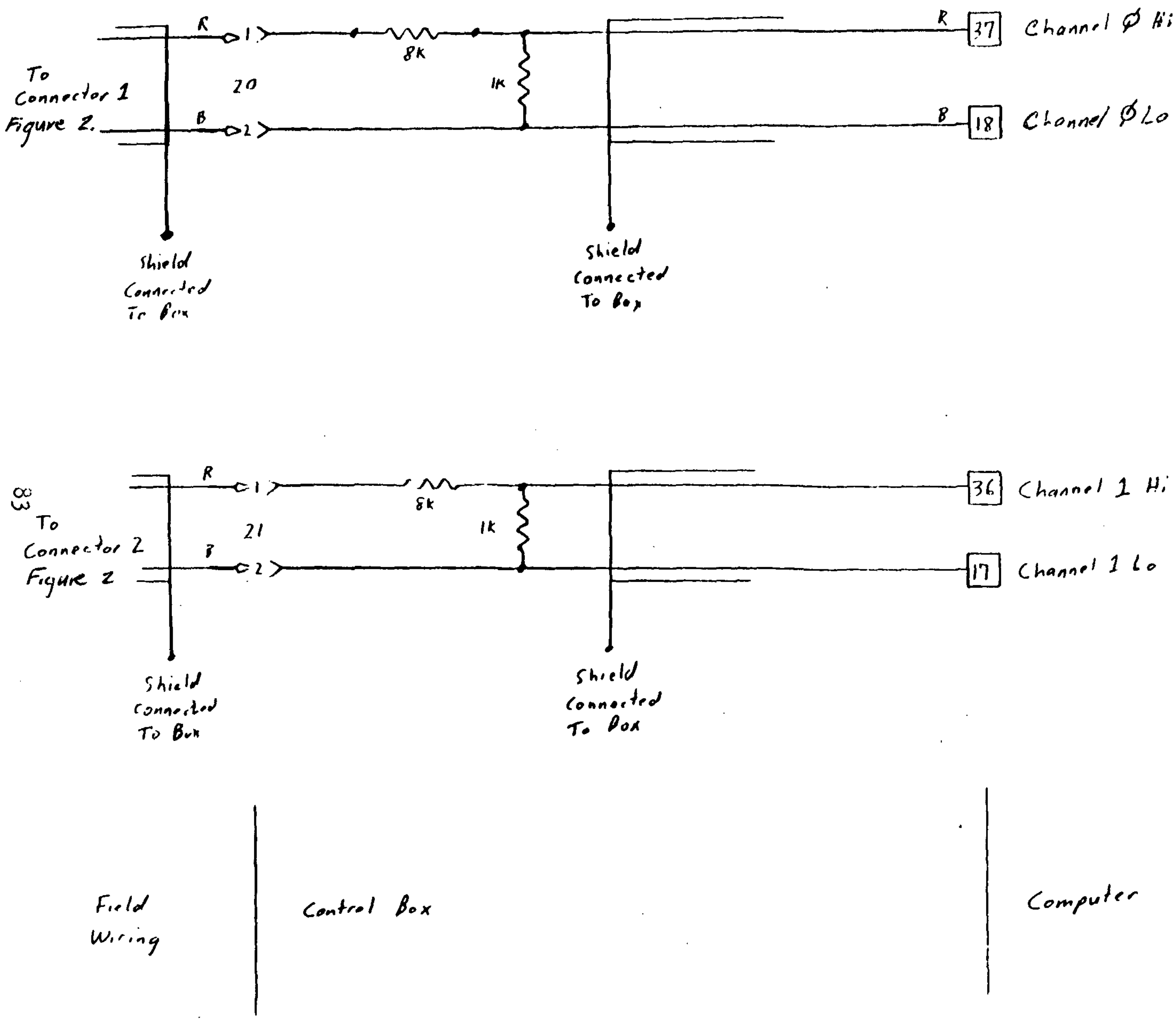


FIGURE 3 - MOTOR TACHOMETER SIGNALS

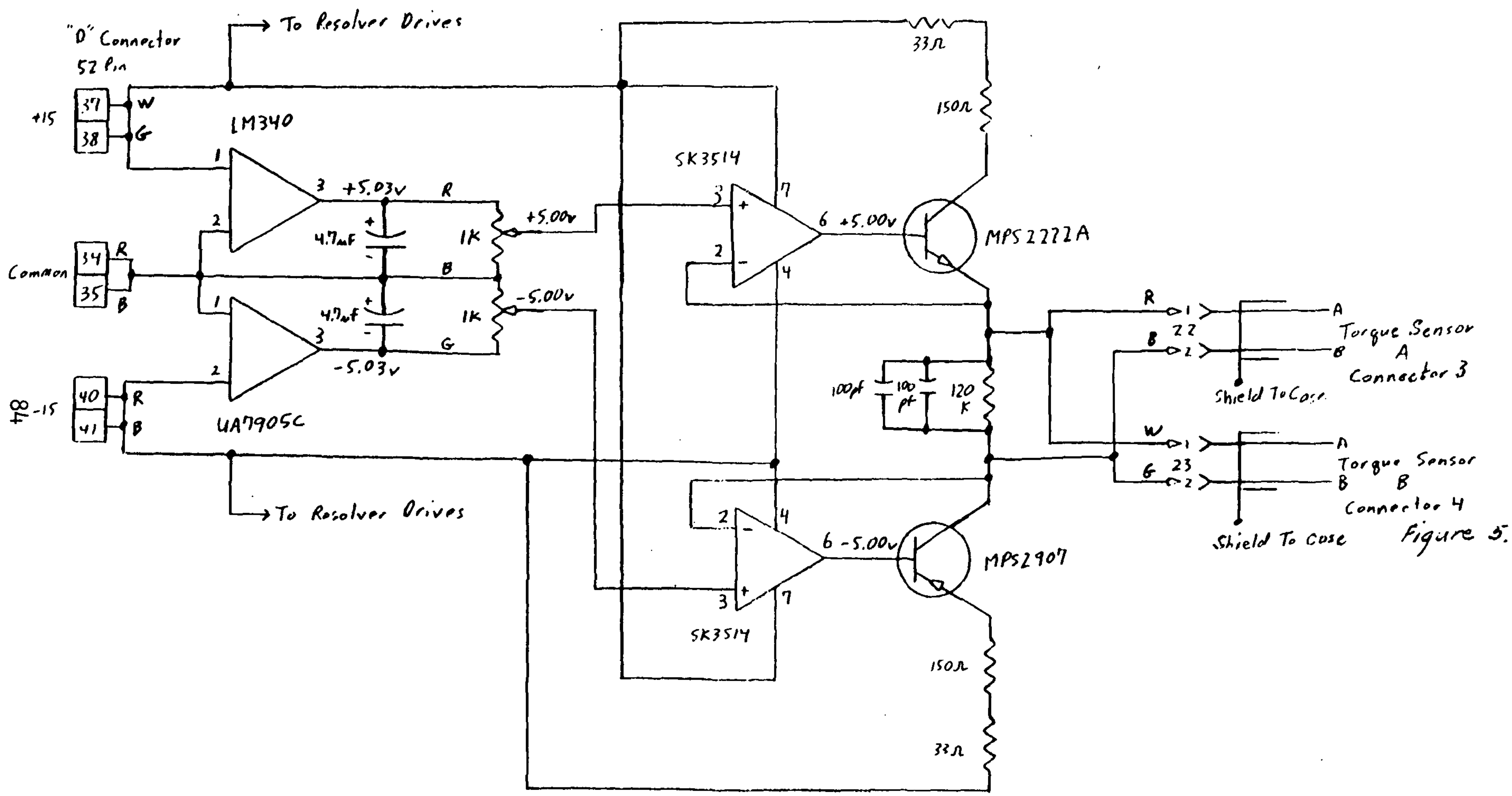


FIGURE 4 – TORQUE SENSOR REFERENCE SIGNAL

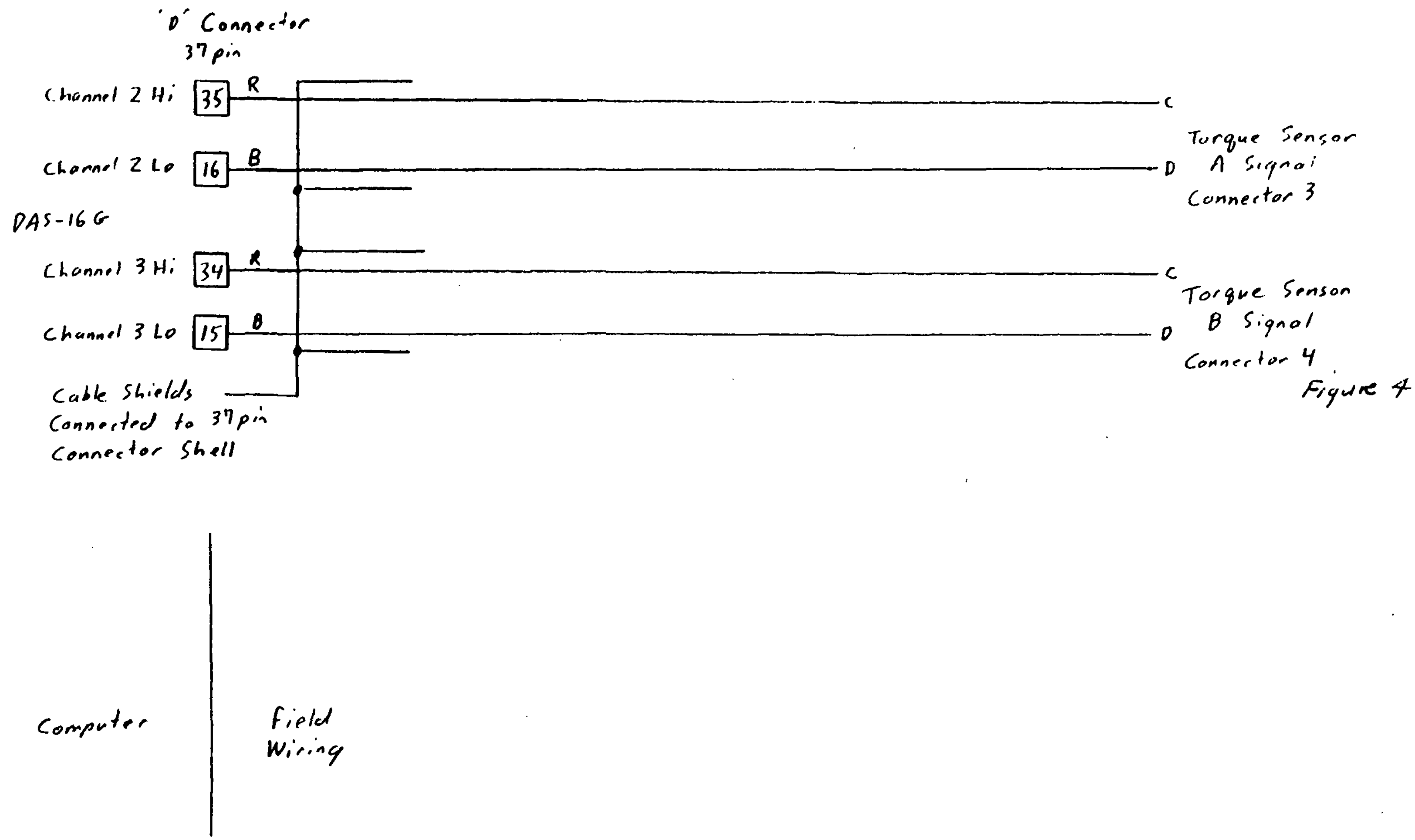


FIGURE 5 – TORQUE SENSOR SIGNAL

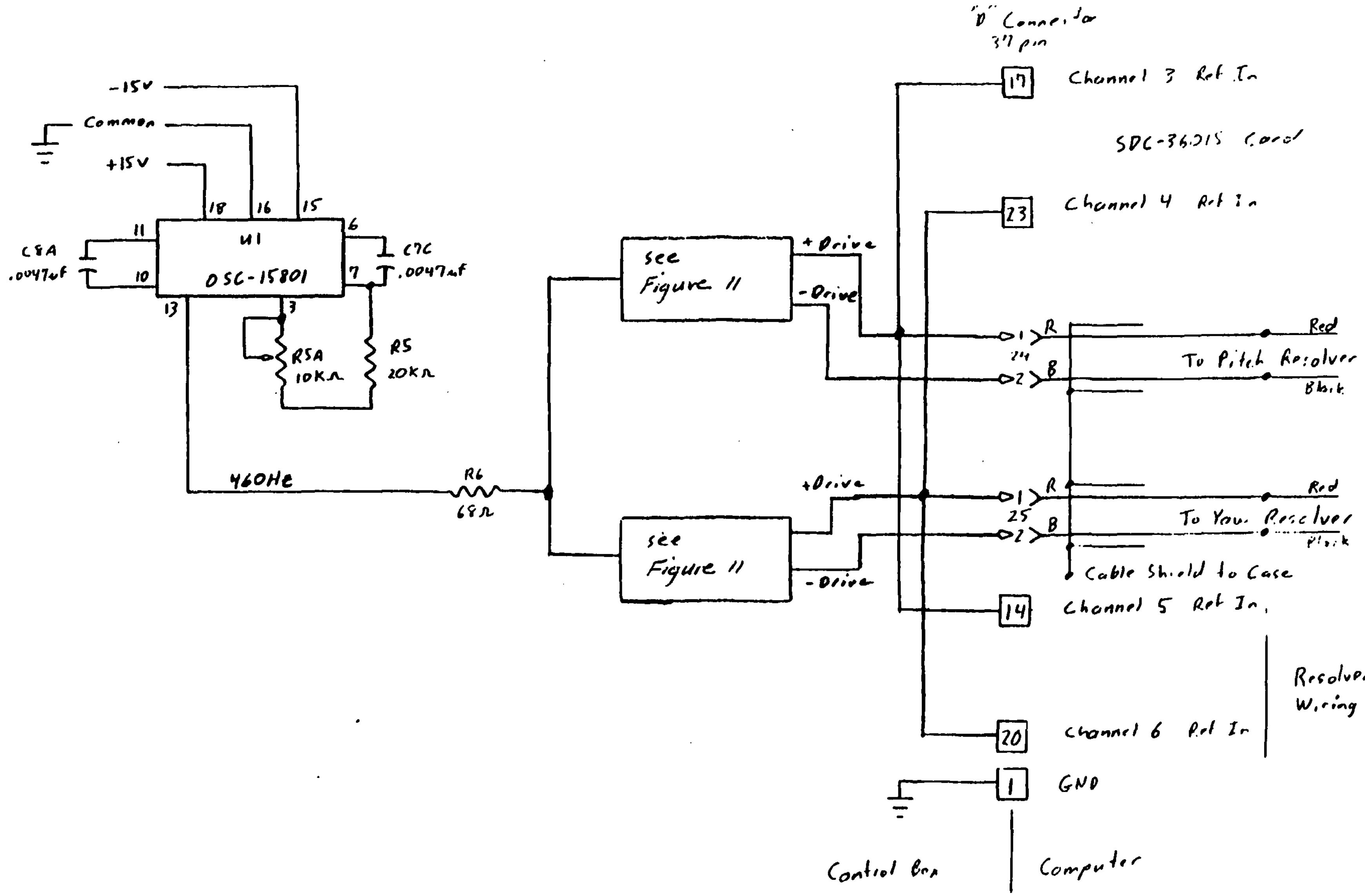


FIGURE 6 - PITCH AND YAW RESOLVER REFERENCE SIGNAL

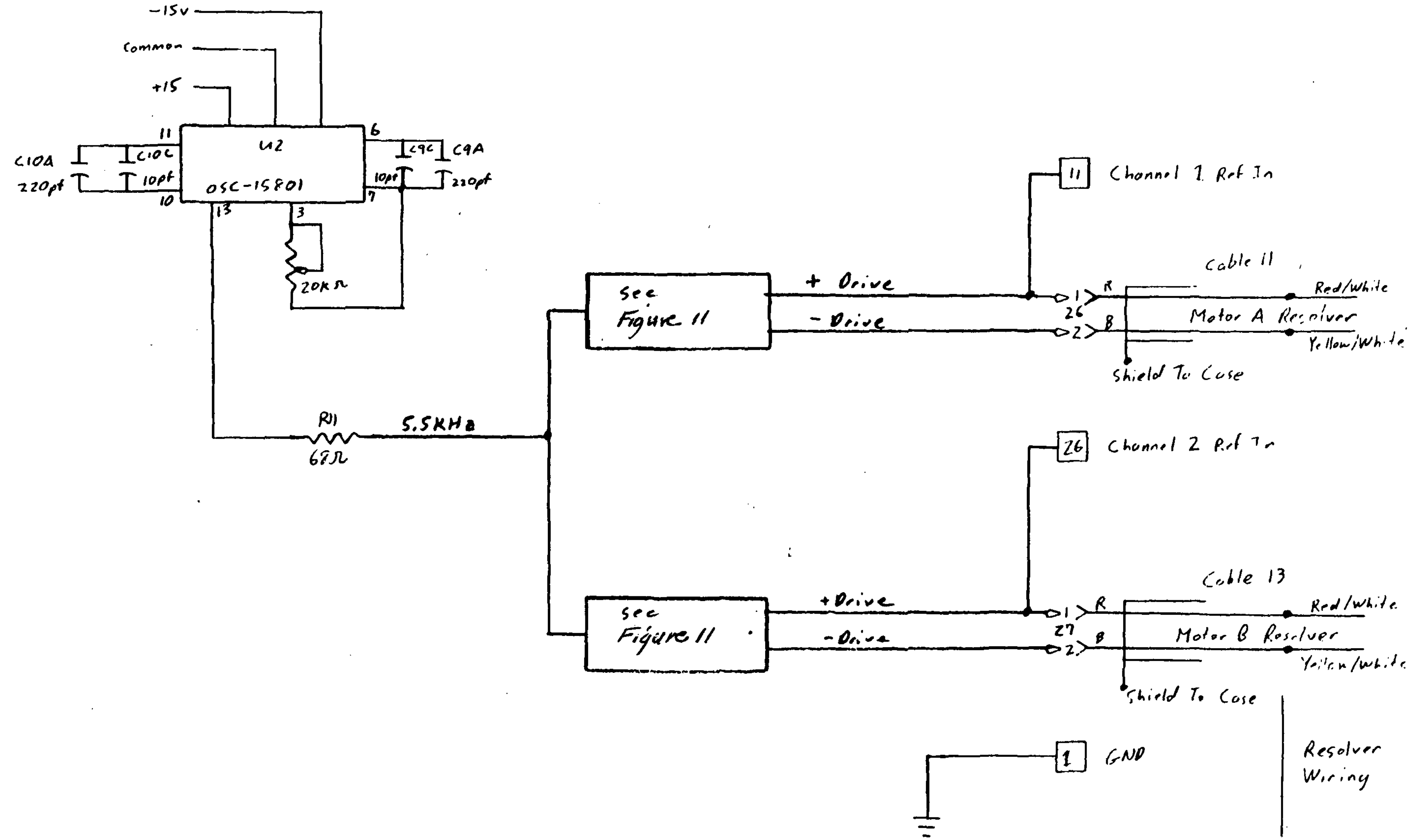


FIGURE 7 - MOTOR SHAFT RESOLVER REFERENCE SIGNAL

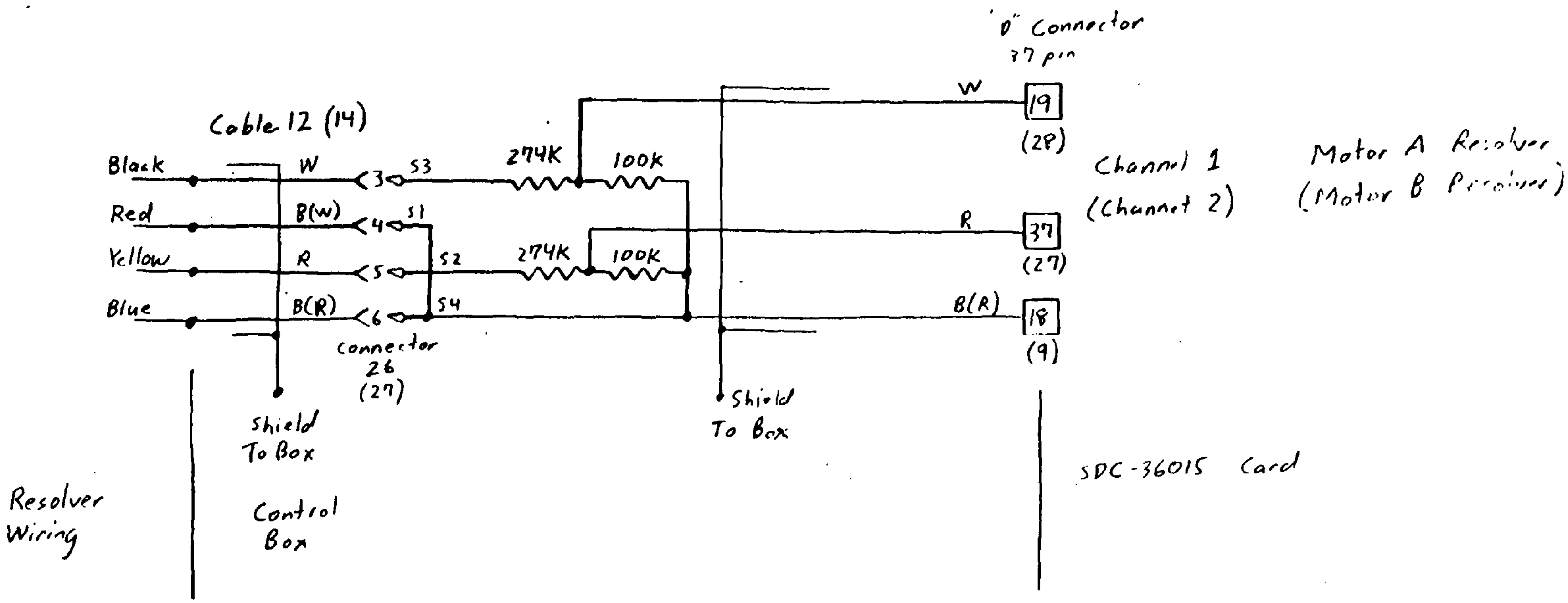


FIGURE 8 – MOTOR SHAFT RESOLVER POSITION SIGNAL

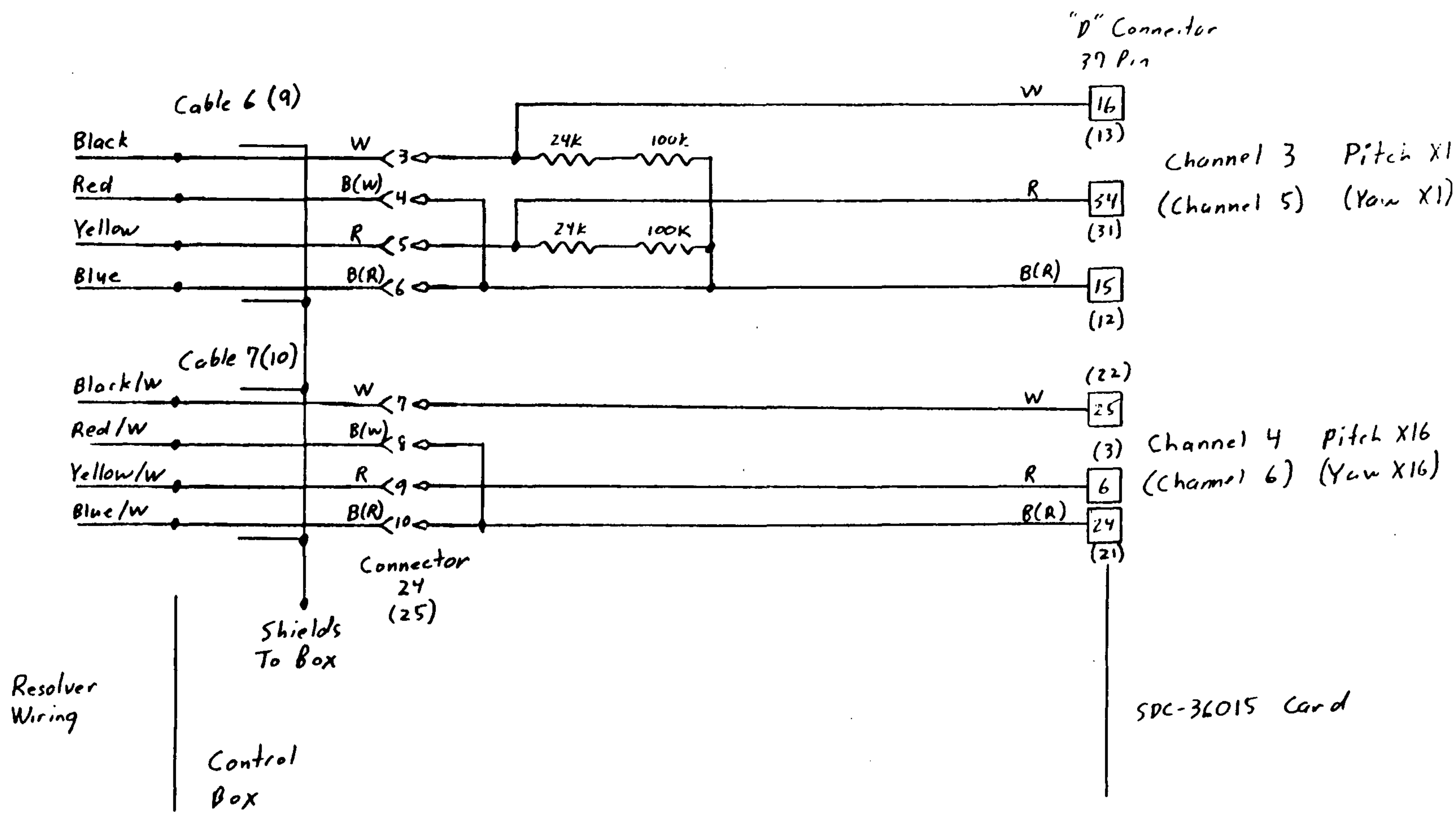


FIGURE 9 - PITCH AND YAW RESOLVER POSITION SIGNAL

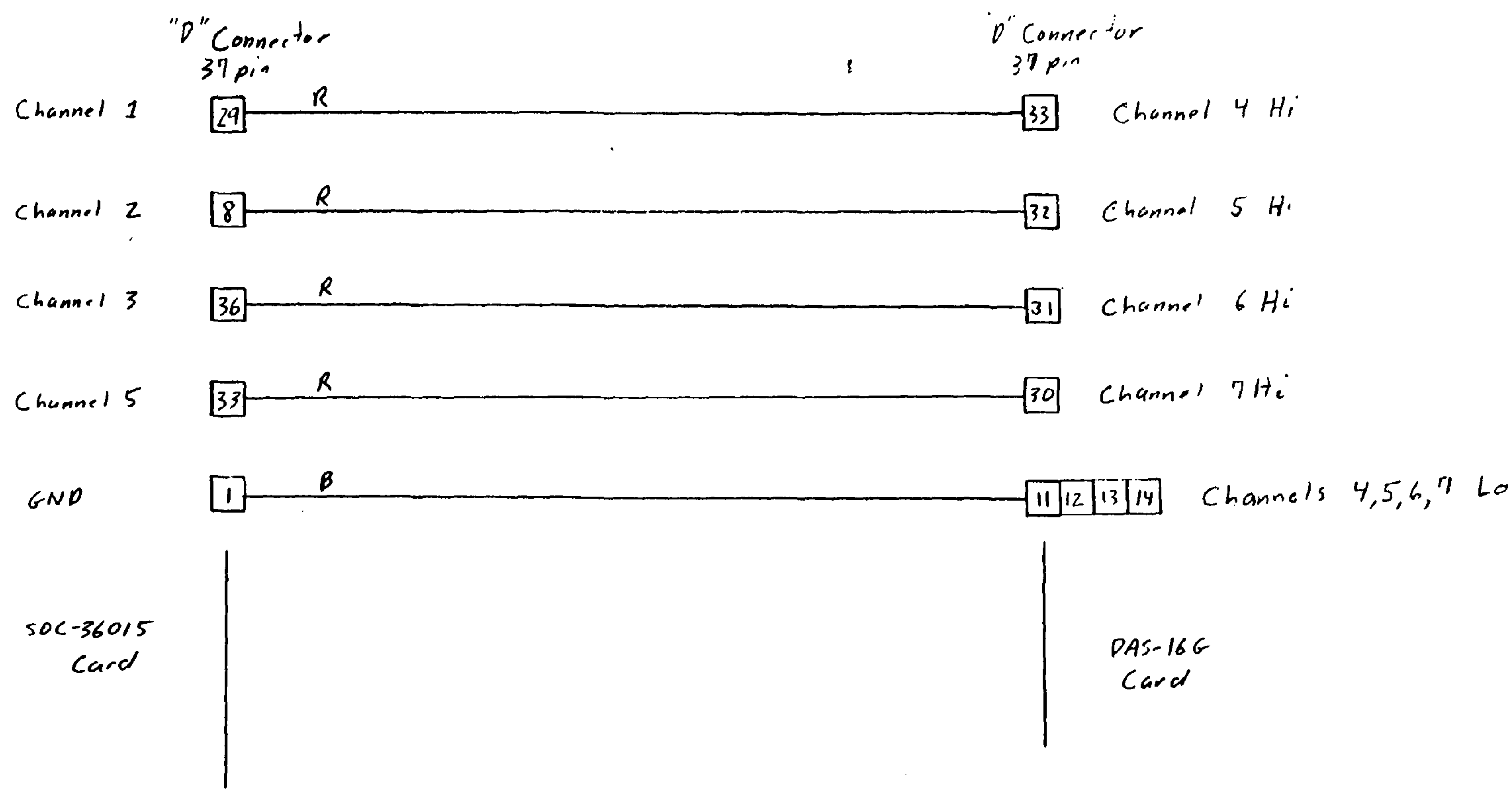


FIGURE 10 - PITCH, YAW AND MOTOR SHAFT RESOLVER VELOCITY SIGNALS

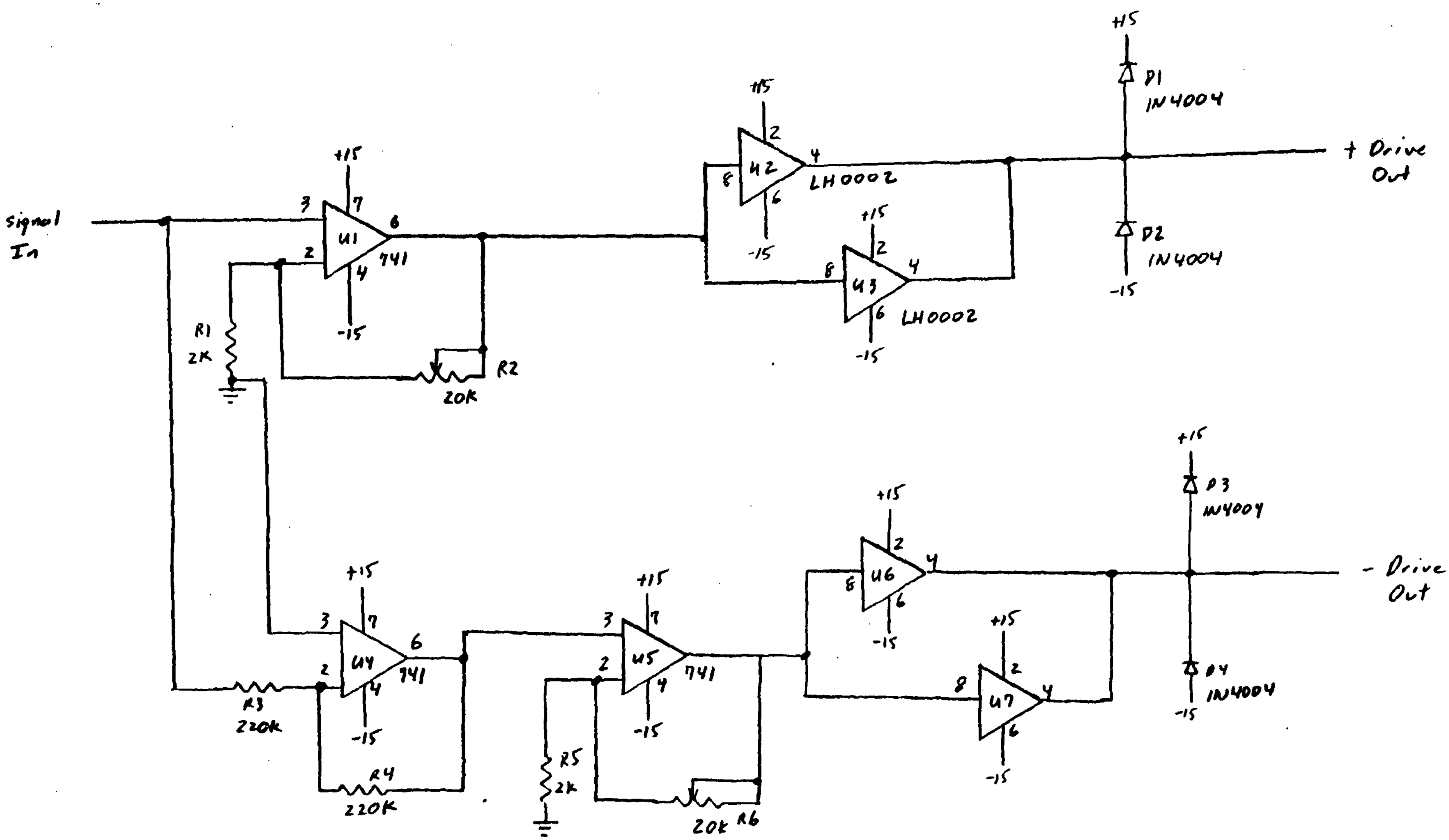


FIGURE 11 - PITCH, YAW AND MOTOR RESOLVER SIGNAL POWER AMPLIFIERS

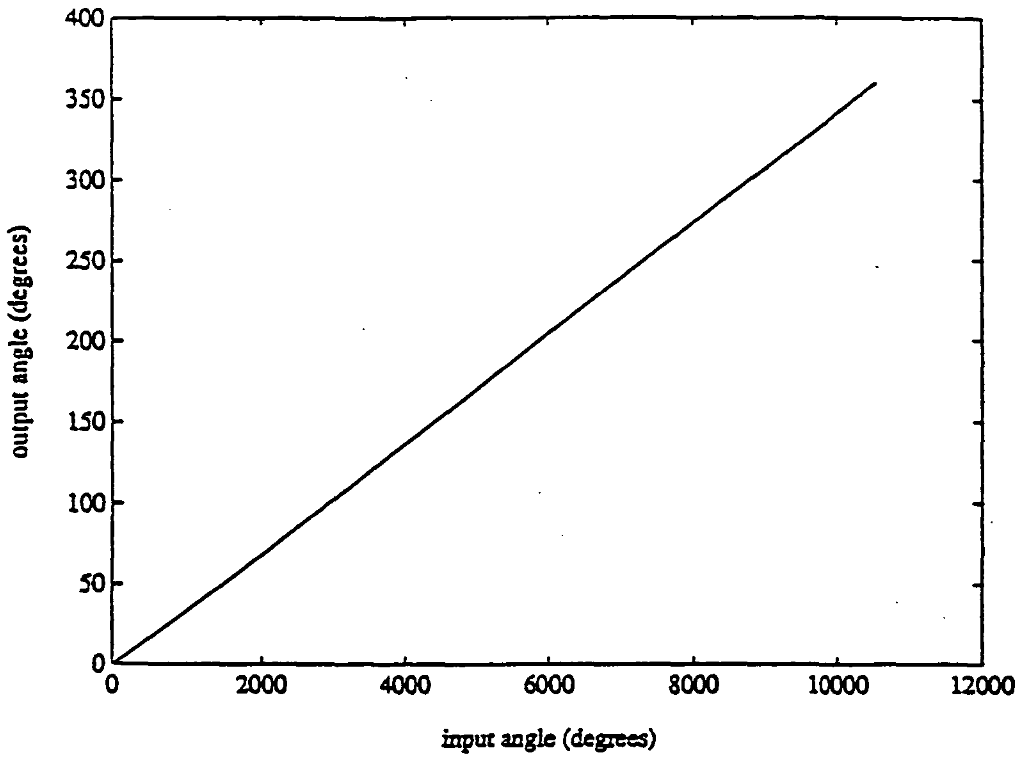


FIGURE 12 - Linearity of Hinge-Joint Drive

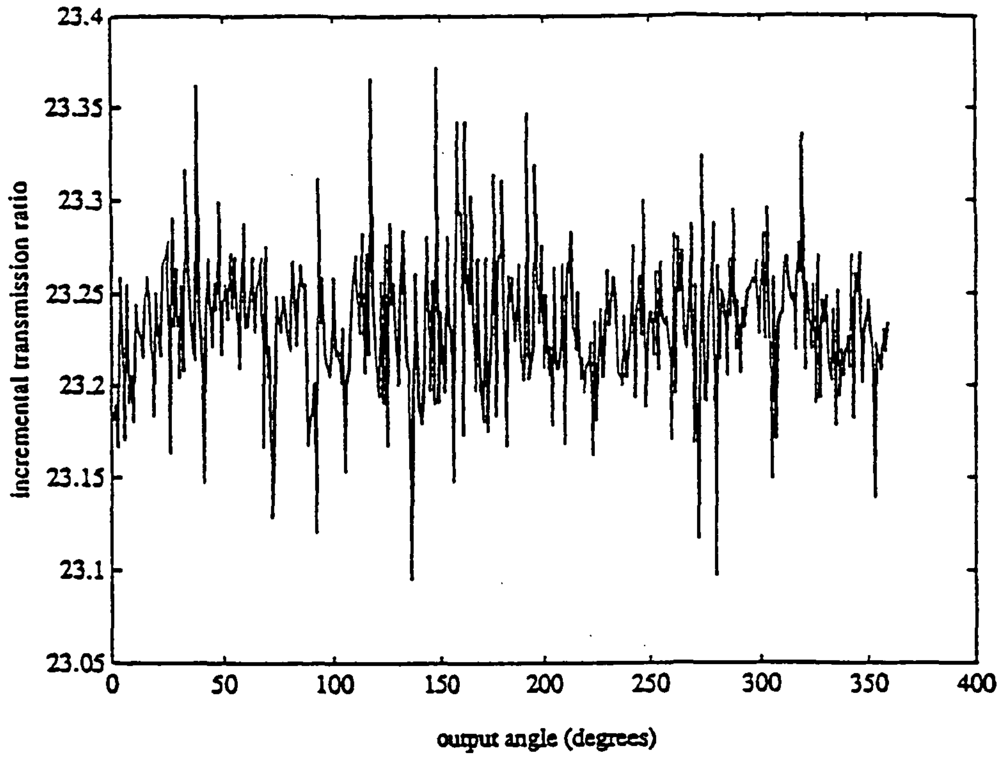


FIGURE 13 - Linearity of Steel Wrist-Roll Drive

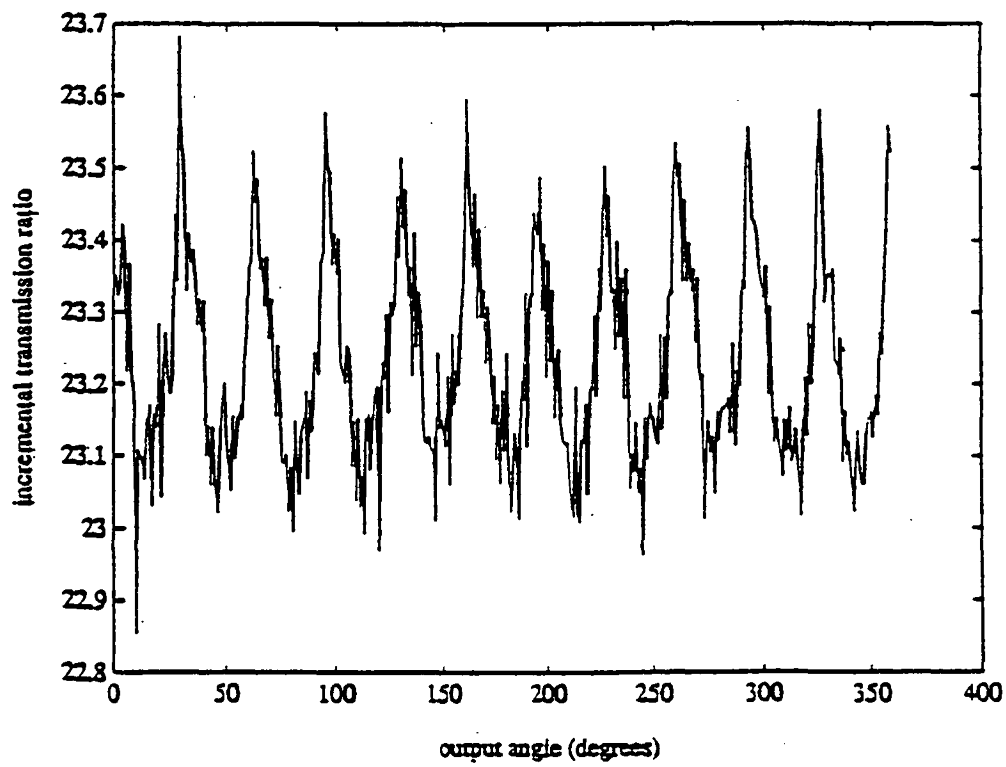


FIGURE 14 - Linearity of Plastic Wrist-Roll Drive

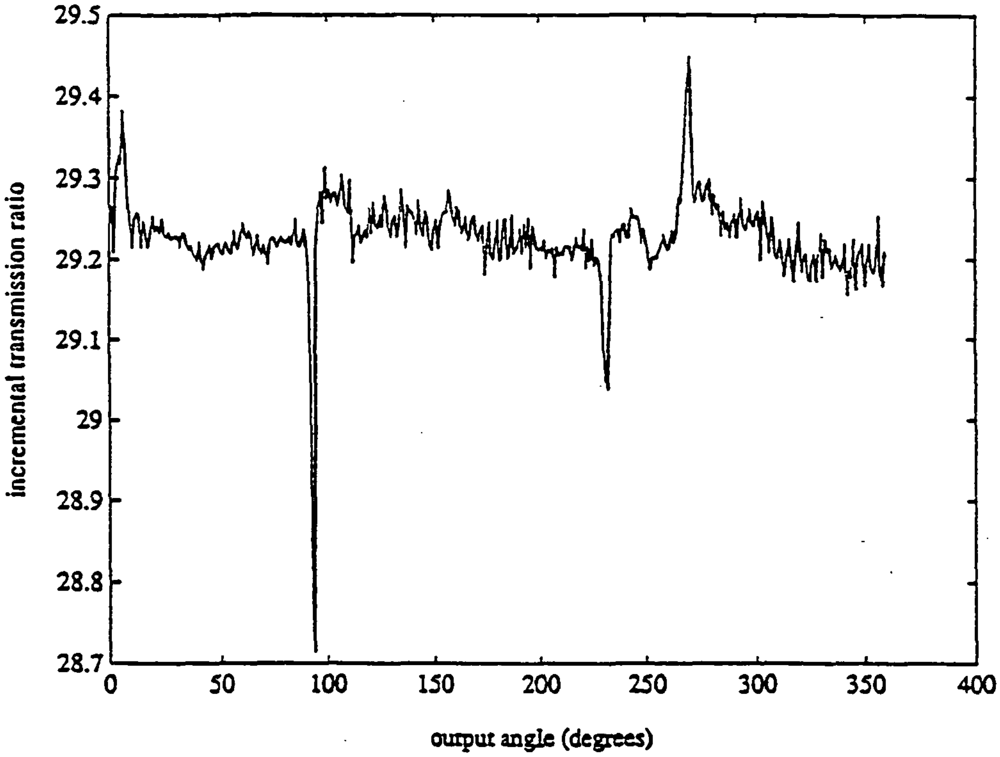


FIGURE 15 - Incremental Linearity of Hinge-Joint Drive

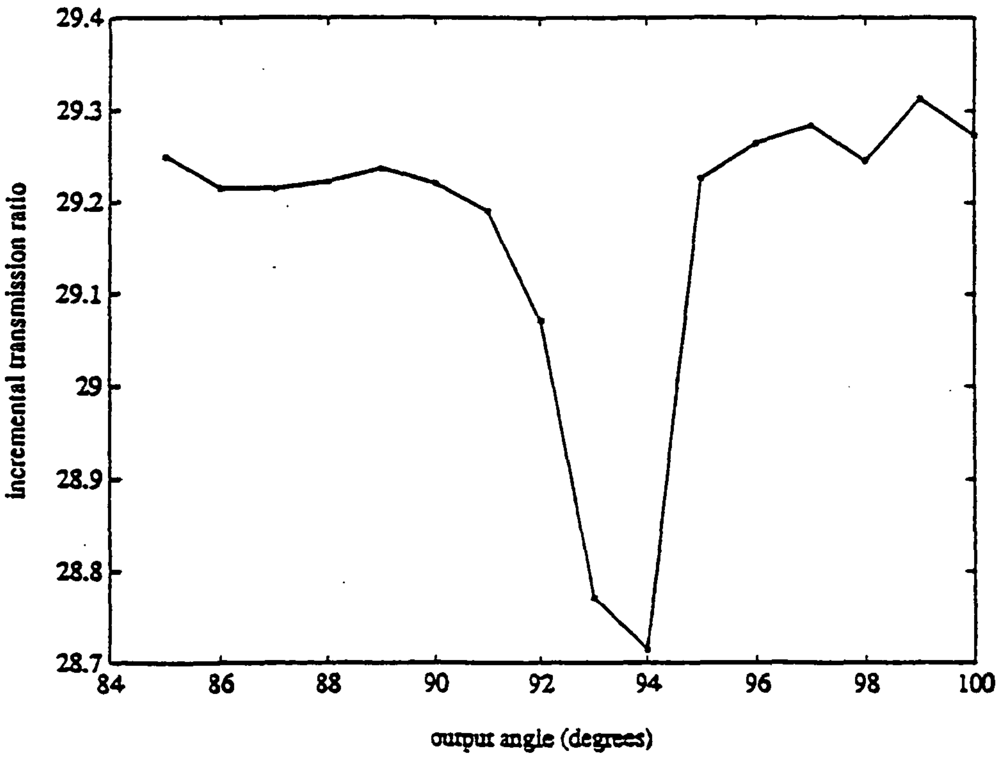


FIGURE 16 - Hinge-Joint Drive worst-case nonlinearity

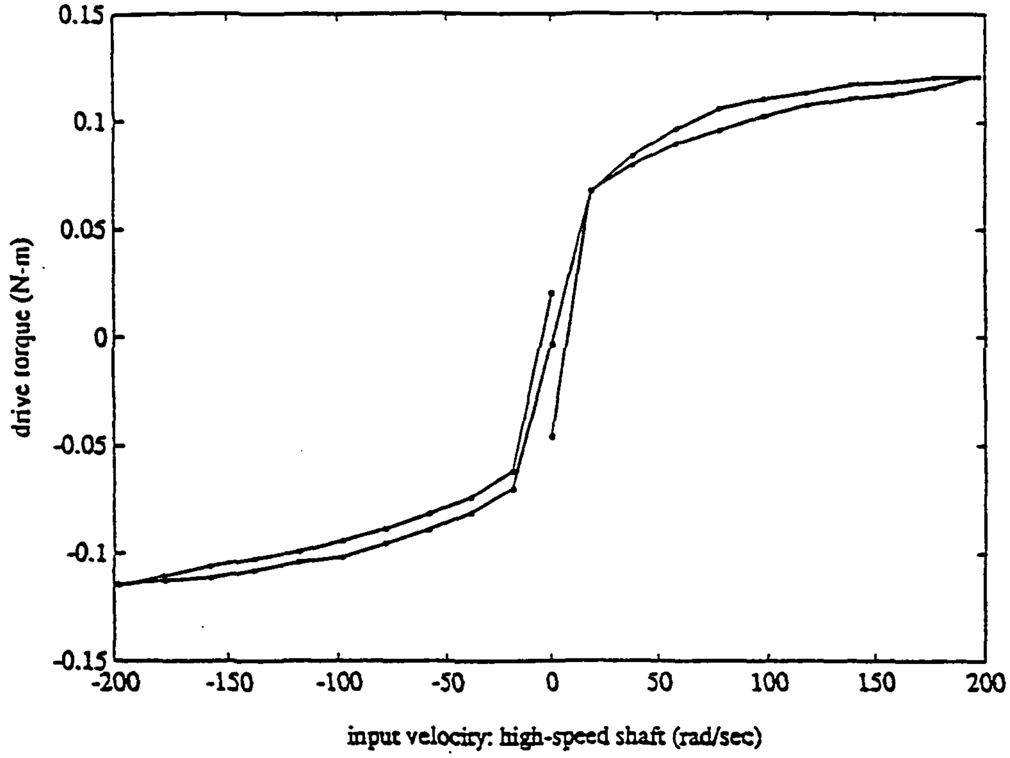


FIGURE 17 - Friction of Steel Wrist-Roll drive, unloaded, as a speed reducer (dry).

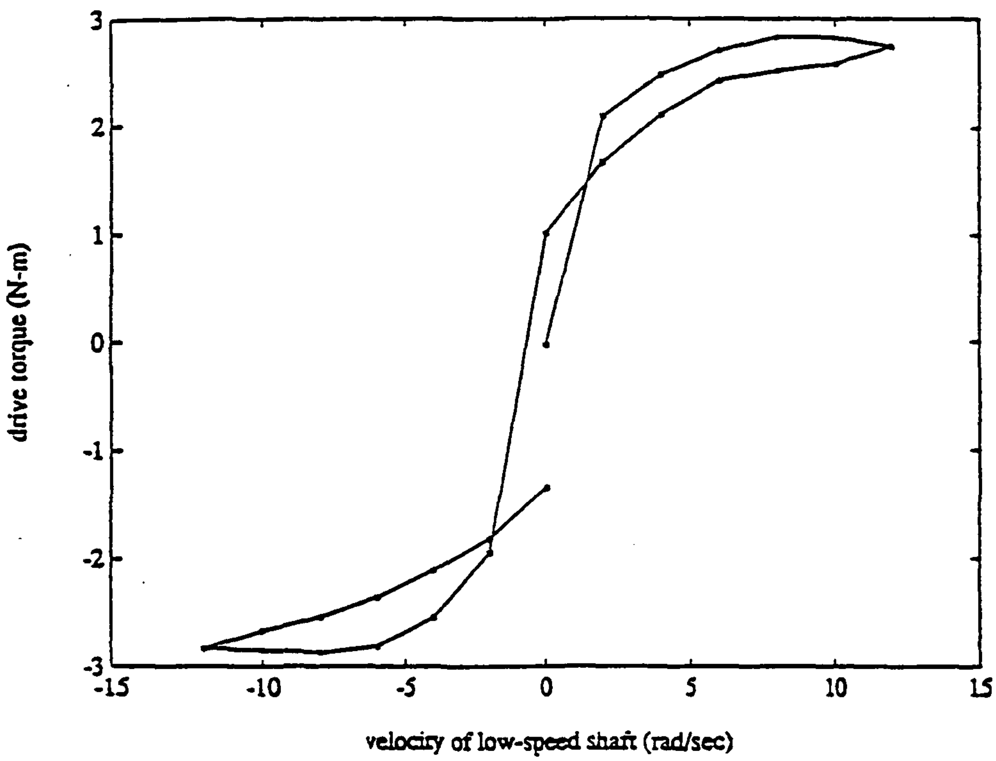


FIGURE 18 - Friction of Steel Wrist-Roll drive, unloaded, as a speed increaser (dry).

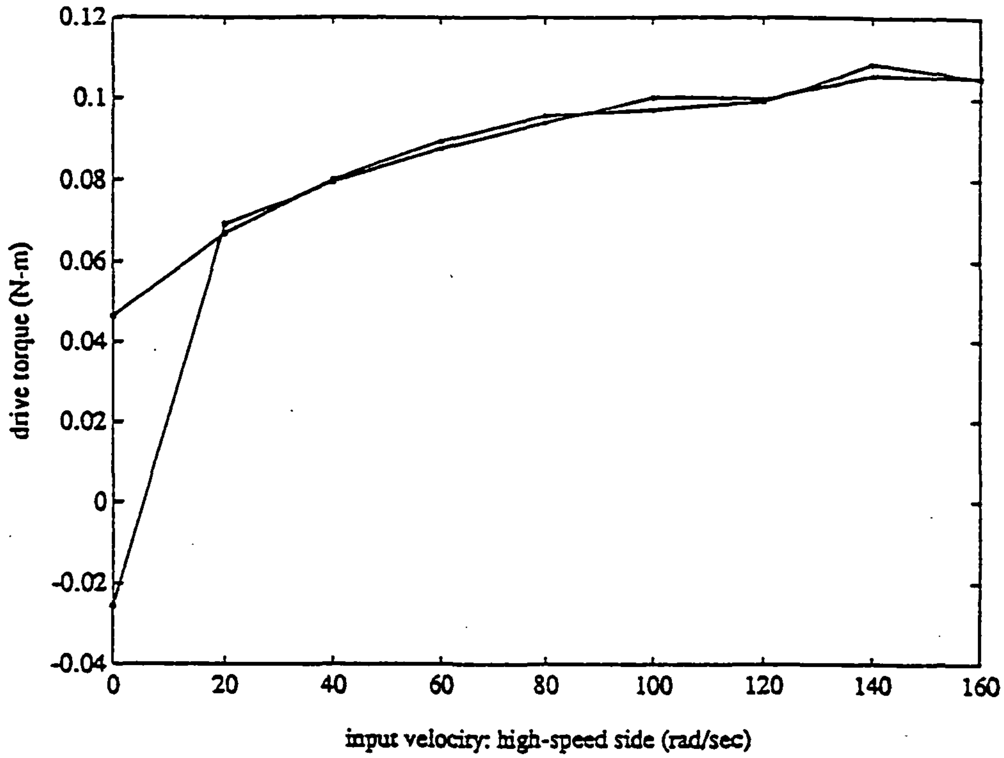


FIGURE 19 - Friction of Steel Wrist-Roll drive, unloaded, as a speed reducer (lubricated).

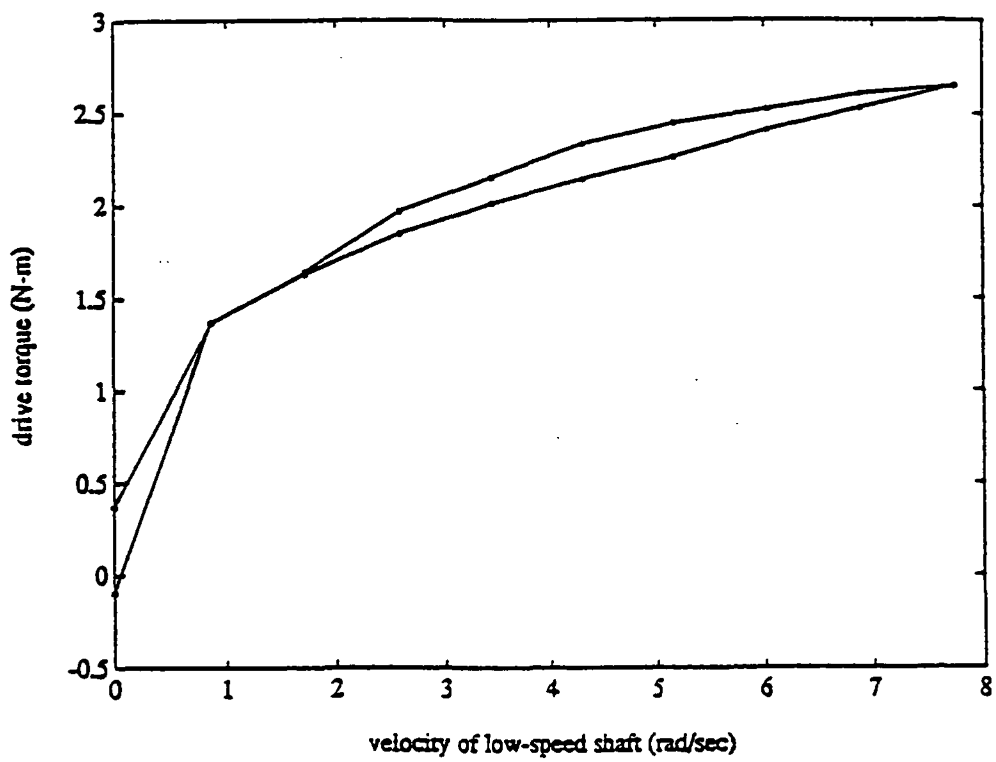


FIGURE 20 - Friction of Steel Wrist-Roll drive, unloaded, as a speed increaser (lubricated).

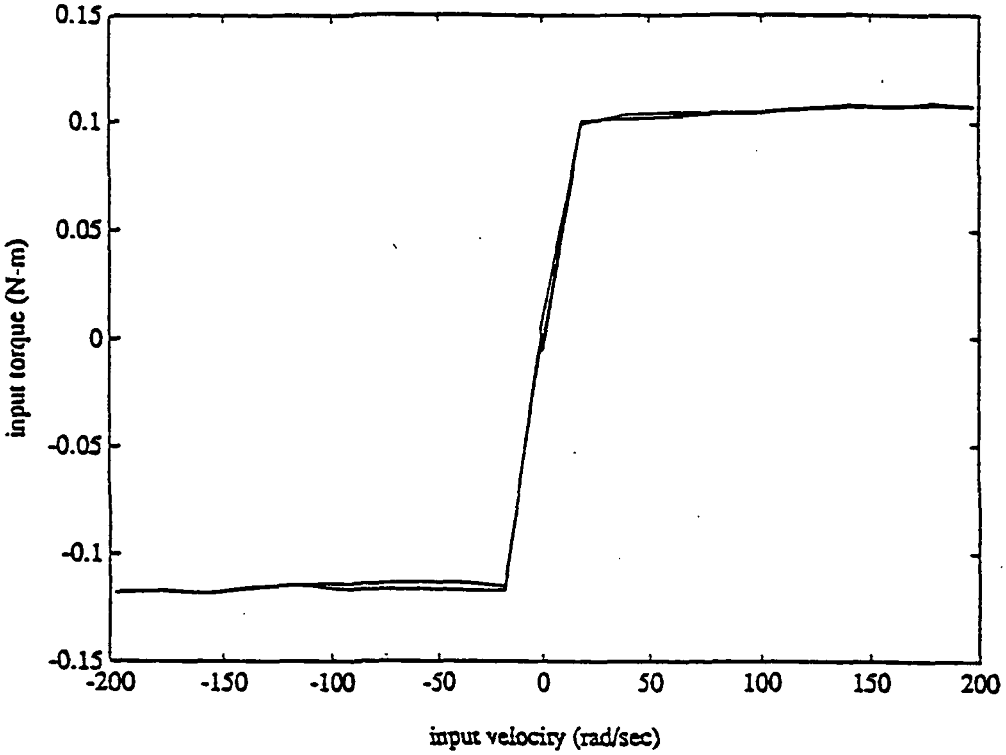
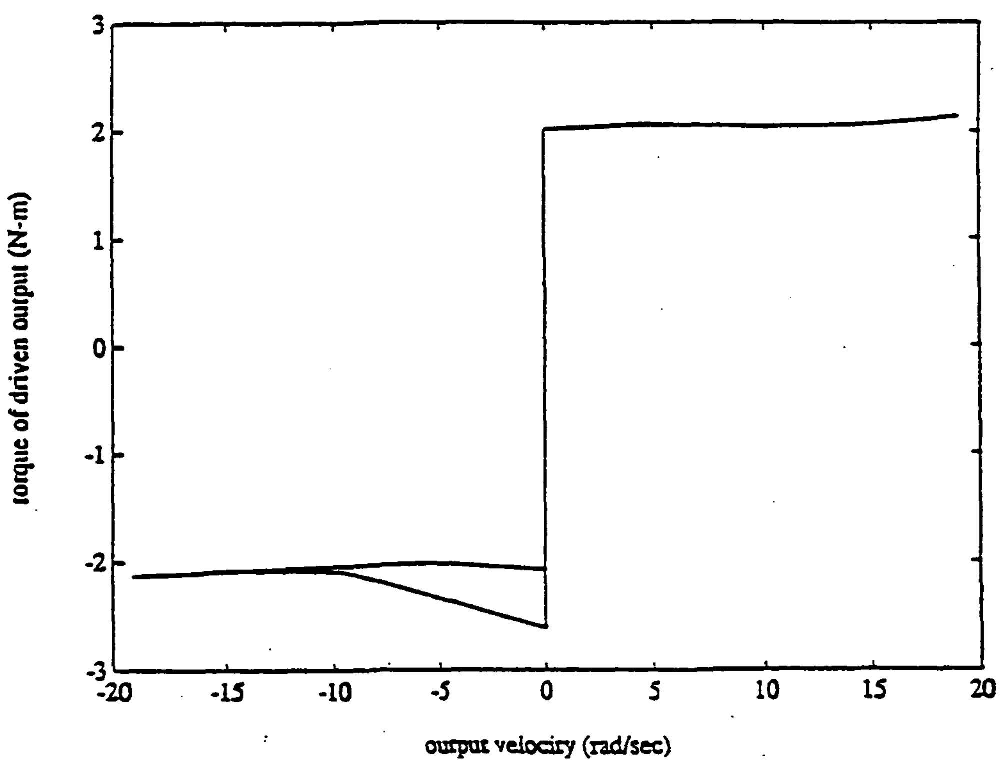


FIGURE 21 - Friction of Plastic Wrist-Roll drive
as a speed reducer.



FUGURE 22 - Friction of Plastic Wrist-Roll drive
as a speed increaser.

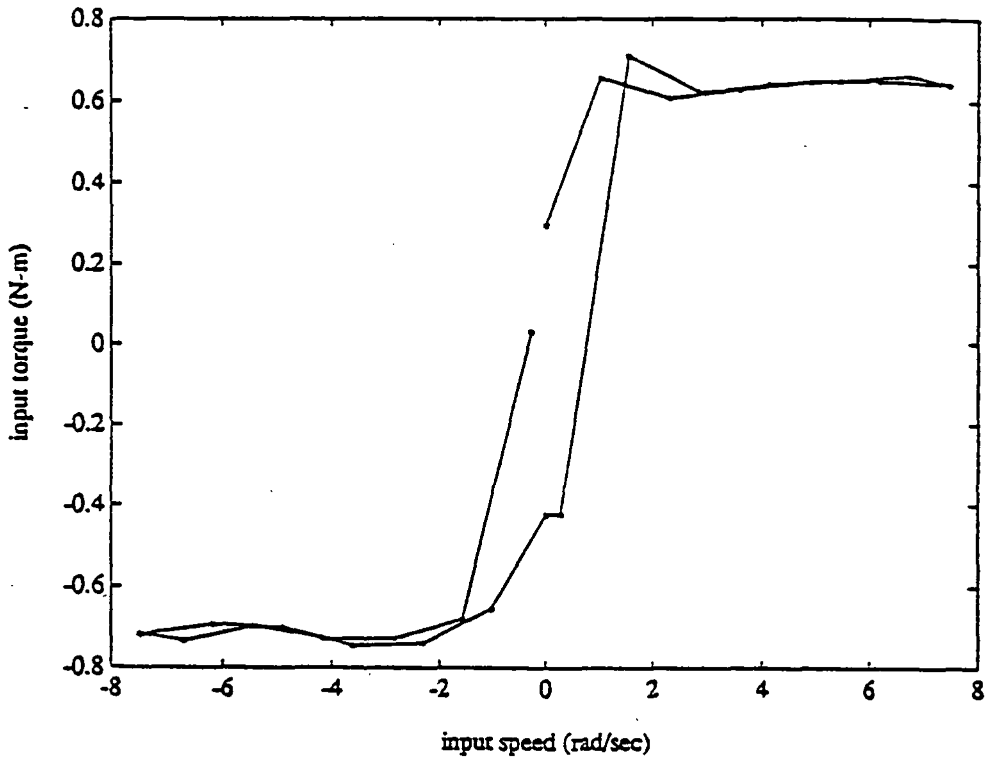


FIGURE 23 - Friction of Hinge-Joint drive, unloaded, as a speed reducer.

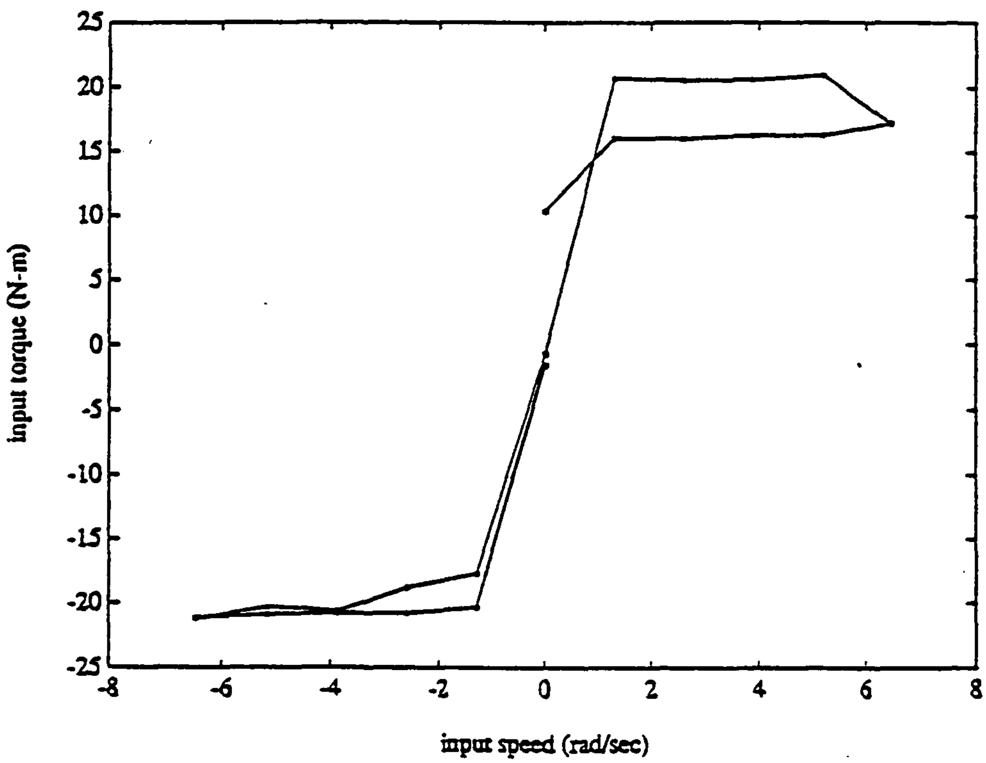


FIGURE 24 - Friction of Hinge-Joint drive, unloaded, as a speed increaser.

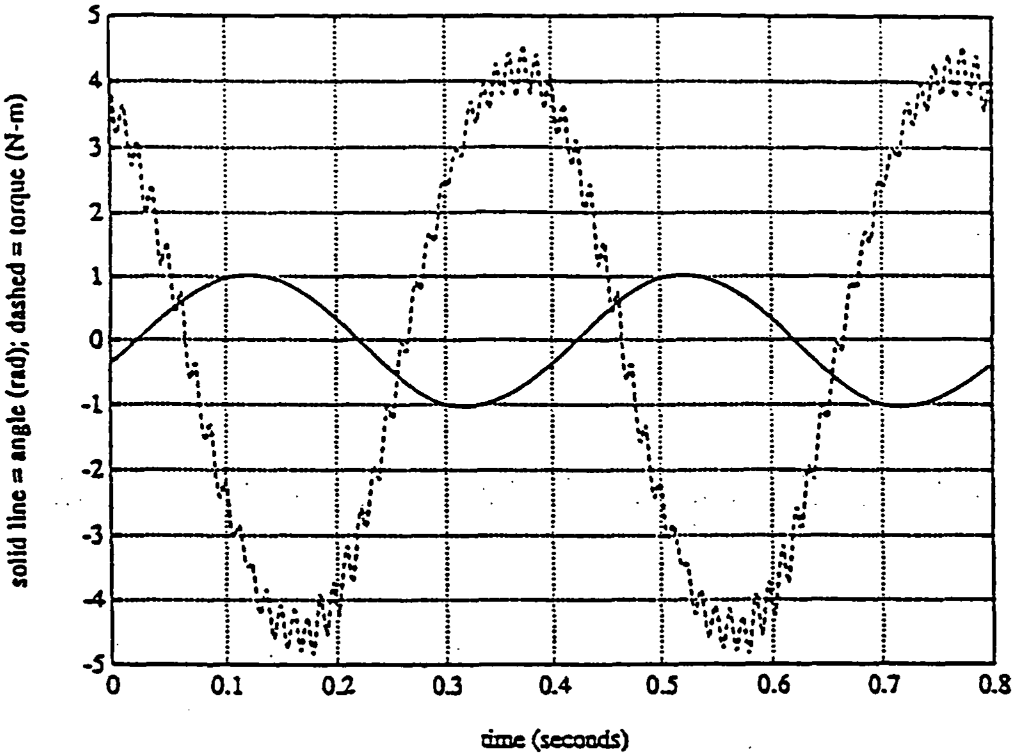


FIGURE 25 - Output Inertia Test of the Steel Wrist-Roll drive.

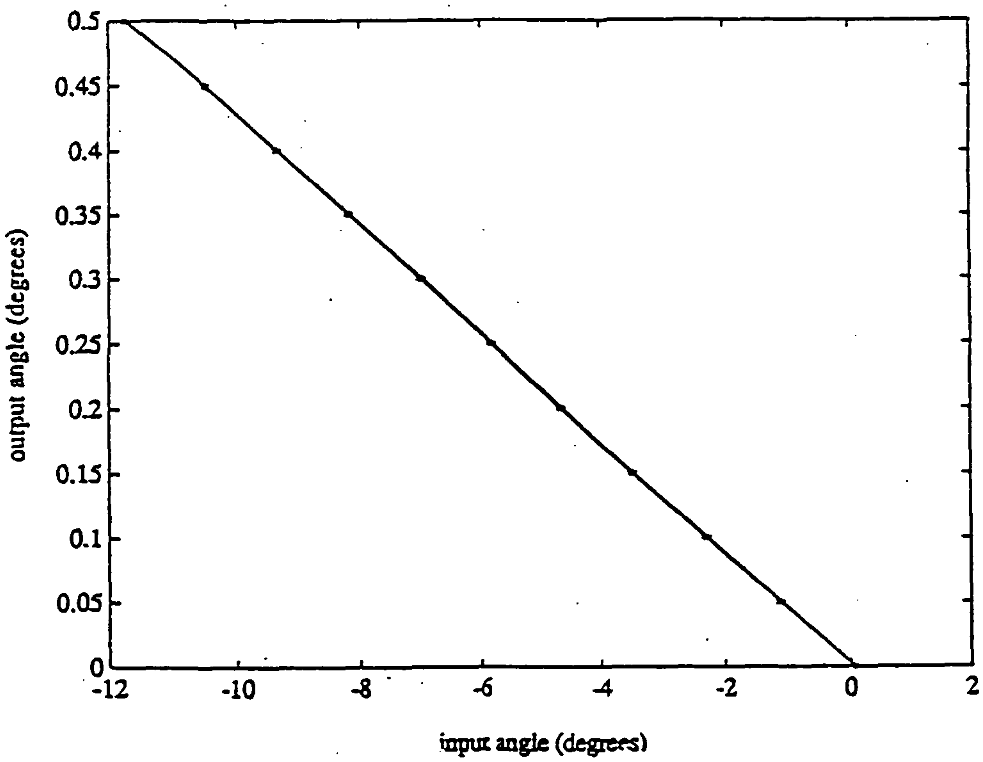


FIGURE 26 - Backlash data for the Steel Wrist-Roll drive.

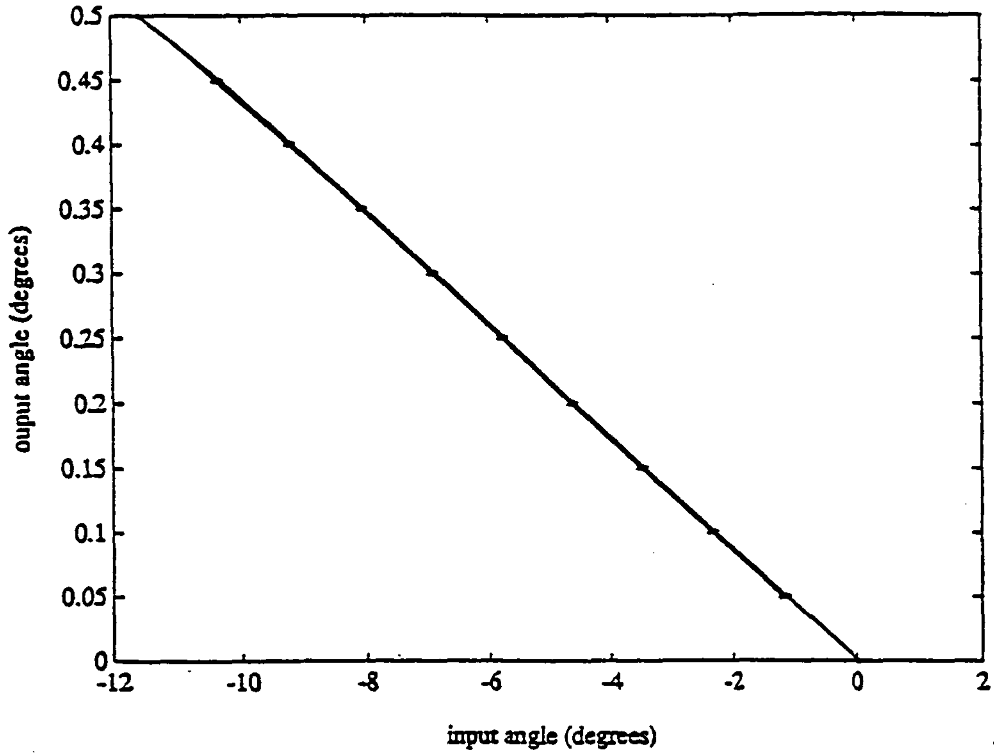


FIGURE 27 - Backlash data for the Plastic Wrist-Roll drive.

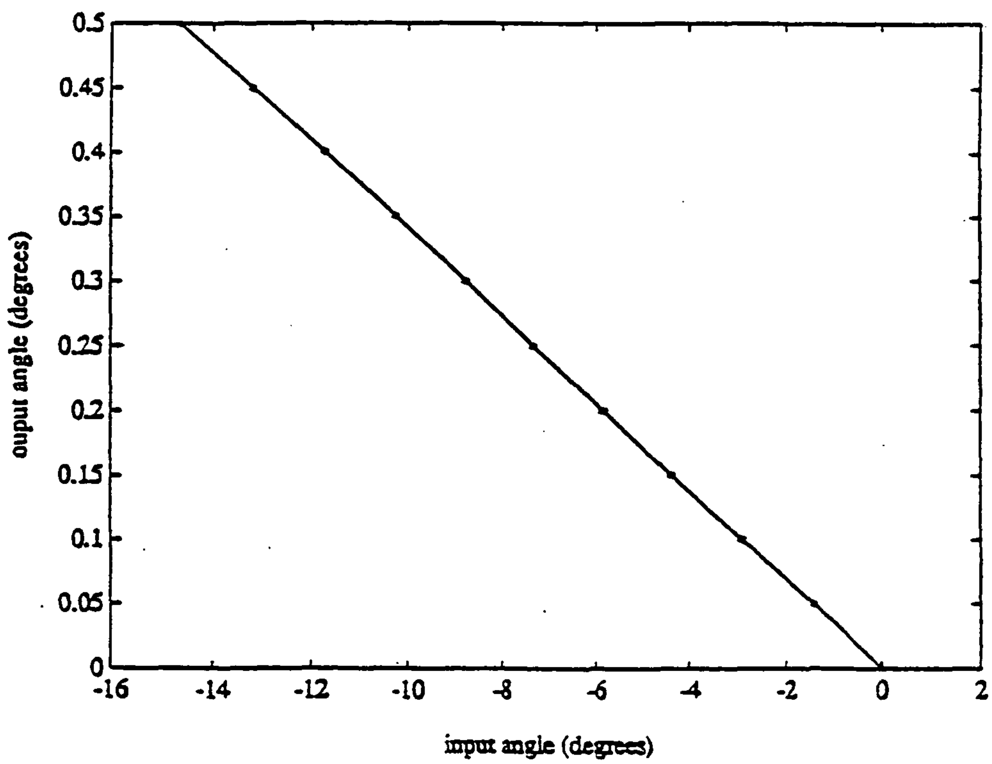
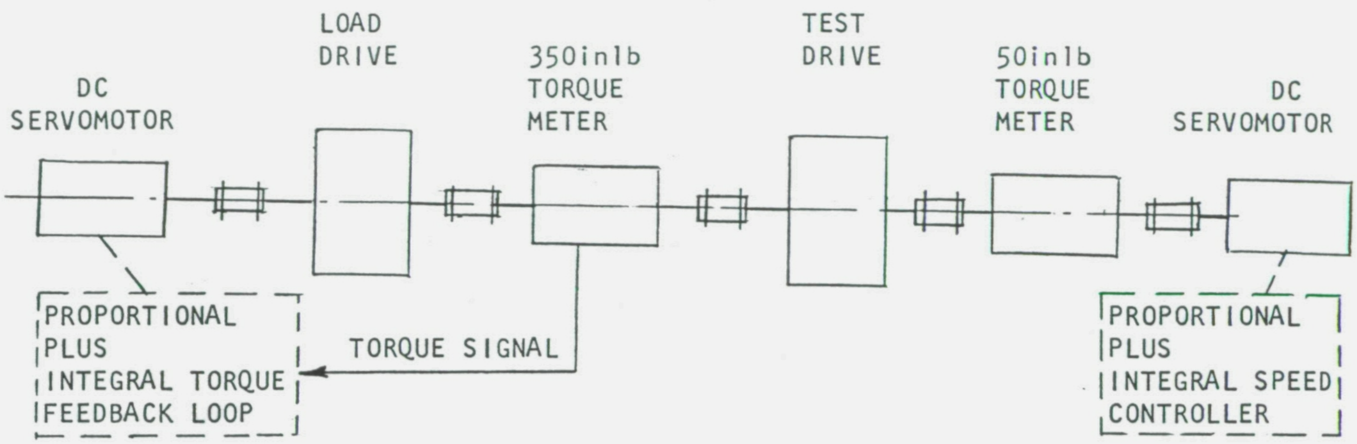
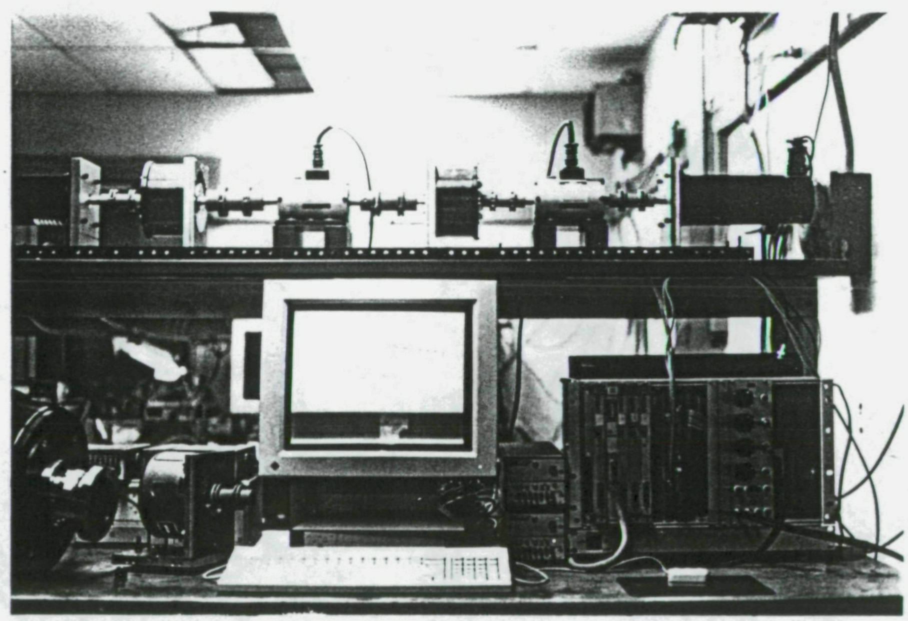


FIGURE 28 - Backlash data for the Hinge-Joint drive.



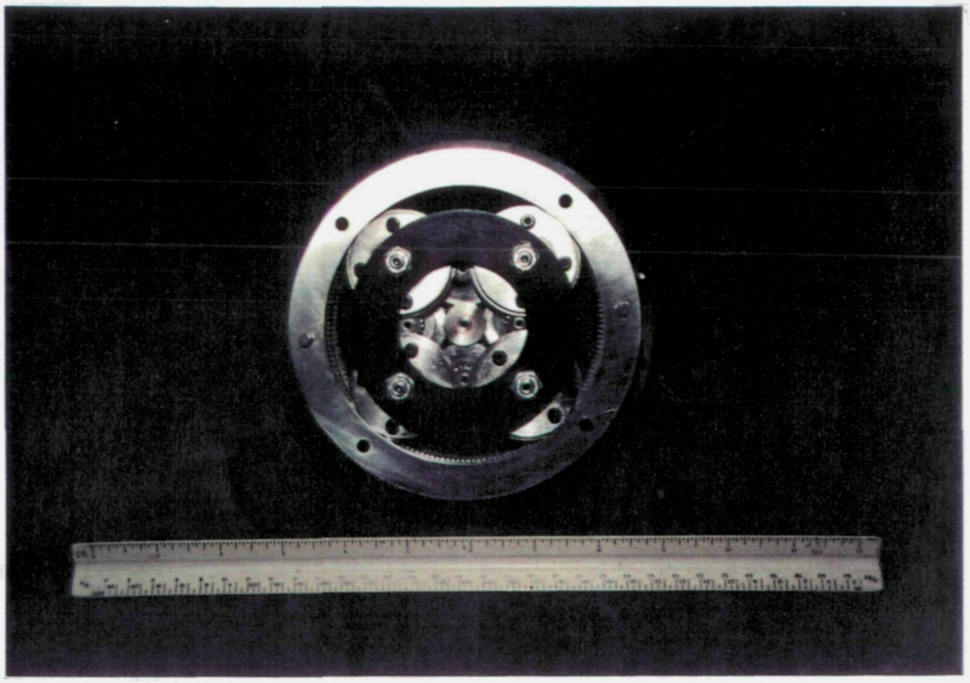
a) Schematic



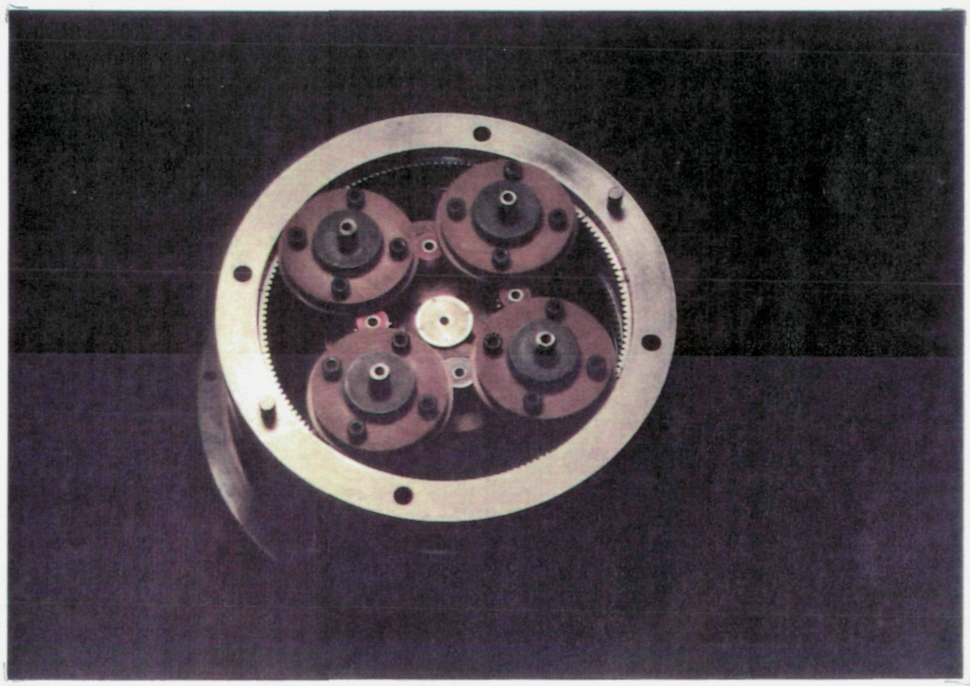
b) Photo showing test arrangement and data processing system

FIGURE 29 - Test setup for efficiency measurements of the Wrist-Roll drives.

ORIGINAL PAGE
COLOR PHOTOGRAPH



c) Steel Wrist-Roll drive, partially assembled



d) Plastic Wrist-Roll drive, partially assembled

FIGURE 29 - Test setup for efficiency measurements
of the Wrist-Roll drives.

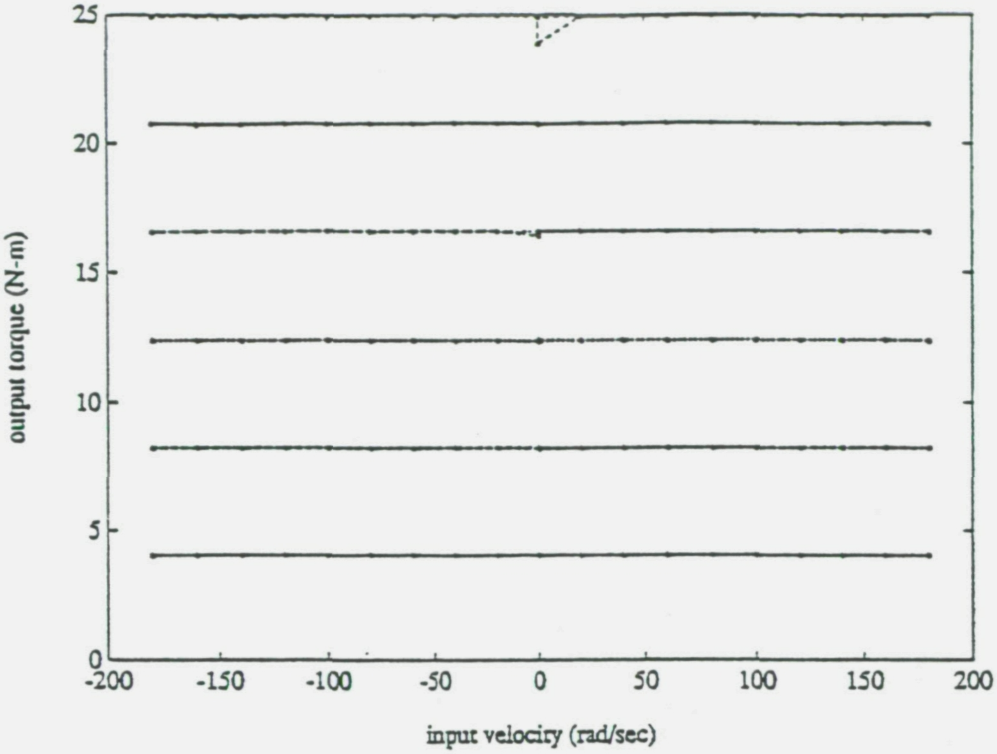


FIGURE 30 - Controlled input velocities and output torques (typ) for the Steel Wrist-Roll drive (lubricated).

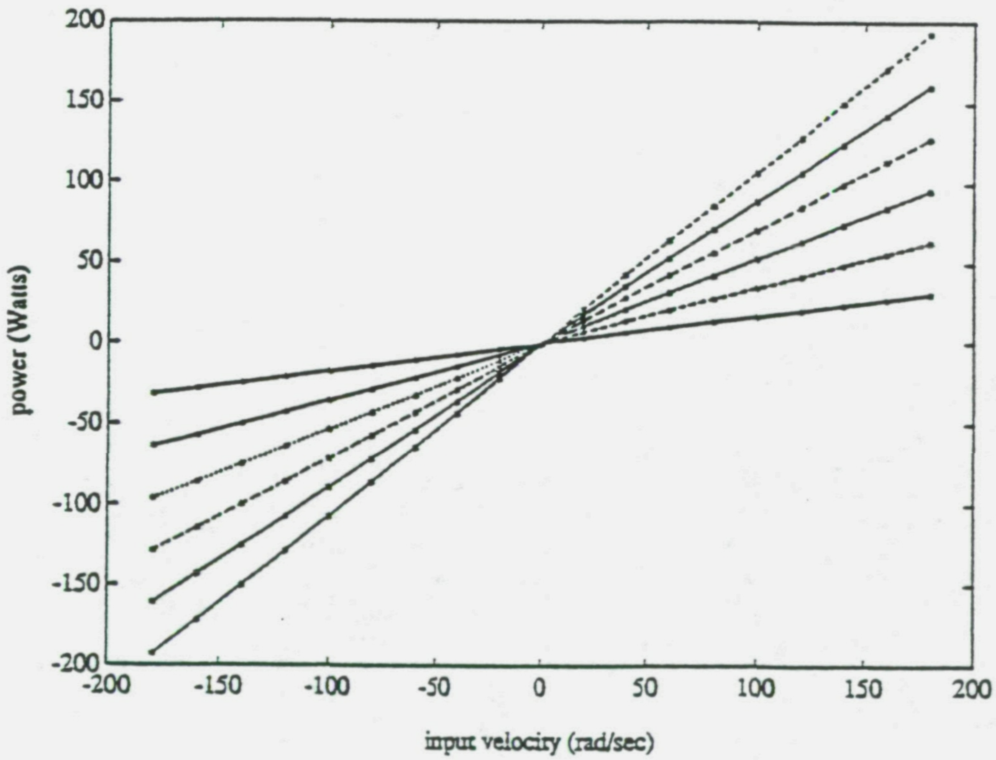
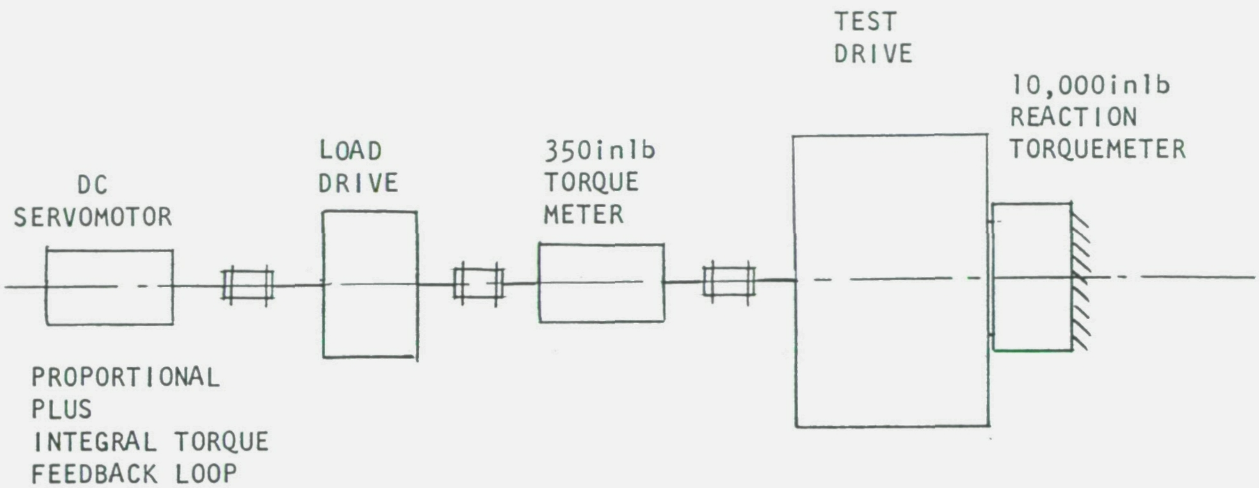
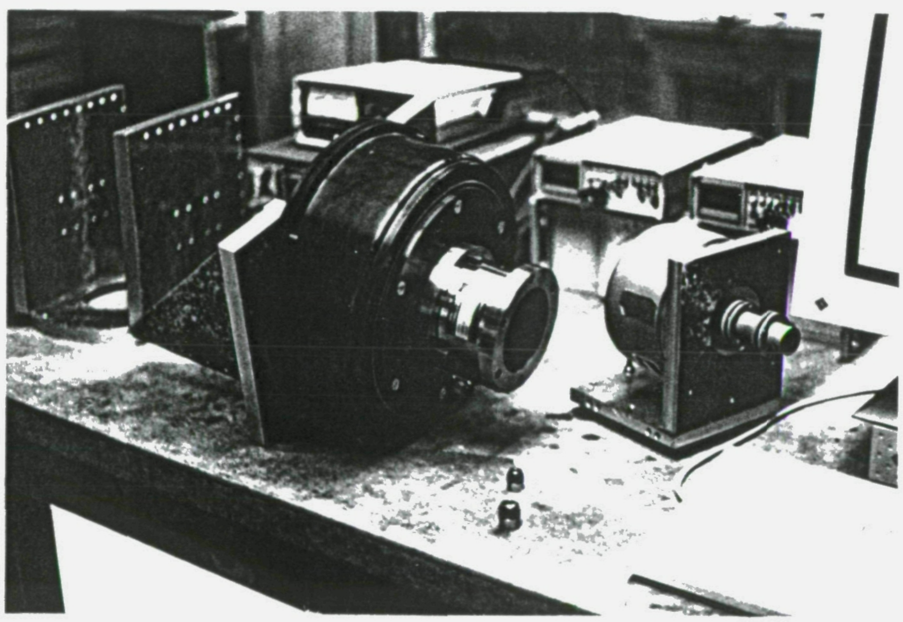


FIGURE 31 - Controlled output (low-speed) power (typ) for the Steel Wrist-Roll drive (lubricated).



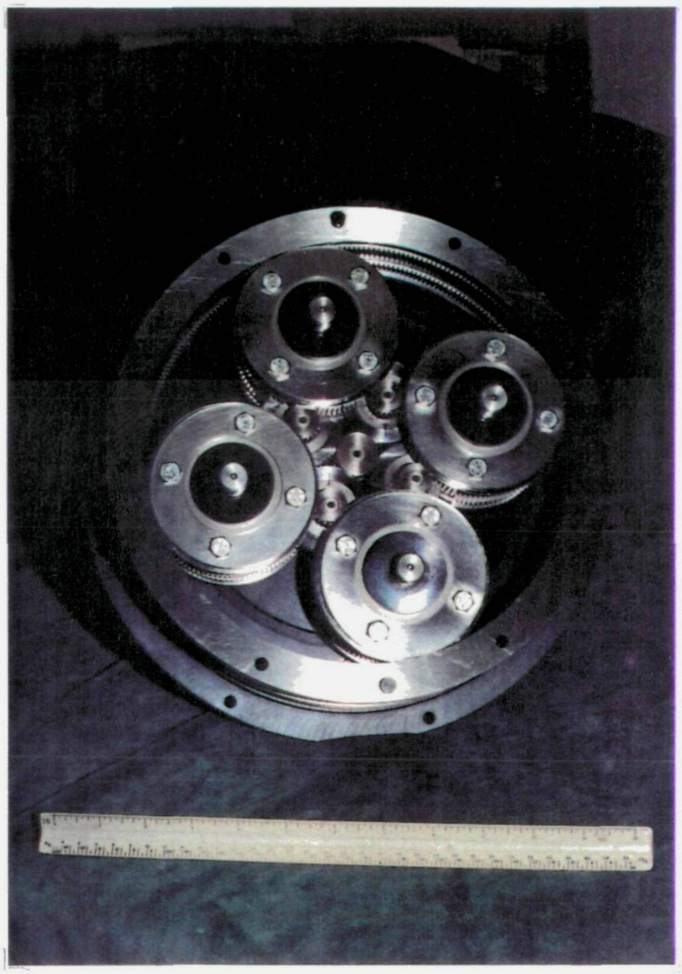
a) Schematic



b) Photo showing Hinge-Joint drive and reaction torquemeter

FIGURE 32 - Test setup for static measurements of the Hinge-Joint drive.

ORIGINAL PAGE
COLOR PHOTOGRAPH



c) Hinge-Joint drive, partially assembled

FIGURE 32 - Test setup for static measurements of the Hinge Joint drive.

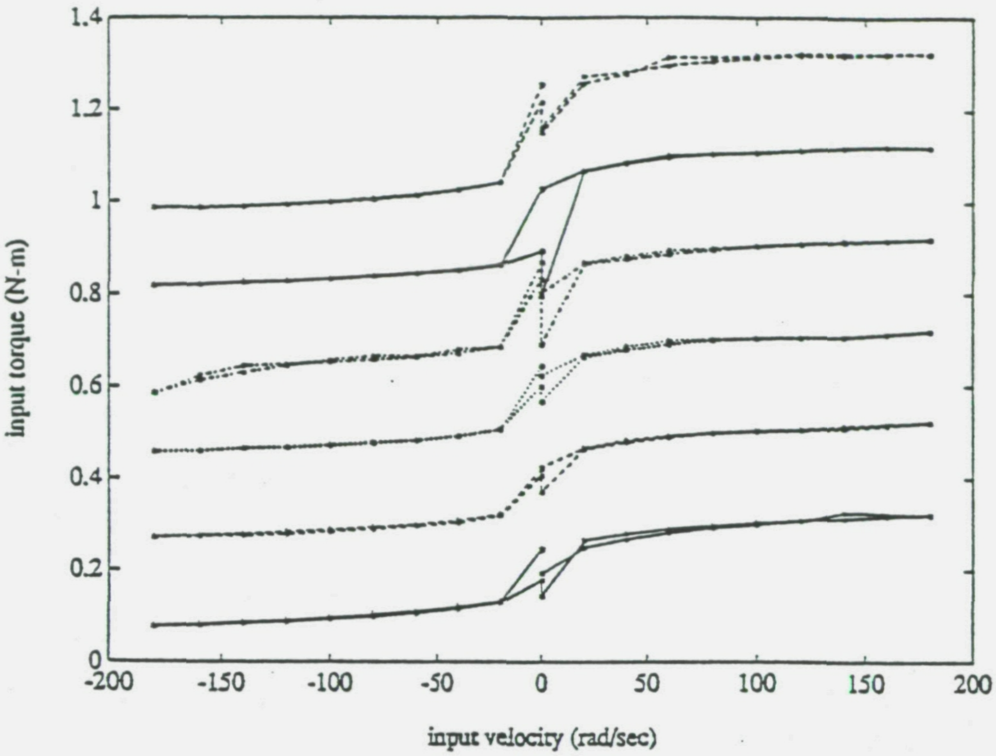


FIGURE 33 - Steel Wrist-Roll drive input torques (dry).

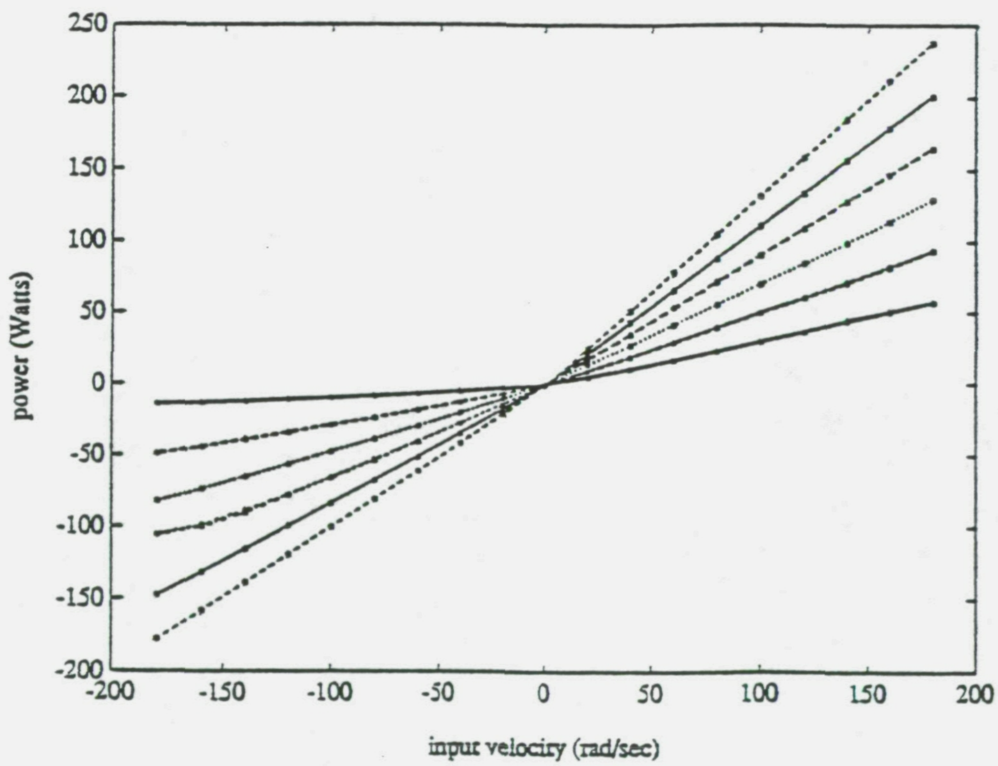


FIGURE 34 - Steel Wrist-Roll drive input power (dry).

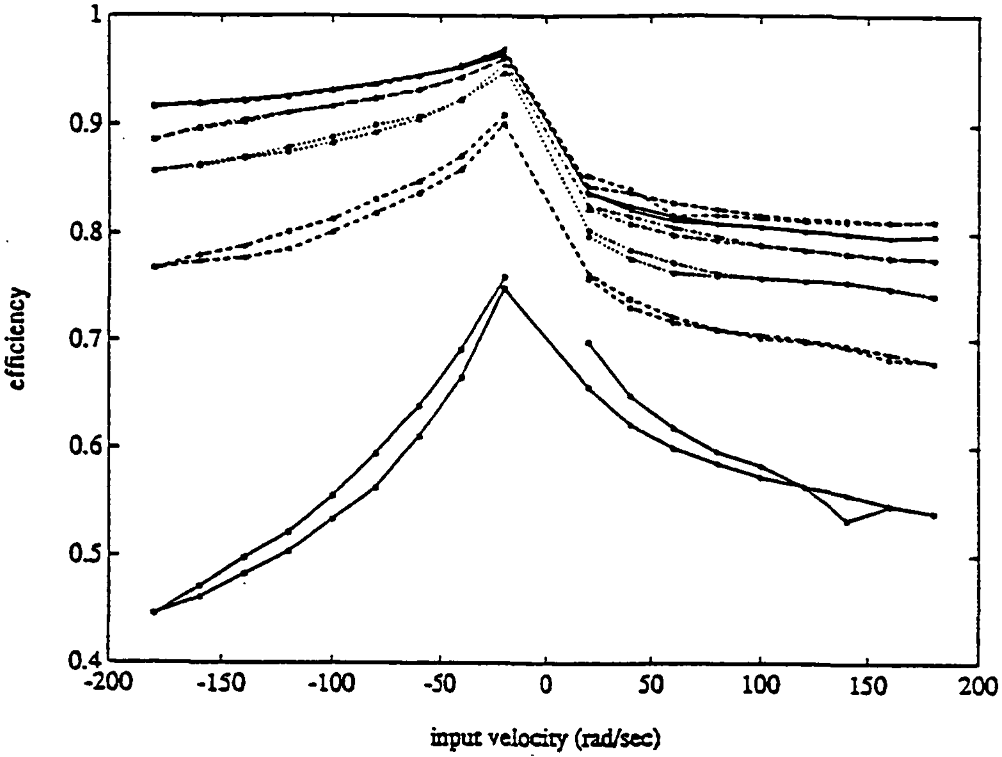


FIGURE 35 - Steel Wrist-Roll drive efficiency (dry).

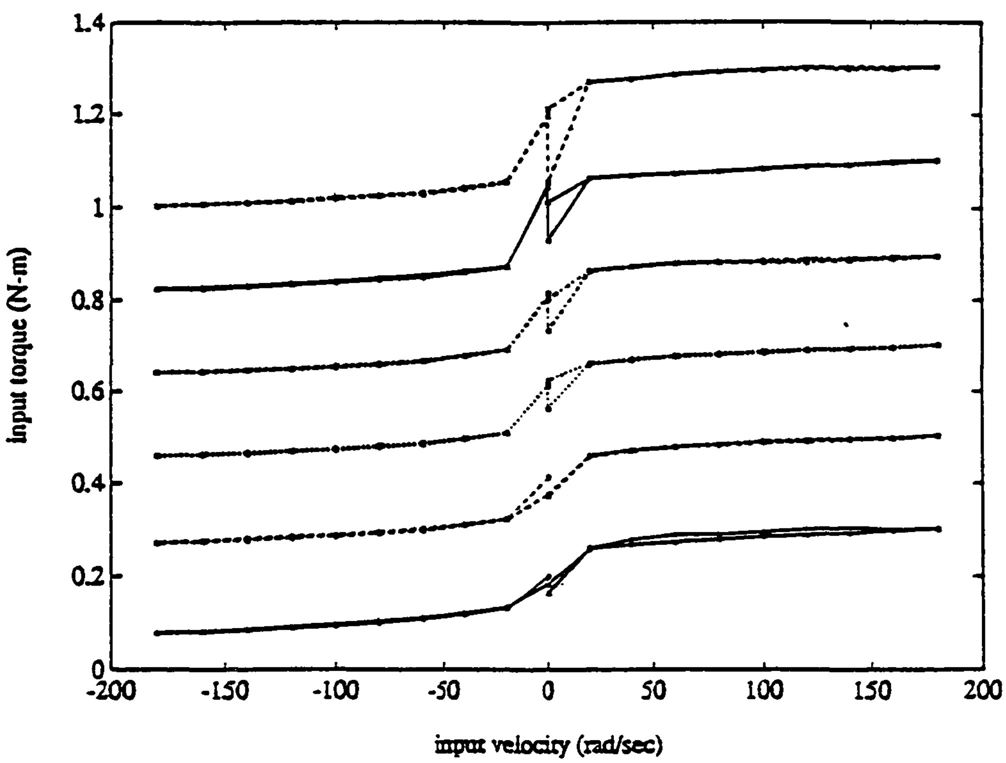


FIGURE 36 - Steel Wrist-Roll drive input torques (lubricated).

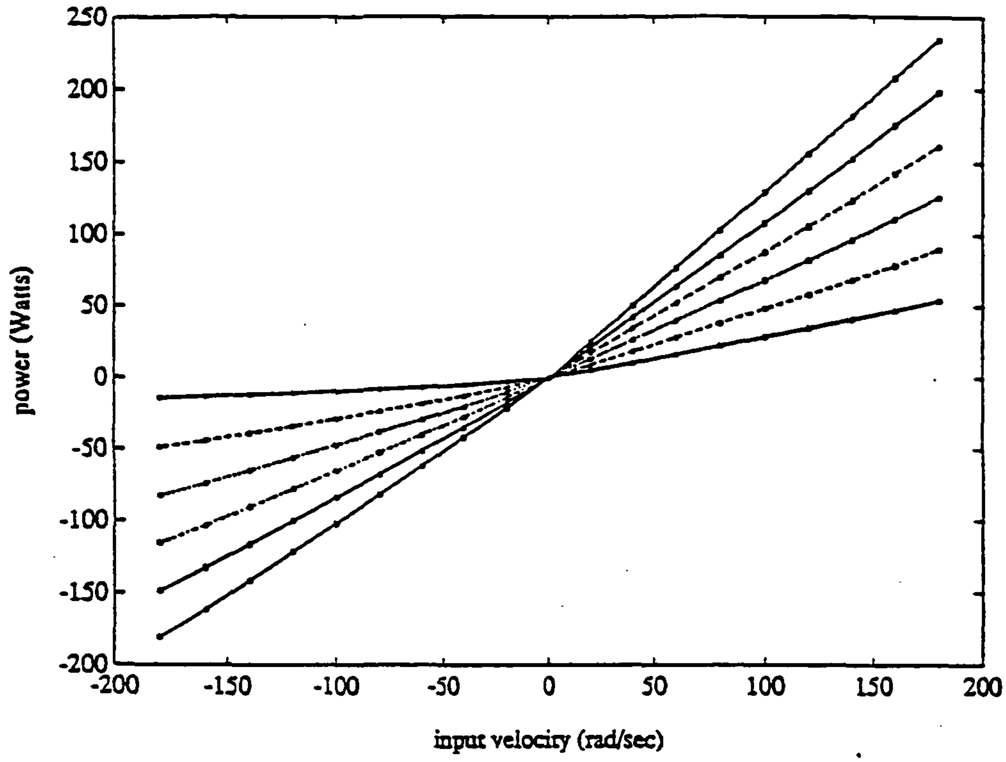


FIGURE 37 - Steel Wrist-Roll drive input power (lubricated).

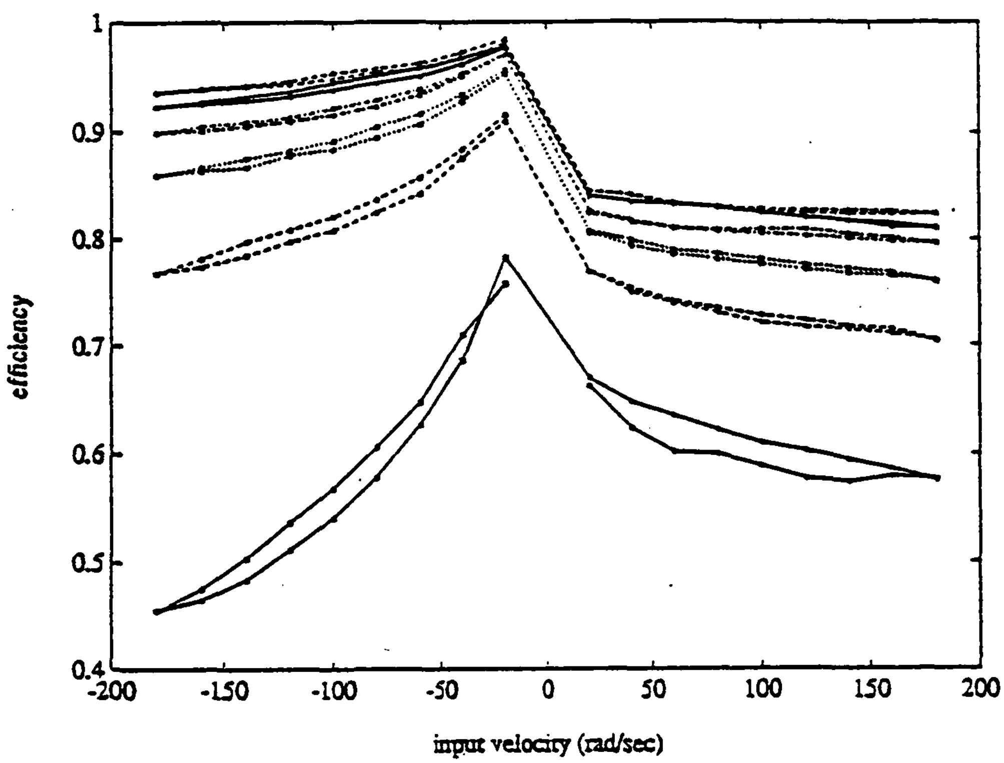


FIGURE 38 - Steel Wrist-Roll drive efficiency (lubricated).

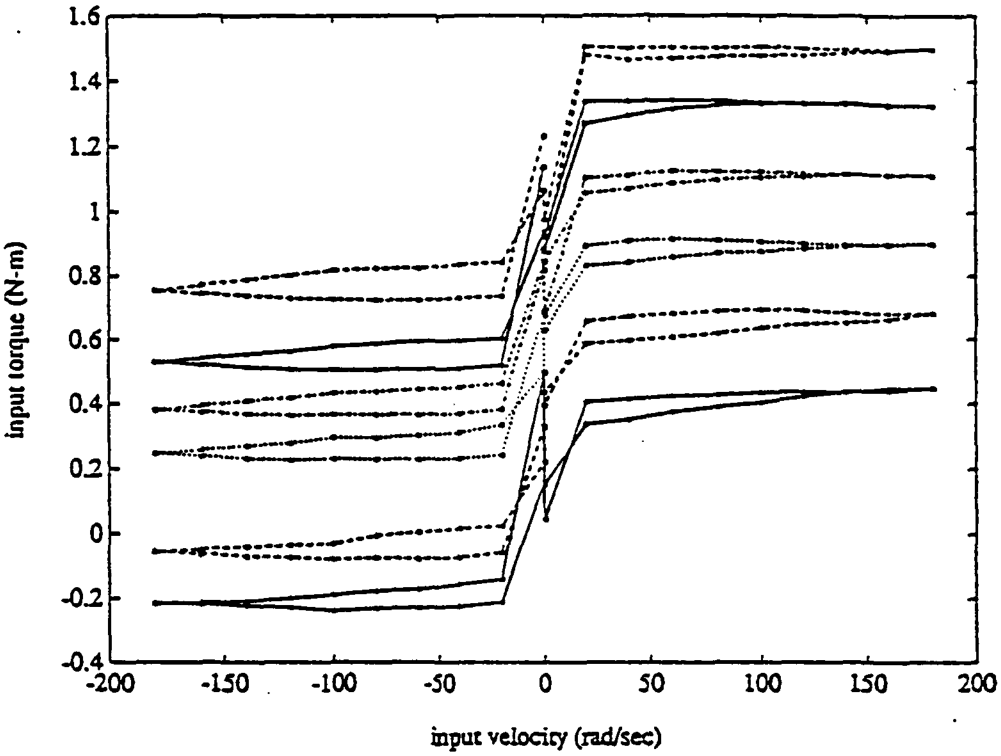


FIGURE 39 - Plastic Wrist-Roll drive input torques.

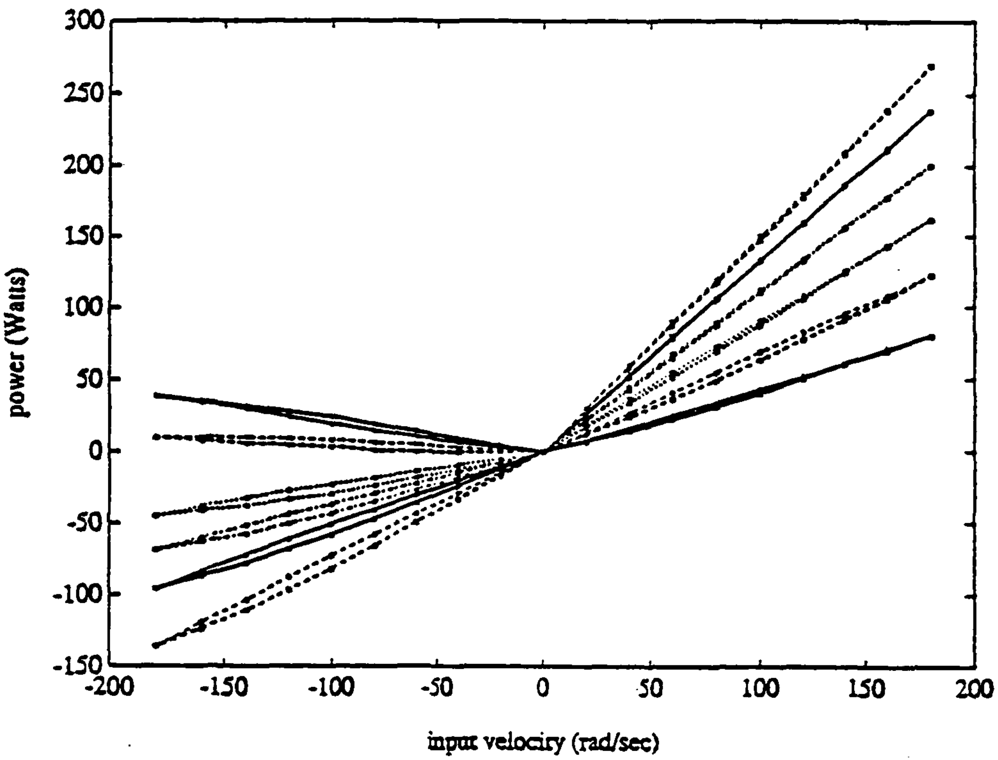


FIGURE 40 - Plastic Wrist-Roll drive input power.

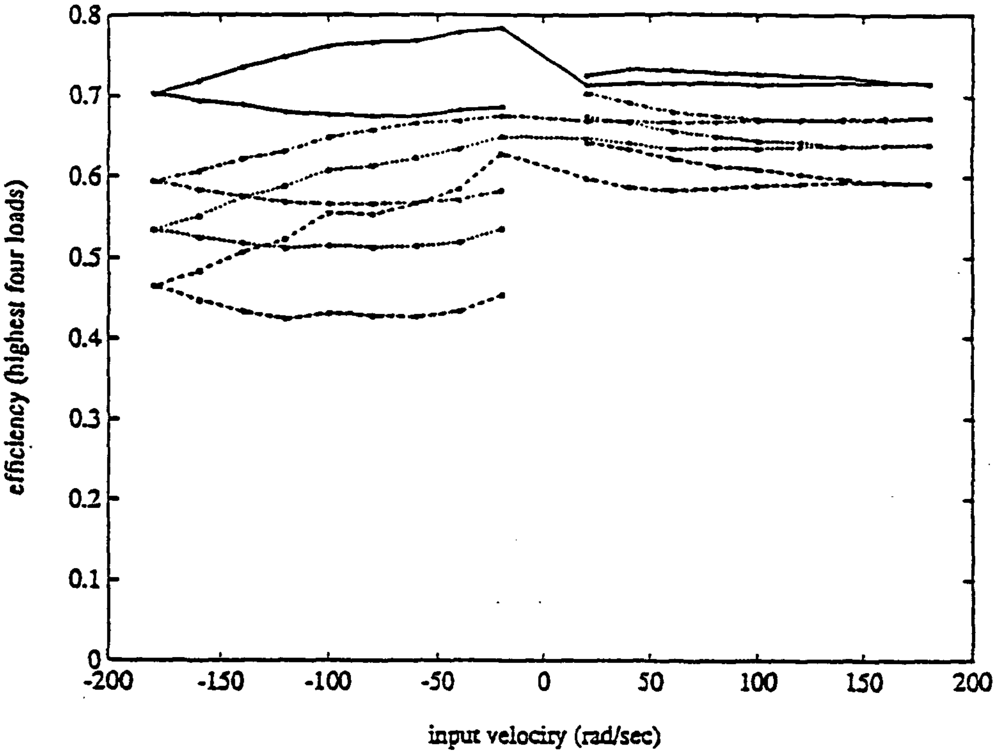


FIGURE 41 - Plastic Wrist-Roll drive efficiency.

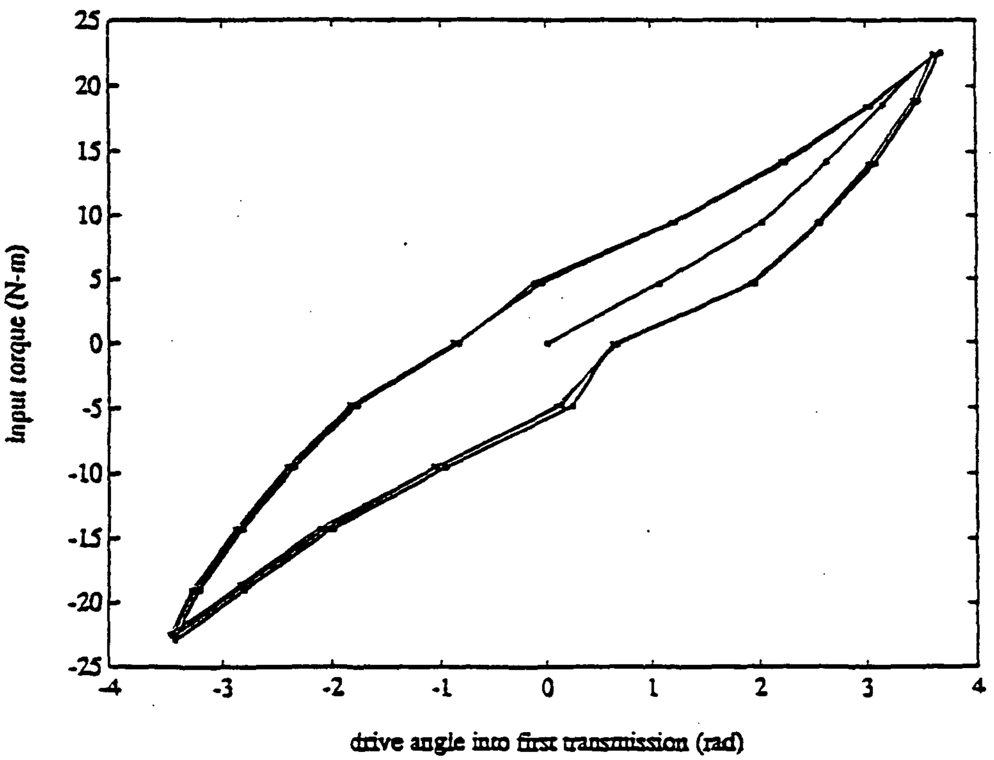


FIGURE 42 - Hinge-Joint drive quasi-static input torques.

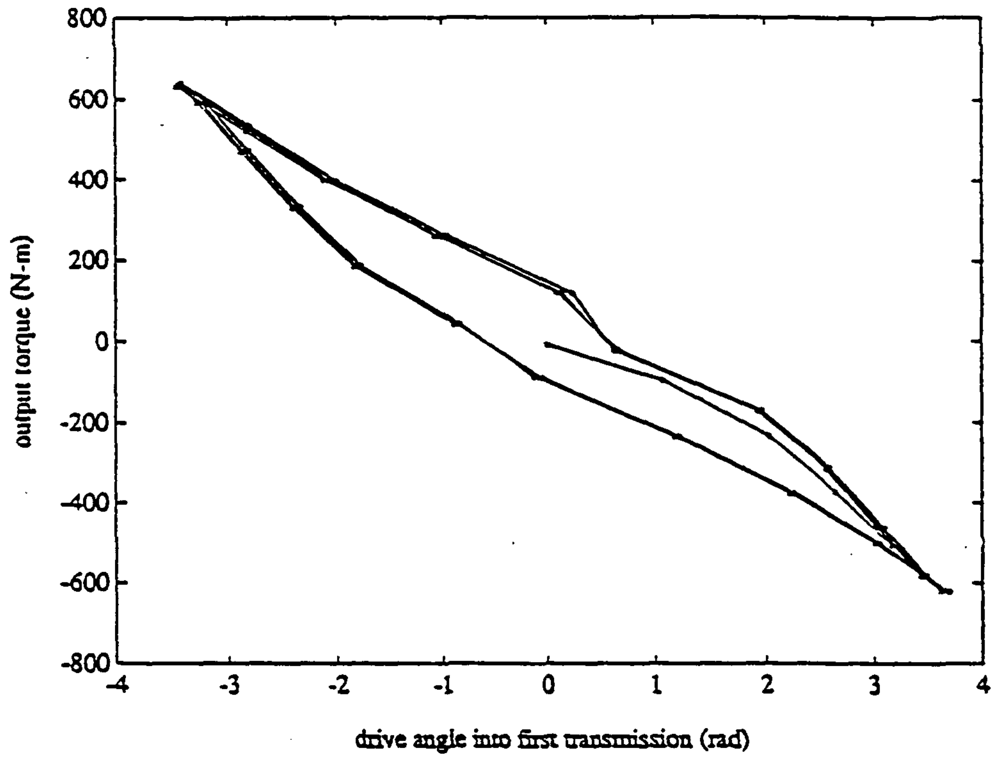


FIGURE 43 - Hinge-Joint drive quasi-static output torques.

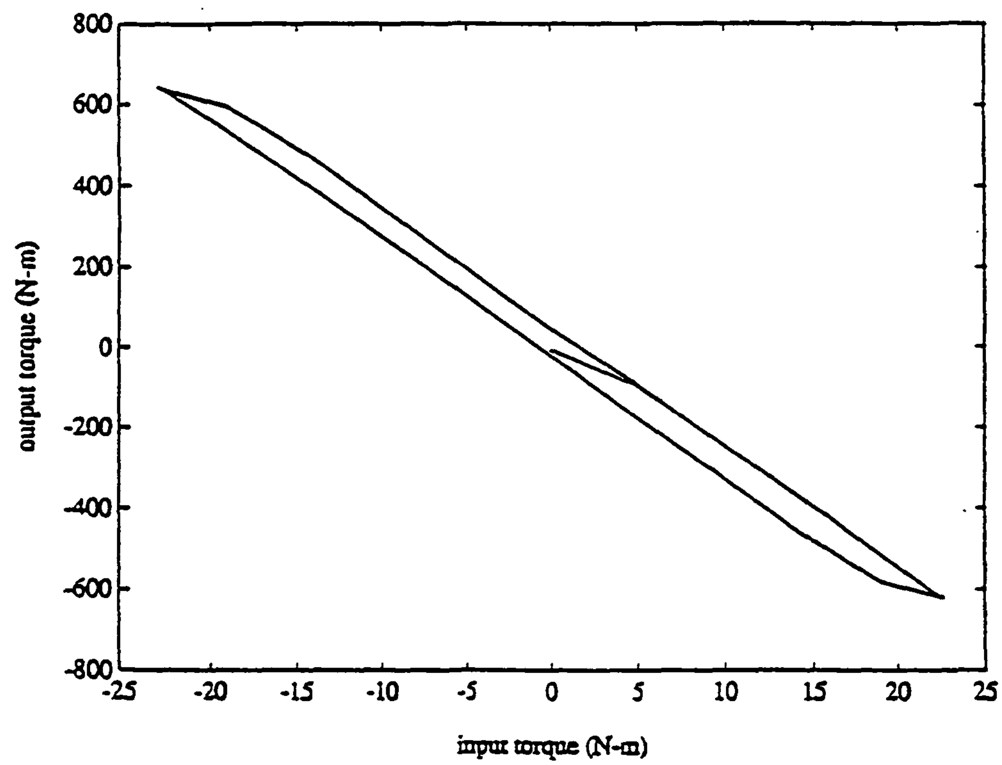


FIGURE 44 - Hinge-Joint drive torque hysteresis.

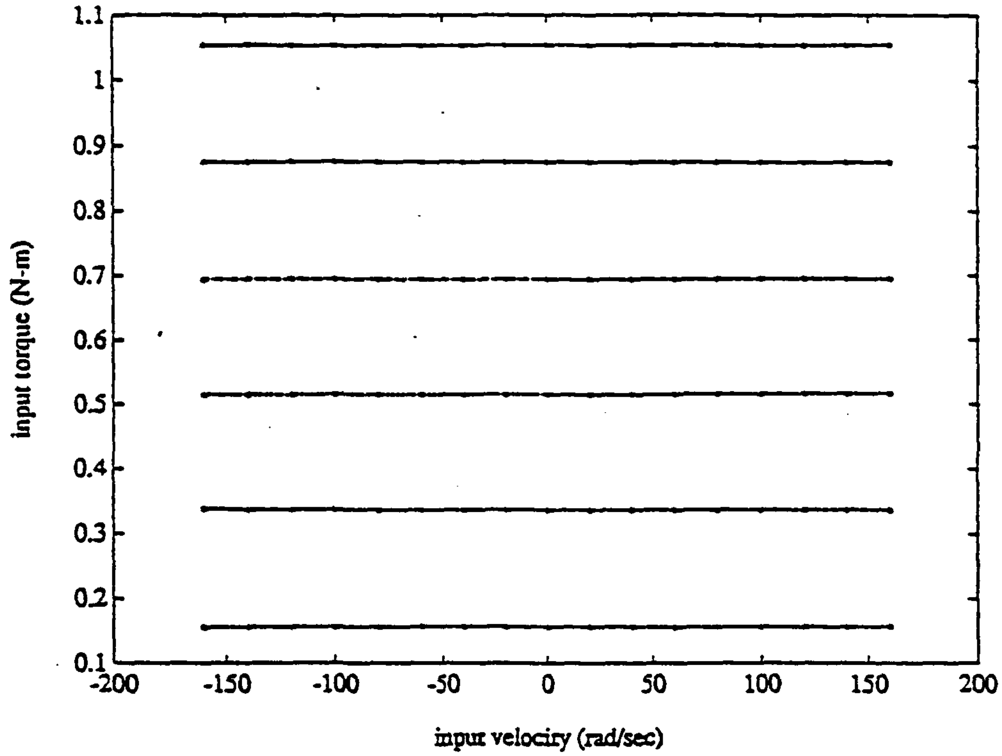


FIGURE 45 - Steel Wrist-Roll drive input torques (lubricated). (Reverse Measurement: high-speed torque source, low-speed velocity source).

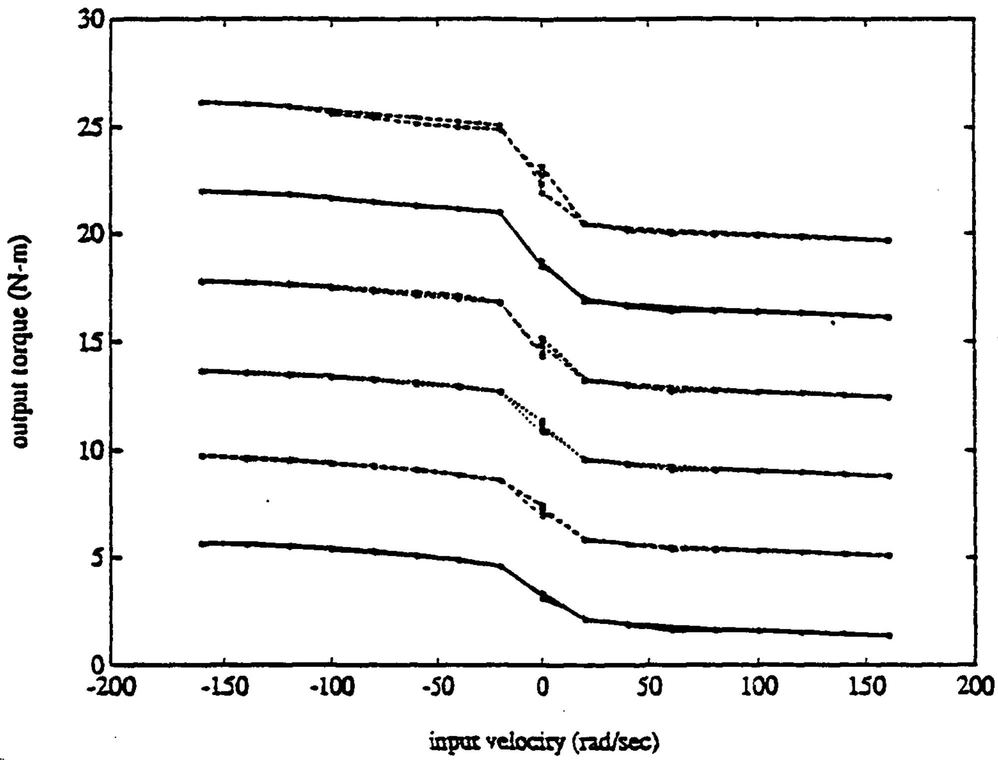


FIGURE 46 - Steel Wrist-Roll drive output torques (lubricated). (Reverse Measurement: high-speed torque source, low-speed velocity source).

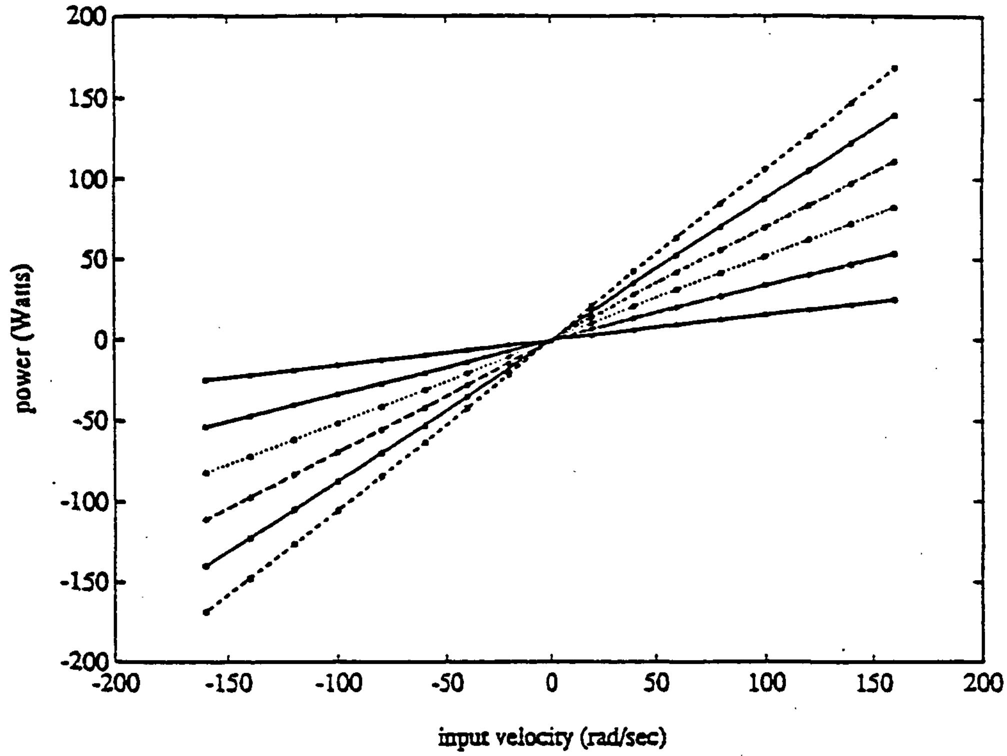


FIGURE 47 - Wrist-Roll drive low-speed power (lubricated). (Reverse Measurement: high-speed torque source, low-speed velocity source).

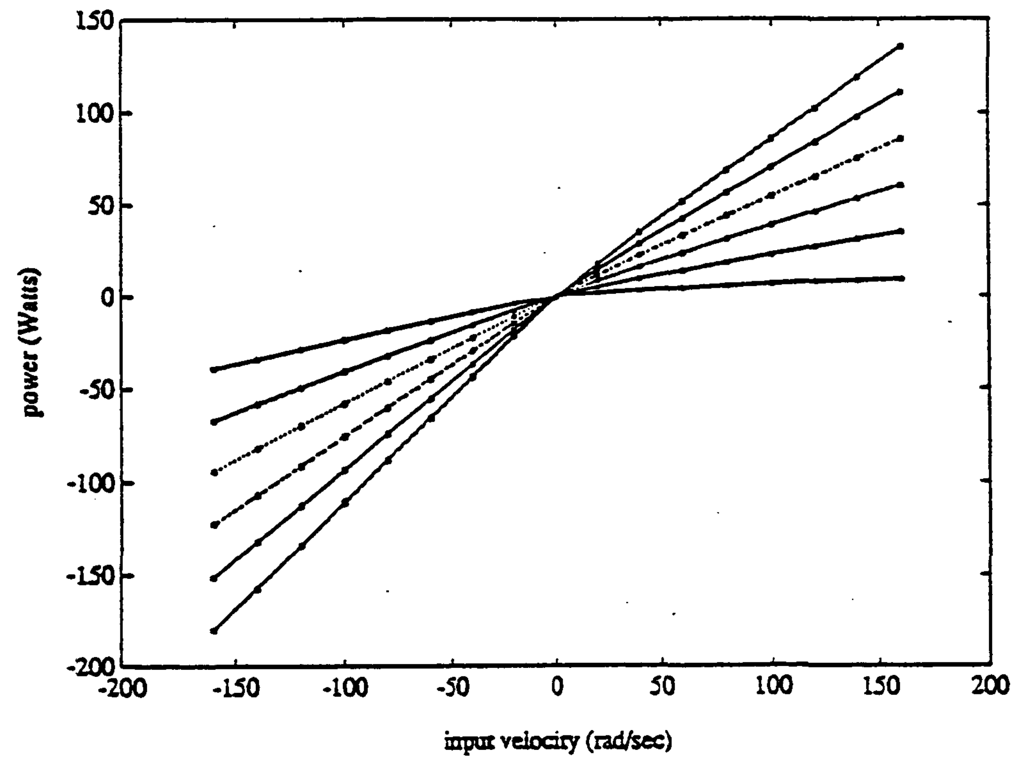


FIGURE 48 - Wrist-Roll drive high-speed power (lubricated). (Reverse Measurement: high-speed torque source, low-speed velocity source).

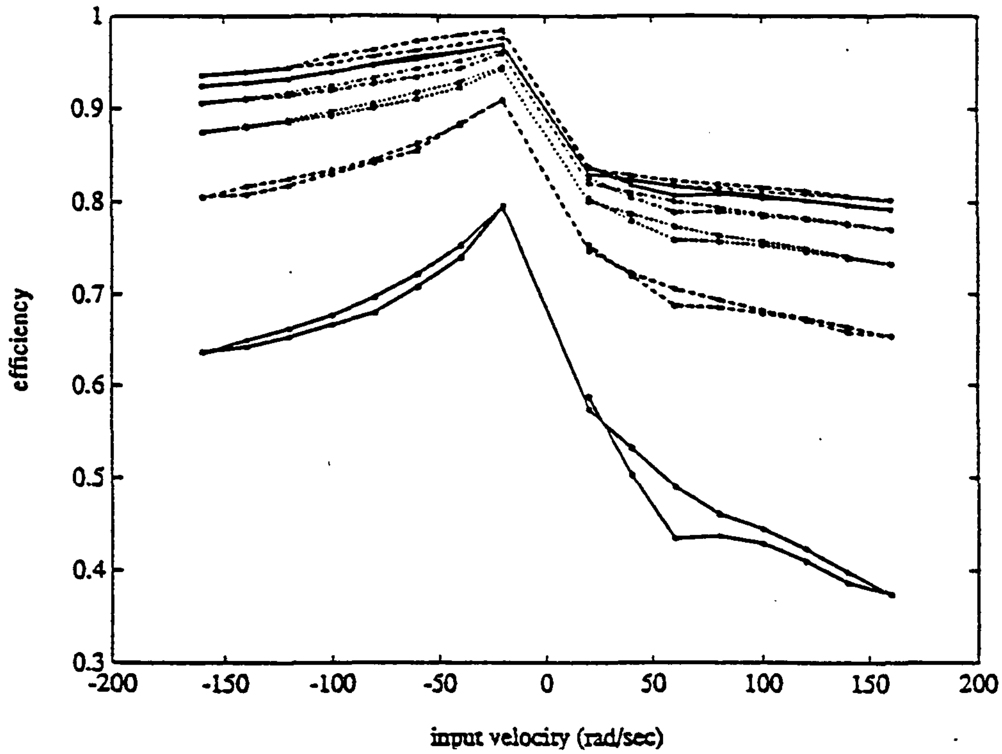
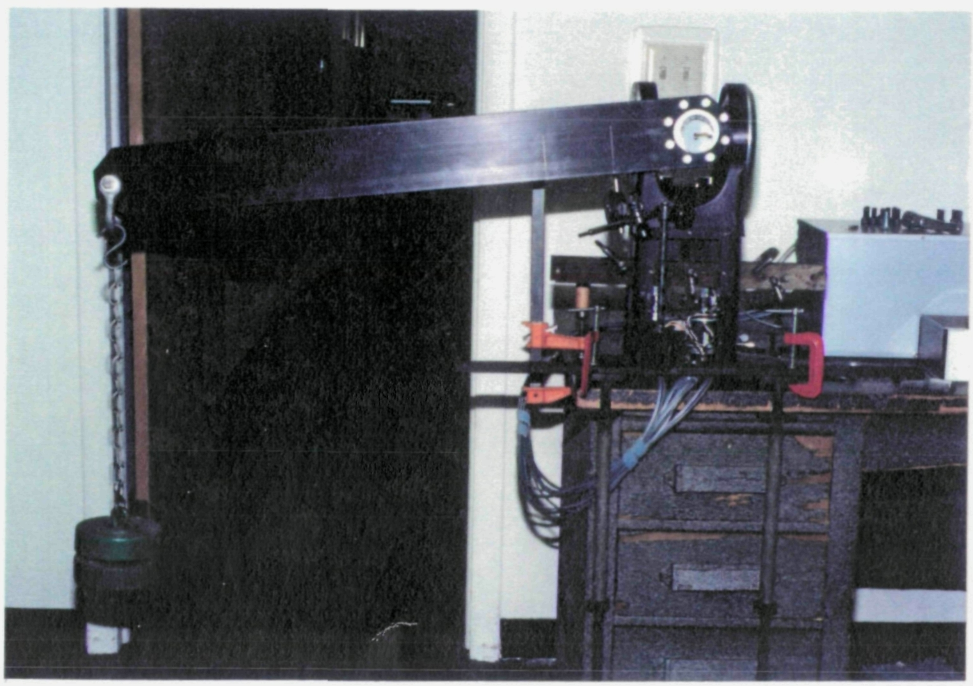


FIGURE 49 - Steel Wrist-Roll drive efficiency (lubricated). (Reverse Measurement: high-speed torque source, low-speed velocity source).

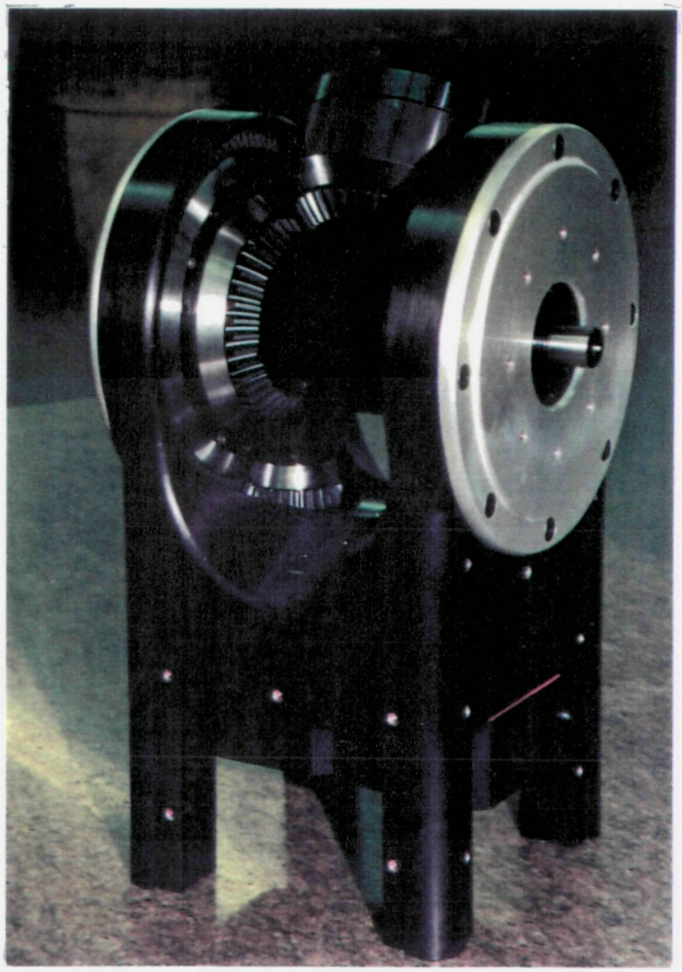
a) Pitch axis



b) Yaw axis

FIGURE 50 - Test setup for Pitch-Yaw Joint drive stiffness measurements.

ORIGINAL PAGE
COLOR PHOTOGRAPH



c) Pitch-Yaw Joint drive, partially assembled

FIGURE 50 - Test setup for Pitch-Yaw Joint drive stiffness measurements.

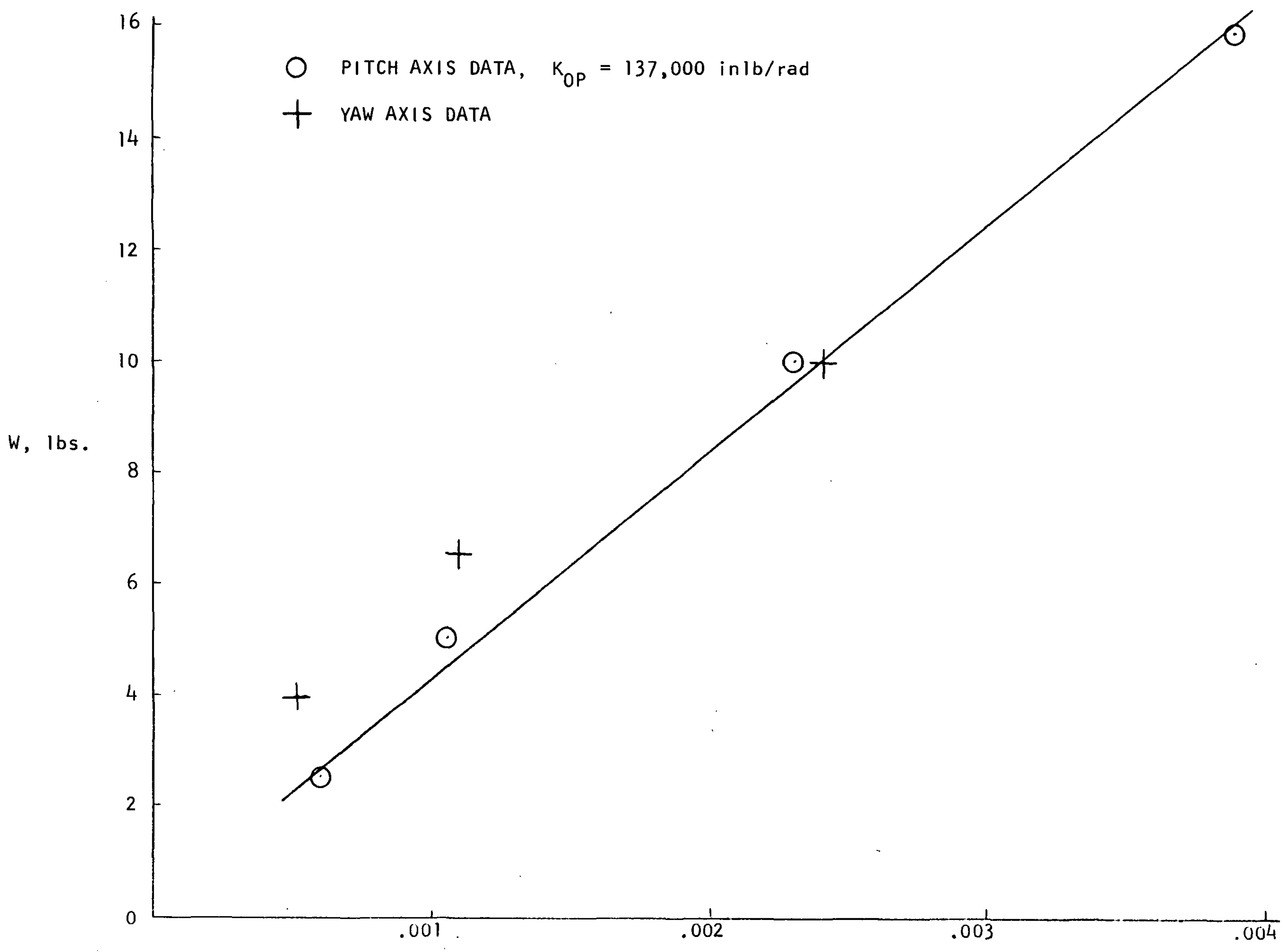


FIGURE 51 - Pitch/Yaw joint stiffness data.

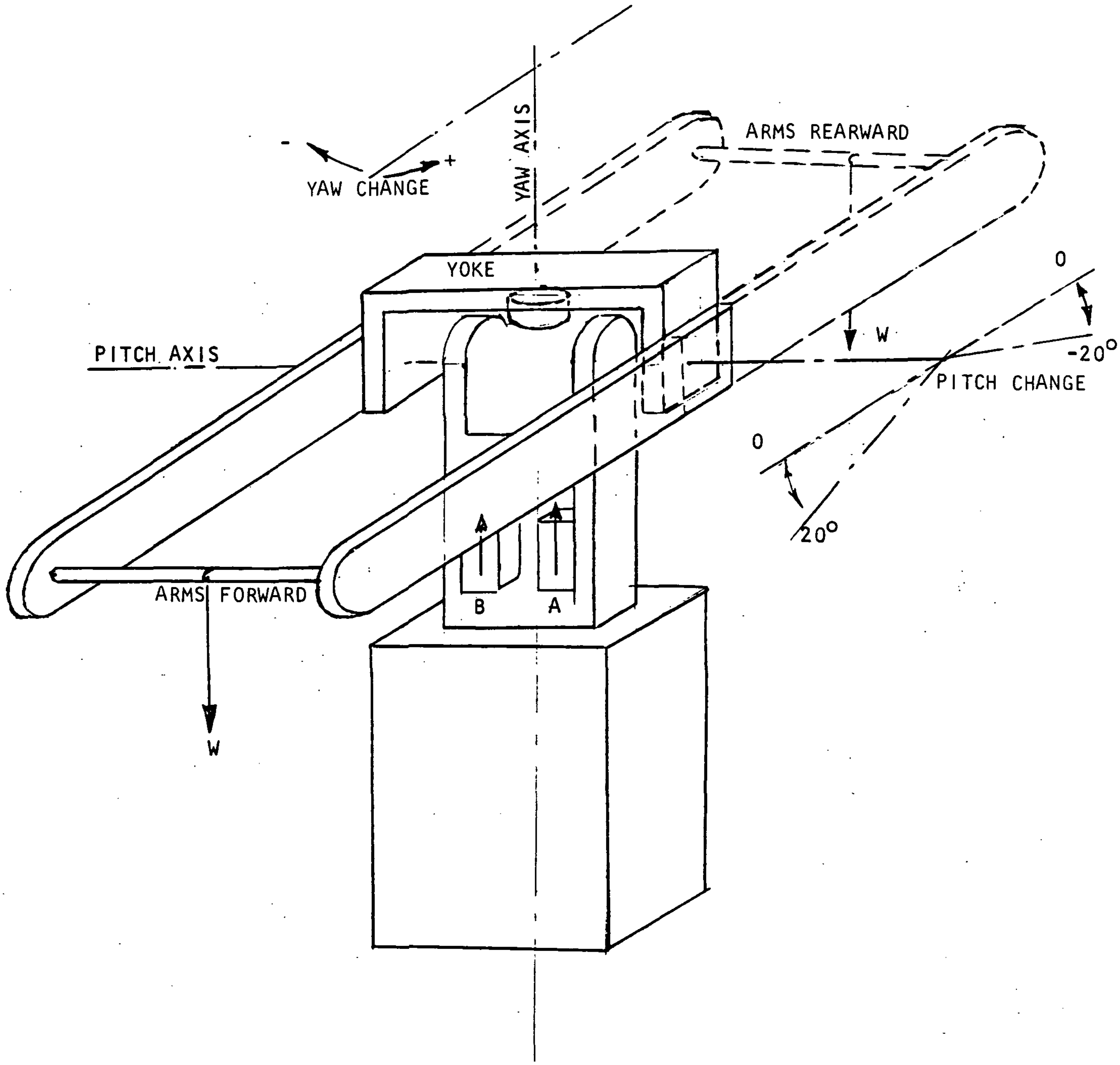


FIGURE 52 - Schematic of Pitch/Yaw Joint loading system.

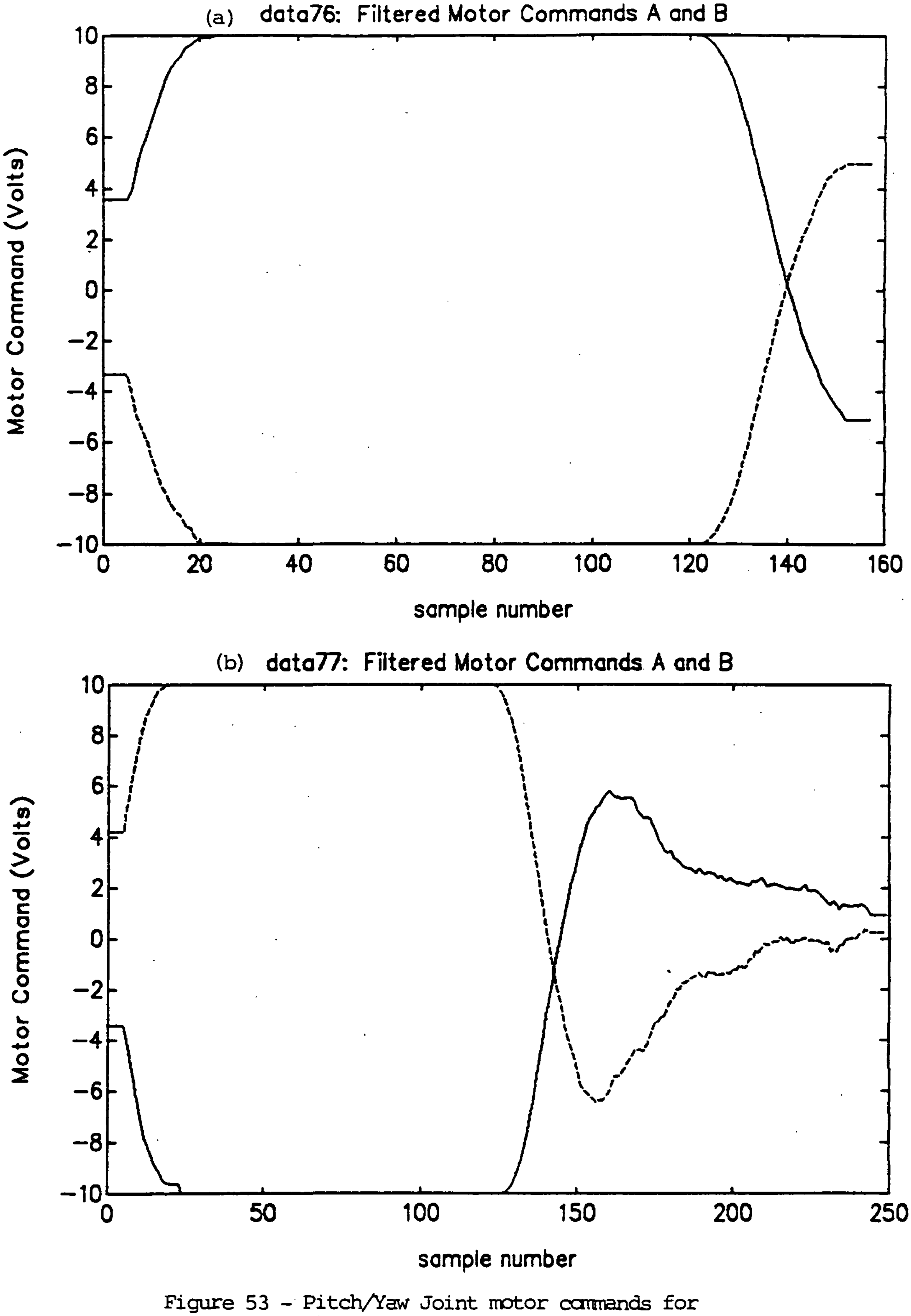


Figure 53 - Pitch/Yaw Joint motor commands for pitch lift (a) and pitch drop (b). No load.

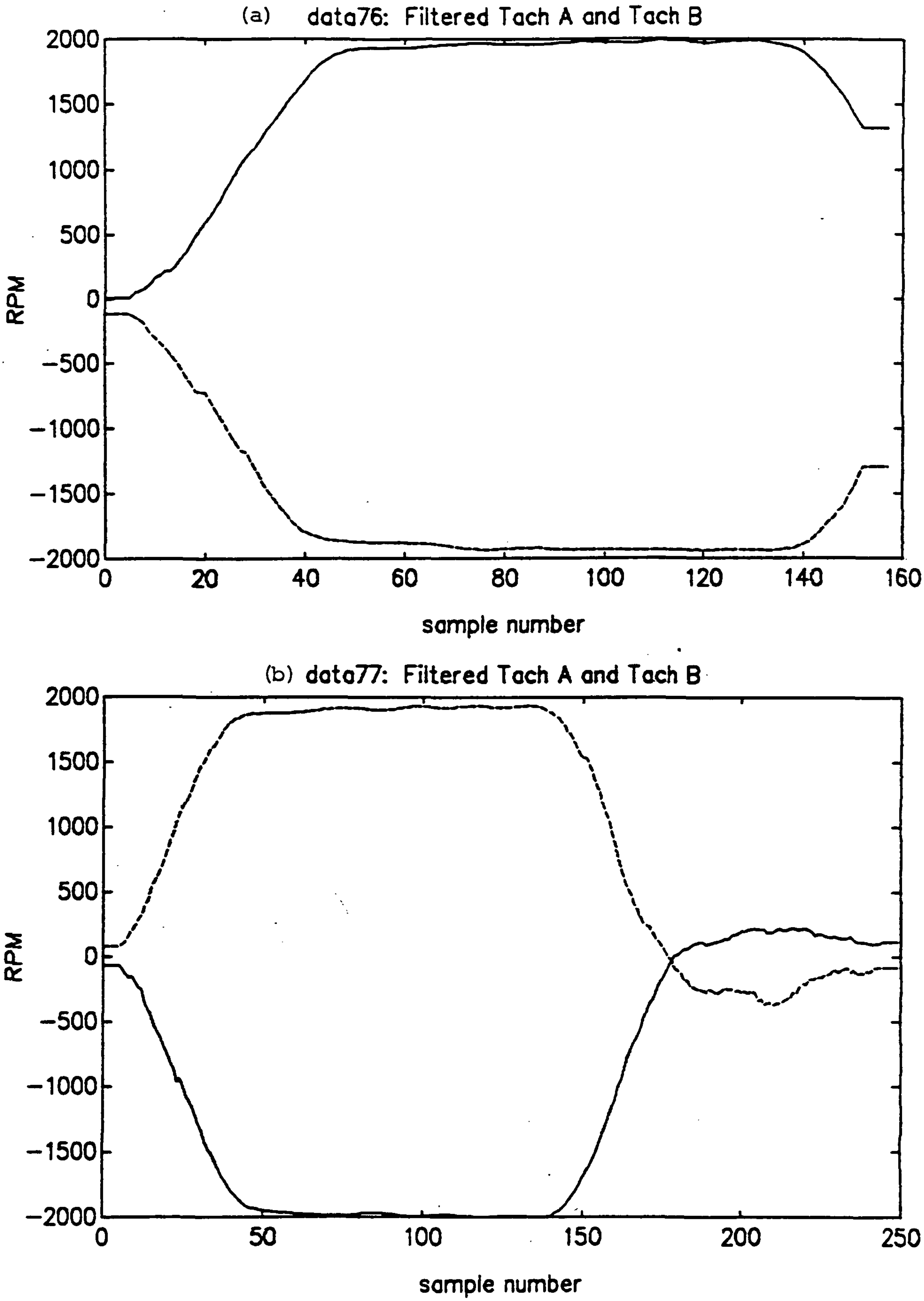


Figure 54 - Pitch/Yaw Joint drive motor speeds for pitch lift (a) and pitch drop (b). No load.

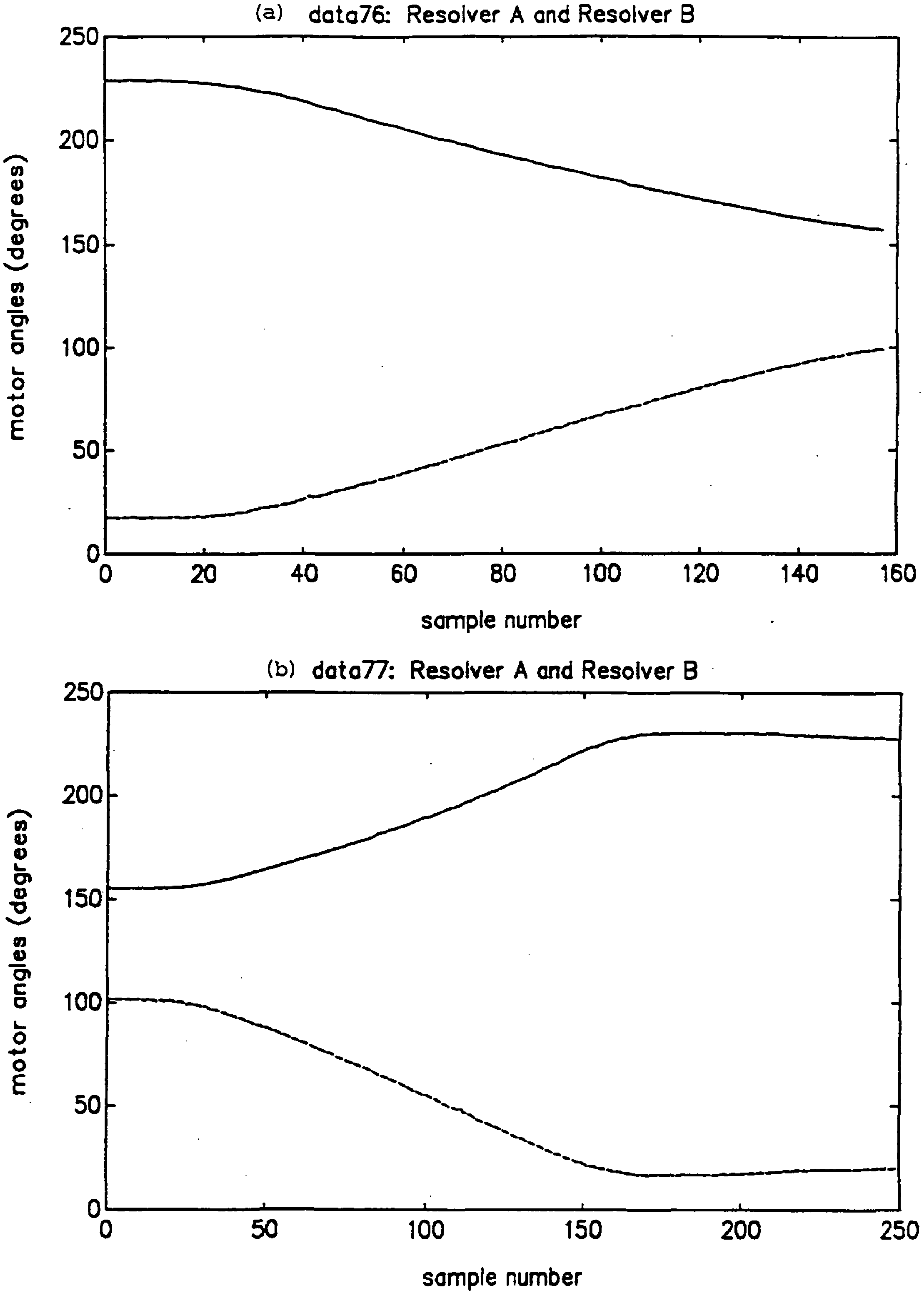


Figure 55 - Pitch/Yaw Joint input resolver positions for a pitch lift (a) and pitch drop (b). No load.

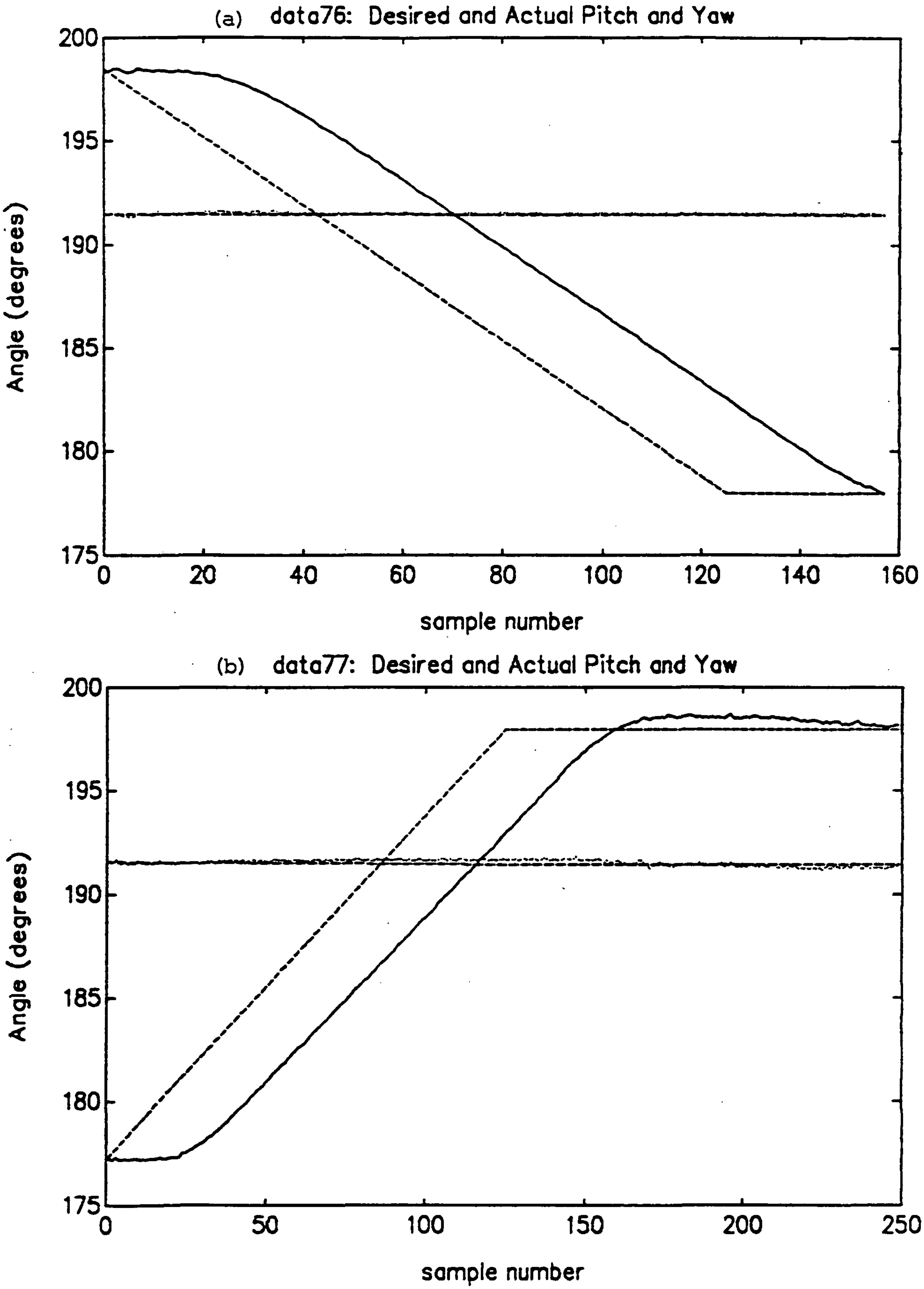
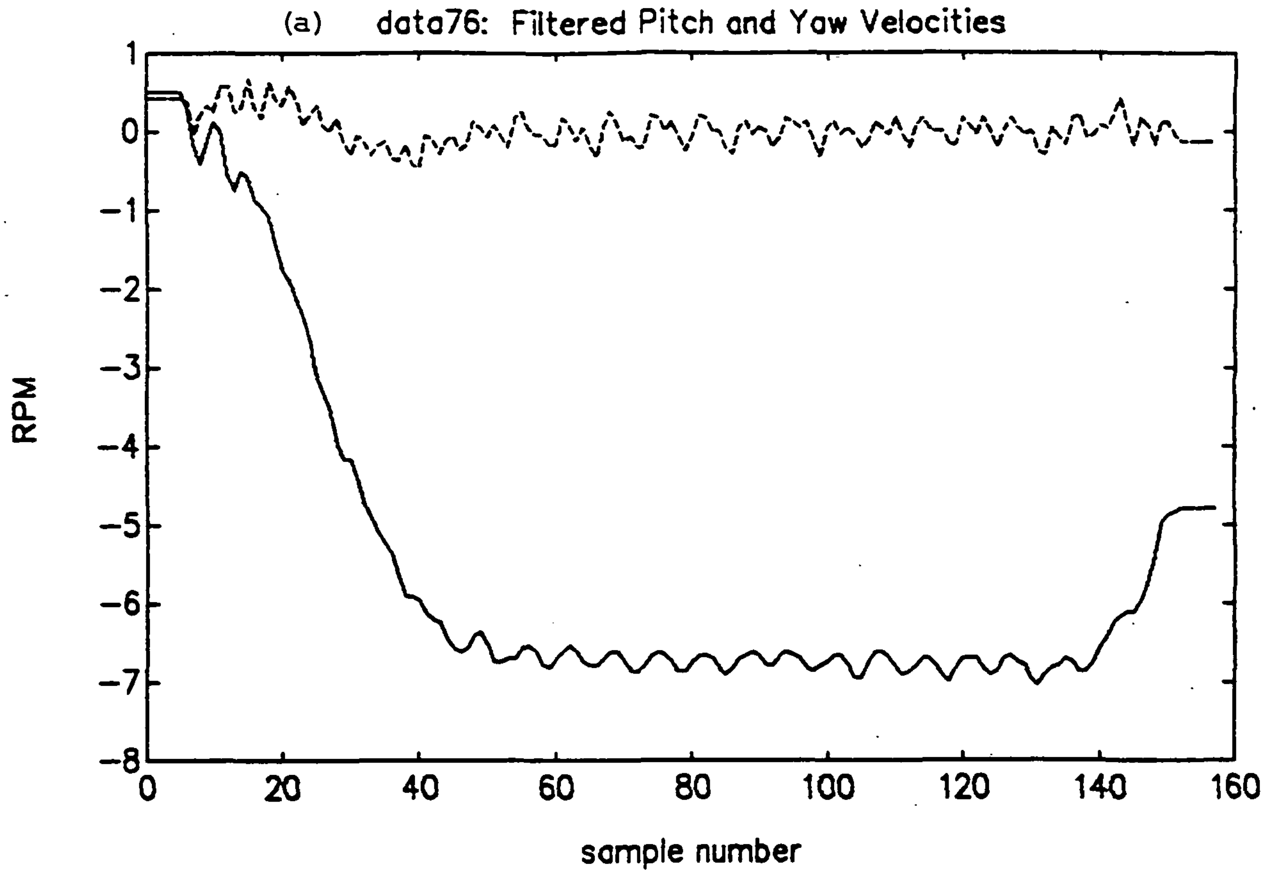


Figure 56 - Pitch/Yaw Joint desired and actual pitch and yaw positions for pitch lift (a) and pitch drop (b). No load.



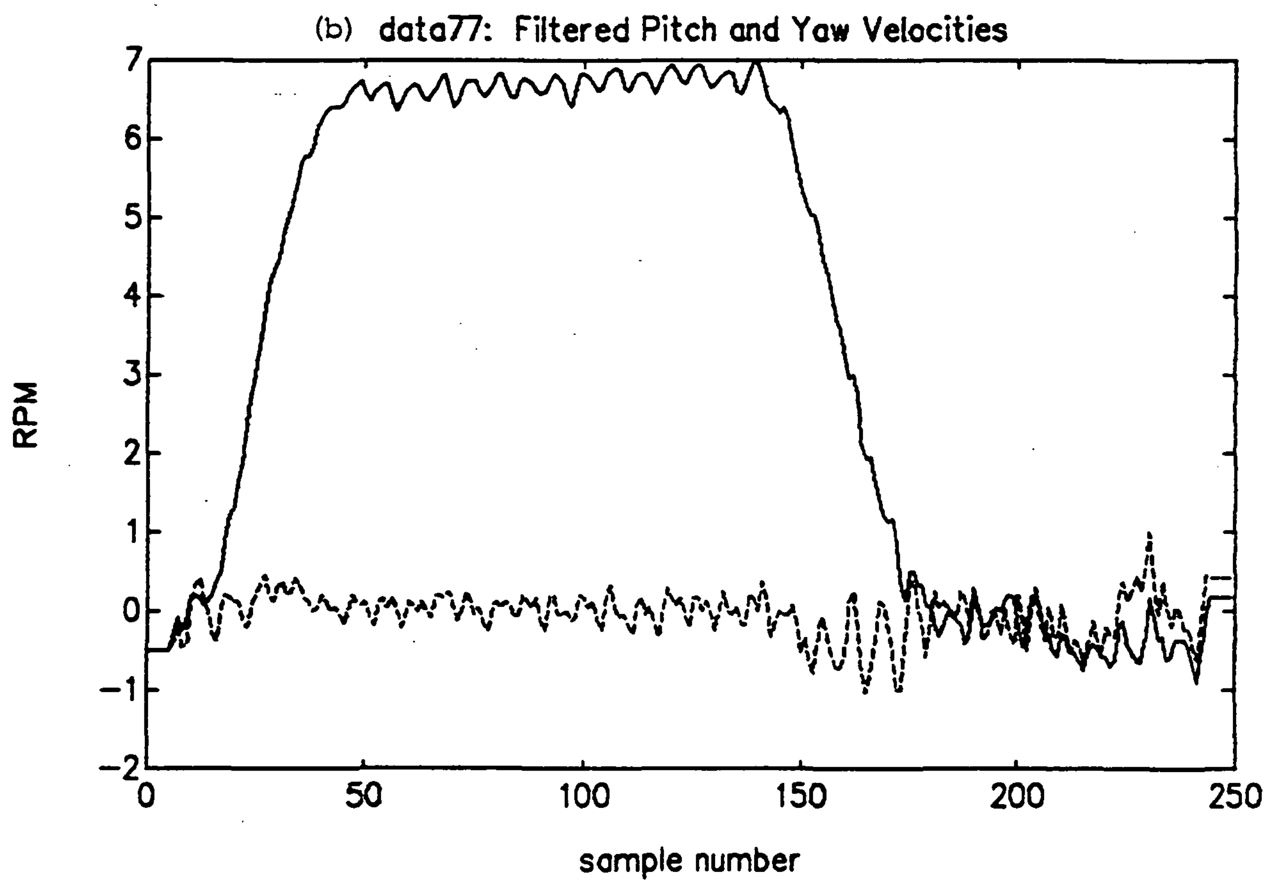
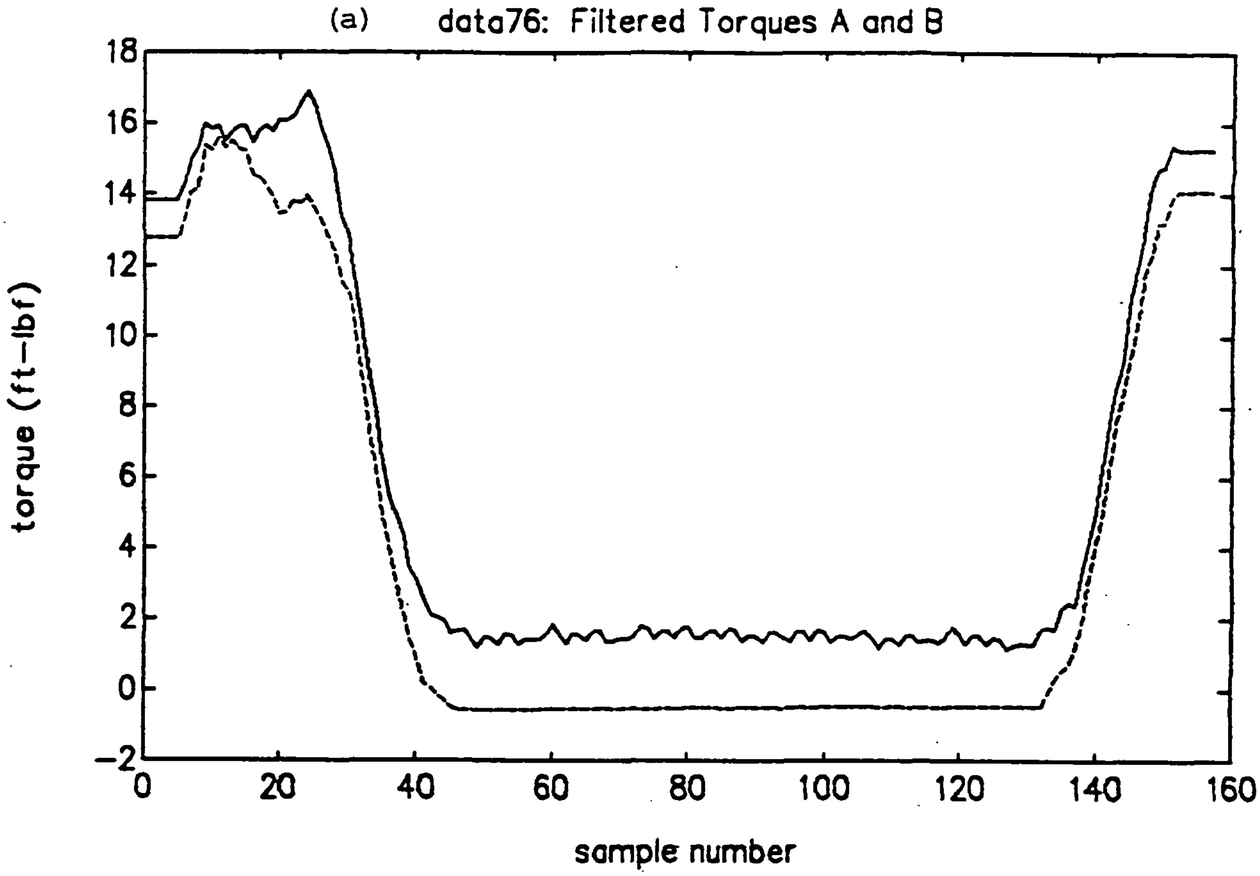


Figure 57 - Pitch/Yaw Joint pitch and yaw velocities for pitch lift (a) and pitch drop (b). No load.



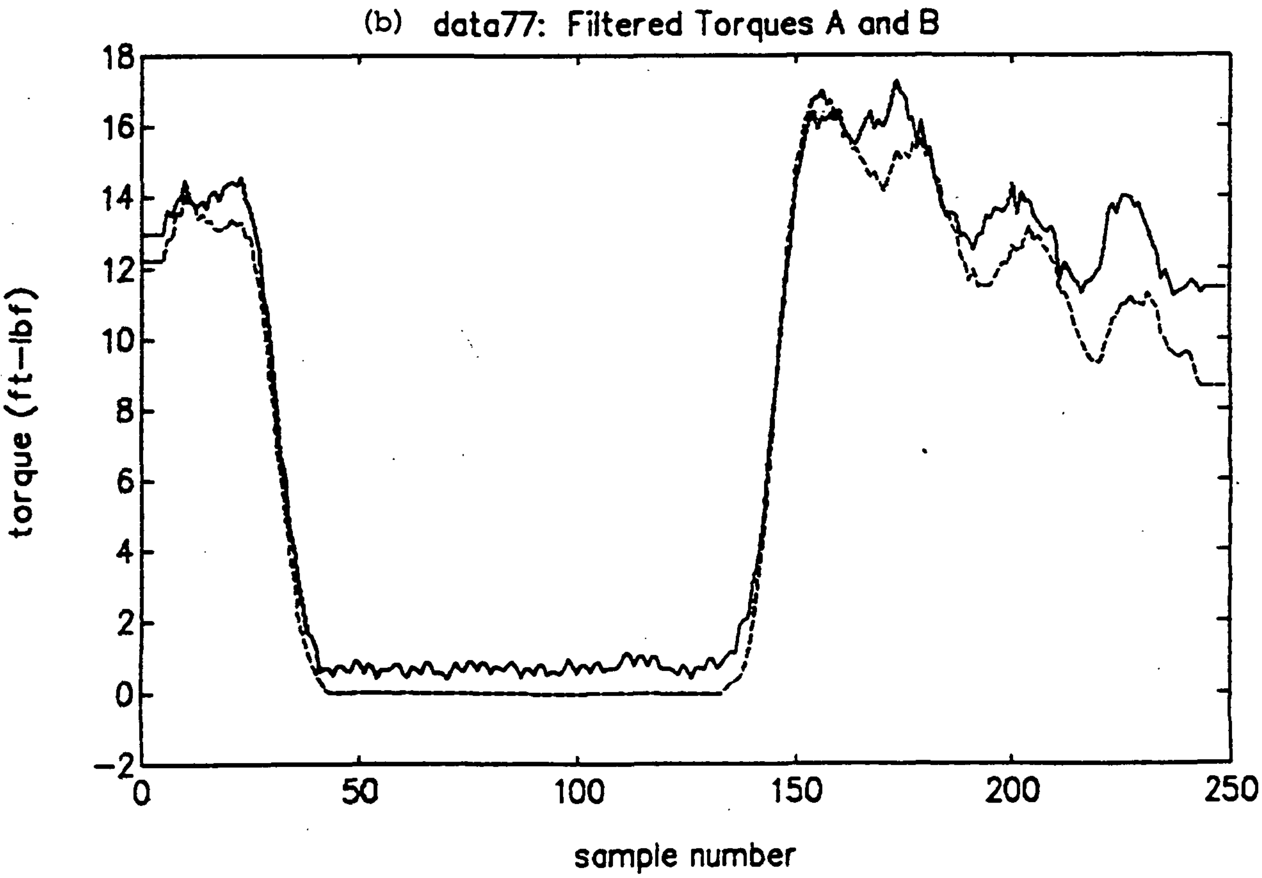


Figure 58 - Pitch/Yaw Joint input torques for pitch lift (a) and pitch drop (b). No load.

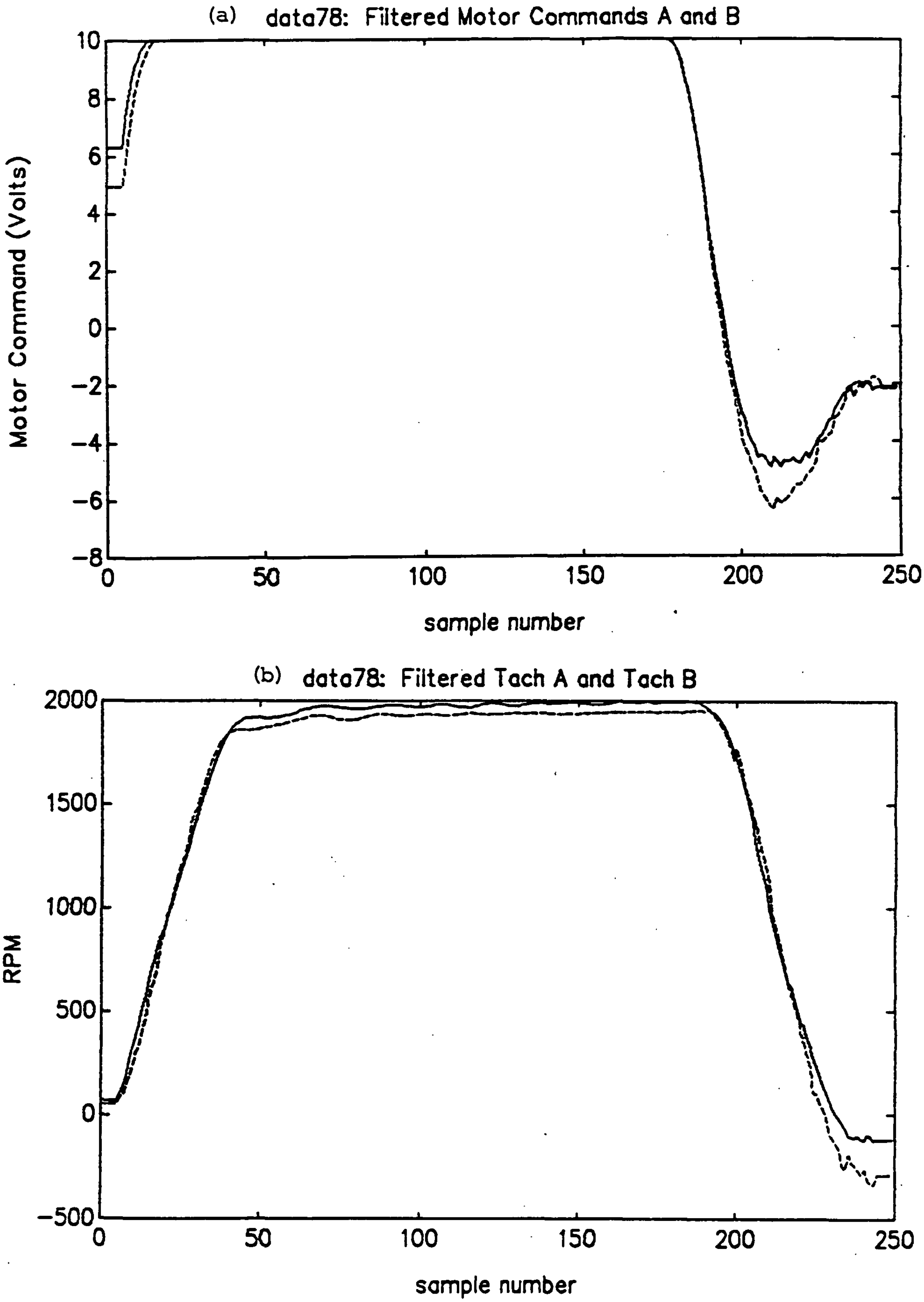


Figure 59 - Pitch/Yaw Joint motor commands (a) and motor speeds (b) for a yaw swing. No load.

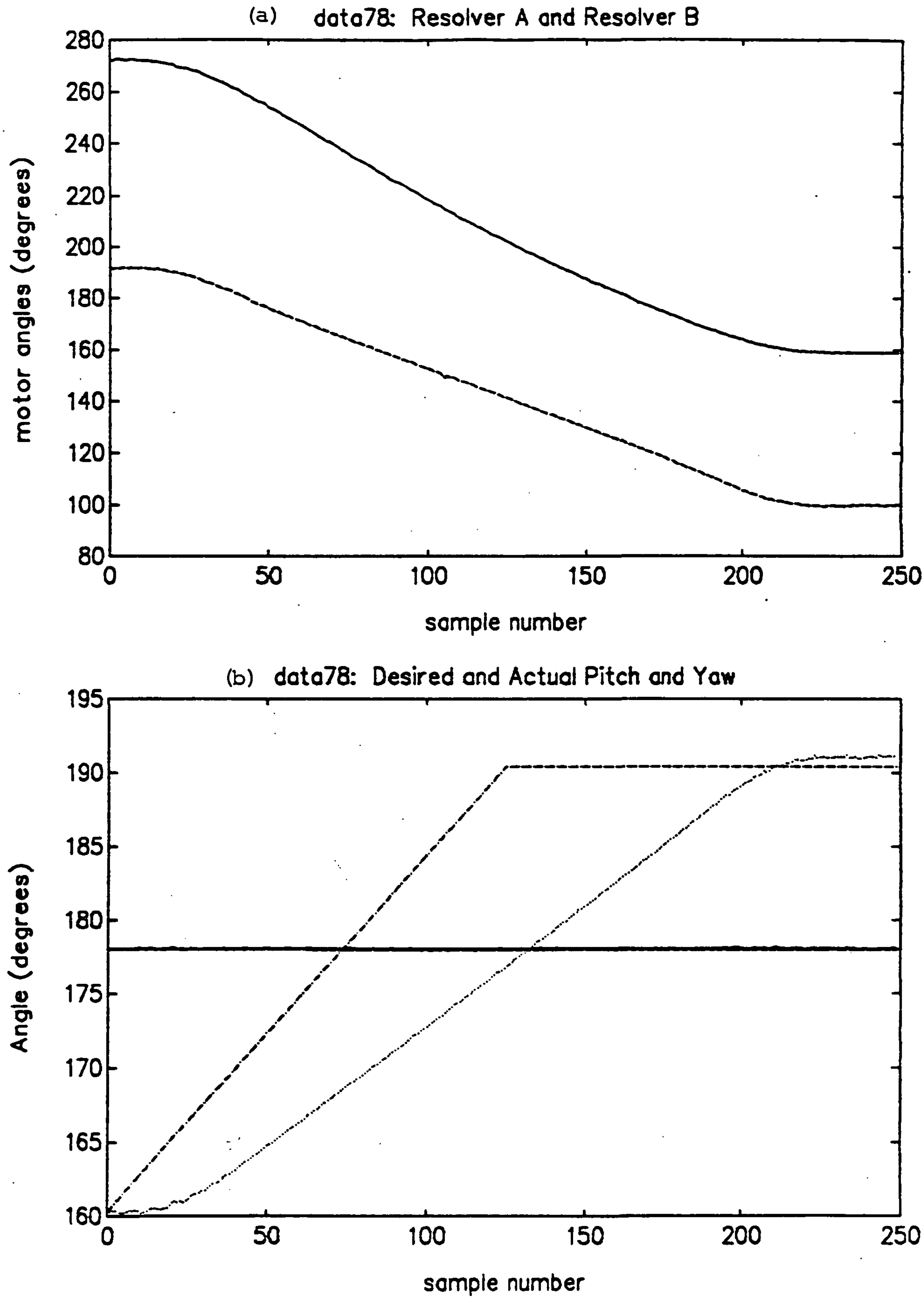


Figure 60 - Pitch/Yaw Joint input resolver positions (a) and desired and actual pitch and yaw positions (b) for a yaw swing. No load.

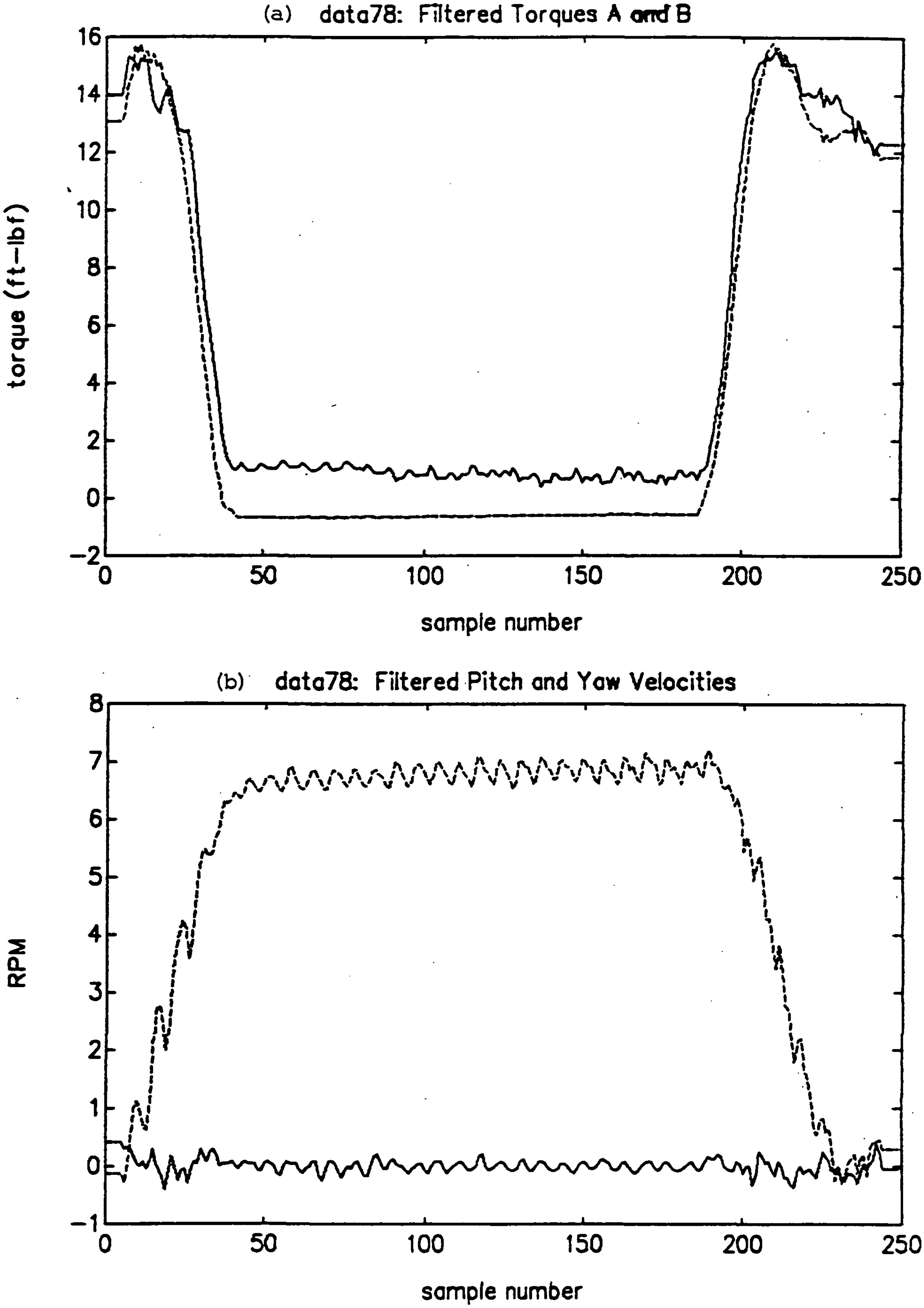


Figure 61 - Pitch/Yaw Joint input torques (a) and pitch and yaw velocities for a yaw swing. No load.

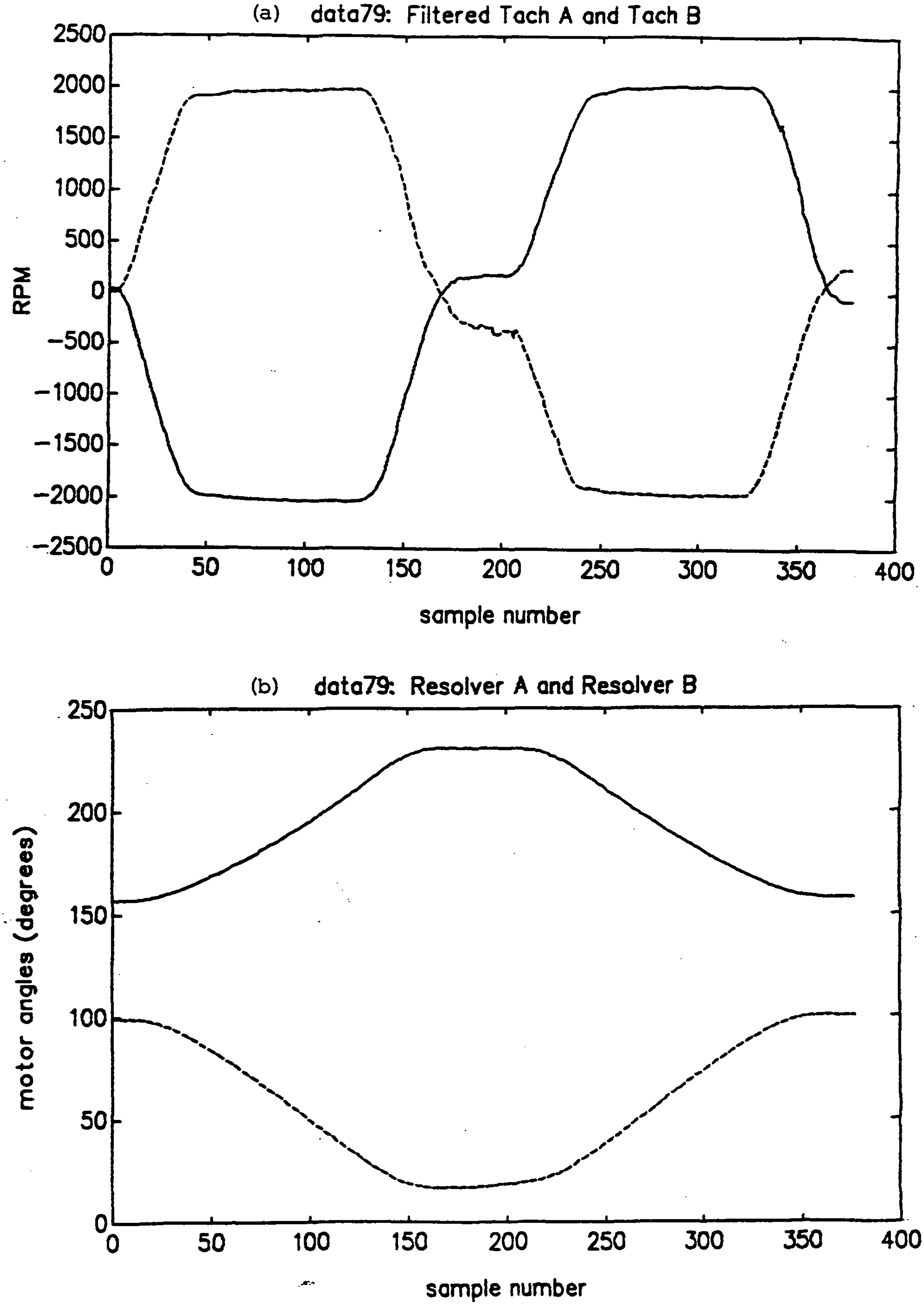


Figure 62 - Pitch/Yaw Joint motor speeds and input resolver positions for a 20° pitch drop, followed by a 20° pitch lift. Zero load.

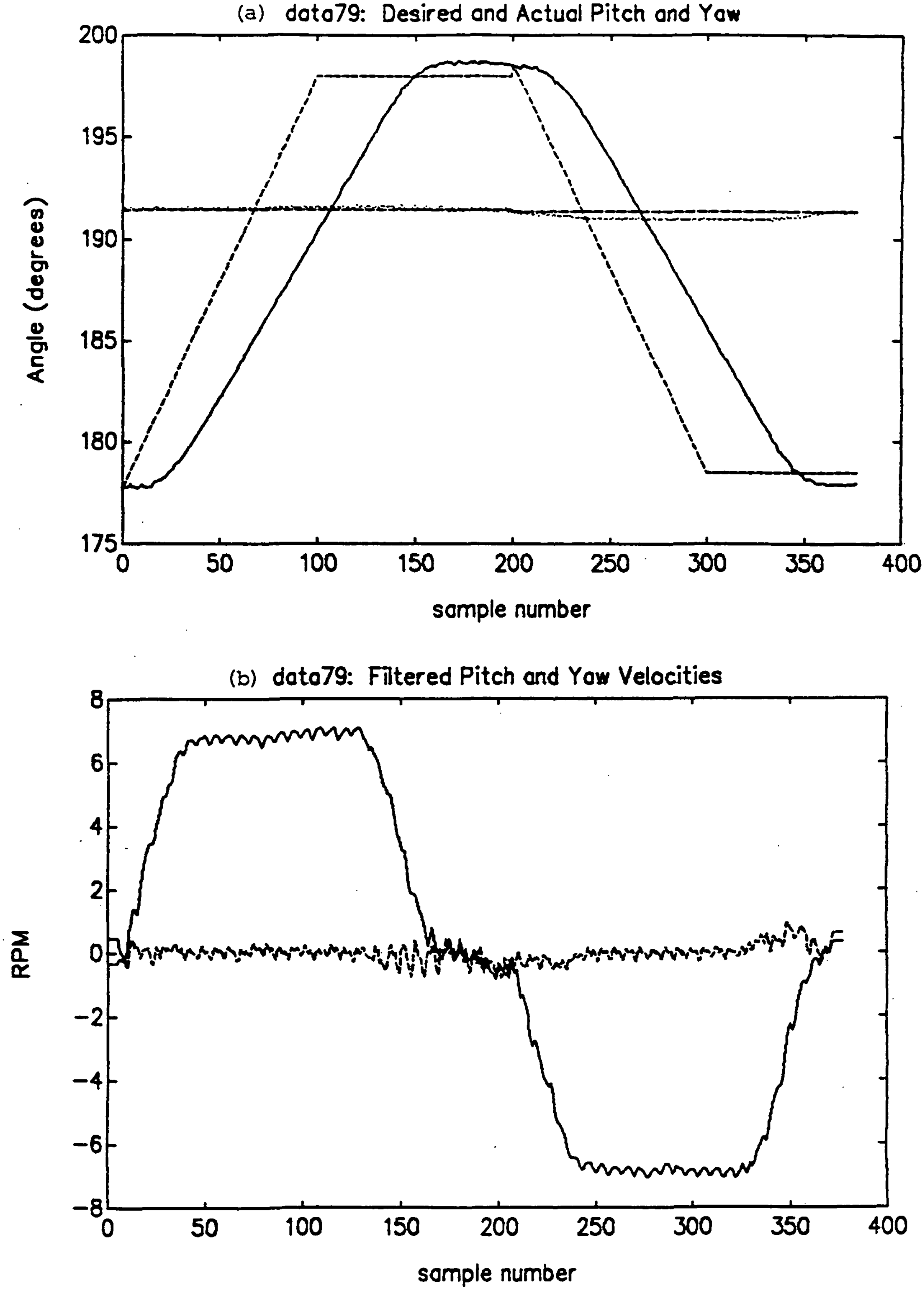
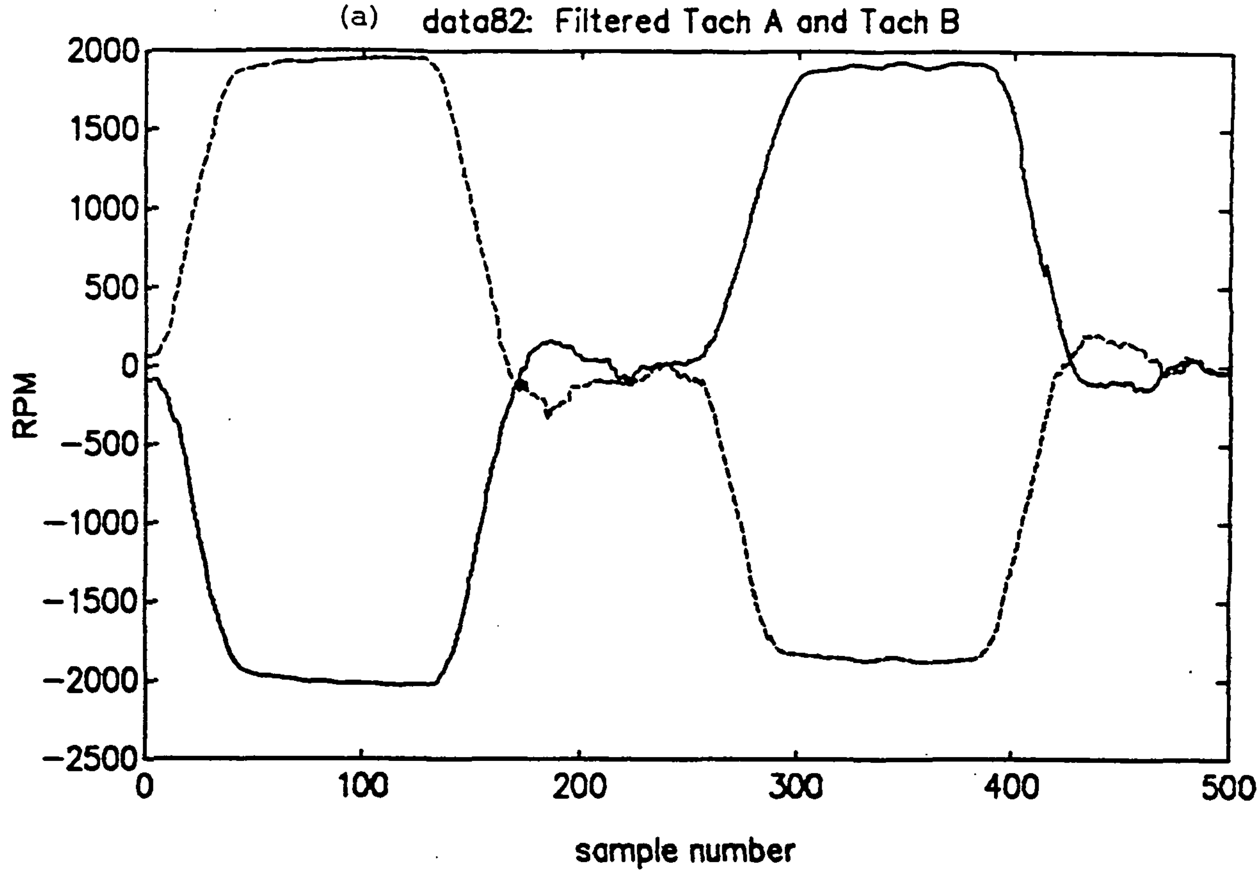


Figure 63 - Pitch/Yaw Joint pitch and yaw positions and velocities for a 20° pitch drop followed by a 20° pitch lift. Zero load.



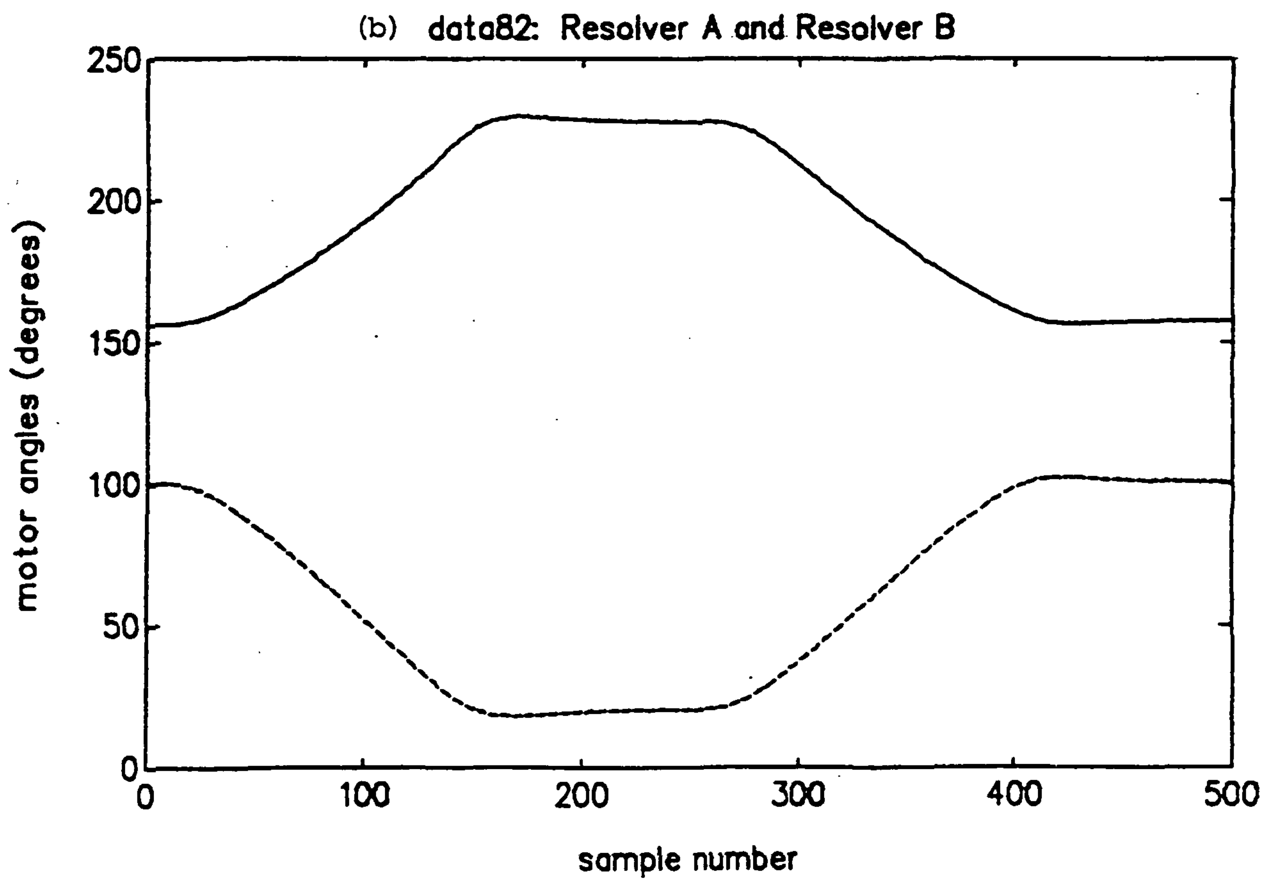


Figure 64 - Pitch/Yaw Joint motor speeds and input resolver positions for a 20° pitch drop followed by a 20° pitch lift. Load, 17.96 ft lbs torque.

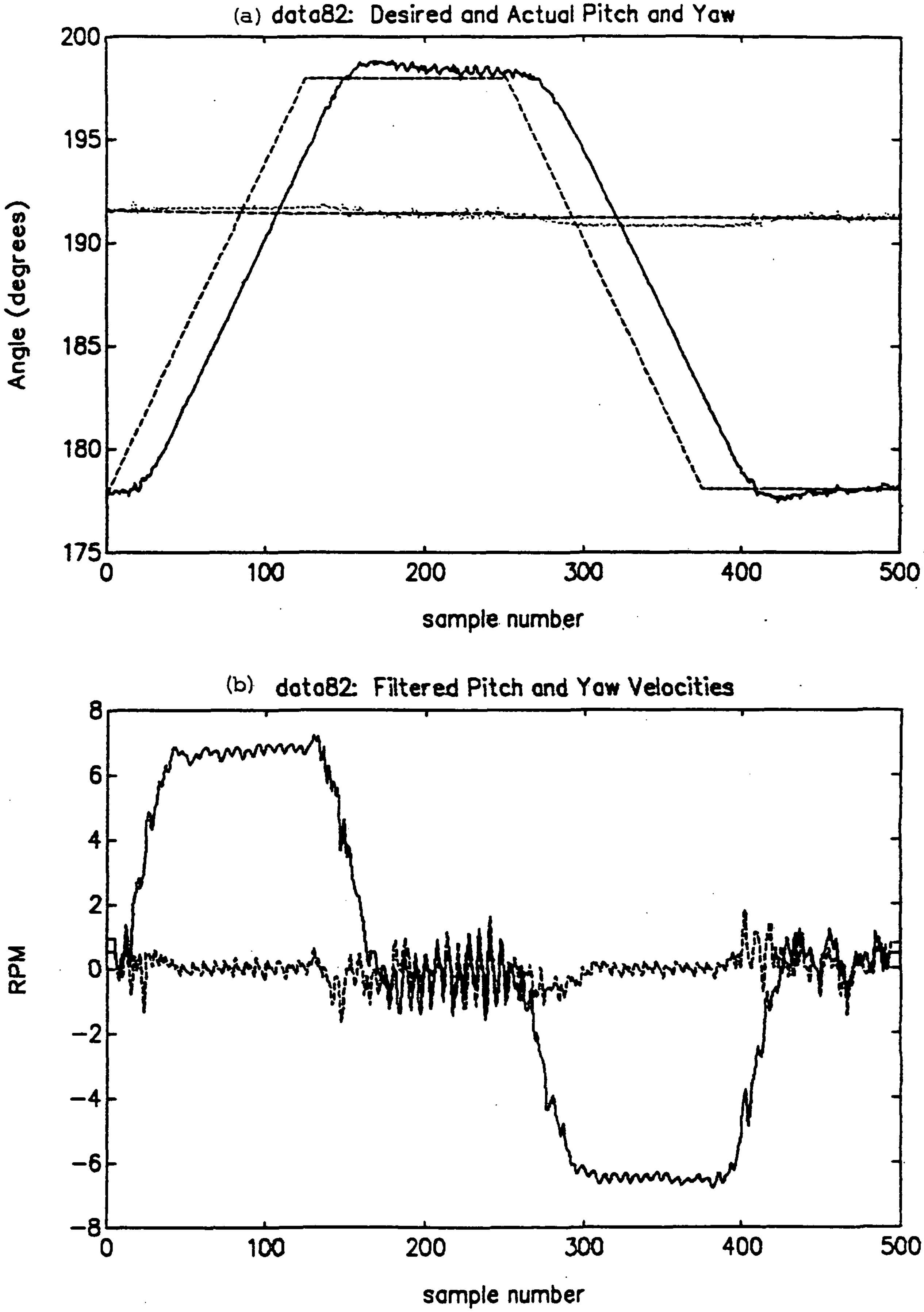
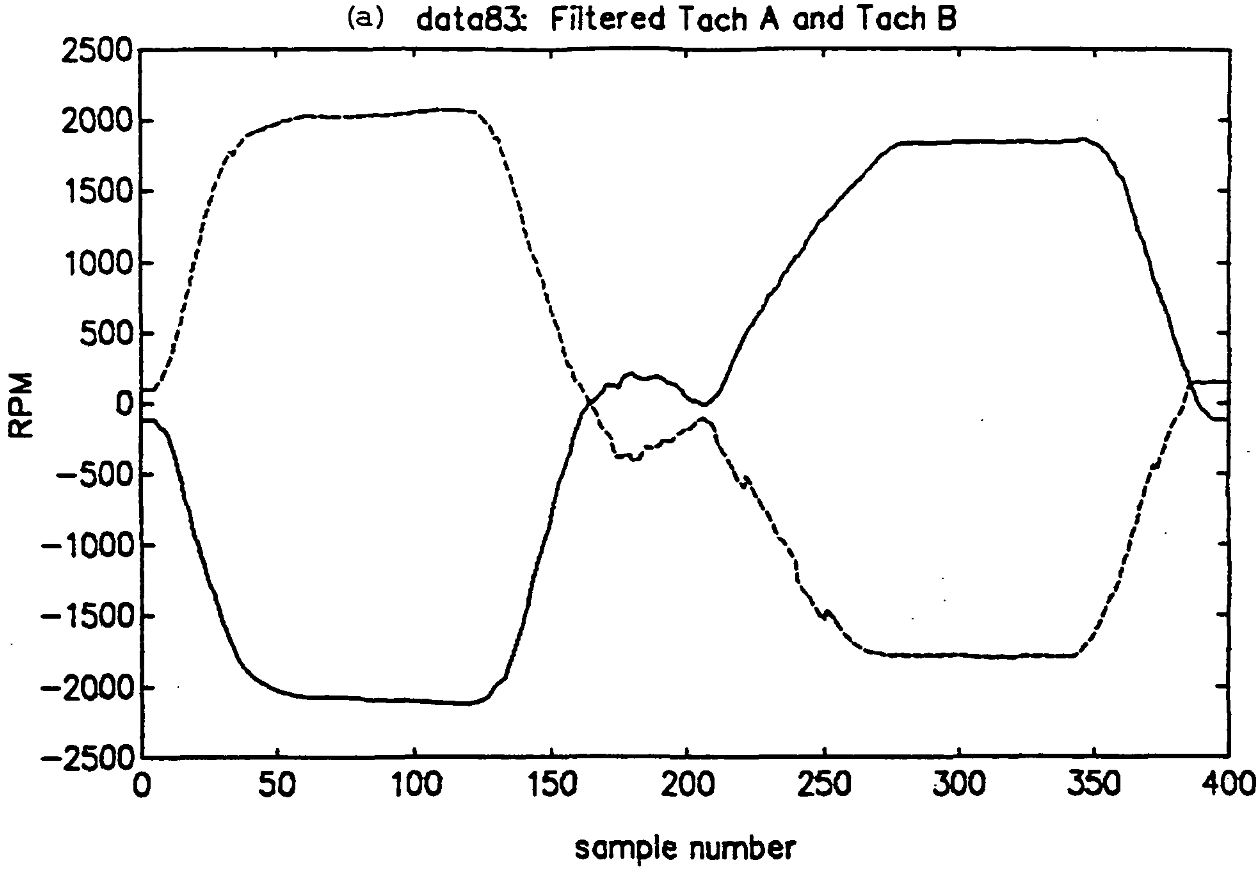


Figure 65 - Pitch/Yaw Joint pitch and yaw positions and velocities for a 20° pitch drop followed by a 20° pitch lift. Load, 17.96 ft lbs torque.



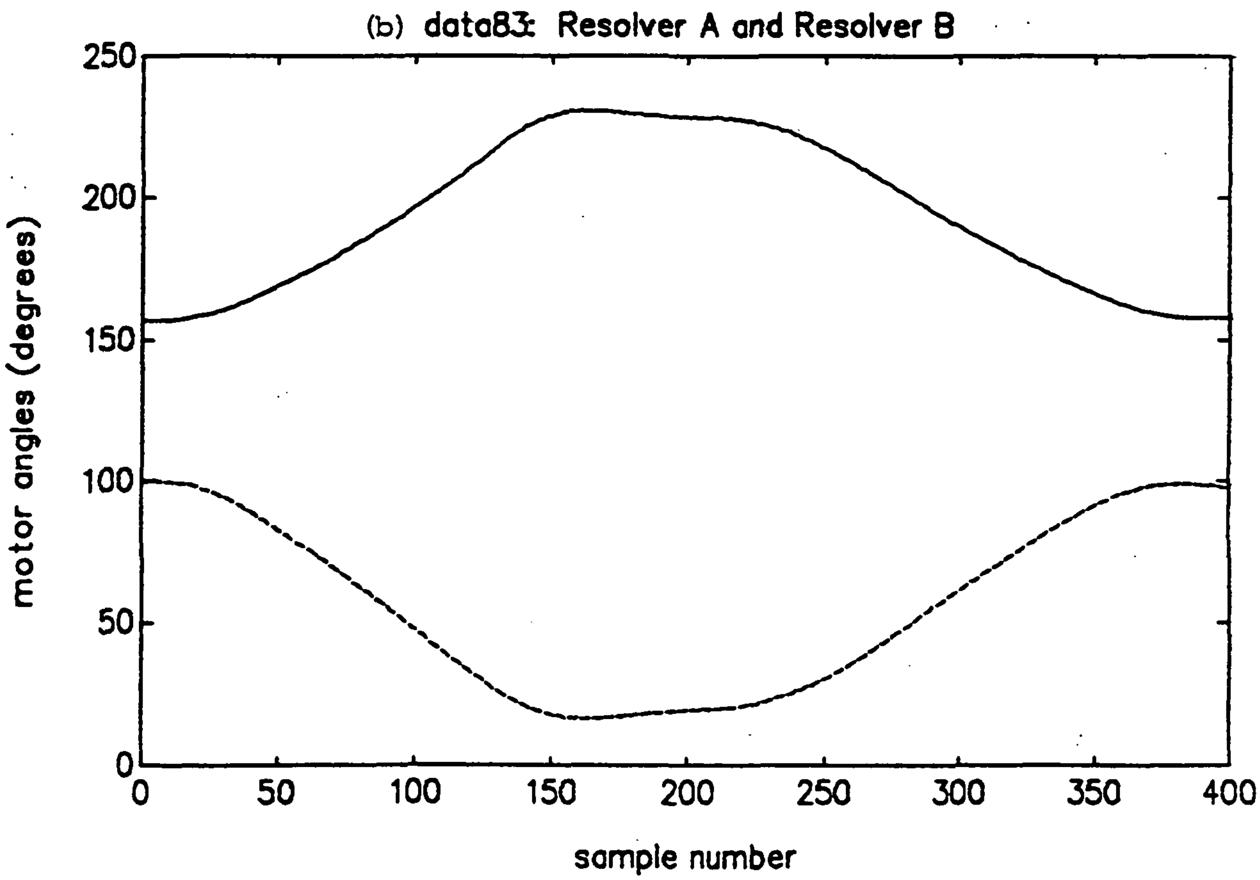


Figure 66 - Pitch/Yaw Joint motor speeds and input resolver positions for a 20^{o} pitch drop followed by a 20^{o} pitch lift. Load, 60.44 ft lbs torque.

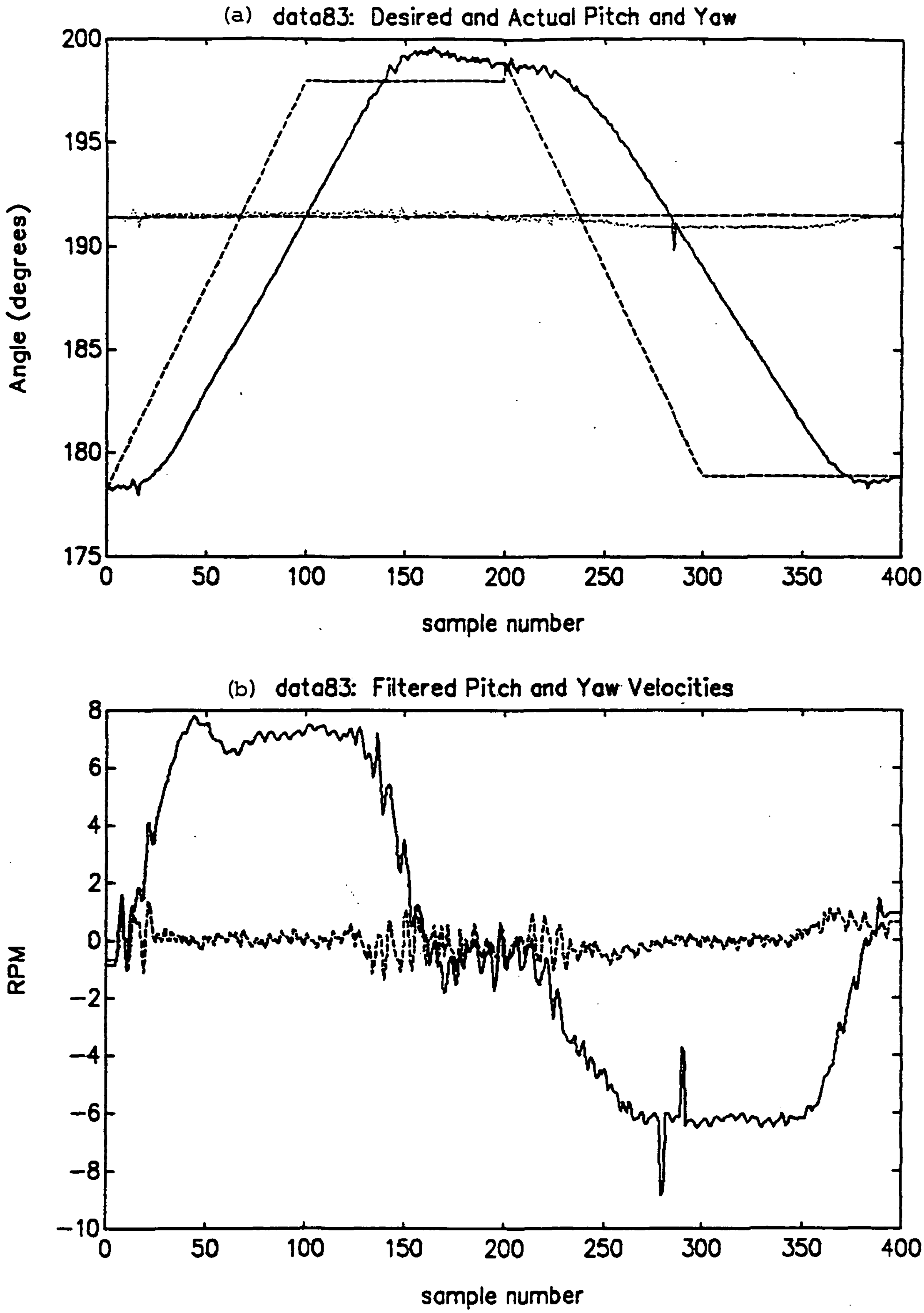


Figure 67 - Pitch/Yaw Joint pitch and yaw positions and velocities for a 20° pitch drop followed by a 20° pitch lift. Load, 60.44 ft lbs torque.

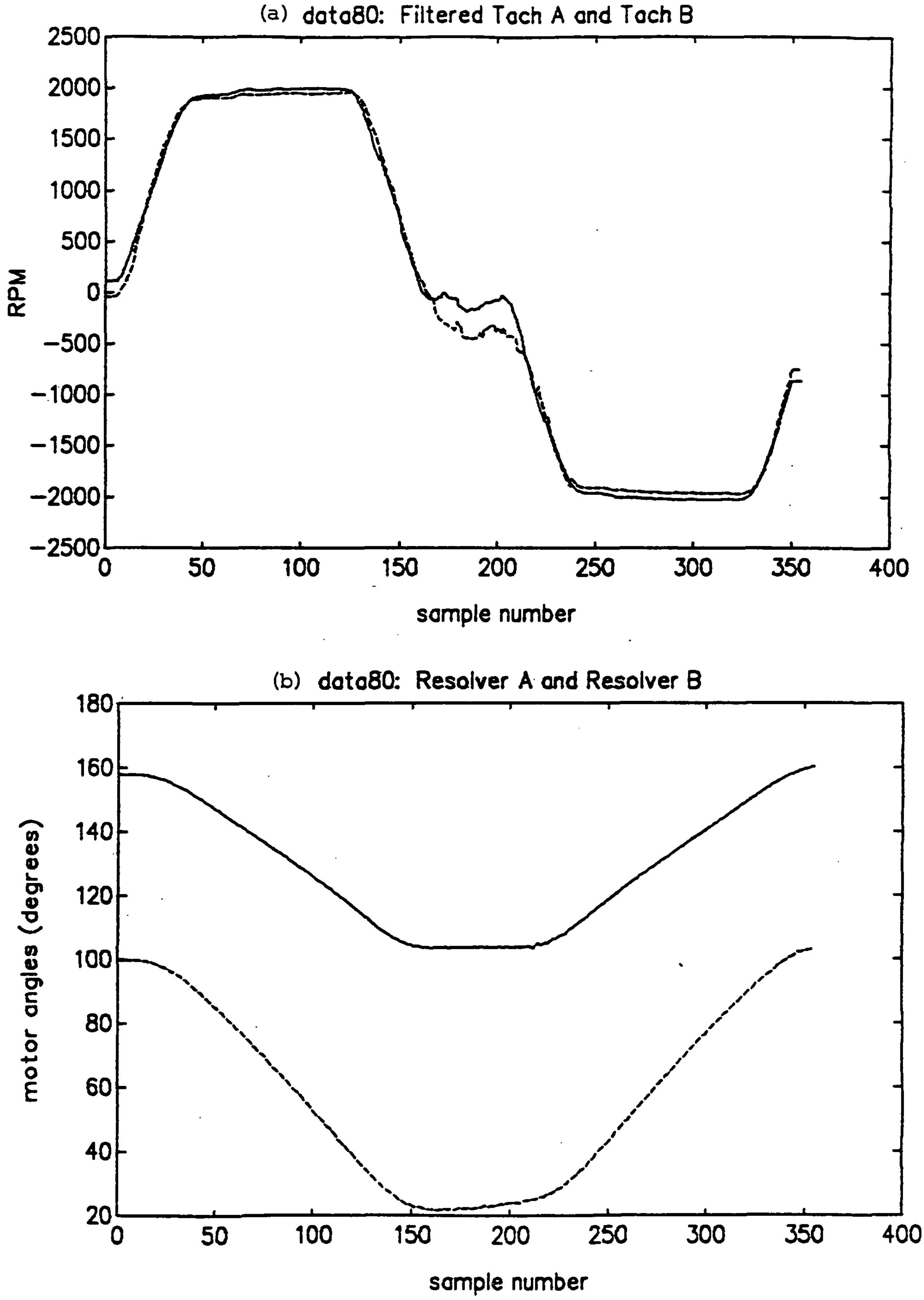


Figure 68 - Pitch/Yaw Joint motor speeds and input resolver positions for 20° yaw swings back and forth. Zero load.

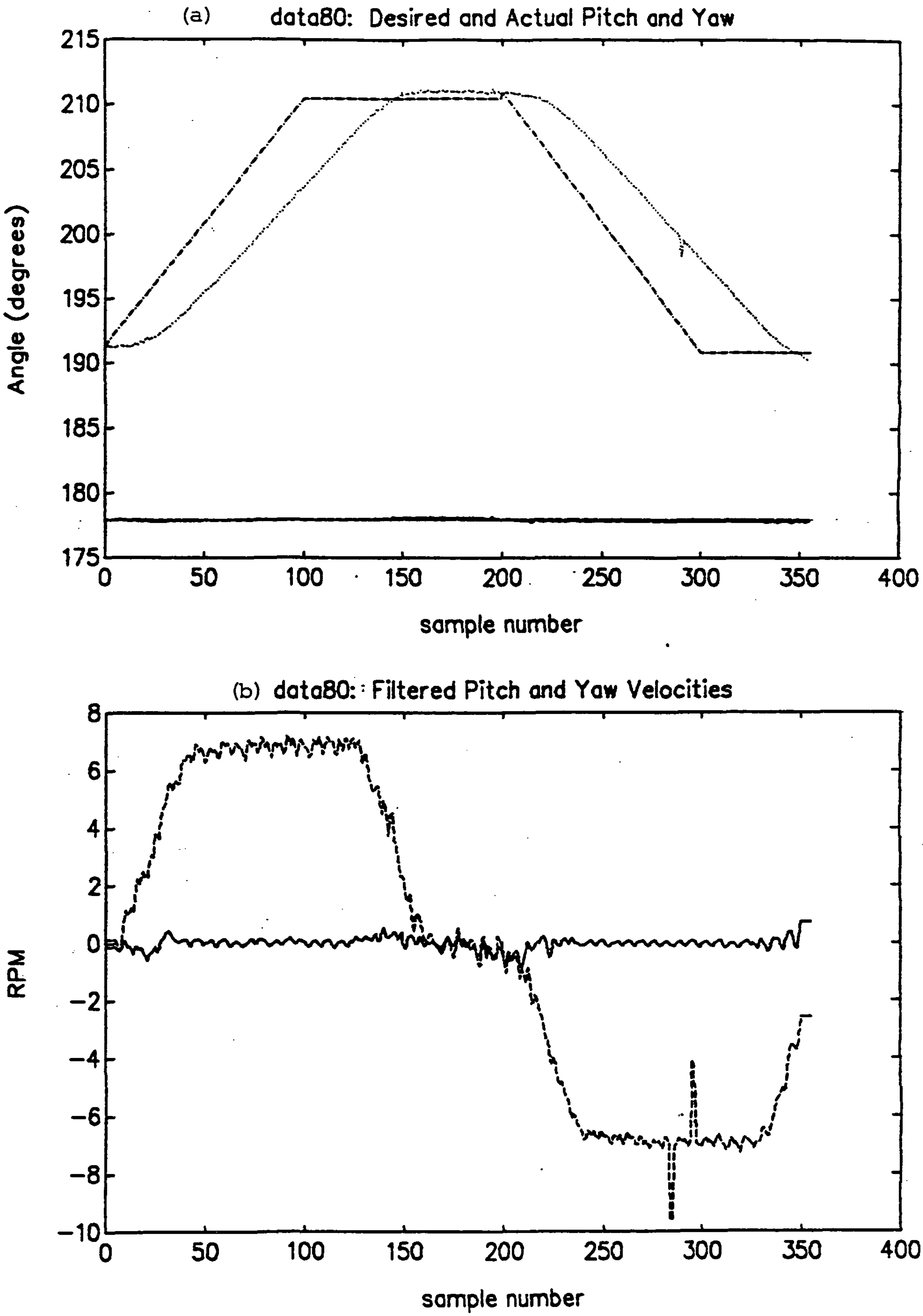


Figure 69 - Pitch/Yaw Joint pitch and yaw positions and velocities for 20° yaw swings back and forth. Zero load.

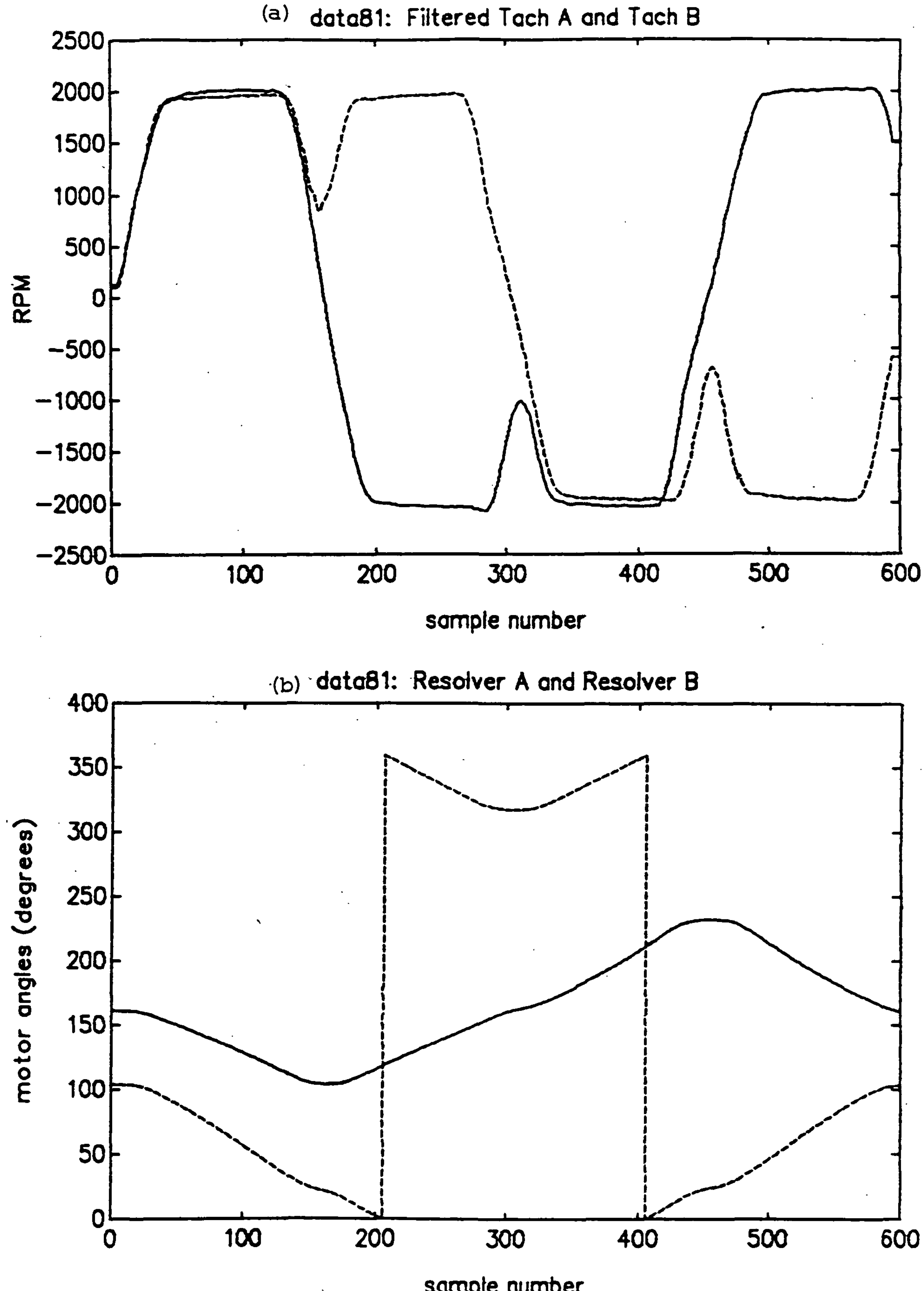
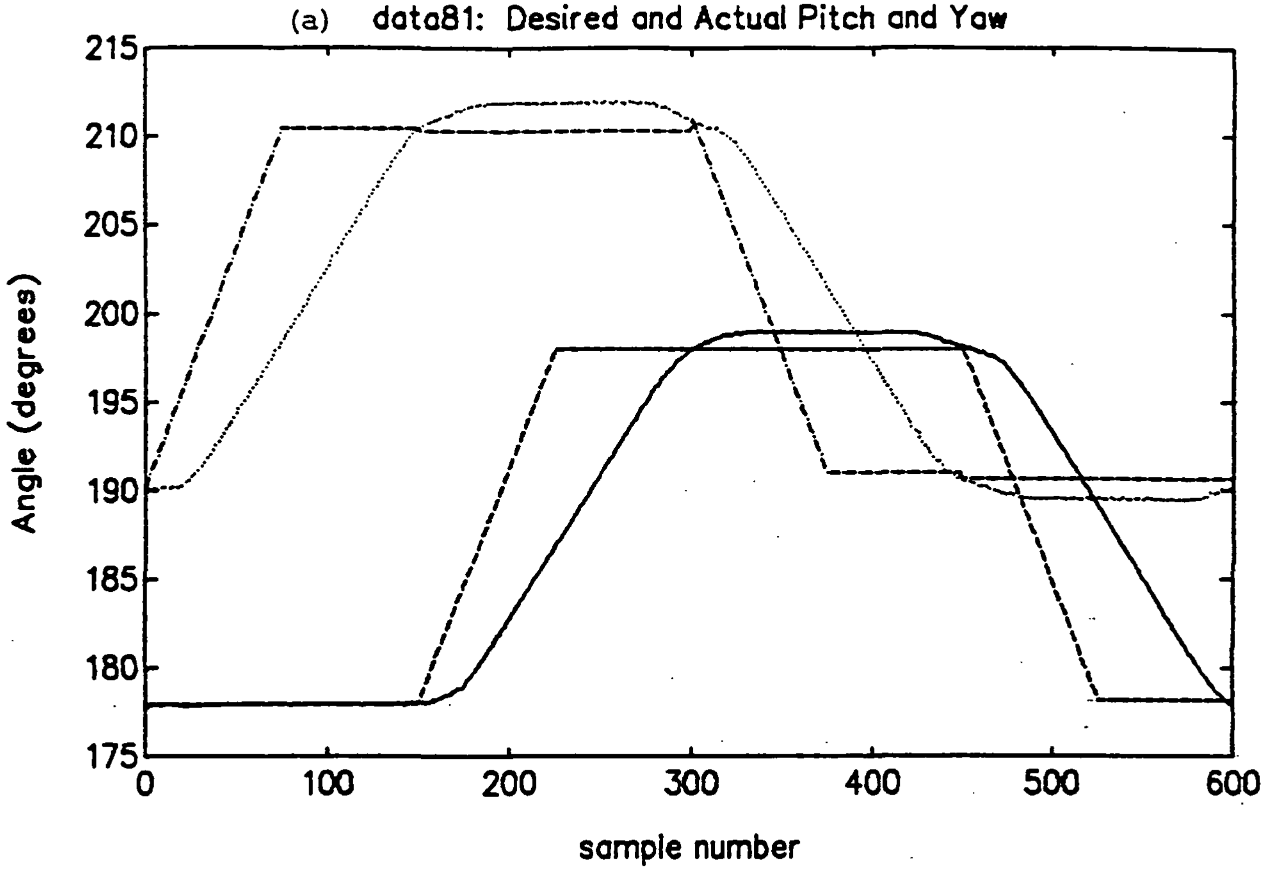


Figure 70 - Pitch/Yaw Joint motor speeds and input resolver positions for a 20° yaw swing followed by a 20° pitch drop followed by a 20° yaw swing back and a 20° pitch lift. Zero load.



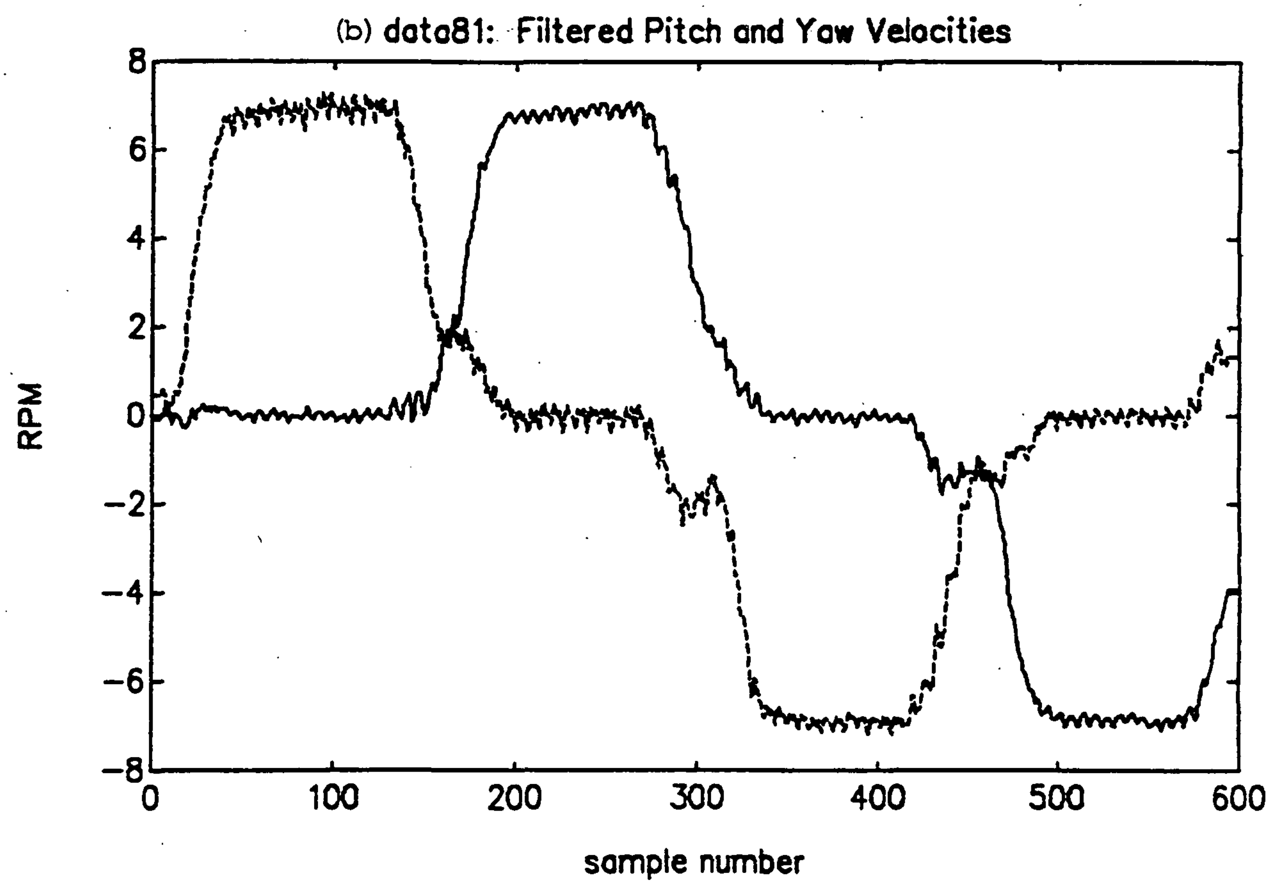
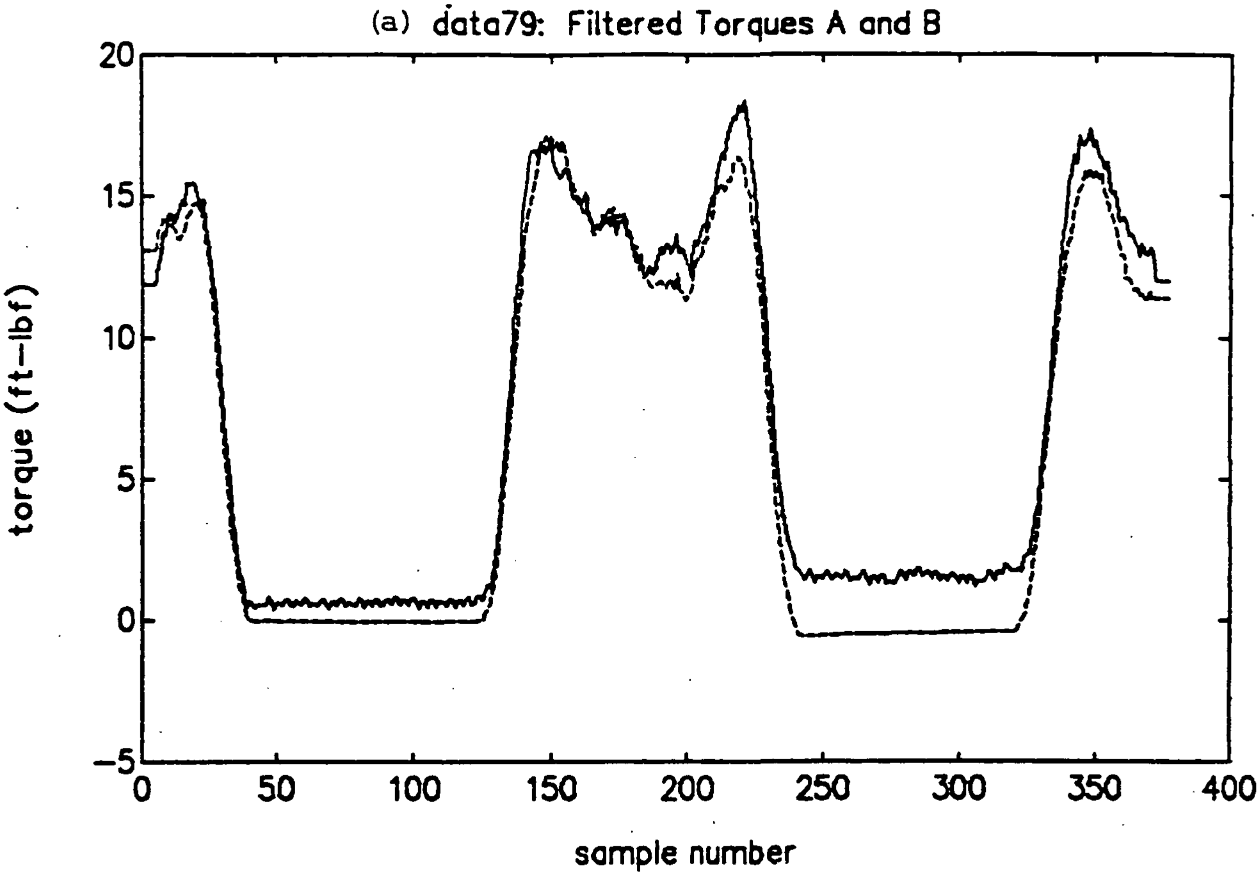


Figure 71 - Pitch/Yaw Joint pitch and yaw positions and velocities for a 20° yaw swing followed by a 20° pitch drop followed by a 20° yaw swing back and a 20° pitch lift. Zero load.



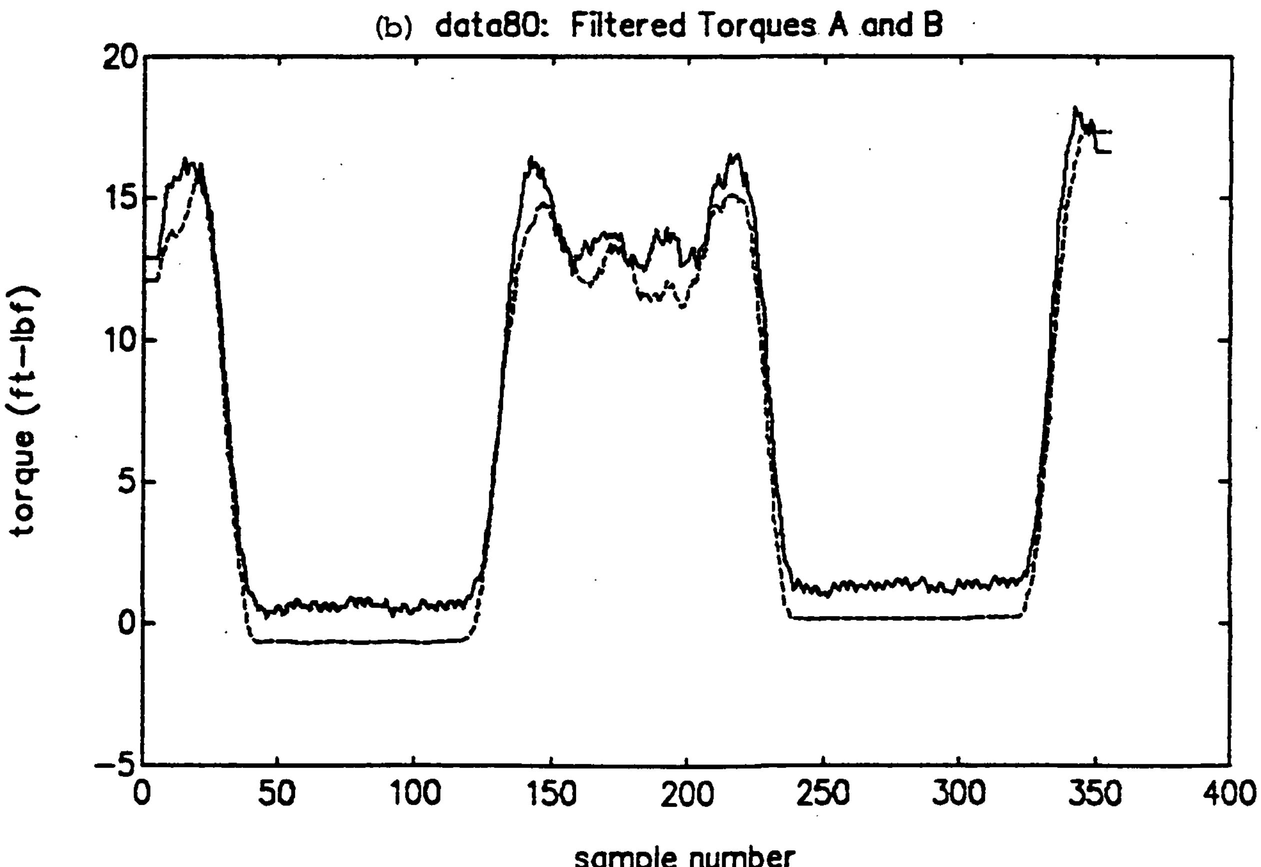
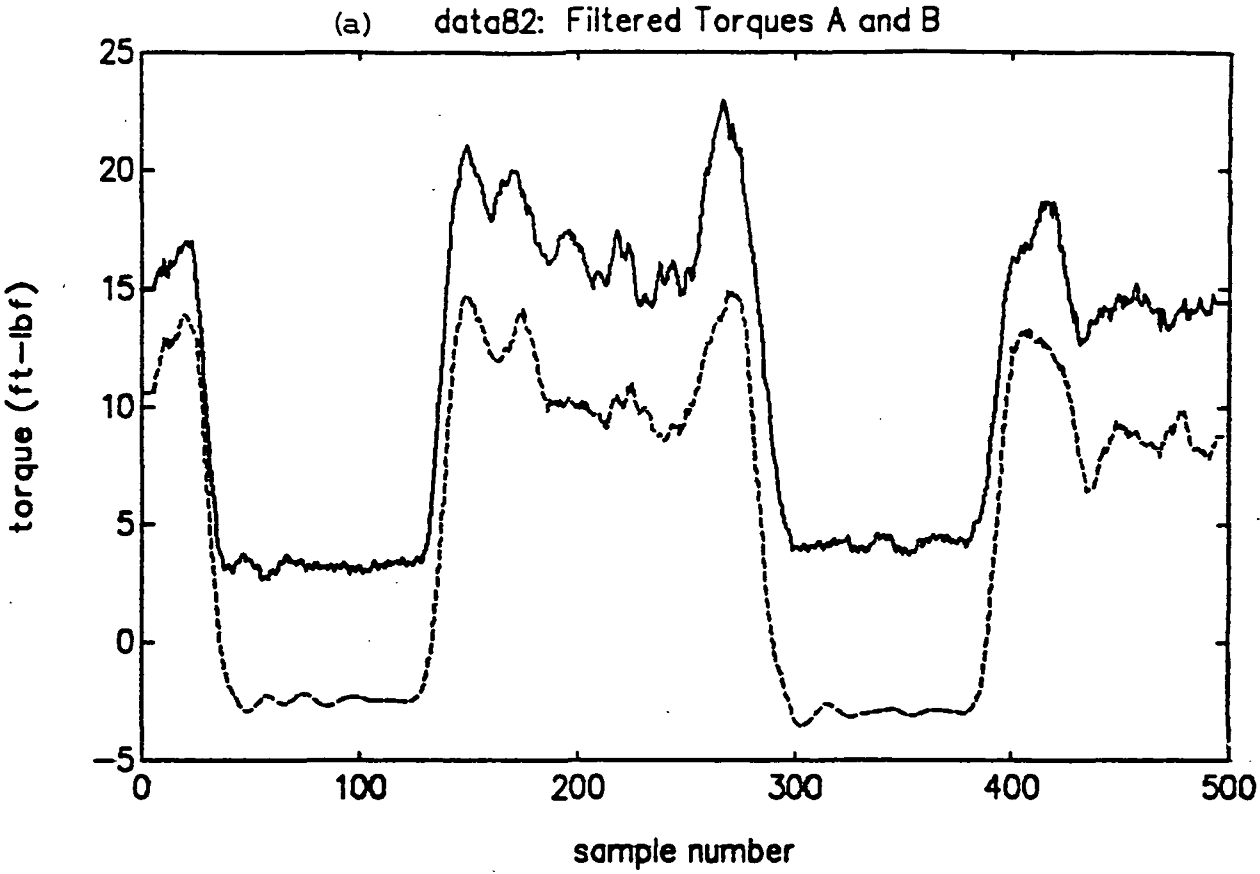


Figure 72 - Pitch/Yaw Joint input torques for (a) 20° pitch drop-pitch lift and (b) 20° yaw swing and return. Zero load.



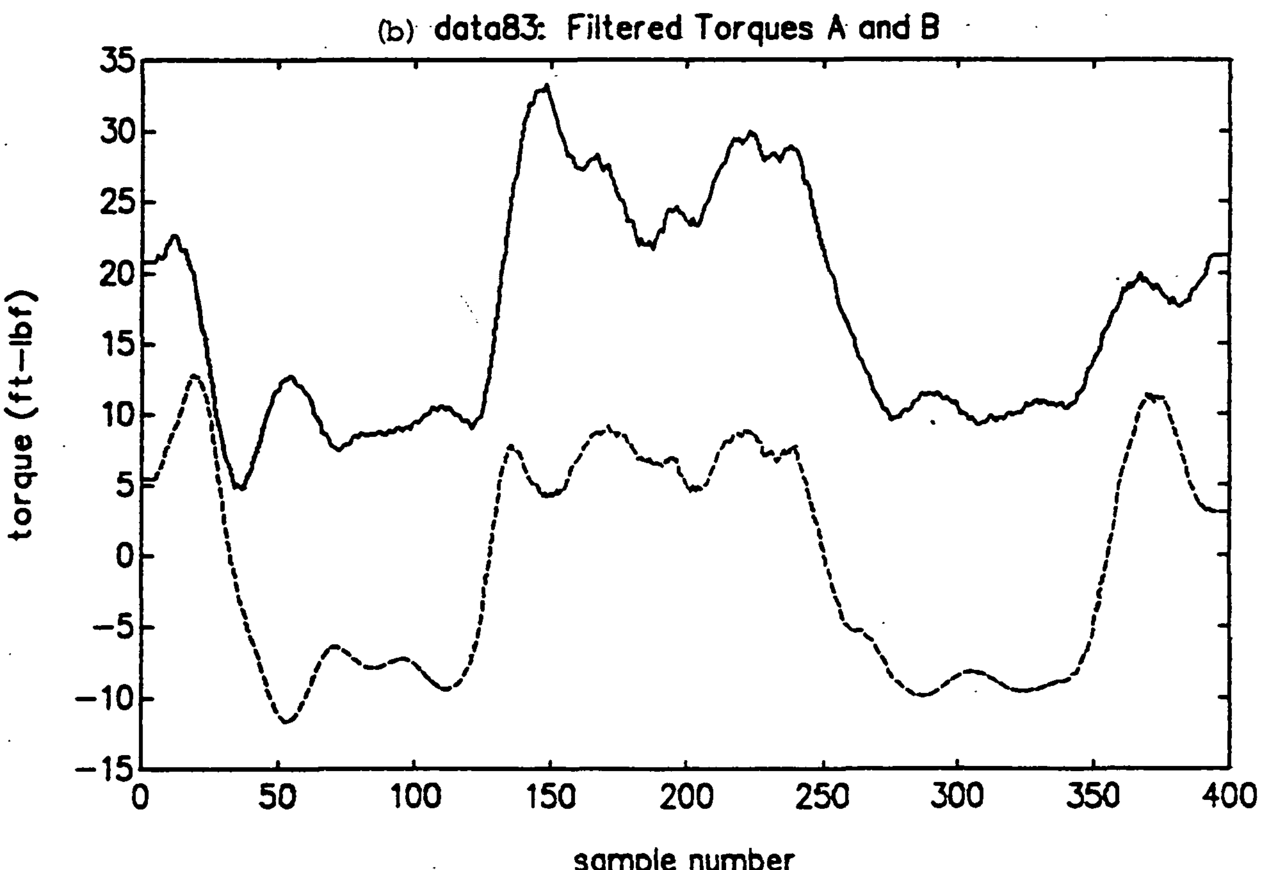


Figure 73 - Pitch/Yaw Joint input torques for 20° pitch drop followed by 20° pitch lift. (a) Load, 17.96 ft lbs torque; (b) Load, 60.44 ft lbs torque.

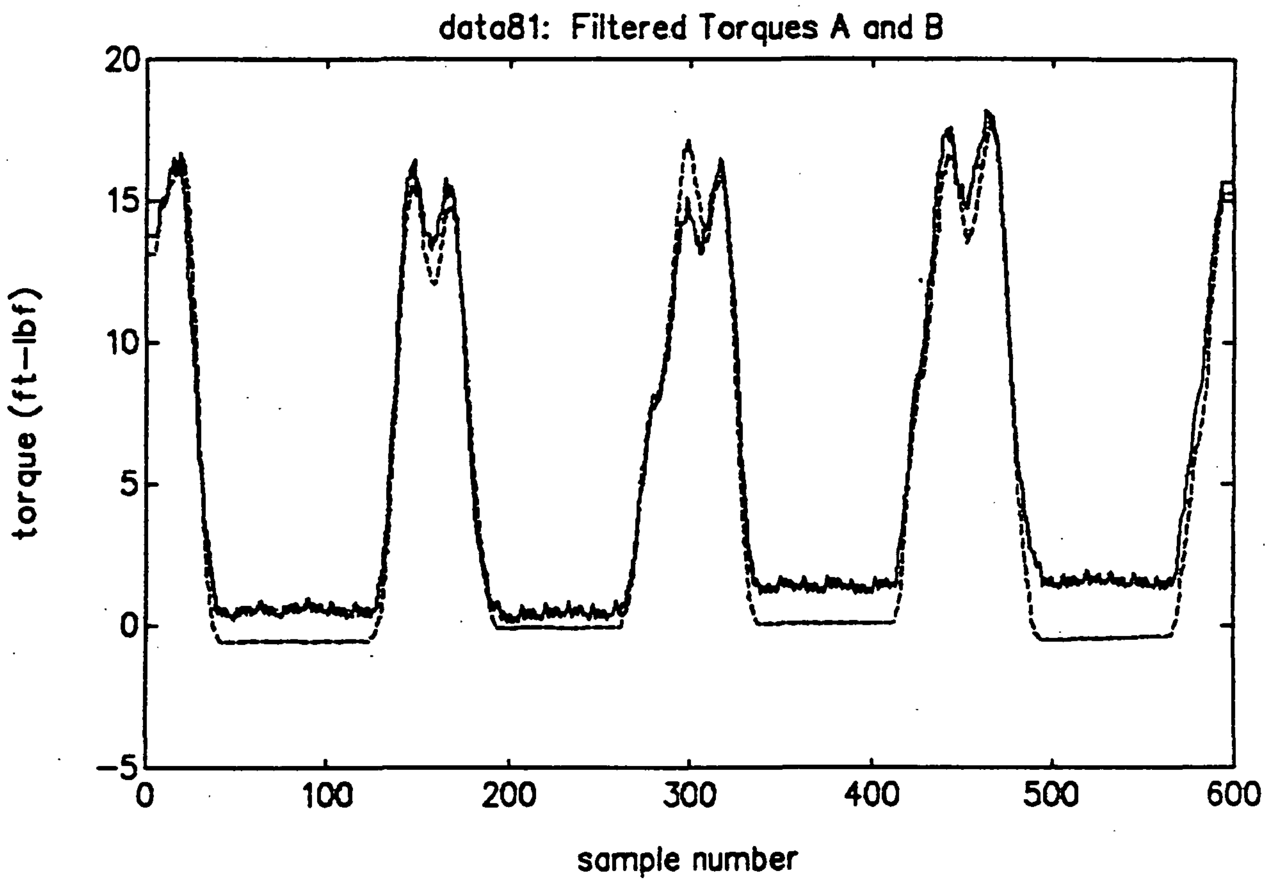


Figure 74 - Pitch/Yaw Joint input torques for a 20° yaw swing followed by a 20° pitch drop, followed by a 20° yaw swing back and a 20° pitch lift. Zero load.

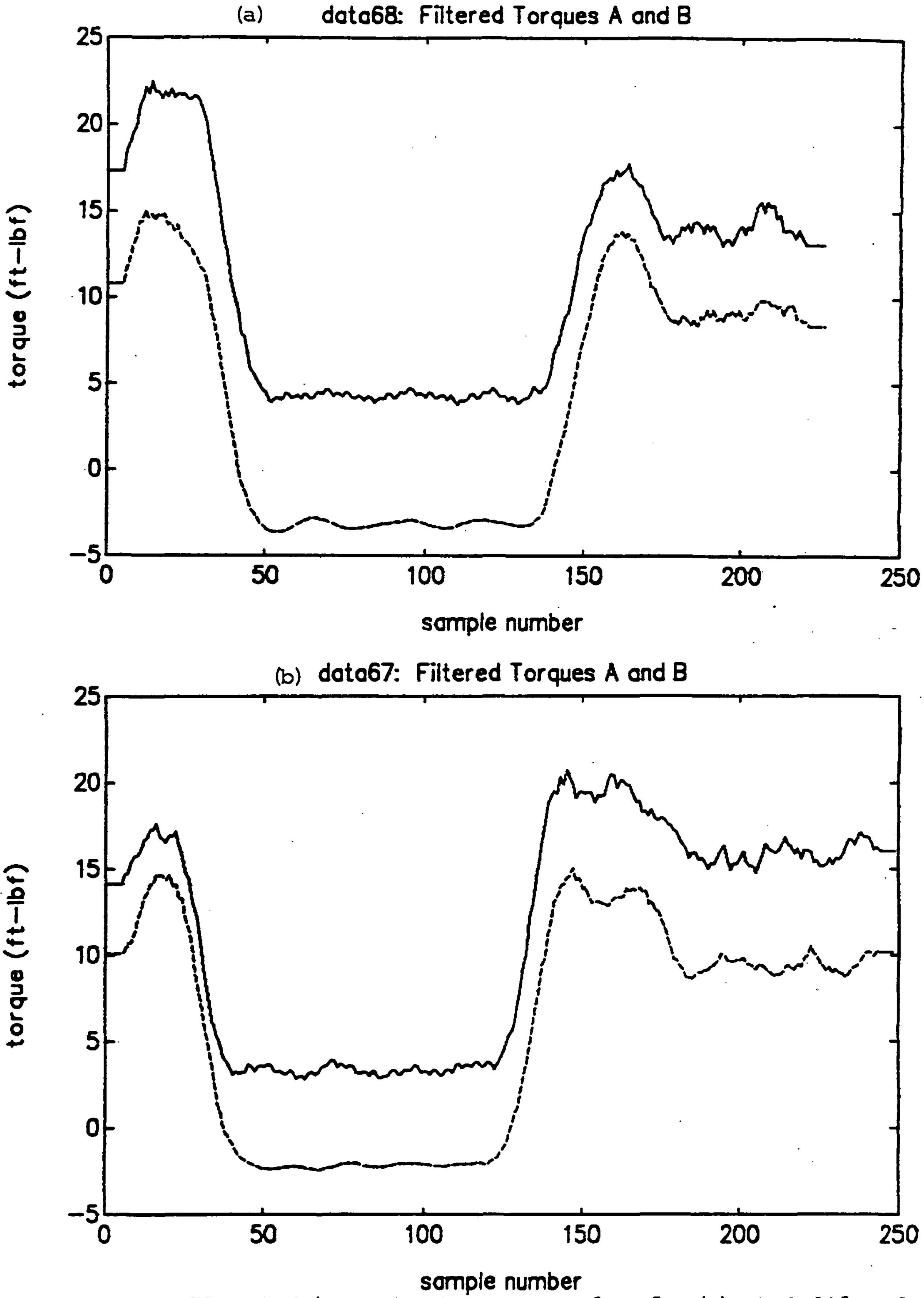
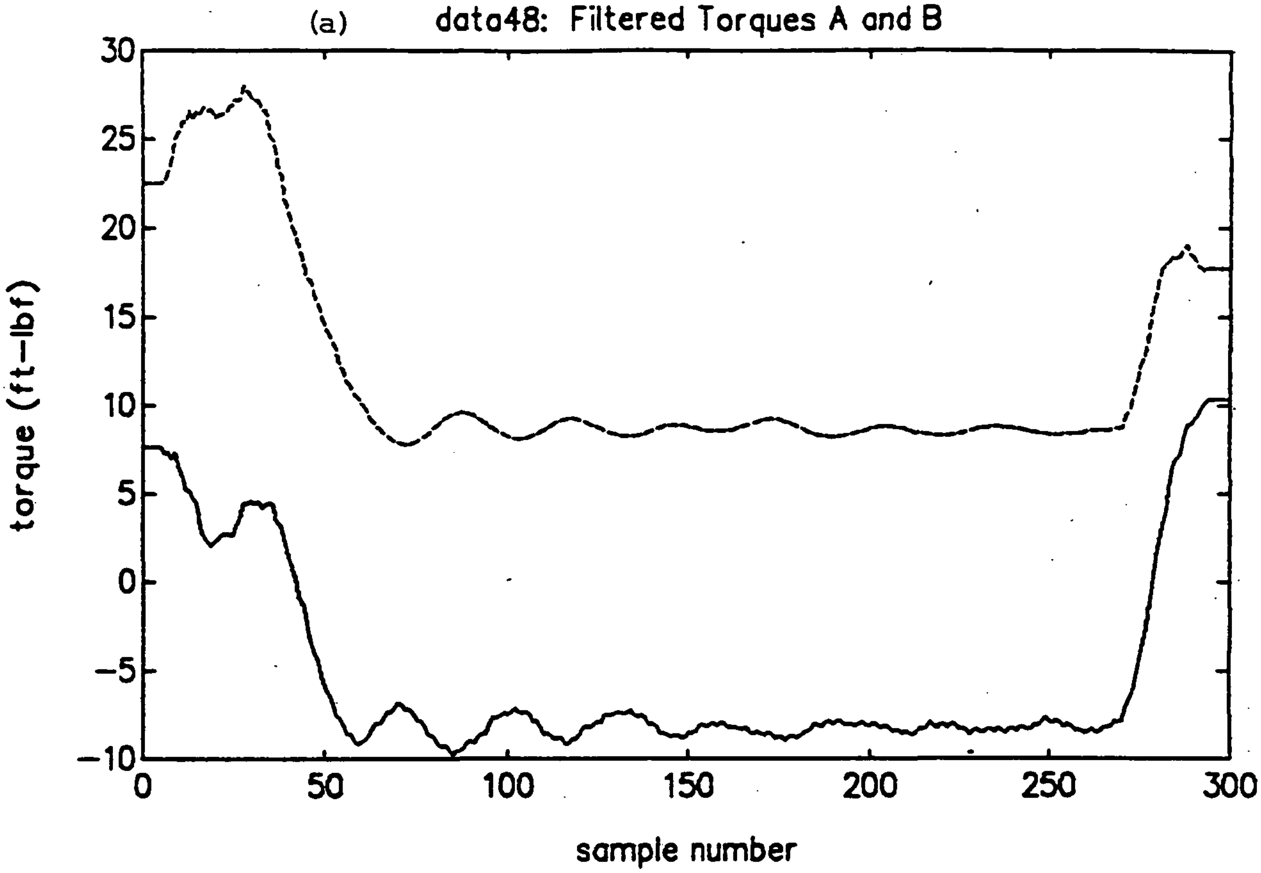


Figure 75 - Pitch/Yaw Joint input torque data for (a) pitch lift and (b) pitch drop. Load, 17.96 ft lbs torque.



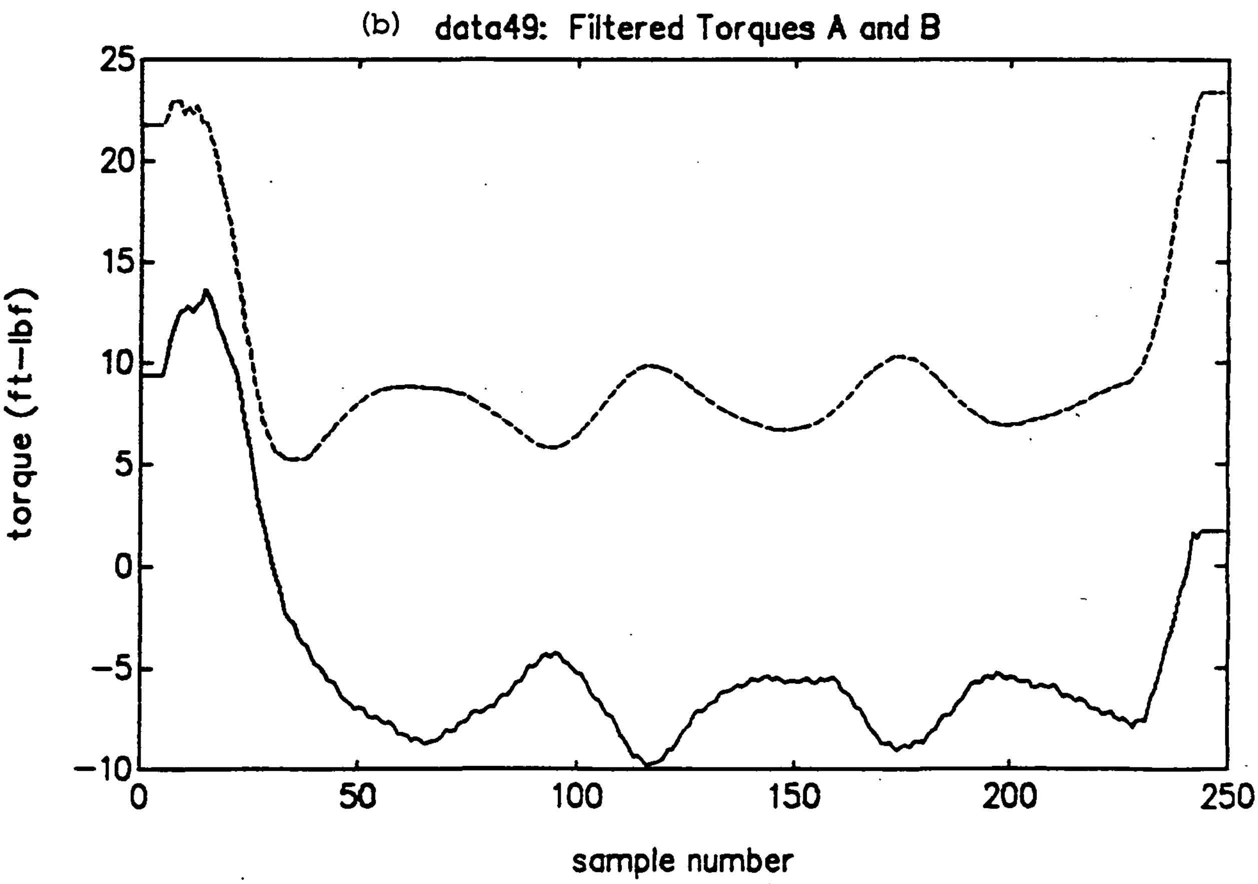


Figure 76 - Pitch/Yaw Joint input torque data for (a) pitch lift and (b) pitch drop. Load, 17.96 ft lbs torque.

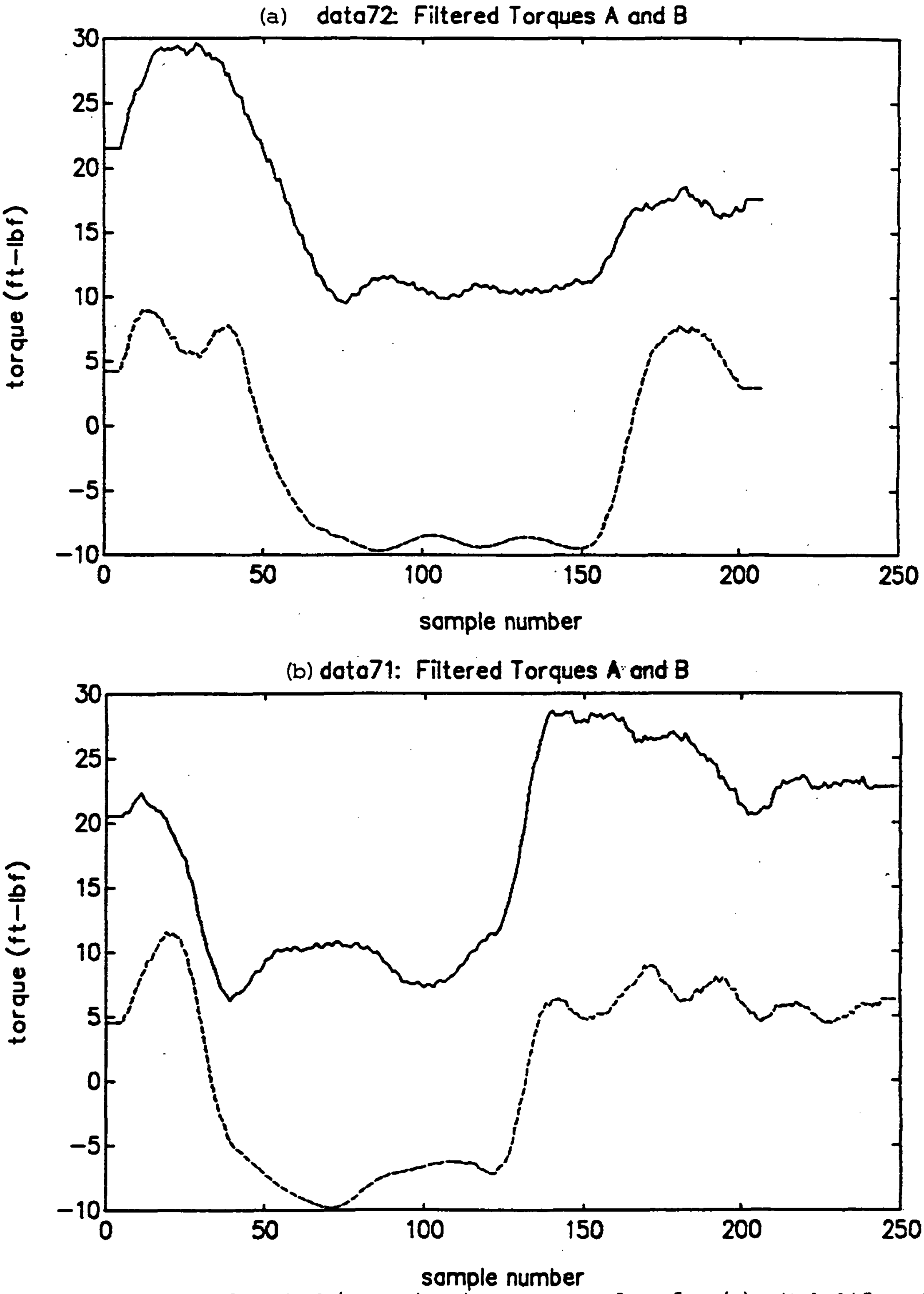
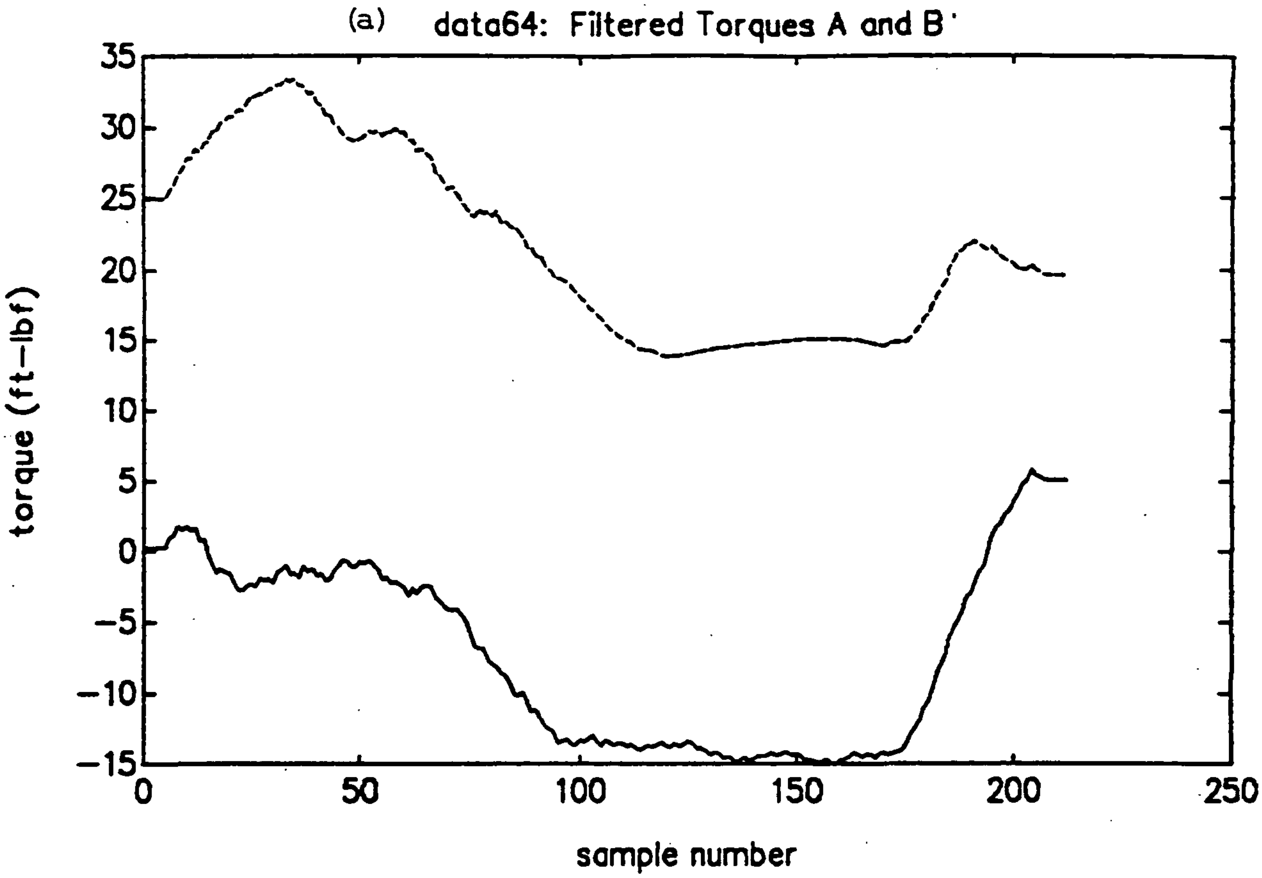


Figure 77 - Pitch/Yaw Joint input torque data for (a) pitch lift and (b) pitch drop. Load, 60.44 ft lbs torque.



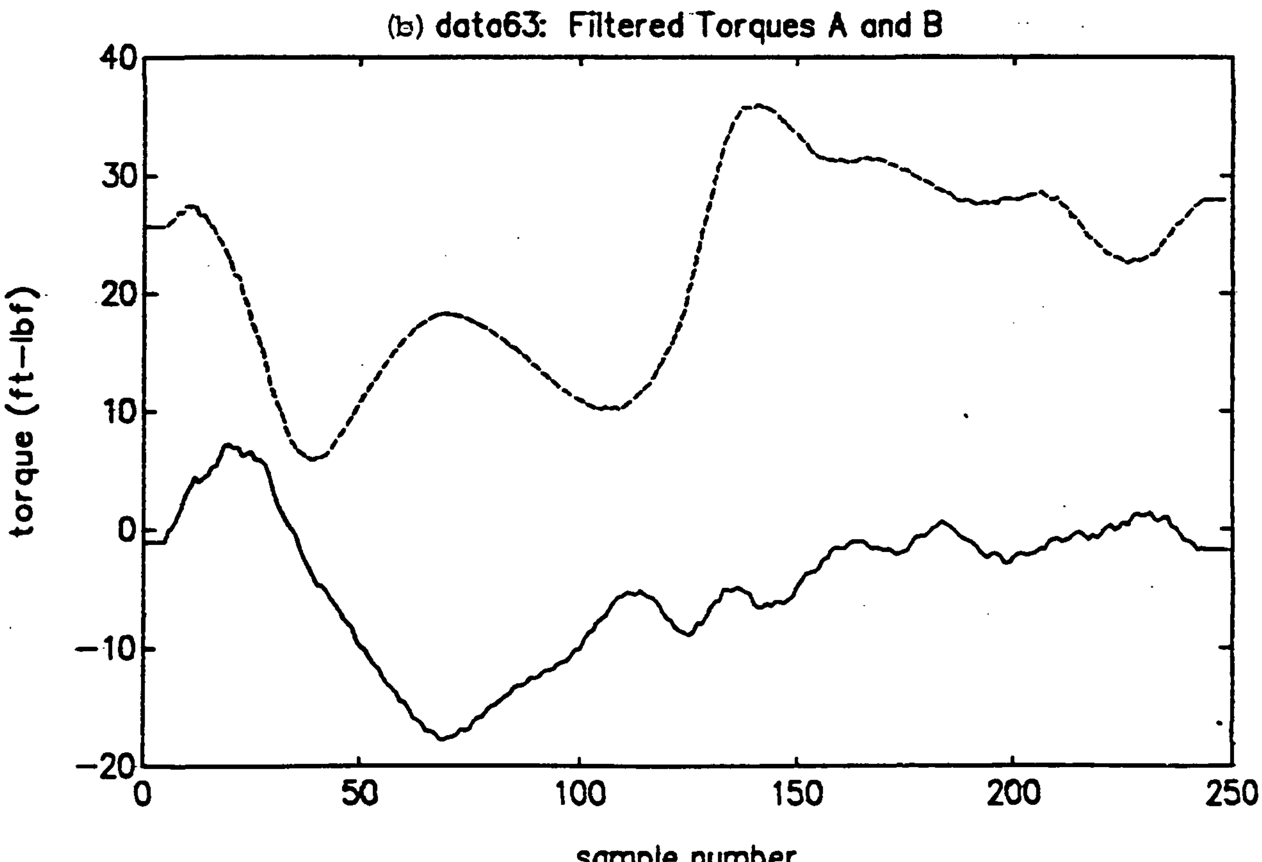


Figure 78 - Pitch/Yaw Joint input torque data for (a) pitch lift and (b) pitch drop. Load, 100.73 ft lbs torque.

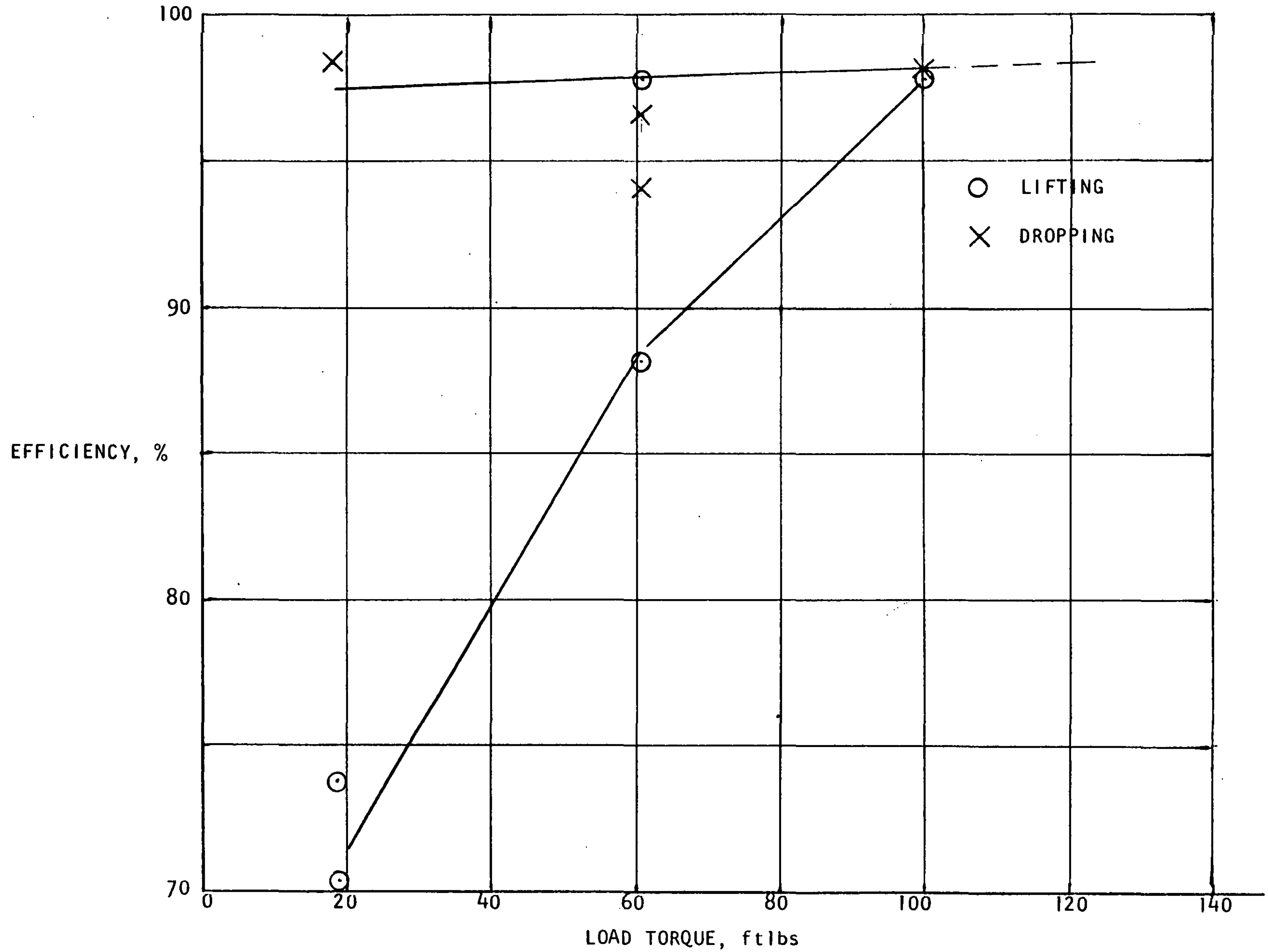


Figure 79 - Pitch/Yaw Joint efficiency as a function of load torque in pitch moves.

APPENDICES

A. WRIST ROLL JOINT ROLLER-GEAR DRIVE GEAR DATA

B. HINGE JOINT ROLLER-GEAR DRIVE GEAR DATA

C. PITCH YAW JOINT ROLLER-GEAR DRIVE GEAR DATA

D. SOFTWARE FOR PITCH YAW JOINT

APPENDIX A

WRIST-ROLL JOINT ROLLER-GEAR DRIVE GEAR DATA

Specifications for the Wrist Roll drive were:

Nominal ratio	25:1
Output torque	26 Nm (230 in.lb.)
Output speed	120 rpm
Maximum diameter	152.4 mm (6 inches)

Cluster Geometry

The cluster geometry chosen from the Phase I study (ref. 1), when carried forward into detail design, was found to present some assembly difficulties so a slightly modified four planet system was chosen from Table I of ref. 1:

$\alpha = 6.000^\circ$ $\gamma = 39.000^\circ$

Using equations 1 through 12 of ref. 1:

$(7/15)\ N_{y1} =$ integral number $N_{y1} = 15$ teeth

$(13/60)\ N_{x2} =$ integral number $N_{x2} = 60$ teeth

Selecting

$a + x_1 = .870$

$N_a = 24$ teeth $N_{x1} = 51$ teeth

$x_1/a = 51/24 = 2.125$ $a = .870/3.125 = .2784$ in.

$x_1 = .5916$ in.

Operating P.D. $a = (.2784)(2) = .5568$ $x_1 = (.5916)(2) = 1.1832$

Selecting a cutter with $P = 43$

Cutting P.D. $a = 24/43 = .5581395$ $x_1 = 51/43 = 1.1860465$

Maximum permissible OD of x_1 gear for clearance

$OD_{x1} = (.870)(\sin 45)(2) = 1.2304$

Standard addendum = $1/43 = .02325$

Standard $OD_{x1} = 1.186 + 2(.02325) = 1.2325$

Use a short addendum for $x_1 = .70/43 = .0163$

Long addendum for $a = 1.30/43 = .0302$

$OD_{x1} = 1.18605 + 2(.0163) = 1.21865$ in.

$$(y_1+x_2)/(a+x_1) = (\sin 45^o/\sin 39^o) = 1.1236038 \tag{1}$$

$y_1 + x_2 = .9775353$ $\qquad x_2/y_1 = 60/15 = 4$

$y_1 = .1955071$ in. $\qquad x_2 = .7820284$ in.

Operating P.D. $y_1 = (.1955071)(2) = .3910142$

Operating P.D. $x_2 = (.7820284)(2) = 1.5640568$

Selecting a cutter with $P = 38$

Cutting P.D. $y_1 = 15/38 = .3947368$ $\qquad x_2 = 60/38 = 1.5789474$

$$z = \frac{(y_1+x_2)\ \sin(90+\alpha)}{\sin \theta} = 1.3748705 \text{ in.} \tag{2}$$

Selecting N_c = 164 teeth (divisible by 4)

Cutting P.D. $c = 164/38 = 4.3157895$

$c/x_2 = 164/60 = 2.733333$ $\qquad c - x_2 = 1.3748705$

$x_2 = .7931947$ in. $\qquad c = 2.1680652$ in.

Operating P.D. x_2 with c gear $= (.7931947)(2) = 1.586389$

c with x_2 gear $= (2.1680652)(2) = 4.3361304$

Ratio $= (51/24)(164/15) = 23.233$

Gear Stresses

Input torque = 230/23.233 = 9.9 in.lbs. at 2,788 rpm

ax_1 contacts (4 contacts)

$F_T = (9.90/4)(1/.2784) = 8.89$ lb. (2 N)

Bending Stress

$$S_B = \frac{F_T P}{fY} \qquad f = .25 \text{ in. (face width)} \tag{3}$$

$Y = .572$

Steel:

$$S_B = \frac{(8.89)(43)}{(.25)(.35)} = 4{,}368 \text{ psi } (12.54\ N/m^2)$$

Plastics:

$$S_B = \frac{(8.89)(43)}{(.25)(.572)} = 2,673 \text{ psi} \quad (7.67\ N/m^2)$$

(4,000 to 6,000 psi is OK for 10^7 cycles for Delrin or Zytel nylon)

$$\text{Pitch line velocity} = \frac{(\pi)(.5568)(2788)}{12} = 406 \text{ fpm} \quad (2.06 \text{ m/s})$$

$$m_G = 51/24 = 2.125$$

$$\frac{m_G + 1}{m_G} = 1.47$$

Compressive Stress

$$S_c = 5715\sqrt{\left(\frac{F_T}{D\ f}\right)\left(\frac{m_G + 1}{m_G}\right)} = 55,400 \text{ psi} \quad (159\ N/m^2) \qquad (4)$$

OK for R_c 30-38 steel

y_1x_2 contacts (4 gears, 2 contacts per gear)

Torque = (9.90)(51/24)/4 = 5.26 in.lb. per gear

$$F_T = (5.26)/(.1955)(2) = 13.45 \text{ lb} \quad (59.85\ N)$$

f = .375 for y_1 f = .343 for x_2 f = .375 for c

Steel:

$$S_B = \frac{(13.45)(38)}{(.375)(.28)} = 4,868 \text{ psi} \quad (13.98\ N/m^2)$$

OK for R_c 30-38 steel

Plastics:

$$S_B = \frac{(13.45)(38)}{(.375)(.48)} = 2,840 \text{ psi} \quad (8.15\ N/m^2)$$

OK for Delrin or Zytel

$$\text{Pitch line velocity} = \frac{(2788)(.2784/.5916)(\pi)(.391)}{12} = 134.3 \text{ fpm} \quad (.68 \text{ m/s})$$

$$m_G = 60/15 = 4$$

$$\frac{m_G + 1}{m_G} = 1.25$$

Compressive Stress

$$S_c = 5715\sqrt{\frac{(13.45)(1.25)}{(.343)(.391)}} = 64,000 \text{ psi} \quad (183.7\ N/m^2)$$

x_2c contacts (4 contacts)

Torque = 230 in.lbs.

$$F_T = (230)/(2.168)(4) = 26.52 \text{ lb} \quad (118 \text{ N})$$

Steel:

$$S_B = \frac{(26.52)(38)}{(.343)(.384)} = 7{,}651 \text{ psi} \quad (21.97 \text{ N/m}^2)$$

OK for R_c 30-38 steel

Plastics:

$$S_B = \frac{(26.52)(38)}{(.343)(.713)} = 4{,}120 \text{ psi} \quad (11.83 \text{ N/m}^2)$$

$$\text{Pitch line velocity} = \frac{(120)(4.336)(\pi)}{12} = 136.2 \text{ fpm} \quad (.69 \text{ m/s})$$

$$m_G = 164/60 = 2.733$$

$$\frac{m_G + 1}{m_G} = .634$$

$$S_c = 5715\sqrt{\frac{(26.52)(.634)}{(.343)(1.586)}} = 31{,}800 \text{ psi} \quad (91.2 \text{ N/m}^2)$$

OK for R_c 30-38 steel

Gear Geometry and Measurement Data

$a - x_1$ mesh ($P = 43$; $\phi = 20^\circ$

CD , center distance = .870 in..

Operating P.D. a = .5568 x_1 = 1.1832

Cutting P.D. a = 24/43 = .5581 x_1 = 51/43 = 1.1860

Operating pressure angle ϕ :

$$\text{Cutting P.D.} \times \cos\phi = \text{Operating P.D.} \times \cos\phi_1 \tag{5}$$

$$.5581 \cos 20^\circ = .5568 \cos\phi_1$$

$$\phi_1 = 19.61778^\circ$$

$$BC, \text{ Base circle dia.} = \text{Cutting P.D.} \times \cos\phi \qquad (6)$$

Base circle dia. a = .5245 x_1 = 1.1145

Operating pinion addendum = 1.20/43 = .0279

Operating gear addendum = .80/43 = .0186

O.D. Pinion a = .5568 + 2(.0279) = .6126 in.

O.D. Gear x_1 = 1.1832 + 2(.0186) = 1.2204 in.

Cutting Standard addendum = 1/43 = .0233

Cutting circular pitch = π/43 = .0730

Nominal Tooth thickness = .073/2 = .0365 in. (.0363 with min. backlash)

Pinion a addendum with respect to the cutting P.D.:

addendum = (.5)(.6126 - .5581) = .02725 (addendum increases)

Pinion a tooth thickness at cutting P.D.:

$$T_1 = .0363 + (.02725 - .0233)(2 \tan 20^\circ) = .0392 \text{ in.}$$

Gear x_1 addendum with respect to the cutting P.D.:

addendum = (.5)(1.2204 - 1.1860) = .0172 (addendum decreases)

Gear x_1 tooth thickness at cutting P.D.:

$$T_1 = .0363 - (.0233 - .0172)(2 \tan 20^\circ) = .0319 \text{ in.}$$

Pinion a tooth thickness, T_2, at operating P.D.:

$$T_2 = 2r_2\left[\frac{T_1}{2r_1} + \text{inv}\,\phi - \text{inv}\,\phi_1\right]^{(1)} \qquad (7)$$

$2r_1$ = cutting P.D. = .5581 $2r_2$ = operating P.D. = .5568

$$\text{inv}\,\phi = \tan\phi - \phi$$

inv ϕ = .01490438 inv ϕ_1 = .01403884

$$T_2 = .0396 \text{ in.}$$

(1) Ref. 6 p 25, Sect. 2

Gear x_1 tooth thickness, T_2, at operating P.D.:

T_1 = .0319 $\quad$ $2r_1$ = 1.1860

$2r_2$ = 1.1832 $\quad$ T_2 = .0328 in. (eq. 7)

Operating circular pitch = $(\pi)(.5568)/(2)(12)$ = .072885

Minimum backlash = .072885 - (.0396 + .0328) = .0005 in.

Recommended tooth thicknesses:

Pinion a = .0392/.0387 at cutting P.D.

Gear x_1 = .0319/.0314 at cutting P.D.

Expected backlash .0005/.0015

Pinion a

Maximum Root Dia. = .6126 - 4.32/43 = .5121

Minimum Root Dia. = .6126 - 4.60/43 = .5056

Gear x_1

Maximum Root Dia. = 1.2204 - 4.32/43 = 1.1199

Minimum Root Dia. = 1.2204 - 4.60/43 = 1.1134

$$\text{Pinion a Form Dia} = \sqrt{(BC)_P^{\,2} + \left[2CD\sin\phi_1 - \left\{(OD)_G^{\,2} - (BC)_G^{\,2}\right\}^{\frac{1}{2}}\right]^2} \qquad (8)$$

$(BC)_P$ = .5245 $\quad$ CD = .870

$(OD)_P$ = .6126 $\quad$ $(OD)_G$ = 1.220

$(BC)_G$ = 1.145 $\quad$ ϕ_1 = 19.61778°

Pinion a Form Dia = .5318

$$\text{Gear } x_1 \text{ Form Dia} = \sqrt{(BC)_G^{\,2} + \left[2CD\sin\phi - \left\{(OD)_P^{\,2} - (BC)_P\right\}^{\frac{1}{2}}\right]^2} \qquad (9)$$

Gear x_1 Form Dia = 1.1462

$$\text{Contact Ratio} = \frac{P_P + P_G - CD\sin\phi_1}{BP} \tag{10}$$

where:

$$BP\text{, Base Pitch} = \frac{(\pi)(.5245)}{24} = .0687$$

$$P_P = \sqrt{(OD)_P^2 - (BC)_P^2}\Big/2 \tag{11}$$

$$P_G = \sqrt{(OD)_G^2 - (BC)_G^2}\Big/2 \tag{12}$$

$$\text{Contact Ratio} = 1.66$$

Minimum clearance for x_1 gear $= 2(.870)(\sin 45) - 1.220 = .0104$ in.

M , Measurement over pins:

Pinion a

$$M_1 = \frac{2r_1\cos\phi_1}{\cos\phi_2} + 2x \tag{13}$$

$$\text{inv}\,\phi_2 = \frac{T_1}{2r_1} + \text{inv}\,\phi + \frac{x}{r_1\cos\phi} - \frac{\pi}{N} \tag{14}$$

ϕ_2 = Pressure angle at center of roll ; $\phi = 20^\circ$

inv ϕ = .01490438 ; 2x = pin diameter

For $T_1 = .0392$; $2r_1 = .5581$; $N = 24$; $2x = .040$

$$\text{inv}\,\phi_2 = .030504$$

$$\phi_2 = 25.13842^\circ$$

$$M_1 = .61935$$

For $T_1 = .0387$; inv $\phi_2 = .02961$; $\phi_2 = 24.90283^\circ$

$$M_1 = .61824$$

Pin measurement .6194/.6182 with .040 diameter pins

Gear x_1

$$M_2 = 2r_3\cos\frac{90}{N} + 2x \qquad (15)$$

r_3 = radius to center of roll

$$r_3 = \frac{r_1 \cos \phi}{\cos \phi_2} \qquad (16)$$

For T_1 = .0319 ; $2r_1$ = 1.1860 ; N = 51 ; 2x = .040

$\text{inv}\,\phi_2 = .01609$

$\phi_2 = 20.4990^\circ$

$M_2 = 1.2293$

For T_1 = .0314

$\text{inv}\,\phi_2 = .01567$

$\phi_2 = 20.32459^\circ$

$M_2 = 1.2279$

Pin measurement 1.2293/1.2279 with .040 diameter pins

<u>y_1x_2 mesh</u> (P = 38 ; $\phi = 20^\circ$)

CD = .9775

Operating P.D. y_1 = .3910 ; x_2 = 1.5640

Cutting P.D. y_1 = 15/38 = .3947 ; x_2 = 60/38 = 1.5789

Operating pressure angle ϕ_1 (eq. 5):

$\phi_1 = 18.45278^\circ$

Base circle dia. y_1 = .3709 (eq. 6) ; x_2 = 1.4837 (eq. 6)

Operating Pinion addendum = 1.22/38 = .0321

Operating Gear addendum = .78/38 = .0205

O.D. Pinion y_1 = .3910 + 2(.0321) = .4552

O.D. Gear x_2 = 1.5640 + 2(.0205) = 1.6050

Cutting standard addendum = 1/38 = .0263

Cutting circular pitch = π /38 = .0827

Nominal tooth thickness = .0827/2 = .0413 (with min. backlash allowance of .0002, use .0411)

Pinion y_1 addendum with respect to the cutting P.D.:

addendum = .5(.4552 - .3947) = .03025 (addendum increases)

Pinion y_1 tooth thickness at cutting P.D.:

$$T_1 = .0411 + (.03025 - .0263)(2\tan 20^\circ) = .0440 \text{ in.}$$

Gear x_2 addendum with respect to the cutting P.D.:

addendum = .5(1.605 - 1.5789) = .01305 (addendum decreases)

Gear x_2 tooth thickness at cutting P.D.:

$$T_1 = .0411 - (.0263 - .01305)(2\tan 20^\circ) = .0315 \text{ in.}$$

Pinion y_1 tooth thickness, T_2, at operating P.D.:

$2r_1 = .3947$ $\phi = 20^\circ$

$\text{inv}\,\phi = .014904$ $\phi_1 = 18.45278^\circ$

$\text{inv}\,\phi_1 = .011617$ $2r_2 = .3910$

$T_1 = .0440$

$T_2 = .0449$ in. (eq. 7)

Gear x_2 tooth thickness, T_2, at operating P.D.:

$2r_1 = 1.5789$ $2r_2 = 1.564$

$T_1 = .0315$

$T_2 = .0363$ in (eq. 7)

Operating circular pitch = (π)(.3910)/15 = .0819

Minimum backlash = .0819 - (.0449 + .0363) = .0007 in.

Recommended tooth thicknesses:

Pinion y_1 = .0440/.0435 at cutting P.D.

Gear x_2 = .0315/.0310 at cutting P.D.

Expected backlash .0007 to .0017

Pinion y_1

Maximum Root Dia. = .4552 - 4.32/38 = .3415

Minimum Root Dia. = .4552 - 4.60/38 = .3341

Gear x_2

Maximum Root Dia. = 1.605 - 4.32/38 = 1.4913

Minimum Root Dia. = 1.605 - 4.60/38 = 1.4839

Pinion y_1 Form Dia (eq. 8):

$(BC)_P = .3709$ $\quad$ $CD = .9775$

$(OD)_G = 1.605$ $\quad$ $(OD)_P = .4552$

$(BC)_G = 1.4837$ $\quad$ $\phi_1 = 18.45278^\circ$

Pinion y_1 Form Dia = .371 in.

Gear x_1 Form Dia (eq. 9): Gear x_1 Form Dia = 1.5255 in.

Contact Ratio (eq. 10):

$P_P = .1319$ (eq. 11) ; $P_G = .3061$ (eq. 12)

Contact Ratio = 1.66

M , Measurement over pins:

Pinion y_1

For $T_1 = .0440$; $2r_1 = .3947$; $\phi = 20^\circ$

$\text{inv}\,\phi = .0149$; $N = 15$; $2x = .050$

$\text{inv}\,\phi_2 = .05174$ (eq. 14) ; $\phi_2 = 29.6493^\circ$

$r_3 = .2134$ (eq. 16) ; $M_3 = .4744$ (eq. 15)

For $T_1 = .0435$; $\text{inv}\,\phi_2 = .0505$ (eq. 14)

$\phi_2 = 29.4232^\circ$; $r_3 = .2129$ (eq. 16)

$M_3 = .4735$ (eq. 15)

Pin measurement .4744/.4735 with .050 diameter pins

Gear x_2

For $T_1 = .0315$; $2r_1 = 1.5789$; $\phi = 20^\circ$

$\text{inv}\,\phi = .0149$; $N = 60$; $2x = .050$

$\text{inv}\,\phi_2 = .0162$ (eq. 14) ; $\phi_2 = 20.5418^\circ$

$M_4 = 1.6344$ (eq. 13)

For $T_1 = .0310$; $\text{inv}\,\phi_2 = .0159$ (eq. 14)

$\phi_2 = 20.4117^\circ$; $M_4 = 1.6331$ (eq. 13)

Pin measurement 1.6344/1.6331 with .050 diameter pins

<u>x_2c mesh</u> ($P = 38$; $\phi = 20$)

$CD = 1.3749$

Operating P.D. $x_2 = 1.5864$; $c = 4.3361$

Cutting P.D. $x_2 = 60/38 = 1.5789$; $c = 164/38 = 4.3158$

Operating pressure angle ϕ_1 (eq. 5)

$\phi_1 = 20.73142^\circ$

Base circle dia. $x_2 = 1.4837$ (previously derived) ; $c = 4.0555$ (eq. 6)

Pinion x_2 tooth thickness, T_2, at operating P.D. with gear c :

$2r_1 = 1.5789$; $\phi = 20^\circ$; $\text{inv}\,\phi = .014904$

$\phi_1 = 20.73142^\circ$; $\text{inv}\,\phi_1 = .01666$; $2r_2 = 1.5864$

$T_2 = .0289$ in. (eq. 7)

In order to have a backlash of .0005/.002 between pinion x_2 and ring gear c the space width of gear c at its P.D. must be .0294 min. to .0304 max.

Operating circular pitch of gear c $= (\pi)(4.3361)/164 = .0831$

Gear c tooth thickness, T_1, at operating P.D. $= .0831 - .0294 = .0537$ max.

$= .0831 - .0304 = .0527$ min.

Gear c tooth thickness, T_2, at cutting P.D. of 4.3158 :

$$T_2 = 2r_2\left[T_1/2r_1 - \text{inv}\phi_1 + \text{inv}\ \phi\right] \tag{17}$$

$2r_1 = 4.3361$; $2r_2 = 4.3158$; $\phi_1 = 20.73142^\circ$

$\text{inv}\ \phi_1 = .01666$; $\phi = 20^\circ$; $\text{inv}\ \phi = .01490$

when $T_1 = .0537$ $T_2 = .0459$

when $T_1 = .0527$ $T_2 = .0449$

Gear c tooth thickness at cutting P.D. = .0459/.0449

Expected backlash with x_2 gear = .0005/.002

Gear c circular pitch at cutting P.D. = $(\pi)(4.3158)/164 = .08267$

Space widths at cutting P.D. = .0827 - .0459 = .0368 min.

= .0827 - .0449 = .0378 max.

Gear c major dia. = Operating P.D. + (2)(gear x_2 Operating addendum) + (2)(Clearance)

Minimum clearance = .150/38 = .00395

Maximum clearance = .250/38 = .0066

Operating P.D. = 4.3361

Pinion x_2 operating addendum = (1.605 - 1.5864)/2 = .0093

Gear c major dia. = 4.3679/4.3626

Gear c tooth thickness, T_3, at major dia.:

$$T_3 = 2r_3\left[T_2/2r_2 - \text{inv}\ \phi + \text{inv}\ \phi_3\right] \tag{18}$$

$2r_2 = 4.3158$; $2r_3 = 4.3679$; $\phi = 20^\circ$

$\text{inv}\ \phi = .01490$; $\cos\phi_3 = (2r_1)(\cos\phi)/2r_3 = .928484$

$\phi_3 = 21.8003^\circ$; $\text{inv}\ \phi_3 = .01949$; $T_2 = .0459$

$T_3 = .06649$

Gear c circular pitch at major dia = $(\pi)(4.3679)/164$ = .08367

Circular space at major dia = .08367 - .06649 = .017 min.

This looks acceptable for this 38P cutter design.

Calculation of gear c inside diameter for a minimum contact ratio of 1.3

$$\text{Contact Ratio} = (P_P + CD\sin\phi_1 - P_G)/BP \qquad (19)$$

$(OD)_P = 1.605$; $(BC)_P = .74184$; $P_P = .3061$ (eq. 11)

$CD\sin\phi_1 = .4867$; $BP = (\pi)(4.0555)/164 = .0777$

$$P_G = \sqrt{(ID)_G^{\,2} - (BC)_G^{\,2}}\Big/2 \qquad (20)$$

$(ID)_G$ to be solved for.

Eq. 19:

$$1.30 = (.3061 + .4867 - P_G)/.0777$$

$$P_G = .6918 \quad ; \quad (BC)_G = 4.0555$$

Eq. 20:

$$(ID)_G = 4.285 \quad ; \quad \text{Use } (ID)_G = 4.285/4.284$$

Calculation for SAP (Start of Active Profile, i.e., True Involute Form)

$$\text{SAP pinion } x_2 = \left\{ (BC)_P^{\,2} + \left[((ID)_G^{\,2} - (BC)_G^{\,2})^{\frac{1}{2}} - 2CD\sin\phi_1 \right]^2 \right\}^{\frac{1}{2}} \qquad (21)$$

$(BC)_P$ (pinion x_2 base dia.) = 1.4837

$(ID)_G$ (gear c min. internal dia.) = 4.284

$(BC)_G$ (gear c base dia.) = 4.0555

$2CD\sin\phi_1 = (2)(.4867) = .9734$

SAP pinion x_2 = 1.5385

$$\text{SAP gear c} = \left\{ (BC)_G^{\,2} + \left[2CD\sin\phi_1 + ((OD)_P^{\,2} - (BC)_P^{\,2})^{\frac{1}{2}} \right]^2 \right\}^2 \qquad (22)$$

$(OD)_P$ = 1.605

SAP gear c = 4.354

Tip interference check:

$$CD = 1.3749$$

Gear c inside radius, $(IR)_G = 4.284/2 = 2.142$

Gear c base radius, $(BR)_G = 4.0555/2 = 2.0278$

Pinion x_2 outside radius, $(OR)_P = 1.605/2 = .8025$

Pinion x_2 base radius, $(BR)_P = 1.4837/2 = .7149$

Pressure angle at pinion x_2 OD $= \phi_4$

Pressure angle at gear c ID $= \phi_5$

Operating pressure angle, $\phi_1 = 20.7314^{\circ}$

$\cos\phi_4 = \dfrac{(BR)_P}{(OR)_P} = .9244$ $\qquad\qquad$ $\cos\phi_5 = \dfrac{(BR)_G}{(IR)_G} = .9467$

$\phi_4 = 22.42^{\circ}$ $\qquad\qquad$ $\phi_5 = 18.80^{\circ}$

$\text{inv}\,\phi_4 = .02128$ $\qquad\qquad$ $\text{inv}\,\phi_5 = .01231$

$\sin\phi_1 = .3540$ $\qquad\qquad$ $\cos\phi_1 = .9353$

$$\phi_1 = .3618 \text{ rad.}$$

$$\phi_6 = \text{arc cos}\left[\frac{(IR)_G^2 - (OR)_P^2 - CD^2}{2CD(OR)_P}\right] = \text{arc cos } .9307 \qquad (23)$$

$\phi_6 = 21.45^{\circ}$ $\qquad\qquad$ $\phi_6 = .3744$ rad.

$$\phi_7 = \text{arc cos}\left[\frac{(IR)_G^2 + CD^2 - (OR)_P^2}{2CD(IR)_G}\right] = \text{arc cos } .9906 \qquad (24)$$

$\phi_7 = 7.876^{\circ}$ $\qquad\qquad$ $\phi_7 = .1375$ rad.

For no tip interference

$$X > Y$$

Where

$$X = CD(\sin\phi_1 - \phi_1\cos\phi_1) + (BR)_P(\phi_6 + \text{inv}\phi_4) \qquad (25)$$

$$Y = (BR)_G\left[\phi_7 + \text{inv}\phi_2\right] \qquad (26)$$

$X = .3149$

$Y = .3037$

Since $X > Y$ there will not be any tip interference.

M, Measurement between pins:

For $T_1 = .0459$; $2r_1 = 4.3158$

$\phi = 20^{o}$; $\text{inv}\ \phi = .0149$

$N = 164$; $2x = .050$

$$\text{inv}\ \phi_8 = \frac{\pi}{N} + \text{inv}\ \phi - \frac{T_1}{2r_1} - \frac{x}{r_1 \cos \phi} \tag{27}$$

$\text{inv}\ \phi_8 = .0111$; $\phi_8 = 18.1803^{o}$

$$M_5 = (2r_1 \cos \phi)/\cos \phi_8 - 2x \tag{28}$$

$M_5 = 4.2186$

For $T_1 = .0449$

$\text{inv}\ \phi_8 = .0113$; $\phi_8 = 18.3025^{o}$

$M_5 = 4.2216$

Measurement between pins 4.2186/4.2216 with .050 diameter pins.

APPENDIX B

HINGE-JOINT ROLLER-GEAR DRIVE GEAR DATA

Specifications for the Hinge Joint drive were changed from those used for the original design in the Phase I study (ref. 1):

Nominal ratio target	30:1 or higher
Input torque	28.9 Nm (256 in.lb.)
Output speed	120 rpm
Maximum diameter target	12 in. (approx.)

Cluster Geometry

A cluster geometry with a somewhat greater toggle angle than that used in ref. 1 was chosen in an effort to meet the diameter target of 12 inches. The solution chosen from Table I of ref. 1 is

$\alpha = 18^\circ$ $\gamma = 27^\circ$

Using equations 1 through 12 and 15 of ref. 1:

$(2/5)\ N_{y1}$ = integral number $N_{y1} = 20$ teeth

$(3/20)\ N_{x2}$ = integral number $N_{x2} = 100$ teeth

Selecting

$a + x_1 = 1.390$ $N_a = 28$ teeth (divisible by 4)

$N_{x1} = 62$ teeth $x_1/a = 62/28 = 2.214286$

$x_1 = 2.214286a$ $a = 1.390/3.214286 = .432445$

$x_1 = .957556$

Operating P.D. $a = .8649$ $x_1 = 1.9151$

Selecting a cutter with P = 32 (20° pressure angle)

Cutting P.D. $a = 28/32 = .8649$ $x_1 = 62/32 = 1.9375$

Max. permissable x_1 OD = (1.390)(sin 45)(2) = 1.9658

Standard addendum = 1/32 = .03125

Use short addendum = .6/32 = .01875

x_1 OD = 1.9151 + 2(.01875) = 1.9526

Clearance = 1.9658 - 1.9526 = .0132 OK

From eq. (1): (All equation nos. taken from ref. 2)

$$(y_1+x_2)/(a+x_1) = (\sin 45^o/\sin 27^o) = 1.55754$$

$y_1 + x_2 = 2.164976$ $\qquad$ $x_2/y_1 = 100/20 = 5$

$y_1 = .360829$ $\qquad$ $x_2 = 1.804147$

Operating P.D. $y_1 = (.3608293)(2) = .7216586$

Operating P.D. $x_2 = (1.8041465)(2) = 3.608293$

Selecting a cutter with $P = 28$ (20^o pressure angle)

Cutting P.D. $y_1 = 20/28 = .7142857$ $\qquad$ $x_2 = 100/28 = 3.5714286$

From eq. (2):

$$z = \frac{(y_1 + x_2)\sin(90 + \alpha)}{\sin \theta} = 2.911886$$

Selecting $N_c = 264$ (divisible by 4)

Cutting P.D. $c = 264/28 = 9.428571$

$c/x_2 = 264/100 = 2.64$ $\qquad$ $c - x_2 = 2.911886$

$x_2 = 1.775540$ $\qquad$ $c = 4.687426$

Operating P.D. x_2 with c gear = 3.551080

Operating P.D. c with x_2 gear = 9.374852

Ratio = (62/28)(264/20) = 29.23

Gear Stresses (Methods of ref. 3)

Input torque = 256 in.lbs.

Output torque = (256)(29.23) = 7483 in.lbs.

ax_1 contacts (4 contacts)

$F_T = (256/4)(1/.4324) = 148$ lbs.

Selecting .312 in. face width, $P = 32$, $y = .344$

Bending Stress: eq. (3):

$$S_B = (148)(32)/(.312)(.344) = 44{,}100 \text{ psi}$$

$$m_G = 62/28 = 2.214 \qquad \frac{m_G + 1}{m_G} = 1.45$$

Compressive Stress, eq. (4):

$$S_c = 5715 \sqrt{\frac{(148)(1.45)}{(2)(.4324)(.312)}} = 162{,}000 \text{ psi}$$

<u>y_1x_2 contacts</u> (8 contacts)

Torque = (256)(2.214)(1/4) = 141.7 in. lbs.

$$F_T = (141.7/2)(1/.361) = 196.3 \text{ lbs.}$$

Selecting .56 in. face width, $P = 28$, $y = .33$

$$S_B = (196.3)(28)/(.56)(.33) = 29{,}700 \text{ psi}$$

$$m_G = 100/20 = 5 \qquad \frac{m_G + 1}{m_G} = 1.2$$

$$K = (196.3)(1.2)/(2)(.361)(.56) = 583$$

$$S_c = 5715\sqrt{583} = 138{,}000 \text{ psi}$$

<u>x_2c contacts</u> (4 contacts)

Torque = 7483 in. lbs.

$$F_T = (7483/4)(1/4.687) = 399 \text{ lbs.}$$

Selecting .56 in. face width (gear x_2) , $P = 28$, $y = .38$

$$S_B = (399)(28)/(.56)(.38) = 52{,}500 \text{ psi in } x_2$$

Selecting .60 in. face width (gear c) , $P = 28$, $y = .38$

$$S_B = (399)(28)/(.60)(.38) = 49{,}000 \text{ psi in } c$$

$$m_G = 264/100 = 2.64 \qquad \frac{m_G - 1}{m_G} = .62$$

$$K = (399)(.62)/(2)(1.7755)(.56) = 124$$

$$S_c = 5715\sqrt{124} = 63{,}800 \text{ psi}$$

With these levels of S_B and S_c , a , x_1 , y_1 and x_2 gears should be case carburized 8620 steel or equivalent, case hardened to R_c 58-61, or nitralloy 135M nitrided to a case hardness of R_c 60-63. Gear C need not have a hardened face.

Gear Geometry and Measurement Data

$a - x_1$ mesh $(P = 32 \; ; \; \phi = 20^\circ)$

CD, center distance = 1.390 in.

Operating P.D. $a = .8649$; $x_1 = 1.9151$

Cutting P.D. $a = 28/32 = .875$; $x_1 = 62/32 = 1.9375$

Operating pressure angle ϕ :

From eq. (5)

$$\text{Cutting P.D.} \times \cos \phi = \text{Operating P.D.} \times \cos \phi_1$$

$$.875 \cos 20^\circ = .8649 \cos \phi_1$$

$$\phi_1 = 18.0723^\circ$$

From eq. (6)

$$\text{BC, Base circle dia.} = \text{Cutting P.D.} \times \cos \phi$$

Base circle dia. a = .8222 ; x_1 = 1.8207

Operating pinion addendum = 1.30/32 = .0406

Operating gear addendum = .70/32 = .0219

O.D. Pinion a = .8649 + 2(.0406) = .9461 in. Use .9461/.9455

O.D. Gear x_1 = 1.9151 + 2(.0219) = 1.9589 in.

O.D. clearance = (1.390)(sin 45°)(2) - 1.9589 = .0069 (Too close)

Stub O.D. to 1.9529

Then O.D. clearance = 1.9658 - 1.9529 = .0129 OK

Make O.D. 1.9259/1.9519

Cutting Standard addendum = 1/32 = .0313

Cutting circular pitch = π/32 = .0982

Nominal Tooth thickness = .0982/2 = .0491 in. (.0488 with .0003 backlash)

Pinion a addendum with respect to the cutting P.D.:

addendum = (.5)(.9461 - .875) = .0356 (addendum increases)

Pinion a tooth thickness at cutting P.D.:

$$T_1 = .0488 + (.0356 - .0313)(2 \tan 20^\circ) = .0519 \text{ in.}$$

Gear x_1 addendum with respect to the cutting P.D.:

addendum = (.5)(1.9589 - 1.9375) = .0107 (addendum decreases)

Gear x_1 tooth thickness at cutting P.D.:

$$T_1 = .0488 - (.0313 - .0107)(2 \tan 20^\circ) = .0338 \text{ in.}$$

Pinion a tooth thickness, T_2, at operating P.D. (eq. (7)):

$$T_2 = \left[2r_2 \; \frac{T_1}{2r_1} + \text{inv } \phi - \text{inv } \phi_1 \right]^{(1)}$$

$2r_1$ = cutting P.D. = .875 ; $2r_2$ = operating P.D. = .8649

$$\text{inv } \phi \equiv \tan \phi - \phi$$

inv ϕ = .01490438 inv ϕ_1 = .01089422

$$T_2 = .0548 \text{ in.}$$

(1) Ref.4, p. 25, Sect. 2

Gear x_1 tooth thickness, T_2 , at operating P.D.

T_1 = .0338 $2r_1$ = 1.9375

$2r_2$ = 1.9151 T_2 = .0411 in. (eq. 7)

Operating circular pitch = $(\pi)(.8649)/(28)$ = .09704

Minimum backlash = .09704 - (.0548 + .0411) = .0011 in.

Recommended tooth thicknesses:

Pinion a = .0519/.0514 at cutting P.D.

Gear x_1 = .0338/.0333 at cutting P.D.

Expected backlash .001/.002

Pinion a

Maximum Root Dia. = .9461 - 4.32/32 = .8111

Minimum Root Dia. = .9461 - 4.60/32 = .8023

Gear x_1

Maximum Root Dia. = 1.9589 - 4.32/32 = 1.8239

Minimum Root Dia. = 1.9589 - 4.60/32 = 1.8151

From eq. (8):

$$\text{Pinion a Form Dia} = \sqrt{(BC)_P^2 + \left[2CD\sin\phi_1 - \left\{(OD)_G^2 - (BC)_G^2\right\}^{\frac{1}{2}}\right]^2}$$

$(BC)_P$ = .8222 CD = 1.390

$(OD)_P$ = .9461 $(OD)_G$ = 1.9259

$(BC)_G$ = 1.8207 ϕ_1 = 18.0723°

Pinion a Form Dia = .8369

From eq. (9):

$$\text{Gear } x_1 \text{ Form Dia} = \sqrt{(BC)_G^2 + \left[2CD\sin\phi - \left\{(OD)_P^2 - (BC)_P\right\}^{\frac{1}{2}}\right]^2}$$

Gear x_1 Form Dia = 1.8629

From eq. (10):

$$\text{Contact Ratio} = \frac{P_P + P_G - CD\sin\phi_1}{BP}$$

where:

$$BP, \text{ Base Pitch} = \frac{(\pi)(.8222)}{28} = .0923$$

From eq. (11):

$$P_P = \sqrt{(OD)_P^2 - (BC)_P^2}\ /2$$

From eq. (12):

$$P_G = \sqrt{(OD)_G^2 - (BC)_G^2}\ /2$$

Contact Ratio = 1.69

M , Measurement over pins:

Pinion a (eq. (13)):

$$M_1 = \frac{2r_1\cos\phi}{\cos\phi_2} + 2x$$

From eq. (14):

$$\text{inv } \phi_2 = T_1/2r_1 + \text{inv } \phi + x/r_1\cos\phi - \pi/N$$

ϕ_2 = Pressure angle at center of roll ; $\phi = 20^\circ$

inv ϕ = .01490438 ; 2x = pin diameter

For T_1 = .0519 ; $2r_1$ = .875 ; N = 28 ; 2x = .060

inv ϕ_2 = .03499 ; $\phi_2 = 26.2487^\circ$

M_1 = .9768

For T_1 = .0514 ; inv ϕ_2 = .03442 ; $\phi_2 = 26.1133^\circ$

M_1 = .9757

Pin measurement .9768/.9757 with .060 diameter pins

Gear x_1 (eq. (13)): $M_2 = \dfrac{2r_1\cos\phi}{\cos\phi_2} + 2x$

For T_1 = .0338 ; $2r_1$ = 1.9375 ; N = 62 ; 2x = .60

inv ϕ_2 = .01463 ; $\phi_2 = 19.8823^\circ$

M_2 = 1.9961

For $T_1 = .0333$

$\text{inv } \phi_2 = .01438$; $\phi_2 = 19.7685^\circ$

$M_2 = 1.9947$

Pin measurement 1.9961/1.9947 with .060 diameter pins

<u>y_1x_2 mesh</u> ($P = 28$; $\phi = 20^\circ$)

$CD = 2.1650$

Operating P.D. $y_1 = .7217$; $x_2 = 3.6083$

Cutting P.D. $y_1 = 20/28 = .7143$; $x_2 = 100/28 = 3.5714$

Operating pressure angle ϕ_1 (eq. 5):

$\phi_1 = 21.5562^\circ$

Base circle dia. $y_1 = .6712$ (eq. 6) ; $x_2 = 3.3560$ (eq. 6)

Operating Pinion addendum = 1.30/28 = .0464

Operating Gear addendum = .70/28 = .0250

O.D. Pinion $y_1 = .7217 + 2(.0464) = .8145$ use .8145/.8135

O.D. Gear $x_2 = 3.6083 + 2(.0250) = 3.6583$ use 3.6583/3.6573

Cutting standard addendum = 1/28 = .0357

Cutting circular pitch = $\pi/28 = .1122$

Nominal tooth thickness = .1122/2 = .0561 (with min. backlash allowance of .0003, use .0558)

Pinion y_1 addendum with respect to the cutting P.D.:

addendum = .5(.8145 - .7143) = .0501 (addendum increases)

Addendum increase (hob pullout) = .0501 -.0357 = .0144

Pinion y_1 tooth thickness at cutting P.D.:

$T_1 = .0558 + (.0144)(2)(\tan 20^\circ) = .0663$ in.

Gear x_2 addendum with respect to the cutting P.D. :

addendum = .5(3.6583 - 3.5714) = .0435 (addendum decreases)

Addendum increase (hob pullout) = .0435 - .0357 = .0078

Gear x_2 tooth thickness at cutting P.D. :

$$T_1 = .0558 + (.0078)(2)(\tan 20^o) = .0614 \text{ in.}$$

Pinion y_1 tooth thickness, T_2 , at operating P.D.:

$2r_1 = .7143$ $\qquad\qquad$ $\emptyset = 20^o$

inv $\emptyset = .014904$ $\qquad\qquad$ $\emptyset_1 = 21.5562^o$

inv $\emptyset_1 = .018817$ $\qquad\qquad$ $2r_2 = .7217$

$T_1 = .0663$

$T_2 = .0642$ in. (eq. 7)

Gear x_2 tooth thickness, T_2 , at operating P.D.:

$2r_1 = 3.5714$ $\qquad\qquad$ $2r_2 = 3.6083$

$T_1 = .0614$

$T_2 = .0479$ in. (eq. 7)

Operating circular pitch = $(\pi)(.7217)/20 = .1134$

Minimum backlash = $.1134 - (.0642 + .0479) = .0013$ in. (OK - target was .001)

Recommended tooth thicknesses:

Pinion y_1 = .0663/.0658 at cutting P.D.

Gear x_2 = .0614/.0609 at cutting P.D.

Pinion y_1 tooth thickness, T_3 , at O.D. (for adequacy of land width for shrink fit of x_2 gear)

$2r_1 = .7143$ $\qquad\qquad$ $\emptyset_3$ (Pressure angle at O.D.) from eq. (5):

$2r_3 = .8145$ (O.D.) $\qquad\qquad$ $\cos \emptyset_3 = (.7143)(\cos 20^o)/.8145 = .8241$

$T_1 = .0663$ $\qquad\qquad$ $\emptyset_3 = 34.5035^o$

inv $\emptyset_3 = .08517$

$T_3 = .018$ (eq. 7)

This should be enough land to support the shrink fit.

Pinion y_1

Maximum Root Dia. = .8145 - 4.32/28 = .6602

Minimum Root Dia. = .8145 - 4.60/28 = .6502

Gear x_2

Maximum Root Dia. = 3.6583 - 4.32/28 = 3.5040

Minimum Root Dia. = 3.6583 - 4.60/28 = 3.4940

Pinion y_1 Form Dia. (eq. 8):

$(BC)_P = .6712$ $\qquad$ $CD = 2.1650$

$(OD)_G = 3.6583$ $\qquad$ $(OD)_P = .8145$

$(BC)_G = 3.3560$ $\qquad$ $\phi_1 = 21.5562^\circ$

Pinion y_1 Form Dia = .6846 in.

Gear x_1 Form Dia (eq. 9):

Gear x_1 Form Dia = 3.5410 in.

Contact Ratio (eq. 10):

$P_P = .2307$ (eq. 11) $\qquad$ $P_G = .7281$ (eq. 12)

Contact Ratio = 1.55

M, Measurement over pins:

Pinion y_1

For $T_1 = .0663$; $2r_1 = .7143$; $\phi = 20^\circ$; $\text{inv}\ \phi = .0149$

$N = 20$; $2x = .0686$; $\text{inv}\ \phi_2 = .0528$ (eq. 14) ; $\phi_2 = 29.8359^\circ$

$M_3 = .8423$ (eq. 13)

For $T_1 = .0658$; $\text{inv}\ \phi_2 = .0521$ (eq. 14) ; $\phi_2 = 29.7134^\circ$

$M_3 = .8414$ (eq. 13)

Pin measurement .8423/.8414 with .0686 diameter pins

Gear x_2

For $T_1 = .0614$; $2r_1 = 3.5714$; $\emptyset = 20^o$; inv $\emptyset = .0419$

$N = 100$; $2x = .0686$; inv $\emptyset_2 = .0211$ (eq. 14) ; $\emptyset_2 = 20.3649^o$

$M_4 = 3.6975$ (eq. 13)

For $T_1 = .0609$; inv $\emptyset_2 = .02097$ (eq. 14) ; $\emptyset_2 = 22.3174^o$

$M_4 = 3.6963$ (eq. 13)

Pin measurement 3.6975/3.6963 with .0686 diameter pins

<u>x_2c mesh</u> ($P = 28$; $\emptyset = 20$)

$CD = 2.9119$

Operating P.D. $x_2 = 3.5511$ $c = 9.3749$

Cutting P.D. $x_2 = 100/28 = 3.5714$; $c = 264/28 = 9.4286$

Operating pressure angle $\emptyset_1$ (eq. 5)

$\emptyset_1 = 19.0798^o$

Base circle dia. $x_2 = 3.3560$ (previously derived) ; $c = 8.8600$ (eq. 6)

Pinion x_2 tooth thickness, T_2 , at operating P.D. with gear c:

$T_1 = .0614$; $2r_1 = 3.5714$; $\emptyset = 20^o$; inv $\emptyset = .014904$

$\emptyset_1 = 19.0798^o$; inv $\emptyset_1 = .01288$; $2r_2 = 3.5511$

$T_2 = .0682$ in. (eq. 7)

In order to have a backlash of .001/.002 between pinion x_2 and ring gear c, the space width of gear c at its P.D. must be .0692 min. to .0702 max.

Operating circular pitch of gear c = $(\pi)(9.3749/264) = .1116$

Gear c tooth thickness, T_1 , at operating P.D. = .1116 - .0692 = .0424 max.

= .1116 - .0702 = .0414 min.

Gear c tooth thickness, T_2, at cutting P.D. of 9.4286 (eq. 17):

$$T_2 = 2r_2 \left[T_1/2r_1 - \text{inv } \phi_1 + \text{inv } \phi \right]$$

$2r_1 = 9.3749$; $2r_2 = 9.4286$; $\phi_1 = 19.0798^\circ$

inv $\phi_1 = .01288$; $\phi = 20^\circ$; inv $\phi = .01490$

when $T_1 = .0424$ $T_2 = .0617$

when $T_1 = .0414$ $T_2 = .0607$

Gear c tooth thickness at cutting P.D. = .0617/.0607

Expected backlash with x_2 gear = .001/.002

Gear c circular pitch at cutting P.D. = $(\pi)(9.4286)/264 = .1122$

Space widths at cutting P.D. = .1122 - .0617 = .0505 min.

= .1122 - .0607 = .0515 max.

Gear c major dia. = Operating P.D. + (2)(Gear x_2 Operating addendum) + (2)(Clearance)

Minimum clearance = .150/28 = .0054

Maximum clearance = .250/28 = .0089

Operating P.D. = 9.3749

Pinion x_2 operating addendum = (3.6583 - 3.5511)/2 = .0536

Gear c major dia. = 9.5000/9.4929

Gear c tooth thickness, T_3, at major dia. (eq. 18):

$$T_3 = 2r_3 \left[T_2/2r_2 - \text{inv } \phi + \text{inv } \phi_3 \right]$$

$2r_2 = 9.4286$; $2r_3 = 9.500$; $\phi = 20^\circ$

inv $\phi = .01490$; $\cos \phi_3 = (2r_2)(\cos \phi)/2r_3 = .9326$

$\phi_3 = 21.1514^\circ$; inv $\phi_3 = .01774$; $T_2 = .0617$

$T_3 = .0891$

Gear c circular pitch at major dia. = $(\pi)(9.500)/264 = .1130$

Circular space at major dia. = .1130 - .0891 = .0239

This looks acceptable for this 28P cutter design.

Calculation of gear c inside diameter for a minimum contact ratio of 1.48

Contact Ratio = $(P_P + CD\sin\emptyset_1 - P_G)/BP$ (eq. 19)

$\emptyset_1 = 19.0798^o$; $CD = 2.9119$; $(OD)_P = 3.6583$; $(BC)_P = 1.678$

$P_P = .7281$ (eq. 11) ; $CD\sin\emptyset_1 = .9519$; $BP = (\pi)(8.8600)/264 = .1054$

$$P_G = \sqrt{(ID)_G^{\,2} - (BC)_G^{\,2}}\ /\ 2 \qquad \text{(eq. 20)}$$

$(ID)_G$ to be solved for.

Eq. 19:

$$1.48 = (.7281 + .9519 - P_G)/.1054$$

$P_G = 1.5239$; $(BC)_G = 8.860$

Eq. 20:

$(ID)_G = 9.3696$; Use $(ID)_G = 9.370/9.372$

Calculation for SAP (Start of Active Profile, i.e., True Involute Form), eq. (21):

$$\text{SAP pinion } x_2 = \left\{(BC)_P^{\,2} + \left[\left((ID)_G^{\,2} - (BC)_G^{\,2}\right)^{\frac{1}{2}} - 2CD\sin\emptyset_1\right]^2\right\}^{\frac{1}{2}}$$

$(BC)_P$ (Pinion x_2 base dia.) = 3.3560

$(ID)_G$ (Gear c min. internal dia.) = 9.370

$(BC)_G$ (Gear c base dia.) = 8.860

$2CD\sin\emptyset_1 = 1.9037$

SAP pinion x_2 = 3.546 (eq. 21)

(Can use previously calculated T.I.F. dia. of 3.541)

$$\text{SAP gear c} = \left\{(BC)_G^{\,2} + \left[2CD\sin\emptyset_1 + \left((OD)_P^{\,2} - (BC)_P^{\,2}\right)^{\frac{1}{2}}\right]^2\right\}^{\frac{1}{2}} \qquad \text{(eq. 22)}$$

$(OD)_P = 3.6583$

SAP gear c = 9.476 (eq. 22)

Tip interference check: CD = 2.9119

Gear c inside radius, $(IR)_G = 9.370/2 = 4.685$

Gear c base radius, $(BR)_G = 8.860/2 = 4.430$

Pinion x_2 outside radius, $(OR)_P = 3.6583/2 = 1.8292$

Pinion x_2 base radius, $(BR)_P = 3.356/2 = 1.678$

Pressure angle at pinion x_2 OD = ϕ_4

Pressure angle at gear c ID = ϕ_5

Operating pressure angle, $\phi_1 = 19.0798^\circ$

$\cos\phi_4 = (BR)_P/(OR)_P = .9174$

$\phi_4 = 23.4560^\circ$

$\text{inv } \phi_4 = .0245$

$\sin\phi_1 = .3269$

$\cos\phi_5 = (BR)_G/(IR)_G = .9456$

$\phi_5 = 18.9908^\circ$

$\text{inv } \phi_5 = .0127$

$\cos\phi_1 = .9451$

$\phi_1 = .3330$ rad.

$$\phi_6 = \text{arc cos}\left[\frac{(IR)_G{}^2 - (OR)_P{}^2 - CD^2}{2CD(OR)_P}\right] = \text{arc cos } .9504 \quad \text{(eq. 23)}$$

$\phi_6 = 18.1207^\circ$ $\qquad$ $\phi_6 = .3163$ rad.

$$\phi_7 = \text{arc cos}\left[\frac{(IR)_G{}^2 + CD^2 - (OR)_P{}^2}{2CD(IR)_G}\right] = \text{arc cos } .9926 \quad \text{(eq. 24)}$$

$\phi_7 = 6.9746^\circ$ $\qquad$ $\phi_7 = .1217$ rad.

For no tip interference

$$X > Y$$

Where

$$X = CD(\sin\phi_1 - \phi_1\cos\phi_1) + (BR)_P(\phi_6 + \text{inv } \phi_4) \quad \text{(eq. 25)}$$

$$Y = (BR)_G\left[\phi_7 + \text{inv } \phi_2\right] \quad \text{(eq. 26)}$$

$X = .6073$

$Y = .5955$

Since $X > Y$ there will not be any tip interference.

M, Measurement between pins:

For $T_1 = .0617$; $2r_1 = 9.4286$; $\phi = 20^\circ$

$\text{inv } \phi = .0149$; $N = 264$; $2x = .0686$

$$\text{inv } \phi_8 = (\pi/N) + \text{inv } \phi - (T_1/2r_1) - x/(r_1\cos\phi) \qquad \text{(eq. 27)}$$

$$\text{inv } \phi_8 = .01252 \qquad \phi_8 = 18.9058^\circ$$

$$M_5 = (2r_1\cos\phi)/\cos\phi_8 - 2x \qquad \text{(eq. 28)}$$

$$M_5 = 9.2967$$

For $T_1 = .0607$; $\text{inv } \phi_8 = .01263$; $\phi_8 = 18.9574^\circ$

$$M_5 = 9.2996$$

Measurement between pins 9.2967/9.2996 with .0686 diameter pins.

Gear calculations are summarized in TABLE 5.

APPENDIX C

PITCH YAW JOINT ROLLER-GEAR DRIVE GEAR DATA

Specifications for the Pitch Yaw Joint drive were:

Nominal ratio	3.43:1
Output torque	186.5 Nm (1650 in. lb.)
Dual input torques	27.1 Nm (240 in. lb.)
Maximum size	no larger than ORNL LTM large P/Y Unit

Gear Stresses

Input torque to each bevel roller-gear = 240 in. lb.

Bevel pinion P.D. = 1.75 in.

Bevel gear P.D. = 6.00 in.

First reduction ratio = 6/1.75 = 3.4286

Second bevel reduction ratio = 1

Output torque, first stage bevel reduction = (240) (3.4286) = 823 in. lb.

Total output torque = (823)(2) = 1646 in. lb.

Select for bevel pinion P = 16, 20° pressure angle

Then

$$N_P = 28$$

$$N_G = 96$$

Select face width f = .31 in.

$$F_T = (240)(2)/1.75 = 274.3 \text{ lb.}$$

From standard gear design texts

Bending stress:

$$S_B = \frac{F_T K_o P\ K_s K_m}{K_v\ f\ J}$$

$$S_B = \frac{(274.3)(16)(.5)(1.1)}{(1)(.31)(.318)}$$

$$S_B = 24{,}500 \text{ psi} \qquad \text{OK}$$

$P = 16$
$K_o = 1.0$
$K_v = 1.0$
$K_m = 1.1$
$K_s = 0.5$
$J = 0.318$

Compressive stress:

$$S_c = C_p \sqrt{\frac{F_T C_o C_s C_m C_f}{C\ f\ d\ I}}$$

$C_o = C_v = C_s = C_f = 1.0$
$C_p = 2800$
$C_m = 1.1$
$d = 1.75$
$I = .104$

$$S_c = 2800 \sqrt{\frac{(274.3)(1.1)}{(.31)(1.75)(.104)}}$$

$$S_c = 204{,}800 \text{ psi}$$

Use AISI 9310 steel, case hardened to R_c 60-62.

For second stage

Bevel pinion P.D. = 3.25 in.

Bevel gear P.D. = 3.25 in.

Select P = 12 , 20^o pressure angle

Then

$N_P = 39$

$N_G = 39$

Select face width f = .50 in.

$$F_T = (823)(2)/3.25 = 506.5 \text{ lb.}$$

Bending stress:

$$S_B = \frac{(506.5)(12)(.54)(1.1)}{(.50)(.270)}$$

$$S_B = 26{,}740 \text{ psi}$$

$P = 12$
$K_o = 1$
$K_v = 1$
$K_m = 1.1$
$K_s = .54$
$J = .270$

Compressive stress:

$C_o = C_v = C_s = C_f = 1.0$
$C_p = 2800$
$C_m = 1.1$
$d = 3.25$
$I = .070$

$$S_c = 2800 \sqrt{\frac{(506.5)(1.1)}{(.50)(3.25)(.07)}}$$

$$S_c = 196{,}000 \text{ psi}$$

Use AISI 9310 steel, case hardened to R_c 60-62.

These stresses are acceptable for hardened steel gears. The loading is too high for soft metallic or non-metallic gears. This is in contrast to the low torque Wrist-Roll drive in which non-metallic gears could be used.

Gear Forces

After executing a layout, mean P.D.'s of the gears were determined by measurement for calculation of reaction forces.

First stage:

Mean P.D. of gear = 5.68 in.

$$F_T = \frac{(823)(2)}{5.68} = 289.8 \text{ lb.}$$

Bevel gear angle

$$\tan \alpha = 6.00/1.75 = 3.4285714$$

$$\alpha = 73.7397950^\circ$$

Separating force

$$F_S = 289.8 \tan 20^\circ = 105.5 \text{ lb.}$$

Radial reaction

$$T_G = F_S \sin \alpha = 101.3 \text{ lb.}$$

Axial reaction

$$T_P = F_S \cos \alpha = 29.5 \text{ lb.}$$

Second stage:

Mean P.D. of gears = 2.88 in.

$$F_T = (823)(2)/2.88 = 571.5 \text{ lb.}$$

$$\alpha = 45^\circ$$

$$F_S = 571.5 \tan 20^\circ = 208 \text{ lb.}$$

$$T_G = 208 \sin 45^\circ = 147 \text{ lb.}$$

$$T_P = 208 \cos 45^\circ = 147 \text{ lb.}$$

These forces are used in final design for computation of bearing loads and structural deflections.

To summarize the gear data, all gears are 20° pressure angle standard involute AISI 9310 steel, case carburized to R_c 60-62. These data are shown in Tables 8 and 9.

APPENDIX D

SOFTWARE FOR PITCH YAW JOINT

NASA 1.C
NASA 2.C
NASA 3.C
NASA 4.C
NASA 5.C
NASA 6.C
NASA 7.C
NASA 8.C
NASADATA.H
EXTDATA.H
NASADEF.H

```
/************************
 *      FILE: NASA1.C
 *      DATE: July 28, 1990
 *
 *   ROUTINES
 *      System_Startup
 *      System_Shutdown
 *      Restart_Operation
 *      Parameter_Operation
 *      Quit_Operation
 *      Select_Function
 *      main
 *
 *
 ************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>

/***********************
 *  perform the initial startup of the system cards in a safe and
 *  known condition and all motors locked
 ***********************/

void System_Startup(void)
        {
        Set_AD_Card();
        Lock_Brakes();
        Load_Parameters();
        MPTR->py.old_pitch = MPTR->sp.pitch_neutral;
        MPTR->py.old_yaw   = MPTR->sp.yaw_neutral;
        MPTR->ma.res_rev = 0;
        MPTR->mb.res_rev = 0;
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
/* for testing only REMOVE FOR OPERATION*/
/*      MPTR->py.pitch_pos = MPTR->sp.pitch_neutral;
        MPTR->py.yaw_pos   = MPTR->sp.yaw_neutral;*/
        }


/***********************
 *  perform the system startdown
 ***********************/

void System_Shutdown(void)
        {
        Save_Parameters();
        textcolor(WHITE);
        textbackground(BLACK);
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        clrscr();
        }


/***********************
 *  restart the system operating program
 ***********************/

void Restart_Operation(void)
        {
        Set_AD_Card();
        Load_Parameters();
        ERROR_FLAG = RESET;
        Display_Error(TRUE);
        }


/***********************
 *  function to display system data,
 *  setpoint values and gain values
 ***********************/

void Parameter_Operation(void)
        {
        int     done_flag;

        done_flag = FALSE;
        while (!done_flag)
                {
                textcolor(BLACK);
                textbackground(WHITE);
                flushall();
                gotoxy(20,15);
                printf("                                 ");
                gotoxy(20,15);
                printf("  D - DISPLAY SYSTEM DATA        ");
                gotoxy(20,16);
                printf("  S - DISPLAY SYSTEM SETPOINTS   ");
                gotoxy(20,17);
                printf("  G - DISPLAY SYSTEM GAINS       ");
                gotoxy(20,18);
                printf("  E - EXIT                       ");
                gotoxy(20,20);
                printf("  Enter In Desired Operation:    ");
                gotoxy(50,20);
                switch(toupper(getche()))
                        {
                        case 'D': Display_System_Data();               break;
                        case 'S': Display_System_Setpoints();   break;
                        case 'G': Display_System_Gains();              break;
                        case 'E': done_flag = TRUE;                    break;
                        default: break;
                        }
                }
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(20,14);
        clreol();
        gotoxy(20,15);
        clreol();
        gotoxy(20,16);
        clreol();
        gotoxy(20,17);
        clreol();
        gotoxy(20,18);
        clreol();
```

```
        gotoxy(20,20);
        clreol();
        }

/**********************
 *  does system shutdown
 *  will save system parameters if
 *  given the OK
 **********************/

void Quit_Operation(void)
        {
        Disable_AD_Interrupt();
        System_Shutdown();
        OPERATE = FALSE;
        }


/**********************
 *  displays the command options
 **********************/

void Select_Function(void)
        {
        int loop;

        for (loop = 0; loop < 30; ++loop)
                Get_Joint_Data();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(40,24);
        cputs("CONDITION");
        gotoxy(1,1);
        clreol();
        gotoxy(17,1);
        cputs("NASA LeRC - TRANSMISSION RESEARCH TEST CENTER");
        gotoxy(50,24);
        cputs("                    ");
        gotoxy(1,22);
        cputs("PITCH ANGLE =              ");
        gotoxy(15,22);
        printf("%.2f",MPTR->py.pitch_ang);
        gotoxy(1,23);
        cputs("YAW ANGLE   =              ");
        gotoxy(1,24);
        cputs("BRAKES :                   ");
        Lock_Brakes();
        gotoxy(15,23);
        printf("%.2f",MPTR->py.yaw_ang);
        gotoxy(20,4);
        cprintf("  R - RESTART THE SYSTEM             ");
        gotoxy(20,5);
        cprintf("  S - SINGLE COMMAND INPUT           ");
        gotoxy(20,6);
        cprintf("  M - MULTIPLE COMMAND INPUT         ");
        gotoxy(20,7);
        cprintf("  J - JOYSTICK SYSTEM OPERATION      ");
        gotoxy(20,8);
        cprintf("  N - MOVE TO NEUTRAL POSITION       ");
        gotoxy(20,9);
        cprintf("  P - SYSTEM PARAMETER OPEATION      ");
        gotoxy(20,10);
        cprintf("  E - EXIT                           ");
        gotoxy(20,12);
        cprintf("  Enter In Desired Operation:        ");
        gotoxy(50,12);
```

```
        switch(toupper(getche()))
                {
                case 'R': Restart_Operation();  break;
                case 'S': Command_Operation();  break;
                case 'M': M_Command_Operation();        break;
                case 'J': Joystick_Operation(); break;
                case 'N': Neutral_Operation();      break;
                case 'P': Parameter_Operation();        break;
                case 'E': Quit_Operation();             break;
                default: break;
                }
        textcolor(WHITE);
        textbackground(BLACK);
        }


void main(void)
        {
        OPERATE = TRUE;
        System_Startup();
        while (OPERATE) Select_Function();
        System_Shutdown();
        }
```

```
/************************
 *      FILE: NASA2.C
 *      DATE: July 28, 1990
 *
 *   ROUTINES
 *      Timer
 *      Set_Clock
 *      Unset_Clock
 *      Set_AD_Card
 *      Lock_Brakes
 *      Unlock_Brakes
 *      Enable_AD_Interrupt
 *      Disable_AD_Interrupt
 *      Read_AD_Channel
 *      Read_Resolvers
 *      Get_Joint_Data
 *
 ************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>


/************************
 *  global variables
 ************************/
float pitch_new, pitch_old;
float yaw_new, yaw_old;
float res_a_new, res_a_old;
float res_b_new, res_b_old;

/************************
 *  save the address of the printer
 *  interrupt vector for replacement
 ************************/

void interrupt ( *oldhandler)(void);     /*vector of interrupt routine
                  that will be replace for A/D clock*/

/************************
 *  this is the new interrupt handler, on interrupt the flag START_LOOP
 *  is set, the program monitors the flag for sync operation
 ************************/

void interrupt Timer(void)
        {
        disable();
        START_LOOP = TRUE;
        outportb(AD_STATUS, CLR_INT);           /*enables A/D bd interrupt*/
        outportb(0x0020, 0x20);            /*reset interrupt chip*/
        enable();
```

```
        }

/***********************
 *  replace the printer interrupt handler with the above handler
 ***********************/

void Set_Clock_Up(void)
        {
        START_LOOP = FALSE;
        oldhandler = getvect(INTR_LOC);
        setvect(INTR_LOC, Timer);
        }


/***********************
 *  return the printer handler to the interrupt vector location
 ***********************/

void Unset_Clock(void)
        {
        START_LOOP = FALSE;
        outportb(AD_CON, INT_OFF);
        setvect(INTR_LOC, oldhandler);
        }


/***********************
 *  set up the A/D - D/A card for initial operation
 ***********************/

void Set_AD_Card(void)
        {
        int temp, low, high;
        unsigned char value;
        outportb(AD_CON, INT_OFF);
        outportb(AD_SCAN, 0x00);                        /*one channel scan*/
        outportb(AD_STATUS, CLR_INT);             /*clear any interrupts*/
        outportb(DA0_LOW, 0x00);              /*set D/A's to mid range*/
        outportb(DA0_HI, 0x80);
        outportb(DA1_LOW, 0x00);
        outportb(DA1_HI, 0x80);
        outportb(IO_OUT, LOCK_B);
        outportb(AD_TIME,0x01);      /*allow counters 1 & 2 to count*/
        outportb(AD_GAIN,GAIN_1);
        outportb(ADc_CON, 0xb4);         /*cntr 2, r/l low then high byte, rate g
en*/
        outportb(ADc_CON, 0x74);         /*cntr 1, r/l low then high byte, rate g
en*/
        outportb(AD_CNTR_2, 0x14);
        outportb(AD_CNTR_2, 0x00);    /*load counter two with 20          */
        temp = 50000 / FREQUENCY;     /* A/D internal clock is 1000000Hz  */
        high = temp / 256;            /*1000000 / 20 = 50000  using freq, */
        low  = temp - (256 * high);   /*now find the low & high bytes for */
        outportb(AD_CNTR_1, low);     /*counter one and load these in     */
        outportb(AD_CNTR_1, high);    /*see A/D manual                    */
        pitch_old = MPTR->sp.pitch_neutral;
        yaw_old = MPTR->sp.yaw_neutral;
        }


/***********************
 *  lock the motor brakes
 ***********************/

void Lock_Brakes(void)
        {
```

```
        outportb(IO_OUT, LOCK_B);
        MPTR->ma.mot_drive = 2048;
        MPTR->mb.mot_drive = 2048;
        Motor_Drive();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(10,24);
        cputs("              ");
        gotoxy(10,24);
        cputs("  LOCK  ");
        sound(800);
        delay(50);
        nosound();
        }


/************************
 *  unlock the motor brakes
 ************************/

void Unlock_Brakes(void)
        {
        MPTR->ma.mot_drive = 2048;
        MPTR->mb.mot_drive = 2048;
        Motor_Drive();
        outportb(IO_OUT, UNLOCK_B);
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(10,24);
        cputs("              ");
        gotoxy(10,24);
        cputs(" UNLOCK ");
        sound(400);
        delay(50);
        nosound();
        }


/************************
 * enable the A/D interrupts for sync operations
 ************************/

void Enable_AD_Interrupt(void)
        {
        outportb(AD_CON, INT_ON);
        outportb(AD_STATUS, CLR_INT);
        enable();
        }


/************************
 *  disable the A/D interrupts for sync operations
 ************************/

void Disable_AD_Interrupt(void)
        {
        outportb(AD_CON, INT_OFF);
        }

/************************
 *  read the value out of an A/D channel
 ************************/

int Read_AD_Chan(channel)
        unsigned char   channel;
        {
```

```
	int			loop;
	unsigned char  low, high;
	unsigned short raw;
	int			data;

	outportb(0x332, channel);
	for (loop = 0; loop <= 1; ++loop);
	outportb(0x330, 0x00);
	for (loop = 0; loop <= 2; ++loop);
	low  = inportb(0x330);
	high = inportb(0x331);
	raw = ((((high << 8) + low) >> 4) & 0x0fff);
	data = (int) raw - 2048;
/*	gotoxy(40, 23);
	clreol();
	gotoxy(40,23);
	if(channel == 0x00)
		{
		printf("ch %x    raw %x   data = %d", channel, raw, data);
		delay(10);
		}		test only*/
	return data;
	}


/***********************
 *  The the six channels of the resolvers
 *  check for error conditions
 *  on pitch and yaw, read the x1 and x16 channels
 *  and find the final value
 ***********************/

void Read_Resolvers()
	{
	unsigned char error, count, low_b, high_b;
	unsigned short x1, x16, temp_x1, temp_x16;
	static  unsigned int  ma_res_old = 32768;
	static    unsigned int  mb_res_old = 32768;

	outportb(BASE, M_A_R);			/*read motor A input resolver*/
	low_b  = inportb(LSB);
	high_b = inportb(MSB);
	x1 = (high_b * 256 + low_b) & 0xfff0;  /*12 bits*/
	if (x1 < 49152) x1 = x1 + 16384;
	else x1 = x1 - 49152;
	MPTR->ma.res_cnt = x1;
	if ((ma_res_old > 65280) && (MPTR->ma.res_cnt < 256)) MPTR->ma.res_rev +
= 1;
	else if ((ma_res_old < 256) && (MPTR->ma.res_cnt > 65280)) MPTR->ma.res_
rev -= 1;
	ma_res_old = MPTR->ma.res_cnt;
	outportb(BASE, M_B_R);				/*read motor B input resolver*/
	low_b  = inportb(LSB);
	high_b = inportb(MSB);
	x1 = (high_b * 256 + low_b) & 0xfff0;
	if (x1 < 49152) x1 = x1 + 16384;
	else x1 = x1 - 49152;
	MPTR->mb.res_cnt = x1;
	if ((mb_res_old > 65280) && (MPTR->mb.res_cnt < 256)) MPTR->mb.res_rev +
= 1;
	else if ((mb_res_old < 256) && (MPTR->mb.res_cnt > 65280)) MPTR->mb.res_
rev -= 1;
	mb_res_old = MPTR->mb.res_cnt;

	outportb(BASE, P_LO);			/*read pitch resolver X1 windings*/
	low_b  = inportb(LSB);
```

```
        high_b = inportb(MSB);
        x1 = (high_b * 256 + low_b) & 0xfffc;    /*get 14 bits*/
        if (x1 < 49152) x1 = x1 + 16384;                    /*adjust for offset*/
        else x1 = x1 - 49152;
        outportb(BASE, P_HI);          /*(not used)read pitch resolver X16 winding
s*/
        low_b  = inportb(LSB);
        high_b = inportb(MSB);
        x16 = (high_b * 256 + low_b) & 0xfffc;
/*      temp_x1  = x1 & 0x0f00;           */       /*check for section boundarys*/
/*      temp_x16 = x16 & 0xf000;
        if ((temp_x1 >= 0x0e00) && (temp_x16 <= 0x2000)) x1 = x1 + 0x1000;
        else if ((temp_x1 <= 0x0200) && (temp_x16 >= 0xe000)) x1 = x1 - 0x1000;
        MPTR->py.pitch_pos = (x1 & 0xf000) | ((x16 & 0xfff0) >> 4);  */  /*find
final result*/
/* PUT IN FOR REAL OPERATION*/
        MPTR->py.pitch_pos = x1;                /*find final result*/

        outportb(BASE, Y_LO);           /*read yaw resolver X1 windings*/
        low_b  = inportb(LSB);
        high_b = inportb(MSB);
        x1 = (high_b * 256 + low_b) & 0xfffc;    /*get 14 bits*/
        if (x1 < 49152) x1 = x1 + 16384;                    /*adjust for offset*/
        else x1 = x1 - 49152;
/*      outportb(BASE, Y_HI);*/ /*(not used)read yaw resolver X16 windings*/
/*      low_b  = inportb(LSB);
        high_b = inportb(MSB);
        x16 = (high_b * 256 + low_b) & 0xfffc;*/
/*      temp_x1  = x1 & 0x0f00; */                 /*check for section boundarys*/
/*      temp_x16 = x16 & 0xf000; */
/*      if ((temp_x1 >= 0x0e00) && (temp_x16 <= 0x2000)) x1 = x1 + 0x1000;
        else if ((temp_x1 <= 0x0200) && (temp_x16 >= 0xe000)) x1 = x1 - 0x1000;*
/
/*      MPTR->py.yaw_pos = (x1 & 0xf000) | ((x16 & 0xfff0) >> 4);*/      /*find f
inal result*/
/*PUT IN FOR REAL OPERATION*/
/*      yaw_new = 0.0 * yaw_old + 1.0 * (float) (x1);
        yaw_old = yaw_new;
        MPTR->py.yaw_pos = (int)(yaw_new);*/
        MPTR->py.yaw_pos = x1;

        MPTR->ma.res_ang    = 360.0 * (float)(MPTR->ma.res_cnt) / 65536.0;
        MPTR->mb.res_ang    = 360.0 * (float)(MPTR->mb.res_cnt) / 65536.0;
        MPTR->py.pitch_ang = (360.0 * (float)(MPTR->py.pitch_pos - MPTR->sp.pitc
h_neutral) / 65536.0);
        MPTR->py.yaw_ang    = (360.0 * (float)(MPTR->py.yaw_pos   - MPTR->sp.yaw_
neutral)   / 65536.0);
        if (MPTR->py.pitch_ang > 180.0) MPTR->py.pitch_ang -= 360.0;
        if (MPTR->py.yaw_ang > 180.0) MPTR->py.yaw_ang -= 360.0;
        }


/************************
 *  collect the joint data and store away
 ************************/

void Get_Joint_Data(void)
        {
        int loop;
        int tempa, tempb;
        int torq_loop = 16;


        outportb(AD_CON, AD_SET);
        outportb(AD_STATUS, CLR_INT);
/* PUT IN FOR REAL OPERTION next 6 lines */
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
MPTR->ma.tach       = Read_AD_Chan(AD_TACH_A);
MPTR->mb.tach       = Read_AD_Chan(AD_TACH_B);
MPTR->ma.res_vel    = Read_AD_Chan(AD_RD_V_A);
MPTR->mb.res_vel    = Read_AD_Chan(AD_RD_V_B);
MPTR->py.pitch_vel  = Read_AD_Chan(AD_RD_V_P);
MPTR->py.yaw_vel    = Read_AD_Chan(AD_RD_V_Y);
outportb(AD_GAIN, GAIN_500);
Read_Resolvers();
/*test only  next four lines*/
MPTR->py.pitch_vel = (int)(MPTR->py.pitch_pos - MPTR->py.old_pitch);
MPTR->py.yaw_vel   = (int)(MPTR->py.yaw_pos - MPTR->py.old_yaw);
MPTR->py.old_pitch = MPTR->py.pitch_pos;
MPTR->py.old_yaw   = MPTR->py.yaw_pos;
tempa = 0;
tempb = 0;
for (loop = 0; loop < torq_loop; ++loop)
        tempa = Read_AD_Chan(AD_TORQ_A) + tempa;
for (loop = 0; loop < torq_loop; ++loop)
        tempb = Read_AD_Chan(AD_TORQ_B) + tempb;
outportb(AD_GAIN, GAIN_1);
MPTR->ma.torq = (int) (tempa / torq_loop);
MPTR->mb.torq = (int) (tempb / torq_loop);
Enable_AD_Interrupt();
}
```

```
/************************
 *       FILE: NASA3.C
 *       DATE: July 28, 1990
 *
 *    ROUTINES
 *       Initial_System
 *       Display_Error
 *       Motor_Drive
 *       Data_Record
 *       Data_Save
 *       Load_Parameters
 *       Save_Parameters
 *
 *
 *************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>


/************************
 *  routine to set up system for arm motion
 ************************/

void Initial_System(void)
        {
        RUN_ARM = TRUE;
        ERROR_FLAG = FALSE;
        MPTR->sp.slimit_pitch = FALSE;
        MPTR->sp.slimit_yaw   = FALSE;
        MPTR->ma.mot_drive = 2048;
        MPTR->mb.mot_drive = 2048;
        MPTR->ds.counter = 0;
        LOG_CNT = 0;
        LOG_REV = FALSE;
/* for testing only*/
/*      MPTR->py.pitch_vel = 0;
        MPTR->py.yaw_vel   = 0;
        MPTR->ma.tach      = 2048;
        MPTR->mb.tach      = 2048;*/
        Motor_Initial();
        }


/************************
 *  routine to display the reason for the system shutdown
 ************************/

void Display_Error(int flag)
        {
        if (flag == TRUE)
```

```
                gotoxy(50,24);
        else gotoxy(25,18);
        textcolor(BLACK);
        textbackground(WHITE);
        if (ERROR_FLAG == P_HI_LIMIT) cputs("PITCH LIMIT - HIGH");
        else if (ERROR_FLAG == P_LO_LIMIT) cputs("PITCH LIMIT - LOW");
        else if (ERROR_FLAG == Y_HI_LIMIT) cputs("YAW LIMIT - HIGH");
        else if (ERROR_FLAG == Y_LO_LIMIT) cputs("YAW LIMIT - LOW");
        else if (ERROR_FLAG == RESOLVER_ERROR) cputs("RESOLVER ERROR");
        else if (ERROR_FLAG == NORMAL_SHUTDOWN) cputs("NORMAL SHUTDOWN");
        else if (ERROR_FLAG == RESET) cputs("RESTARTING COMPUTER");
        else if (ERROR_FLAG == SOFT_LIMIT_P) cputs("PITCH SOFT LIMIT");
        else if (ERROR_FLAG == SOFT_LIMIT_Y) cputs("YAW SOFT LIMIT");
        else if (ERROR_FLAG == TIME_OUT) cputs("TIME OUT ERROR");
        else if (ERROR_FLAG == AD_FAIL) cputs("AD INTERRUPT FAIL");
        sound(500);
        delay(100);
        nosound();
        }


/***********************
 *  output values for motor drive
 *  check for proper range of values
 ***********************/

void Motor_Drive(void)
        {
        unsigned int      temp_a, temp_b;
        unsigned char     hi_a, lo_a, hi_b, lo_b;

        if (MPTR->ma.mot_drive > 4095) MPTR->ma.mot_drive = 4094;
          else if (MPTR->ma.mot_drive < 0) MPTR->ma.mot_drive = 0;
        if (MPTR->mb.mot_drive > 4095) MPTR->mb.mot_drive = 4094;
          else if (MPTR->mb.mot_drive < 0) MPTR->mb.mot_drive = 0;
        temp_a = (MPTR->ma.mot_drive << 4);
        hi_a   = ((temp_a & 0xff00) >> 8);
        lo_a   = (temp_a & 0x00ff);
        temp_b = (MPTR->mb.mot_drive << 4);
        hi_b   = ((temp_b & 0xff00) >> 8);
        lo_b   = (temp_b & 0x00ff);
        outportb(DA0_LOW,lo_a);
        outportb(DA0_HI, hi_a);
        outportb(DA1_LOW,lo_b);
        outportb(DA1_HI, hi_b);
/*      printf("lo hi a-b  %x  %x  %x  %x\n", hi_a, lo_a, hi_b, lo_b);  */
        }


/***********************
 *  routine to record joint data for later display
 ***********************/

void Data_Record(void)
        {
        log_tach_a[LOG_CNT] = MPTR->ma.tach;
        log_tach_b[LOG_CNT] = MPTR->mb.tach;
        log_torq_a[LOG_CNT] = MPTR->ma.torq;
        log_torq_b[LOG_CNT] = MPTR->mb.torq;
        log_res_cnt_a[LOG_CNT] = MPTR->ma.res_cnt;
        log_res_cnt_b[LOG_CNT] = MPTR->mb.res_cnt;
        log_res_rev_a[LOG_CNT] = MPTR->ma.res_rev;
        log_res_rev_b[LOG_CNT] = MPTR->mb.res_rev;
        log_motor_cmd_a[LOG_CNT] = MPTR->ma.mot_drive;
        log_motor_cmd_b[LOG_CNT] = MPTR->mb.mot_drive;
        log_pitch_pos[LOG_CNT] = MPTR->py.pitch_pos;
```

ORIGINAL PAGE IS OF POOR QUALITY

```
        log_pitch_vel[LOG_CNT] = MPTR->py.pitch_vel;
        log_yaw_pos[LOG_CNT] = MPTR->py.yaw_pos;
        log_yaw_vel[LOG_CNT] = MPTR->py.yaw_vel;
        log_desire_pitch_pos[LOG_CNT] = MPTR->ds.pitch_pos;
        log_desire_yaw_pos[LOG_CNT] = MPTR->ds.yaw_pos;
        LOG_CNT = LOG_CNT + 1;
        if (LOG_CNT == MAX_LOG)
                {
                LOG_CNT = 0;
                LOG_REV = TRUE;
                }
        }

/***********************
 *  routine to save joint data for later display
 ***********************/

void Data_Save(void)
        {
        FILE                                    *param_file, *fopen();
        unsigned int            ustemp;




        int                     itemp;
        float                   ftemp;
        int                     loop, count;
        char                            answer;
        char                    logfilename[12];
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(20,15);
        cputs("SYSTEM DATA HAS BEEN COLLECTED                  ");
        gotoxy(20,16);
        cputs("DO YOU WISH TO SAVE THE DATA  Y/N               ");
        gotoxy(60,16);
        flushall();
        answer = fgetchar();
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(20,15);
        clreol();
        gotoxy(20,16);
        clreol();
        gotoxy(50,24);
        clreol();
        gotoxy(50,24);
        textcolor(BLACK);
        textbackground(WHITE);
        cputs("                    ");
        if ((answer == 'Y') || (answer == 'y'))
                {
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(20,15);
                cputs("ENTER IN FILE NAME                          ");
                gotoxy(45,15);
                flushall();
                scanf("%11s", logfilename);
                textcolor(WHITE);
                textbackground(BLACK);
                gotoxy(20,15);
                clreol();
```

```
		param_file = fopen(logfilename, "w");
		if (param_file == NULL)
			{
			textcolor(BLACK);
			textbackground(WHITE);
			gotoxy(50,24);
			cputs("FILE ERROR");
			sound(500);
			delay(1500);
			nosound();
			delay(5000);
			}
		textcolor(BLACK);
		textbackground(WHITE);
		gotoxy(50,24);
		printf("WORKING");
		if (LOG_REV == FALSE)
			{
			fprintf(param_file, "       TACH A                 TACH B
     TORQUE A              TORQUE B\n");
			for (loop = 0; loop < LOG_CNT; loop++)
				{
				itemp = log_tach_a[loop];
				ftemp = (float)itemp * 2.09263;
				fprintf(param_file, "%d     %f                ", loop,f
temp);

				itemp=log_tach_b[loop];
				ftemp = (float)itemp * 2.09263;
				fprintf(param_file, "%f                ", ftemp);
				itemp=log_torq_a[loop];
				ftemp = (float)itemp * 0.024414;
				fprintf(param_file, " %f               ", ftemp);
				itemp = log_torq_b[loop];
				ftemp = (float)itemp * 0.024414;
				fprintf(param_file, "%f  \n",ftemp);
				}
			fprintf(param_file, "       RESOLVER A          RESOLVER A
REV          RESOLVER B          RESOLVER B REV\n");                    for (loo
p = 0; loop < LOG_CNT; loop++)
				for (loop=0; loop< LOG_CNT; loop++)
				{
				ustemp = log_res_cnt_a[loop];
				ftemp = (float)ustemp * 0.005493;
				fprintf(param_file, "%d     %f                ",lo
op,ftemp);

				itemp = log_res_rev_a[loop];
				fprintf(param_file, "%d                ", itemp)
;

				ustemp = log_res_cnt_b[loop];
				ftemp = (float)ustemp * 0.005493;
				fprintf(param_file, "%f                ", ftemp);
				itemp = log_res_rev_b[loop];
				fprintf(param_file, "%d  \n", itemp);
				}
			fprintf(param_file, "       MOTOR COMMAND A          MOTOR C
OMMAND B       PITCH POSITION          PITCH VELOCITY\n");
				for (loop = 0; loop < LOG_CNT; loop++)
				{
				itemp = log_motor_cmd_a[loop]-2048;
				ftemp = (float)itemp * 0.004883;
				fprintf(param_file, "%d     %f                ", loop
,ftemp);

				itemp = log_motor_cmd_b[loop]-2048;
				ftemp = (float)itemp * 0.004883;
				fprintf(param_file, "%f                ",ftemp);
				ustemp = log_pitch_pos[loop];
```

```
                                ftemp = (float)ustemp * 0.005493;
                                fprintf(param_file, "%f              ", ftemp);
                                itemp = log_pitch_vel[loop];
                                fprintf(param_file, "%d  \n", itemp);
                                }
                        fprintf(param_file, "       YAW POSITION          YAW VELOC
ITY         DESIRE PITCH          DESIRE YAW\n");
                        for (loop = 0; loop < LOG_CNT; loop++)
                                {
                                ustemp = log_yaw_pos[loop];
                                ftemp = (float)ustemp * 0.005493;
                                fprintf(param_file, "%d    %f            ", loop,ft
emp);

                                itemp = log_yaw_vel[loop];
                                fprintf(param_file, " %d            ",itemp);
                                itemp = log_desire_pitch_pos[loop];
                                ftemp = (float)itemp * 0.005493;
                                fprintf(param_file, "%f            ",ftemp);
                                itemp = log_desire_yaw_pos[loop];
                                ftemp = (float)itemp * 0.005493;
                                fprintf(param_file, "%f  \n",ftemp);
                        }

/*test routines*/
/*  fprintf(param_file, "TORQUE A\n");
for (loop = 0; loop < LOG_CNT; loop++)
        {
        itemp = log_torq_a[loop];
        fprintf(param_file, "%d    %d\n", loop,itemp);
        }
fprintf(param_file, "TORQUE B\n");
for (loop = 0; loop < LOG_CNT; loop++)
        {
        itemp = log_torq_b[loop];
        fprintf(param_file, "%d    %d\n", loop,itemp);
        }               torque test routines*/

                        }
                else
                        {
                        if ((LOG_CNT - 1) < 0) LOG_CNT = 1;

                        count = 0;
                        fprintf(param_file,  "      TACH A              TACH B
       TORQUE A              TORQUE B\n");
                        for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
                                {
                                itemp = log_tach_a[loop];
                                ftemp = (float)itemp * 2.09263;
                  %f            ", count,ftemp);
                                fprintf(param_file, "%d     %f            ", count
,ftemp);

                                itemp = log_tach_b[loop];
                                ftemp = (float)itemp * 2.09263;
                                                                  ftemp = (float)i
temp * 1.348895;
                                fprintf(param_file, "%f             ",ftemp);
                                itemp = log_torq_a[loop];
                                ftemp = (float)itemp * 0.024414;
                                fprintf(param_file, " %f             ",ftemp);
                                itemp = log_torq_b[loop];
                                ftemp = (float)itemp * 0.024414;
                                fprintf(param_file, "%f  \n",ftemp);
                                count += 1;
                                }
                        for (loop = 0; loop< (LOG_CNT-1); loop++)
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
                                        {
                                        itemp = log_tach_a[loop];
                                        ftemp =(float)itemp * 2.09263;
                                        fprintf(param_file, "%d     %f              ", count,
ftemp);
                                        itemp = log_tach_b[loop];
                                        ftemp = (float)itemp * 2.09263;
                                        fprintf(param_file,"%f             ",ftemp);
                                        itemp = log_torq_a[loop];
                                        ftemp = (float)itemp * 0.024414;
                                        fprintf(param_file, " %f              ",ftemp);
                                        itemp =log_torq_b[loop];
                                        ftemp = (float)itemp * 0.024414;
                                        fprintf(param_file,"%f  \n", ftemp);
                                        count +=1;
                                        }
                                        count =0;
                                        fprintf(param_file, "         RESOLVER  A          RE
SOLVER A REV          RESOLVER B          RESOLVER B REV\n");
                              for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
                                        {
                                        ustemp = log_res_cnt_a[loop];
                                        ftemp = (float)ustemp * 0.005493;
                                        fprintf(param_file, "%d     %f              ", c
ount,ftemp);
                                        itemp = log_res_rev_a[loop];
                                        fprintf(param_file, "%d                    ",itemp);
                                        ustemp = log_res_cnt_b[loop];
                                        ftemp = (float)ustemp*0.005493;
                                        fprintf(param_file,"%d                    ", ftemp);
                                        itemp = log_res_rev_b[loop];
                                        fprintf(param_file, "%d  \n", count,itemp);
                                        count += 1;
                                        }
                              for (loop = 0; loop < (LOG_CNT - 1); loop++)
                                        {
                                        ustemp = log_res_cnt_a[loop];
                                        ftemp = (float)ustemp * 0.005493;
                                        fprintf(param_file, "%d     %f              ",co
unt,ftemp);
                                        itemp = log_res_rev_a[loop];
                                        fprintf(param_file,"%d                    ", itemp);
                                        ustemp = log_res_cnt_b[loop];
                                        ftemp = (float)ustemp * 0.005493;
                                        fprintf(param_file, "%d                    ", ftemp);
                                        itemp = log_res_rev_b[loop];
                                        fprintf(param_file, " %d  \n",itemp);
                                        count += 1;
                                        }
                              count = 0;
                               fprintf(param_file, "       MOTOR COMMAND A        MOTOR
COMMAND B       PITCH POSITION       PITCH VELOCITY\n");
                              for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
                                        {
                                        itemp = log_motor_cmd_a[loop]-2048;
                                        ftemp = (float)itemp * 0.004883;
                                        fprintf(param_file, "%d     %f              ", coun
t,ftemp);
                                        itemp = log_motor_cmd_b[loop]-2048;
                                        ftemp = (float)itemp * 0.004883;
                                        fprintf(param_file, " %f              ",ftemp);
                                        ustemp = log_pitch_pos[loop];
                                        ftemp = (float)ustemp * 0.005493;
                                        fprintf(param_file, "%f              ", ftemp);
                                        itemp = log_pitch_vel[loop];
                                        fprintf(param_file, " %d  \n", itemp);
```

```
				count += 1;
				}
			for (loop = 0; loop < (LOG_CNT - 1); loop++)
				{
				itemp=log_motor_cmd_a[loop];
				ftemp = (float)itemp * 0.004883;
				fprintf(param_file, "%d  %f			", count,
 ftemp);
				itemp =log_motor_cmd_b[loop];
				ftemp = (float)itemp * 0.004883;
				fprintf(param_file, "%f			", ftemp);
				ustemp = log_pitch_pos[loop];
				ftemp = (float)ustemp * 0.005493;
				fprintf(param_file, "%f			", ftemp);
				itemp = log_pitch_vel[loop];
				fprintf(param_file, " %d  \n",itemp);
				count += 1;
				}
			count = 0;
			fprintf(param_file, "      YAW POSITION        YAW VELOC
 ITY         DESIRE PITCH          DESIRE YAW\n");
			for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
				{
				ustemp = log_yaw_pos[loop];
				ftemp = (float)ustemp * 0.005493;
				fprintf(param_file, "%d    %f          ", count,f
 temp);
				itemp = log_yaw_vel[loop];
				fprintf(param_file, "%d          ", itemp);
				itemp = log_desire_pitch_pos[loop];
				ftemp = (float)itemp * 0.005493;
				fprintf(param_file, " %f          ", ftemp);
				itemp = log_desire_yaw_pos[loop];
				ftemp = (float)itemp * 0.005493;
				fprintf(param_file, " %f  \n", ftemp);
				count += 1;
				}
			for (loop = 0; loop < (LOG_CNT - 1); loop++)
				{
				ustemp = log_yaw_pos[loop];
				ftemp = (float)ustemp * 0.005493;
				fprintf(param_file, "%d    %f          ",count,ft
 emp);
				itemp = log_yaw_vel[loop];
				fprintf(param_file, "%d          ", itemp);
				itemp = log_desire_pitch_pos[loop];
				ftemp = (float)itemp * 0.005493;
				fprintf(param_file,"%f          ", ftemp);
				itemp = log_desire_yaw_pos[loop];
				ftemp = (float)itemp * 0.005493;
				fprintf(param_file,"%f  \n", ftemp);
				count +=1;
				}
 /*test routines*/
 /*   count = 0;
 fprintf(param_file, "TORQUE A\n");
 for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
	{
	itemp = log_torq_a[loop];
	fprintf(param_file, "%d    %d\n", count,itemp);
	count += 1;
	}
 for (loop = 0; loop < (LOG_CNT - 1); loop++)
	{
	itemp = log_torq_a[loop];
	fprintf(param_file, "%d    %d\n", count, itemp);
```

```
        count += 1;
        }
count = 0;
fprintf(param_file, "TORQUE B\n");
for (loop = (LOG_CNT - 1); loop < MAX_LOG; loop++)
        {
        itemp = log_torq_b[loop];
        fprintf(param_file, "%d    %d\n", count,itemp);
        count += 1;
        }
for (loop = 0; loop < (LOG_CNT - 1); loop++)
        {
        itemp = log_torq_b[loop];
        fprintf(param_file, "%d    %d\n",count,itemp);
        count += 1;
        }                torque test routines  */

                        }
                fclose(param_file);
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(50,24);
                cputs("                    ");
                textcolor(WHITE);
                textbackground(BLACK);
                }
  }




/***********************
 *  routine to get setpoint data off of disk
 *  This data includes the joint operating limits
 ***********************/

void Load_Parameters(void)
        {
        FILE                            *param_file, *fopen();
        float                   ftemp;
        int                             itemp;
        unsigned short          ustemp;

        param_file = fopen("NASA.PAR", "r");
        if (param_file == NULL)
                {
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(50,25);
                cputs("FILE ERROR");
                }
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.pitch_hi_limit = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.pitch_hi_alarm = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.pitch_neutral = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.pitch_lo_alarm = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.pitch_lo_limit = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.delta_pitch = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.yaw_hi_limit = ustemp;
        fscanf(param_file, "%u", &ustemp);
```

```
        MPTR->sp.yaw_hi_alarm = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.yaw_neutral = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.yaw_lo_alarm = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.yaw_lo_limit = ustemp;
        fscanf(param_file, "%u", &ustemp);
        MPTR->sp.delta_yaw = ustemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_pos_motion_scale = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_pos_motion_scale = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_vel_motion_scale = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_vel_motion_scale = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_motor_vel = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_motor_vel = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_torque = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_torque = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_position = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_position = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.A_output_vel = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->gn.B_output_vel = ftemp;
        fscanf(param_file, "%d", &itemp);
        MPTR->gn.direction = itemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->jy.x_gain = ftemp;
        fscanf(param_file, "%f", &ftemp);
        MPTR->jy.y_gain = ftemp;
        fclose(param_file);
        PARAMETER_FLAG = FALSE;                     /*set to false, set true if valu
e*/
        }                                          /*change*/


                /*routine to save setpoint data and parameters
                  will check to see if data has been changed,  if no
                  the routine will not do anything*/

void Save_Parameters(void)
        {
        FILE                                *param_file, *fopen();
        float                    ftemp;
        int                   itemp;
        unsigned short  ustemp;
        char                                answer;

        if (PARAMETER_FLAG == TRUE)
                {
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(20,15);
                cputs("SYSTEM PARAMETERS HAVE BEEN CHANGED          ");
                gotoxy(20,16);
                cputs("DO YOU WISH TO SAVE NEW PARAMETERS  Y/N      ");
```

```
answer = fgetchar();
textcolor(WHITE);
textbackground(BLACK);
gotoxy(20,15);
clreol();
gotoxy(20,16);
clreol();
gotoxy(50,11);
if ((answer == 'Y') || (answer == 'y'))
        {
        param_file = fopen("NASA.PAR", "w");
        if (param_file == NULL)
                {
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(50,25);
                cputs("FILE ERROR");
                sound(500);
                delay(1500);
                nosound();
                delay(7500);
                }
        ustemp = MPTR->sp.pitch_hi_limit;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.pitch_hi_alarm;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.pitch_neutral;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.pitch_lo_alarm;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.pitch_lo_limit;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.delta_pitch;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.yaw_hi_limit;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.yaw_hi_alarm;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.yaw_neutral;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.yaw_lo_alarm;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.yaw_lo_limit;
        fprintf(param_file, "%u\n", ustemp);
        ustemp = MPTR->sp.delta_yaw;
        fprintf(param_file, "%u\n", ustemp);
        ftemp = MPTR->gn.A_pos_motion_scale;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.B_pos_motion_scale;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.A_vel_motion_scale;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.B_vel_motion_scale;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.A_motor_vel;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.B_motor_vel;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.A_torque;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.B_torque;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.A_position;
        fprintf(param_file, "%f\n", ftemp);
        ftemp = MPTR->gn.B_position;
        fprintf(param_file, "%f\n", ftemp);
```

```
                    fprintf(param_file, "%f\n", ftemp);
                    ftemp = MPTR->gn.A_output_vel;
                    fprintf(param_file, "%f\n", ftemp);
                    ftemp = MPTR->gn.B_output_vel;
                    fprintf(param_file, "%f\n", ftemp);
                    itemp = MPTR->gn.direction;
                    fprintf(param_file, "%d\n", itemp);



                    ftemp = MPTR->jy.x_gain;
                    fprintf(param_file, "%f\n", ftemp);
                    ftemp = MPTR->jy.y_gain;
                    fprintf(param_file, "%f\n", ftemp);
                    fclose(param_file);
                    }
          }
}
```

```
/************************
 *       FILE: NASA4.C
 *       DATE: July 28, 1990
 *
 *    ROUTINES
 *       Brake_Lock
 *       Brake_Unlock
 *       Display_System_Data
 *       Mod_Setpoint
 *       Mod_Gain
 *       Range_Check
 *       Display_System_Setpoints
 *       Display_System_Gains
 *
 *
 ************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>


/************************
 *  lock the motor brakes
 ************************/

void Brake_Lock(void)
        {
        outportb(IO_OUT, LOCK_B);
        MPTR->ma.mot_drive = 2048;
        MPTR->mb.mot_drive = 2048;
        Motor_Drive();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(60,24);
        cputs("                  ");
        gotoxy(60,24);
        cputs("  LOCK  ");
        textcolor(WHITE);
        textbackground(BLACK);
        sound(800);
        delay(50);
        nosound();
        }

/************************
 *  unlock the motor brakes
 ************************/

void Brake_Unlock(void)
        {
        MPTR->ma.mot_drive = 2048;
```

```
        Motor_Drive();
        outportb(IO_OUT, UNLOCK_B);
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(60,24);
        cputs("                ");
        gotoxy(60,24);
        cputs(" UNLOCK ");
        textcolor(WHITE);
        textbackground(BLACK);
        sound(400);
        delay(50);
        nosound();
        }


/***********************
 *  routine to show system data, can unlock brakes and
 *  move joint and watch data changes
 ***********************/

void    Display_System_Data(void)
        {
        int             done_flag;
        char            c;

        done_flag = FALSE;
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,1);
        cputs("  MOTOR A DATA  ");
        gotoxy(1,2);
        cputs(" Tachometer     ");
        gotoxy(1,3);
        cputs(" Torque         ");
        gotoxy(1,4);
        cputs(" Resolver Count ");
        gotoxy(1,5);
        cputs(" Resolver Revs  ");
        gotoxy(1,6);
        cputs(" Resolver Vel   ");
        gotoxy(1,7);
        cputs(" Resolver Angle ");
        gotoxy(1,10);
        cputs("  MOTOR B DATA  ");
        gotoxy(1,11);
        cputs(" Tachometer     ");
        gotoxy(1,12);
        cputs(" Torque         ");
        gotoxy(1,13);
        cputs(" Resolver Count ");
        gotoxy(1,14);
        cputs(" Resolver Revs  ");
        gotoxy(1,15);
        cputs(" Resolver Vel   ");
        gotoxy(1,16);
        cputs(" Resolver Angle ");
        gotoxy(1,18);
        cputs("  PITCH / YAW DATA     ");
        gotoxy(1,19);
        cputs(" Pitch Resolver Count ");
        gotoxy(1,20);
```

```
cputs(" Pitch Resolver Vel      ");
gotoxy(1,21);
cputs(" Pitch Angle             ");
gotoxy(1,22);
cputs(" Yaw Resolver Count      ");
gotoxy(1,23);
cputs(" Yaw Resolver Vel        ");
gotoxy(1,24);
cputs(" Yaw Angle               ");
textcolor(BLACK);
textbackground(WHITE);
gotoxy(1,25);
cputs("ENTER 'E' TO EXIT    'U' TO UNLOCK & 'L' TO LOCK BRAKES");
textcolor(WHITE);
textbackground(BLACK);
while (!done_flag)
        {
        Get_Joint_Data();
        gotoxy(18,2);
        clreol();
        gotoxy(18,2);
        printf("%d", MPTR->ma.tach);
        gotoxy(18,3);
        clreol();
        gotoxy(18,3);
        printf("%d", MPTR->ma.torq);
        gotoxy(18,4);
        clreol();
        gotoxy(18,4);
        printf("%u   %x", MPTR->ma.res_cnt, MPTR->ma.res_cnt);
        gotoxy(18,5);
        clreol();
        gotoxy(18,5);
        printf("%d", MPTR->ma.res_rev);
        gotoxy(18,6);
        clreol();
        gotoxy(18,6);
        printf("%d", MPTR->ma.res_vel);
        gotoxy(18,7);
        clreol();
        gotoxy(18,7);
        printf("%.2f", MPTR->ma.res_ang);
        gotoxy(18,11);
        clreol();
        gotoxy(18,11);
        printf("%d", MPTR->mb.tach);
        gotoxy(18,12);
        clreol();
        gotoxy(18,12);
        printf("%d", MPTR->mb.torq);
        gotoxy(18,13);
        clreol();
        gotoxy(18,13);
        printf("%u   %x", MPTR->mb.res_cnt, MPTR->mb.res_cnt);
        gotoxy(18,14);
        clreol();
        gotoxy(18,14);
        printf("%d", MPTR->mb.res_rev);
        gotoxy(18,15);
        clreol();
        gotoxy(18,15);
        printf("%d", MPTR->mb.res_vel);
        gotoxy(18,16);
        clreol();
        gotoxy(18,16);
        printf("%.2f", MPTR->mb.res_ang);
```

```
                gotoxy(24,19);
                clreol();
                gotoxy(24,19);
                printf("%u   %x", MPTR->py.pitch_pos, MPTR->py.pitch_pos);
                gotoxy(24,20);
                clreol();
                gotoxy(24,20);
                printf("%d", MPTR->py.pitch_vel);
                gotoxy(24,21);
                clreol();
                gotoxy(24,21);
                printf("%.2f", MPTR->py.pitch_ang);
                gotoxy(24,22);
                clreol();
                gotoxy(24,22);
                printf("%u   %x", MPTR->py.yaw_pos, MPTR->py.yaw_pos);
                gotoxy(24,23);
                clreol();
                gotoxy(24,23);
                printf("%d", MPTR->py.yaw_vel);
                gotoxy(24,24);
                clreol();
                gotoxy(24,24);
                printf("%.2f", MPTR->py.yaw_ang);
                delay(100);
                if (kbhit() )
                        {
                        c = getch();
                        if ((c == 'E') || (c == 'e'))   done_flag = TRUE;
                        else if ((c == 'U') || (c == 'u')) Brake_Unlock();
                        else if ((c == 'L') || (c == 'l')) Brake_Lock();
                        }
                }
        clrscr();
        }


/************************
 *  routine to change a setpoint during system operation
 ************************/

void Mod_Setpoint(int max_y)
        {
        int                     select;
        int                     y;
        unsigned int    value;

        select          = 0x00;
        y                       = 2;

        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(1,25);
        clreol();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,25);
        cputs("KEYPAD NUM LOCK ON   USE UP/DOWN KEYS   ENTER TO SELECT");
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(1,y);
        cputs("*");
        while (select != 13)
                {
                delay(100);
                select = getch();
```

```
        switch (select)
                {
                case 50 :  y = y + 1;
                                textcolor(BLACK);
                                textbackground(WHITE);
                                gotoxy(1,y-1);
                                cputs(" ");
                                if (y > max_y) y = 2;
                                textcolor(WHITE);
                                textbackground(BLACK);
                                gotoxy(1,y);
                                cputs("*");
                                break;
                case 56 :  y = y - 1;
                                textcolor(BLACK);
                                textbackground(WHITE);
                                gotoxy(1,y+1);
                                cputs(" ");
                                if (y < 2) y = max_y;
                                textcolor(WHITE);
                                textbackground(BLACK);
                                gotoxy(1,y);
                                cputs("*");
                                break;
                default : break;
                }
        }
textcolor(WHITE);
textbackground(BLACK);
gotoxy(1,25);
clreol();
textcolor(BLACK);
textbackground(WHITE);
gotoxy(1,24);
cputs("ENTER IN NEW SETPOINT VALUE  ");
textcolor(WHITE);
textbackground(BLACK);
gotoxy(30,24);
scanf("%u", &value);
gotoxy(1,24);
clreol();
textcolor(BLACK);
textbackground(WHITE);
gotoxy(1,y);
cputs(" ");
PARAMETER_FLAG = TRUE;
switch (y)
        {
        case 2  : MPTR->sp.pitch_hi_limit = value;      break;
        case 3  : MPTR->sp.pitch_hi_alarm = value;      break;
        case 4  : MPTR->sp.pitch_neutral  = value;      break;
        case 5  : MPTR->sp.pitch_lo_alarm = value;      break;
        case 6  : MPTR->sp.pitch_lo_limit = value;      break;
        case 7  : MPTR->sp.delta_pitch    = value;      break;
        case 8  : MPTR->sp.yaw_hi_limit   = value;      break;
        case 9  : MPTR->sp.yaw_hi_alarm   = value;      break;
        case 10 : MPTR->sp.yaw_neutral    = value;      break;
        case 11 : MPTR->sp.yaw_lo_alarm   = value;      break;
        case 12 : MPTR->sp.yaw_lo_limit   = value;      break;
        case 13 : MPTR->sp.delta_yaw      = value;      break;
        default : break;
        }
textcolor(BLACK);
textbackground(WHITE);
gotoxy(1,25);
cputs("ENTER 'E' TO EXIT    'C' TO CHANGE A SETPOINT  ");
```

```
        textcolor(WHITE);
        textbackground(BLACK);
        }


/***********************
 *  routine to change a gain during system operation
 ***********************/

void Mod_Gain(int max_y)
        {
        int       select;
        int       y;
        float     value;

        select      = 0x00;
        y           = 2;

        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(1,25);
        clreol();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,25);
        cputs("KEYPAD NUM LOCK ON   USE UP/DOWN KEYS   ENTER TO SELECT");
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(1,y);
        cputs("*");
        while (select != 13)
                {
                delay(100);
                select = getch();
                switch (select)
                        {
                        case 50 :  y = y + 1;
                                        textcolor(BLACK);
                                        textbackground(WHITE);
                                        gotoxy(1,y-1);
                                        cputs(" ");
                                        if (y > max_y) y = 2;
                                        textcolor(WHITE);
                                        textbackground(BLACK);
                                        gotoxy(1,y);
                                        cputs("*");
                                        break;
                        case 56 :  y = y - 1;
                                        textcolor(BLACK);
                                        textbackground(WHITE);
                                        gotoxy(1,y+1);
                                        cputs(" ");
                                        if (y < 2) y = max_y;
                                        textcolor(WHITE);
                                        textbackground(BLACK);
                                        gotoxy(1,y);
                                        cputs("*");
                                        break;
                        default : break;
                        }
                }
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(1,25);
        clreol();
        textcolor(BLACK);
```

```
        textbackground(WHITE);
        gotoxy(1,24);
        cputs("ENTER IN NEW GAIN VALUE  ");
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(30,24);
        scanf("%f", &value);
        gotoxy(1,24);
        clreol();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,y);
        cputs(" ");
        PARAMETER_FLAG = TRUE;
        switch (y)
                {
                case 2  : MPTR->gn.A_pos_motion_scale = value;    break;
                case 3  : MPTR->gn.B_pos_motion_scale = value;    break;
                case 4  : MPTR->gn.A_vel_motion_scale = value;    break;
                case 5  : MPTR->gn.B_vel_motion_scale = value;    break;
                case 6  : MPTR->gn.A_motor_vel        = value;    break;
                case 7  : MPTR->gn.B_motor_vel        = value;    break;
                case 8  : MPTR->gn.A_torque           = value;    break;
                case 9  : MPTR->gn.B_torque           = value;    break;
                case 10  : MPTR->gn.A_position        = value;    break;
                case 11  : MPTR->gn.B_position        = value;    break;
                case 12  : MPTR->gn.A_output_vel      = value;    break;
                case 13  : MPTR->gn.B_output_vel      = value;    break;
                case 14  : MPTR->gn.direction         = (int) value;   break;
                case 15  : MPTR->jy.x_gain            = value;    break;
                case 16  : MPTR->jy.y_gain            = value;    break;
/*              case 17  : MPTR->gn.                  = value;    break;
                case 18  : MPTR->gn.                  = value;    break;
                case 19  : MPTR->gn.                  = value;    break;
*/
                default : break;
                }
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,25);
        cputs("ENTER 'E' TO EXIT    'C' TO CHANGE A SETPOINT  ");
        textcolor(WHITE);
        textbackground(BLACK);
        }


/***********************
 *  routine to check that alarms and limits are where they
 *  should be
 ***********************/

void Range_Check(void)
        {
        char     *error;

        error = NULL;
        if (MPTR->sp.pitch_hi_limit < MPTR->sp.pitch_hi_alarm)
                error = "PITCH HIGH LIMIT OUT OF SPEC";
        else if (MPTR->sp.pitch_hi_alarm <= MPTR->sp.pitch_neutral)
                error = "PITCH HIGH ALARM OUT OF SPEC";
        else if (MPTR->sp.pitch_lo_alarm >= MPTR->sp.pitch_neutral)
                error = "PITCH LOW ALARM OUT OF SPEC";
        else if (MPTR->sp.pitch_lo_limit > MPTR->sp.pitch_lo_alarm)
                error = "PITCH LOW LIMIT OUT OF SPEC";
        else if (MPTR->sp.yaw_hi_limit < MPTR->sp.yaw_hi_alarm)
                error = "YAW HIGH LIMIT OUT OF SPEC";
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        else if (MPTR->sp.yaw_hi_alarm <= MPTR->sp.yaw_neutral)
                error = "YAW HIGH ALARM OUT OF SPEC";
        else if (MPTR->sp.yaw_lo_alarm >= MPTR->sp.yaw_neutral)
                error = "YAW LOW ALARM OUT OF SPEC";
        else if (MPTR->sp.yaw_lo_limit > MPTR->sp.yaw_lo_alarm)
                error = "YAW LOW LIMIT OUT OF SPEC";
        if (error != NULL)
                {
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(1,25);
                clreol();
                printf("%s", error);
                sound(500);
                delay(500);
                nosound();
                gotoxy(1,25);
                clreol();
                gotoxy(1,25);
                cputs("ENTER 'F' TO FINISH     'C' TO CHANGE A SETPOINT  ");
                textcolor(WHITE);
                textbackground(BLACK);
                }
        }


/***********************
 *  routine to display system setpoints and allow changing
 *  the setpoints during system operation
 ***********************/

void Display_System_Setpoints(void)
        {
        int         done_flag;
        char        c;
        int         x;
        int         max_y = 13;

        x = 26;
        done_flag = FALSE;
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,1);
        cputs("  SYSTEM SETPOINTS       ");
        gotoxy(1,2);
        cputs(" Pitch Limit High        ");
        gotoxy(1,3);
        cputs(" Pitch Alarm High        ");
        gotoxy(1,4);
        cputs(" Pitch Neutral Position ");
        gotoxy(1,5);
        cputs(" Pitch Alarm Low         ");
        gotoxy(1,6);
        cputs(" Pitch Limit Low         ");
        gotoxy(1,7);
        cputs(" Pitch Delta             ");
        gotoxy(1,8);
        cputs(" Yaw Limit High          ");
        gotoxy(1,9);
        cputs(" Yaw Alarm High          ");
        gotoxy(1,10);
        cputs(" Yaw Neutral Position    ");
        gotoxy(1,11);
```

```
cputs(" Yaw Alarm Low            ");
gotoxy(1,12);
cputs(" Yaw Limit Low            ");
gotoxy(1,13);
cputs(" Yaw Delta                ");
gotoxy(1,25);
cputs("ENTER 'E' TO EXIT    'C' TO CHANGE A SETPOINT  ");
textcolor(WHITE);
textbackground(BLACK);
while (!done_flag)
        {
        gotoxy(x,2);
        clreol();
        gotoxy(x,2);
        printf("%u", MPTR->sp.pitch_hi_limit);
        gotoxy(x,3);
        clreol();
        gotoxy(x,3);
        printf("%u", MPTR->sp.pitch_hi_alarm);
        gotoxy(x,4);
        clreol();
        gotoxy(x,4);
        printf("%u", MPTR->sp.pitch_neutral);
        gotoxy(x,5);
        clreol();
        gotoxy(x,5);
        printf("%u", MPTR->sp.pitch_lo_alarm);
        gotoxy(x,6);
        clreol();
        gotoxy(x,6);
        printf("%u", MPTR->sp.pitch_lo_limit);
        gotoxy(x,7);
        clreol();
        gotoxy(x,7);
        printf("%u", MPTR->sp.delta_pitch);
        gotoxy(x,8);
        clreol();
        gotoxy(x,8);
        printf("%u", MPTR->sp.yaw_hi_limit);
        gotoxy(x,9);
        clreol();
        gotoxy(x,9);
        printf("%u", MPTR->sp.yaw_hi_alarm);
        gotoxy(x,10);
        clreol();
        gotoxy(x,10);
        printf("%u", MPTR->sp.yaw_neutral);
        gotoxy(x,11);
        clreol();
        gotoxy(x,11);
        printf("%u", MPTR->sp.yaw_lo_alarm);
        gotoxy(x,12);
        clreol();
        gotoxy(x,12);
        printf("%u", MPTR->sp.yaw_lo_limit);
        gotoxy(x,13);
        clreol();
        gotoxy(x,13);
        printf("%u", MPTR->sp.delta_yaw);
        delay(500);
        if (kbhit() )
                {
                c = getch();
                if ((c == 'E') || (c == 'e'))   done_flag = TRUE;
                else if ((c == 'C') || (c == 'c'))
                        {
```

```
                                            Mod_Setpoint(max_y);
                                            Range_Check();
                                            }
                                }
                        }
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
        }


/***********************
 *  routine to display system gains and allow changing
 *  the gains during system operation
 ***********************/

void Display_System_Gains(void)
        {
        int             done_flag;
        char            c;
        int             x;
        int             max_y = 16;

        x = 26;
        done_flag = FALSE;
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(1,1);
        cputs("  SYSTEM GAINS          ");
        gotoxy(1,2);
        cputs(" A Position Motion Scale");
        gotoxy(1,3);
        cputs(" B Position Motion Scale");
        gotoxy(1,4);
        cputs(" A Velocity Motion Scale");
        gotoxy(1,5);
        cputs(" B Velocity Motion Scale");
        gotoxy(1,6);
        cputs(" A Motor Velocity       ");
        gotoxy(1,7);
        cputs(" B Motor Velocity       ");
        gotoxy(1,8);
        cputs(" A Torque               ");
        gotoxy(1,9);
        cputs(" B Torque               ");
        gotoxy(1,10);
        cputs(" A Position             ");
        gotoxy(1,11);
        cputs(" B Position             ");
        gotoxy(1,12);
        cputs(" A Output Velocity      ");
        gotoxy(1,13);
        cputs(" B Output Velocity      ");
        gotoxy(1,14);
        cputs(" Direction              ");
        gotoxy(1,15);
        cputs(" Joystick Yaw Gain      ");
        gotoxy(1,16);
        cputs(" Joystick Pitch Gain    ");
/*      gotoxy(1,17);
        cputs("                        ");
        gotoxy(1,18);
        cputs("                        ");
```

```
gotoxy(1,19);
cputs("                                 "); */    /*reserved*/
gotoxy(1,25);
cputs("ENTER 'E' TO EXIT     'C' TO CHANGE A GAIN VALUE  ");
textcolor(WHITE);
textbackground(BLACK);
while (!done_flag)
        {
        gotoxy(x,2);
        clreol();
        gotoxy(x,2);
        printf("%.2f", MPTR->gn.A_pos_motion_scale);
        gotoxy(x,3);
        clreol();
        gotoxy(x,3);
        printf("%.2f", MPTR->gn.B_pos_motion_scale);
        gotoxy(x,4);
        clreol();
        gotoxy(x,4);
        printf("%.2f", MPTR->gn.A_vel_motion_scale);
        gotoxy(x,5);
        clreol();
        gotoxy(x,5);
        printf("%.2f", MPTR->gn.B_vel_motion_scale);
        gotoxy(x,6);
        clreol();
        gotoxy(x,6);
        printf("%.2f", MPTR->gn.A_motor_vel);
        gotoxy(x,7);
        clreol();
        gotoxy(x,7);
        printf("%.2f", MPTR->gn.B_motor_vel);
        gotoxy(x,8);
        clreol();
        gotoxy(x,8);
        printf("%.2f", MPTR->gn.A_torque);
        gotoxy(x,9);
        clreol();
        gotoxy(x,9);
        printf("%.2f", MPTR->gn.B_torque);
        gotoxy(x,10);
        clreol();
        gotoxy(x,10);
        printf("%.2f", MPTR->gn.A_position);
        gotoxy(x,11);
        clreol();
        gotoxy(x,11);
        printf("%.2f", MPTR->gn.B_position);
        gotoxy(x,12);
        clreol();
        gotoxy(x,12);
        printf("%.2f", MPTR->gn.A_output_vel);
        gotoxy(x,13);
        clreol();
        gotoxy(x,13);
        printf("%.2f", MPTR->gn.B_output_vel);
        gotoxy(x,14);
        clreol();
        gotoxy(x,14);
        printf("%d", MPTR->gn.direction);
        gotoxy(x,15);
        clreol();
        gotoxy(x,15);
        printf("%.2f", MPTR->jy.x_gain);
        gotoxy(x,16);
        clreol();
```

```
                gotoxy(x,16);
                printf("%.2f", MPTR->jy.y_gain);
/*              gotoxy(x,17);
                clreol();
                gotoxy(x,17);
                printf("%d", MPTR->gn. );
                gotoxy(x,18);
                clreol();
                gotoxy(x,18);
                printf("%d", MPTR->gn. );
                gotoxy(x,19);
                clreol();
                gotoxy(x,19);
                printf("%d", MPTR->gn. );        */    /*reserved*/
                delay(500);
                if (kbhit() )
                        {
                        c = getch();
                        if ((c == 'E') || (c == 'e'))    done_flag = TRUE;
                        else if ((c == 'C') || (c == 'c')) Mod_Gain(max_y);
                        }
                }
        textcolor(WHITE);
        textbackground(BLACK);
        clrscr();
        }
```

```
/************************
 *      FILE: NASA5.C
 *      DATE: August 4, 1990
 *
 *   ROUTINES
 *      Motion_Done
 *      Get_Command_Position
 *      Error_Check
 *      Calculate_Position
 *      Calculate_Drive
 *      Command_Operation
 *      Neutral_Command
 *      Neutral_Operation
 *
 *
 ************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>

/***********************
 *  DATA
 ***********************/

 int                soft_pitch_flag;
 int                soft_yaw_flag;
 int                ad_check;
 unsigned int       counter;

/***********************
 *  routine to check when joint motion is done
 *  joystick is no longer active or on commanded position
 *  as been reached
 ***********************/

void Motion_Done(void)
        {
        int     checkp, checky;

        if ((abs(MPTR->py.pitch_pos - MPTR->ds.final_pitch)) <= MPTR->sp.delta_p
itch)
                checkp = TRUE;
        else checkp = FALSE;
        if ((abs(MPTR->py.yaw_pos - MPTR->ds.final_yaw)) <= MPTR->sp.delta_yaw)
                checky = TRUE;
        else checky = FALSE;
        if (checkp && checky && (MPTR->ds.counter >= MPTR->ds.steps))
                {
                RUN_ARM = FALSE;
                ERROR_FLAG = NORMAL_SHUTDOWN;
                }
        }
```

```
/************************
 *  routine to get desired position of the joint
 ************************/

void Get_Command_Position(void)
        {
        float           value;
        float           low_limit;
        float           high_limit;
        int             moves;
        short           temp;
        unsigned int    steps;
        int             flag;

        MPTR->ds.counter = 0;
        textcolor(BLACK);
        textbackground(WHITE);
        do
                {
                flag = TRUE;
                gotoxy(20,15);
                cputs("  ENTER DESIRED MOVE TIME (seconds)           ");
                gotoxy(58,15);
                scanf("%f", &value);
                if (value > 0.0) flag = FALSE;
                }
        while (flag);
        MPTR->ds.time = value;
        do
                {
                flag = TRUE;
                gotoxy(20,16);
                cputs("  ENTER IN DESIRED PITCH ANGLE                ");
                gotoxy(58,16);
                scanf("%f", &value);
                low_limit = ((float)(MPTR->sp.pitch_lo_alarm - MPTR->sp.pitch_ne
utral) / 65536.0);
                low_limit = (low_limit * 360.0) - 360.0;
                high_limit = ((float)(MPTR->sp.pitch_hi_alarm - MPTR->sp.pitch_n
eutral) / 65536.0);
                high_limit = high_limit * 360.0;
                if ((value < high_limit) && (value > low_limit)) flag = FALSE;
                }
        while (flag);
        MPTR->ds.pitch_angle = value;
        value = (value * 65536.0) / 360.0;
        MPTR->ds.delta_pitch_pos   = (int) value;
        MPTR->ds.final_pitch = (unsigned int) (MPTR->sp.pitch_neutral + MPTR->ds
.delta_pitch_pos);
        do
                {
                flag = TRUE;
                gotoxy(20,17);
                cputs("  ENTER IN DESIRED YAW ANGLE                  ");
                gotoxy(58,17);
                scanf("%f", &value);
                low_limit = ((float)(MPTR->sp.yaw_lo_alarm - MPTR->sp.yaw_neutra
l) / 65536.0);
                low_limit = (low_limit * 360.0) - 360.0;
                high_limit = ((float)(MPTR->sp.yaw_hi_alarm - MPTR->sp.yaw_neutr
al) / 65536.0);
                high_limit = high_limit * 360.0;
                if ((value < high_limit) && (value > low_limit)) flag = FALSE;
                }
```

```
        while (flag);
        MPTR->ds.yaw_angle = value;
        value = (value * 65536.0) / 360.0;
        MPTR->ds.delta_yaw_pos   = (int) value;
        MPTR->ds.final_yaw = (unsigned int)(MPTR->sp.yaw_neutral + MPTR->ds.delt
a_yaw_pos);
        MPTR->ds.steps = (unsigned int) (MPTR->ds.time * (float)FREQUENCY);
        MPTR->ds.pitch_step = (float) (((float)MPTR->ds.final_pitch - (float)MPT
R->py.pitch_pos) / (float)MPTR->ds.steps);
        MPTR->ds.yaw_step   = (float) (((float)MPTR->ds.final_yaw - (float)MPTR-
>py.yaw_pos) / (float)MPTR->ds.steps);
        MPTR->ds.pitch_pos = MPTR->py.pitch_pos;
        MPTR->ds.yaw_pos   = MPTR->py.yaw_pos;
        MPTR->od.pitch_pos = MPTR->ds.pitch_pos;
        MPTR->od.yaw_pos   = MPTR->ds.yaw_pos;
        MPTR->ds.pitch_start = MPTR->py.pitch_pos;
        MPTR->ds.yaw_start   = MPTR->py.yaw_pos;
        gotoxy(50,24);
        cputs("WORKING");
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(20,15);
        clreol();
        gotoxy(20,16);
        clreol();
        gotoxy(20,17);
        clreol();
        }


/***********************
 *  routine to check for porper arm (joint) operation
 *  check motor and joint limits
 ***********************/

void Error_Check(void)
        {
        if (MPTR->py.pitch_pos > MPTR->sp.pitch_hi_limit)  /*check for*/
                ERROR_FLAG = P_HI_LIMIT;                      /*fatal errors*/
        if (MPTR->py.pitch_pos < MPTR->sp.pitch_lo_limit)
                ERROR_FLAG = P_LO_LIMIT;
        if (MPTR->py.yaw_pos > MPTR->sp.yaw_hi_limit)
                ERROR_FLAG = Y_HI_LIMIT;
        if (MPTR->py.yaw_pos < MPTR->sp.yaw_lo_limit)
                ERROR_FLAG = Y_LO_LIMIT;
        if (ERROR_FLAG != FALSE) RUN_ARM = FALSE;
        if (MPTR->py.pitch_pos >= MPTR->sp.pitch_hi_alarm)
                MPTR->sp.slimit_pitch = 1;
        if (MPTR->py.pitch_pos <= MPTR->sp.pitch_lo_alarm)
                MPTR->sp.slimit_pitch = -1;
        if (MPTR->py.yaw_pos >= MPTR->sp.yaw_hi_alarm)
                MPTR->sp.slimit_yaw = 1;
        if (MPTR->py.yaw_pos <= MPTR->sp.yaw_lo_alarm)
                MPTR->sp.slimit_yaw = -1;
        }


/***********************
 *  routine to find the next desired joint position
 *  input from commanded position or the joystick
 ***********************/

void Calculate_Position(void)
        {
        float   temp;
```

```
        if (MPTR->ds.counter <= MPTR->ds.steps)
                {
                temp = (MPTR->ds.pitch_step * (float) MPTR->ds.counter) + (float
) MPTR->ds.pitch_start;
                MPTR->ds.pitch_pos = (int) temp;
                temp = (MPTR->ds.yaw_step * (float) MPTR->ds.counter) + (float)
MPTR->ds.yaw_start;
                MPTR->ds.yaw_pos    = (int) temp;
/*printf("%u    %u\n",MPTR->ds.pitch_pos,MPTR->ds.yaw_pos);*/
                }

        MPTR->ds.counter += 1;

        if (MPTR->ds.counter == (2 * MPTR->ds.steps))
                {
                RUN_ARM = FALSE;
                if (MPTR->sp.slimit_pitch != 0) ERROR_FLAG = SOFT_LIMIT_P;
                else if (MPTR->sp.slimit_yaw != 0) ERROR_FLAG = SOFT_LIMIT_Y;
                else    ERROR_FLAG = TIME_OUT;
                }

        MPTR->ds.pitch_vel = (int)(MPTR->ds.pitch_pos - MPTR->od.pitch_pos);
        MPTR->ds.yaw_vel    = (int)(MPTR->ds.yaw_pos - MPTR->od.yaw_pos);
        MPTR->od.pitch_pos = MPTR->ds.pitch_pos;
        MPTR->od.yaw_pos    = MPTR->ds.yaw_pos;
        }


/************************
 *  routine to find the motor input (drive) values based
 *  on system conditions, angles desired and angles true
 ************************/

void Calculate_Drive(void)
        {
        float   diffa, diffb;
        float   diffav, diffbv;
        float   tacha, tachb;
        float   drivea, driveb;
        int     pitch, yaw;
        int     pitchv, yawv;


        /*find if soft limits are in effect*/
        if (MPTR->sp.slimit_pitch != FALSE)
                {
                if (soft_pitch_flag == 0)
                        {
                        soft_pitch_flag = 1;
                        MPTR->od.slimit_pitch = MPTR->py.pitch_pos;
                        }
                else if (MPTR->sp.slimit_pitch == 1)
                        {
                        if (MPTR->py.pitch_pos >= MPTR->od.slimit_pitch) MPTR->d
s.pitch_pos = MPTR->od.slimit_pitch;
                        }
                else
                        {
                        if (MPTR->ds.pitch_pos <= MPTR->od.slimit_pitch) MPTR->d
s.pitch_pos = MPTR->od.slimit_pitch;
                        }
                }
        else soft_pitch_flag = 0;
        if (MPTR->sp.slimit_yaw != FALSE)
                {
                if (soft_yaw_flag == 0)
```

```
                                {
                                soft_yaw_flag = 1;
                                MPTR->od.slimit_yaw = MPTR->ds.yaw_pos;
                                }
                        else if (MPTR->sp.slimit_yaw == 1)
                                {
                                if (MPTR->ds.yaw_pos >= MPTR->od.slimit_yaw) MPTR->ds.ya
w_pos = MPTR->od.slimit_yaw;
                                }
                        else
                                {
                                if (MPTR->ds.yaw_pos <= MPTR->od.slimit_yaw) MPTR->ds.ya
w_pos = MPTR->od.slimit_yaw;
                                }
                        }
                else soft_yaw_flag = 0;

                /*find motor A drive signal*/
                /*find motor B drive signal*/

                pitch  = MPTR->ds.pitch_pos - MPTR->py.pitch_pos;
                yaw    = MPTR->py.yaw_pos - MPTR->ds.yaw_pos;
                pitchv = MPTR->ds.pitch_vel - MPTR->py.pitch_vel;
                yawv   = MPTR->ds.yaw_vel - MPTR->py.yaw_vel;
                diffa  = -(MPTR->gn.A_position * (float) (pitch + yaw));
                diffb  = MPTR->gn.B_position * (float) (pitch - yaw);
                diffav = (MPTR->gn.A_output_vel * (float) (pitchv + yawv));
                diffbv = (MPTR->gn.B_output_vel * (float) (pitchv - yawv));
                tacha  = MPTR->gn.A_motor_vel * (float) (MPTR->ma.tach);
                tachb  = MPTR->gn.B_motor_vel * (float) (MPTR->mb.tach);
                drivea = diffa + diffav - tacha;
                driveb = diffb + diffbv - tachb;
/*printf("drivea/b   %f   %f\n", drivea, driveb);*/
                MPTR->ma.mot_drive = (((int) (drivea)) * MPTR->gn.direction) + 2048;
                MPTR->mb.mot_drive = (((int) (driveb)) * MPTR->gn.direction) + 2048;
/*printf("dfa %f dfb %f dfav %f dfbv %f dva %f dvb %f\n",diffa,diffb,diffav,diff
bv,drivea,driveb);*/
/*printf("drive a %d  b %d\n",MPTR->ma.mot_drive, MPTR->mb.mot_drive);*/
                }


/************************
 *  performs the moving of the arm
 *  given the final pitch and yaw angles
 *  will check for errors and final position
 ************************/

void Command_Operation(void)
        {
        Set_Clock_Up();
        Enable_AD_Interrupt();
        Initial_System();
        Get_Command_Position();
        ad_check = 0;
        Unlock_Brakes();
        while (RUN_ARM)
                {
                ad_check++;
                if (ad_check > 15000)
                        {
                        ERROR_FLAG = AD_FAIL;
                        RUN_ARM = FALSE;
                        }
                while (START_LOOP)
                        {
```

```
                                Calculate_Position();
                                Error_Check();
                                Calculate_Drive();
                                Motor_Drive();
                                Motion_Done();
                                Data_Record();
                                ad_check = 0;
                                START_LOOP = FALSE;
                                }
                        }
        Lock_Brakes();
        Unset_Clock();
        Display_Error(TRUE);
        Data_Save();
        }


/************************
 *  routine to get neutral position of the joint
 ************************/

void Neutral_Command_Position(void)
        {
        float           valuet = 5.0;
        float           valuep = 0.0;
        float           valuey = 0.0;
        short           temp;
        unsigned int    steps;

        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(50,24);
        cputs("WORKING");
        textcolor(WHITE);
        textbackground(BLACK);
        MPTR->ds.time = valuet;
/*      MPTR->ds.pitch_angle =  MPTR->py.pitch_ang;*/ /*valuep*/
/*      valuep = (valuep * 65536.0) / 360.0;
        MPTR->ds.delta_pitch_pos   = (int) valuep;
        MPTR->ds.final_pitch = (unsigned int) (MPTR->sp.pitch_neutral + MPTR->ds
.delta_pitch_pos);*/
        MPTR->ds.final_pitch = (unsigned int) (MPTR->sp.pitch_neutral);
/*      MPTR->ds.yaw_angle = MPTR->py.yaw_ang;  */   /*valuey*/
/*      valuey = (valuey * 65536.0) / 360.0;
        MPTR->ds.delta_yaw_pos   = (int) valuey;
        MPTR->ds.final_yaw = (unsigned int)(MPTR->sp.yaw_neutral + MPTR->ds.delt
a_yaw_pos);*/
        MPTR->ds.final_yaw = (unsigned int)(MPTR->sp.yaw_neutral);
        MPTR->ds.steps = (unsigned int) (MPTR->ds.time * (float)FREQUENCY);
        MPTR->ds.pitch_step = (float) (((float)MPTR->ds.final_pitch - (float)MPT
R->py.pitch_pos) / (float)MPTR->ds.steps);
        MPTR->ds.yaw_step   = (float) (((float)MPTR->ds.final_yaw - (float)MPTR-
>py.yaw_pos) / (float)MPTR->ds.steps);
        MPTR->ds.pitch_pos = MPTR->py.pitch_pos;
        MPTR->ds.yaw_pos   = MPTR->py.yaw_pos;
        MPTR->od.pitch_pos = MPTR->ds.pitch_pos;
        MPTR->od.yaw_pos   = MPTR->ds.yaw_pos;
        MPTR->ds.pitch_start = MPTR->py.pitch_pos;
        MPTR->ds.yaw_start   = MPTR->py.yaw_pos;
        }


/************************
 *  move the joint to the neutral position
 ************************/
```

```
void Neutral_Operation(void)
        {
        Set_Clock_Up();
        Enable_AD_Interrupt();
        Initial_System();
        ad_check = 0;
        Neutral_Command_Position();
        Unlock_Brakes();
        while (RUN_ARM)
                {
                ad_check++;
                if (ad_check > 15000)
                        {
                        ERROR_FLAG = AD_FAIL;
                        RUN_ARM = FALSE;
                        }
                while (START_LOOP)
                        {
                        Get_Joint_Data();
                        Calculate_Position();
                        Error_Check();
                        Calculate_Drive();
                        Motor_Drive();
                        Motion_Done();
                        Data_Record();
                        ad_check = 0;
                        START_LOOP = FALSE;
                        }
                }
        Lock_Brakes();
        Unset_Clock();
        Display_Error(TRUE);
        Data_Save();
        }
```

```
/************************
 *      FILE: NASA6.C
 *      DATE: July 28, 1990
 *
 *   ROUTINES
 *      Motor_Initial
 *      Motor_Sim
 *
 *   WARNING    THIS ROUTINE WILL NOT WORK WITH THE NEW CONTROL
 *              SYSTEM
 ************************/
#include "extdata.h"
#include "nasadef.h"


        int             mot_drive_a, mot_drive_b;
        float   rpc_motor_a, rpc_motor_b;
        float     total_motor_a = 0.0;
        float     total_motor_b = 0.0;
        float   theta_a, theta_b;
        float     theta_gear_a, theta_gear_b;
        float     theta_py_a, theta_py_b;
        int             tach_a, tach_b;
        float   pitch_angle, yaw_angle;
        int             pitch_res, yaw_res;
        long int                des_pitch, des_yaw;
        int             old_pitch_pos, old_yaw_pos;
        int             loop = 0;
        char    dump;
        float   temp_p, temp_y;

        float   k1              = 0.000163;     /*5000/60/100/2048*/
        float   k2              = 0.00488;      /*20v/4096*/
        float     k3        = 0.005493;          /* 360/65536*/
        float     k4            = 15000.0;              /*100hz*60*/
        float   k5              = 500.0;                /*500rpm-1vout*/
        int       gear          = 90;
        float   py_gear   = 4.3;
        float   time_div  = 250;        /*100hz*/
        int             done;
        float   gain_pos  = 0.8;
        float   gain_vel  = 0.0;
        float   gain_tach = 0.6;

        int             desire_pitch, desire_yaw;
        float   step_pitch, step_yaw;


void Motor_Initial(void)
        {
        loop = 0;
        old_pitch_pos = MPTR->py.pitch_pos;
        old_yaw_pos   = MPTR->py.yaw_pos;
        }
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
void Motor_Sim(void)
        {
                loop = loop + 1;
                rpc_motor_a  = (MPTR->ma.mot_drive - 2048) * k1;
                rpc_motor_b  = (MPTR->mb.mot_drive - 2048) * k1;
                total_motor_a = total_motor_a + rpc_motor_a;
                total_motor_b = total_motor_b + rpc_motor_b;
                MPTR->ma.res_ang = total_motor_a;
                MPTR->mb.res_ang = total_motor_b;
                MPTR->ma.tach = (int) ((((rpc_motor_a * k4) / k5) / k2));
                MPTR->mb.tach = (int) ((((rpc_motor_b * k4) / k5) / k2));
                if (MPTR->ma.tach > 4096) MPTR->ma.tach = 4096;
                else if (MPTR->ma.tach < 0) MPTR->ma.tach = 0;
                if (MPTR->mb.tach > 4096) MPTR->mb.tach = 4096;
                else if (MPTR->mb.tach < 0) MPTR->mb.tach = 0;
                theta_a      = total_motor_a * 360;
                theta_b      = total_motor_b * 360;
                theta_gear_a = 1.0 * theta_a / gear;
                theta_gear_b = 1.0 * theta_b / gear;
                theta_py_a   = theta_gear_a / py_gear;
                theta_py_b   = theta_gear_b / py_gear;
                pitch_angle  = (theta_py_a - theta_py_b) / 2;
                yaw_angle    = (theta_py_a + theta_py_b) / 2;
                MPTR->py.pitch_pos  = (unsigned int) (pitch_angle / k3) + MPTR->
sp.pitch_neutral;
                MPTR->py.yaw_pos    = (unsigned int) (yaw_angle / k3) + MPTR->sp
.yaw_neutral;

/*              printf("lp= %u, dr= %d  %d  ta= %d %d dp= %u dy= %u pi= %u  ya=
%u\n", MPTR->ds.counter, MPTR->ma.mot_drive, MPTR->mb.mot_drive, MPTR->ma.tach,
MPTR->mb.tach, MPTR->ds.pitch_pos, MPTR->ds.yaw_pos, MPTR->py.pitch_pos, MPTR->p
y.yaw_pos);
                scanf("%c", &dump);          */
        }
```

```
/************************
 *       FILE: NASA7.C
 *       DATE: August 14, 1990
 *
 *    ROUTINES
 *       Joy_Motion_Done
 *       Joy_Error_Check
 *       Get_Joy_Position
 *       Read_Joystick
 *       Joystick_Operation
 *
 *
 ************************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>

/************************
 *  DATA
 ************************/

 int   view_flag;
 int   neutral_x;
 int   neutral_y;
 int   jad_check;
 int   button_1;

/************************
 *  routine to check when joint motion is done
 *  joystick is no longer active or on commanded position
 *  as been reached
 ************************/

void Joy_Motion_Done(void)
        {
        if (MPTR->jy.button_2)
                {
                RUN_ARM = FALSE;
                ERROR_FLAG = NORMAL_SHUTDOWN;
                }
        else if ((ERROR_FLAG >= P_HI_LIMIT) && (ERROR_FLAG <= Y_LO_LIMIT))
                RUN_ARM = FALSE;
        }

/************************
 *  routine to check for proper arm (joint) operation
 *  check motor and joint limits
 ************************/

void Joy_Error_Check(void)
```

```
	{
	if (MPTR->py.pitch_pos > MPTR->sp.pitch_hi_limit)  /*check for*/
		ERROR_FLAG = P_HI_LIMIT;                        /*fatal errors*/
	if (MPTR->py.pitch_pos < MPTR->sp.pitch_lo_limit)
		ERROR_FLAG = P_LO_LIMIT;
	if (MPTR->py.yaw_pos > MPTR->sp.yaw_hi_limit)
		ERROR_FLAG = Y_HI_LIMIT;
	if (MPTR->py.yaw_pos < MPTR->sp.yaw_lo_limit)
		ERROR_FLAG = Y_LO_LIMIT;
	if (MPTR->py.pitch_pos >= MPTR->sp.pitch_hi_alarm)
		{
		MPTR->sp.slimit_pitch = 1;
		ERROR_FLAG = SOFT_LIMIT_P;
		}
	else if (MPTR->py.pitch_pos <= MPTR->sp.pitch_lo_alarm)
		{
		MPTR->sp.slimit_pitch = -1;
		ERROR_FLAG = SOFT_LIMIT_P;
		}
	else MPTR->sp.slimit_pitch = FALSE;
	if (MPTR->py.yaw_pos >= MPTR->sp.yaw_hi_alarm)
		{
		MPTR->sp.slimit_yaw = 1;
		ERROR_FLAG = SOFT_LIMIT_Y;
		}
	else if (MPTR->py.yaw_pos <= MPTR->sp.yaw_lo_alarm)
		{
		MPTR->sp.slimit_yaw = -1;
		ERROR_FLAG = SOFT_LIMIT_Y;
		}
	else MPTR->sp.slimit_yaw = FALSE;
	}

/************************
 *  routine to get initial position of the joint
 ************************/

void Get_Joy_Position(void)
	{
	int gameport = 0x0201;
	int counter_x, counter_y;
	int total_x, total_y;
	int loop;
	unsigned char  game;
	unsigned char  mask_x = 0x01;
	unsigned char  mask_y = 0x02;

	total_x = 0;
	total_y = 0;
	for (loop = 0; loop <100; ++loop)
		{
		counter_x = 0;
		counter_y = 0;
		outportb(gameport, 0x03);
		do
			{
			game = inportb(gameport);
			if (game & mask_x) counter_x += 1;
			if (game & mask_y) counter_y += 1;
			}
		while ((game & mask_x) || (game & mask_y));
		total_x = total_x + counter_x;
		total_y = total_y + counter_y;
		}
	neutral_x = total_x / 100;
	neutral_y = total_y / 100;
```

```
/*printf("neutral x = %d  y = %d \n", neutral_x, neutral_y);*/
/*      neutral_x = 22;*/                              /*just set to mid position*/
/*      neutral_y = 22;*/
/*printf("neutral x = %d  y = %d \n", neutral_x, neutral_y);*/
        MPTR->ds.pitch_pos = MPTR->py.pitch_pos;
        MPTR->ds.yaw_pos   = MPTR->py.yaw_pos;
        MPTR->od.pitch_pos = MPTR->ds.pitch_pos;
        MPTR->od.yaw_pos   = MPTR->ds.yaw_pos;
        MPTR->ds.pitch_start = MPTR->py.pitch_pos;
        MPTR->ds.yaw_start   = MPTR->py.yaw_pos;
        MPTR->jy.total_x = 0;
        MPTR->jy.total_y = 0;
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(50,24);
        cputs("WORKING");
        }


/***********************
 * routine to read joystick and convert to joint angles
 ***********************/

void Read_Joystick(void)
        {
        int gameport = 0x0201;
        int deadband  = 10;
        int counter_x, counter_y;
        unsigned char  game;
        unsigned char  mask_x = 0x01;
        unsigned char  mask_y = 0x02;
        unsigned char  mask_1 = 0x10;
        unsigned char  mask_2 = 0x20;
        float  temp;

        counter_x = 0;
        counter_y = 0;
        button_1  = FALSE;
        MPTR->jy.button_2  = FALSE;
        outportb(gameport, 0x03);
        do
                {
                game = inportb(gameport);
                if (game & mask_x) counter_x += 1;
                if (game & mask_y) counter_y += 1;
                }
        while ((game & mask_x) || (game & mask_y));
/*printf("x = %d   y = %d \n", counter_x, counter_y);*/
        if (game & mask_1) button_1 = FALSE;
        else button_1 = TRUE;
        if (game & mask_2) MPTR->jy.button_2 = FALSE;
        else MPTR->jy.button_2 = TRUE;

        if (button_1)
                {
                if (view_flag == TRUE)
                        {
                        view_flag = FALSE;
                        Unlock_Brakes();
                        }
                counter_x = counter_x - neutral_x;
                counter_y = counter_y - neutral_y;
/*printf("x = %d   y = %d \n", counter_x, counter_y);*/
                if ((abs(counter_y)) > deadband)
                        {
                        if (counter_y > 0)
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
                                        {
                                        MPTR->jy.total_y = MPTR->jy.total_y + ((float) (
counter_y /* - deadband*/) * MPTR->jy.y_gain);
                                        temp = MPTR->jy.total_y + (float) MPTR->ds.pitch
_start;
                                        MPTR->ds.pitch_pos = (int) temp;
                                        }
                                else if (counter_y < 0)
                                        {
                                        MPTR->jy.total_y = MPTR->jy.total_y + ((float) (
counter_y /*+ deadband*/) * MPTR->jy.y_gain);
                                        temp = MPTR->jy.total_y + (float) MPTR->ds.pitch
_start;
                                        MPTR->ds.pitch_pos = (int) temp;
                                        }
                                }
                        else
                                {
                                temp = MPTR->jy.total_y + (float) MPTR->ds.pitch_start;
                                MPTR->ds.pitch_pos = (int) temp;
                                }
/*printf("p_pos = %d ", MPTR->ds.pitch_pos);*/
                        if ((abs(counter_x)) > deadband)
                                {
                                if (counter_x > 0)
                                        {
                                        MPTR->jy.total_x = MPTR->jy.total_x + ((float) (
counter_x /*- deadband*/) * MPTR->jy.x_gain);
                                        temp = MPTR->jy.total_x + (float) MPTR->ds.yaw_s
tart;
                                        MPTR->ds.yaw_pos = (int) temp;
                                        }
                                else if (counter_x < 0)
                                        {
                                        MPTR->jy.total_x = MPTR->jy.total_x + ((float) (
counter_x /*+ deadband*/) * MPTR->jy.x_gain);
                                        temp = MPTR->jy.total_x + (float) MPTR->ds.yaw_s
tart;
                                        MPTR->ds.yaw_pos = (int) temp;
                                        }
                                }
                        else
                                {
                                temp = MPTR->jy.total_x + (float) MPTR->ds.yaw_start;
                                MPTR->ds.yaw_pos = (int) temp;
                                }
/*printf("y_pos = %d \n", MPTR->ds.yaw_pos);*/
                        }
                else
                        {
                        if (!view_flag)
                                {
                                Lock_Brakes();
                                textcolor(BLACK);
                                textbackground(WHITE);
                                gotoxy(15,22);
                                printf("%.3f",MPTR->py.pitch_ang);
                                gotoxy(15,23);
                                printf("%.3f",MPTR->py.yaw_ang);
                                textcolor(WHITE);
                                textbackground(BLACK);
                                view_flag = TRUE;
                                }
                        }
                MPTR->ds.pitch_vel = (int)(MPTR->ds.pitch_pos - MPTR->od.pitch_pos);
                MPTR->ds.yaw_vel   = (int)(MPTR->ds.yaw_pos - MPTR->od.yaw_pos);
```

```
        MPTR->od.pitch_pos = MPTR->ds.pitch_pos;
        MPTR->od.yaw_pos   = MPTR->ds.yaw_pos;
/*printf(" p = %u    y = %u \n", MPTR->ds.pitch_pos, MPTR->ds.yaw_pos);*/
        }

/************************
 *  moves the arm with the joystick
 *  inputting command values
 ************************/

void Joystick_Operation(void)
        {
        Set_Clock_Up();
        Enable_AD_Interrupt();
        Get_Joint_Data();
        jad_check = 0;
        Initial_System();
        Get_Joy_Position();
        while (RUN_ARM)
                {
                jad_check++;
                if (jad_check > 15000)
                        {
                        ERROR_FLAG = AD_FAIL;
                        RUN_ARM = FALSE;
                        }
                while (START_LOOP)
                        {
                        Get_Joint_Data();
                        Read_Joystick();
                        Joy_Error_Check();
                        if (button_1)
                        {
                                Calculate_Drive();
                                Motor_Drive();
                        }
                        Joy_Motion_Done();
                        Data_Record();
                        jad_check = 0;
                        START_LOOP = FALSE;
                        }
                }
        Lock_Brakes();
        Unset_Clock();
        Display_Error(TRUE);
        Data_Save();
        }
```

```
/***********************
 *      FILE: NASA8.C
 *      DATE: December 20, 1990
 *
 *   ROUTINES
 *      M_Motion_Done
 *      M_Get_Command_Position
 *      M_Error_Check
 *      M_Calculate_Position
 *      M_Calculate_Drive
 *      M_Command_Operation
 *      M_Save_Path_File();
 *      M_Get_Path_File();
 *
 *
 ***********************/

#include "extdata.h"
#include "nasadef.h"
#include <dos.h>
#include <conio.h>
#include <stdio.h>

/***********************
 *  DATA
 ***********************/

 int            m_soft_pitch_flag;
 int            m_soft_yaw_flag;
 int            m_ad_check;
 int            mt_loop;
 int                      junk;

/***********************
 *  routine to check when joint motion is done
 *  on commanded position as been reached
 ***********************/

void M_Motion_Done(void)
        {
        int     checkp, checky;

        if ((abs(MPTR->py.pitch_pos - mt_final_pitch[mt_loop])) <= MPTR->sp.delt
a_pitch)
                checkp = TRUE;
        else checkp = FALSE;
        if ((abs(MPTR->py.yaw_pos - mt_final_yaw[mt_loop])) <= MPTR->sp.delta_ya
w)
                checky = TRUE;
        else checky = FALSE;
/*test*/
/*      checkp = FALSE;
checky = FALSE;
if (junk == 250)
```

```
{
checkp = TRUE;
checky = TRUE;
}    */
        if (checkp && checky)
                {
                RUN_ARM = FALSE;
                ERROR_FLAG = NORMAL_SHUTDOWN;
                mt_loop = mt_loop + 1;
                }
        }


/***********************
 *  routine to get joint path
 *
 ***********************/

void M_Get_Path_File(void)
        {
        FILE      *param_file, *fopen();
        char      name[12];
        int       loop;
        int       itmp;
        float     ftmp;
        unsigned short     ustmp;

        gotoxy(20,15);
        cputs("ENTER IN FILE NAME                                   ");
        gotoxy(45,15);
        flushall();
        scanf("%12s", name);
        param_file = fopen(name, "r");
        if (param_file == NULL)
                {
                gotoxy(50,24);
                cputs("FILE ERROR");
                sound(500);
                delay(1500);
                nosound();
                delay(7500);
                }
        gotoxy(50,24);
        printf("WORKING");
        fscanf(param_file, "%d", &itmp);
        mt_moves = itmp;
/*printf("moves = %d\n", mt_moves);
delay(2000);*/
        for (loop = 1; loop <= mt_moves; ++loop)
                {
                fscanf(param_file, "%u", &ustmp);
                mt_final_pitch[loop]      = ustmp;
                fscanf(param_file, "%u", &ustmp);
                mt_final_yaw[loop]        = ustmp;
                fscanf(param_file, "%u", &ustmp);
                mt_steps[loop]            = ustmp;
                fscanf(param_file, "%d", &itmp);
                mt_delta_pitch_pos[loop] = itmp;
                fscanf(param_file, "%d", &itmp);
                mt_delta_yaw_pos[loop]   = itmp;
                fscanf(param_file, "%f", &ftmp);
                mt_pitch_angle[loop]     = ftmp;
                fscanf(param_file, "%f", &ftmp);
                mt_yaw_angle[loop]       = ftmp;
                fscanf(param_file, "%f", &ftmp);
                mt_pitch_step[loop]      = ftmp;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
                fscanf(param_file, "%f", &ftmp);
                mt_yaw_step[loop]       = ftmp;
                fscanf(param_file, "%f", &ftmp);
                mt_time[loop]           = ftmp;
/*      printf("final pitch = %u\n", mt_final_pitch[loop]);
printf("final yaw   = %u\n", mt_final_yaw[loop]);
printf("steps = %u\n",mt_steps[loop]);
printf("delta pitch = %d\n", mt_delta_pitch_pos[loop]);
printf("delta yaw   = %d\n", mt_delta_yaw_pos[loop]);
printf("pitch angle = %f\n", mt_pitch_angle[loop]);
printf("yaw angle   = %f\n", mt_yaw_angle[loop]);
printf("pitch step = %f\n", mt_pitch_step[loop]);
printf("yaw step   = %f\n", mt_yaw_step[loop]);
printf("time = %f\n", mt_time[loop]);
delay(2000);*/
                }
        fclose(param_file);
        }

/************************
 *  routine to save joint path
 *
 ************************/

void M_Save_Path_File(void)
        {
        FILE        *param_file, *fopen();
        char        name[12];
        int         loop;

        gotoxy(20,15);
        cputs("ENTER IN FILE NAME                              ");
        gotoxy(45,15);
        flushall();
        scanf("%12s", name);
        param_file = fopen(name, "w");
        if (param_file == NULL)
                {
                gotoxy(50,24);
                cputs("FILE ERROR");
                sound(500);
                delay(1500);
                nosound();
                delay(7500);
                }
        gotoxy(50,24);
        printf("WORKING");
        fprintf(param_file, "%d\n", mt_moves);
        for (loop = 1; loop <= mt_moves; ++loop)
                {
                fprintf(param_file, "%u\n", mt_final_pitch[loop]);
                fprintf(param_file, "%u\n", mt_final_yaw[loop]);
                fprintf(param_file, "%u\n", mt_steps[loop]);
                fprintf(param_file, "%d\n", mt_delta_pitch_pos[loop]);
                fprintf(param_file, "%d\n", mt_delta_yaw_pos[loop]);
                fprintf(param_file, "%f\n", mt_pitch_angle[loop]);
                fprintf(param_file, "%f\n", mt_yaw_angle[loop]);
                fprintf(param_file, "%f\n", mt_pitch_step[loop]);
                fprintf(param_file, "%f\n", mt_yaw_step[loop]);
                fprintf(param_file, "%f\n", mt_time[loop]);
                }
        fclose(param_file);
        }


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

```
 *  routine to get desired position of the joint
 ************************/

void M_Get_Command_Position(void)
        {
        float           value;
        float           low_limit;
        float           high_limit;
        int             moves;
        short           temp;
        unsigned int    steps;
        int             flag;
        int             loop;
        char            answer;
        unsigned int    temp_pitch[20];
        unsigned int    temp_yaw[20];

        mt_counter = 0;
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(20,15);
        cputs("ENTER BY FILE OR HAND (F or H)       ");
        gotoxy(52,15);
        flushall();
        answer = fgetchar();
        if ((answer == 'H') || (answer == 'h'))
        {
                do
                {
                        flag = TRUE;
                        gotoxy(20,15);
                        cputs("ENTER NUMBER OF DESIRED MOTION STEPS (MAX 10)
    ");
                        gotoxy(68,15);
                        scanf("%d", &moves);
                        if (moves > 0) flag = FALSE;
                }
                while (flag);
                mt_moves = moves;
                temp_pitch[1] = MPTR->py.pitch_pos;
                temp_yaw[1]   = MPTR->py.yaw_pos;
                textcolor(WHITE);
                textbackground(BLACK);
                gotoxy(20,15);
                clreol();
                textcolor(BLACK);
                textbackground(WHITE);
                for (loop = 1; loop <= mt_moves; ++loop)
                {
                        gotoxy(20,15);
                        printf("ENTER VALUES FOR MOTION %d    ",loop);
                        do
                        {
                                flag = TRUE;
                                gotoxy(20,16);
                                cputs("  ENTER DESIRED MOVE TIME (seconds)
    ");
                                gotoxy(58,16);
                                scanf("%f", &value);
                                if (value > 0.0) flag = FALSE;
                        }
                        while (flag);
                        mt_time[loop] = value;
                        do
                        {
                                flag = TRUE;
```

```
                                        gotoxy(20,17);
                                        cputs("  ENTER IN DESIRED PITCH ANGLE
     ");
                                        gotoxy(58,17);
                                        scanf("%f", &value);
                                        low_limit = ((float)(MPTR->sp.pitch_lo_alarm - M
PTR->sp.pitch_neutral) / 65536.0);
                                        low_limit = (low_limit * 360.0) - 360.0;
                                        high_limit = ((float)(MPTR->sp.pitch_hi_alarm -
MPTR->sp.pitch_neutral) / 65536.0);
                                        high_limit = high_limit * 360.0;
                                        if ((value < high_limit) && (value > low_limit))
 flag = FALSE;
                                }
                                while (flag);
                                mt_pitch_angle[loop] = value;
                                value = (value * 65536.0) / 360.0;
                                mt_delta_pitch_pos[loop]   = (int) value;
                                mt_final_pitch[loop] = (unsigned int) (MPTR->sp.pitch_ne
utral + mt_delta_pitch_pos[loop]);
                                do
                                {
                                        flag = TRUE;
                                        gotoxy(20,18);
                                        cputs("  ENTER IN DESIRED YAW ANGLE
     ");
                                        gotoxy(58,18);
                                        scanf("%f", &value);
                                        low_limit = ((float)(MPTR->sp.yaw_lo_alarm - MPT
R->sp.yaw_neutral) / 65536.0);
                                        low_limit = (low_limit * 360.0) - 360.0;
                                        high_limit = ((float)(MPTR->sp.yaw_hi_alarm - MP
TR->sp.yaw_neutral) / 65536.0);
                                        high_limit = high_limit * 360.0;
                                        if ((value < high_limit) && (value > low_limit))
 flag = FALSE;
                                }
                                while (flag);
                                mt_yaw_angle[loop] = value;
                                value = (value * 65536.0) / 360.0;
                                mt_delta_yaw_pos[loop]   = (int) value;
                                mt_final_yaw[loop] = (unsigned int)(MPTR->sp.yaw_neutral
 + mt_delta_yaw_pos[loop]);
                                mt_steps[loop] = (unsigned int) (mt_time[loop] * (float)
FREQUENCY);
/* printf("mt_final_pitch = %f  temp_pitch = %f \n",(float)mt_final_pitch[loop],
 (float)temp_pitch[loop]);
printf("mt_final_yaw   = %f  temp_yaw   = %f \n",(float)mt_final_yaw[loop], (flo
at)temp_yaw[loop]);*/
                                mt_pitch_step[loop] = (float) (((float)mt_final_pitch[lo
op] - (float)temp_pitch[loop]) / (float)mt_steps[loop]);
                                mt_yaw_step[loop]   = (float) (((float)mt_final_yaw[loop
] - (float)temp_yaw[loop]) / (float)mt_steps[loop]);
                                temp_pitch[loop + 1] = mt_final_pitch[loop];
                                temp_yaw[loop + 1]   = mt_final_yaw[loop];
/*printf("loop = %d\n",loop);
printf("mt_steps  = %u\n", mt_steps[loop]);
printf("mt_final_pitch = %u  mt_pitch_step = %f\n",mt_final_pitch[loop], mt_pitc
h_step[loop]);
printf("mt_final_yaw   = %u  mt_yaw_step   = %f\n",mt_final_yaw[loop], mt_yaw_st
ep[loop]);
printf("temp_pitch     = %u  temp_yaw      = %u\n",temp_pitch[loop+1], temp_yaw[
loop+1]);*/
                        }
                        textcolor(WHITE);
                        textbackground(BLACK);
```

```
                gotoxy(20,15);
                clreol();
                gotoxy(20,16);
                clreol();
                gotoxy(20,17);
                clreol();
                gotoxy(20,18);
                clreol();
                textcolor(BLACK);
                textbackground(WHITE);
                gotoxy(20,15);
                cputs("SAVE PATH IN A FILE YES or NO (Y or N)                  ");
                gotoxy(60,15);
                flushall();
                answer = fgetchar();
                if ((answer == 'Y') || (answer == 'y'))
                        M_Save_Path_File();
        }
        else
                {
                M_Get_Path_File();
                }
        flushall();
        mt_pitch_pos = MPTR->py.pitch_pos;
        mt_yaw_pos   = MPTR->py.yaw_pos;
        MPTR->od.pitch_pos = mt_pitch_pos;
        MPTR->od.yaw_pos   = mt_yaw_pos;
        mt_pitch_start = MPTR->py.pitch_pos;
        mt_yaw_start   = MPTR->py.yaw_pos;
        textcolor(BLACK);
        textbackground(WHITE);
        gotoxy(50,24);
        cputs("WORKING");
        textcolor(WHITE);
        textbackground(BLACK);
        gotoxy(20,15);
        clreol();
        gotoxy(20,16);
        clreol();
        gotoxy(20,17);
        clreol();
        gotoxy(20,18);
        clreol();
        }


/************************
 *  routine to check for proper arm (joint) operation
 *  check motor and joint limits
 ************************/

void M_Error_Check(void)
        {
        if (MPTR->py.pitch_pos > MPTR->sp.pitch_hi_limit)  /*check for*/
                ERROR_FLAG = P_HI_LIMIT;                       /*fatal errors*/
        if (MPTR->py.pitch_pos < MPTR->sp.pitch_lo_limit)
                ERROR_FLAG = P_LO_LIMIT;
        if (MPTR->py.yaw_pos > MPTR->sp.yaw_hi_limit)
                ERROR_FLAG = Y_HI_LIMIT;
        if (MPTR->py.yaw_pos < MPTR->sp.yaw_lo_limit)
                ERROR_FLAG = Y_LO_LIMIT;
        if (ERROR_FLAG != FALSE) RUN_ARM = FALSE;
        if (MPTR->py.pitch_pos >= MPTR->sp.pitch_hi_alarm)
                MPTR->sp.slimit_pitch = 1;
        if (MPTR->py.pitch_pos <= MPTR->sp.pitch_lo_alarm)
                MPTR->sp.slimit_pitch = -1;
```

```
	if (MPTR->py.yaw_pos >= MPTR->sp.yaw_hi_alarm)
		MPTR->sp.slimit_yaw = 1;
	if (MPTR->py.yaw_pos <= MPTR->sp.yaw_lo_alarm)
		MPTR->sp.slimit_yaw = -1;
	}


/************************
 *  routine to find the next desired joint position
 *  input from commanded position or the joystick
 ************************/

void M_Calculate_Position(void)
	{
	float	temp;

	if (mt_counter <= mt_steps[mt_loop])
		{
		temp = (mt_pitch_step[mt_loop] * (float) mt_counter) + (float) m
t_pitch_start;
		mt_pitch_pos = (int) temp;
		MPTR->ds.pitch_pos = mt_pitch_pos;
		temp = (mt_yaw_step[mt_loop] * (float) mt_counter) + (float) mt_
yaw_start;
		mt_yaw_pos    = (int) temp;
		MPTR->ds.yaw_pos = mt_yaw_pos;
/*	printf("%u   %u\n",mt_pitch_pos,mt_yaw_pos);	*/
		}

	mt_counter += 1;

	if (mt_counter == (2 * mt_steps[mt_loop]))
		{
		RUN_ARM = FALSE;
		if (MPTR->sp.slimit_pitch != 0) ERROR_FLAG = SOFT_LIMIT_P;
		else if (MPTR->sp.slimit_yaw != 0) ERROR_FLAG = SOFT_LIMIT_Y;
		else	ERROR_FLAG = TIME_OUT;
		mt_loop = mt_loop + 1;
		}

	mt_pitch_vel = (int)(mt_pitch_pos - MPTR->od.pitch_pos);
	mt_yaw_vel   = (int)(mt_yaw_pos - MPTR->od.yaw_pos);
	MPTR->od.pitch_pos = mt_pitch_pos;
	MPTR->od.yaw_pos   = mt_yaw_pos;
	}


/************************
 *  routine to find the motor input (drive) values based
 *  on system conditions, angles desired and angles true
 ************************/

void M_Calculate_Drive(void)
	{
	float	diffa, diffb;
	float	diffav, diffbv;
	float	tacha, tachb;
	float	drivea, driveb;
	int	pitch, yaw;
	int	pitchv, yawv;


	/*find if soft limits are in effect*/
	if (MPTR->sp.slimit_pitch != FALSE)
		{
```

```
                if (m_soft_pitch_flag == 0)
                        {
                        m_soft_pitch_flag = 1;
                        MPTR->od.slimit_pitch = MPTR->py.pitch_pos;
                        }
                else if (MPTR->sp.slimit_pitch == 1)
                        {
                        if (MPTR->py.pitch_pos >= MPTR->od.slimit_pitch) mt_pitc
h_pos = MPTR->od.slimit_pitch;
                        }
                else
                        {
                        if (mt_pitch_pos <= MPTR->od.slimit_pitch) mt_pitch_pos
= MPTR->od.slimit_pitch;
                        }
                }
        else m_soft_pitch_flag = 0;
        if (MPTR->sp.slimit_yaw != FALSE)
                {
                if (m_soft_yaw_flag == 0)
                        {
                        m_soft_yaw_flag = 1;
                        MPTR->od.slimit_yaw = mt_yaw_pos;
                        }
                else if (MPTR->sp.slimit_yaw == 1)
                        {
                        if (mt_yaw_pos >= MPTR->od.slimit_yaw) mt_yaw_pos = MPTR
->od.slimit_yaw;
                        }
                else
                        {
                        if (mt_yaw_pos <= MPTR->od.slimit_yaw) mt_yaw_pos = MPTR
->od.slimit_yaw;
                        }
                }
        else m_soft_yaw_flag = 0;

        /*find motor A drive signal*/
        /*find motor B drive signal*/

        pitch  = mt_pitch_pos - MPTR->py.pitch_pos;
        yaw    = MPTR->py.yaw_pos - mt_yaw_pos;
        pitchv = mt_pitch_vel - MPTR->py.pitch_vel;
        yawv   = mt_yaw_vel - MPTR->py.yaw_vel;
        diffa  = -(MPTR->gn.A_position * (float) (pitch + yaw));
        diffb  = MPTR->gn.B_position * (float) (pitch - yaw);
        diffav = (MPTR->gn.A_output_vel * (float) (pitchv + yawv));
        diffbv = (MPTR->gn.B_output_vel * (float) (pitchv - yawv));
        tacha  = MPTR->gn.A_motor_vel * (float) (MPTR->ma.tach);
        tachb  = MPTR->gn.B_motor_vel * (float) (MPTR->mb.tach);
        drivea = diffa + diffav - tacha;
        driveb = diffb + diffbv - tachb;
/*printf("drivea/b   %f   %f\n", drivea, driveb);*/
        MPTR->ma.mot_drive = (((int) (drivea)) * MPTR->gn.direction) + 2048;
        MPTR->mb.mot_drive = (((int) (driveb)) * MPTR->gn.direction) + 2048;
/*printf("dfa %f dfb %f dfav %f dfbv %f dva %f dvb %f\n",diffa,diffb,diffav,diff
bv,drivea,driveb);*/
/*printf("drive a %d  b %d\n",MPTR->ma.mot_drive, MPTR->mb.mot_drive);*/
        }


/***********************
 *  performs the moving of the arm
 *  given the final pitch and yaw angles
 *  will check for errors and final position
 ***********************/
```

```
void M_Command_Operation(void)
        {
        Set_Clock_Up();
        Enable_AD_Interrupt();
        Initial_System();
        M_Get_Command_Position();
        m_ad_check = 0;
        Unlock_Brakes();
        mt_loop = 1;
        do
                {
                mt_counter      = 0;
                mt_pitch_pos    = MPTR->py.pitch_pos;
                mt_yaw_pos      = MPTR->py.yaw_pos;
                MPTR->od.pitch_pos    = mt_pitch_pos;
                MPTR->od.yaw_pos      = mt_yaw_pos;
                mt_pitch_start = MPTR->py.pitch_pos;
                mt_yaw_start   = MPTR->py.yaw_pos;
                RUN_ARM = TRUE;
                ERROR_FLAG = FALSE;
                while (RUN_ARM)
                        {
                        m_ad_check++;
                        if (m_ad_check > 15000)
                                {
                                ERROR_FLAG = AD_FAIL;
                                RUN_ARM = FALSE;
                                mt_loop = mt_moves + 1;
                                }
                        while (START_LOOP)
                                {
                                Get_Joint_Data();
                                M_Calculate_Position();
                                M_Error_Check();
                                M_Calculate_Drive();
                                Motor_Drive();
                                M_Motion_Done();
                                Data_Record();
                                m_ad_check = 0;
                                START_LOOP = FALSE;
                                }
                        }
                }
        while (mt_loop <= mt_moves);
        Lock_Brakes();
        Unset_Clock();
        Display_Error(TRUE);
        Data_Save();
        }
```

```
/************************
 *       FILE: NASADATA.H
 *       DATE: July 28, 1990
 *
 *
 *
 ************************/

int  OPERATE;           /*flag to stop program*/

int  START_LOOP;        /*flag for sync operations */

int  RUN_ARM;           /*flag to allow arm operations*/

int  ERROR_FLAG;        /*flag to hold the cause of the shutdown*/

int  PARAMETER_FLAG;    /*flag to inform system of data parameter change*/

int  LOG_CNT;           /*counter for data logging*/

int  LOG_REV;           /*flag for data log looping on itself*/

struct MTR
        {
        int             tach;
        int             res_vel;
        int             torq;
        unsigned int    res_cnt;
        float           res_ang;
        unsigned int    res_rev;
        int             mot_drive;
        unsigned char   brake;
        };


struct PY
        {
        unsigned int    pitch_pos;
        int             pitch_vel;
        unsigned int    old_pitch;
        float           pitch_ang;
        unsigned int    yaw_pos;
        int             yaw_vel;
        unsigned int    old_yaw;
        float           yaw_ang;
        };

struct DESIRED
        {
        unsigned int    pitch_pos;
        unsigned int    yaw_pos;
        unsigned int    final_pitch;
        unsigned int    final_yaw;
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        unsigned int            steps;
        unsigned int            counter;
        unsigned int            pitch_start;
        unsigned int            yaw_start;
        int                     delta_pitch_pos;
        int                     delta_yaw_pos;
        float                   pitch_angle;
        float                   yaw_angle;
        float                   pitch_step;
        int                     pitch_vel;
        float                   yaw_step;
        int                     yaw_vel;
        float                   time;
        };


struct SETPOINT
        {
        unsigned int            pitch_hi_limit;
        unsigned int            pitch_hi_alarm;
        unsigned int            pitch_neutral;
        unsigned int            pitch_lo_alarm;
        unsigned int            pitch_lo_limit;
        unsigned int            delta_pitch;
        unsigned int            yaw_hi_limit;
        unsigned int            yaw_hi_alarm;
        unsigned int            yaw_neutral;
        unsigned int            yaw_lo_alarm;
        unsigned int            yaw_lo_limit;
        unsigned int            delta_yaw;
        int                     slimit_pitch;
        int                     slimit_yaw;
        };


struct GAIN
        {
        float     A_pos_motion_scale;
        float     A_vel_motion_scale;
        float     A_motor_vel;
        float     A_torque;
        float     A_position;
        float     A_output_vel;
        float     B_pos_motion_scale;
        float     B_vel_motion_scale;
        float     B_motor_vel;
        float     B_torque;
        float     B_position;
        float     B_output_vel;
        int       direction;
        };


struct OLD
        {
        unsigned int           pitch_pos;
        unsigned int           yaw_pos;
        unsigned int           slimit_pitch;
        unsigned int           slimit_yaw;
        };

struct JOY
        {
        int             button_2;
        float           total_x;
        float           total_y;
```

```
        float       x_gain;
        float       y_gain;
        };


struct MAIN
        {
        struct    MTR       ma;
        struct    MTR       mb;
        struct    PY        py;
        struct    DESIRED   ds;
        struct    SETPOINT  sp;
        struct    GAIN      gn;
        struct    OLD       od;
        struct    JOY       jy;
        }     M, *MPTR;


/************************
 *  arrays for mult moves
 ************************/

unsigned int        mt_pitch_pos;
unsigned int        mt_yaw_pos;
unsigned int        mt_final_pitch[10];
unsigned int        mt_final_yaw[10];
unsigned int        mt_steps[10];
unsigned int        mt_counter;
unsigned int        mt_pitch_start;
unsigned int        mt_yaw_start;
int                 mt_delta_pitch_pos[10];
int                 mt_delta_yaw_pos[10];
float               mt_pitch_angle[10];
float               mt_yaw_angle[10];
float               mt_pitch_step[10];
int                 mt_pitch_vel;
float               mt_yaw_step[10];
int                 mt_yaw_vel;
float               mt_time[10];
int                 mt_moves;


/************************
 *  arrays for data logging
 ************************/

#define        log_range      1250

int            log_tach_a[log_range];
int            log_tach_b[log_range];
int            log_torq_a[log_range];
int            log_torq_b[log_range];
unsigned int   log_res_cnt_a[log_range];
int            log_res_rev_a[log_range];
unsigned int   log_res_cnt_b[log_range];
int            log_res_rev_b[log_range];
int            log_motor_cmd_a[log_range];
int            log_motor_cmd_b[log_range];
unsigned int   log_pitch_pos[log_range];
int            log_pitch_vel[log_range];
unsigned int   log_yaw_pos[log_range];
int            log_yaw_vel[log_range];
int            log_desire_pitch_pos[log_range];
int            log_desire_yaw_pos[log_range];
```

```
/************************
 *      FILE: EXTDATA.H
 *      DATE: July 28, 1990
 *
 *
 *
 ************************/

extern  int  OPERATE;            /*flag to stop program*/

extern  int  START_LOOP;        /*flag for sync operations */

extern  int  RUN_ARM;           /*flag to allow arm operations*/

extern  int  ERROR_FLAG;        /*flag to hold the cause of the shutdown*/

extern  int  PARAMETER_FLAG;    /*flag to inform system of data parameter change
*/

extern  int  LOG_CNT;           /*counter for data logging*/

extern  int  LOG_REV;           /*flag for data log looping on itself*/

extern  struct MTR
        {
        int                 tach;
        int                 res_vel;
        int                 torq;
        unsigned int        res_cnt;
        float               res_ang;
        unsigned int        res_rev;
        int                 mot_drive;
        unsigned char       brake;
        };


extern  struct PY
        {
        unsigned int        pitch_pos;
        int                 pitch_vel;
        unsigned int        old_pitch;
        float               pitch_ang;
        unsigned int        yaw_pos;
        int                 yaw_vel;
        unsigned int        old_yaw;
        float               yaw_ang;
        };

extern  struct DESIRED
        {
        unsigned int        pitch_pos;
        unsigned int        yaw_pos;
        unsigned int        final_pitch;
```

```
        unsigned int            final_yaw;
        unsigned int            steps;
        unsigned int            counter;
        unsigned int            pitch_start;
        unsigned int            yaw_start;
        int                     delta_pitch_pos;
        int                     delta_yaw_pos;
        float                   pitch_angle;
        float                   yaw_angle;
        float                   pitch_step;
        int                     pitch_vel;
        float                   yaw_step;
        int                     yaw_vel;
        float                   time;
        };


extern  struct SETPOINT
        {
        unsigned int            pitch_hi_limit;
        unsigned int            pitch_hi_alarm;
        unsigned int            pitch_neutral;
        unsigned int            pitch_lo_alarm;
        unsigned int            pitch_lo_limit;
        unsigned int            delta_pitch;
        unsigned int            yaw_hi_limit;
        unsigned int            yaw_hi_alarm;
        unsigned int            yaw_neutral;
        unsigned int            yaw_lo_alarm;
        unsigned int            yaw_lo_limit;
        unsigned int            delta_yaw;
        int                     slimit_pitch;
        int                     slimit_yaw;
        };


extern  struct GAIN
        {
        float      A_pos_motion_scale;
        float      A_vel_motion_scale;
        float      A_motor_vel;
        float      A_torque;
        float      A_position;
        float      A_output_vel;
        float      B_pos_motion_scale;
        float      B_vel_motion_scale;
        float      B_motor_vel;
        float      B_torque;
        float      B_position;
        float      B_output_vel;
        int        direction;
        };


extern  struct OLD
        {
        unsigned int           pitch_pos;
        unsigned int           yaw_pos;
        unsigned int           slimit_pitch;
        unsigned int           slimit_yaw;
        };

extern  struct JOY
        {
        int             button_2;
        float           total_x;
```

```
        float         total_y;
        float         x_gain;
        float         y_gain;
        };


extern  struct MAIN
        {
        struct    MTR         ma;
        struct    MTR         mb;
        struct    PY          py;
        struct    DESIRED     ds;
        struct    SETPOINT    sp;
        struct    GAIN        gn;
        struct    OLD         od;
        struct    JOY         jy;
        }       M, *MPTR;

/************************
 *  arrays for mult moves
 ************************/

extern  unsigned int        mt_pitch_pos;
extern  unsigned int        mt_yaw_pos;
extern  unsigned int        mt_final_pitch[];
extern  unsigned int        mt_final_yaw[];
extern  unsigned int        mt_steps[];
extern  unsigned int        mt_counter;
extern  unsigned int        mt_pitch_start;
extern  unsigned int        mt_yaw_start;
extern  int                 mt_delta_pitch_pos[];
extern  int                 mt_delta_yaw_pos[];
extern  float               mt_pitch_angle[];
extern  float               mt_yaw_angle[];
extern  float               mt_pitch_step[];
extern  int                 mt_pitch_vel;
extern  float               mt_yaw_step[];
extern  int                 mt_yaw_vel;
extern  float               mt_time[];
extern  int                 mt_moves;

/************************
 *  arrays for data logging
 ************************/
#define        log_range_e

extern  int             log_tach_a[log_range_e];
extern  int             log_tach_b[log_range_e];
extern  int             log_torq_a[log_range_e];
extern  int             log_torq_b[log_range_e];
extern  unsigned int    log_res_cnt_a[log_range_e];
extern  int             log_res_rev_a[log_range_e];
extern  unsigned int    log_res_cnt_b[log_range_e];
extern  int             log_res_rev_b[log_range_e];
extern  int             log_motor_cmd_a[log_range_e];
extern  int             log_motor_cmd_b[log_range_e];
extern  unsigned int    log_pitch_pos[log_range_e];
extern  int             log_pitch_vel[log_range_e];
extern  unsigned int    log_yaw_pos[log_range_e];
extern  int             log_yaw_vel[log_range_e];
extern  int             log_desire_pitch_pos[log_range_e];
extern  int             log_desire_yaw_pos[log_range_e];
```

```c
/*************************
 *      FILE: NASADEF.H
 *      DATE: July 28, 1990
 *
 *
 *
 *************************/

#define TRUE        1
#define FALSE       0

/* A/D - D/A - I/O control definitions */

#define AD_BASE     0x330
#define AD_LOW_B    0X330
#define AD_HI_B     0x331
#define AD_SCAN     0x332
#define IO_OUT      0x333
#define IO_IN       0x333
#define DA0_LOW     0x334
#define DA0_HI      0x335
#define DA1_LOW     0x336
#define DA1_HI      0x337
#define AD_STATUS   0x338
#define AD_CON      0x339
#define AD_TIME     0x33a
#define AD_GAIN     0x33b
#define AD_CNTR_0   0x33c
#define AD_CNTR_1   0x33d
#define AD_CNTR_2   0x33e
#define ADc_CON     0x33f
#define GAIN_1      0x00
#define GAIN_500    0x03
#define INT_OFF     0x53  /*interrupt #5 & off, counter start*/
#define INT_ON      0xd3  /*interrupt #3 & on, counter start*/
#define CLR_INT     0x00  /*write to status reg only to clear*/
#define LOCK_B      0x03  /*lock the motor brakes*/
#define UNLOCK_B    0x00  /*unlock the motor brakes*/
#define INTR_LOC    0x0d  /*interrupt vector for interrupt # 5*/
#define INT_MASK    0x80  /*mask for checking EOC flag*/
#define AD_SET      0x00

/* A/D channel definitions */

#define AD_TACH_A   0x00
#define AD_TACH_B   0x11
#define AD_TORQ_A   0x22
#define AD_TORQ_B   0x33
#define AD_RD_V_A   0X44
#define AD_RD_V_B   0x55
#define AD_RD_V_P   0x66
#define AD_RD_V_Y   0x77
```

```
/* Resolver definitions */
#define BASE          0x300
#define LSB           0x300
#define MSB           0x301
#define S_C           0x302
#define ERR           0x303
#define M_A_R         0x30
#define M_B_R         0x31
#define P_LO          0x32
#define P_HI          0x33
#define Y_LO          0x34
#define Y_HI          0x35
#define LOOP_MASK     0x0f
#define LOS_MASK      0x10
#define BIT_MASK      0x20


/*shutdown definitions*/

#define   P_HI_LIMIT           0x0001
#define   P_LO_LIMIT           0x0002
#define   Y_HI_LIMIT           0x0004
#define   Y_LO_LIMIT           0x0008
#define   RESOLVER_ERROR       0x0010
#define   RESET                0x0020
#define   TIME_OUT             0x0040
#define   SOFT_LIMIT_P         0x0100
#define   SOFT_LIMIT_Y         0x0200
#define   AD_FAIL              0x0400
#define   NORMAL_SHUTDOWN      0x8000

/*speed of loop rate  per second*/

#define   FREQUENCY            250

/*defintions for finding the motor drive commands*/

#define  SCALED_REMOTE_YAW     ((MPTR->gn.pos_motion_scale * MPTR->ds.yaw_pos) /
128)

#define  SCALED_REMOTE_PITCH   ((MPTR->gn.pos_motion_scale * MPTR->ds.pitch_pos)
/ 128)

#define  DECOUPLED_POS_ERR_A   (-(SCALED_REMOTE_PITCH + SCALED_REMOTE_YAW))

#define  DECOUPLED_POS_ERR_B   (SCALED_REMOTE_PITCH - SCALED_REMOTE_YAW)

#define  SCALED_PITCH_VEL_ERR  (((MPTR->gn.vel_motion_scale * MPTR->ds.pitch_vel)
 / 128) - MPTR->py.pitch_vel)

#define  SCALED_YAW_VEL_ERR    (((MPTR->gn.vel_motion_scale * MPTR->ds.yaw_vel) /
 128) - MPTR->py.yaw_vel)

#define  DECOUPLED_VEL_ERR_A   (SCALED_YAW_VEL_ERR + SCALED_PITCH_VEL_ERR)

#define  DECOUPLED_VEL_ERR_B   (SCALED_YAW_VEL_ERR - SCALED_PITCH_VEL_ERR)

#define  SCALED_MOTOR_VEL_A    ((MPTR->gn.motor_velocity * MPTR->ma.tach) / 128)

#define  SCALED_MOTOR_VEL_B    ((MPTR->gn.motor_velocity * MPTR->mb.tach) / 128)

#define  SCALED_TORQ_A         ((MPTR->gn.torque * MPTR->ma.torq) / 128)

#define  SCALED_TORQ_B         ((MPTR->gn.torque * MPTR->mb.torq) / 128)

#define  MAX_LOG               1250
```