North Carolina State University

EXECUTIVE SUMMARY

The overall goal of "Telerobotic Control of a Mobile Coordinated Robotic Servicer" project is to develop advance control methods that would enhance the usage of robotic systems for space applications. Towards this end, several algorithms have been developed from this project. One area of development was to extend the methodology of the Observer/Kalman Filter Identification (OKID) approach, developed at NASA Langley, to such design problems as frequency spectrum reconstruction, improved parameter estimation from frequency data and recursion structures to improve computational performance. This area addressed the identification issue of systems which can then be followed by regulation design as is typical in self-tuning adaptive control. The approach has applicability to many types of systems, including robotics, when the system structure or parameter set is unknown or has variations.

The second area of control research focused on fuzzy control which is a non-parametric (non-model-based) knowledge-based approach. In this area, adaptive algorithms were developed using self-tuning scaling factor schemes in the fuzzifier, self-learning schemes in the control rulebase and optimization to extend the method to multi-input, multi-output systems. As a knowledge-based approach, the MIMO adaptive fuzzy controller uses a computationally efficient rulebase to determine control commands when the system model (the robot dynamics) is partially unknown or varies with time.

The final phase of this effort was devoted to the design, fabrication and testing of a robot manipulator arm which is attached to a mobile robotic system, a rover, built at the Mars Mission Research Center. The rover is currently under teleoperation mode and will have capabilities for full autonomy. The manipulator arm along with the mobile robotic system will be used to test all of the control algorithms that have been developed though this effort as well as other programs at the Mars Mission Research Center.

What follows is the MS thesis of Mr. Mike Brown. Mike spent a summer at NASA Langley working in the Spacecraft Dynamics Branch. His thesis develops the design and testing of the manipulator arm on the teleoperated mobile robotic system.

ABSTRACT

BROWN, JR., ROBERT MICHAEL. A Microcontroller-Based Three Degree-of-Freedom Manipulator Testbed. (Under the direction of Gordon K. F. Lee.)

A wheeled exploratory vehicle is under construction at the Mars Mission Research Center at North Carolina State University. In order to serve as more than an inspection tool, this vehicle requires the ability to interact with its surroundings. A crane-type manipulator, as well as the necessary control hardware and software, has been developed for use as a sample gathering tool on this vehicle. The system is controlled by a network of four Motorola M68HC11 microcontrollers. Control hardware and software were developed in a modular fashion so that the system can be used to test future control algorithms and hardware. Actuators include three stepper motors and one solenoid. Sensors include three optical encoders and one cable tensiometer.

The vehicle supervisor computer provides the manipulator system with the approximate coordinates of the target object. This system maps the workspace surrounding the given location by lowering the claw, along a set of evenly spaced vertical lines, until contact occurs. Based on this measured height information and prior knowledge of the target object size, the system determines if the object exists in the searched area. The system can find and retrieve a 1.25 in diameter by 1.25 in tall cylinder placed within the 47.5 in^2 search area in less than 12 minutes. This manipulator hardware may be used for future control algorithm verification and serves as a prototype for other manipulator hardware.

A MICROCONTROLLER-BASED THREE DEGREE-OF-FREEDOM MANIPULATOR TESTBED

by

ROBERT MICHAEL BROWN, JR.

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

MECHANICAL ENGINEERING

Raleigh

1995

APPROVED BY:

_____ _____

_____
Chair of Advisory Committee

BIOGRAPHY

Robert M. Brown Jr. was born in Rocky Mount, NC, on December 1, 1965, to Mike and Marie Brown. He graduated from the North Carolina School of Science and Mathematics in June 1984. While attending NCSU and taking part in the cooperative engineering program, he spent five semesters working for NASA at Wallops Island, VA. He received a B. S. of Aerospace Engineering from NCSU in May 1989. He was married to Kathy Tyndall (NCSU '89) in June 1989 after which he spent two years working for NASA at Wallops Island, VA. In May of 1991, Mr. Brown left NASA to work at the National Undersea Research Center at the University of North Carolina at Wilmington. In January 1993 he enrolled in the graduate program at NCSU.

ACKNOWLEDGMENTS

I would like to recognize my family, professors, and friends. Without the assistance and support of Kathy Tyndall Brown, Mike and Marie Brown, Dr. Gordon Lee, Dr. Larry Silverberg, Chih-Kang Chao, Keita Ikeda, and the faculty and staff of the Mars Mission Research Center this goal would have been unattainable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

pp ..... two ASCII characters representing vehicle pitch (degrees)

qq ..... two ASCII characters representing vehicle roll (degrees)

r ..... radius coordinate of claw position (inches)

rr.r ..... four ASCII characters representing the radius coordinate of claw position (inches)

z ..... height coordinate of claw position (inches)

zz.z ..... four ASCII characters representing the height coordinate of claw position (inches)

θ ..... angular coordinate of claw position (degrees)

$\theta\theta\theta$ ..... three ASCII characters representing the angular coordinate of claw position (degrees)

$( )_a$ ..... actual value

$( )_d$ ..... desired value

$( )_t$ ..... target value

Chapter One: INTRODUCTION

Robotic vehicles are ideal for the exploration of hostile environments. These devices allow humans to investigate areas that would otherwise be difficult or impossible to reach. In order to serve as more than inspection tools, these robots must have the ability to interact with their surroundings. An undersea vehicle on a scientific mission must often collect sediment and water samples[1]. A Space Station assembly vehicle must be able to position and connect building materials. An emergency response robot could open doors and move debris while searching for injured victims in a burning building. A robotic vehicle in a hazardous material spill area could be used to locate and close a critical valve.

A. Background Information

Three basic types of joints, revolute, prismatic, and suspended cable, are typically used by manipulator systems. Revolute joints, like a human elbow, rotate about an axis. Prismatic joints, like an extension ladder, extend or retract along a linear path. Suspended cable systems, used in place of rigid structural members on crane systems, also extend or retract. The key difference is that the path followed by a payload suspended by a cable is a function of gravitational effects and environmental disturbances.

A revolute joint system, such as the manipulator arm used on the NASA space shuttle, is very maneuverable. The variable direction of approach, made

possible by the slender structural members and multiple revolute joints, allows the retrieval of unsymmetrical payloads. In the weightless environment of space, the joint actuators must position the end-effector and damp unwanted motion. However, in a similar system operating vertically in a gravity field, the joint actuators must also support the manipulator structure and payload. Lower payload capacities, relative to a crane with identical actuators, result.

Prismatic joints are often used in systems where precision is more important than range of motion. Extremely fine control of an end effector trajectory is possible with rigid links and prismatic joints. This advantage is gained at the expense of mechanical complexity and additional weight. Loss of mobility also results since the distance that a joint can extend is limited by the length of the telescoping member.

Suspended cable joints, found in crane systems, are capable of extreme ranges of motion. Cable, unlike rigid members, can be stored in great lengths on winch drums. Since the structure, not the actuators, of a crane carries bending loads, relatively high payload-to-system-weight ratios can be achieved. The key disadvantages of a crane system are the difficulties in controlling all six degrees-of-freedom of the end effector and in damping undesired motion.

The National Institute of Standards and Technology (NIST) has developed a six degree-of-freedom crane called ROBOCRANE[2]. This system uses cables as structural links, winches as actuators, and cable travel encoders as sensors. A cable, a winch actuator, and an encoder are required for each controlled

degree-of-freedom to ensure a fully constrained system. This technology is applicable to various types of crane platforms, such as tower, boom, and overhead, as well as lower degree-of-freedom systems.

Depending on system requirements and research objectives, algorithms used in crane control vary from classical to discrete[3] to fuzzy logic[4] schemes. The Motorola M68HC11 is a relatively inexpensive microcontroller allowing the use of both classical[5,6] and fuzzy control techniques[7].

B. Research Objectives and Problem Development

An autonomous wheeled exploratory vehicle is currently under construction at the Mars Mission Research Center at North Carolina State University[8,9,10]. This vehicle, pictured in Figure 1-1, will be tasked with the exploration of unfamiliar terrain. In order to effectively carry out this mission, the vehicle must avoid dangers, such as boulders and crevasses, gather information, such as visual images and sensor data, and collect physical samples, such as rock and soil. The latter mission requirement makes a manipulator subsystem necessary.

The purpose of the research presented in this thesis is to develop a manipulator that serves two purposes. The first goal is to provide the required environmental sample gathering tool for a Mars vehicle prototype. The second goal is to provide a platform for future robotic manipulator research activities. A crane-type manipulator system configuration was selected to insure adequate payload capacity. Motorola microcontrollers were selected so that the prototype

payload capacity. Motorola microcontrollers were selected so that the prototype system will be capable of implementing both traditional and modern control techniques. The structural hardware, electrical hardware, and control software have been designed and constructed in a modular fashion. Future researchers will be able to further optimize the system by modifying individual hardware and software components.

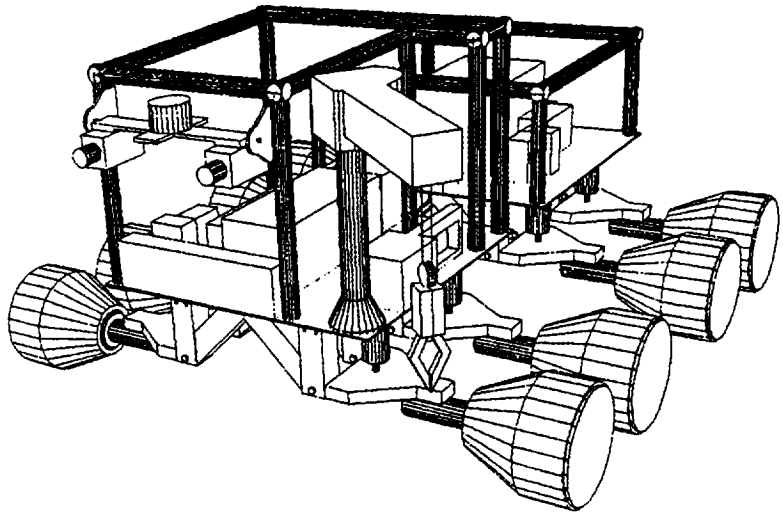


Figure 1-1: NCSU Mars Rover

Based on the existing vehicle design, the following criteria must be met by the vehicle and manipulator subsystem.

1) The vehicle must

- Provide 12 V and 48 V electrical power.

- Provide an ASCII string, via a serial link, containing manipulator platform pitch and roll as well as object location in cylindrical coordinates. The string format will be *±pp,±qq,±θθθ,±rr.r,±zz.z.*
- Disable the wheel motor subsystem during manipulator operation.
- Confirm retrieval of desired object with vehicle sensor devices, such as vision or ultrasound.

2) The manipulator system must

- Fit in a space that measures 18 in long by 8 in wide by 18 in tall.
- Weigh no more than 20 lb.
- Find and retrieve a typical environmental sample, approximated by a 1.25 in diameter by 1.25 in tall cylinder, when provided with a target location inside the workspace and within 4 in of actual object location.
- Be capable of lifting a payload weighing up to 1 lb.

C. Thesis Organization

This thesis is divided into seven chapters. Chapter 2, *System Hardware Description,* describes mechanical and electrical system components. In Chapter 3, *Control Software Description,* a discussion is presented on the software embedded in each of the four controllers. Chapter 4, *Application of Control Software to Crane System,* details the system tests and results. Finally, Chapter 5, *Conclusions and Suggestions for Future Work*, states conclusions

and offers ideas for system improvements. *Appendices* contains commented computer code for each type of controller and programming instructions.

Chapter Two: SYSTEM HARDWARE DESCRIPTION

The manipulator system is composed of the master controller, three motor controllers, three motor drivers, three stepper motors, three optical encoders, a solenoid actuated claw, a tensiometer, and the crane structure. These components are pictured in Figure 2-1 and are discussed in the following sections of this chapter.

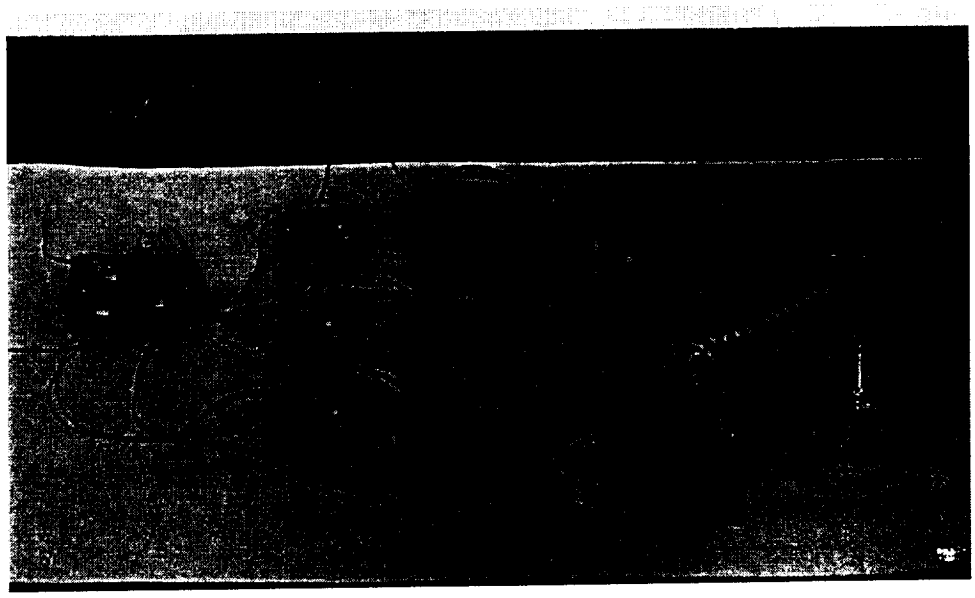


Figure 2-1: Manipulator System

A. Crane Structure

Four major components, illustrated in Figure 2-2, make up the structure of the manipulator. The *tower* is the vertical structure about which the *boom* pivots. The lower flange of the boom acts as a track for the *trolley*. The *claw* is suspended on a aramid fiber cable from the trolley. The tower, boom, and trolley

are constructed of readily available components to allow modification by future users. The main structural components are formed from prefabricated fiberglass I-beam, channel, and angle stock. The boom and cable drum are driven directly by their respective motors. The trolley position is controlled via a chain drive with the third motor.

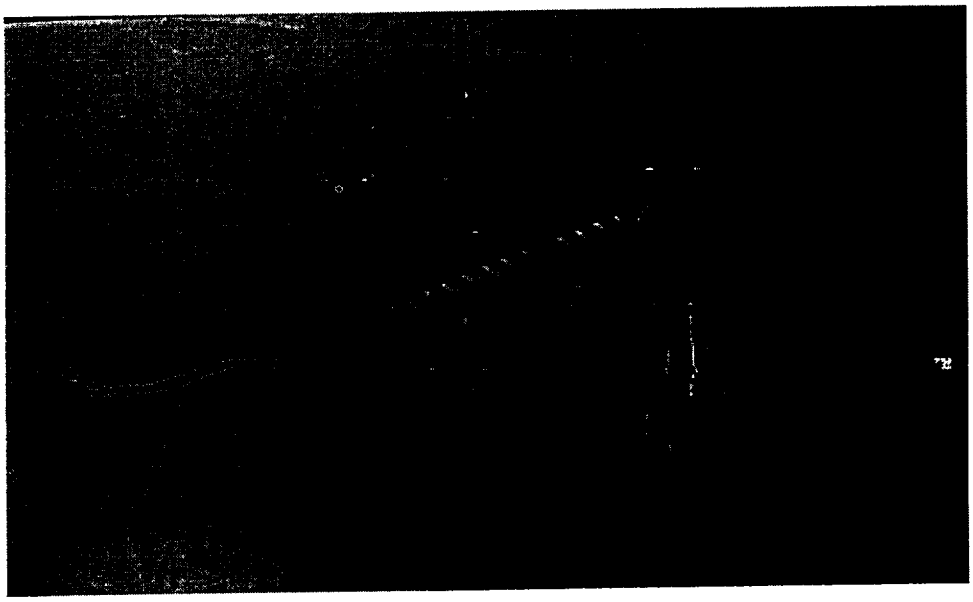


Figure 2-2: Crane Structure

B. Coordinate System and Workspace

The three coordinates used to describe the position of the closed manipulator claw tips are illustrated in Figure 2-3. The radius (r) and angle (θ) are standard polar coordinates when the system is viewed from above. The radius is measured from the rotational axis to the trolley center. The angle is measured counter-clockwise from home position. Note that the angle illustrated in Figure 2-3 is in the negative direction. The height (z) is the distance from the

baseplane to the claw tips, where a positive value of z is used for points above the plane.

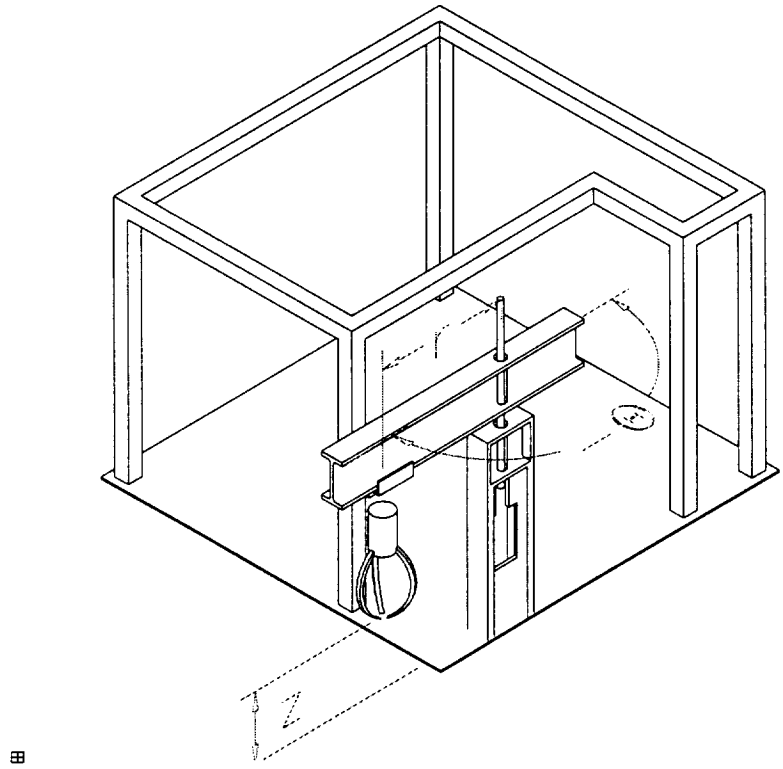


Figure 2-3: Coordinate System

C. Actuators

Stepper Motors and Drivers

The three positioning actuators in this device are Pacific Scientific Powermax P21NRXA-LDF-M1-00 stepper motors. Each is driven by a Pacific Scientific Sigma Model 5210 motor driver. Each motor requires 2.5 A at 12 V and provides a holding torque of 114 oz-in. In addition to power, the motor driver requires two logic inputs. The level of the direction input determines the direction

of rotation. A square wave applied to the second input results in a motor step for every wave period.

The motor responsible for lifting the claw and captured object was selected to maximize the payload capacity of the system. Identical motors were selected to control boom and trolley location, to standardize hardware, and to increase modularity.

Solenoid Actuated Claw

The claw, illustrated in Figure 2-4, is the same type used in arcade games. A more forgiving control system and minor claw modification result in a much better success rate. The claw has three fingers, located 120° apart, that are activated by a 48V solenoid. Original tests, involving a range of object types, demonstrated that, while the claw was very effective at "scooping" up a large object, such as a four inch diameter sphere, it was not capable of holding most smaller objects in its fingertips. To include small objects in the target range, these fingers were modified by the addition of claw tips. These tips, acting as fingernails, are 0.063 inch diameter rods protruding one-quarter inch from the fingertips. They result in a great enhancement in gripping capability.

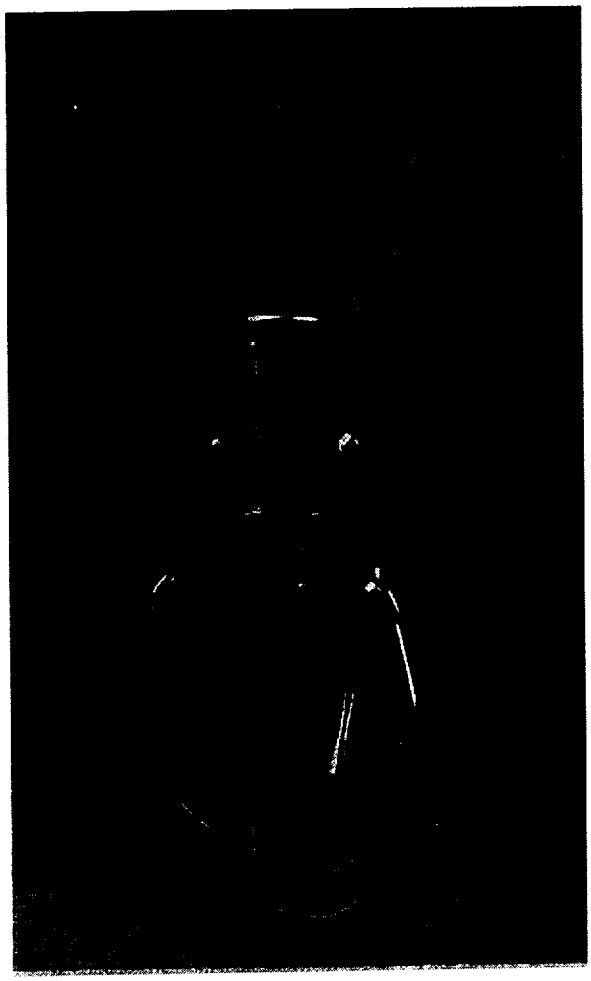


Figure 2-4: End Effector With Sample Object

D. Sensors

Optical Encoders

A U. S. Digital Model E2-512-250-IE optical encoder is mounted on the shaft of each stepper motor. The resolution of each encoder is 512 counts per revolution. An index pulse, once per revolution, allows the motor controllers to find home position from the power-on position. Sensor output consists of two

square wave signals that, except for phase, are identical. The lead-lag relationship of these two signals reflects the direction of motion of the encoder. In general, these signals can be decoded by the motor controller. However, due to high frequency "ringing" of the stepper motor after a single step command, a separate chip was used. This chip, an LSI Computer Systems LS7166 24 bit multimode counter, can accurately decode the encoder signals even with the high frequency changes in direction of encoder rotation associated with ringing. The internal 24 bit counter containing the motor position can be read by the motor controller via an 8 bit data bus. The use of this chip relieves the microcontroller of the burden of constantly monitoring the encoder output.

Cable Tensiometer

A slight loss of cable tension, such as occurs when the claw makes contact with some surface or object, causes an interrupt service routine on the master controller to be activated. A discussion of this software routine can be found in Chapter 3. Figure 2-5 illustrates the mechanical components of the sensor. The key electrical component that enables this interrupt is a conditioned single-pole double-throw (SPDT) switch. This switch is mounted on a lever whose position is controlled by the cable tension. The switch and conditioning circuit[11], detailed in Appendix A, control the state of pin PA3 on the master controller. This pin has input capture capabilities that are used to trigger the interrupt service routine.

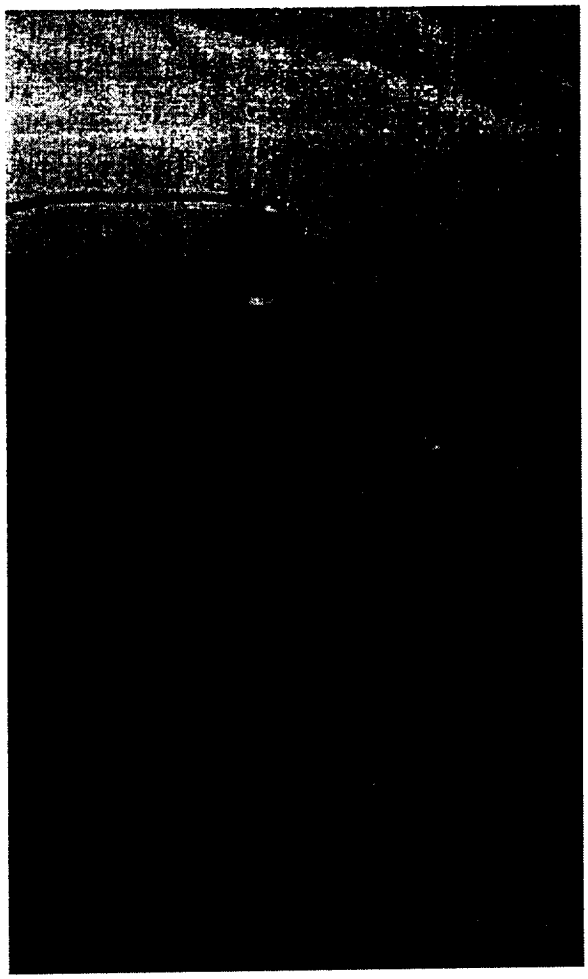


Figure 2-5: Tensiometer Hardware

E. Microcontrollers

Controller hardware consists of one master controller and three motor controllers. These components are pictured in Figure 2-6 and are discussed in the following sections.

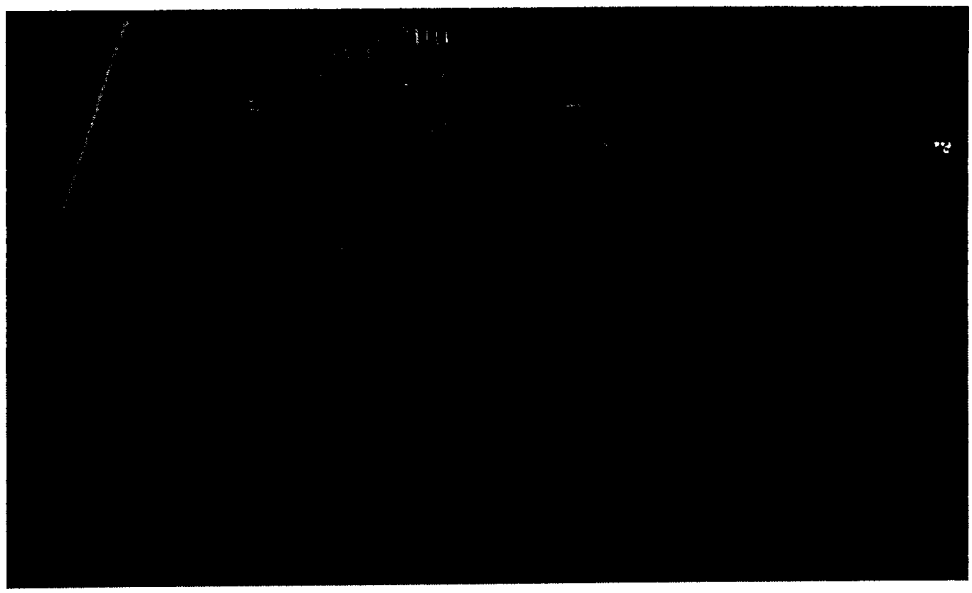


Figure 2-6: Controller Hardware

Motorola 68HC11E9 General Description

The Motorola 68HC11E9 is a one of a family of devices called microcontrollers or MCUs. An MCU combines discrete communication circuitry, a processor, a data bus, and memory into a small, low power, single chip computer. The 68HC11 MCU has 40 input/output pins that allow serial and parallel communication and analog-to-digital conversion. The 16 bit memory address of the 68HC11 allows the use of 64K bytes of memory. Internal memory in the 68HC11E9 consists of 512 bytes of random access memory (RAM), 512 bytes of electronically erasable programmable read only memory (EEPROM), 12K bytes of erasable programmable read only memory (EPROM), a 64 byte register block, and a 64 byte bootstrap interrupt vector block. The remainder of the 64K space can be accessed using an external memory chip. Detailed

hardware information can be found in M68HC11 Reference Manual[12], M68HC11 E Series Technical Data[13], and M68HC11 E Series Programming Reference Guide[14].

While hardware defines the limits of a microcontroller's capability, the software is the tool necessary to realize these limits. Program size, execution speed, mathematical capability, and design are all important considerations when measuring the effectiveness of any computer code.

Code for the 68HC11 can be written using a number of high level languages and assemblers. While languages such as Lisp and C provide data structures and mathematical functions that allow the intuitive coding of complex behavior, Motorola Assembly Language was used to generate all code used in this project. The primary reason for this selection was to simplify code troubleshooting. Debugging code in the PCbug11 or BUFFALO environment is straightforward using the disassembler. When using this feature, code is viewed as assembly code regardless of the original language. The usable result of debugged and assembled code is an ASCII file in S-record format. This machine language version of the original program can be read, edited, and transported to the MCU.

In order to program and debug the MCU, a Universal Evaluation Board (EVBU) is used . This board, with an MCU inserted, has many of the same functions as the final version of the motor controller. It provides the MCU with regulated power, a RS-232 serial interface, an oscillator, and access to all MCU

pin logic levels. Since the MCU is a self-contained computer, some software must be present and running before any meaningful communication with any other system can occur. Two software applications, provided by Motorola with the EVBU, are BUFFALO and PCbug11. While allowing the user to perform nearly the same tasks, these two applications work in very different ways.

BUFFALO is a complex piece of code that must be previously loaded into MCU memory. When the EVBU is reset, BUFFALO begins execution. This program allows the user to use a VT100 terminal emulation program and serial link to connect to the MCU. Once the connection is established, code can be loaded into RAM or EEPROM, executed, and debugged. The two most significant limitations of BUFFALO are that it must already be loaded into MCU memory and that it cannot modify EPROM. BUFFALO is well documented and discussed in the User's Manual provided with the EVBU[15].

PCbug11 is a DOS-based application capable of connecting to an unprogrammed MCU. During initialization, a small program, called a talker, is loaded in to MCU memory. This small but powerful piece of code, allows the user to read and program any available MCU memory location in RAM, EEPROM, or EPROM. Since only a small portion of memory is used temporarily for the talker, a much larger block of code can be transferred to MCU memory. The use of PCbug11 is documented in the PCbug11 User's Manual[16].

One important feature of the MCU is the receiver wake-up operation. When multiple controllers are used, every controller receives any message sent

by a controller on the network. A system must be devised to allow a receiving MCU to determine if it is being addressed. The address-mark wake-up feature, available on the 68HC11 MCU, solves this dilemma. Each MCU is placed in a dormant state by enabling the RWU bit in the SCCR2 register. In order to select any controller, a byte of information must be sent in which the most significant bit is set. The remaining seven bits are used as a coded address. This byte of information will wake up each controller. The software running on each MCU is responsible for determining if the encoded address matches its own. If no match exists, the software is responsible for placing the MCU back in the dormant mode.

Master Controller

The master controller includes a 5 V voltage regulator, a M68HC11 microcontroller, a crystal oscillator circuit, and a reset circuit. In addition, the master controller, detailed in Appendix A, also contains an external 32K RAM chip, a claw solenoid activation circuit, the conditioning circuit for the tensiometer switch, and serial communications hardware for five serial ports. The external RAM allows for increased program size and faster reprogramming time than using internal EPROM. Using this external memory requires that the MCU be used in expanded mode. As a result, ports B and C are no longer usable as external I/O pins. The claw activation circuit consists of a transistor driven relay switch. The conditioning circuit for the tensiometer is discussed in the tensiometer section. The communication hardware consists of three RS-232

serial port drivers and one asynchronous communications interface adapter (ACIA). Two of the RS-232 drivers each control two serial ports. These four ports, connected to the TxD and RxD MCU pins, are used to communicate with the motor controllers. The last driver is used, along with the ACIA, to allow serial communication with the supervisor computer via the Port C data bus.

Motor Controller

Like the master controller, the motor controller, detailed in Appendix B, contains a 5 V voltage regulator, a M68HC11 microcontroller, a crystal oscillator circuit, and a reset circuit. In addition, it contains optical encoder decoding circuit and one RS-232 serial driver. The decoding circuit is discussed in the section on optical encoders. The single serial driver is used to allow communication with the master controller via the RxD and Txd MCU pins.

Chapter Three: CONTROL SOFTWARE DESCRIPTION

Each of the four microcontrollers contains embedded control software. These programs were coded in assembly language for the Motorola M68HC11 series microcontroller. Each program is written as a text file, assembled, and downloaded to the microcontroller RAM, EPROM, or EEPROM using PCBUG11 or BUFFALO. A listing of the code used in the master controller can be found in Appendix C. A listing of the code used in a typical motor controller can be found in Appendix D. Appendix E contains Motorola application notes outlining the steps necessary to write, assemble, store, and run a piece of sample code.

The master controller contains the code necessary to service the supervisor computer, the claw actuator and sensor, and each of the three motor controllers. The motor controller code, identical in each case except for constants defining position limits, controller address, and motor speed, is responsible for driving and sensing motor position and communicating with the master controller.

Since the exploratory vehicle system is currently under construction, an IBM-compatible 486DX-33 personal computer running PROCOMM terminal emulation software is used in place of the supervisor computer. Any computer with a 9600 baud serial connection and software to access that port can be substituted for the supervisor computer.

Discussion of the two programs is divided into four sections. Each section focuses on the algorithm used to control the interaction between two hardware component systems.

A. Supervisor / Master Controller Loop

The link between the supervisor and the master controller has two functions. First, the supervisor must provide the master controller with an ASCII string containing platform orientation and target coordinates. Second, the master controller provides claw trajectory information that can be used for system monitoring. The supervisor must examine the sample object and resubmit the command if the correct object was not retrieved.

All data transfer is accomplished via a 9600 baud serial connection. Data strings are in ASCII format to allow easier debugging and system monitoring. These strings are converted into hexadecimal values upon receipt by the master.

Each time the master receives information from a motor controller or sensor, a character string is sent to the supervisor. This string contains position information for each motor as well as the current state of the claw actuator. Motor positions are written as absolute angles, in degrees, in hexadecimal form.

B. Master Controller / Motor Controller Loop

The master controller, upon receiving the approximate target location, calculates the desired motor positions. These positions are functions of the desired position $(\theta,r,z)_d$ and platform orientation (p,q). The master algorithm

approximates the motor positions by assuming that the platform is level during manipulator operation. This assumption decouples the effects of motor positions on claw position. Each motor is assumed to control one degree-of-freedom and have no effect on the other two degrees-of-freedom.

Since the claw is suspended on a cable, it will always move along a vertical line. When pitch and roll are both zero, this vertical line is parallel with the z axis. In this situation, the distance that the claw must be extended or retracted is a function only of the desired z coordinate. Similarly, the trolley motor only affects r and the boom motor only affects θ. When some platform pitch or roll exists, there will be an error in claw position whose magnitude varies with motor positions and platform orientation. Since the object of the maneuver is not to reach some given position, but rather to find some object within the search area, the claw position error only becomes a problem if it is sufficient to position the search area away from the target object. The amount of error that is acceptable in the system is a function of the search pattern area and grid resolution.

Given that the platform orientation is neglected, the system is completely decoupled. As a result, each motor controls a single degree-of-freedom. The desired motor positions are calculated based on measured values of trolley and claw travel in degrees per inch, measured motor angles when the claw is in the home position, and the desired cylindrical coordinates. Values for these constants can be found the software listings included in Section 0.

The master controller issues all commands to the motor controllers and claw in a serial fashion. After sending each motor command, the master waits for the motor controller to achieve and feedback its desired position. Two routines, *docmds* and *docmds2*, are used to issue a string of commands. The first, *docmds*, performs them in the order of boom motion, trolley motion, claw motion, and claw activation. The second, *docmds2*, commands the motors in the reverse order and but still activates the claw last. The first routine is used to approach an object. Since this object could be in a depression, the claw is kept at as high as possible until directly over the target site. The second routine, used after the object is captured, lifts the claw completely before moving the boom or trolley.

A portion of the main routine of the master controller software, called whenever a valid target position input string is received, is listed below.

```
jsr     findit
jsr     putaway
jsr     gohome
```

The first routine, *findit*, is responsible for searching for and grasping the object. The search algorithm is addressed in Chapter 4. The second routine, *putaway*, directs the claw to move to a receptacle and release the object. The last routine, *gohome*, sends the claw back to its home position.

C. Motor Controller / Motor Loop

The motor controller receives a hexadecimal number, in ASCII characters, representing the desired absolute angle of the motor. The actual motor position

is read from the decoder chip and converted into degrees. These two values are compared and a desired rotation, in degrees, is calculated. This number is converted into motor steps. The motor controller then drives the direction pin on the motor driver high for forward motion or low for reverse motion. A square wave is then applied to the driver input pin. The number of pulses in this wave corresponds to the number of desired steps. The frequency of the wave determines motor speed. The motor controller again reads the decoder chip and the process is repeated as necessary. An error of 1° is allowed between desired and final actual motor position. This allowance is necessary due to the encoder resolution and integer division necessary to convert the encoder value to degrees. When the final position is attained, the motor controller echoes its current absolute position to the master controller.

D. Master Controller / Claw Loop

The master controller, in addition to performing high level motor position control and communication with the supervisor, is responsible for claw activation and obstacle contact detection. The claw is commanded via a relay on the master controller circuit board. Contact between the claw and some obstacle is monitored via a boom mounted switch and conditioning circuitry mounted on the master controller circuit board.

The master controller enables or disables the claw by varying the state of one of the microcontroller output pins. The pin indirectly drives the claw solenoid using a transistor and a relay.

A sudden and sustained loss of tension occurs as a result of contact between the claw and some surface. The loss of cable tension causes the activation of an interrupt service routine on the master controller to command the motor controller to stop and slightly raise the claw. This motion allows the claw to better grip the target object.

This feature is vital to the success of the searching algorithm. The claw is closed to minimize the projected area on the work surface and to limit contact to one point instead of three points. The claw is then lowered at predetermined points until contact is made. The absolute heights of these points are stored until all points are searched. After all nine heights are measured, they are converted into quarters of an inch above the lowest of the nine points. The predetermined object height is 1.25 in. Software selects the first of the nine points that happens to be higher than 1 in. If no object tall enough is found then the search pattern may be repeated at another location. In general, a complete search includes the mapping of 47.5 in^2. The search pattern and mapping technique are discussed in more detail in Chapter 4.

Chapter Four: APPLICATION OF CONTROL SOFTWARE TO CRANE SYSTEM

A. Test Scenarios

A series of three test cases was used to determine the ability of this manipulator device to retrieve an object. In case one, the claw moves to a specified location, grips the object if one exists, moves to the drop zone, releases the object, and returns home. In case two, the boom and trolley are moved to the positions specified. At this point, the claw is lowered until it makes contact with the ground or an object. The claw then lifts slightly, grips the object, and completes the maneuver. In case three, a relief map of the area surrounding the point of initial contact is created. Based on information contained in this map and prior knowledge of the target object, either the object is located or a new area is searched. Ultimately, either the object is found and the maneuver completed or the search is abandoned.

In case one, the claw moves toward the specified location $(\theta,r,z)_d$ by first swinging the boom into position (θ_d) and then moving the trolley (r_d). Once the claw is suspended above the desired point, it is lowered to the appropriate height (z_d). In the trajectory used in this case, only one motor is in motion at any given time. Controller hardware does not limit the system to this serial motion. This method is used to avoid undesired contact between the claw and the environment. Due to the crane structure, any motion of boom or trolley requires

that no obstacles be present in the space through which the suspended claw moves. Keeping the claw retracted until all other motion is complete decreases the chances of an unwanted collision with obstacles in the workspace. Upon reaching the desired location, the claw is activated. No method of target object confirmation is currently in place as part of this system. The vehicle supervisor computer is responsible for confirming that the correct object was retrieved using some part of its sensor array. The object, having been retrieved, is moved to a previously defined point and dropped into a receptacle. The claw is then returned to home position.

The key difference in case two is that the final height of the claw (z) is not necessarily the height specified (z_d). When the exact height is unknown by the supervisor, a value at the limit or beyond the reachable workspace is used. Whenever the claw makes contact with some object before the specified height is reached, the claw descent is halted by the master controller. Next, the claw suspension cable is retracted a distance of between one-half inch and one inch. This claw height above the ground was determined by trial-and-error to be ideal for gripping the target object. The claw is then closed and the remainder of the maneuver is identical to case one.

In case three, the specified boom and trolley positions $(\theta,r)_d$ define the central vertical axis of a search space instead of the vertical line along which the object lies. As in case two, the claw is lowered until it makes contact with the workspace. In case three, however, this motion occurs at least at the nine points

illustrated in grid 1 of Figure 4-1. The height coordinate (z) at each point is measured and is normalized by subtracting the lowest height found. The heights are then converted to a number of quarter inches. A typical resulting relief map is illustrated in Figure 4-2. Since the object dimensions are assumed known, the maximum measured height can be compared with a minimum anticipated object height. If the object is determined to exist at one of the search points, then the maneuver is completed as in the first two cases. If the object is not found at any of the search points, then a new section of the workspace is searched. This mapping process is repeated in the pattern shown in Figure 4-1 until either all the specified areas are searched or the object is found.

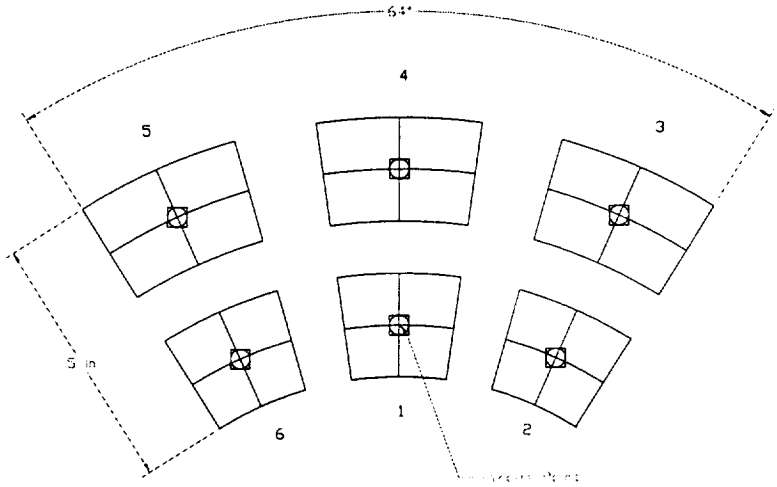


Figure 4-1: Search Pattern

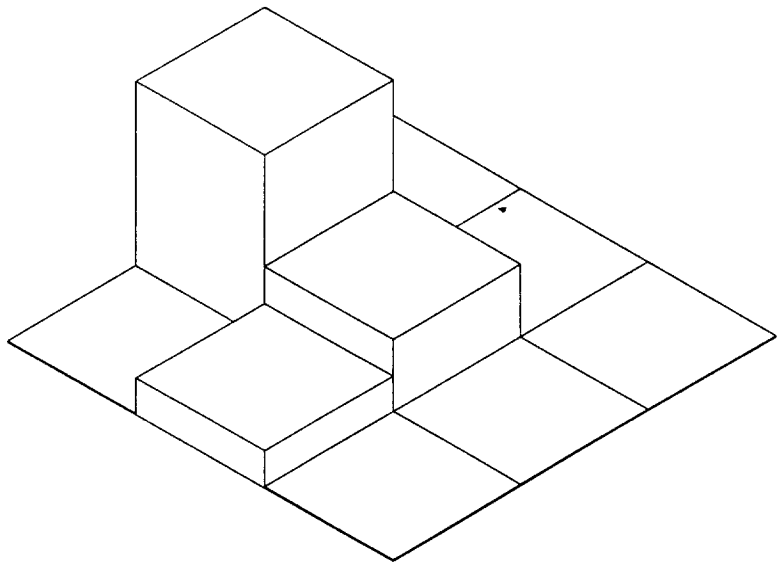


Figure 4-2: Typical Relief Map of Minor Grid

B. Results and Evaluation

In case one, the object was retrieved if the boom position was within approximately 3°, the trolley position was within 0.5 in, and the claw height was within 0.5 in of the object center. The primary limitation of this scheme is that the specified position provided by the vehicle supervisor must be fairly accurate. Precise object location may prove difficult for the vehicle sensors to measure due to the small size of the object, the natural terrain background, and the difficulty in sensing within 8 in of the vehicle. Actuator output is plotted in Figure 4-3. The target position used in this case is +00,+00,-180,+07.0,-10.0.

In case two, the burden of sensing object height is shouldered by the manipulator device. The vehicle sensors must still synthesize a fairly accurate two dimensional image of the object as viewed from above. While an

improvement over case one, the vehicle sensors still limit mission success. The target object will typically be sighted before it is within reach of the manipulator. The vehicle must then move into position and stop before retrieval can occur. As in case one, this close proximity surface may be difficult to map. Actuator output is plotted in Figure 4-4. The target position used in this case is +00,+00,-180,+07.0,-10.0. Note that the only difference in case one and case two is that the claw height did not reach -10 in. Instead, a collision with an obstacle forced the claw to stop and grip at approximately -2 in.

Case three actuator outputs are illustrated in Figure 4-5. The boom angle, trolley radius, and claw height all demonstrate the multipoint sounding technique used to map the area. As in the previous two cases, the input string was +00,+00,-180,+07.0,-10.0. In this case, however, the object was actually located at +00,+00,-150,+07.0,-02.0. The resulting relief map is illustrated in Figure 4-6. In this case, the specified coordinates must only be accurate enough to ensure that the object is within the search area. As long as the object is within two horizontal inches in the radial direction and four horizontal inches in the tangential direction of the specified position, the object can be found and retrieved. The ability of the manipulator to map the area and find the object is a big advantage. Mission success in retrieving the object is much more likely with this scenario. The major limitation of this method is that the maximum resolution of the search pattern is limited by the physical dimensions of the claw. Due to the large diameter of the closed claw, undesired contact between claw and

object sometimes occurs. This unwanted contact can cause the object to move to a previously mapped location and be missed in the search or the claw height to be inaccurately measured. A future modification of the system would be to modify the claw so that contact between the claw and objects not directly beneath it would be reduced.

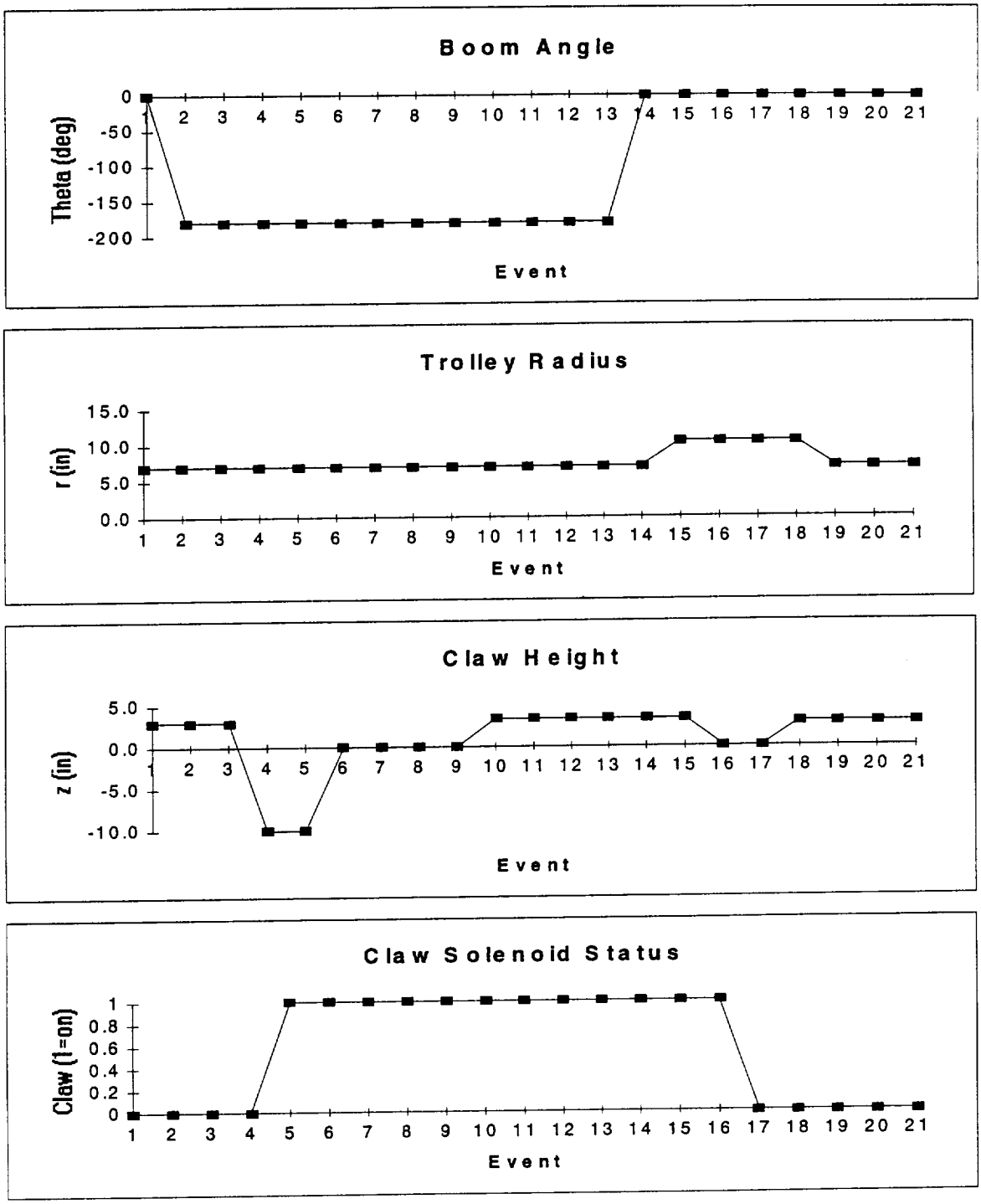


Figure 4-3: Case 1 Actuator Output

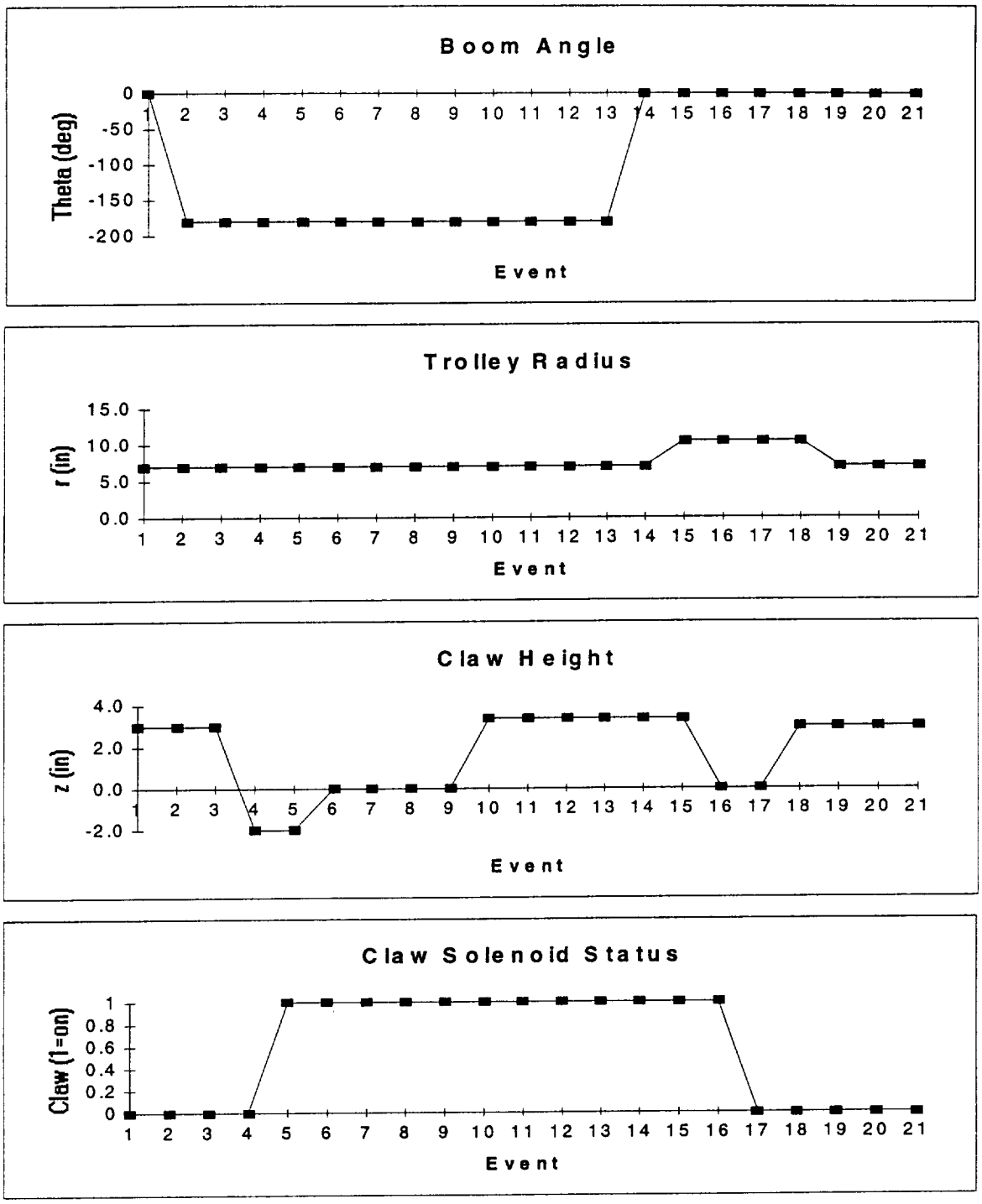


Figure 4-4: Case 2 Actuator Output

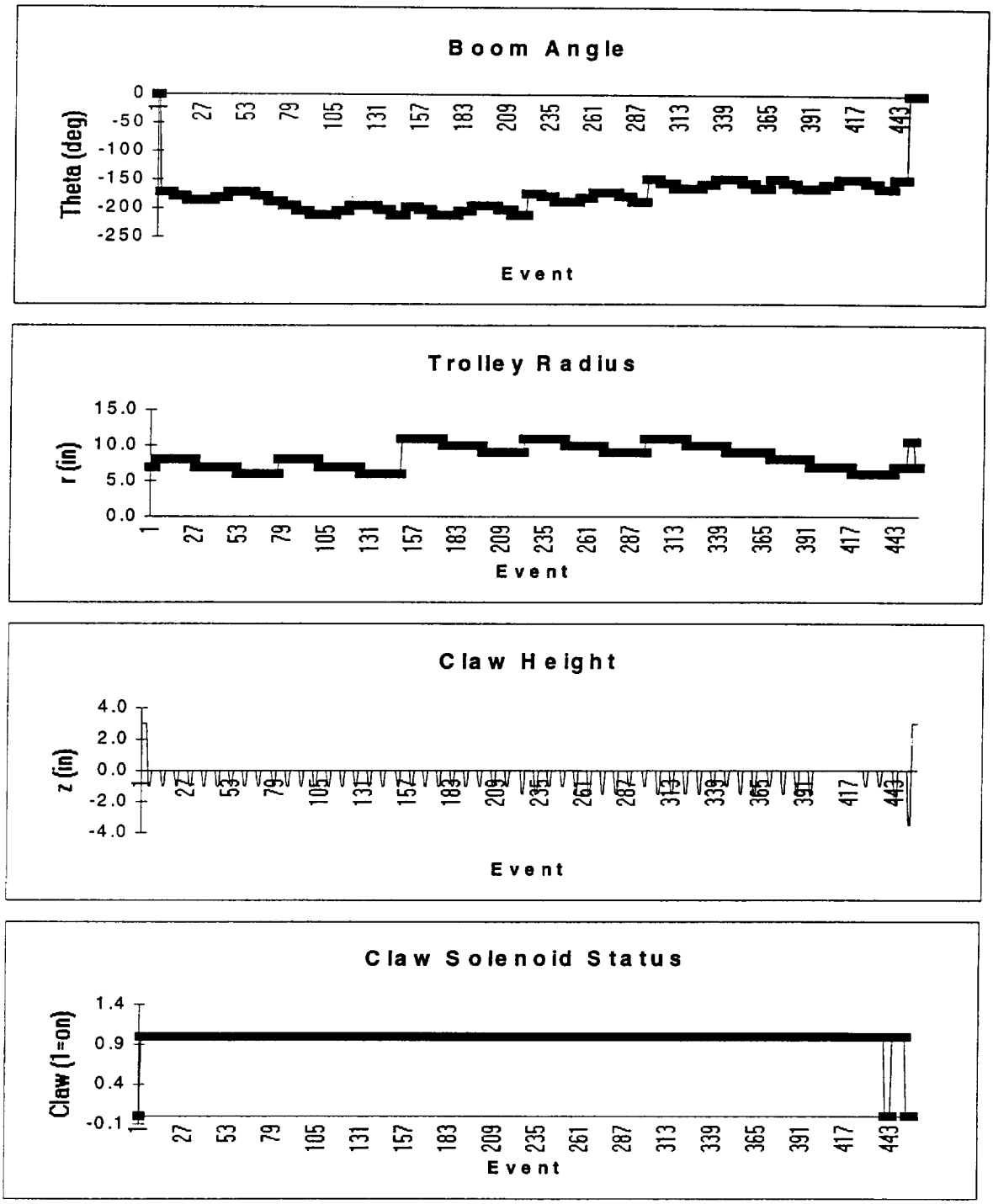


Figure 4-5: Case 3 Actuator Output

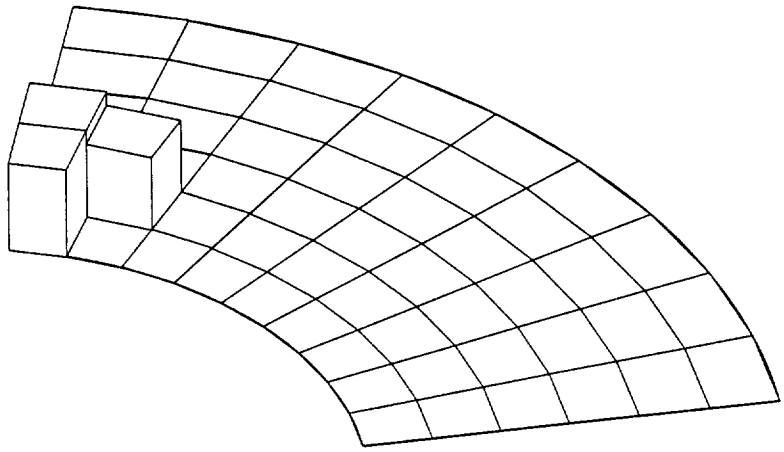


Figure 4-6: Case 3 Relief Map

In all three cases, an intermittent hardware or software error occasionally gives grossly inaccurate count values in motor position. This problem has been resolved in software by comparing the actual counter value and an estimated value. When gross differences occur, the estimate is used by the motor controller and the master controller is notified.

Chapter Five: CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

The manipulator system successfully meets the stated requirements of size, weight, and functionality. A clay cylinder measuring 1.25 in by 1.25 in was used as the target object during system testing. Given an initial boom angle within 24° and an initial trolley position within 4 in of the actual values, the system will locate and retrieve the target object approximately 80% of the time.

The search and recovery algorithm eliminates the need for precisely measured target coordinates. This feature allows the rover to dedicate its sensors to more important data gathering. It also eliminates the need for sensors dedicated to surveying the area within 8 in of the vehicle base. The vehicle can locate an object at a distance limited only by its sensor arrays, move into position near the object, and provide the manipulator with estimated coordinates relative to the vehicle.

The manipulator does occasionally miss the target in its search and recovery effort. In these cases, the claw typically causes the object to move into an already measured point when sounding the area immediately surrounding the object. As a result, the object does not show up in the search map. In these situations, the vehicle sensor system will sense the absence of the target object and will resubmit the command. This tactic of multiple attempts is usually successful.

The system is constructed in a modular form that will allow future users to optimize the interaction between the vehicle system and the manipulator subsystem. Sensors and actuators can be modified to provide new performance characteristics. Control software can be altered for testing of new control and search algorithms.

Modification of the end effector could improve system effectiveness in three ways. A smaller claw would both allow a higher resolution mapping algorithm and reduce unwanted claw/object interaction. This change will allow a finer resolution and smaller object search capabilities. Redesigning the fingers of the claw could increase the acceptable variation of object size, shape, and consistency.

The master controller software listed in the Appendices neglects the effects of platform orientation on actual position. Since the claw is suspended by a cable, a change in position of the motor controlling claw height results in claw motion along the z axis of the world coordinate system. Any pitch or roll of the vehicle will cause the z axis of the manipulator coordinate system to no longer be parallel with the z axis of the world coordinate system. Inclusion of these effects when calculating desired motor positions based on a specified target coordinates would result in increased accuracy in claw placement.

Chapter Six: REFERENCES

[1] R. M. Brown, Jr., G. K. F. Lee, and A. W. Hulbert. Environmental Sampling Tools Designed For Use On A Low Cost Remotely Operated Vehicle (LCROV). Proceedings of Diving For Science 1992, 23-30, 1992.

[2] J. Albus, R. Bostelman, and N. Dagalakis. The NIST ROBOCRANE. Journal of Robotic Systems, 10(5):709-724, 1993.

[3] R. M. DeSantis and S. Krau. Bang Bang Control of An Overhead Cartesian Crane. Proceedings of the 1993 American Control Conference, 1:971-975, 1993.

[4] H. Ihara. Fuzzy Logic For Control Systems. Automatic Control in Aerospace 1992, 251-255, 1992

[5] J. L. Jones and Anita M. Flynn. Mobile Robots: Inspiration to Implementation, A K Peters, Ltd., Wellesly, MA, 210-224, 1993.

[6] C. I. Ume, J. Ward, and J. Amos. Application of MC68HC11 Microcontroller For Speed Control Of DC Motor. Journal of Microcomputer Applications, 15(4):373-386, 1992.

[7] Motorola Background Information. Fuzzy Logic And Embedded Control. Third Workshop on Neural Networks: Academic/Industrial/NASA/Defense, WNN92:611-621, 1993.

[8] S. Lee et al. A Mars Surface Exploration Vehicle Testbed For Control Algorithm Verification. Proceedings of ISRAM Conference, Maui, HI, 1994.

[9] S. Lee et al. The Mars Mission Research Center Exploration Vehicle Testbed: A Platform For System Integration Studies. Proceedings of the AIAA Space Programs & Technologies Conference, Huntsville, AL, 1994.

[10] S. Lee et al. A Distributed Architecture For A Mars Surface Exploration Vehicle Testbed. Proceedings of the ISCA Conference on Computers and Their Applications in Industry and Engineering, San Diego, CA, 1994.

[11] D. Lancaster and H. M. Berlin. CMOS Cookbook: Second Edition, SAMS, Carmel, Indiana, 256-259, 1993.

[12] Motorola. M68HC11 Reference Manual, Motorola Literature Distribution Center, Phoenix, Arizona, 1991.

[13] Motorola. M68HC11 E Series Technical Data, Motorola Literature Distribution Center, Phoenix, Arizona, 1993.

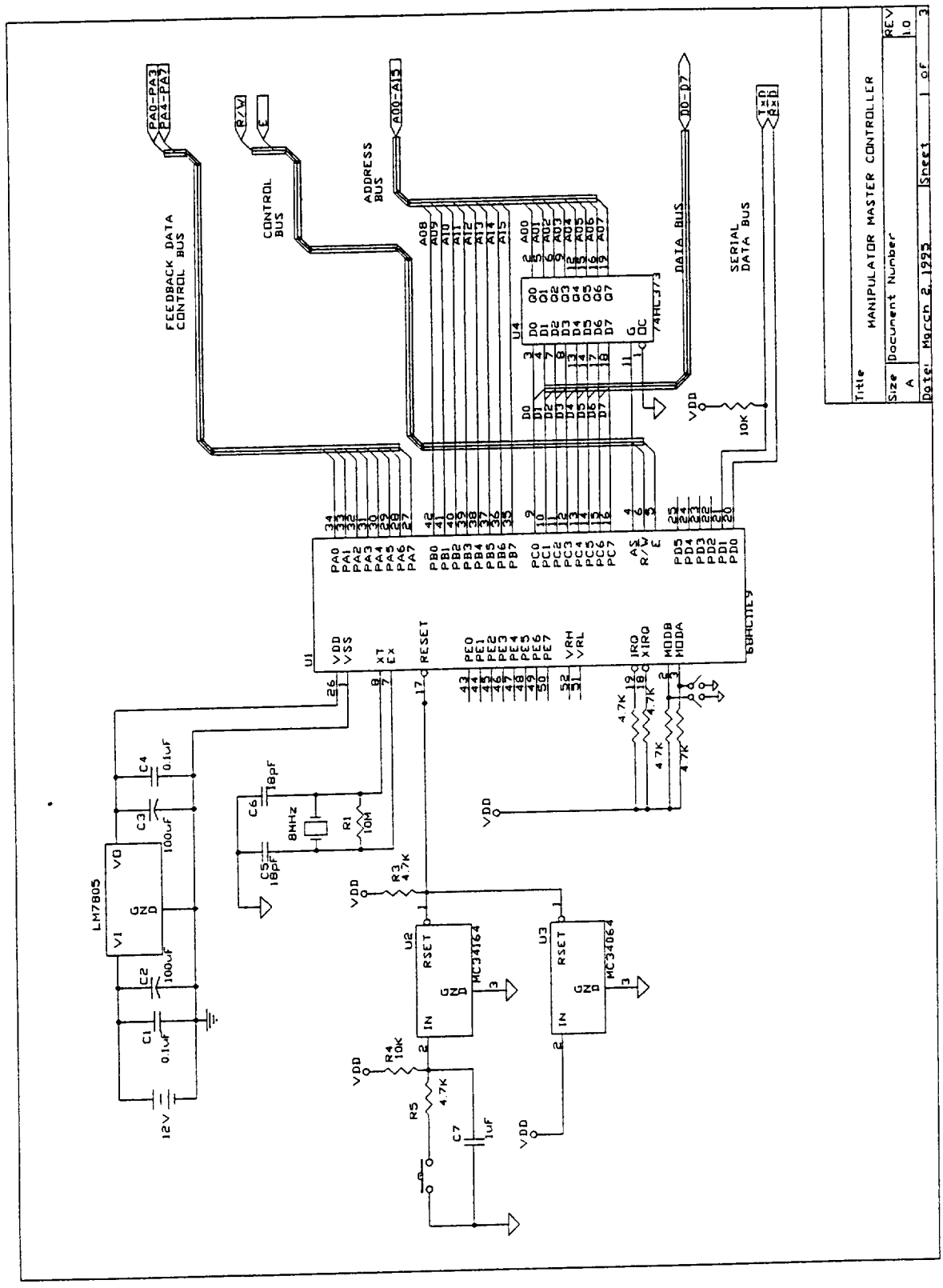
[14] Motorola. M68HC11 E Series Programming Reference Guide, Motorola Literature Distribution Center, Phoenix, Arizona, 1993.

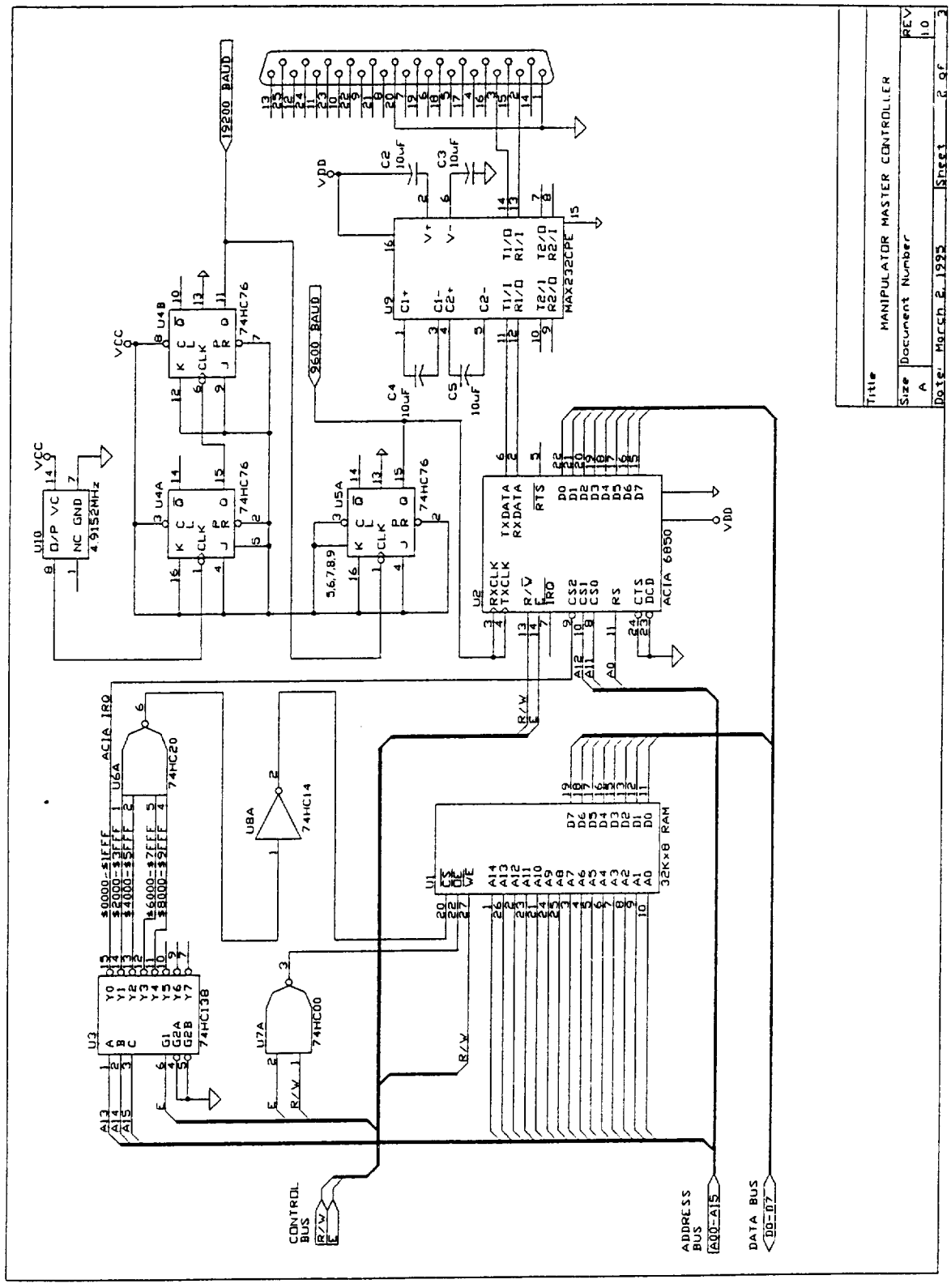
[15] Motorola, Inc. M68HC11EVBU Universal Evaluation Board User's Manual. Motorola Literature Distribution Center, Phoenix, Arizona, 1992.

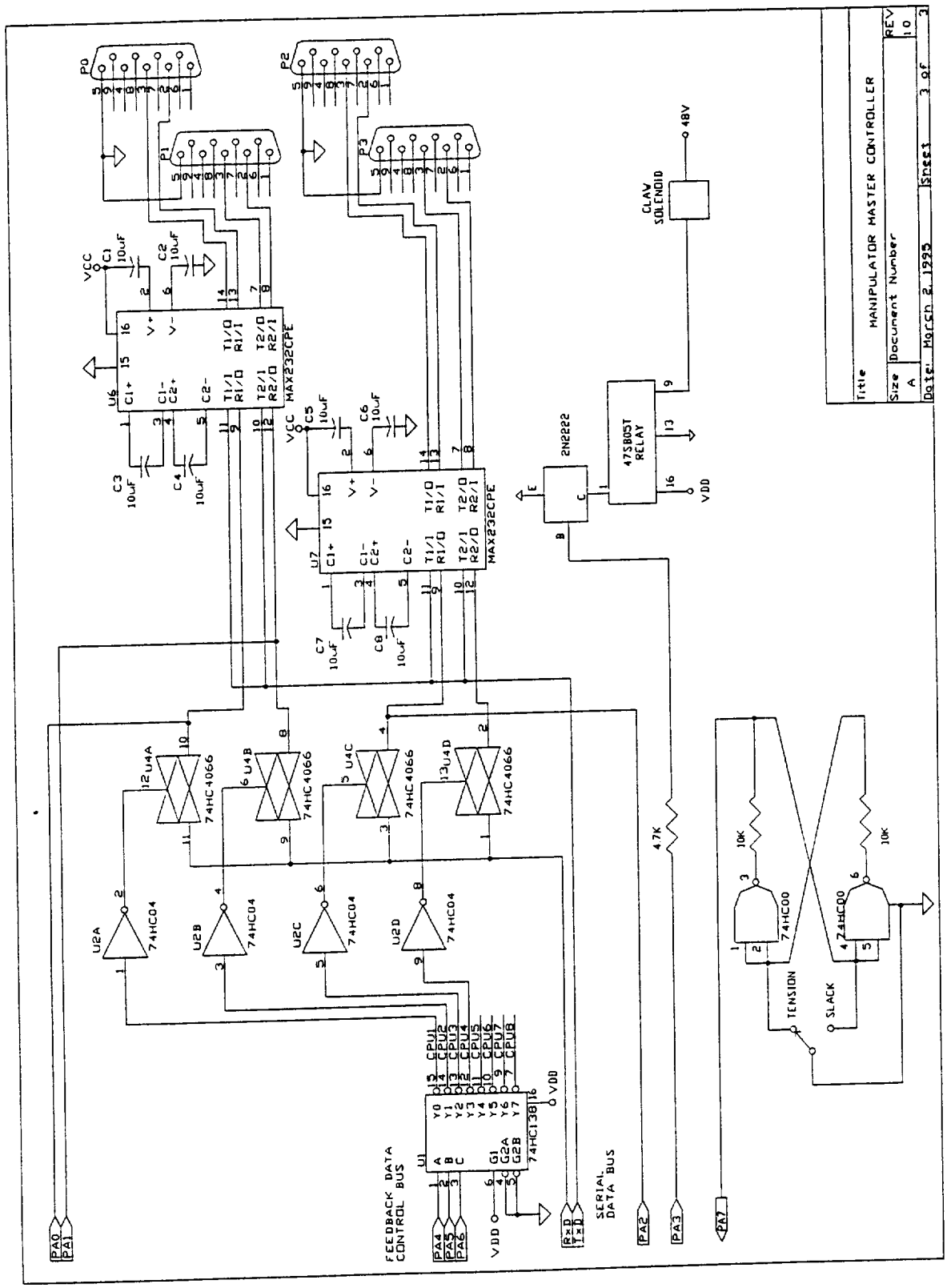
[16] Motorola, Inc. MC68HC11 Pcbug11 User's Manual. Motorola Literature Distribution Center, Phoenix, Arizona, 1992.

APPENDICES

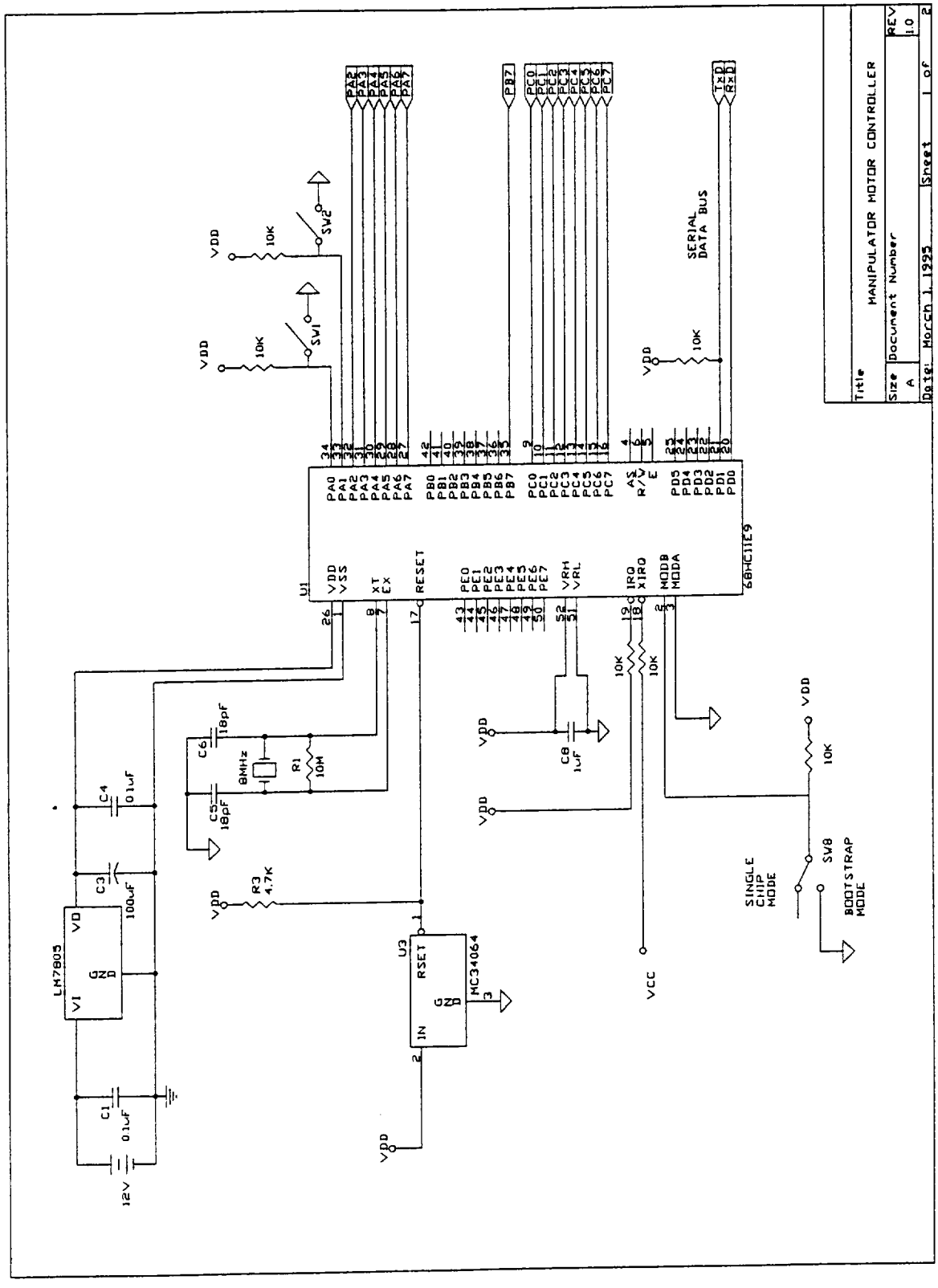
A. Master Controller Circuit Diagram

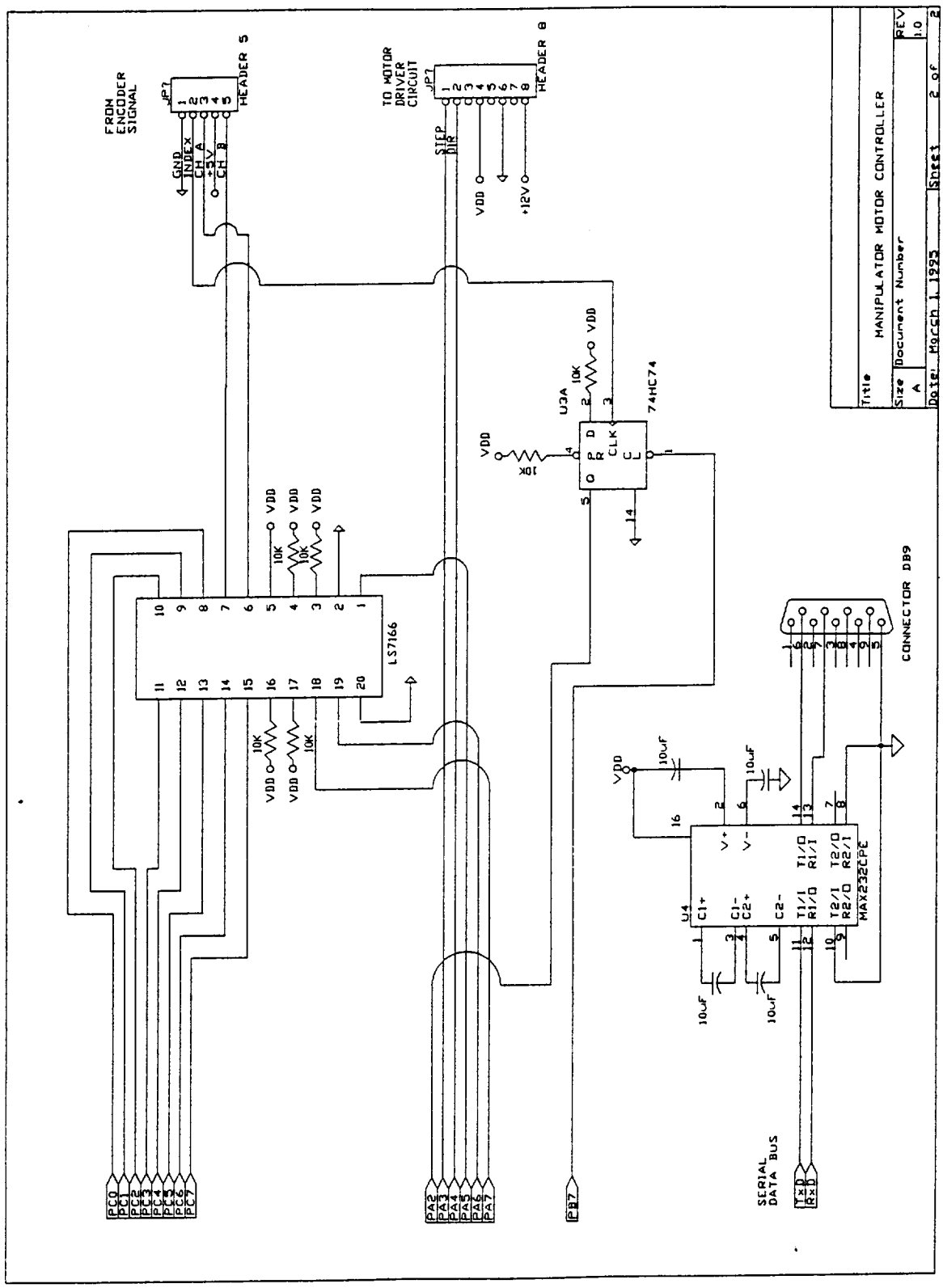

MANIPULATOR MASTER CONTROLLER
Document Number
REV 1.0
Sheet 1 of 3
Date: March 2, 1995
FEEDBACK DATA CONTROL BUS
CONTROL BUS
ADDRESS BUS
DATA BUS
SERIAL DATA BUS
74HC373
6811CTE9
MC34164
LM7805
8MHz
12V
VDD
10K
4.7K
10M
18pF
0.1uF
100uF
1uF


MANIPULATOR MASTER CONTROLLER
Document Number
Sheet 2 of 3
Date: March 2, 1995
19200 BAUD
9600 BAUD
MAX232CPE
ACIA 6850
32K x 8 RAM
74HC138
74HC00
74HC14
74HC20
74HC76
4.9152MHz
ACIA IRQ
CONTROL BUS
ADDRESS BUS
A00-A15
DATA BUS
D0-D7


MANIPULATOR MASTER CONTROLLER
Title
Size
A
Document Number
REV
1.0
Date: March 2, 1995
Sheet
3 of 3
P0
P1
P2
P3
MAX232CPE
74HC4066
74HC04
74HC00
74HC138
2N2222
47SBOST RELAY
CLAW SOLENOID
48V
VDD
VCC
10uF
4.7K
10K
TENSION
SLACK
FEEDBACK DATA CONTROL BUS
SERIAL DATA BUS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7

B. Motor Controller Circuit Diagram


MANIPULATOR MOTOR CONTROLLER
Title
Size
Document Number
REV
A
1.0
Date: March 1, 1995
Sheet 1 of 2
SERIAL DATA BUS
SINGLE CHIP MODE
BOOTSTRAP MODE
LM7805
MC34064
68HC11E9
8MHz
12V
VDD
VCC
SW1
SW2
SW8


FROM ENCODER SIGNAL
JP7
HEADER 5
GND
INDEX
CH A
+5V
CH B
TO MOTOR DRIVER CIRCUIT
JP7
HEADER 8
STEP
DIR
VDD
+12V
VDD
10K
U3A
74HC74
LS7166
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PA2
PA3
PA4
PA5
PA6
PA7
PB7
U4
MAX232CPE
C1+
C1-
C2+
C2-
V+
V-
T1/I
R1/O
T2/I
R2/O
T1/O
R1/I
T2/O
R2/I
10uF
SERIAL DATA BUS
TxD
RxD
CONNECTOR DB9
MANIPULATOR MOTOR CONTROLLER
Title
Size
A
Document Number
REV
1.0
Date: March 1, 1995
Sheet 2 of 2

C. Master Controller Program Listing

```
*****************************************************************
*
* MASTER SOFTWARE FOR MARS ROVER MANIPULATOR
*
*****************************************************************
*
*
* Based on:
* 08/19/94  ct0.asm  acia in / sci out ;with sequence input ok
*    ------------ CK.Chao
* Modifications:
* 11/01/94    mast1.asm  Read MAXIN bytes of data & echo to motor CPU.
* 11/07/94    mast2.asm  Run main loop until command received from
*             supervisor.  Echo command once then return to main loop
* 11/08/94    mast3.asm  PA7 commands claw upon receipt of C1 command.
*             C(anything besides 1) results in claw off command.  Any other
*             string is sent out SCI
* 11/10/94              added lines in INITPA to disable IRQ on pins
*             PA0, PA1, PA2, PA3.
* 12/01/94    mast4.asm -- Modified input command structure to
*                address(one byte hex), theta(four ASCII characters
*                representing a two byte hex).
* 12/03/94    mast5.asm -- Receive and echo output from motor controllers.
*                Each motor controller sends the master address (δ), four
*                ASCII characters representing a two byte hex number (the
*                motor position in degrees), two ASCII characters
*                representing the controller mode (closed loop (C0) or open
*                loop (O1, O2, or O3)), and four ASCII characters representing
*                the command received by the motor (desired motor positon
*                in degrees.
* 12/12/94  mast6.asm -- Rearrange functions to allow for easier
*                inclusion of conversion routine.  Output printed whenever
*                flag (ffbk1,ffbk2,ffbk3,ffbkc) set.  Input command syntax
*                modified to "±pp,±qq, ±ttt, ±rr.r, ±zz.z".  Command syntax is
*                verified after receipt.  An error message is written to
*                ACIA if an invalid command is received.
* 01/03/95  mast7.asm -- Configure PA3 (IC4) as the interrupt pin (on
*                falling edge) indicating loss of tension in crane cable.
*                Interrupt Service Routine (SLAK_ISR) will be called.
*                Modified putaway routine to take claw all the way to the
*                top before moving to drop zone.
* 01/09/95  mast8.asm -- Add routines to search around nominal target for
*                high spot.  After identification, lift at high spot.
*                Modify output format to eliminate mode information and
*                add 0 as prefix and h as suffix to angle information.  This
*                will allow Mathcad plot.  Add PRNTDAT call in ACIA_ISR to
*                print home position after valid input string to add
*                home position to plot.  Added routine findit to search
*                for object around input (theta,r).  Moved "docmds" from main
*                to new routine.
*

B96      equ      %00110000
R_DATA   equ      $4000
CR       equ      $0D
SPACE    equ      $20
LF       equ      $0A
ACIA     equ      $1800
PCHAR    equ      $03
QCHAR    equ      $03
TCHAR    equ      $04
RCHAR    equ      $05
ZCHAR    equ      $05
MAXIN    equ      PCHAR+QCHAR+TCHAR+RCHAR+ZCHAR+$04
MAXSCI   equ      $06
RDPI     equ      $51
RZERO    equ      $38B
ZDPI     equ      $74
ZZERO    equ      $FE78
```

```
***********
* jump table
***********
        org     #$00ee
        jmp     ACIA_ISR                *IRQ
        org     #$00e2
        jmp     SCI0_ISR                *IC3
        org     #$00e5
        jmp     SCI1_ISR                *IC2
        org     #$00e8
        jmp     SCI2_ISR                *IC1
        org     #$00d3
        jmp     SLAK_ISR                *IC4/OC5

**********
* EEPROM contants
**********
        ORG del_r               * Search pattern constants
        FDB 2                   * del_r (half inches)
        ORG del_theta
        FDB 8                   * del_theta (degrees)

***********
* the variables
***********
        org     #R_DATA
rdata   rmb     MAXIN
box1    rmb     MAXIN
box2    rmb     MAXIN
box3    rmb     MAXIN
box4    rmb     MAXIN
box5    rmb     MAXIN
box6    rmb     MAXIN
box7    rmb     MAXIN
box8    rmb     MAXIN
box9    rmb     MAXIN
cntr1   rmb     MAXIN
cntr2   rmb     MAXIN
cntr3   rmb     MAXIN
cntr4   rmb     MAXIN
cntr5   rmb     MAXIN
cntr6   rmb     MAXIN
cntr7   rmb     MAXIN
cntr8   rmb     MAXIN
cntr9   rmb     MAXIN
hex2    rmb     1
sci0_in rmb     MAXSCI
sci1_in rmb     MAXSCI
sci2_in rmb     MAXSCI
datain  rmb     1
ffbkc   rmb     1
ffbk1   rmb     1
ffbk2   rmb     1
ffbk3   rmb     1
fcmdc   rmb     1
fcmd1   rmb     1
fcmd2   rmb     1
fcmd3   rmb     1
fstop   rmb     1
pitch   rmb     PCHAR
roll    rmb     QCHAR
theta   rmb     TCHAR
radius  rmb     RCHAR
height  rmb     ZCHAR
th1     rmb     2
th2     rmb     2
th3     rmb     2
th3a    rmb     4
```

```
th3s    rmb     2
temp    rmb     2
tmp16   rmb     2
clawcmd rmb     1
stphgt  rmb     2
hgt1    rmb     2
hgt2    rmb     2
hgt3    rmb     2
hgt4    rmb     2
hgt5    rmb     2
hgt6    rmb     2
hgt7    rmb     2
hgt8    rmb     2
hgt9    rmb     2
target  rmb     1
try     rmb     1
whole_in        rmb     2
half_in         rmb     2
        rmb     40
stack   rmb     1
SHFTREG RMB  2      input shift register
TMP1    RMB  1

        org     $4500           * eventually B700
del_r           rmb     2
del_theta       rmb     2

**********
* the code starts here.
**********
        org     $2000
        jmp     START

*****************************************************************************
*
*  THE FUNCTION LIBRARY
*
*****************************************************************************
**********
* bin2hex -- Separates each character of a hex number and calls outhex.
**********
bin2hex pshb
        psha
        lsra
        lsra
        lsra
        lsra
        anda    #%00001111
        jsr     outhex
        pula
        anda    #%00001111
        jsr     outhex
zb2hex  pulb
        rts

**********
* getsci -- if a character has been received on sci port, this character
*       retrieves and places in accumulator A.
**********
getsci  ldx     #REGBAS
        ldab    SCSR,X                  * if RDRF is 0 then wait
        bitb    #$20
        beq     getsci
        ldaa    SCDR,X
zg_sci  rts

*****************
*   HEXBIN(a) - Convert the ASCII character in A
* to binary and shift into shftreg.
```

```
*****************
HEXBIN  PSHA
        PSHB
        PSHX
*        JSR  UPCASE      convert to upper case
        CMPA #'0'
        BLT  HEXNOT      jump if a < $30
        CMPA #'9'
        BLE  HEXNMB      jump if 0-9
        CMPA #'A'
        BLT  HEXNOT      jump if $39> a <$41
        CMPA #'F'
        BGT  HEXNOT      jump if a > $46
        ADDA #$9         convert $A-$F
HEXNMB  ANDA #$0F        convert to binary
        LDX  #SHFTREG
        LDAB #4
HEXSHFT ASL  1,X         2 byte shift through
        ROL  0,X                carry bit
        DECB
        BGT  HEXSHFT     shift 4 times
        ORAA 1,X
        STAA 1,X
        BRA  HEXRTS
HEXNOT  nop
*       INC  TMP1        indicate not hex
HEXRTS  PULX
        PULB
        PULA
        RTS

**********
* onacia -- initializes acia port
**********
onacia: ldx     #REGBAS
        cli                             * enable IRQ (clear I in CCR)
        ldaa    OPTION,X
        ora     #%00100000              * set IRQ to recognize falling edge
        staa    OPTION,X
        ldaa    #$03                    * master reset of ACIA
        staa    ACIA                    * ACIA is at $1800
        ldaa    #%00010110              * cr4,cr3,cr2 = 101 : 8bit 1stop bit
*                                       * cr1,cr0 = 10 : ö 64 (IRQ enabled)
        ora     #%10000000              * enable IRQ
        staa    ACIA
zonacia rts

**********
* onsci -- initializes sci port
**********
onsci:  ldx     #REGBAS
        ldaa    #B96
        staa    BAUD,X                  * set baud rate
        ldaa    #%00001100              *m WAS 00001100 made 00101110
        staa    SCCR2,X                 * enable transmit & receive
        ldaa    #%00000000              *m was 00000000 made 00001000
        staa    SCCR1,X                 * set data is 8-bit mode
zonsci  rts

**********
* outhex -- converts hex to ASCII and transmits out ACIA port
**********
outhex  cmpa    #10
        bge     ge2A
        adda    #$30
        jsr     putacia
        bra     zo_hex
ge2A:   adda    #$37
        jsr     putacia
```

```
zo_hex  rts

**********
* outhexs -- converts hex to ASCII and transmits out SCI port
**********
outhexs cmpa    #10
        bge     ge2As
        adda    #$30
        jsr     putsci
        bra     zo_hexs
ge2As:  adda    #$37
        jsr     putsci
zo_hexs rts

**********
* putacia -- puts byte in accumulator A out acia port.
**********
putacia ldab    ACIA
        bitb    #$02
        beq     putacia
        anda    #$ff
        staa    ACIA+1
zp_acia rts

**********
* putsci -- puts byte in accumulator A out sci port.
**********
putsci  ldx     #REGBAS
        ldab    SCSR,X
        bitb    #$80
        beq     putsci                  * if TDRE is 0 loop back to putsci
        anda    #$ff                    * (not ready to be sent)
        staa    SCDR,X
zp_sci  rts

**********
* scib2h -- Separates each character of a hex number and calls outhexs.
**********
scib2h  psha
        lsra
        lsra
        lsra
        lsra
        anda    #%00001111
        jsr     outhexs
        pula
        anda    #%00001111
        jsr     outhexs
zscib2h rts

**********
* SLODOWN -- kills time when necessary
**********
SLODOWN psha
        pshb
        ldad    #$FFFF
SLO1    subd    #$0001
        bne     SLO1
        pulb
        pula
        rts

*************************************************************************
*
*  THE FUNCTIONS SPECIFIC TO MASTER OPERATION
*
*************************************************************************
**********
* ASC2HEX -- Converts an ASCII character (0-9) in accumulator B to a
```

```
*               hex number.  This number is returned in accumulator B.
**********
ASC2HEX cmpb    #'0'
        blt     za2h
        cmpb    #'9'
        bgt     za2h
        subb    #$30
za2h    rts

**********
* checkin -- Read input string and verify syntax
**********
checkin ldy     #R_DATA
        ldaa    0,Y
        jsr     issign
        cmpb    #'1'
        bne     synerr1
        ldaa    1,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr1
        ldaa    2,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr1
        ldaa    3,Y
        jsr     iscomsp
        cmpb    #'1'
        bne     synerr1
        ldaa    4,Y
        jsr     issign
        cmpb    #'1'
        bne     synerr1
        ldaa    5,Y
        jsr     isint
        cmpb    #'1'
        beq     set2

synerr1 jmp     synerr

set2    ldaa    6,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr2
        ldaa    7,Y
        jsr     iscomsp
        cmpb    #'1'
        bne     synerr2
        ldaa    8,Y
        jsr     issign
        cmpb    #'1'
        bne     synerr2
        ldaa    9,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr2
        ldaa    10,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr2
        ldaa    11,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr2
        ldaa    12,Y
        jsr     iscomsp
        cmpb    #'1'
        beq     set3
```

```
synerr2 jmp     synerr

set3    ldaa    13,Y
        jsr     issign
        cmpb    #'1'
        bne     synerr3
        ldaa    14,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr3
        ldaa    15,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr3
        ldaa    16,Y
        jsr     isdec
        cmpb    #'1'
        bne     synerr3
        ldaa    17,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr3
        bra     set4

synerr3 jmp     synerr

set4    ldaa    18,Y
        jsr     iscomsp
        cmpb    #'1'
        bne     synerr
        ldaa    19,Y
        jsr     issign
        cmpb    #'1'
        bne     synerr
        ldaa    20,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr
        ldaa    21,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr
        ldaa    22,Y
        jsr     isdec
        cmpb    #'1'
        bne     synerr
        ldaa    23,Y
        jsr     isint
        cmpb    #'1'
        bne     synerr

noerr   ldaa    #'1'
        staa    datain
        bra     zcheck

synerr  jsr     wrerr0
        ldaa    #'0'
        staa    datain
        staa    clawcmd
        bra     zcheck

zcheck  nop
        rts

**********
* docmds -- Move manipulator to commanded position
**********
docmds  ldaa    fcmd1
        cmpa    #'0'
```

```
        beq     doem1
        jsr     outcmd1
wait1   ldaa    ffbk1
        cmpa    #'1'
        beq     adocmds
        bra     wait1
adocmds jsr     doout
doem1   ldaa    fcmd2
        cmpa    #'0'
        beq     doem2
        jsr     outcmd2
wait2   ldaa    ffbk2
        cmpa    #'1'
        beq     bdocmds
        bra     wait2
bdocmds jsr     doout
doem2   nop
        ldaa    fcmd3
        cmpa    #'0'
        beq     doem3
        jsr     outcmd3
wait3   ldaa    ffbk3
        cmpa    #'1'
        beq     cdocmds
        ldaa    fstop
        cmpa    #'1'
        bne     wait3
        ldaa    #'0'
        staa    fstop
        staa    fcmd1
        staa    fcmd2
        bra     bdocmds
cdocmds jsr     doout
doem3   jsr     grip
        jsr     doout
        bset    TFLG1,X #%00001000      * Clear IC4 flag
        bset    TMSK1,X #%00001000      * Enable IC4 interrupt
ddocmds ldaa    #'0'
        staa    datain
zdocmds rts

**********
* docmds2 -- Move manipulator to commanded position in reverse order
**********
docmds2 ldaa    fcmd3
        cmpa    #'0'
        beq     doem21
        jsr     outcmd3
wait21  ldaa    ffbk3
        cmpa    #'1'
        beq     adocmd2
        bra     wait21
adocmd2 jsr     doout
doem21  ldaa    fcmd2
        cmpa    #'0'
        beq     doem22
        jsr     outcmd2
wait22  ldaa    ffbk2
        cmpa    #'1'
        beq     bdocmd2
        bra     wait22
bdocmd2 jsr     doout
doem22  ldaa    fcmd1
        cmpa    #'0'
        beq     doem23
        jsr     outcmd1
wait23  ldaa    ffbk1
        cmpa    #'1'
        beq     cdocmd2
```

```
        bra     wait23
cdocmd2 jsr     doout
doem23  jsr     grip
        jsr     doout
        ldaa    #'0'
        staa    datain
zdocmd2 rts

**********
* doout -- write output to screen is necessary.
**********
doout   ldaa    ffbk1
        suba    #$30
        adda    ffbk2
        suba    #$30
        adda    ffbk3
        suba    #$30
        adda    ffbkc
        suba    #$30
        cmpa    #$00
        beq     zdoout
        ldaa    #'0'
        staa    ffbk1
        staa    ffbk2
        staa    ffbk3
        staa    ffbkc
        jsr     PRNTDAT
zdoout  rts

**********
* findit -- Search for object around given approximate position.  Algorithm
*               assumes that the beginning (input) point is the beginning of
*               the search.  This point lies at the center of an imaginary
*               tic-tac-toe board.  All nine points are sounded and their
*               heights (th3 values) are stored in hgt1-hgt9.  If the center
*               point is del_z above the perimeter, then the claw makes a
*               grab at the center point.  If any of the perimeter squares
*               are del_z taller than the center,
*               that point becomes the new center and the search pattern is
*               run again.  If two perimeter points are both equal and
*               taller than the center, then...  This process repeats
*               until...
*               If (hgt5-del_z > all other hgt values)
*                       Lift at center
*               Else if (hgti
*               Case 2:
*
*               cntri (i=1:9) are the centers of imaginary tic-tac-toe
*               boards that will be used in the search for the object.
*               boxi (i=1:9) are the individual squares of the particular
*               tic-tac-toe board currently being searched.
**********
findit  psha
        pshb
        pshx
        pshy

        ldy     #rdata          * Save rdata to cntr1 for later use.
        ldx     #cntr1
svcntr1 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr1

        ldy     #rdata          * Save rdata to cntr2 for later use.
        ldx     #cntr2
svcntr2 ldaa    0,Y
```

```
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr2

        ldy     #rdata          * Save rdata to cntr3 for later use.
        ldx     #cntr3
svcntr3 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr3

        ldy     #rdata          * Save rdata to cntr4 for later use.
        ldx     #cntr4
svcntr4 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr4

        ldy     #rdata          * Save rdata to cntr5 for later use.
        ldx     #cntr5
svcntr5 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr5

        ldy     #rdata          * Save rdata to cntr6 for later use.
        ldx     #cntr6
svcntr6 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr6

        ldy     #rdata          * Save rdata to cntr7 for later use.
        ldx     #cntr7
svcntr7 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr7

        ldy     #rdata          * Save rdata to cntr8 for later use.
        ldx     #cntr8
svcntr8 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr8

        ldy     #rdata          * Save rdata to cntr9 for later use.
        ldx     #cntr9
svcntr9 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svcntr9
```

```
        jsr     pattbig         * Modify cntr values to show spread.

        ldaa    #'1'            * Set cntr to search around first
        staa    try

search  ldy     #rdata          * Save cntri to rdata to be searched.  If
        ldaa    try             * all 9 grids have been searched, then
try1    cmpa    #'1'            * give up.
        bne     try2
        ldx     #cntr1
        bra     strdata
try2    cmpa    #'2'
        bne     try3
        ldx     #cntr2
        bra     strdata
try3    cmpa    #'3'
        bne     try4
        ldx     #cntr3
        bra     strdata
try4    cmpa    #'4'
        bne     try5
        ldx     #cntr4
        bra     strdata
try5    cmpa    #'5'
        bne     try6
        ldx     #cntr5
        bra     strdata
try6    cmpa    #'6'
        bne     nomore          * six square case
        ldx     #cntr6
        bra     strdata
nomore  jmp     giveup
strdata ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     strdata

        ldy     #rdata          * Save rdata to box1 for later use.
        ldx     #box1
svrdat1 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat1

        ldy     #rdata          * Save rdata to box2 for later use.
        ldx     #box2
svrdat2 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat2

        ldy     #rdata          * Save rdata to box3 for later use.
        ldx     #box3
svrdat3 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat3

        ldy     #rdata          * Save rdata to box4 for later use.
        ldx     #box4
svrdat4 ldaa    0,Y
```

```
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat4

        ldy     #rdata          * Save rdata to box5 for later use.
        ldx     #box5
svrdat5 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat5

        ldy     #rdata          * Save rdata to box6 for later use.
        ldx     #box6
svrdat6 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat6

        ldy     #rdata          * Save rdata to box7 for later use.
        ldx     #box7
svrdat7 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat7

        ldy     #rdata          * Save rdata to box8 for later use.
        ldx     #box8
svrdat8 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat8

        ldy     #rdata          * Save rdata to box9 for later use.
        ldx     #box9
svrdat9 ldaa    0,Y
        staa    0,X
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat9

        jsr     pattern         * Modify box positions to spread pattern

        ldaa    #'1'            * Close claw
        staa    clawcmd
        jsr     docmds

        ldy     #rdata          * Save box1 to rdata for sounding.
        ldx     #box1
ldrdat1 ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     ldrdat1

        ldaa    #'1'            * Sound box1
        staa    datain
        jsr     makecmd
```

```
         jsr     docmds
         ldd     th3s
         std     hgt1
         jsr     gotop

         ldy     #rdata          * Save box2 to rdata for sounding.
         ldx     #box2
ldrdat2  ldaa    0,X
         staa    0,Y
         iny
         inx
         cpy     #rdata+#MAXIN
         blt     ldrdat2

         ldaa    #'1'            * Sound box2
         staa    datain
         jsr     makecmd
         jsr     docmds
         ldd     th3s
         std     hgt2
         jsr     gotop

         ldy     #rdata          * Save box3 to rdata for sounding.
         ldx     #box3
ldrdat3  ldaa    0,X
         staa    0,Y
         iny
         inx
         cpy     #rdata+#MAXIN
         blt     ldrdat3

         ldaa    #'1'            * Sound box3
         staa    datain
         jsr     makecmd
         jsr     docmds
         ldd     th3s
         std     hgt3
         jsr     gotop

         ldy     #rdata          * Save box6 to rdata for sounding.
         ldx     #box6
ldrdat6  ldaa    0,X
         staa    0,Y
         iny
         inx
         cpy     #rdata+#MAXIN
         blt     ldrdat6

         ldaa    #'1'            * Sound box6
         staa    datain
         jsr     makecmd
         jsr     docmds
         ldd     th3s
         std     hgt6
         jsr     gotop

         ldy     #rdata          * Save box5 to rdata for sounding.
         ldx     #box5
ldrdat5  ldaa    0,X
         staa    0,Y
         iny
         inx
         cpy     #rdata+#MAXIN
         blt     ldrdat5

         ldaa    #'1'            * Sound box5
         staa    datain
         jsr     makecmd
         jsr     docmds
```

```
        ldd     th3s
        std     hgt5
        jsr     gotop

        ldy     #rdata          * Save box4 to rdata for sounding.
        ldx     #box4
ldrdat4 ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     ldrdat4

        ldaa    #'1'            * Sound box4
        staa    datain
        jsr     makecmd
        jsr     docmds
        ldd     th3s
        std     hgt4
        jsr     gotop

        ldy     #rdata          * Save box7 to rdata for sounding.
        ldx     #box7
ldrdat7 ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     ldrdat7

        ldaa    #'1'            * Sound box7
        staa    datain
        jsr     makecmd
        jsr     docmds
        ldd     th3s
        std     hgt7
        jsr     gotop

        ldy     #rdata          * Save box8 to rdata for sounding.
        ldx     #box8
ldrdat8 ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     ldrdat8

        ldaa    #'1'            * Sound box8
        staa    datain
        jsr     makecmd
        jsr     docmds
        ldd     th3s
        std     hgt8
        jsr     gotop

        ldy     #rdata          * Save box9 to rdata for sounding.
        ldx     #box9
ldrdat9 ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     ldrdat9

        ldaa    #'1'            * Sound box9
        staa    datain
        jsr     makecmd
        jsr     docmds
        ldd     th3s
```

```
        std     hgt9
        jsr     gotop


        ldd     hgt1            * Search all values of hgt to find the lowest
        cpd     hgt2            * value.  Subtract this value from each
        ble     sort1           * hgt.
        ldd     hgt2
sort1   cpd     hgt3
        ble     sort2
        ldd     hgt3
sort2   cpd     hgt4
        ble     sort3
        ldd     hgt4
sort3   cpd     hgt5
        ble     sort4
        ldd     hgt5
sort4   cpd     hgt6
        ble     sort5
        ldd     hgt6
sort5   cpd     hgt7
        ble     sort6
        ldd     hgt7
sort6   cpd     hgt8
        ble     sort7
        ldd     hgt8
sort7   cpd     hgt9
        ble     sorted
        ldd     hgt9
sorted  std     tmp16
        ldd     hgt1
        subd    tmp16
        std     hgt1
        ldd     hgt2
        subd    tmp16
        std     hgt2
        ldd     hgt3
        subd    tmp16
        std     hgt3
        ldd     hgt4
        subd    tmp16
        std     hgt4
        ldd     hgt5
        subd    tmp16
        std     hgt5
        ldd     hgt6
        subd    tmp16
        std     hgt6
        ldd     hgt7
        subd    tmp16
        std     hgt7
        ldd     hgt8
        subd    tmp16
        std     hgt8
        ldd     hgt9
        subd    tmp16
        std     hgt9

        nop                     * Integer divide each hgt by ZDPI/4 (1/4")
        ldd     #ZDPI
        ldx     #4
        idiv
        pshx
        pula
        pulb
        std     tmp16
        ldd     hgt1
        ldx     tmp16
        idiv
```

```
        pshx
        pula
        pulb
        std     hgt1
        ldd     hgt2
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt2
        ldd     hgt3
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt3
        ldd     hgt4
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt4
        ldd     hgt5
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt5
        ldd     hgt6
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt6
        ldd     hgt7
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt7
        ldd     hgt8
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt8
        ldd     hgt9
        ldx     tmp16
        idiv
        pshx
        pula
        pulb
        std     hgt9

* Decide whether to:
*       (1)  Lift at a box within grid.
*       (2)  Make another location the center of new search pattern.
*       (3)  Give up.


        ldaa    #'1'
        staa    target
```

```
        ldd     hgt1            * Search all values of hgt to find the
        cpd     hgt2            * greatest value of hgt.  Set target =
        bge     tall1           * the box number of the first occurence
        ldaa    #'2'            * of the greatest value.
        staa    target
        ldd     hgt2
tall1   cpd     hgt3
        bge     tall2
        ldaa    #'3'
        staa    target
        ldd     hgt3
tall2   cpd     hgt4
        bge     tall3
        ldaa    #'4'
        staa    target
        ldd     hgt4
tall3   cpd     hgt5
        bge     tall4
        ldaa    #'5'
        staa    target
        ldd     hgt5
tall4   cpd     hgt6
        bge     tall5
        ldaa    #'6'
        staa    target
        ldd     hgt6
tall5   cpd     hgt7
        bge     tall6
        ldaa    #'7'
        staa    target
        ldd     hgt7
tall6   cpd     hgt8
        bge     tall7
        ldaa    #'8'
        staa    target
        ldd     hgt8
tall7   cpd     hgt9
        bge     tall8
        ldaa    #'9'
        staa    target
        ldd     hgt9
tall8   cpd     #3
        bgt     pickup
        ldaa    try
        inca
        staa    try
        jmp     search

pickup  nop
        ldy     #rdata          * Save target box to rdata for pickup.
        ldaa    #MAXIN          *       Convert target to hex, subtract 1,
        ldab    target          *       multiply by MAXIN, Add box1 address,
        subb    #$30            *       store in X.
        subb    #1
        mul
        addd    #box1
        pshb
        psha
        pulx
svrdat  ldaa    0,X
        staa    0,Y
        iny
        inx
        cpy     #rdata+#MAXIN
        blt     svrdat

        jsr     wrbox           * Print search points
        jsr     showmap         * Print map
```

```
        ldaa    #'0'            * Open claw to prepare for pickup
        staa    clawcmd
        jsr     docmds

        ldaa    #'1'            * Pick up object
        staa    clawcmd
        staa    datain
        jsr     makecmd
        jsr     docmds
giveup  jsr     gotop

        puly
        pulx
        pulb
        pula
        rts

**********
* gohome -- Return to home position
**********
homept  fcc     '+00,+00,+000,+07.0,+03.0'
gohome  ldy     #R_DATA
        ldx     #homept
        ldaa    #'1'
        staa    fcmd1
        staa    fcmd2
        staa    fcmd3
        staa    fcmdc
        ldaa    #'0'
        staa    clawcmd
gohome1 ldaa    0,X
        staa    0,Y
*        jsr     putacia
        inx
        iny
        cpy     #R_DATA+#MAXIN
        blt     gohome1
*        ldaa    #CR
*        jsr     putacia
        jsr     checkin
        jsr     makecmd
        jsr     docmds2
*        jsr     doout
        rts

**********
* gotop -- Take claw from current position all the way to the top.
**********
gotop   psha
        pshb
        pshx
        pshy
        ldaa    #'1'                    * Take claw to top
        staa    fcmd1
        staa    fcmd2
        staa    fcmd3
        staa    fcmdc
        ldaa    #'1'
*c        staa    clawcmd
        ldx     #R_DATA+#MAXIN-#ZCHAR   * Note +09.0 is out of range
        ldaa    #' '                    * and forces the claw all the way up.
        staa    0,X
        inx
        ldaa    #'0'
        staa    0,X
        inx
*        ldaa    #'9'                    * Temporarily make position 00.0
        ldaa    #'0'
        staa    0,X
```

```
        inx
        ldaa    #'.'
        staa    0,X
        inx
        ldaa    #'0'
        staa    0,X
        jsr     checkin
        jsr     makecmd
        jsr     docmds2
        puly
        pulx
        pulb
        pula
        rts


**********
* grip -- Activates or deactivates claw
**********
grip    ldx     #REGBAS
        ldaa    clawcmd
        cmpa    #'1'                    * If '1' then turn claw on
        beq     clawon
clawoff bclr    PORTA,X #%10000000      * Otherwise turn claw off
        bra     zgrip                   * return
clawon  bset    PORTA,X #%10000000
zgrip   ldaa    #'1'
        staa    ffbkc
        rts

**********
* inacia -- reads MAXIN characters in ACIA. Stores at R_DATA.
**********
inacia  jsr     syntax
        ldaa    #'0'
        staa    datain
        ldaa    #'1'
*c        staa    clawcmd               * close claw when position reached
        ldy     #R_DATA
        ldab    ACIA                    * read ACIA status register
        bitb    #$01                    * check LSB, if 1, then new character
        beq     zinacia                 * if no new character, return
READIT  ldaa    ACIA+1                  * else read character one
        anda    #$ff                    * this changes CCR, not needed here
        staa    0,Y                     * store charater one
        jsr     putacia                 * echo character
        iny                             * increment character pointer
        cpy     #R_DATA+#MAXIN-#$01     * compare pointer to max pointer
        bls     GETNEXT                 * if not all in, goto GETNEXT
        ldy     #R_DATA                 * else reset character pointer
        bra     zinacia                 * always return
GETNEXT ldab    ACIA                    * read ACIA status register
        bitb    #$01                    * check LSB, if 1, then new character
        beq     GETNEXT                 * repeat until new character in
        bra     READIT                  * read and save character
zinacia nop
        ldaa    #CR
        jsr     putacia
        rts                             * return

**********
* init -- initial process for the program.  Call this function in the
*       first step of the main program.
**********
init    ldaa    #SPACE                  ** Without these three lines,
        jsr     putacia                 ** approximately 10 characters
        jsr     putacia                 ** of nonsense print prior to
*                                       ** the first outcmd. ?????????
        ldx     #REGBAS
```

```
        ldy     #R_DATA
        jsr     onacia
        jsr     onsci
        jsr     INITPA
        jsr     INIVAR
zinit   rts

**********
* INITPA -- initializes PORT A (68HC11E9)
**********
INITPA  ldx     #REGBAS
        ldaa    #%10000100
        staa    PACTL,X
        ldaa    #%01110000              * P0:000,P1:001,P2:010,P3:011
        staa    PORTA,X                 * P4:100,P5:101,P6:110,P7:111
        ldaa    #%00001111              * reset interrupt flags IC4-IC1
        staa    TFLG1,X
        ldaa    #%00001111              * Enable ICi interrupts
        staa    TMSK1,X
        bset    TCTL2,X #%10101010      * Interupt ICi on falling edge
        rts

**********
* INIVAR -- Initializes variables
**********
INIVAR  ldy     #R_DATA
        ldaa    #SPACE
INIVAR1 staa    0,Y
        iny
        cpy     #R_DATA+#MAXIN-#$01
        bls     INIVAR1
*        ldy     #R_DATA
        ldaa    #'0'
        staa    fcmd1
        staa    fcmd2
        staa    fcmd3
        staa    fcmdc
        staa    ffbk1
        staa    ffbk2
        staa    ffbk3
        staa    ffbkc
        staa    fstop
        staa    clawcmd
        ldd     #$0000
        std     th1
        std     th2
        std     th3
        ldaa    #'X'
        ldy     #sci0_in
inits0  staa    0,Y
        iny
        cpy     #sci0_in+#MAXSCI-#$01
        bls     inits0
        ldy     #sci1_in
inits1  staa    0,Y
        iny
        cpy     #sci1_in+#MAXSCI-#$01
        bls     inits1
        ldy     #sci2_in
inits2  staa    0,Y
        iny
        cpy     #sci2_in+#MAXSCI-#$01
        bls     inits2
        ldaa    #0
        staa    hgt1
        staa    hgt1+1
        staa    hgt2
        staa    hgt2+1
        staa    hgt3
```

```
        staa     hgt3+1
        staa     hgt4
        staa     hgt4+1
        staa     hgt5
        staa     hgt5+1
        staa     hgt6
        staa     hgt6+1
        staa     hgt7
        staa     hgt7+1
        staa     hgt8
        staa     hgt8+1
        staa     hgt9
        staa     hgt9+1
ZINIVAR rts

**********
* iscomsp -- Writes '1' to accumulator B if character in accumulator A is
*               ',' or ' ' else writes '0'.
**********
iscomsp ldab     #'1'
        cmpa     #','
        beq      ziscom
        cmpa     #' '
        beq      ziscom
        ldab     #'0'
ziscom  rts

**********
* isdec -- Writes '1' to accumulator B if character in accumulator A is
*               '.' else writes '0'.
**********
isdec   ldab     #'1'
        cmpa     #'.'
        beq      zisdec
        ldab     #'0'
zisdec  rts

**********
* isint -- Writes '1' to accumulator B if character in accumulator A is
*               '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', else
*               writes '0'.
**********
isint   ldab     #'1'
        cmpa     #'0'
        beq      zisint
        cmpa     #'1'
        beq      zisint
        cmpa     #'2'
        beq      zisint
        cmpa     #'3'
        beq      zisint
        cmpa     #'4'
        beq      zisint
        cmpa     #'5'
        beq      zisint
        cmpa     #'6'
        beq      zisint
        cmpa     #'7'
        beq      zisint
        cmpa     #'8'
        beq      zisint
        cmpa     #'9'
        beq      zisint
        ldab     #'0'
zisint  rts

**********
* issign -- Writes '1' to accumulator B if character in accumulator A is
*               '+' or '-' else writes '0'.
```

```
**********
issign  ldab    #'1'
        cmpa    #'+'
        beq     zissign
        cmpa    #'-'
        beq     zissign
        cmpa    #' '
        beq     zissign
        ldab    #'0'
zissign rts

**********
* makecmd -- Convert input string into motor commands.
**********
makecmd ldaa    datain                  * if data in no good, quit
        cmpa    #'1'
        beq     begmake
        jmp     zmakcmd

begmake nop                             * parse received string into
*                                       * individual strings

        ldx     #R_DATA
        ldy     #pitch
getp    ldaa    0,X                     * store string in pitch
        staa    0,Y
        inx
        iny
        cpy     #pitch+#PCHAR
        blt     getp


        inx
        ldy     #roll
getq    ldaa    0,X                     * store string in roll
        staa    0,Y
        inx
        iny
        cpy     #roll+#QCHAR
        blt     getq


        inx
        ldy     #theta
getth   ldaa    0,X                     * store string in theta
        staa    0,Y
        inx
        iny
        cpy     #theta+#TCHAR
        blt     getth


        inx
        ldy     #radius
getrad  ldaa    0,X                     * store string in radius
        staa    0,Y
        inx
        iny
        cpy     #radius+#RCHAR
        blt     getrad


        inx
        ldy     #height
gethght ldaa    0,X                     * store string in height
        staa    0,Y
        inx
        iny
        cpy     #height+#ZCHAR
```

```
        blt     gethght

maketh1 ldab    theta+#TCHAR-#$01       * Convert theta string to th1 (hex)
        jsr     ASC2HEX                 * 1's place
        ldaa    #$00
        std     th1
        ldab    theta+#TCHAR-#$02       * 10's place
        jsr     ASC2HEX
        ldaa    #10
        mul
        addd    th1
        std     th1
        ldab    theta+#TCHAR-#$03       * 100's place
        jsr     ASC2HEX
        ldaa    #100
        mul
        addd    th1
        std     th1                     * Absolute value ok
        ldab    theta                   * If "-" leave positive
        cmpb    #'-'                    * If "+' make negative
        beq     mkth1a                  * Since motor sign opposite of
        ldd     #$0000                  * coordinate system sign
        subd    th1
        std     th1
mkth1a  ldaa    #'1'
        staa    fcmd1

maketh2 ldab    radius+#RCHAR-#$01      * Convert radius string to th2 (hex)
        cmpb    #'5'                    * If tenths place >= 5 then
        blt     mkth2a                  *    use 1/2 in. else truncate.
        ldd     #RDPI
        ldx     #$0002
        idiv
        pshx
        pula
        pulb
        bra     mkth2b
mkth2a  ldd     #$0000
mkth2b  std     th2
        ldab    radius+#RCHAR-#$03
        jsr     ASC2HEX                 * 1's place
        ldaa    #RDPI                   * Degrees per one inch
        mul
        addd    th2
        std     th2
        ldab    radius+#RCHAR-#$04
        jsr     ASC2HEX                 * 10's place
        ldaa    #$00
        pshb
        psha
        puly
mkth2c  cpy     #$0000
        beq     mkth2d
        ldx     #10
mkth2f  ldd     #RDPI                   * Degrees per one inch
        addd    th2
        std     th2
        dex
        bne     mkth2f
        dey
        bra     mkth2c
mkth2d  nop
        ldab    radius                  * If "-" leave positive
        cmpb    #'-'                    * If "+' make negative
        beq     mkth2e                  * Since motor sign opposite of
        ldd     #$0000                  * coordinate system sign
        subd    th2
        std     th2
```

```
mkth2e  ldd     #RZERO                  * Add count/coordinate system offset
        addd    th2
        std     th2
        ldaa    #'1'                    * Set command flag
        staa    fcmd2

maketh3 ldab    height+#RCHAR-#$01      * Convert height string to th3 (hex)
        cmpb    #'5'                    * If tenths place >= 5 then
        blt     mkth3a                  *    use 1/2 in. else truncate.
        ldd     #ZDPI
        ldx     #$0002
        idiv
        pshx
        pula
        pulb
        bra     mkth3b
mkth3a  ldd     #$0000
mkth3b  std     th3
        ldab    height+#RCHAR-#$03
        jsr     ASC2HEX                 * 1's place
        ldaa    #ZDPI                   * Degrees per one inch
        mul
        addd    th3
        std     th3
        ldab    height+#RCHAR-#$04
        jsr     ASC2HEX                 * 10's place
        ldaa    #$00
        pshb
        psha
        puly
mkth3c  cpy     #$0000
        beq     mkth3d
        ldx     #10
mkth3f  ldd     #ZDPI                   * Degrees per one inch
        addd    th3
        std     th3
        dex
        bne     mkth3f
        dey
        bra     mkth3c
mkth3d  nop
        ldab    height                  * If "-" make negative
        cmpb    #'+'                    * otherwise make positive
        beq     mkth3e                  * Since motor sign is the same as
        cmpb    #' '                    * otherwise make positive
        beq     mkth3e                  * Since motor sign is the same as
        ldd     #$0000                  * coordinate system sign
        subd    th3
        std     th3
mkth3e  ldd     #ZZERO                  * Add count/coordinate system offset
        addd    th3
        std     th3
        ldaa    #'1'                    * Set command flag
        staa    fcmd3


zmakcmd nop
        rts

**********
* outcmd1 -- Sends motor 1 command out sci port.
**********
outcmd1 ldaa    #241
        jsr     putsci
        ldaa    th1
        jsr     scib2h
        ldaa    th1+#$01
        jsr     scib2h
        ldaa    #'0'
```

```
        staa    fcmd1
        rts

**********
* outcmd2 -- Sends motor 2 command out sci port.
**********
outcmd2 ldaa    #242
        jsr     putsci
        ldaa    th2
        jsr     scib2h
        ldaa    th2+#$01
        jsr     scib2h
        ldaa    #'0'
        staa    fcmd2
        rts

**********
* outcmd3 -- Sends motor 3 command out sci port.
**********
outcmd3 ldaa    #243
        jsr     putsci
        ldaa    th3
        jsr     scib2h
        ldaa    th3+#$01
        jsr     scib2h
        ldaa    #'0'
        staa    fcmd3
        rts

**********
* pattbig -- Define centers of overall search pattern.  These values
*               will be used to start
*               searches if the first grid search is not successful.
*
*               cntr2:  r = r5 + 3*del_r
*                       theta2 = theta5
*               cntr3:  r = r5 + 3*del_r
*                       theta3 = theta5 + 3*del_theta
*               cntr4:  r = r5
*                       theta3 = theta5 + 3*del_theta
*               cntr5:  r = r5 - 3*del_r
*                       theta3 = theta5 + 3*del_theta
*               cntr6:  r = r5 - 3*del_r
*                       theta3 = theta5
*               cntr7:  r = r5 - 3*del_r
*                       theta3 = theta5 - 3*del_theta
*               cntr8:  r = r5
*                       theta3 = theta5 - 3*del_theta
*               cntr9:  r = r5 + 3*del_r
*                       theta3 = theta5 - 3*del_theta
*********
pattbig psha
        pshb
        pshx
        pshy

* Determine the number of whole & half inches in 3*del_r.
        ldd     del_r
        addd    del_r
        addd    del_r
        ldx     #2
        idiv
        std     half_in
        pshx
        pula
        pulb
        std     whole_in

* Modify r values of cntr3, cntr4, and cntr5.
```

```
        ldaa    #$00            * Read string.  Convert to hex.
        ldab    cntr1+#15
        subb    #$30
        std     tmp16
        ldab    cntr1+#14
        subb    #$30
        ldaa    #10
        mul
        addd    tmp16
        std     tmp16

        addd    whole_in        * Add whole_in to hex version of rr string

        ldx     #10             * Convert new hex value to rr ASCII string
        idiv
        addd    #$30
        stab    cntr5+#15
        stab    cntr4+#15
        stab    cntr3+#15
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr5+#14
        stab    cntr4+#14
        stab    cntr3+#14

        ldd     half_in         * if (half_in==1) then rr.r = rr.r+.5
        cpd     #1
        bne     nohafs1
        ldaa    cntr1+#17
        cmpa    #'5'
        beq     hafsin1
        ldaa    #'5'
        staa    cntr5+#17
        staa    cntr4+#17
        staa    cntr3+#17
        bra     nohafs1
hafsin1 ldaa    #'0'
        staa    cntr5+#17
        staa    cntr4+#17
        staa    cntr3+#17
        ldaa    cntr1+#15
        adda    #1
        cmpa    #'9'
        bgt     carrys1
        staa    cntr5+#15
        staa    cntr4+#15
        staa    cntr3+#15
        bra     nohafs1
carrys1  ldaa    #'0'
        staa    cntr5+#15
        staa    cntr4+#15
        staa    cntr3+#15
        ldaa    cntr1+#14
        adda    #1
        staa    cntr5+#14
        staa    cntr4+#14
        staa    cntr3+#14
nohafs1 nop

* Modify r values of cntr7, cntr8, and cntr9
        ldd     tmp16           * Load hex version of rr string

        subd    whole_in        * Subtract whole_in from hex version of rr

        ldx     #10             * Convert new hex value to rr ASCII string
        idiv
        addd    #$30
```

```
        stab    cntr7+#15
        stab    cntr8+#15
        stab    cntr9+#15
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr7+#14
        stab    cntr8+#14
        stab    cntr9+#14

        ldd     half_in         * if (half_in==1) then rr.r = rr.r+.5
        cpd     #1
        bne     nohafs2
        ldaa    cntr1+#17
        cmpa    #'0'
        beq     hafsin2
        ldaa    #'0'
        staa    cntr7+#17
        staa    cntr8+#17
        staa    cntr9+#17
        bra     nohafs2
hafsin2 ldaa    #'5'
        staa    cntr7+#17
        staa    cntr8+#17
        staa    cntr9+#17
        ldaa    cntr1+#15
        suba    #1
        cmpa    #'0'
        blt     carrys2
        staa    cntr7+#15
        staa    cntr8+#15
        staa    cntr9+#15
        bra     nohafs2
carrys2  ldaa    #'9'
        staa    cntr7+#15
        staa    cntr8+#15
        staa    cntr9+#15
        ldaa    cntr1+#14
        suba    #1
        staa    cntr7+#14
        staa    cntr8+#14
        staa    cntr9+#14
nohafs2 nop

* Modify theta values of cntr5, cntr6, and cntr7
        ldaa    #$00            * Read string.  Convert to hex.
        ldab    cntr1+#11
        subb    #$30
        std     tmp16
        ldab    cntr1+#10
        subb    #$30
        ldaa    #10
        mul
        addd    tmp16
        std     tmp16
        ldab    cntr1+#9
        subb    #$30
        ldaa    #100
        mul
        addd    tmp16
        std     tmp16

        subd    del_theta       * Subtract 3*del_theta from hex version of
*                               * theta string. Same as adding and including
*                               * sign on theta.  Sign ignored since always
*                               * negative.
        subd    del_theta
        subd    del_theta
```

```
        ldx     #100            * Convert hex value to theta ASCII string
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr5+#9
        stab    cntr6+#9
        stab    cntr7+#9
        pulb
        pula
        ldx     #10
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr5+#10
        stab    cntr6+#10
        stab    cntr7+#10
        pulb
        pula
        addd    #$30
        stab    cntr5+#11
        stab    cntr6+#11
        stab    cntr7+#11

* Modify theta values of cntr3, cntr2, and cntr9
        ldd     tmp16           * Load hex version of rr string

        addd    del_theta       * Add 3*del_theta from hex version of
*                               * theta string. Same as subtracting and
*                               * including sign on theta.  Sign ignored
*                               * since always negative.
        addd    del_theta
        addd    del_theta

        ldx     #100            * Convert hex value to theta ASCII string
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr3+#9
        stab    cntr2+#9
        stab    cntr9+#9
        pulb
        pula
        ldx     #10
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    cntr3+#10
        stab    cntr2+#10
        stab    cntr9+#10
        pulb
        pula
        addd    #$30
```

```
        stab    cntr3+#11
        stab    cntr2+#11
        stab    cntr9+#11

        jsr     wrcntr          * Write out the calculated grid centers.

        puly
        pulx
        pulb
        pula
        rts

**********
* pattern -- Modify box strings to spread pattern.  Looking down from the
*               boom and towards boom end, search pattern looks like
*
*                       box1    box2    box3
*                       box4    box5    box6
*                       box7    box8    box9
*
*               Top row:        r = r5 + del_r
*               Bottom row:     r = r5 - del_r
*               Left Column:    theta = theta5 + del_theta
*               Right Column:   theta = theta5 - del_theta
**********
pattern psha
        pshb
        pshx
        pshy

* Determine the number of whole & half inches that r must be modified
        ldd     del_r
        ldx     #2
        idiv
        std     half_in
        pshx
        pula
        pulb
        std     whole_in

* Modify r values of box1, box2, and box3.
        ldaa    #$00            * Read string.  Convert to hex.
        ldab    box5+#15
        subb    #$30
        std     tmp16
        ldab    box5+#14
        subb    #$30
        ldaa    #10
        mul
        addd    tmp16
        std     tmp16

        addd    whole_in        * Add whole_in to hex version of rr string

        ldx     #10             * Convert new hex value to rr ASCII string
        idiv
        addd    #$30
        stab    box1+#15
        stab    box2+#15
        stab    box3+#15
        pshx
        pula
        pulb
        addd    #$30
        stab    box1+#14
        stab    box2+#14
        stab    box3+#14

        ldd     half_in         * if (half_in==1) then rr.r = rr.r+.5
```

```
        cpd     #1
        bne     nohalf1
        ldaa    box5+#17
        cmpa    #'5'
        beq     halfin1
        ldaa    #'5'
        staa    box1+#17
        staa    box2+#17
        staa    box3+#17
        bra     nohalf1
halfin1 ldaa    #'0'
        staa    box1+#17
        staa    box2+#17
        staa    box3+#17
        ldaa    box5+#15
        adda    #1
        cmpa    #'9'
        bgt     carry1
        staa    box1+#15
        staa    box2+#15
        staa    box3+#15
        bra     nohalf1
carry1  ldaa    #'0'
        staa    box1+#15
        staa    box2+#15
        staa    box3+#15
        ldaa    box5+#14
        adda    #1
        staa    box1+#14
        staa    box2+#14
        staa    box3+#14
nohalf1 nop

* Modify r values of box7, box8, and box9
        ldd     tmp16           * Load hex version of rr string

        subd    whole_in        * Subtract whole_in from hex version of rr

        ldx     #10             * Convert new hex value to rr ASCII string
        idiv
        addd    #$30
        stab    box7+#15
        stab    box8+#15
        stab    box9+#15
        pshx
        pula
        pulb
        addd    #$30
        stab    box7+#14
        stab    box8+#14
        stab    box9+#14

        ldd     half_in         * if (half_in==1) then rr.r = rr.r+.5
        cpd     #1
        bne     nohalf2
        ldaa    box5+#17
        cmpa    #'0'
        beq     halfin2
        ldaa    #'0'
        staa    box7+#17
        staa    box8+#17
        staa    box9+#17
        bra     nohalf2
halfin2 ldaa    #'5'
        staa    box7+#17
        staa    box8+#17
        staa    box9+#17
        ldaa    box5+#15
        suba    #1
```

```
        cmpa    #'0'
        blt     carry2
        staa    box7+#15
        staa    box8+#15
        staa    box9+#15
        bra     nohalf2
carry2  ldaa    #'9'
        staa    box7+#15
        staa    box8+#15
        staa    box9+#15
        ldaa    box5+#14
        suba    #1
        staa    box7+#14
        staa    box8+#14
        staa    box9+#14
nohalf2 nop

* Modify theta values of box1, box4, and box7
        ldaa    #$00            * Read string.  Convert to hex.
        ldab    box5+#11
        subb    #$30
        std     tmp16
        ldab    box5+#10
        subb    #$30
        ldaa    #10
        mul
        addd    tmp16
        std     tmp16
        ldab    box5+#9
        subb    #$30
        ldaa    #100
        mul
        addd    tmp16
        std     tmp16

        subd    del_theta       * Subtract del_theta from hex version of
*                               * theta string. Same as adding and including
*                               * sign on theta.  Sign ignored since always
*                               * negative.

        ldx     #100            * Convert hex value to theta ASCII string
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    box1+#9
        stab    box4+#9
        stab    box7+#9
        pulb
        pula
        ldx     #10
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    box1+#10
        stab    box4+#10
        stab    box7+#10
        pulb
        pula
        addd    #$30
        stab    box1+#11
        stab    box4+#11
```

```
        stab    box7+#11

* Modify theta values of box3, box6, and box9
        ldd     tmp16           * Load hex version of rr string

        addd    del_theta       * Add del_theta from hex version of
*                               * theta string. Same as subtracting and
*                               * including sign on theta.  Sign ignored
*                               * since always negative

        ldx     #100            * Convert hex value to theta ASCII string
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    box3+#9
        stab    box6+#9
        stab    box9+#9
        pulb
        pula
        ldx     #10
        idiv
        psha
        pshb
        pshx
        pula
        pulb
        addd    #$30
        stab    box3+#10
        stab    box6+#10
        stab    box9+#10
        pulb
        pula
        addd    #$30
        stab    box3+#11
        stab    box6+#11
        stab    box9+#11

        jsr     wrbox           * Print search box coordinates to screen

        puly
        pulx
        pulb
        pula
        rts

**********
* putaway -- Move claw to drop location and open.
**********
droppt  fcc     '+00,+00,-000,+10.5,-03.5'
putaway nop
*! new stuff to take claw to top before approaching drop point
*       ldx     #R_DATA
*deb1   ldaa    0,X
*       inx
*       cpx     #R_DATA+#MAXIN
*       blt     deb1
*
*       ldaa    #'1'                    * Take claw to top
*       staa    fcmd1
*       staa    fcmd2
*       staa    fcmd3
*       staa    fcmdc
*       ldaa    #'1'
*       staa    clawcmd
*       ldx     #R_DATA+#MAXIN-#ZCHAR   * Note +09.0 is out of range
```

```
*         ldaa    #' '                    * and forces the claw all the way up.
*         staa    0,X
*         inx
*         ldaa    #'0'
*         staa    0,X
*         inx
*         ldaa    #'9'
*         staa    0,X
*         inx
*         ldaa    #'.'
*         staa    0,X
*         inx
*         ldaa    #'0'
*         staa    0,X
*         jsr     checkin
*         jsr     makecmd
*         jsr     docmds2
*
*! end new stuff
          ldy     #R_DATA         * Go to drop point
          ldx     #droppt
          ldaa    #'1'
          staa    fcmd1
          staa    fcmd2
          staa    fcmd3
          staa    fcmdc
          ldaa    #'0'
          staa    clawcmd
ptaway2   ldaa    0,X
          staa    0,Y
          inx
          iny
          cpy     #R_DATA+#MAXIN
          blt     ptaway2
          jsr     checkin
          jsr     makecmd
          jsr     docmds
*          jsr     doout
          rts

**********
* PRNTDAT -- OUTPUT RELEVANT DATA TO SCREEN
**********
PRNTDAT   psha
          pshb
          pshx
          pshy

*         ldaa    #'0'                  * added for plot
*         jsr     putacia               * added for plot
          ldx     #REGBAS
          ldy     #sci0_in
prth1     ldaa    0,Y
          jsr     putacia
          iny
          cpy     #sci0_in+#MAXSCI-#2
          bne     prth1z
*         ldaa    #'h'
*         jsr     putacia
          ldaa    #SPACE
          jsr     putacia
prth1z    nop
*p          cpy     #sci0_in+#MAXSCI-#$01       * mode info removed for plot
          bls     prth1

          ldaa    #SPACE
          jsr     putacia
          jsr     putacia
```

```
*        ldaa    #'0'                    * added for plot
*        jsr     putacia                 * added for plot
         ldy     #sci1_in
prth2    ldaa    0,Y
         jsr     putacia
         iny
         cpy     #sci1_in+#MAXSCI-#2
         bne     prth2z
*        ldaa    #'h'                      * added for plot
*        jsr     putacia
         ldaa    #SPACE
         jsr     putacia
prth2z   nop
*p         cpy     #sci1_in+#MAXSCI-#$01       * mode info removed for plot
         bls     prth2

         ldaa    #SPACE
         jsr     putacia
         jsr     putacia

*        ldaa    #'0'                    * added for plot
*        jsr     putacia                 * added for plot
         ldy     #sci2_in
prth3    ldaa    0,Y
         jsr     putacia
         iny
         cpy     #sci2_in+#MAXSCI-#2
         bne     prth3z
*        ldaa    #'h'                      * added for plot
*        jsr     putacia
         ldaa    #SPACE
         jsr     putacia
prth3z   nop
*p         cpy     #sci2_in+#MAXSCI-#$01       * mode info removed for plot
         bls     prth3

         ldaa    #SPACE
         jsr     putacia
         jsr     putacia

prclw    ldab    PORTA,X
         bmi     itson
itsoff   ldaa    #'0'
         jsr     putacia
*        ldaa    #'O'
*        jsr     putacia
*        ldaa    #'p'
*        jsr     putacia
*        ldaa    #'e'
*        jsr     putacia
*        ldaa    #'n'
*        jsr     putacia
*        ldaa    #SPACE
*        jsr     putacia
*        jsr     putacia
         bra     ZPRNTDT
itson    ldaa    #'1'
         jsr     putacia
*        ldaa    #'C'
*        jsr     putacia
*        ldaa    #'l'
*        jsr     putacia
*        ldaa    #'o'
*        jsr     putacia
*        ldaa    #'s'
*        jsr     putacia
*        ldaa    #'e'
*        jsr     putacia
*        ldaa    #'d'
```

```
*         jsr     putacia

ZPRNTDT ldaa    #CR
        jsr     putacia

        puly
        pulx
        pulb
        pula
        rts

**********
* showmap -- print depth map to screen (3x3 grid)
**********
showmap psha
        pshb
        pshx
        pshy

        ldaa    hgt1
        jsr     bin2hex
        ldaa    hgt1+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt2
        jsr     bin2hex
        ldaa    hgt2+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt3
        jsr     bin2hex
        ldaa    hgt3+1
        jsr     bin2hex
        ldaa    #CR
        jsr     putacia
        ldaa    hgt4
        jsr     bin2hex
        ldaa    hgt4+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt5
        jsr     bin2hex
        ldaa    hgt5+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt6
        jsr     bin2hex
        ldaa    hgt6+1
        jsr     bin2hex
        ldaa    #CR
        jsr     putacia
        ldaa    hgt7
        jsr     bin2hex
        ldaa    hgt7+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt8
        jsr     bin2hex
        ldaa    hgt8+1
        jsr     bin2hex
        ldaa    #SPACE
        jsr     putacia
        ldaa    hgt9
        jsr     bin2hex
```

```
        ldaa    hgt9+1
        jsr     bin2hex
        ldaa    #CR
        jsr     putacia
        ldaa    target
        jsr     putacia
        ldaa    #CR
        jsr     putacia

        puly
        pulx
        pulb
        pula
        rts

**********
* syntax -- print input command syntax to screen
**********
syntax  nop
        ldaa    #'ñ'
        jsr     putacia
        ldaa    #'p'
        jsr     putacia
        ldaa    #'p'
        jsr     putacia
        ldaa    #','
        jsr     putacia
        ldaa    #'ñ'
        jsr     putacia
        ldaa    #'q'
        jsr     putacia
        ldaa    #'q'
        jsr     putacia
        ldaa    #','
        jsr     putacia
        ldaa    #'ñ'
        jsr     putacia
        ldaa    #'t'
        jsr     putacia
        ldaa    #'t'
        jsr     putacia
        ldaa    #'t'
        jsr     putacia
        ldaa    #','
        jsr     putacia
        ldaa    #'ñ'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #'.'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #','
        jsr     putacia
        ldaa    #'ñ'
        jsr     putacia
        ldaa    #'z'
        jsr     putacia
        ldaa    #'z'
        jsr     putacia
        ldaa    #'.'
        jsr     putacia
        ldaa    #'z'
        jsr     putacia
        ldaa    #CR
        jsr     putacia
```

```
        rts

**********
*  th3a2th3s() -  Converts 4 ascii bytes in th3a to 2 hex bytes in th3s.
**********
th3a2th3s nop
        psha
        ldaa    th3a
        jsr     HEXBIN
        ldaa    th3a+#1
        jsr     HEXBIN
        ldaa    SHFTREG+#1
        staa    th3s
        ldaa    th3a+#2
        jsr     HEXBIN
        ldaa    th3a+#3
        jsr     HEXBIN
        ldaa    SHFTREG+#1
        staa    th3s+#1
        pula
        rts

**********
*  th32hgts() -  Converts 2 hex bytes in th3 to 4 ascii
*                bytes in hgts (ñhh.h).
**********
th32hgts nop
        rts

**********
* wrbox -- print command lines for nine search points.
**********
wrbox   psha
        pshb
        pshx
        pshy

        ldy     #box1
wrbox1  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box1+#MAXIN
        blt     wrbox1
        ldaa    #CR
        jsr     putacia

        ldy     #box2
wrbox2  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box2+#MAXIN
        blt     wrbox2
        ldaa    #CR
        jsr     putacia

        ldy     #box3
wrbox3  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box3+#MAXIN
        blt     wrbox3
        ldaa    #CR
        jsr     putacia

        ldy     #box4
wrbox4  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box4+#MAXIN
```

```
        blt     wrbox4
        ldaa    #CR
        jsr     putacia

        ldy     #box5
wrbox5  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box5+#MAXIN
        blt     wrbox5
        ldaa    #CR
        jsr     putacia

        ldy     #box6
wrbox6  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box6+#MAXIN
        blt     wrbox6
        ldaa    #CR
        jsr     putacia

        ldy     #box7
wrbox7  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box7+#MAXIN
        blt     wrbox7
        ldaa    #CR
        jsr     putacia

        ldy     #box8
wrbox8  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box8+#MAXIN
        blt     wrbox8
        ldaa    #CR
        jsr     putacia

        ldy     #box9
wrbox9  ldaa    0,Y
        jsr     putacia
        iny
        cpy     #box9+#MAXIN
        blt     wrbox9
        ldaa    #CR
        jsr     putacia

        puly
        pulx
        pulb
        pula
        rts


**********
* wrcntr -- print command lines for nine search grid centers.
**********
wrcntr  psha
        pshb
        pshx
        pshy

        ldy     #cntr1
wrcntr1 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr1+#MAXIN
```

```
        blt     wrcntr1
        ldaa    #CR
        jsr     putacia

        ldy     #cntr2
wrcntr2 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr2+#MAXIN
        blt     wrcntr2
        ldaa    #CR
        jsr     putacia

        ldy     #cntr3
wrcntr3 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr3+#MAXIN
        blt     wrcntr3
        ldaa    #CR
        jsr     putacia

        ldy     #cntr4
wrcntr4 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr4+#MAXIN
        blt     wrcntr4
        ldaa    #CR
        jsr     putacia

        ldy     #cntr5
wrcntr5 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr5+#MAXIN
        blt     wrcntr5
        ldaa    #CR
        jsr     putacia

        ldy     #cntr6
wrcntr6 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr6+#MAXIN
        blt     wrcntr6
        ldaa    #CR
        jsr     putacia

        ldy     #cntr7
wrcntr7 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr7+#MAXIN
        blt     wrcntr7
        ldaa    #CR
        jsr     putacia

        ldy     #cntr8
wrcntr8 ldaa    0,Y
        jsr     putacia
        iny
        cpy     #cntr8+#MAXIN
        blt     wrcntr8
        ldaa    #CR
        jsr     putacia

        ldy     #cntr9
wrcntr9 ldaa    0,Y
```

```
        jsr     putacia
        iny
        cpy     #cntr9+#MAXIN
        blt     wrcntr9
        ldaa    #CR
        jsr     putacia

        puly
        pulx
        pulb
        pula
        rts


**********
* wrerr0 -- "syntax error"
**********
wrerr0  nop
        ldaa    #'W'
        jsr     putacia
        ldaa    #'R'
        jsr     putacia
        ldaa    #'E'
        jsr     putacia
        ldaa    #'R'
        jsr     putacia
        ldaa    #'R'
        jsr     putacia
        ldaa    #'0'
        jsr     putacia
        ldaa    #':'
        jsr     putacia
        ldaa    #SPACE
        jsr     putacia
        ldaa    #SPACE
        jsr     putacia
        ldaa    #'S'
        jsr     putacia
        ldaa    #'y'
        jsr     putacia
        ldaa    #'n'
        jsr     putacia
        ldaa    #'t'
        jsr     putacia
        ldaa    #'a'
        jsr     putacia
        ldaa    #'x'
        jsr     putacia
        ldaa    #SPACE
        jsr     putacia
        ldaa    #'e'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #'o'
        jsr     putacia
        ldaa    #'r'
        jsr     putacia
        ldaa    #CR
        jsr     putacia
        rts

*************************************************************************
*
*  THE INTERRUPT SERVICE ROUTINES
*
*************************************************************************
```

```
**********
* ACIA_ISR -- Interrupt service routine occurs when ACIA input received.
**********
ACIA_ISR nop
        jsr     inacia
        jsr     checkin
*        jsr     makecmd
        ldaa    datain
        cmpa    #'1'
        bne     zacia
        jsr     PRNTDAT          * print home position before maneuver.
zacia   rti

**********
* SCI0_ISR -- Interrupt service routine occurs when SCI0 input received.
**********
SCI0_ISR nop
*        ldaa    #'A'
*        jsr     putacia
        ldx     #REGBAS
        bclr    PORTA,X #%01110000
        jsr     getsci                  * read trigger character & discard
*        jsr     putacia
        ldy     #$FFFF                  * Y = $FFFF
getsci0 dey                             * Y = Y - 1
        beq     ZSCI0                   * if Y=0 then return
        ldab    SCSR,X
        bitb    #$20                    * if (character not received) then
        beq     getsci0                 *       goto getsci0

        ldaa    SCDR,X                  * read received character
*        jsr     putacia
        cmpa    #$F0                    * if (char != F0) then return
        bne     ZSCI0

GETEM0  ldy     #sci0_in                * read and store MAXSCI characters
sci0    jsr     getsci                  * echo characters, space, 0, CR
        cmpa    #$F0
        beq     GETEM0
        staa    0,Y
*        jsr     putacia
        iny
        cpy     #sci0_in+#MAXSCI-#$01
        bls     sci0
*        ldaa    #SPACE
*        jsr     putacia
*        ldaa    #'0'
*        jsr     putacia
*        ldaa    #CR
*        jsr     putacia
        ldaa    #'1'
        staa    ffbk1

ZSCI0   ldaa    #%00000001
        staa    TFLG1,X                 * Clear IC3 bit (PA0)
        ldaa    PORTA,X                 * PA6,PA5,PA4
        ora     #%01110000              * P0:000,P1:001,P2:010,P3:011
        staa    PORTA,X                 * P4:100,P5:101,P6:110,P7:111
        rti

**********
* SCI1_ISR -- Interrupt service routine occurs when SCI1 input received.
**********
SCI1_ISR nop
*        ldaa    #'B'
*        jsr     putacia
        ldx     #REGBAS
        bclr    PORTA,X #%01100000
        jsr     getsci                  * read trigger character & discard
```


C-2

```
*        jsr     putacia
         ldy     #$FFFF                  * Y = $FFFF
getsci1  dey                             * Y = Y - 1
         beq     ZSCI1                   * if Y=0 then return
         ldab    SCSR,X
         bitb    #$20                    * if (character not received) then
         beq     getsci1                 *       goto getsci1

         ldaa    SCDR,X                  * read received character
*        jsr     putacia
         cmpa    #$F0                    * if (char != F0) then return
         bne     ZSCI1

GETEM1   ldy     #sci1_in                * read and store MAXSCI characters
sci1     jsr     getsci                  * echo characters, space, 0, CR
         cmpa    #$F0
         beq     GETEM1
         staa    0,Y
*        jsr     putacia
         iny
         cpy     #sci1_in+#MAXSCI-#$01
         bls     sci1
*        ldaa    #SPACE
*        jsr     putacia
*        ldaa    #'1'
*        jsr     putacia
*        ldaa    #CR
*        jsr     putacia
         ldaa    #'1'
         staa    ffbk2

ZSCI1    ldaa    #%00000010
         staa    TFLG1,X                 * Clear IC2 bit (PA1)
         ldaa    PORTA,X                 * PA6,PA5,PA4
         ora     #%01110000              * P0:000,P1:001,P2:010,P3:011
         staa    PORTA,X                 * P4:100,P5:101,P6:110,P7:111
         rti


**********
* SCI2_ISR -- Interrupt service routine occurs when SCI2 input received.
**********
SCI2_ISR nop
*        ldaa    #'C'
*        jsr     putacia
         ldx     #REGBAS
         bclr    PORTA,X #%01010000
         jsr     getsci                  * read trigger character & discard
*        jsr     putacia
         ldy     #$FFFF                  * Y = $FFFF
getsci2  dey                             * Y = Y - 1
         beq     ZSCI2                   * if Y=0 then return
         ldab    SCSR,X
         bitb    #$20                    * if (character not received) then
         beq     getsci2                 *       goto getsci2

         ldaa    SCDR,X                  * read received character
*        jsr     putacia
         cmpa    #$F0                    * if (char != F0) then return
         bne     ZSCI2

GETEM2   ldy     #sci2_in                * read and store MAXSCI characters
sci2     jsr     getsci                  * echo characters, space, 0, CR
         cmpa    #$F0
         beq     GETEM2
         staa    0,Y
*        jsr     putacia
         iny
         cpy     #sci2_in+#MAXSCI-#$01
```

```
        bls     sci2
*        ldaa    #SPACE
*        jsr     putacia
*        ldaa    #'2'
*        jsr     putacia
*        ldaa    #CR
*        jsr     putacia
        ldaa    #'1'
        staa    ffbk3

ZSCI2   ldaa    #%00000100
        staa    TFLG1,X                 * Clear IC1 bit (PA2)
        ldaa    PORTA,X                 * PA6,PA5,PA4
        ora     #%01110000              * P0:000,P1:001,P2:010,P3:011
        staa    PORTA,X                 * P4:100,P5:101,P6:110,P7:111
        rti

**********
* SLAK_ISR -- Interrupt service routine occurs when PA3 goes low.
*             This occurs whenever cable tension goes slack.
**********
SLAK_ISR ldx    #REGBAS

        brclr   PORTA,X #%00001000 aslak    * Continue only if PA3 is low.
        jmp     zslak

aslak   bclr    TMSK1,X #%00001000      * Disable IC4 interrupt

        ldaa    #243            * write to SCI to stop motor 3
        jsr     putsci
        ldaa    #'S'
        jsr     putsci
        ldaa    #'T'
        jsr     putsci
        ldaa    #'O'
        jsr     putsci
        ldaa    #'P'
        jsr     putsci

* Wait for reply.  Update sci2_in.
        ldx     #REGBAS
        bclr    PORTA,X #%01010000
        jsr     getsci                  * read trigger character & discard
        ldy     #$FFFF                  * Y = $FFFF
getslk2 dey                             * Y = Y - 1
        beq     ZSLK2                   * if Y=0 then return
        ldab    SCSR,X
        bitb    #$20                    * if (character not received) then
        beq     getslk2                 *       goto getslk2

        ldaa    SCDR,X                  * read received character
        cmpa    #$F0                    * if (char != F0) then return
        bne     ZSLK2

GETSLK3 ldy     #sci2_in                * read and store MAXSCI characters
slk2    jsr     getsci                  * echo characters, space, 0, CR
        cmpa    #$F0
        beq     GETSLK3
        staa    0,Y
        iny
        cpy     #sci2_in+#MAXSCI-#$01
        bls     slk2

ZSLK2   ldaa    #%00000100
        staa    TFLG1,X                 * Clear IC1 bit (PA2)
        ldaa    PORTA,X                 * PA6,PA5,PA4
        ora     #%01110000              * P0:000,P1:001,P2:010,P3:011
        staa    PORTA,X                 * P4:100,P5:101,P6:110,P7:111
* above lines taken from sci2_isr
```

```
        jsr     doout

** Convert received position in sci2_in (ASCII) to desired th3 (ASCII)
* Strip th3a from sci2_in
        ldaa    sci2_in
        staa    th3a
        ldaa    sci2_in+#1
        staa    th3a+#1
        ldaa    sci2_in+#2
        staa    th3a+#2
        ldaa    sci2_in+#3
        staa    th3a+#3

* Convert th3a to th3s
        jsr     th3a2th3s
        ldd     th3s
        std     tmp16

* Increase height by .5" to 1" to restore tension.
        ldd     th3s            * Convert th3s to angle from ZZERO
        subd    #ZZERO
        std     tmp16
        ldd     tmp16           * If angle is negative, take absolute
        ble     negth3s         * value and set flag (stphgt) to '-'
        ldaa    #' '
        staa    stphgt
        bra     th3cont
negth3s ldaa    #'-'
        staa    stphgt
        ldd     #$0000
        subd    tmp16
        std     tmp16

th3cont nop                     * Divide angle by ZDPI, to determine
        ldx     #ZDPI           * ZZERO
        ldd     tmp16
        idiv
        pshb

*
        ldd     #0              * Calculate absolute value of angle
stp1    addd    #ZDPI           * that corresponds to number of whole
        std     tmp16           * inches
        dex
        bne     stp1

        pulb                    * Recall remainder (R)
        ldaa    #2              * If 2*R<ZDPI then
        mul                             If (angle below zero) then
        cpd     #ZDPI                           goto case3
        bge     anglge                  else
angl1t  nop                                     goto case2
        ldaa    stphgt                  Endif
        cmpa    #'-'              else
        beq     case3                   If (angle ò zero) then
        bra     case2                           goto case4
anglge  nop                             else
        ldaa    stphgt                          goto case1
        cmpa    #'-'                    Endif
        beq     case4             Endif
        bra     case1

case1   nop                     tmp16 = tmp16 + 1.5*ZDPI
        ldd     tmp16
        addd    #ZDPI
        std     tmp16
        ldd     #ZDPI
        ldx     #2
```

```
        idiv
        pshx
        pula
        pulb
        addd    tmp16
        std     tmp16
        bra     stp2
case2   nop                     tmp16 = tmp16 + 1.0*ZDPI
        ldd     tmp16
        addd    #ZDPI
        std     tmp16
        bra     stp2
case3   nop                     tmp16 = tmp16 - 0.5*ZDPI
        ldd     #ZDPI
        ldx     #2
        idiv
        pshx
        pula
        pulb
        subd    tmp16
        std     tmp16
        ldd     #0
        subd    tmp16
        std     tmp16
        bra     stp2
case4   nop                     tmp16 = tmp16
        bra     stp2

stp2    ldaa    stphgt          * Convert angle from Z coordinate system
        cmpa    #'-'            * to absolute motor angle by adding angle
        beq     stp3            * to ZZERO if positive and subtracting
        ldd     #ZZERO          * angle from ZZERO if negative.
        addd    tmp16
        std     th3
        bra     stp4
stp3    ldd     #ZZERO
        subd    tmp16
        std     th3
stp4    std     tmp16

        ldd     th3             * Add an additional inch to height
        addd    #ZDPI           * so that clay can be pinned instead
        std     th3             * of scooped.

        ldaa    #'1'            * set stop flag
        staa    fstop
        staa    fcmd3           * set new command3 flag
zslak   nop
        ldaa    #%00001000      * reset interrupt flag IC4
        staa    TFLG1,X
        rti

*************************************************************************
*
*  THE MAIN PROGRAM
*
*************************************************************************

START   lds     #stack
        jsr     init
        jsr     gohome

zmain   ldaa    datain
        cmpa    #'1'
        beq     mna
        ldaa    ffbk1
        cmpa    #'1'
        beq     mna
        ldaa    ffbk2
```

```
        cmpa    #'1'
        beq     mna
        ldaa    ffbk3
        cmpa    #'1'
        beq     mna
        bra     zmain
mna     bra     main1

main1   ldaa    datain
        cmpa    #'1'
        beq     main2
        jsr     doout           * unsolicited feedback received
        jmp     zmain
main2   nop
*        jsr     docmds
        jsr     findit
        jsr     putaway
        jsr     gohome
        jmp     zmain

        jmp     zmain
```

D. Motor Controller Program Listing

```
**********
**********
*  MTR9.ASM -- Must be linked with MTR1INC.ASM
**********
**********
*  PORTA pins:
*       0:  low bit in 2 bit CPU address
*               Read from DIP switch 1.
*       1:  hi bit in 2 bit CPU address
*               Read from DIP switch 1.
*       2:  Q input from 74LS74.  Driven hi
*               by index pulse.  Driven lo
*               by reset CLR (lo-hi on PB7)
*       3:  Step       (5210)
*       4:  Direction (5210)
*       5:  Write Input (LS7166)
*       6:  Read Input  (LS7166)
*       7:  Control/Data Input (LS7166)
*
*  PORTB pins:
*       7:  lo-hi clears 74LS74 (CLR1)
*
*       7/18/94 -- RMB
*       Added routine to seek index (home
*       position) and reset counter.
*       Reads PA2 and steps is not high.
*       When     high, toggles CLR on 74LS74
*       (PB7).
*
*       10/18/94 -- RMB (mtr2.asm)
*       Added SCI Interrupt Service Routine
*       to accept incoming command only if
*       fa or f1 is first character.
*
*       10/19/94 -- RMB (mtr3.asm)
*       Added small, medium, and large step option
*
*       11/15/94 -- RMB (mtr4.asm)
*       Repaired home seek on power up.  Routine disturbed by s/m/l option.
*
*       11/16/94 -- RMB (mtr5.asm)
*       Modified input command format to a one byte address
*       and four bytes representing
*       the ascii version of a hexadecimal number of steps.
*       During the SCI_ISR, the command is check to ensure it is valid.
*       A flag is set to show that a new command has been received.
*       During the main loop, this command is interpreted and sent to
*       the motor.  Each time a command is executed, a ASCII string is output
*       consisting of ð (master address, "f0"), six characters representing
*       the hex value of the count (512 counts / 360 degrees), and four
*       characters representing the issued command.
*
*       11/28/94 -- RMB (mtr6.asm)
*       Make input command and output count in degrees.  Appropriate
*       conversions must take place in code (400 steps/360 degrees and
*       360 degrees/512 counts).  Position output is limited to a two
*       byte hex number (represented by four ASCII characters).  Maximum
*       and minimum values of theta are stored as lim_min and lim_max.
*       Theta_d is forced to remain within this range.
*
*       11/29/94 -- RMB (mtr7.asm)
*       Controller can operate closed loop (using encoder feedback) or
*       open loop (using estimated theta).  On power-up, mode is closed
*       loop.  Mode (o or c) is echoed with theta and command.  Cause
*       (1:  nonchanging count, 2:  nonzero at startup, 3:  unreasonable
*       variation between actual and estimated count)
*
*       12/8/94 -- RMB (mtr8.asm)
*       Pause after two address output to allow master
```

```
*       hardware time to select port.
*
*       12/19/94 -- RMB (mtr9.asm)
*       Modified sci_isr to allow stop command (address,S,***).  Modified
*       STEPIT routine to check for new command before every step.  If new
*       command is received, routine resets.  If new command is STOP,
*       theta_d is set to theta.  Modified estimated theta (in STEPIT) to
*       allow for an interrupt during motion.

COMSIZE         equ $0004
R_DATA          equ $0100
CR              equ $0D
LF              equ $0A

        org  R_DATA
rdata           rmb 4
tdata           rmb 1
byte2           rmb 1
byte1           rmb 1
byte0           rmb 1
theta_d         rmb 2
dxflag          rmb 1
fstop           rmb 1
theta           rmb 2
delta_th        rmb 2
mode            rmb 1
modeflag        rmb 1
theta_e         rmb 2
theta_o         rmb 2
cnterr          rmb 2
temp            rmb 2
tmp16           rmb 2
steps           rmb 2
SHFTREG         rmb 2
TMP1            rmb 1
STACKAREA       rmb 30
STACKTOP        rmb 1

        ORG $b700
waitcnt         rmb 2
ADDRESS         rmb 1
lim_min         rmb 2
lim_max         rmb 2
MAXERR          rmb 2



        ORG $b600
        jmp $d000
        ORG $b604
        jmp SCI_ISR
        ORG waitcnt
        FDB $08FF       * M# mtr1: FFFF    mtr2: 08FF      mtr3: 08FF
        ORG ADDRESS
        FCB $F3         * M# mtr1: F1      mtr2: F2        mtr3: F3
        ORG lim_min
        FDB $F1F0       * M# mtr1: 0000    mtr2: 0000      mtr3: F1F0
        ORG lim_max
        FDB $0000       * M# mtr1: 00E1    mtr2: 0276      mtr3: 0000
        ORG MAXERR
        FDB $0010
        ORG $fffe
        FDB $b600
        ORG $ffd6
        FDB $b604

*  EPROM begins at $D000.
        ORG $D000
```

```
        jmp  STARTUP


INIT    ldx  #REGBAS
        jsr  INITA
        jsr  INITOP
        jsr  ONSCI
        jsr  GOHOME
        jsr  INITVAR
        cli
        rts

**********
*INITVAR -- Initialize variables
**********
INITVAR ldaa #0
        staa rdata
        staa rdata+1            * rdata:rdata+1 = 0
        ldaa #'0'
        staa dxflag
        staa fstop
        staa modeflag           * modeflag = '0'
        ldaa #'c'
        staa mode               * mode = 'c'
        ldd  #0
        std  theta_e            * theta_e:theta_e+1 = 0
        std  theta_o            * theta_o:theta_o+1 = 0
        rts

**********
*INITA -- MAKE PINS 3 & 7 OF PORTA OUTPUTS
**********
INITA   ldaa #%10001000
        staa PACTL,X
        rts

*  INITIALIZE THE OPTICAL ENCODER
*  COUNTER CHIP (LS7166)
INITOP  jsr  RSTCNTR
        jsr  SETQR
        jsr  SETICR
        rts

*  RESET COUNTER TO ZERO ON STARTUP
RSTCNTR psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa #%11111111
        staa DDRC,X
        ldaa #%00000100
        staa PORTC,X
        jsr  WRREG
        puly
        pulx
        pulb
        pula
        rts


*  TOGGLE WRITE BIT TO WRITE BYTE ON
*  PORTC TO LS7166.
WRREG   psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa PORTA,X
```

```
        ora  #%11100000
        staa PORTA,X
        anda #%11011111
        staa PORTA,X
        ora  #%11100000
        staa PORTA,X
        puly
        pulx
        pulb
        pula
        rts


*  SET QUADRATURE REGISTER FOR X1
*  OPERATION
SETQR   psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa #%11111111
        staa DDRC,X
        ldaa #%11111101
        staa PORTC,X
        jsr  WRREG
        puly
        pulx
        pulb
        pula
        rts


*  SET INPUT CONTROL RESIGTER
*  ENABLE INPUTS A & B
SETICR  psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa #%11111111
        staa DDRC,X
        ldaa #%01001000
        staa PORTC,X
        jsr  WRREG
        puly
        pulx
        pulb
        pula
        rts


*  turn SCI on.   9600 baud.
ONSCI   psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa #BAUD_9600
        staa BAUD,X
        ldaa #%00001000 * wakeup by address mark (MSB=1)
        staa SCCR1,X
        ldaa #%00101110
        staa SCCR2,X    * enable SCI transmit & receive (wake-up mode)
        puly
        pulx
        pulb
        pula
        rts
```

```
*  STEP MOTOR BACKWARD UNTIL OPTO INDEX
*  GOES HIGH.  FORWARD ONE STEP. RESET
*  COUNTER.  RESET 74LS74.
GOHOME  psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
*        * toggle CLR1 (PB7) on 74LS74 to set
*        * Q (PA2) low
        ldaa PORTB,X
        anda #%01111111
        staa PORTB,X
        ora  #%10000000
        staa PORTB,X
*        * check for high on PA2
*        * (implies that index has
*        *  pulsed).
GOHOME1 ldaa PORTA,X
        anda #%00000100
        bne  RESETEM
* backward one step since not yet at index
        jsr MFWD1       * M# mtr1: MREV1   mtr2: MREV1     mtr3: MFWD1
* pause before next step necessary so that motor has time to respond
        jsr SLODOWN
        bra GOHOME1

RESETEM nop
* Clear 74ls74 (set Q low) step FWD (REV) one step so that zero cound is
* first step with low Q.
        jsr MREV1       * M# mtr1: MFWD1   mtr2: MFWD1     mtr3: MREV1
        jsr SLODOWN
* reset 74LS74 (Q is lo)
        ldaa PORTB,X
        anda #%01111111
        staa PORTB,X
        ora  #%10000000
        staa PORTB,X
        jsr  RSTCNTR            * reset counter to 0 (LS7166)

        jsr  RDCNTR             * read counter
        ldd  byte1              * if ( -1 < byte1:byte0 < 1 )
        cpd  #$0001             *       goto CLOSED2
        bgt  OPEN2              * else
        cpd  #$FFFF             *       goto OPEN2
        blt  OPEN2              * endif
        bra  CLOSED2

OPEN2   ldaa #'o'
        staa mode               * mode = 'o'
        ldaa #'2'
        staa modeflag           * modeflag = '2'
        bra  RESETZ             * goto RESETZ

CLOSED2 ldaa #'c'
        staa mode               * mode = 'c'
        ldaa #'0'
        staa modeflag           * modeflag = '0'
        bra  RESETZ             * goto RESETZ

RESETZ  puly
        pulx
        pulb
        pula
        rts

* loop counts down from waitcnt
* to kill time between step
* commands
```

```
SLODOWN psha
        pshb
        ldad    waitcnt
SLO1     subd    #$0001
        bne     SLO1
        pulb
        pula
        rts


*  Send a char out of SCI.
OUTSCI  psha
        pshb
        pshx
        pshy
        ldx  #REGBAS

OUTSCI1 ldaa SCSR,X
        bita #$80
        beq  OUTSCI1     * loop if not
*                        * ready/ still
*                        * xmitting.
        ldaa tdata
        staa SCDR,X        * send char
        puly
        pulx
        pulb
        pula
OUTSCIX rts


*  SET UP THE LS7166 TO READ THE
*  COUNTER REGISTER ON
*  THE LS7166.
RDCNTR  psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa #%11111111
        staa DDRC,X
        ldaa #%00000011
        staa PORTC,X
        jsr  WRREG
        ldaa #%00000000
        staa DDRC,X
        jsr  RDDATA
        puly
        pulx
        pulb
        pula
        rts


*  READ THE THREE BYTE COUNTER
*  REGISTER ON THE LS7166
RDDATA  psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        ldaa PORTA,X
        anda #%01111111
        ora  #%01100000
        staa PORTA,X
        anda #%00111111
        staa PORTA,X
        ldab PORTC,X
```

```
        stab byte0
        ora  #%01100000
        staa PORTA,X
        anda #%00111111
        staa PORTA,X
        ldab PORTC,X
        stab byte1
        ora  #%01100000
        staa PORTA,X
        anda #%00111111
        staa PORTA,X
        ldab PORTC,X
        stab byte2
        ora  #%01100000
        staa PORTA,X
        puly
        pulx
        pulb
        pula
        rts


**********
* PRCNT -- Convert the six byte counter value to a two byte value (degrees)
* and store at theta:theta+1
**********
PRCNT   psha
        pshb
        pshx
        pshy
        ldx  #REGBAS

        ldaa byte1
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
        jsr  OUTSCI

        ldaa byte0
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
        jsr  OUTSCI

PRCNTX  puly
        pulx
        pulb
        pula
        rts


**********
* PRTHETA -- Convert the six byte counter value to a two byte value (degrees)
* and store at theta:theta+1
**********
PRTHETA psha
        pshb
        pshx
        pshy
        ldx  #REGBAS
        jsr  DIV2DEG

        ldaa theta
```

```
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
        jsr  OUTSCI

        ldaa theta+1
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
        jsr  OUTSCI

PRTHETX puly
        pulx
        pulb
        pula
        rts


**********
* PRMODE -- Print mode and modeflag
**********
PRMODE  psha
        pshb
        pshx
        pshy

        ldaa    mode
        staa    tdata
        jsr     OUTSCI
        ldaa    modeflag
        staa    tdata
        jsr     OUTSCI

        puly
        pulx
        pulb
        pula
        rts

**********
* PREST -- Print estimated theta
**********
PREST   psha
        pshb
        pshx
        pshy

        ldaa theta_e
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
        jsr  OUTSCI

        ldaa theta_e+1
        jsr  TOASCII
        pshb
        staa tdata
        jsr  OUTSCI
        pula
        staa tdata
```

```
        jsr  OUTSCI


        puly
        pulx
        pulb
        pula
        rts

**********
*       8-bit binary in A -> 2 ascii digits in A:B
**********
TOASCII tab
        rora
        rora
        rora
        rora
        anda #$0F
        adda #$30
        cmpa #$39
        ble  TASC1
        adda #7
TASC1   andb #$0F
        addb #$30
        cmpb #$39
        ble  TASCX
        addb #7
TASCX   rts


*  WRITE A SPACE TO THE SCI
WRSPACE psha
        pshb
        ldaa #$20
        staa tdata
        jsr  OUTSCI
        pulb
        pula
        rts

**********
*  STEPIT -- Calculate and send motor command.
**********
STEPIT  NOP
*  Save old accumulator values
        psha
        pshb
        pshx
        pshy
*  Reset steps
        ldd    #$0000
        std    steps

* Calculate new estimate for theta using theta_e and steps (number of
* steps since last update.
STEPIT1 nop
        ldd     steps
        cpd     #$0000
        blt     netback
        bra     netfwd

netback ldd     #$0000
        subd    steps
        jsr     stp2deg
        subd    theta_e
        std     theta_e
        ldd     #$0000
        subd    theta_e
        std     theta_e
```

```
        bra     newest

netfwd  ldd     steps
        jsr     stp2deg
        addd    theta_e
        std     theta_e
        bra     newest

newest  ldd     #$0000
        std     steps


* Determine mode of operation
        jsr     RDCNTR          * read counter value into byte2,byte1,byte0
        jsr     DIV2DEG         * convert counter value to degrees (theta)

        ldaa    mode            * if open loop mode goto CONT1
        cmpa    #'o'
        beq     CONT1
        ldd     theta           * cnterr = theta - theta_e
        subd    theta_e
        std     cnterr
        cpd     #$0000          * cnterr = abs(cnterr)
        bge     MODECHK
        ldd     #$0000
        subd    cnterr
MODECHK cpd     MAXERR          * if cnterr>MAXERR goto OPEN3
        bgt     OPEN3
        bra     CLOSED3         * jump to CLOSED3

OPEN3   ldaa    #'o'
        staa    mode
        ldaa    #'3'
        staa    modeflag
        bra     CONT1

CLOSED3 ldaa    #'c'
        staa    mode
        ldaa    #'0'
        staa    modeflag
        bra     CONT1

* Determine number and direction of steps
CONT1   ldaa    fstop           * if fstop = '1'
        cmpa    #'1'            *       fstop = '0'
        bne     CONT2           *       if rdata = 'S'
        ldaa    #'0'            *               if mode=closed
        staa    fstop           *                       theta_d = theta
        ldaa    rdata           *               else
        cmpa    #'S'            *                       theta_d = theta_e
        bne     CONT2           *               endif
        ldaa    mode
        cmpa    #'o'
        beq     olstop
        ldd     theta           *       endif
        bra     storit
olstop  ldd     theta_e
storit  std     theta_d         * endif

* Calculate delta theta
CONT2   nop

        ldaa    mode
        cmpa    #'c'
        beq     CLOSED          * if mode = closed then
        ldd     theta_d         *       delta_th = theta_d - theta
        subd    theta_e         * else
        std     delta_th        *       delta_th = theta_d - theta_e
        bra     MOVEON          * endif
```

```
CLOSED  ldd     theta_d
        subd    theta
        std     delta_th

* Use delta theta to determine motor direction
MOVEON  pshb
        psha
        puly                    * Y = delta_th

        cpy  #$FFFF
        blt  JGOREV              * if delta_th < -1 GOREV
        cpy  #$0001
        bgt  GOFWD              * if delta_th >  1 GOFWD
        jmp  STEPITX            * branch to STEPITX (Good enough!)
JGOREV  jmp  GOREV

* Move motor forward if necessary
GOFWD   ldd  theta
        std  theta_o
        cpy  #$00B4
        ble  GOFWD1
        ldy  #$00B4
GOFWD1  nop
        pshy
        pula
        pulb                    * B = LSB of Y (MSB = $00)
        ldaa #$0A               * A = $0A (10)
        mul                     * D = B*$0A
        ldx  #$09               * X = $09 (9)
        idiv                    * X = LSB*10/9
        pshx
        puly                    * Y = X (steps)
GOFWD2  ldaa    #'1'
        cmpa    fstop
        bne     fwda
        jmp     STEPIT1
fwda    ldd     steps
        addd    #$01
        std     steps
        jsr  MFWD1              * Take one step forward
        jsr  SLODOWN            * Pause between steps
        dey                     * Y = Y - 1
        cpy  #$0000
        bne  GOFWD2             * If Y <> 0 jump to GOFWD2

        ldaa    mode            * if open loop mode goto CONT3
        cmpa    #'o'
        beq     CONT3
        ldd  theta_o
        jsr     RDCNTR          * read counter value into byte2,byte1,byte0
        jsr     DIV2DEG         * convert counter value to degrees (theta)
        cpd  theta
        beq  FOPEN1
        jmp  STEPIT1            * Jump to STEPIT1
FOPEN1  ldaa #'o'
        staa mode
        ldaa #'1'
        staa modeflag
CONT3   jmp  STEPIT1

* Move motor backward if necessary
GOREV   ldd  theta
        std  theta_o
        ldd #$0000              * D = 0
        subd delta_th           * D = D - delta_th
        pshb
        psha
        puly                    * Y = D (Y = -delta_th)
        cpy  #$00B4
```

```
        ble   GOREV1
        ldy   #$00B4
GOREV1   nop
        pshy
        pula
        pulb                    * B = LSB of Y (MSB = $00)
        ldaa #$0A               * A = $0A (10)
        mul                     * D = B*$0A
        ldx   #$09              * X = $09 (9)
        idiv                    * X = LSB*10/9
        pshx
        puly                    * Y = X (steps)
GOREV2   nop
        ldaa    #'1'
        cmpa    fstop
        bne     reva
        jmp     STEPIT1
reva     ldd     steps
        subd    #$01
        std     steps
        jsr   MREV1             * Take one step backward
        jsr   SLODOWN           * Pause between steps
        dey                     * Y = Y - 1
        cpy   #$0000
        bne   GOREV2            * If Y <> 0 jump to GOREV2

        ldaa    mode            * if open loop mode goto CONT4
        cmpa    #'o'
        beq     CONT4
        ldd   theta_o
        jsr     RDCNTR          * read counter value into byte2,byte1,byte0
        jsr     DIV2DEG         * convert counter value to degrees (theta)
        cpd   theta
        beq   ROPEN1
        jmp   STEPIT1           * Jump to STEPIT1
ROPEN1   ldaa #'o'
        staa mode
        ldaa #'1'
        staa modeflag
CONT4    jmp   STEPIT1           * Jump to STEPIT1

*       * Recall old accumulator values
STEPITX puly
        pulx
        pulb
        pula
        rts


*  Issue command for one step backward
MREV1    psha
        pshb
        pshx
        pshy
        ldx   #REGBAS
        ldab PORTA,X
        andb #%11100111
        stab PORTA,X
        orab #%00001000
        stab PORTA,X
        puly
        pulx
        pulb
        pula
        rts

*  Issue command for one step forward
MFWD1    psha
        pshb
```

```
        pshx
        pshy
        ldx  #REGBAS
        ldab PORTA,X
        orab #%00010000
        andb #%11110111
        stab PORTA,X
        orab #%00001000
        stab PORTA,X
        puly
        pulx
        pulb
        pula
        rts

*****************
*   HEXBIN(a) - Convert the ASCII character in a
* to binary and shift into shftreg.  Returns value
* in TMP1 incremented if a is not hex.
*****************
HEXBIN  PSHA
        PSHB
        PSHX
        pshy
        JSR  UPCASE     convert to upper case
        CMPA #'0'
        BLT  HEXNOT     jump if a < $30
        CMPA #'9'
        BLE  HEXNMB     jump if 0-9
        CMPA #'A'
        BLT  HEXNOT     jump if $39> a <$41
        CMPA #'F'
        BGT  HEXNOT     jump if a > $46
        ADDA #$9        convert $A-$F
HEXNMB  ANDA #$0F       convert to binary
        LDX  #SHFTREG
        LDAB #4
HEXSHFT ASL  1,X        2 byte shift through
        ROL  0,X              carry bit
        DECB
        BGT  HEXSHFT    shift 4 times
        ORAA 1,X
        STAA 1,X
        BRA  HEXRTS
HEXNOT  INC  TMP1       indicate not hex
HEXRTS  puly
        PULX
        PULB
        PULA
        RTS

*****************
*  UPCASE(a) - If the contents of A is alpha,
* returns a converted to uppercase.
*****************
UPCASE   CMPA #'a'
         BLT  UPCASE1      jump if < a
         CMPA #'z'
         BGT  UPCASE1      jump if > z
         SUBA #$20         convert
UPCASE1  RTS

**********************
* DIV2DEG -- Converts Count (in divisions) to theta (in degrees)
**********************
DIV2DEG psha
        pshb
        pshx
        pshy                    * Push Y onto stack
```

```
        ldaa    TMP1
        psha

        ldaa    #$00
        staa    TMP1            * TMP1 = 0
        ldy     #$0000          * Y = 0
        ldd     byte1           * D = COUNT (in divisions) (512 div/rev)
        cpd     #$0000
        bge     NOCHNG          * If byte >= 0, jump to NOCHNG
        ldd     #$0000
        subd    byte1           * D = -byte1:byte0
        psha
        ldaa    #1
        staa    TMP1            * TMP1 = 1
        pula
NOCHNG  nop

* Determine how many times D is divisible by $100.  $100 div = $B4 deg.
CNT1    cpd     #$0100
        ble     CNT2            * Jump to CNT2 if D <= $0100
        subd    #$0100          * D = D - $100
        iny                     * Y = Y + 1
        bra     CNT1            * Jump always to CNT1

CNT2    ldaa    #$2D            * A = $2D (45)
        mul                     * D = A * B  (B is LSB of COUNT, MSB = $00)
        ldx     #$0040          * IX = $40 (64)
        idiv                    * IX = D/IX
        pshx                    * Push X onto Stack
        pula                    * Pull A off stack (A = MSB of D)
        pulb                    * Pull B off stack (B = LSB of D)
        std     theta           * theta = IX (second term of theta)
        pshy                    * Push Y onto stack
        pula                    * Pull A off stack (A = MSB of IY)
        pulb                    * Pull B off stack (B = LSB of IY)
        ldaa    #$B4            * A = $B4 (180)
        mul                     * D = A * B
        addd    theta           * D = D + theta
        std     theta           * theta = D
        ldaa    TMP1            * Look at TMP1
        cmpa    #1              *
        bne     NOCHNG2         * If TMP1 is not equal to 1, jump to NOCHNG2
        ldd     #$0000          *
        subd    theta           *
        std     theta           * theta = -theta
NOCHNG2 nop

        pula
        staa    TMP1
        puly                    * Pull Y off stack
        pulx                    * Pull X off stack
        pulb                    * Pull B off stack
        pula                    * Pull A off stack
        rts

**********************
* stp2deg -- Converts steps in D (400/rev) to degrees (360/rev) in D.
**********************
stp2deg pshx                    * Push X onto stack
        pshy                    * Push Y onto stack

        std     tmp16

s2d1    ldd     #$0000
        ldy     #9
mulit   addd    tmp16
        dey
        bne     mulit
        std     tmp16
```

```
        ldx     #10
        idiv
        stx     tmp16
        pshx
        pula
        pulb
        std     tmp16

s2dz    puly                    * Pull Y off stack
        pulx                    * Pull X off stack
        rts

**********
* SCI Interupt Service Routine
* If first byte is fa or ADDRESS,
* dxflag is set to $31 and
* remaining bytes are stored in rdata
**********
SCI_ISR ldx     #REGBAS
        ldaa    SCSR,X                  * necessary to reset flag
        ldaa    SCDR,X                  * read first byte
        cmpa    #$fa
        beq     isr2                    * if byte1 .eq. $fa then goto isr2
        cmpa    ADDRESS
        beq     isr2                    * if byte1 .ne. ADDRESS goto isr4
        jmp     isr4

isr2    ldy     #rdata
isr3    brclr   SCSR,X #%00100000 isr3    * stop here until byte is received
        ldaa    SCDR,X                  * read & save byte
        staa    0,Y
        iny                             * increment pointer
        cpy     #COMSIZE+#R_DATA          * compare pointer to max pointer
        blo     isr3                    * if all bytes not read goto isr3

* if rdata:rdata+3 = 'S***' then
*       fstop = '1'
*       rti
* end
        ldaa    rdata               * A = rdata
        cmpa    #'S'
        bne     nostop
        ldaa    #'1'
        staa    fstop                   * fstop = '1' (valid data)
        jmp     isr4                    * rti
nostop  nop


* if all rdata byte 0-9 or A-F then
*       set flag
* else
*       rdata = 0000
* end
* return
        ldaa #$00
        staa TMP1
        ldy  #rdata
        ldaa 0,Y
        jsr  HEXBIN
        ldaa TMP1
        bne  isr4
        ldaa 1,Y
        jsr  HEXBIN
        ldaa SHFTREG+1
        staa theta_d
        ldaa 2,Y
        jsr  HEXBIN
        ldaa TMP1
        bne  isr4
```

```
        ldaa 3,Y
        jsr  HEXBIN
        ldaa SHFTREG+1
        staa theta_d+1
        ldaa    #$31                    * set good data received flag
        staa    dxflag

        ldd     theta_d                 * if theta_d < lim_min then
        cpd     lim_min                 *       theta_d = lim_min
        bge     LIM1                    * endif
        ldd     lim_min
        std     theta_d
LIM1    cpd     lim_max                 * if theta_d > lim_max then
        ble     LIM2                    *       theta_d = lim_max
        ldd     lim_max                 * endif
        std     theta_d
LIM2    nop

isr4    nop
        bset    SCCR2,X #%00000010      * RWU == 1 (MCU in wakeup mode)
zs_isr  rti                             * return from interupt


**********
* STARTUP -- MAIN
**********
STARTUP lds  #STACKTOP
        jsr  INIT
        ldaa #CR        * Does not affect master.  Looks better on screen.
        staa tdata
        jsr  OUTSCI

LOOP0   jsr  RDCNTR
        ldaa #$f0       * master address
        staa tdata
        jsr  OUTSCI
        ldaa #$f0       * master address
        staa tdata
        jsr  OUTSCI

        ldy     #$FF  * pause to allow master to select port
pause1  dey
        bne     pause1

        ldaa mode
        cmpa #'c'
        beq  CLLOOP
        jsr  PREST
        bra  GOLOOP
CLLOOP  jsr  PRTHETA
GOLOOP  jsr  PRMODE

*  Write command to SCI.
*  Check for received command
LOOP1   ldaa dxflag
        cmpa #$31
        bne  LOOP1
        jsr  STEPIT
        ldaa #$00
        staa dxflag
ZLOOP0  bra  LOOP0
```

E. Using Pcbug11 to Program the Motorola 68HC11E9

S68HC11EVBU Customer:

Both the standard and student EVBUs come with an MC68HC11E9FN1 MCU installed on the board and the BUFFALO monitor program stored in the MCU-internal ROM. But the student EVBU kit also contains a blank MC68HC711E9 MCU and PCbug11, a PC-based monitor program. A complete description of the PCbug11 is provided in the PCbug11 User's Manual, M68PCBUG11/D1. While a detailed description of the BUFFALO monitor is available in the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

You may install the blank MC68HC711E9 MCU in the EVBU socket at location U3. After installing the 711E9 MCU on the EVBU it may be programmed using PCbug11. Either BUFFALO or a user-developed program can be stored in the 711E9 MCU-internal EPROM. Step-by-step EPROM programming instructions are provided in this letter or refer to page 4-12 of the PCbug11 user's manual for additional information.

WHAT IS PCbug11?

PCbug11 is a software package for easy access to and simple experimentation with M68HC11 microcontroller unit (MCU) devices. PCbug11 lets you program any member of the M68HC11 MCU family and examine the behavior of internal peripherals under specific conditions. In addition, you may run your own programs on the MCU; breakpoint processing and trace processing are available.

To configure the EVBU to use PCbug11:

1. Remove the jumper from J7, and place it across J3. Moving the jumper to J3 grounds the MODB pin and at reset places the HC11 in BOOTSTRAP mode.

NOTE

Refer to Figure 2-1 of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1, for jumper header and connector locations.

2. Connect the EVBU to your PC serial port via a user-supplied 25-pin cable. The PC serial port can be either COM1 or COM2. The cable must be a Hayes-compatible modem cable and is available at most electronic supply stores.

3. Apply power to the EVBU.

4. To start PCbug11 from the command line:

   PCbug11 -E port=1<CR> Where I/O is COM2, use port=2.
   <CR> is the symbol for carriage return.

5. The registers should be displayed on the screen, and a >> prompt in the window at the bottom of the screen.

6. With PCbug11 version 3.24A, enter on the PC keyboard:

   CONTROL BASE HEX<CR>

   This defines the keyboard input default as hexadecimal. By doing this, you do not have to add the $ to inputs.

This should get you started with PCbug11. Because the TALKER code used in this example resides in RAM, you are limited to the amount of free space that you can use for variables. It may be useful to put the TALKER into EPROM (it takes about 200 bytes), and leave most of your user space free. For more detail on the TALKER refer to paragraph 4.4 of the PCbug11 manual.

PROGRAMMING EEPROM

Files to be programmed into the 711E9 MCU-internal EEPROM must be in S-record format. The S-record format is explained in Appendix A of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

NOTE

The S-record to be downloaded into the 711E9 MCU-internal EEPROM must be ORGed at address $B600.

Enter on the PC keyboard:

EEPROM $B600 $B7FF<CR>	This lets PCbug11 know that these addresses are EEPROM and that it should use a EEPROM algorithm.
MS $1035 00<CR>	This clears the block protect register (BPROT) and lets you program the EEPROM section of the 711E9 MCU.
LOADS filename<CR>	This loads an S-record format file into the EEPROM section of the 711E9 MCU.
VERF filename<CR>	This verifies that the S-record format file was successfully loaded into EEPROM.

PROGRAMMING EPROM

Files to be programmed into the 711E9 MCU-internal EPROM must be in S-record format. The S-record format is explained in Appendix A of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

NOTE

The S-record to be downloaded into the 711E9 MCU-internal EPROM must be ORGed at address $D000.

To program the MCU-internal EPROM enter on the PC keyboard:

EPROM $D000 $FFFF<CR> — This lets PCbug11 know that these addresses are EPROM and that it should use a EPROM algorithm.

Apply +12Vdc to the $\overline{\text{XIRQ}}$ pin — To program the MCU-internal EPROM, +12Vdc must be applied to the $\overline{\text{XIRQ}}$ pin of the 711E9 MCU. Attach a +12Vdc power supply to the MCU I/O port connector P4, pin-18. A 100Ω resister must be installed in series with the +12Vdc power supply and P4, pin-18.

CAUTION

Do not apply a +12Vdc programming voltage power source when the main V_{DD} (+5Vdc) power is off; doing so will damage the EVBU intrgrated circuits. Always turn on the main V_{DD} (+5Vdc) power before the +12Vdc programming voltage is applied.

LOADS filename<CR> — This loads an S-record format file into the EPROM section of the 711E9 MCU.

VERF filename<CR> — This verifies that the S-record format file was successfully loaded into EPROM.

PCbug11 HINTS

- The EPROM and EEPROM commands must be entered before you can program EPROM and EEPROM. This sets-up PCbug11 EPROM and EEPROM programming routines.

- Don't forget to clear the BPROT register before trying to modify EEPROM locations.

- Initially, you should work on your routines in EEPROM. Since you can trace through EEPROM like RAM, it is best to try them out there before committing to EPROM. When tracing the EEPROM use the memory set (MS) command to modify the block protect register to 00 (MS $1035 00) and the EEPROM command (EEPROM $B600 $B7FF).

- Be sure to set your stack pointer where it will not interfere with the PCbug11 stack pointer. The TALKER program starts at $0000 in RAM, with the first free byte at $0100. The PCbug11 stack pointer is set to $01FF. Set your stack pointer at least 20 bytes ($01EB) lower than this.

- If a COM fault occurs while entering commands on the command line:

  1. Check the cable between the EVBU and the PC. If the connection is okay, try issuing the control timeout command (CONTROL TIMEOUT 10000). This gives the MCU more time to respond (needed when PCbug11 is running on a fast PC).

  2. Make sure the transmit pin for the PC connects to the receive pin of the MCU. The transmit pin will have approximately 9 to 12 volts on it. The receive pin will only have a few millivolts if any.

  3. Remove the MCU from the socket and check the pins for damage. if the pins are shorted, straighten the pins and carefully reinsert the MCU.

- If you are using an XT-PC and the display locks-up, try issuing a MODE CO80 at the DOS prompt.