To the Graduate Council:

I am submitting herewith a thesis written by Martin B. Holder entitled "Design and Implementation of an Integrated Multi-processor Mobile Robot for Experimental Research in Cooperative Robotics." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mohan M. Trivedi

Dr. Mohan Trivedi, Major ~~Professor~~

We have read this thesis
and recommend its acceptance:

Robert E. Bodenheimer

Donald W. Bouldin

Rangasami

Accepted for the Council:

Carolyn Winkel

Associate Vice Chancellor
and Dean of The Graduate School

DESIGN AND IMPLEMENTATION OF AN INTEGRATED MULTI-PROCESSOR MOBILE ROBOT FOR EXPERIMENTAL RESEARCH IN COOPERATIVE ROBOTICS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Martin B. Holder

December 1994

Dedicated to my

loving parents

Acknowledgments

I would like to thank several people for their help and input into this thesis. I can not name everyone but would like to mention a few of those with key input. First and foremost, I especially would like to thank my committee members, Dr. Mohan Trivedi, Dr. Suresh Marapane, Dr. Robert Bodenheimer, and Dr. Don Bouldin for their inputs and patience in the preparation of this thesis. I would also like to thank the following at the Computer Vision and Robotics Research Laboratory at the University of Tennessee: Scott Thayer for his constructive critisism. Chris Gourley for aiding my fight in producing this thesis with Latex. Doug Shamblin for his input on numerous hardware issues. Mark Eklund for his S19 to HEX code converter program and help with some of the assembly language debugging for the RF subsystem. Arun Dalmia for his program used in placing the Latex postscript figures without having to guess at their locations. Fred DePiero for being available for technical help at the lab everyday, all day. Nils Lassiter in keeping ELVIS operational and showing me the necessary procedures used in purchasing parts through the university. Dr. Gopalan Ravichandrin for his help regarding C programming and Latex. For others outside of the lab: I would like to thank Mark Allen for taking the time to proofread the thesis and adding detailed comments and suggestions concerning the necessary fixes. I owe special thanks to Tom Collins for his time and effort in designing and building the mechanical system of the robot and side-stepping the problems which arose from having a very limited amount of machining tools to work with.

Abstract

A distributed multiprocessor mobile robot was designed and implemented to perform cooperative robotic tasks. SMAR-T is a 35 pound 18 inch Circular robot able to reach motor torques of 250 oz.in. with speeds of 1 ft/sec. SMAR-T is capable of inter-communicating with additional robots over wireless modems.

The robot's control was comprised of two levels: three dedicated low level subsystems (Motor Control, Ultrasonic, and RF Serial) and a 486 laptop PC for higher level control. The two levels communicate via serial interface. The lower level are designed with the 68HC11 8-bit micro-controllers. The high level control consists of C routines providing a multitude of functions. A serial multiplexer board was designed for intra-communications between the low and high level control. The three subsystems provide the ability to sense the robot's environment, control two stepper motors, and communicate with up to 5 additional robots over wireless modem. The robot utilizes an ultrasonic transducer mounted on a stepper motor to obtain radar-type range information. Two stepper motors are used as the mobility source. Speed and acceleration rates are selectable by the host system with three modes of motor operation available. A protocol was developed for inter-communication between the various robots. Algorithms for wall-following and a cooperative task regarding a rescue mission are realized.

TABLE OF CONTENTS

CHAPTER **PAGE**

CHAPTER PAGE

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

FIGURE **PAGE**

FIGURE **PAGE**

CHAPTER 1

Introduction

Mobile robots have become a very interesting topic for many people. The ability to mimic human perception and ultimately intelligence is no trivial matter. Numerous approaches have been made in the area of artificial intelligence in hopes of providing intelligent decisions to be made on given sensory information. Having different types of sensory information present for processing, provides for a better understanding of the robots surroundings (more information), and thus gives way to a more intelligent means for accomplishing some given task. As the different types of sensory information increase, so too the programming complexity and processing power necessary to make vital decisions in a reasonable amount of time. As technology becomes more advanced and various research issues are resolved, robots will become more intelligent. Currently, it is not an easy task to have a mobile robot provide flawless results. Certain conditions always seem to exist in dynamic environments, resulting in erroneous decissions by the robot, whether due to improper sensory information, missed information, or bad judgement.

The term *cooperative robotics* is fundamentally defined as having two or more robotic systems operating in an intelligent manner to provide an effective, and efficient means of working towards some common goal. There usually exists a group of such robots with a leader providing the necessary intelligence for making important decisions regarding the actions the different robots within the group should take. It is pertinent to have some

means of communicating within the existing group to coordinate the necessary actions each robot must undergo to accomplish some collective goal. This usually assumes each robot is capable of simple navigation and contributes some useful and possibly unique skill to the group.

Many different approaches have been taken to provide cooperative efforts between robots and attempt to resolve the issues regarding intelligence. Some researchers believe using a multitude of small, simplistic robots comprised of the same architecture would prove more effective than say two heavy, complex robots. Depending on the task, a balance between the two extremes should be given thought. Two examples regarding searching and mapping will be further examined to show the importance of such a balance depending on the task.

For search related tasks, such as finding toxic gases within a building, it might prove more beneficial to use more than two robots, each utilizing a simplistic architecture. Each robot would be capable of roaming around and periodically taking samples from their sensors. Having a large number of such robots provides the ability to cover more area in less time. Another important issue in using many robots regards malfunctions. Should a robot malfunction, other robot(s) exist, such that the mission does not have to be terminated. Beacons can be placed throughout the building to provide rough estimations of the robots current location. As the number of robots increase, the ability to effectively communicate becomes overwhelming. To aid in such a problem, the robots typically don't communicate unless they have a very important discovery, such as finding the toxic gas in this example. Upon discovering the gas, other robots could be called upon to come to the general location and aid in providing a more detailed, in depth search.

Map making is another issue in which it could be very beneficial to use more than one single robot. However, these robots must have a more intelligent architecture, such that accurate mapping and navigation can be accomplished. Having robots of different sizes may prove beneficial in this task. Some areas may not be accessible by the larger robots and the smaller robots would have to be called upon to perform the mapping. As the robots begin mapping the environment, they each communicate to the others to update their maps, assuming each robot has its own local map it is generating.

The above two scenarios merely reflect the usefulness of cooperative robotics. There are a number of situations which could be given for validating the need of cooperative robotics. Nonetheless, it takes more than one such robot system to investigate the necessary issues regarding cooperative robotics. The CVRR Laboratory currently has one mobile robot (ELVIS), a VME-based robot which is more than capable of performing cooperative tasks. This thesis will focus on an effort directed towards the development of an integrated mobile robotic system which is well suited for experimental research in cooperative robotics. The following vital necessities of the development of such a system will be considered:

1. Architecture
2. Hardware
3. Motion Control System (Mobility)
4. Sensing System
5. Communication System
6. Software / Programming

The hardware architecture of this robot was guided by the available funds and resources. The expense of the robot included the mechanical assembly and customized electrical design, and was to cost no more than $1000. A laptop 486 personal computer, two 68HC11EVB microcontrollers, and two wireless modems were currently available for use on the robot.

Stepper motors were used to provide a means of locamotion for the robot. The majority of mobile robots typically use DC motors, however, the costs associated with DC motors was estimated to be $300 more. Stepper motors, providing no loss of phase occurs, can be accurately controlled at a minimal cost by simply switching the motor coils through a necessary sequence. The positional error with regard to stepper motors is not accumulative resulting in accurate positional control. When compared to DC motors for drive capability, they do have shortcomings. For very slow motor movements, the current through the coils reaches its maximum rating, thus results in large power consumption. DC motors dont exhibit this effect and are very efficient as compared to steppers. Since a sequence of switching must occur to advance the motor, current must build up based on the associated inductance and resistance of the motor coils each time the switching occurs. To obtain fast speeds from the motors, the switching must occur at faster rates. A minimum time period exists for the current within the coils to reach its rated value, and if not given adequate time to reach such a level, as in the case of faster switching, there will be a reduction in the available torque. The reduction, if significant, can prevent the rotor from keeping up with the proper sequence and cause the motors to loose synchronization. Thus a maximum speed exists for a given amount of torque, over which the motors can not be guaranteed to operate in a proper fashion. Since DC motors don't have a sequence of steps through which the current must change

polarities, they can be applied some maximum voltage and reach faster rotational speeds as a result. For accurate speed control, DC motors must have an additional means of detecting positional information such as encoders. The control for DC motors utilizes the encoder information as feedback to adjust the voltage being applied to the motor. The additional encoders and added control for the DC motors acount for an approximate $300 increase in cost for a two motor system as opposed to stepper motors.

Many sensors exist to gather information from the robots surroundings. For mobile robots, the sensory information is usually more specific with regards to the robots ability to see various obstacles in its path. A variety of sensors are available to provide this capability. The different sensing capabilities range from a simplistic infrared emmitter-detector pair to a very complex vision system. The primary limitation of an infrared sensor is its sensitivity to varying colors. Objects of darker colors, tend to absorb the light, causing the robot to not see the object. Objects of brighter colors, better reflect the light. For these reasons, infrared sensors were not chosen. Vision systems offer the most information yet require very expensive, dedicated hardware. Vision was discussed and further decided not to be used on this robot for cost reasons. Ultrasonic transducers are capable of providing very accurate range information. A system was previously developed, by myself, which incorporated a single ultrasonic transducer mounted to a small stepper motor. The transducer could be rotated in a sweeping fashion, to fire the transducer at different angles of interest. Most robots utilize a series of transducers located around the exterior of the robot in particular angular positions. This can effect the ability of the sensors to see objects typically close to the robot which resides inbetween the robots ultrasonic sensors. With the ability to scan in very precise angular movements, such obstacles can be detected. However, additional time is required to take

more readings. Reflections and the ability for the object to reflect the sound source affect the ultrasonic's accuracy. Additional DSP processing with multiple receivers exists to correct for such inaccuracies if desired.

All cooperative systems typically need a means of communication. Pulsed infrared emmitter-detector pairs and wireless modems are options to provide such capabilities. The infrared is useful for relatively close transmissions and the wireless modem for longer distances. The ability to relay information between the various robots allows for possible longer range, if desired.

Once the robot's hardware has been completed, programming provides the intelligence for the robot. Careful consideration of the robots surrounding must be examined and decissions need to be made for the robot to react upon. The ability for the robot to follow walls, provide sonar-based graphical mappings, and cooperate with another robot to attempt a rescue mission was realized.

CHAPTER 2

System Overview

SMAR-T (Small Mobile Autonomous Robotic - Testbed) is a distributed multiprocessor mobile robot. The base is of a circular footprint with an 18 inch diamter. The robot weighs approximately 35 pounds standing approximately 1 foot in height. A two wheel differential stepper motor drive system allows the robot to perform pivotal movements. The drive wheels are driven with a 2:1 gear ratio to account for a total 250 oz.in. of torque delivered to the wheel from each motor. SMAR-T's drive system is capable of attaining speeds up to 1 ft/sec. Figure 2.1 shows the mobile robot SMAR-T.

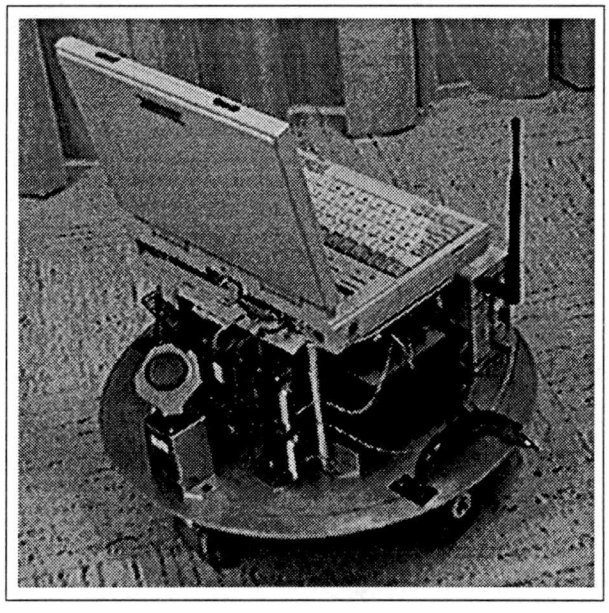

Figure 2.1: *SMAR-T*

The control system of SMAR-T is comprised of two layers. The high level control

is considered to be derived from the host system, a 486 laptop PC, from only a programming perspective utilizing the C language. Lower level, more specific control will come from the dedicated subsystems which are programmed in Motorola assembly and include hardware dependent on the dedicated task. A means of intra-communication was developed between the high level host and lower level subsystems. The completed system is shown in Figure 2.2.

The PC host system interracts with the robot's various subsystems through RS232 communications. Each subsystem has a dedicated 68HC11 microprocessor, which provides for a parallel, distributed, control ability. The dedicated microprocessors can perform various tasks without intervention from the PC, freeing the workload of the PC. Since the laptop PC has only one serial port, and the need to communicate with three different subsystems exists, an RS232 serial multiplexer was designed. The multiplexer is controlled by the host PC's parallel port which selects the subsystem the host wishes to communicate with. An attempt was made in the hardware design and assembly programming of each subsystem to maximize its versatility.

The ultrasonic subsystem is used to provide distance information from the robot to various objects within its immediate surrounding. The subsystem has the ability to rotate its ultrasonic transducer via stepper motor to given angles, thus adding the ability to pan the transducer 360 degrees. The panning and firing of the transducer are accomplished through serial communications with the PC.

The stepper motor control subsystem was developed with three modes of motor operation: *User Controlled*, *Continuous*, and *Incremental*. Under User Controlled mode, all motor movements (pulsing of the stepper motors) are handled through serial communication with the host PC. The Continuous mode of operation provides the robot with

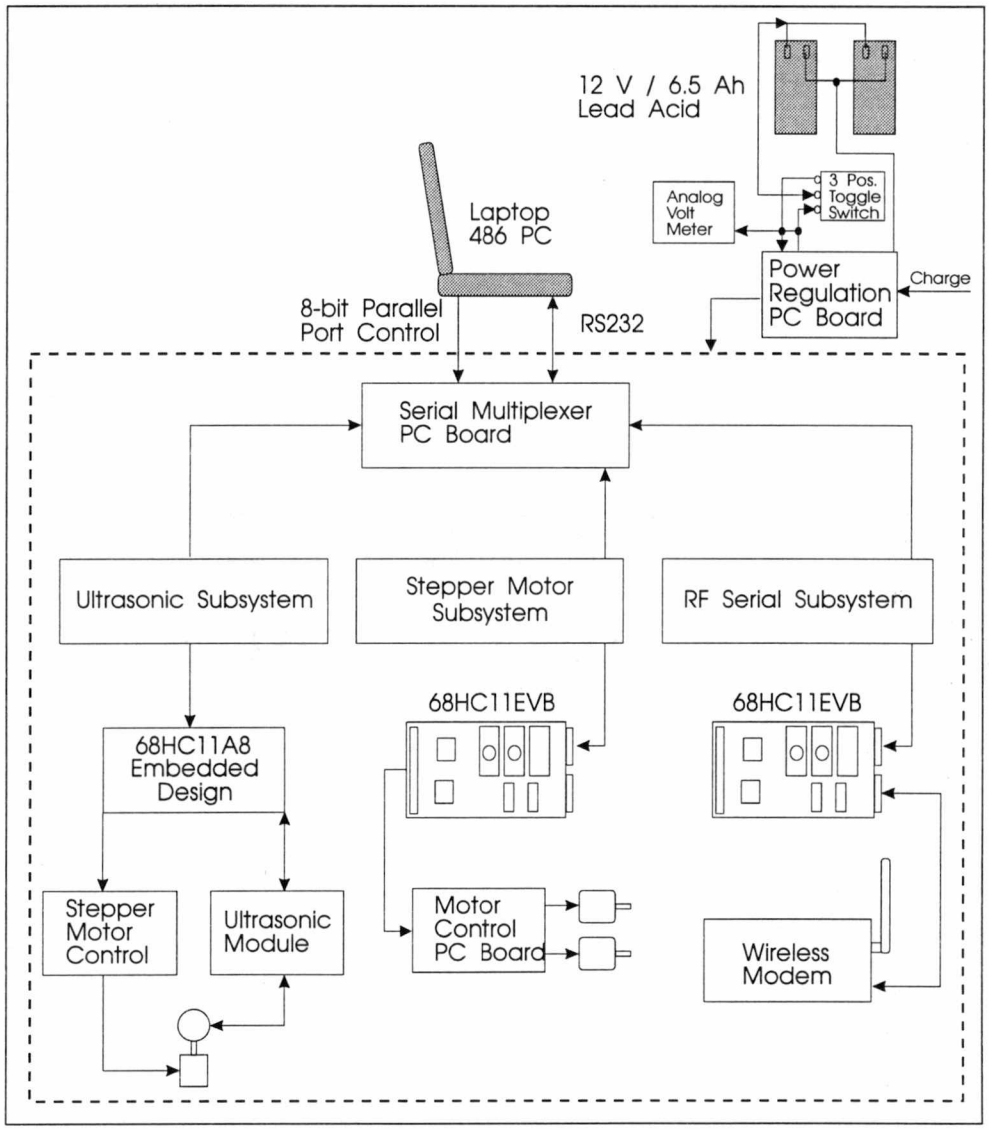

Figure 2.2: *SMAR-T's Electrical Layout and Control System.*

the subsystem's ability to continuously pulse the motors at given rates. The speeds of the motors can be altered by the host at which time the subsystem will converge from present motor speeds to the newly desired speed based on acceleration/deceleration rates selectable by the host. The Continuous mode of operation is the dominant mode used for the robot. Under the Incremental mode, the host can specify a certain number of steps to be given to the motors. This might be advantageous when deciding the robot must move a particular distance, based on a number of pulses to be given to the motors, and then stop. The host can periodically check whether or not the move has been completed by polling the subsystem for a software flag. For any mode, when instructed by the host, the motors can be enabled/disabled, and the step type and direction can be altered. Software encoders have been added to the subsystem for keeping track of the number of pulses sent to each motor when the motors are enabled. Current configuration parameters and pertinent variable values within the subsystem can be retreived by the PC, if so desired.

The RF serial subsystem provides the robot the ability to communicate with up to five additional robots if desired. Incoming serial data from the wireless modem is interrupt driven so as not to miss any packets from the other robots. A protocol was developed and checksums are enforced for the sending of such packets. Upon receiving packets from the other robots, the subsystem stores the packets within internal RAM to be later gathered in a LIFO format by the PC when desired. The PC has been given the ability to querry the subsystem regarding incoming packets received by the subsystem. The PC can send packets of its own to the subsystem for further broadcasting over the wireless modem. The RF subsystem merely handles the reception and transmission of an established protocol regarding packet information. The information within the packet

is further deciphered by the host PC once gathered from the subsystem. This allows for easy modification and addition of broadcasted messages, for cooperative purposes, between the robots without altering the existing assembly code for the subsystem.

CHAPTER 3

Literature Review of Systems/Architectures

A variety of systems and architectures and combinations thereof can be found in the literature regarding mobile robots and how their architectures provide intelligent control for the robot.

3.1 Microprocessor Based Architectures

3.1.1 Tee Toddler: A light seeking robot capable of transmitting sensory information and receiving control commands sent by a PDP-11 computer base.

The Tee Toddler was a four wheeled robot used to find high intensity light sources [AR78]. Though old in technology, many of the principle issues concerning the design of a mobile robot had to be addressed. The mobile robot used an onboard Z-80 microprocessor to collect sensory information and then relayed that information to a stationary PDP-11 computer base over FM data links as seen in Figure 3.1. The links have two pairs of transmitters and receivers which operated at different modulation frequencies and transformed the binary data into its frequency modulated counterpart. The control strategy was to allow the PDP-11 to provide the intelligence for the robot. The on-board Z-80 processor gathered the necessary sensory input and then relayed it to the PDP-11; the Z-80 was also responsible for capturing the incoming PDP-11 motor

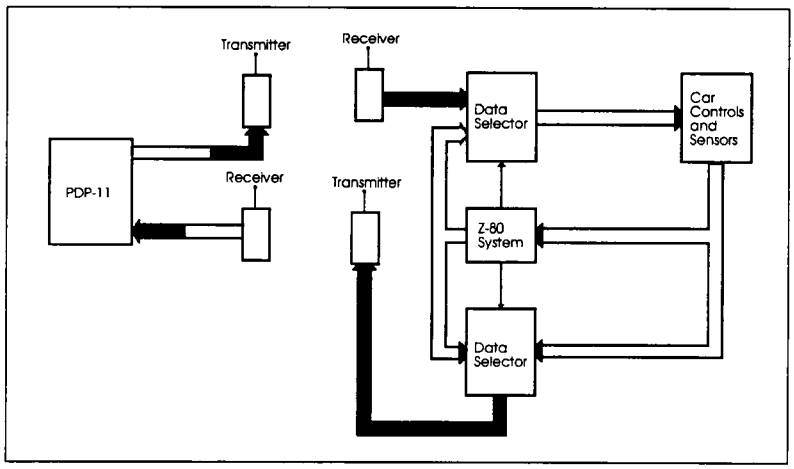

Figure 3.1: *Tee Toddler's computer systems interaction.*

commands and then alter the robot's motors accordingly.

To provide navigational aid, the robot uses ultrasonic sonar to avoid objects lying in its path, and a "rotating eye" to detect the direction of the light source, the robots destination. The robot was first designed to use infrared light to aid in obstacle avoidance. However, since the distance measurement seen using infrared is dependant on the color of the object, the sensor reads varying degrees of distance based on color. The use of ultrasonic was thus employed for detecting nearby objects at a maximum range of five feet.

The light source's direction was found using a mirror placed on top of a rotating disk which contained 16 slots equally spaced around its circumference as seen in Figure 3.2. These slots were sensed by an optical switch as the disk rotated. As the disk rotated, the slots passed over an optical switch which then served as a clock signal to a 4-bit counter. Once the mirror reflected enough light, an optosensor was triggered and the counter's outputs were then latched indicating the direction of the light source. A seperate slot and corresponding optical switch was used to reset the counter. Thus, a total of 16

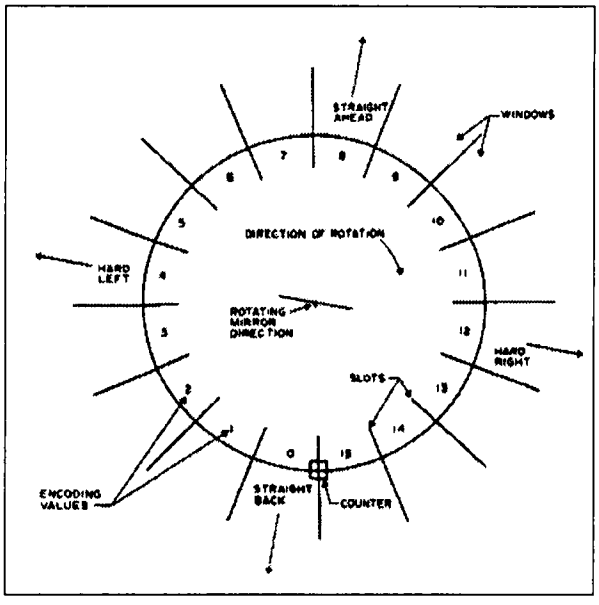

Figure 3.2: *Tee Toddler's "Electronic Eye": A rotating disk to sense the direction and ultimate goal of intense light sources.*

possible directions for the light source were given.

The motor controls for the Tee Toddler included a steering system and drive system, each operating in a similar fashion with minor discrepencies. Servo motors controlled by pulse width modulation were used as shown in Figure 3.3. The motor speeds are given by 0, 1, 2, and 3 which were decoded to stop, slow, medium, and fast. Another bit controlled forward and reverse thus enabling a total of seven motor states.

The final sensory circuit discussed determined the finding of the light source thus completing the task. Three photosensors were placed on the front of the vehicle for the detecting the intense light source (destination) once the robot was close. A trimpot can be altered to adjust the sensitivity of the detecting photosensors shown in Figure 3.4.

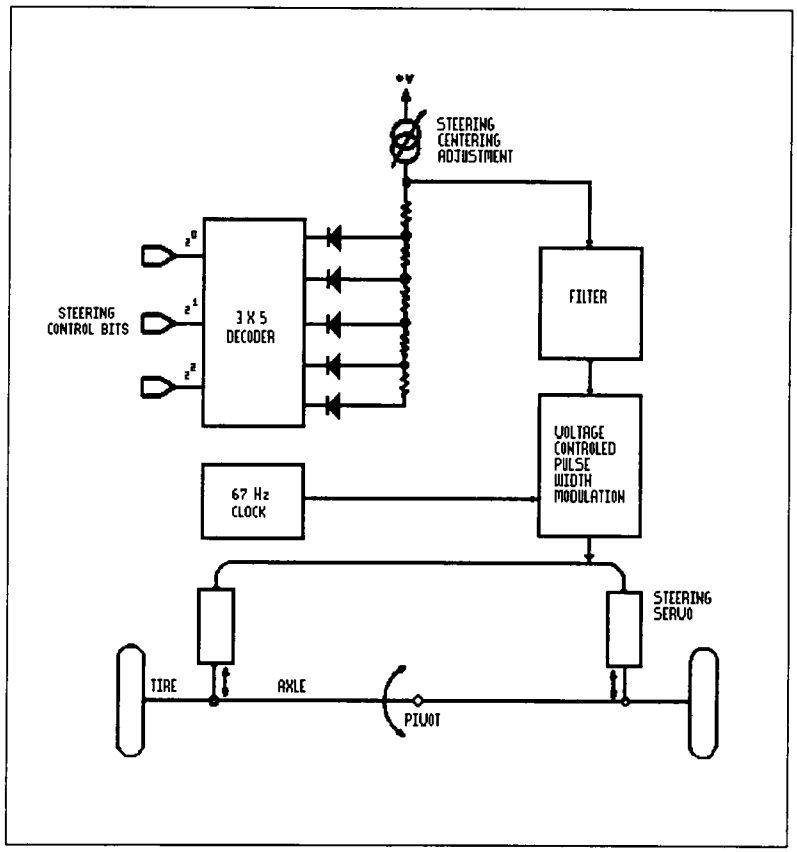

Figure 3.3: *Tee Toddler's servo motor control for steering.*

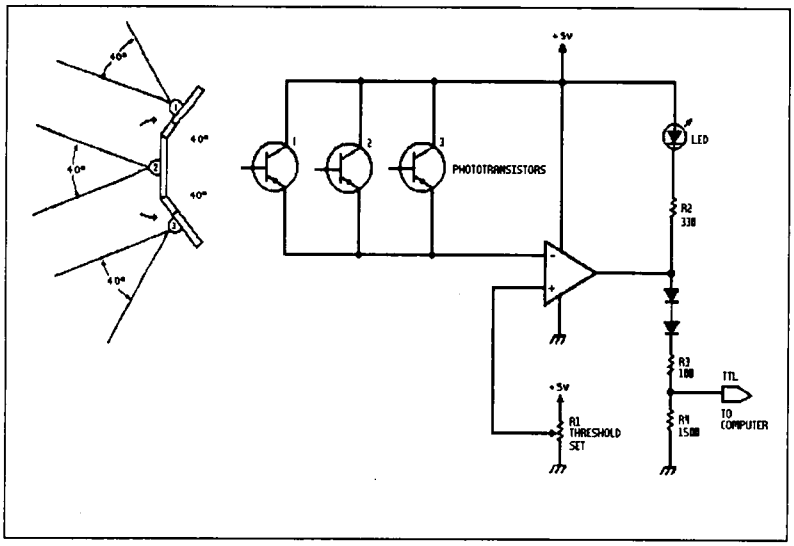

Figure 3.4: *Tee Toddler's sensory system to detect intense light indicating task completion.*

3.1.2 Robbie: One of Many Cooperative Robots Utilizing a Swarm-Like Intelligence in Search of Pollutants

Robbie comprises one of many miniature robot units (RU) in a distributed robotic system whose purpose is to search and find possibly toxic gas odors using a swarm-like fashion [GDMO92]. Each robot will have a variety of sensors, shown in Figure 3.5, including an ultrasonic transducer for obstacle avoidance, microphones, gas sensors, and infrared transmitter-receiver pairs for communicating with other local RU.

The swarm-like behavior is symbolic of a multitude of these robots each in search of pollutants and executed in computer simulation. Each robot would start at some point in space and then begin moving outward in all directions. After some time has expired, the robots spread out and theoretically cover a large area. Once the pollutant has been detected, the detecting robot can call on other robots for investigation purposes and alert the user located at some base station. Having a multiple number of robots search

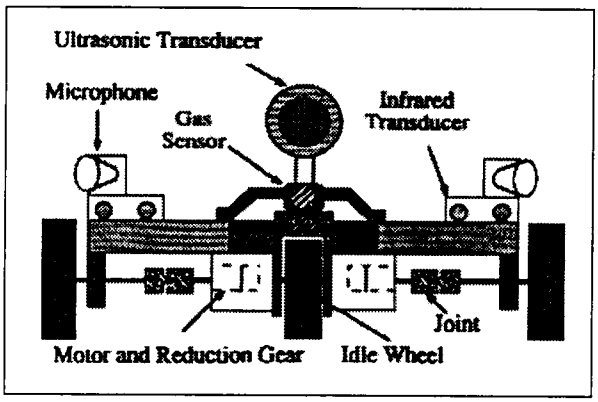

Figure 3.5: *Robbie: One of many miniature robots to be used for the detection of gases in an environment and operating in a swarm-like manner. Each robot is capable of providing cooperative efforts in search of gas pollutants.*

the area reduces the time and chances of detection as opposed to a single, large, and complex robot.

Each robot is to be simplistic in nature, and to be programmed with a subsumption-based architecture as shown in Figure 3.6. Though each robot has the same sensory input capabilities, the robots can be programmed to have slightly different instinctive behaviors. This approach is envisioned because certain conditions may be more favorably suited with a variety of different behaviors from a group perspective. Having a multitude of "personalities" for the robots attempts to provide the ability to overcome various situations which can arise in a dynamic environment and possibly not attainable by a multitude of robots each programmed with the same behavior structure.

The communication scheme is of prime consideration with a host of robots employed in a swarm-like fashion able to communicate with one another. Two levels of communication were presented. A local means of communication was to be achieved by the use of the infrared emmitter-detector pairs. This would be used by robots in the same vacin-

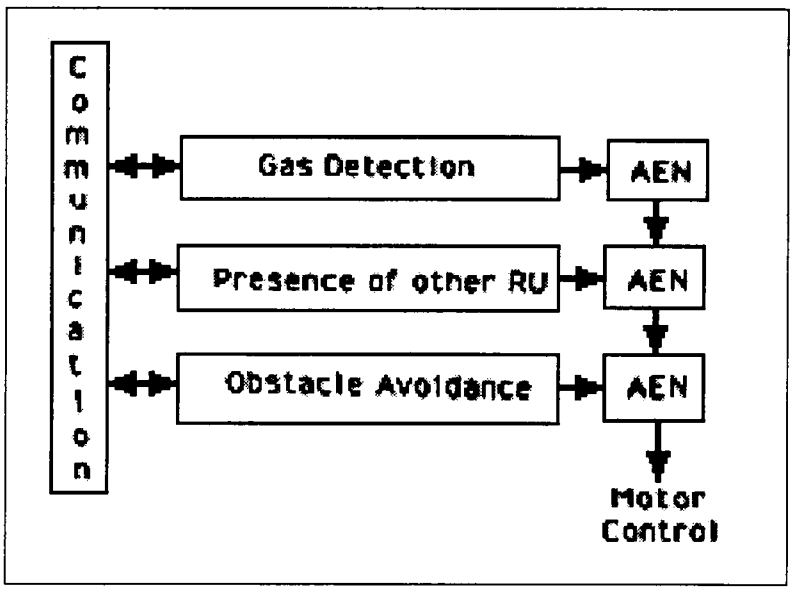

Figure 3.6: *Robbie's subsumption-like architecture. The AEN blocks are the asynchronous exchange network modules to determine priority for motor control.*

ity working in some cooperative fashion. A global means of communicating would be provided by the addition of an FM modulated signal. Since each robot would have the ability to communicate globally at the same modulation frequency, a means of deciding when to broadcast, validating the request, and provide error correction are required.

A robot may only broadcast when the line is not busy, thus none of the other robots are currently broadcasting messages. When broadcasting, the robots would provide checksum capabilities and expect to hear an *acknowledgement* message to be returned from the other robots. Many different robots can provide the *acknowledgement* signal providing they hear the broadcasted message. If the broadcasting line is busy, each robot is to wait a random period of time before trying to broadcast again. A time-out capability is provided for the sender of such a message. If no *acknowledgement* signals are returned after the time-out, the robot will assume the data has been corrupted and

will re-broadcast.

3.2 Subsumption Based Architectures

Rodney Brooks from the Artificial Intelligence Laboratory at M.I.T. pioneered the formulation of a sumbsumption architecture to be used on various mobile robots designed in his lab[Bro90]. The concept involves accomplishing a task by combining a series of simple reflexive behaviors, such that they provide a desired collective behavior. This is accomplished by prioritizing the individual behaviors and allowing high priority behaviors to supercede lower priority responses. Each behavior reflexively responds to certain sensory input. The lowest level behavior is considered to be the default action. Each successively higher behavior supercedes if its conditions are met. Typically this has been accomplished by using a series of microprocessors running in parallel. Each processor is responsible for a particular behavior or group of behaviors. When a processor responds, it intercepts the signal originating from any of its subordinate processors, and replaces the response with its own. Otherwise, the response from the subordinate processor is passed on to the next processor in the link until finally a command reaches the desired hardware. This creates a chain of processors that are typically interconnected by a serial communications link or communicate by a similar means. This paradyme provides for addition of new behaviors without reconfiguring existing behaviors. Also, the distribution of processing eliminates the need of powerful processors to accomplish a complicated task.

3.2.1 Herbert and Toto

Some of the M.I.T. mobile robots exhibited a high degree of intelligence based on the subsumption-based architecture. Herbert, a mobile robot at M.I.T., utilized the subsumption architecture to find and collect soda cans in a dynamic office environment [Con90]. The robot utilized a series of self-contained 8-bit microprocessors to provide the necessary behaviors for performing the task. A total of 41 behaviors were implemented in the subsumption architecture. Herbert's hardware also included a laser scanner with camera, ultrasonic transducers, and a gripper. Toto, another mobile robot from M.I.T., utilized the subsumption architecture to provide collision-free navigation, dynamic landmark detection, map construction, and path planning [Mat92].

3.3 Fuzzy Logic Control

Fuzzy control provides another means of intelligence. This method attempts to provide complex control using simple membership functions. Fuzzy control includes taking crisp sensory readings from various sources to be used as inputs to the system. These inputs are then transformed into corresponding fuzzy linguistic terms. Control rules utilize the fuzzy terms to generate fuzzy output terms. Finally, a method is used to defuzzify these output terms such that a transformation back to a crisp output value is realized and the controlling element can utilize this value. The procedures involved are located in Figure 3.7.

The inputs for the fuzzy system are considered crisp when taken from the sensors. The term crisp refers to actual data values. The data is considered exact, meaning there are no linguistic degrees or vagueness associated with it. To perform fuzzy inference, it is

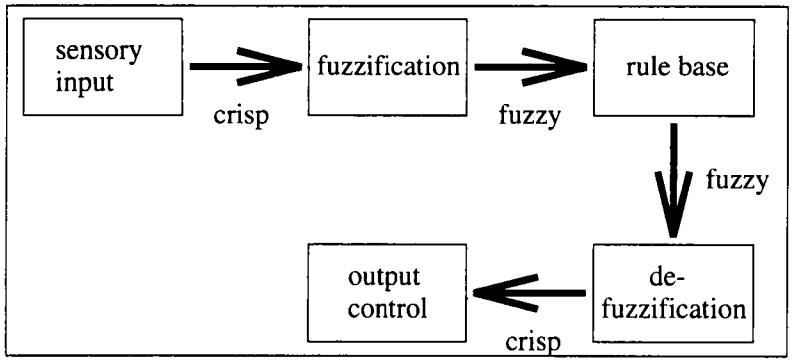

Figure 3.7: *Fuzzy Logic Control Inference Process.*

necessary to fuzzify these crisp values into linguistic counterparts giving them degrees of vagueness ranging from zero to one. The degree is dependent upon the transformation process whereby membership functions for given inputs play an important role. A given input can have a multitude of membership functions describing the various fuzzy linguistic terms associated with each input. The shape and number of the membership functions to use are left to the expert such that enough linguistics for given inputs have adequate meaning or spread to control the overall process. The membership functions typically overlap one another such that a given input when transformed to fuzzy terms gives rise to vague fuzzy terms whose degrees of strength are dependent upon the defining functions residing at the crisp input locations.

Once the inputs have been transformed into their fuzzy terms, control rules are then applied by an expert. A typical control rule might be, if *robot speed* is *very fast* and *obstacle* is *extremely close* then *robot speed* is *stop*. The total number of rules is dependent upon the expert. Primarily, each situation the expert finds to be crucial will warrant a governing rule. Many rules can fire at one time generating a multiple number of varying degrees of vagueness placed upon certain fuzzy output terms. The

fuzzy output terms can now be used to provide another transformation, defuzzification, to return back to crisp or exact values to be utilized by the controller. The centroid method is a popular choice of defuzzification.

MARGE, winner of event III in the 1993 AAAI Mobile Robot Competition, utilized an architecture of distributed fuzzy logic controllers to provide a set of reactive behaviors as seen in subsumption based architectures [GL93]. The VME-based robot utilized three 68030 Motorola processing boards and its sensors includeded 2 CCD cameras, 19 ultrasonic transducers, 28 tactile whiskers, and wheel encoders. It was able to locate certain boxes with its vision system, and provide the necessary navigational abilities to drag the boxes to various locations with an on-board suction device.

3.4 Cooperation Using Visual Observation

Cooperation by visual observation provides the ability for high level task coordination with minimal external communication between robots [KKR+94]. Many issues concerning cooperative task coordination are addressed with primary focus on the ability of a robot to self-detect the need of assistance by another robot. Able to perform such a feat, a robot can provide the necessary assistance with very little or no communication taking place between the cooperating robots. Such a capability reduces the overhead associated with having to constantly send information regarding position, current state of task completion, and other necessary communication parameters for coordination purposes.

The system's architecture for each robot is given in Figure 3.8. Vergence controlled stereo vision cameras are used to detect objects and other robots in the environment.

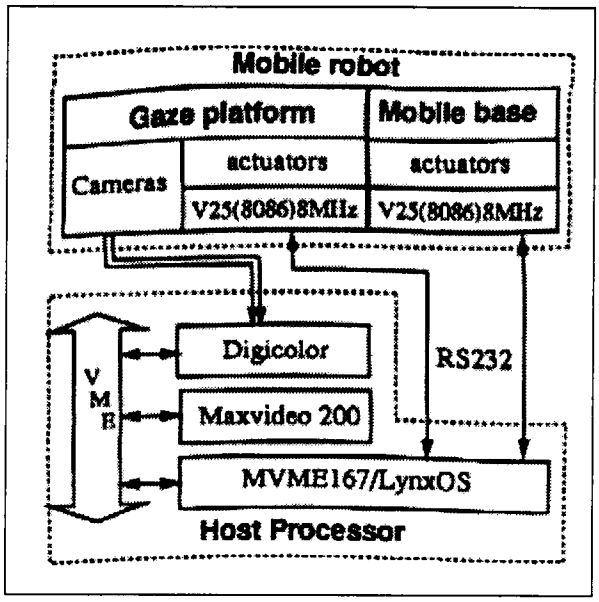

Figure 3.8: *Architecture used for Cooperation by Observation.*

Optical flow, and other types of algorithms were utilized to pre-determine the type of possible needed assistance by other robots. The ability to provide such a level of competence requires the use of expensive hardware for real-time vision processing.

CHAPTER 4

Power Distribution and Regulation

A printed circuit board was designed to provide supply voltages of -12, 5, and 12 volts to the various electrical systems on the robot. Two 12 volt 6 amp/hour lead acid batteries are connected in a parallel fashion as the input power source for the board. The battery source is further distributed to the various components on the board for power regulation and distribution purposes. The -12 volts is generated from a negative converter and is used for the RS232 communications. Two 5 volt regulators are used to supply logic power for all digital systems on-board the robot. The 12 volt battery source supplies the motor voltage, input voltage for voltage regulation, and RS232 communications. Fuses are used for protection from short circuit conditions. A single triple-throw two position switch is used to provide "on", "off", and "charge" states of operation. An external input to the board is provided to charge the batteries. Figure 4.1 shows the designed printed circuit board.

4.1 Batteries / Power Consumption

To provide tetherless operation, mobile robots require the use of batteries. Rechargeable batteries are more desirable for repeated uses. Various types of rechargeable batteries exist, the more popular being lead acid and nicad.

Lead acid batteries, the selected battery, exhibit a gradual loss of voltage, as opposed to a sudden voltage drop by nicads, during charge depletion. Nicad batteries also exhibit

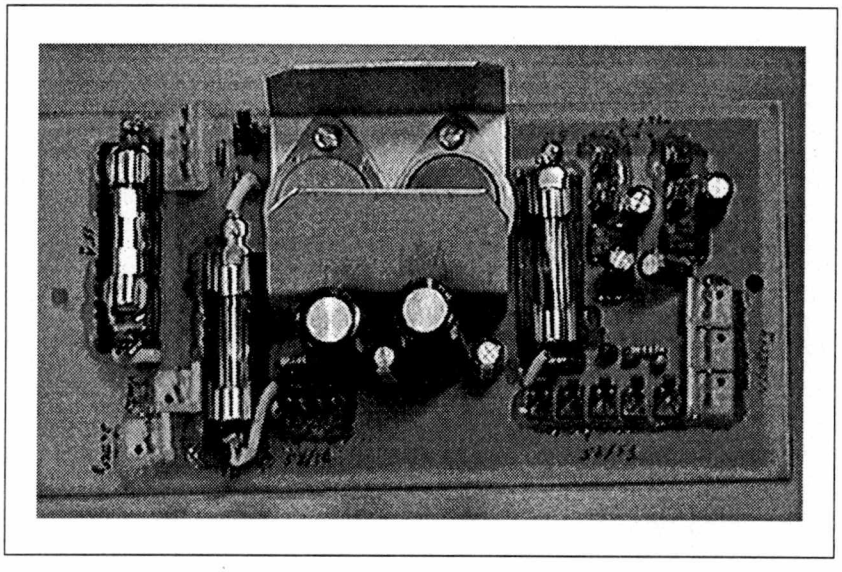

Figure 4.1: *Power Regulation and Distribution printed circuit board to supply the necessary operating voltages required by the robot.*

a strange memory effect which occurs after repeated partial recharging without first fully depleting the charge of the battery. The nicad will memorize this unwanted effect, and will no longer be able to supply a true full charge's worth of power. Lead acid batteries do not exhibit this effect enabling them to handle the partial charging over numerous accounts. Lead acid batteries have a minimal internal resistance associated with them enabling them to source large amounts of current without dropping significant amounts of voltage in the process [Mar92].

An estimated total power consumption of 72W for the robot has been computed for the sources shown in Table *4.1.* The life of the batteries depends heavily on the length of time the robot is moving since the motors draw the majority of power. An average of 30 minutes of robot movement has been observed before the batteries began to effect system reliability due to low voltage levels.

Table 4.1: *Power Consumption.*

Source	Amps	Watts
Motor subsystem	0.6A @ 5V	3.0W
Motors	4.8A @ 12V	57.6W
Cooling fan	0.1A @ 12V	1.2W
RF serial subsystem	0.5A @ 5V	2.5W
Wireless modem	0.1A @ 12V	1.2W
Ultrasonic subsystem	0.5A @ 12V	6.0W
Serial multiplexor	0.05 @ 5V	0.25W

4.2 Converters / Regulation

A means of providing constant voltage values for a fluctuating battery source is needed for proper circuit operation. Negative converters are used to provide negative voltage for RS232 serial transmissions and 5 volt regulators for a stable logic source voltage.

4.2.1 Negative Voltage Converters for Serial Communications

Two negative voltage converters *ICL7662* produced by the Harris Corporation are used to provide the -12 volts for the RS232 communications [Har93]. Each converter is given two external connections. Two 10μF capacitors are the only necessary components for the *H7662*'s onboard charge pump to operate properly. For 10μF capacitors, the oscillator frequency is 10 kHz, and can sink an approximate 100mA of current. The output source resistance is approximately 60 ohms with a minimum voltage input specification of 4.5 volts for correct operation.

4.2.2 5 Volt Regulators for Logic Circuits

Two 5 volt regulators are used for the digital systems on the robot. Three capacitors are placed before each regulators input. The 330, 22, and 0.1μF capacitors are connected in parallel to provide noise suppression for the regulators input. Two capacitors of 22 and 0.1μF are placed in parallel at the outputs of each regulator to suppress noise spikes induced from logic switching. A silicon diode is connected between each 5 volt output and source voltage input. The "flyback" diode prevents the regulator from a reverse biased condition when the regulators output voltage becomes greater than its input voltage. This condition should not occur unless the input voltage undergoes a short circuit. Such a condition can damage the regulator without the connected diode. In this condition, the diode will conduct, thus alleviating the reverse bias of the regulator.

The efficiency of the regulators depends on the load current. Regulators are more efficient when driven to levels approaching their maximum capabilities. Each regulator is heat-sunk by connecting aluminum fins to the regulator's metal casing and placing heat-sink compound between the two aiding the transfer of heat. One particular regulator is capable of supplying a maximum of 1 amp. A total of two external connections can be made to this regulator. A 1 amp fuse is used in series with this 5 volt output for protection from short circuits. Both digital systems, consisting of the motor subsystem and it's external motor control board, are supplied by this regulator. A red LED will light to indicate no blown fuse and proper operation of this regulator. A 5 volt 5 amp regulator is used to power the existing digital systems. A total of five external connections can be made to this regulator. The rf serial subsystem is powered by one of these connections, and is protected from short circuit conditions through a 5 amp fuse connected to its output. An additional red LED details the condition of this regulator.

4.3 External 12 Volt Connections

Four 12 volt external battery connections are made available via connectors. One is used solely as the motor source voltage. Its connection is close to the batteries to prevent voltage deviations from occurring due to the large current requirements of the motors. Three external connections located further from the battery can be made for other 12 volt source requirements. One is used to supply power to a 12 volt 0.1 amp brushless fan. The fan is used to cool the power regulators and power mosfets on the motor control hardware. Both the motor control hardware and power regulation / distribution boards are enclosed in a plastic casing with ventilation holes at one end and the fan at the opposite. The casing allows airflow generated from the fan to pass over the heated components for cooling purposes. The air cooled enclosure is found between the battery sources in Figure 4.2. The two additional 12 volt power connections are used for the sonar subsystem, serial multiplexer, and RS232 serial communications.

4.4 Additional Components

A 15 amp fuse is placed in series with the existing battery sources to provide an overall current protection. A 30-amp triple-throw double-pole power switch is placed after the fuse to provide "on", "off", and "charge" positions. A 15 volt panel meter is connected to this switch to show battery levels when the switch is in the state of "on" or "charge". When the switch is on, the battery source is distributed throughout the board, and motor voltage and power regulation becomes active to further source existing connected systems with power.

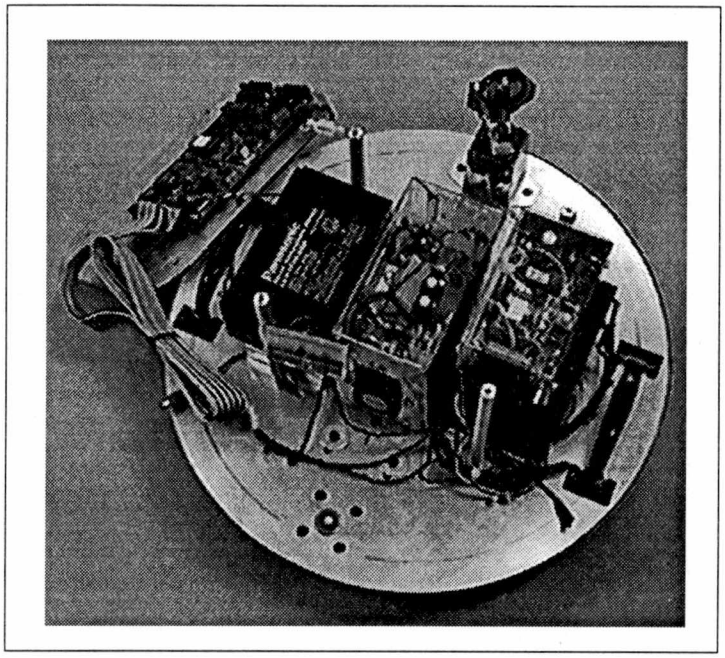

Figure 4.2: *Fan-cooled enclosure for high power motor drive circuitry and power regulation hardware.*

4.5 Battery Charging

When the power switch is placed in a "charge" state, an external connection is made available to charge the batteries. The power regulation and distribution boards components are not connected to the battery in this state. A diode is placed in series with the charging connector to prevent an external short circuit condition of the batteries. When an external voltage is applied at a level greater than the battery level plus an approximate 0.7 volts, the diode becomes forward biased and begins to conduct. A female phono-plug is used as the external connection for charging. The analog meter is active in this state and shows the current battery voltage level while charging. Figure 4.3 shows the full schematic of the PC board.

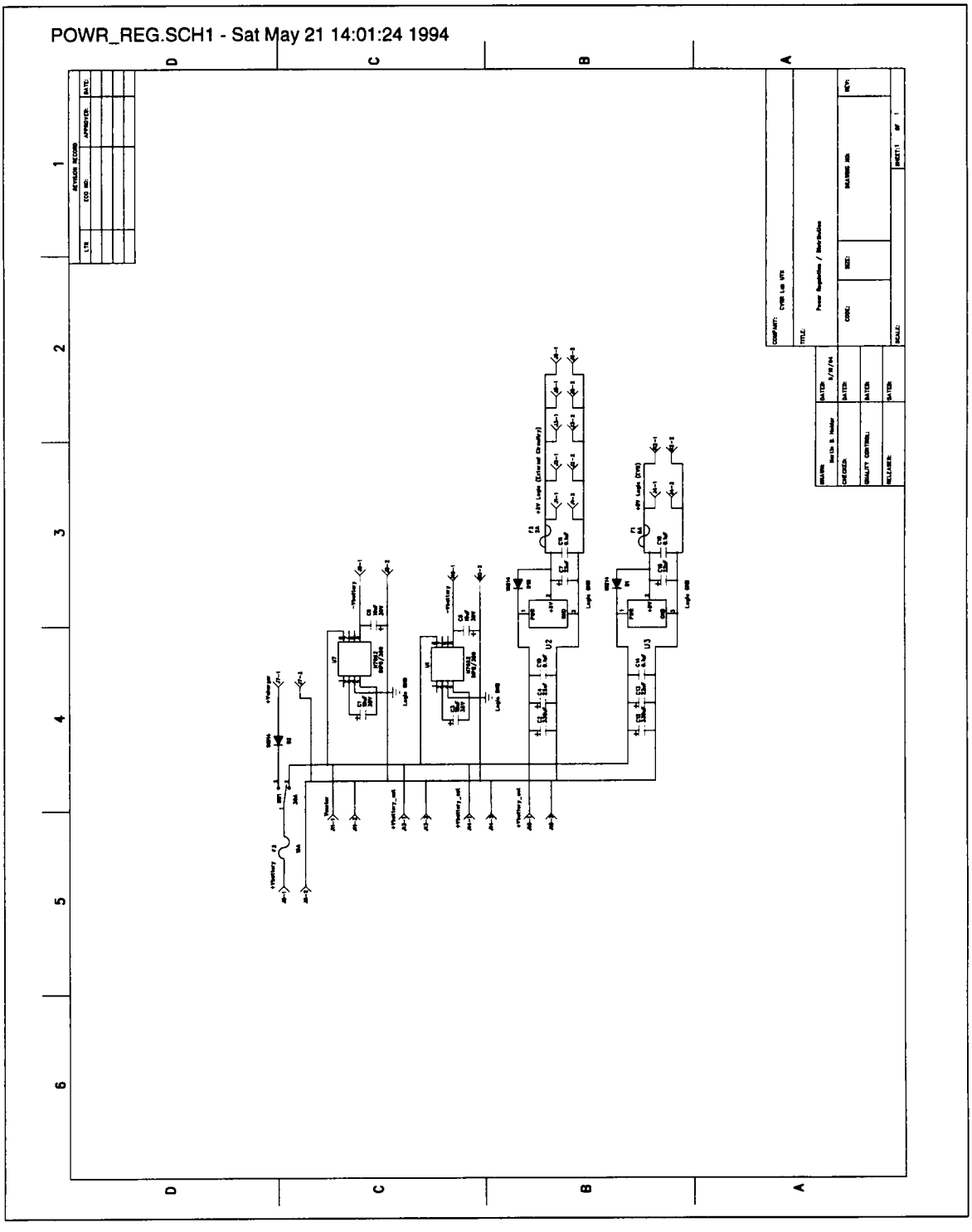

Figure 4.3: *Power regulation and distribution hardware.*

CHAPTER 5

Ultrasonic Transducer Subsystem

An embedded system was developed to detect obstacles around the robot and is controlled by a single 68HC11A1 micro-processor which accesses its programmed code through an external EPROM. A single ultrasonic transducer is mounted to a stepper motor providing the ability to rotate the sensor through various angular positions. An RS232 serial protocol and command set for the system was developed to handle all dedicated control aspects for obstacle detection. The various hardware / firmware issues involved with the embedded system are further discussed along with C routines which were generated to enable simple function calls at a higher level for dictating such controls. Additional C routines were generated to provide a menu driven control set to control all aspects of the embedded system and graphics can optionally be displayed to provide a visual representation of the robot's surrounding obstacles. Further issues concerning the calibration and testing of the subsystem's ability to accurately detect obstacles is discussed.

5.1 Sensors for Object Detection

Autonomous robots working in an unstructured environment need a means of gathering data describing their surroundings. Human beings rely primarily on vision to move around a room. Depth information, along with location is recognized in a very short time period. Human beings, have superb capabilities of translating images into a form

that we can instantly react to it. Thus, moving around a cluttered environment with no prior knowledge of the environment is somewhat an easy task. Machines are capable of such actions, yet one must choose the best solution to the given problem in terms of sensory input and cost. The best design is typically the one which is capable of solving a given problem at the least amount of cost with given time constraints imposed. For mobile robots to be capable of moving in an unstructured environment, various options of sensory input exist. Vision systems and laser scanners are capable of providing large amounts of data at the expense of increased costs and additional processing power. The cheaper alternatives are infrared proximity sensors and ultrasonic transducers which give less information at reduced costs and processing.

5.1.1 Machine Vision / Cameras

Machine vision is a very involved process requiring the use of cameras and specialized hardware for grabbing the images and then processing the information or perceiving the given information to do some useful task. Should depth information be pertinent, the use of two cameras (stereo vision) can be used. By finding identical points in both images, the use of triangulation will give depth information. Thus, calibration of the cameras plays a vital role in the accuracy obtained. Lighting conditions can be a major factor in this type of system.

Laser scanners are an additional option which gives added depth information in the process using a single camera. Lasers are capable of scanning in either single or two-dimensional planes. These devices are susceptible to certain lighting conditions and distances they can handle.

5.1.2 Infrared Proximity Sensors

Infrared proximity sensors are a valid choice in terms of expense for very simple obstacle avoidance maneuvers. They do have limited distances and typically are found to give a binary "on" "off" input. They are sensitive to lighting conditions which generate light in the infrared regions. Such lighting can easily be found from sunlight or fluorescent lights which emit infrared. Most of these types of sensors are pulsed at particular frequencies. A bandpass filter is used to mask out other types of invalid sources for detection purposes. These types of sensors rely on sending out pulsed infrared light; and, if an obstacle is near, the light is reflected back to the sensor. If no obstacle is near, the sensor does not detect the light. Most of these devices with longer range capabilities magnify and focus the light via a lens to gain further distances. Colors of the surrounding area play a crucial part in the detection phase. Obstacles with darker colors tend to absorb the majority of this light; thus, the sensor is usually unable to see such obstacles. Obstacles with lighter colors are seen at further distances, while obstacles with darker colors, if seen, are seen at shorter distances. A typical infrared sensor can be seen in Figure 5.1; the unit contains the receiver and transmitter circuitry sending a binary signal for detection.

5.1.3 Ultrasonic Transducers

Ultrasonic transducers provide yet another feasible alternative to obstacle avoidance. These transducers utilize sound pulses which are sent and later received on the same transducer. By knowing the speed of sound, the time differential between transmission and reception is used to calculate distance. A bandpass filter, which is installed on the receiver section, filters unwanted noise from the ultrasonic transducer. A common

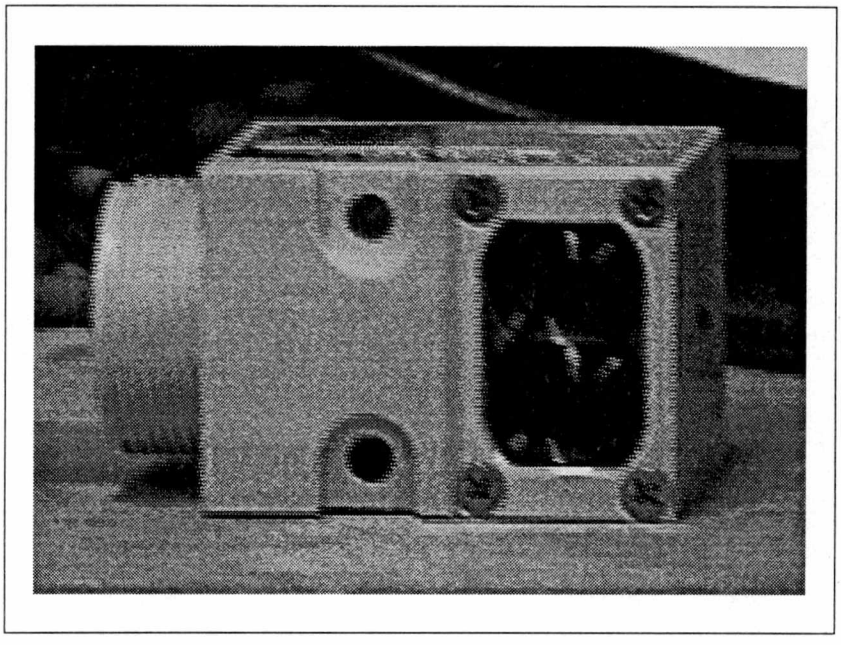

Figure 5.1: *Infrared proximity sensor for detecting binary (close/far) distance.*

chipset duo used in ultrasonic applications are the Texas Instrument (TI) TL851 and TL852 [Tex88]. These chips work in tandem with each other. One is used for the transmission of the sound pulses, the other for receiving. Typically, a pulse train of sixteen pulses at a frequency of 49.4 kilohertz is issued to excite the transducer which provides the sound. The actual transmission is three hundred volts per pulse. After a given time delay to allow for the transducer to settle from initial ringing, the receiver portion begins to listen for incoming sound through a bandpass filter centered at 49.4 kHz with 5 kHz lobes. The receiver has the capability to alter its gain or sensitivity to enable it to hear the pulses which travel further distances and are attenuated as a result. When a given pulse is detected (threshold tripped) the time differential is noticed; and distance of flight can be calculated. The calculated distance must be divided by two to accommodate the sound's travel to and from the given obstacle. These types of

sensors have limitations as well. The typical range of the sensor varies from thirty-five feet maximum to approximately six inches minimum. Additional concerns are with reflections off certain obstacles. Depending on the angle in whichthe sound hits the obstacle, the sound may or may not bounce back properly. Additionally, typical ultrasonic transducers emit sound in a 30° cone. Figure 5.2 illustrates this limitation. Any obstacle within this cone could possibly send the sound back triggering the detection

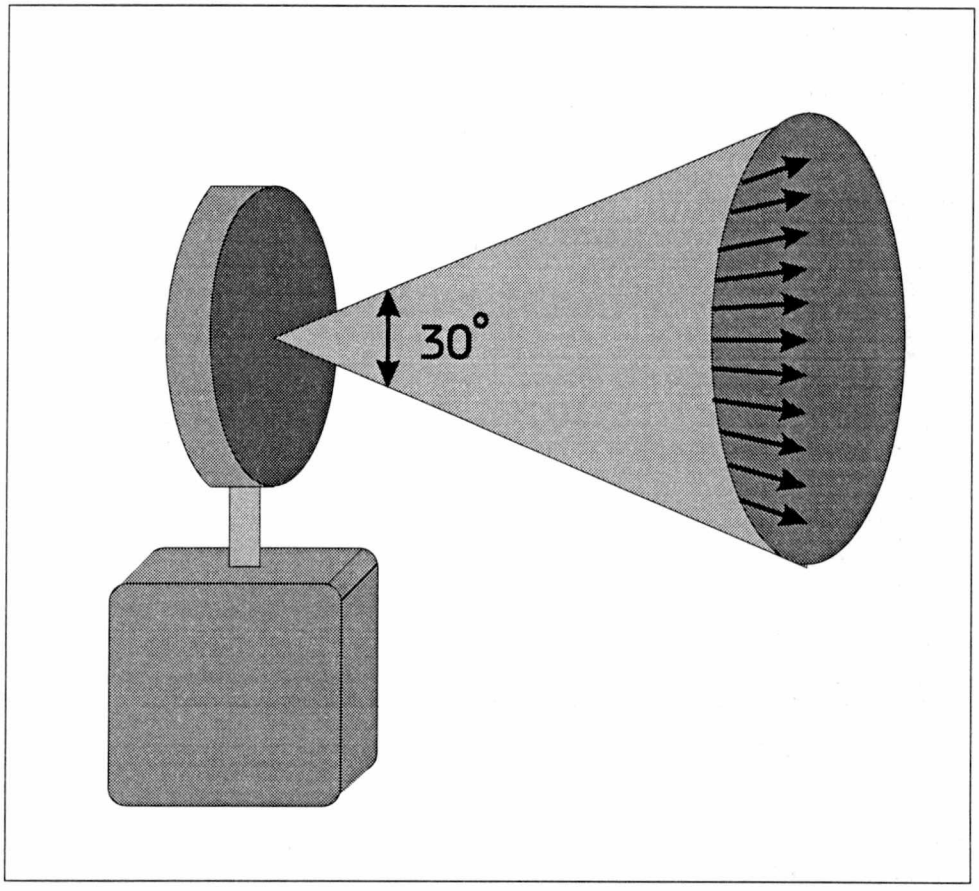

Figure 5.2: *Ultrasonic transducer sound emission after firing.*

system. In short, it's difficult to draw an exact location from which the sound is returned due to the cone.

TRC Embedded Ultrasonic System

Ultrasonic transducers are typically more robust than infrared proximity sensors for mobile robotic applications with the ability to see any color of obstacle. Typical robotic systems with such transducers will usually have multiple transducers mounted around the exterior of the robot in some circular fashion, with the transducers offset by a certain number of degrees from one another. Transitions Research Company (TRC) sells such a system to be employed on mobile robots [Tra91]. This system utilizes the TI chipset to send and receive the ultrasonic pulse train. An on-board 68HC11A8 microprocessor is used to handle all serial calls to and from the board as well as the lower level controls to enable the system to fire and listen for responses of the ultrasonic sound waves. TRC provide a means of setting the fire sequence of the transducers giving certain transducers primary and secondary firing capabilities. Each board is capable of handling eight ultrasonic transducers and eight digital infrared proximity sensors. Additional boards may be stacked, via a common bus, allowing for additional sensors if needed. Relays connect the TI chipset to the particular transducer when firing and listening. Likewise, the board provides the capability to turn certain transducers completely off and provide for a time-out distance for which the system will report no return signal found. As of December 1994, each board costs approximately five hundred dollars, and each transducer is around forty dollars. Figure 5.3 depicts the range system sold by TRC.

Ultrasonic Module for Firing / Listening

An article in Electronics Now revealed a "build it yourself" sonar radar system [Jac93]. The author, Ronald M. Jackson, utilized a kit containing a module which consisted of

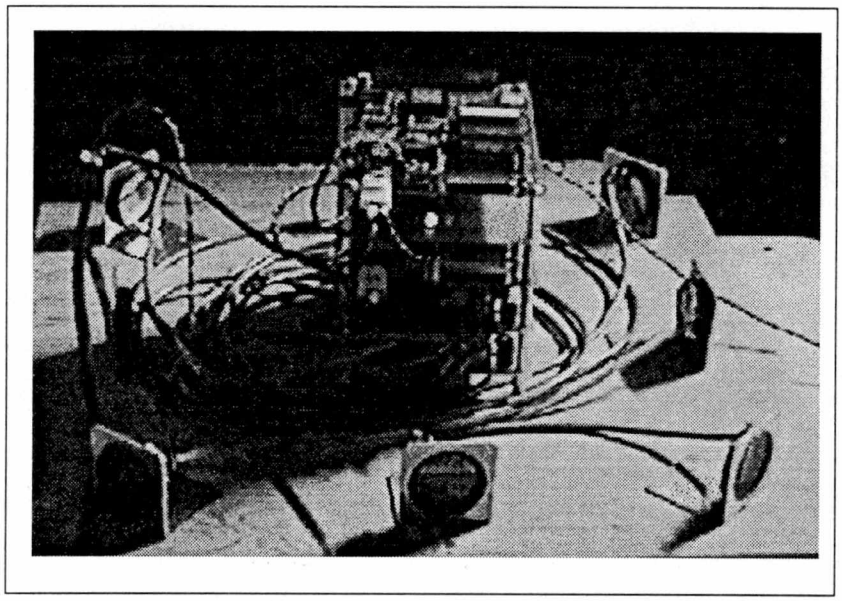

Figure 5.3: *TRC Proximity/Ultrasonic System.*

the TI chipset duo for sending and receiving ultrasonic pulses. The kit can be purchased from Fascinating Electronics and provides the two TI chips, step-up transformer, control circuitry, and a single ultrasonic transducer. Input/output control bits are to be handled by the user to fire the transducer and detect the pulses. As of December 1994, the cost of the sonar range kit is seventy dollars. The hardware for firing and listening is pre-built onto a printed circuit board as seen in Figure 5.4. The user must be able to provide timing capabilities and adhere to the low level control bits for proper operation. The article showed the transducer mounted to a stepper motor for panning purposes to take ultrasonic readings at various angles of interest. Schematics for driving the motor and sending proper signals to the transducer module to fire and listen were previously developed towards hardware utilizing an expansion slot within a personal computer. This was an on-going project in which the previous volume described the hardware design for the driving of the motors. This particular article mentioned how

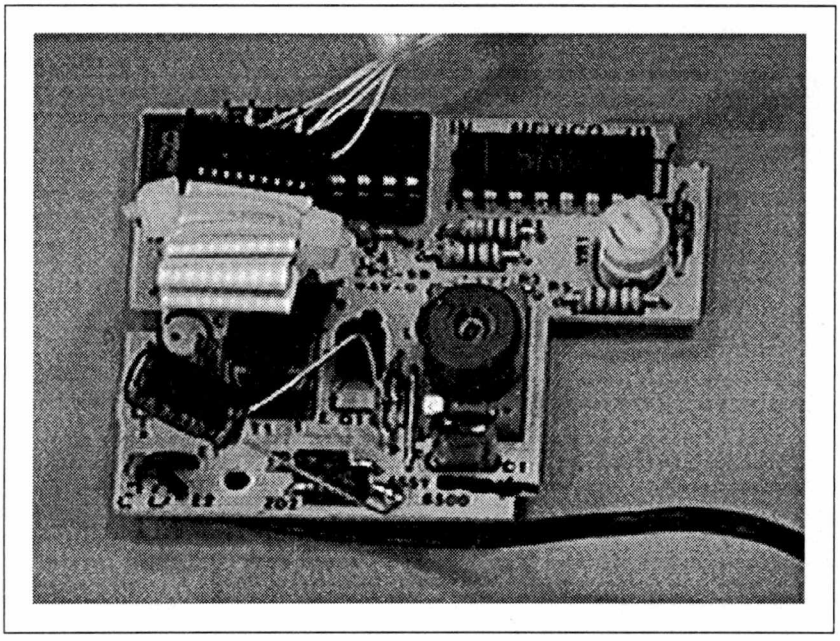

Figure 5.4: *Sonar ranging module to control the firing and listening of ultrasonic pulses.*

the ultrasonic range module functioned and gave an explanation of the needed low level control signals.

5.2 Embedded System Control Hardware for the Ultrasonic Subsystem

The chosen method of sensory input for obstacle detection is the ultrasonic transducer. Cost and robustness was a prime consideration in the decision process. The ultrasonic module is used for the firing and listening of the transducer.

The concept of a panning ultrasonic sensor is very intriguing. A system to provide such a capability can be built for under two hundred dollars. During the design of such a system, a 68HC11EVB was utilized for debugging purposes. Once the subsystem was operational under the evaluation board, the system was moved to a self-contained single-chip embedded system utilizing the 68HC11A1 micro-controller. The following

information is specific to the implemented single-chip system the robot utilizes as seen in Figure 5.5.

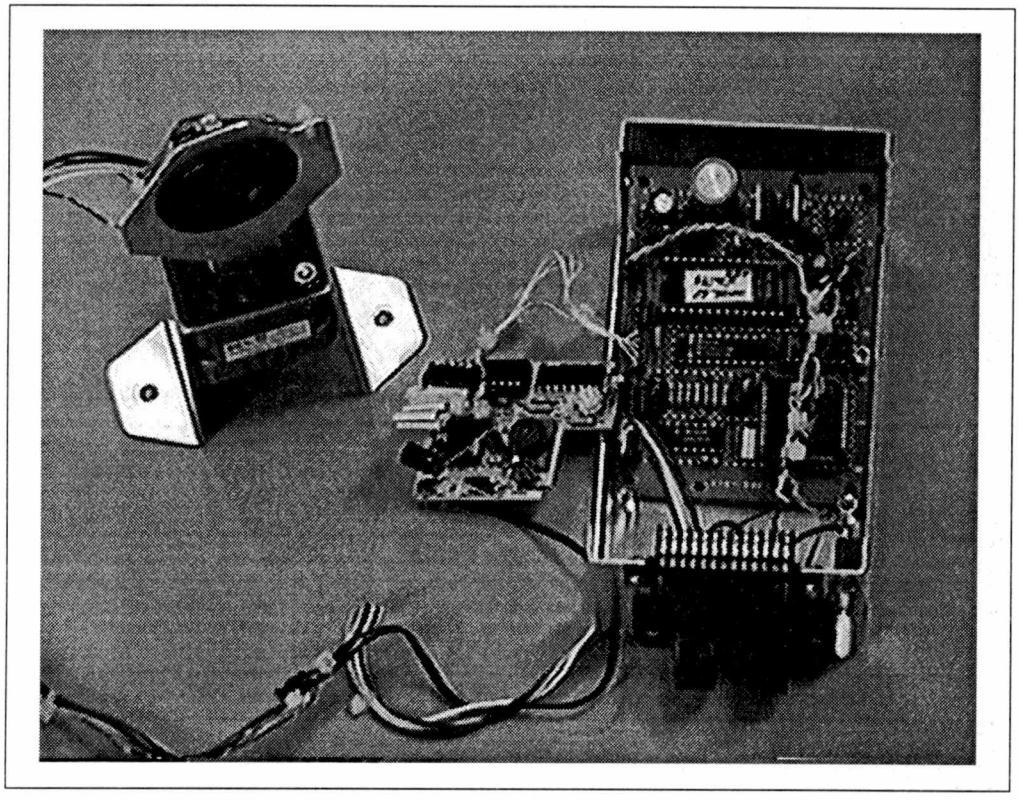

Figure 5.5: *Completed ultrasonic transducer hardware design.*

The focus of the ultrasonic subsystem is the 68HC11A1 micro-controller, which is manufactured by Motorola, and capable of sixteen-bit addressing with an eight-bit data bus. The bus speed of the micro-controller is 2 MHz, which is derived from a divide by four 8 MHz clock. The chip is very versatile, thus enabling the programmer a multitude of features such as interrupts, serial communication, and I/O ports. Primary interests for this micro-controller are based on its timing capabilities along with its ability to handle serial transmissions. The timing capabilities are useful for providing crucial interrupt driven timed responses to enable accurate timing/control for the ultrasonic

module. The ability to handle serial transmissions is useful in providing a command driven subsystem with simple commands to select the control actions of interest.

5.2.1 MCU's Mode of Operation for the Ultrasonic Subsystem

Upon power-up, the chip scans two input pins, **MODA** and **MODB**, to determine the mode of the chip's operation. A total of four modes of operation are available: normal single chip, normal expanded, special bootstrap, and special test. The normal expanded mode of operation is used; and, under this mode of operation, ports **B** and **C** are used for addressing with port **C** multiplexed as a data bus. Port **B** is an eight-bit output port and is used as the high byte of the sixteen bit address bus. Port **C** can be used either as an input or output port. When used as part of the addressing bus, Port **C** is set up as an output port and contains the lower byte of the sixteen-bit address bus. As a data bus, depending on *read* or *write* conditions, Port **C** is used as either an input or an output.

5.2.2 External Eprom Control to Access Firmware for the Ultrasonic Subsystem

Three control lines from the micro-controller enable it to access external hardware such as the EPROM which contains the assembly code (firmware); **AS**, **R**/$\overline{W}$, and **E** are the control signals enabling this capability. Figure 5.6 shows the hardware control for reading programmed data from the external EPROM. When the **AS** (address strobe) control signal goes high, port **C** is latched by the 74HC373 octal latch. The outputs of the latch contain the lower byte of the sixteen bit address bus. As seen in figure 5.6, twelve address lines are used by the 2764A 8K EPROM, and the outputs of the EPROM

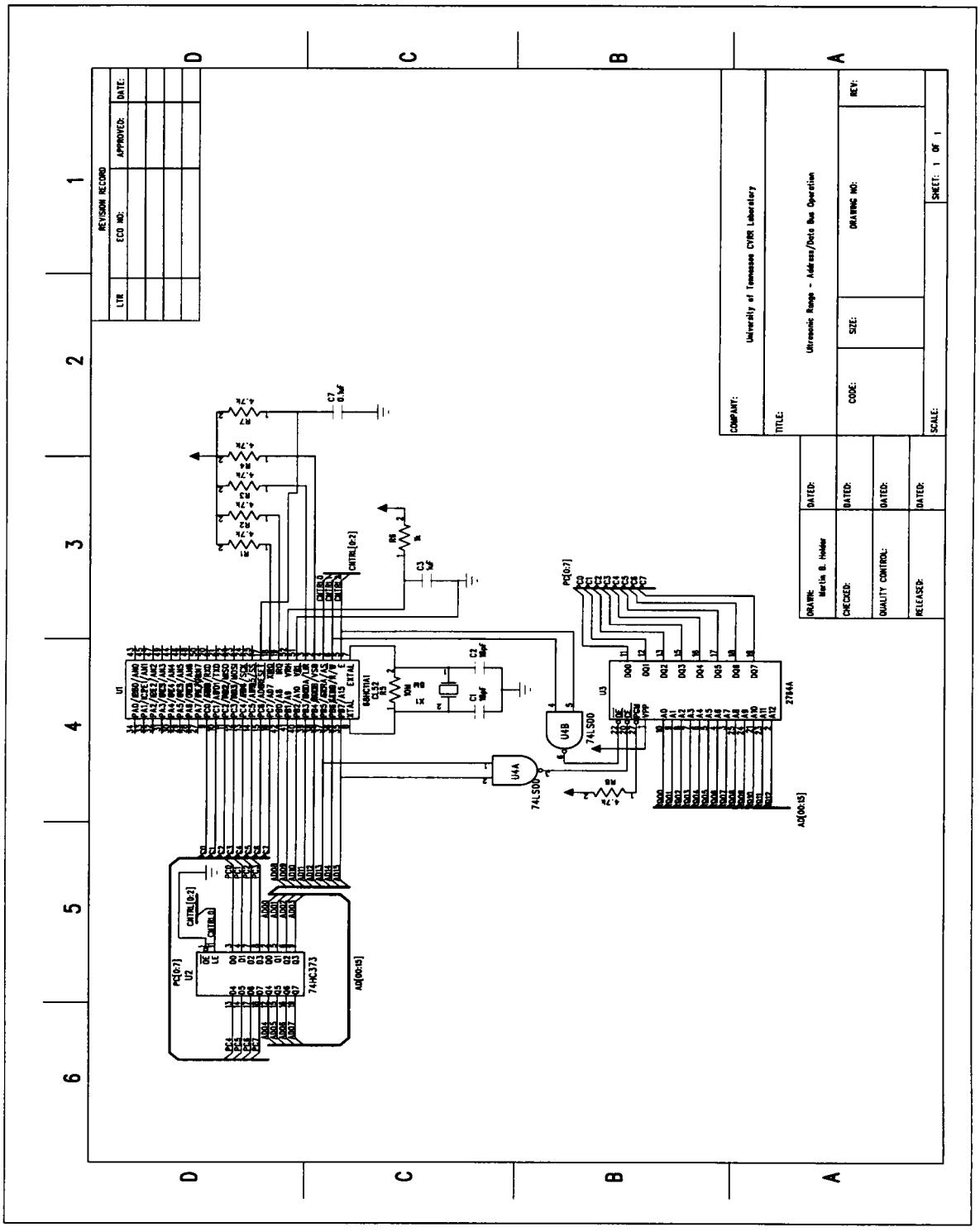

Figure 5.6: *Hardware control for microprocessor reads of programmed code stored on external EPROM.*

are enabled by a nand gate whenever the **R** (Read) and **E** (E-clock 2MHz) are asserted high. The chip enable is active low and is provided via nand gate of which the inputs are address lines **AD15** and **AD13**, which provides the chip to be enabled at hex address locations $A000-$BFFF or $E000-$FFFF. Motorola suggested this type of decoding in order for the reset vector to be fetched from the external EPROM whether the MCU (micro-controller unit) is operating in normal expanded mode or special test mode. Conflicts could arise if one used the EEPROM, which is mapped for $E000, internal to the MCU. The MCU has control registers which enable the internal EEPROM to be used in the memory map. Likewise, one can alter this location via control registers within the MCU, if used; however, since the EEPROM is not used, the potential problem is of no consequence. As a first time decoding scheme for such a system, this method is adopted as described by Motorola [Mot89].

5.2.3 Level Shifting for Serial Communications between Ultrasonic Subsystem and Host

The MCU is capable of sending and receiving serial transmissions via RS232 serial communications; however, the MCU operates on logic-level voltages as opposed to the common +/- 12 volt levels seen by typical RS232 communications. Additional hardware must be included between the MCU and external host system to enable level shifting capabilities for proper communications between the two systems. Maxim manufactures the MAX232 chip which performs the job of level conversion and requires a minimum number of external components [Max94]. The MAX232 operates on a single 5 volt supply and requires a total of four external capacitors for the on-board charge pump to operate properly. The MAX232 is capable of handling two pairs of serial lines of which

only one is used. Figure 5.7 shows the connections of this part in the system.

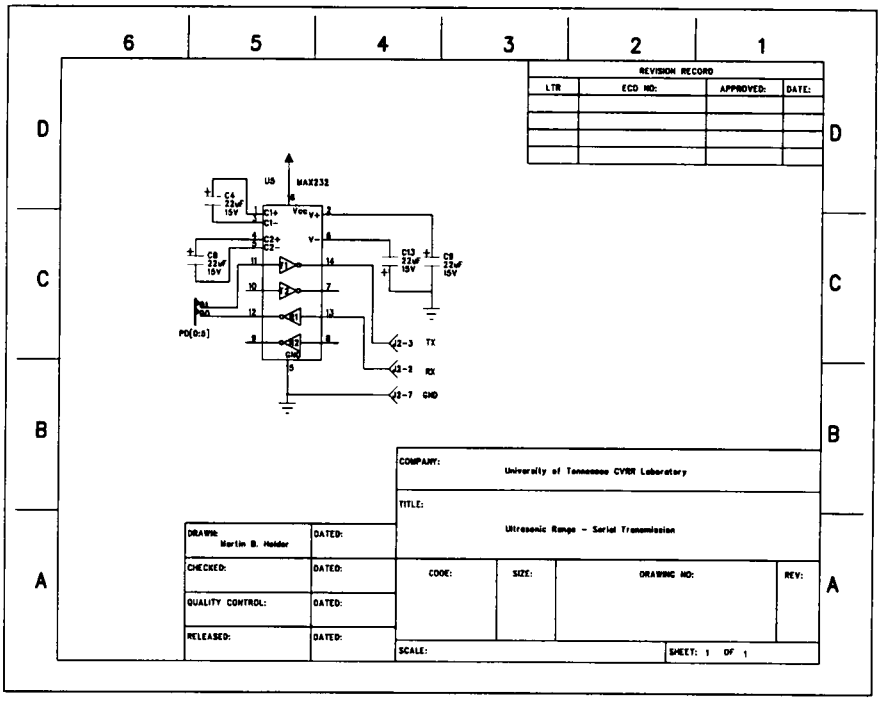

Figure 5.7: *MAX232 level shifter for converting +/-12 volt serial transmissions used by the host PC to corresponding 5 volt levels used by the ultrasonic subsystem's MCU.*

5.2.4 Motor Control for Panning Ultrasonic Transducer

A small 12.5 oz.in. stepper motor is used to rotate the ultrasonic transducer, and the MC3479 chip is used to drive the motor directly. The MC3479 is capable of directly driving stepper motors with a maximum current rating of 350mA per coil [Mot88]. This chip provides three important logic inputs: **STEP**, **CLOCK**, and **DIRECTION**. Internal logic keeps track of the current pulse sequence and for the given inputs, provides internal logic to generate the next correct pulse sequence. Figure 5.8 reveals the connections of the MC3479. All stepper motors operate via sequencing which details the manner in which the coils are given particular polarities for proper motor advancement.

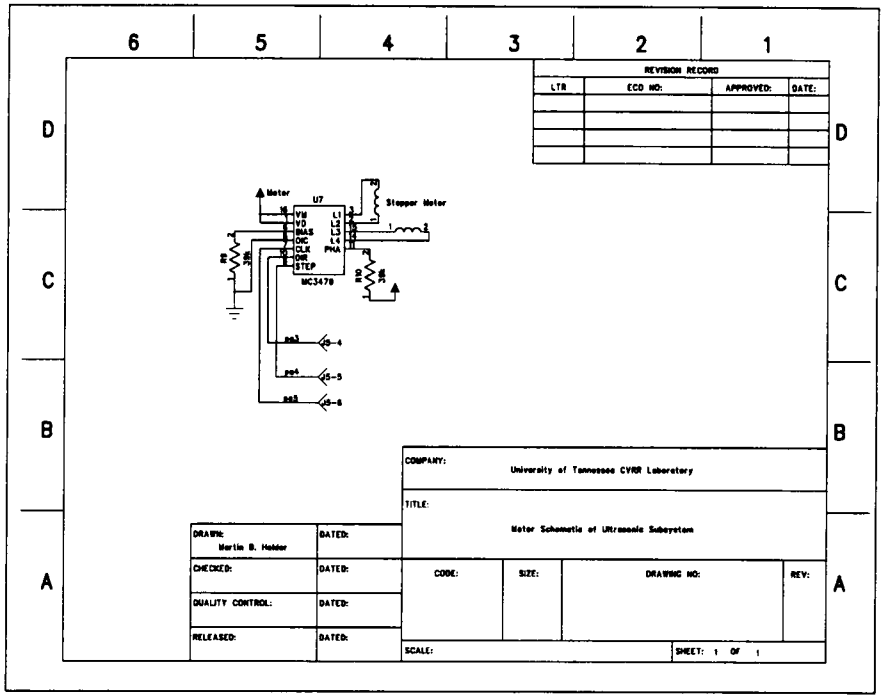

Figure 5.8: *Ultrasonic subsytem's MCU interface to the MC3479 stepper motor controller chip for panning the transducer.*

The MC3479 internally handles this process. Sequence parameters differ depending on the mode (full/half step) of the stepper motor. A different sequence exists for full step operation as opposed to half step operation. Stepper motors have a step resolution associated with them, which is described in degrees/pulse. The particular stepper motor used is capable of rotating 1.8°/pulse in full step operation and 0.9°/pulse in half step operation. Due to the light load of a single ultrasonic transducer, no problems associated with moving such a load by the stepper motor were noticed. Thus, the motor does not get out of phase for the given load and motions it undergoes. For larger loads, stepper motors typically need to be ramped to given speeds; and information regarding specifications of the motor can be found on the manufacturers data sheets. Speed-torque curves are typically given along with an accompanying drive circuit used to generate such graphs. Since current through the coils of a stepper motor is directly proportional to the output torque, elaborate drive methods are used which significantly improve the performance of the motors. The more popular stepper motor drive is the Chopper drive, which utilizes a motor supply voltage that is much larger than the voltage rating of the motor coils. Most speed-torque curves generated utilize such a method to show very good operational characteristics of the motor. Since motors have a large inductance associated with them, the larger supply voltage accelerates the rate at which the current through the coils reaches its rated value. Once the rated value is reached, the drive's circuitry maintains the rated current by turning the motor supply off/on in a repeated fashion for the duration of the current sequence. The SGS Thompson L297 chip supports the "chopper" capability.

The MC3479 chip does not support the Chopper drive method. A resistor, which provides the current limit on each coil of the motor, is placed from one of its pins to

ground. An additional pin which is connected to the logic supply for a 5 volt motor voltage, is used for a separate motor supply as opposed to the logic supply. Formulas are given in the specifications for determining the resistance value to limit the current in the coils with a given motor supply. One could easily provide a power saving feature for the chip by increasing this resistance value when the motor is not in use; however, this feature is not utilized in the current design. Another feature which is not used is the **PHA** open-collector output which is pulled high via **R10**. When the current stepper motor phase is located at the start of the sequence, the open-collector output will drive low.

5.2.5 Ultrasonic Range Module for Firing / Listening

Texas Instruments manufactures a sonar-ranging module which provides all necessary high voltages required to operate the transducer, digitally controlled variable gain amplifiers for detecting echoes, and onboard control logic [Tex83]. Connections are made available to the user for interfacing to the module, five of which are used in this design. Two of the connections are power and ground connections. Five volts should be applied to the V+ input as the source voltage. Due to the large demands of current required during the firing of the transducer, a 300μF capacitor is recommended to be placed on the supply inputs to the ranging module. The firing and listening of the transducer is done with the control bits in Figure 5.9. The **INIT** input bit, when asserted logic high, initiates the firing sequence. The **XDCR** is intended to show the firing of the transducer, which will be exposed to a burst of sixteen 49.4 kHz pulses. The amplitude of this pulse train is approximately three hundred volts. Should the **BINH** input not be used (tied low), the module will begin listening to the transducer 2.3 msec after the

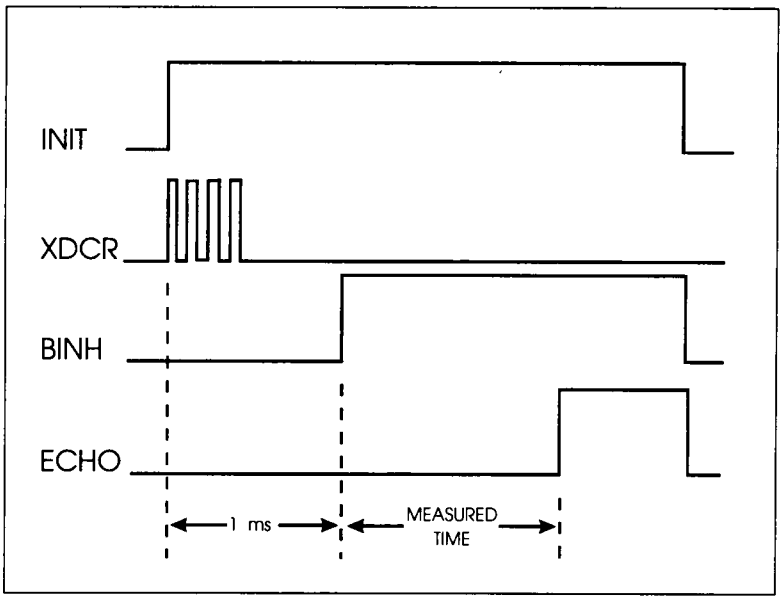

Figure 5.9: *Ultrasonic range module pin control/actions.*

firing of the transducer; and, allows for adequate damping of the ringing transducer after being exposed to firing. If it is used, the **BINH** input will allow the receiver to begin its listening prior to the 2.3 msec wait time, if and when **BINH** is asserted high. In the design, this input is set logic high 1.0 msec after the firing of the transducer. Thus, 1.0 msec corresponds to approximately 13 inches the sound has traveled before listening begins; and 6.5 inches corresponds to the minimum distance of detection. The transducer is held at 150 volts during the listening phase for sensitivity purposes. As time progresses, the range module automatically adjusts its gain in an increasing manner, which is necessary because the sound looses its amplitude over longer distances of flight. **ECHO** is an output from the module which asserts itself high upon detection of the sound, and marks the end of the time period for which the time differential is noted and distance of flight calculated.

5.2.6 Internal Power Distribution and Separation of Ultrasonic Subsystem

All voltage levels within the design are rated for five volts, so a single supply is capable of operating the entire system. An adjustable regulator is used to generate 5.6 volts. The supply should be greater than 8.6 volts for proper regulation giving a 3 volt differential across the regulator. Two portions of this completed design require careful consideration for noise spikes. Motors are inherently noisy devices and can cause large power disturbances, and the ultrasonic range module used for firing the transducer is capable of generating power surges which occur during the firing of the transducer. Both of these conditions are potentially dangerous when operating the MCU from the same power source. Figure 5.10 shows the suppression technique used to isolate such conditions from the MCU. Diodes are used to separate the individual power provided

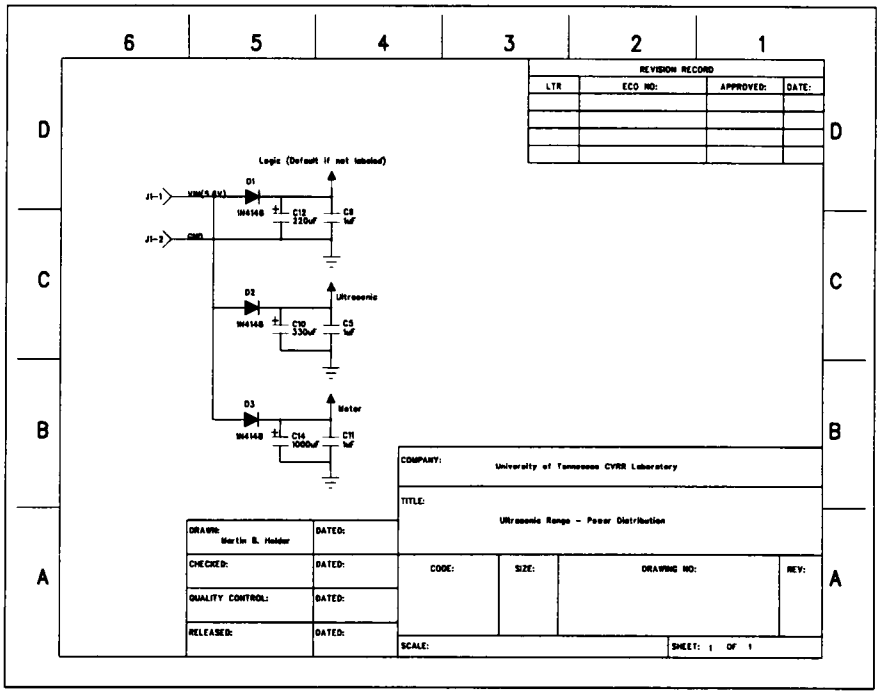

Figure 5.10: *Power distribution and separation specific to the ultrasonic subsystem.*

to the logic, motor, and ultrasonic range module. The 5.6 voltage regulation accommodates an approximate 0.6 volt drop across each of the diodes. Since current only flows when the diode is forward biased, other sources cannot *rob* the separated individual capacitors of their charge. Each source has its own noise filtering consisting of a large capacitor and smaller 0.1μF capacitor. The larger capacitors vary in size depending on the instantaneous current demands of the individual components it supplies. The motor supply is given a 1000μF capacitor, the ultrasonic board a 330μF, and the logic a 220μF. The smaller capacitors of 0.1μF in parallel with each of the larger capacitors have the ability to "snuff" higher frequencies better than larger capacitance values.

5.2.7 Concluding Hardware Remarks on Ultrasonic Subsystem

All pin connections not used on the MCU are tied high via pull-up resistors. The MCU is a CMOS device, and leaving an input floating could create device latch-up which consumes more power and generates noise from repeated switching. The reset pin is connected to a simple RC circuit to allow voltage levels to stabilize before allowing the MCU to begin accessing data from the eprom. Port **E**, an input port, is the only port not utilized. The full schematic of the Ultrasonic Transducer Subsystem is given in figure 5.11.

5.3 Assembly Language Firmware for Embedded Ultrasonic Control

Motorola provides an assembly language compiler **AS11** to be used with the **68HC11XX** microprocessors. The compiler will take assembly language mnemonics and translate them into an **S19** format which is then used by the microprocessor. The

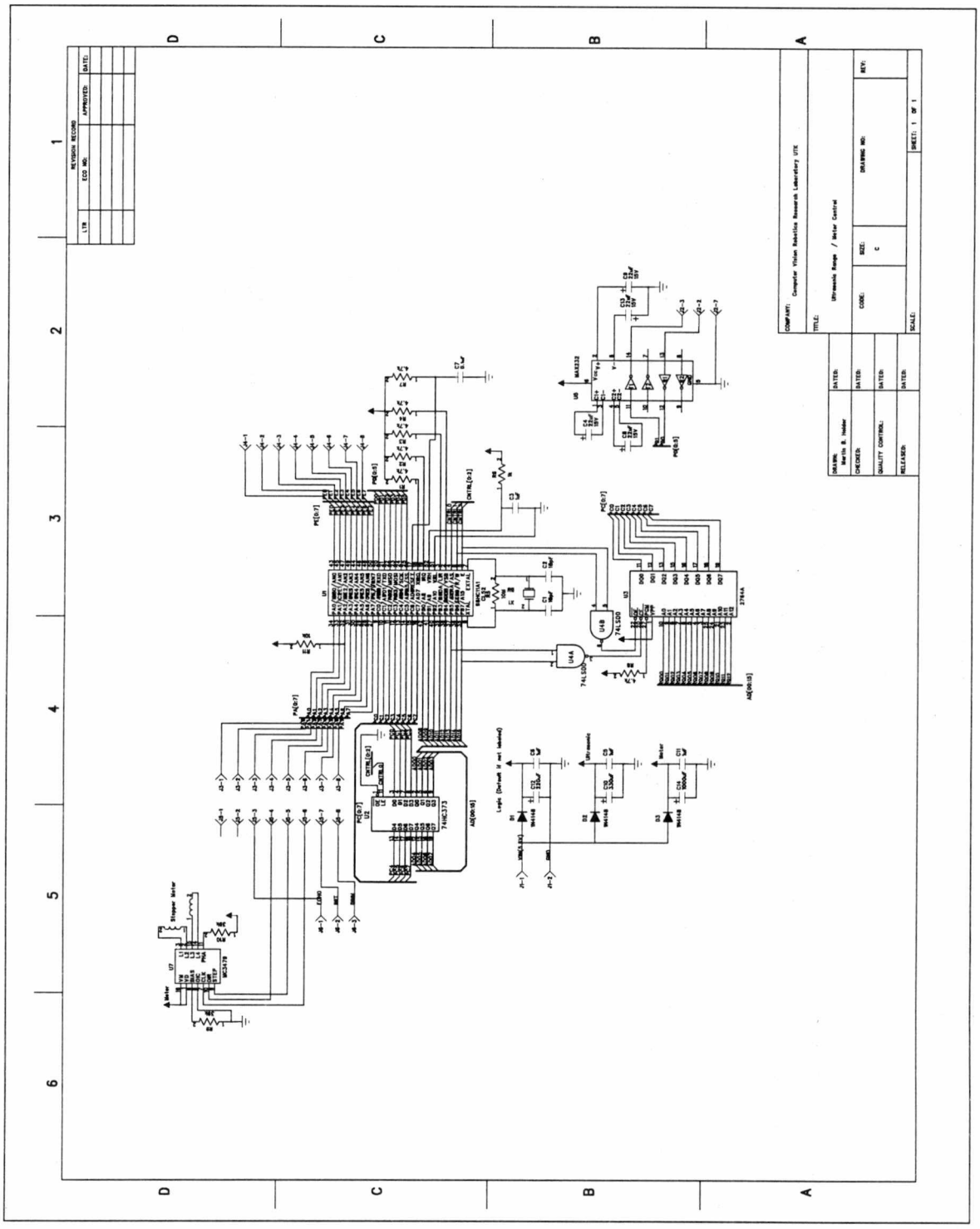

Figure 5.11: *Ultrasonic transducer subsystem dedicated hardware.*

assembly language code allows the user to place programmed code in certain memory locations using the **ORG** mnemonic. Using an external EPROM in normal expanded mode to be included with a single chip design requires the programmer to place data in specific locations on the EPROM. After a reset condition, once the mode of operation has been established, the processors first step is to gather a two byte starting address for program execution to be placed starting at location $FFFE on the EPROM. Due to the decoding scheme, the EPROM is mapped twice in memory for both $A000-$BFFF and $E000-$FFFF addresses. The starting address of $A000 is placed into EPROM location $BFEF. Although the MCU will place $FFFE on its address bus in search of the starting address, the EPROM will still respond due to its dual memory mapping. The jump locations for given interrupts are located in EPROM, starting at memory location $BFD6. These addresses must contain the memory locations for their corresponding interrupt service routines, and each interrupt has its own jump location in EPROM. Three interrupts are used, and the memory addresses for these routines are placed in the EPROM at their respective locations. The unused interrupts are given jump locations of $A000 which will jump the processor to the beginning of the program. These addresses should never be used because the interrupts associated with them are not in use. As a safety measure, they are given addresses which will re-start the firmware.

5.3.1 Selected Ultrasonic MCU Control Pins / Actions

The MCU is programmed to utilize a total of three timer interrupts for firing of and listening to the transducer on the range module. **TOC3** (timer output compare three) is used to generate an interrupt when the free-running sixteen-bit counter of the MCU reaches zero, and is activated only when a firing sequence is in progress. Prior to acti-

vating such a sequence, the variable **OVERFLOW** is set to zero. Each time the free running counter reads zero, the **TOC3** service routine is entered; and **OVERFLOW** is incremented. Within this interrupt service routine, the MCU sets **PA6** high if **OVERFLOW** is equal to one indicating the first time the routine had been entered for the given firing sequence. The **PA6** pin is connected to the **INIT** pin on the sonar range module and will fire the transducer. The service routine for **TOC3** enables **TOC2** which is set to occur 1 ms later; and the **TOC2** service routine sets **PA7** high which is mapped to **BINH** on the range module, which allows the range module to listen for echoes 1 msec after firing as opposed to the 2.3 msec standard by the range module. The 1 msec time delay is adequate to allow for the transducer to settle from ringing prior to listening, and allows for obstacles to be detected at closer distances. The **TOC2** service routine sets up the **TIC1** (timer input capture one) interrupt to look for a detection indication from the range module. The programmer has the capability of selecting when the interrupt will occur based on four features. The interrupt can occur on a high or low level basis, high to low transition, or a low to high transition. **TIC1** is associated with **PA2** on the MCU. **PA2** is an input pin which is connected to the **ECHO** output pin on the range module. The range module, upon detecting the sound wave, will set **ECHO** high, which requires **TIC1** to be set for low to high transitions. Once this service routine is entered, the free-running counter's current value is stored, which represents the sixteen-bit stopping time from the counter. Should the range module not set its **ECHO** line high, as with the case of the module not detecting the sound, the system will time-out; and all bytes returned to the host will be given values of $FF. The time-out in such a case is based on the current count of the **OVERFLOW** variable. If **OVERFLOW** reaches a value of three (indicating distances greater than 35 feet), a

time-out condition occurs. When either a time-out condition exists or a valid detection occurrs, all interrupts are turned off; and the proper bytes are returned to the host system.

To control the stepper motor, the MCU uses three bits: **PA3**, **PA4**, and **PA5**, which are output bits that are connected directly to the MC3479 stepper motor controller chip. **PA3** controls $\overline{cw}$/ccw direction, **PA4** $\overline{full}$/half step operation; and **PA5** is used to pulse the motor. The direction and step imposed on the motor are simply level conditions in which the outputs from the MCU are held at their given values. When pulsing the stepper motor, the MC3479 will change to the next sequence whenever a low to high transition on its **CLK** input occurs. For this reason, **PA5** is brought high and then low to generate such a transition when requested by the host system.

5.3.2 Ultrasonic Serial Protocol / Command Set

All serial transmissions received by the MCU are polled in software, and the MCU utilizes its **SCI** (Serial Communications Interface) for such transmissions. Interrupts are not necessary for the serial system. All transmissions are set for no parity, eight bits, and one stop bit, and a protocol is established for serial communications between the onboard MCU and a connected host computer system. The overall system is setup as a master-slave system. The MCU will react based only on the commands sent by the host computer. At no time will the subsystem send bytes to the host system without the host system's knowledge of such incoming data. The protocol begins with the host computer system sending a start of transmission byte, ASCII 'E'. The next byte sent is the command byte. A total of seven command bytes are used, and since the command set is short, commands are represented by ASCII '1' through '7'. The last value sent

by the host system is the checksum value which is a single, unsigned eight-bit value. The checksum value is the unsigned ASCII addition of 'E' and the command byte. If properly sent, the command will be serviced by the MCU, which will return data if appropriate along with a checksum. The **SYNC** command is a special case and exception to this rule. For commands with no data to be sent back, the MCU will send the received checksum, if valid. Table 5.1 lists the available serial command options.

Table 5.1: *Ultrasonic Serial Command Set.*

Command	Meaning
'1'	CW Direction
'2'	CCW Direction
'3'	Full Step
'4'	Half Step
'5'	Pulse Motor
'6'	Fire Transducer
'7'	System Sync

CW Direction ——————————————————— **'E' '1' 110**

This command will set the stepper motor for the clockwise direction. The only return byte will be the checksum **110** if the command is successful.

CCW Direction ——————————————————— **'E' '2' 111**

This command will set the stepper motor for the counter-clockwise direction. The only return byte will be the checksum **111** if the command is successful.

Full Step ——— **'E' '3' 112**

This command will set the stepper motor for full step operation. This corresponds to a total of 200 pulses per revolution, which yields 1.8°/step. The only return byte will be the checksum **112** if the command is successful.

Half Step ——— **'E' '4' 113**

This command will set the stepper motor for half step operation. This corresponds to a total of 400 pulses per revolution, which yields 0.9°/step. The only return byte will be the checksum **113** if the command is successful.

Pulse Motor ——— **'E' '5' 114**

This command will pulse the stepper motor a single pulse under its current setup, which entails current step type and direction. The only return byte will be the checksum **114** if the command is successful.

Fire Transducer ——— **'E' '6' 115**

This command fires the transducer. Upon reception of the returning sound wave or time-out condition, three bytes and a checksum byte will be returned. The firing of the transducer will start at the beginning of the MCU's onboard free running sixteen bit counter. The counter has a resolution of 0.5 μsec for its given 2MHz bus frequency. The first byte returned will indicate the number of times the counter overflowed. The second and third byte will indicate the stopping time of the sixteen bit counter when the sound was received. The high byte is transmitted first, and is followed by the low byte. Should

the system time-out, indicating no sound wave was detected, all three returned bytes will contain $FF. The host system is responsible for calculating the obstacle's distance given the returned bytes. The checksum byte is the last byte transmitted, and its value is the unsigned eight-bit addition of its current value, **115**, with the three returned decimal bytes.

System Sync _____ **'E' '7' 116**

Upon initial power up of the subsystem, one should flush any existing data within the microprocessors transmit buffer. This was a noted problem during the design of this subsystem. Should the subsystem power up in the proper state, no problems are noticed; however, at times, the system comes up in an odd state. The erroneous state is most likely due to an overlooked initialization method, and, in this odd state, the MCU is capable of noticing incoming serial commands, yet lags by one data byte when returning data to the host system. A solution to such a problem is to read an additional byte from the MCU for synchronization purposes. Once this is done, no syncing problems will be noticed thereafter. To verify the MCU is in synchronization, the system sync command will return four bytes of decimal values 1, 2, 3, and 4 and send no checksum. If the bytes returned are not in order, the host computer should read an additional byte, although this byte is of no use to the host system other than to properly synchronize the MCU with the host. Should the MCU need to be synchronized, the host should verify proper synchronization by once again sending the MCU the system sync command. Once verified, proper operation of the MCU in response to the host computer system will be observed. Synchronization is only needed as a first time power up procedure. A checksum must be given to the MCU to validate the system sync command, yet only

four return bytes are returned to the host system without an accompanying checksum.

5.4 Supplement C Programming Routines Specific to the Ultrasonic Subsystem

C routines were generated to control all functions of the Ultrasonic Transducer Subsystem. The function calls handle all serial communications for the host system to communicate with the subsystems MCU. All routines likewise make the proper connections of the host system to the subsystem via a hardware serial multiplexer. The routines are passed a structure pointer of type ROBOT found in *robot.h* which contains variables pertaining to the subsystem. The variables, along with their descriptions, can be found in Table 5.2. The variables are updated each time a routine is called and completes successfully. Some of the routines rely on the use of angles for the transducer. The front of the robot is considered to be 180°. Directly to the right and left are 90° and 270° respectively. Such a coordinate system is adopted for ease of calculations when rotating the robot's transducer through the various angles. The routines will not

Table 5.2: *ROBOT Structure Variables for Sonar Subsystem.*

Variable	Description
DIR	Current Motor Direction
STEP	Current Motor Step
POS	Current Motor angle
VALUE[365]	Array of Distance Values at Given Angle
LAST_VAL	Last Fire Value
PORT	Multiplexer Port Value

allow an overlap of angles. If the current transducer position is located at 60° (right), to reach 290° (left), the transducer will be turned counter-clockwise passing through 180° (front) to reach the desired angle. This is the longer route of rotation to reach such an angle; however, the rotation, prevents the wires connected to the transducer on the motor from getting tangled. Routines which have return values associated with them return zero upon successful operation or one for failure. The routines available are as follows.

int son_sync(ROBOT *rob)

This command is used to synchronize the subsystem with the host computer, and the function should be called upon when an initial power up condition of the subsystem exists to verify proper operation of MCU's serial transmissions to the host system. Should the subsystem be found in an odd state, this routine will attempt to correct such conditions by reading an additional byte and then once again verifying proper synchronization. Should the subsystem still prove invalid, the routine will return 1 indicating an error. If proper synchronization is noted or corrected, the routine will return 0. Refer to SYSTEM SYNC serial command of the MCU for further explanation.

int son_reset(ROBOT *rob)

This command is used to reset the subsystem. The purpose of such a routine is to fix the location of the transducer to be directly in front of the robot (180°). Such a routine is useful for first time power up purposes to re-orient the transducer to a known starting position. The transducer will begin a successive fire and pulse sequence, rotating the motor clockwise until the transducer reads values less than one foot. The robot has a

68HC11EVB located behind the ultrasonic transducer. This board is the key point in resetting the subsystem. Once the board is located, a series of confirmations are performed to discard possible false readings from the transducer. If confirmation is valid, the routine will then return the transducer to point directly in the front of the robot. If confirmation proved to be invalid, the fire and pulse sequence is again initiated until a valid confirmation exist. **POS** variable is reset to 180.0 in the process.

int son_cwdir(ROBOT *rob)

The function is used to set the transducers stepper motor for clockwise direction. The **DIR** variable is set to 1 if successful.

int son_ccwdir(ROBOT *rob)

The function is used to set the transducers stepper motor for counter-clockwise direction. The **DIR** variable is set to -1 if successful.

int son_fullst(ROBOT *rob)

The function sets the transducers stepper motor for full step operation. Full step operation provides 200 pulses/revolution which yields 1.8°/pulse. The **STEP** variable is set to 1.0 if successful.

int son_halfst(ROBOT *rob)

The function sets the transducers stepper motor for half step operation. Full step operation provides 400 pulses/revolution which yields 0.9°/pulse. The **STEP** variable is set to 0.5 if successful.

int son_pulse(ROBOT *rob)

The function pulses the stepper motor in its current state. The **POS** variable is altered according to current step operation and direction.

int son_fire(ROBOT *rob, float *)

The function fires the transducer and calculates the distance value in feet. This value is then placed into the memory location pointed to by a float pointer passed as an argument to the routine. A maximum distance of 35 feet exists. Should the transducer not hear a returned sound wave, the routine will set the distance to 35 feet returning this value. The **VALUE[365]** array is altered according to current transducer angle position and calculated distance. The **LAST_VAL** variable is stored with calculated distance.

void clr_scan_values(ROBOT *rob)

The routine is used to clear all current distance values within the ROBOT structure array **SONAR_VAL[365]**. Values of 35 feet are placed into each member of the array.

void sonar_control_center(ROBOT *rob)

The sonar control center is a menu based control for all sonar functions. The user has complete control and selection of calling all routines previously described as seen in Figure 5.12. Optional graphics can be displayed for scanning of the transducer. The scan angle and maximum scan distance is selectable. Software delays and multiple firing capabilities are likewise selectable. The multiple firing capability fires the transducer a

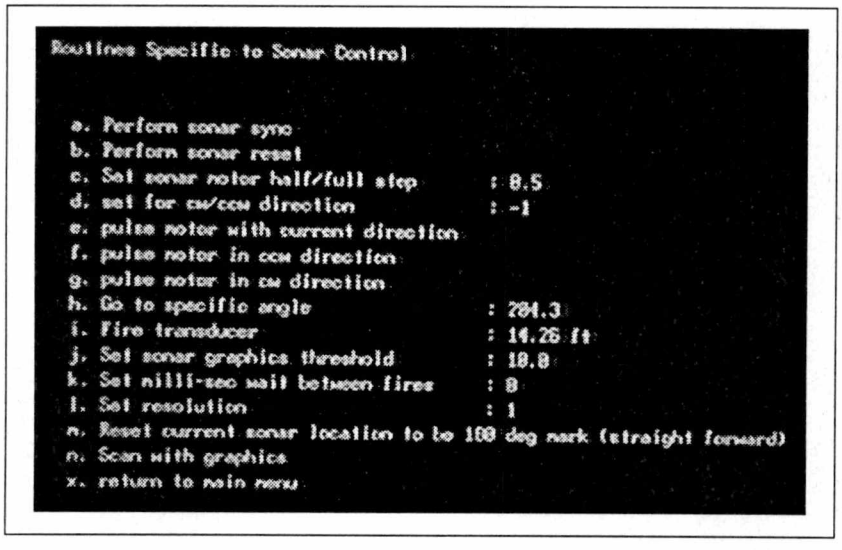

Figure 5.12: *Host system's menu control exploiting function calls specific to the ultrasonic subsystem.*

selectable number of times in each current position of the motor, averaging the distance values to form one averaged value which is then displayed. The motors current step value plays a vital role in the scanning process. Motor movements will be based on the motors current step value. Another option to such scanning is in choosing the resolution of the scanning process, which is a variable located in the menu and represents the number of times the motor will be pulsed before firing the transducer. The default is one indicating the highest resolution possible for the given motor step. Increasing the value increases the individual angle movements between successive fires, thus lowering the resolution which yields faster scanning. Figure 5.13 shows the graphical output for the scanner system.

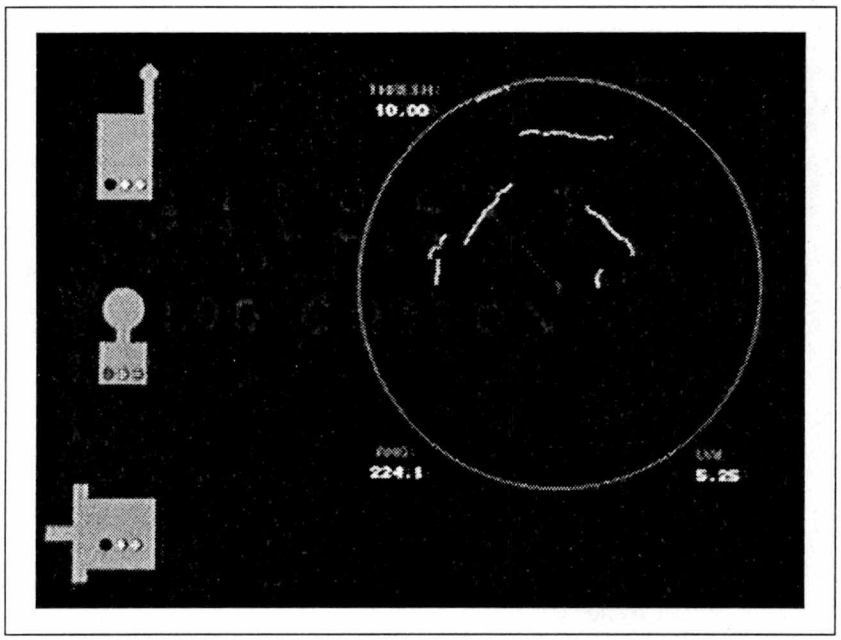

Figure 5.13: *Graphical display of scanned ultrasonic range data.*

5.5 Calibration and Testing of Ultrasonic Subsystem

The purpose of the MCU and its associated hardware for ultrasonic readings is to fire the transducer and return a time measurement for the distance of flight. The host system then computes the distance based on the timer information it receives. Sound travels an approximate 1100 ft/sec with deviations based on environmental conditions such as temperature, humidity, etc which can introduce error into the measurement. Another source of error can be the sensitivity adjustment (potentiometer) on the ultrasonic module for the receiver portion of its circuitry. Computer software was written to find gain and offset adjustments to minimize the error arising from such conditions. Two readings at distances of 3 and 10 feet were taken for calibration purposes. The timing information regarding these distances was received by the MCU and used in

the calibration process which alters the gain from 0.9 to 1.1 in 0.01 increments while making distance calculations with offsets of -500 to 500 timer counter ticks. By altering the gain and offset in this fashion, all possible combinations for gains and offsets within their given ranges are covered. The error measurements, based on each variable gain and offset, are found and a minimal error for the 3 and 10 feet calculations is observed. Once the minimal error was found, its gain and offset are permanently used in determining the calculated distance.

The ultrasonic module claims to be accurate to within 1/100 of an inch assuming head-on measurements of obstacles without reflections or other types of sources to impose additional error. Two significant methods of testing the overall subsystem are: Accuracy (readings taken over a range of distances), and Repeatability (successive readings for a given distance). The subsystem is capable of accurately providing the necessary timing information for the distance calculations to be made. The accuracy to which this occurs is indistinguishable from the naked eye. Without additional precision hardware to provide a basis for the given distance measurements, the test would more likely show the human error involved in setting the correct distance before the measurement is made. The repeatability test is a valid measurement which introduces no human error. The transducer is placed at approximate 1 foot increments and a total of 10 readings are taken without moving the transducer nor its detecting obstacle which was chosen to be a large wooden block (3ft x 3ft x 3ft) placed perpendicular to the transducer. Table *5.3* shows the results of the repeatability test.

Table 5.3: *Ultrasonic Repeatability Test.*

Distance (ft)	*#1*	*#2*	*#3*	*#4*	*#5*	*#6*	*#7*	*#8*	*#9*	*#10*	*Delta*
1.0	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	0.00
2.0	2.00	2.00	1.99	1.99	1.99	2.00	1.99	1.99	1.99	1.99	0.01
3.0	2.99	2.99	2.99	2.99	2.99	2.99	2.99	2.99	2.99	2.99	0.00
4.0	3.99	3.99	3.99	3.99	3.99	3.99	3.99	3.99	3.99	3.99	0.00
5.0	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	0.00
6.0	5.98	5.97	5.97	5.98	5.98	5.97	5.97	5.98	5.97	5.97	0.01
7.0	7.00	6.99	7.01	7.02	7.00	6.99	7.00	7.01	7.02	7.02	0.03
8.0	8.07	8.07	8.07	8.07	8.07	8.07	8.07	8.07	8.07	8.07	0.00
9.0	9.03	9.03	9.03	9.02	9.02	9.03	9.03	9.03	9.03	9.03	0.01
10.0	10.00	9.97	9.98	9.97	10.00	9.99	10.01	10.00	9.99	10.01	0.04

CHAPTER 6

Stepper Motor Control Subsystem

A subsystem dedicated to controlling two stepper motors was developed for the mobility of the robot. The subsystem uses a 68HC11EVB to provide the necessary low level control functions which then interfaces to a custom designed PC board for handling high current requirements of the motors. The subsystem is sent RS232 serial commands dictating the type of control to be maintained by the subsystem. Three modes of motor operation are made available by the subsystem. The hardware / software issues for such control is further discussed along with existing problems. Supplement C routines were written to enable high level programming control. A menu driven software control system capable of accessing all commands handled by the subsystem was generated.

6.1 Hardware Interface for Stepper Motor Control

A PC board which was designed and built for the stepper motor control subsystem. The hardware is capable of driving two independent stepper motors. The MC3479 stepper motor controller chips are included to provide the proper sequence of steps necessary to control the stepper motors as well as direction and step type used. Control logic includes the ability to turn the motors on or off. H-drive configurations consisting of mosfet transistors are used to switch the currents within each coil of the motors. A voltage doubler is used to exceed the necessary threshold voltage to the mosfets when turning each mosfet on. A Reed relay is used to provide protection for the circuit when

applying motor voltage without applying logic voltage. Figure 6.1 shows the PC board for driving stepper motors.

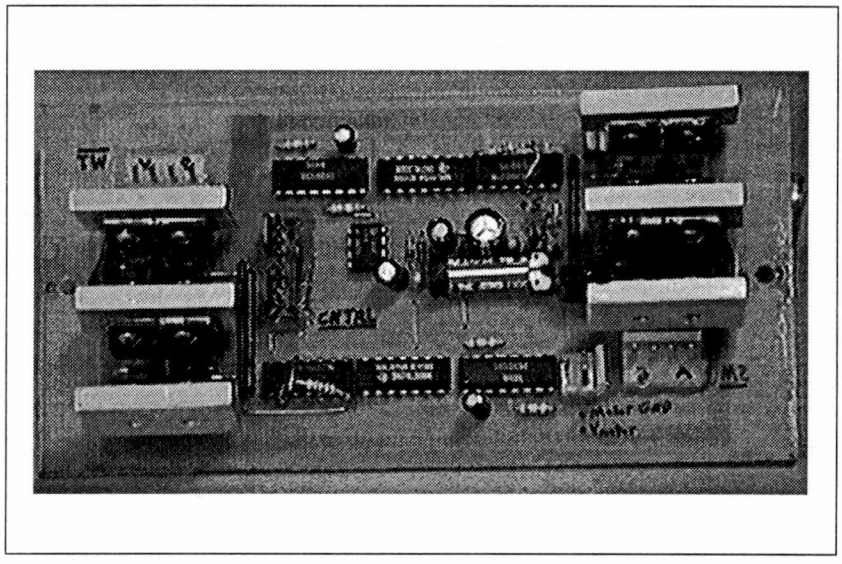

Figure 6.1: *Completed Motor Control PC Board designed to be externally controlled and provide drive capabilities for two high power stepper motors.*

6.1.1 MC3479 Stepper Motor Controller Chip

The MC3479 stepper motor controller chip is used to provide the necessary sequence of steps to the motors causing them to rotate. The chip has 3 primary input pins with which to control the stepper motor. A **DIR** input is provided for clockwise or counter-clockwise rotations. The **STEP** input provides the ability for the motor to be placed in full step or half step mode of operation. All stepper motors are specified with *degrees/step*; and, in full step operation, this specified value is true when pulsing the motors. For half step operation, the *degrees/step* value is divided by two, thus enabling a more fine control of the motors. The **CLK** input, when a low to high transition occurs, will pulse the motor a single pulse in its current setup configuration. The above three

inputs are to be provided by some external source, and each motor includes three such inputs. The MC3479 chips are capable of directly driving stepper motors with currents up to 350mA/coil. For larger motors, additional circuitry is necessary to provide the larger current requirements. The MC3479 chip can not handle the rated currents of the stepper motors used in this design (1.2 amps/coil), and merely served as logic levels which are further used in the overall control scheme.

All outputs of the MC3479 for direct coil connections were sent as logic level outputs to the inputs of "AND" gates. Each additional input of the "AND" gate consists of a control input ENABLE/DISABLE. The "AND" gate provides the ability to turn the motors on or off, and an input to the board is provided to externally control such a feature.

6.1.2 H-drive Configuration for Motor Control

For the switching of the currents within each coil of the stepper motors, an H-drive configuration is used. Each stepper motor has two coils associated with it, and Figure 6.2 shows this type of configuration for excitation of a single coil. This configuration serves as an electronic switch, which provides current flow of either polarity to flow through the coil depending on which transistor pair is turned on. Transistor pair Q2 and Q3 operate together as do Q1 and Q4. At no point in time should both pairs be on. Thus, input1 should always be opposite of input2. Should current flow in the direction of flow1, input1 should be asserted high, turning on Q2 and Q3. Input2 should be low, thus turning off Q1 and Q4. The reverse is true for current flow in the direction of flow2.

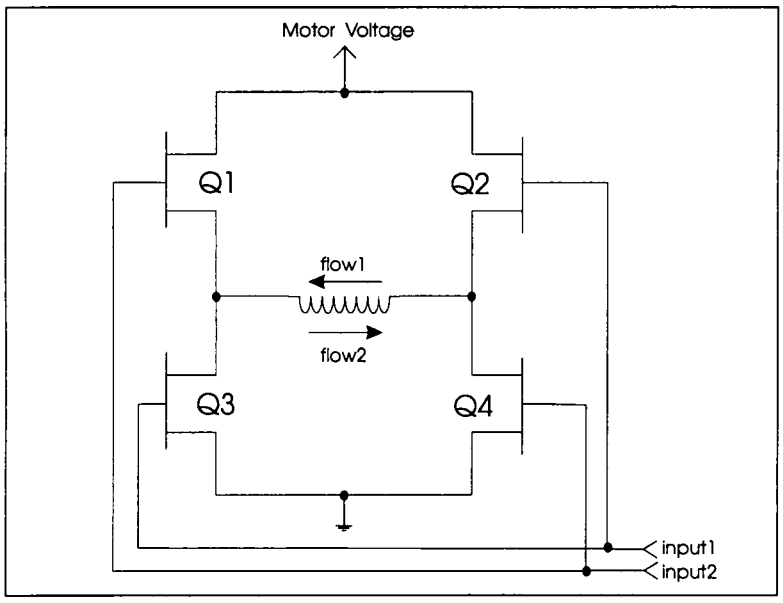

Figure 6.2: *H-drive configuration for the switching of current polarities within the coil of a stepper motor.*

6.1.3 Mosfet vs. Bipolar Transistors for Motor Drive

The use of mosfet transistors within the H-drive configuration provides for a robust and durable system. Mosfet transistors are voltage controlled devices with an extremely large input resistance. Bipolar transistors are current controlled devices whose input current and **Beta** value dictate the current to be supplied to some device. Mosfets employ a negative temperature coefficient enabling them to resist the flow of current as heat builds within the chip. Bipolar transistors have a positive temperature coefficient; and, when heated, they allow more current to flow until a possible *thermal runaway* condition exists [Sav91]. In such a condition, the chip can become severely damaged due to excessive heat. Mosfets typically have lower voltage drops across them in comparison to bipolar transistors. This provides for less power dissipated in the device and more to be applied towards the coils of the motor (more efficient). Since the mosfets are used as

simple "on"/"off" electronic switches, it is necessary to surpass a threshold voltage to turn such devices completely on. V_{GS}, the gate to source voltage of the device, is key for turning on the device. The higher the V_{GS}, the *harder* the device will turn on, and less voltage drop and power consumption across the device will be seen as a result.

6.1.4 Voltage doubler for Controlling Mosfets

A voltage doubler is utilized to turn on the Mosfet transistors within the H-drive configuration. The voltage doubler is generated by the **ICL7662** chip from the Harris Semiconductor Corporation and can serve as either a voltage doubler or negative converter depending on the connections made and parts used. As a voltage doubler, it is necessary to provide four external components, two capacitors and two diodes. The output voltage of the doubler is calculated to be twice the input voltage minus twice the forward voltage drop across the diodes. The motor supply voltage of 12 volts is used as the input to the device, and silicon transistors with an approximate forward voltage drop of 0.6 volts are used. The output of the voltage doubler is therefore computed to be 22.8 volts. To enable the logic chips to interface to such a large voltage, it is necessary to utilize open collector non-inverting buffers with high output voltage capabilities. The **7807** TTL chip does this job nicely, and the part is capable of handling voltages up to 30 volts. A sim resistor pack consisting of 10K resistors, with one side of each internal resistor connected to a common point, are used as pull-up resistors; and the common point is connected directly to the output of the voltage doubler. Each resistor is tied to the open collector outputs of the **7807**. The **7807**, when given logic low inputs, will drive the output low by providing a path to ground. Each output will thus sink approximately 2.8 mA of current when a logic low input is present. For logic inputs

which are asserted high, the output of the **7807** is considered an open-circuit. Since the gate resistance of the mosfet is extremely high, practically no current flow exists thus no voltage drop is developed across the pull-up resistor. The complete 22.8 volts is available as the gate voltage for switching on the given mosfet pair. Figure 6.3 shows the overall hardware scheme.

6.1.5 Purpose of Reed Relay

Once the hardware design was completed, a problem was detected. Two voltage sources are necessary for the hardware to operate properly. The hardware requires logic voltage (5 volts) and a separate motor voltage (12 volts). Should the motor voltage be applied to the hardware without the provision of the logic voltage, a catastrophic effect can take place because all gates to each mosfet will be driven to 22.8 volts. The voltage doubler would be operational from the motor voltage providing the 22.8 volts yet the resistors will not be properly sunk to ground since the **7807** TTL chip would not function due to the absence of its voltage source and thus all mosfet transistors will be turned on. In such a condition, the motors supply voltage is virtually grounded by the mosfets. The mosfets would have to endure an enormous amount of current and would more-than-likely be destroyed in the process. To prevent such a condition, the use of a Reed relay provides adequate protection. The Reed relay closes when logic voltage is present providing a connection of the motor voltage to the voltage doubler circuitry. Should the logic voltage not be present, the voltage doubler will not receive an input voltage and no voltage will be present on the gates of the mosfets to turn them on. Figure 6.4 shows the full schematic of the stepper motor control subsystem.

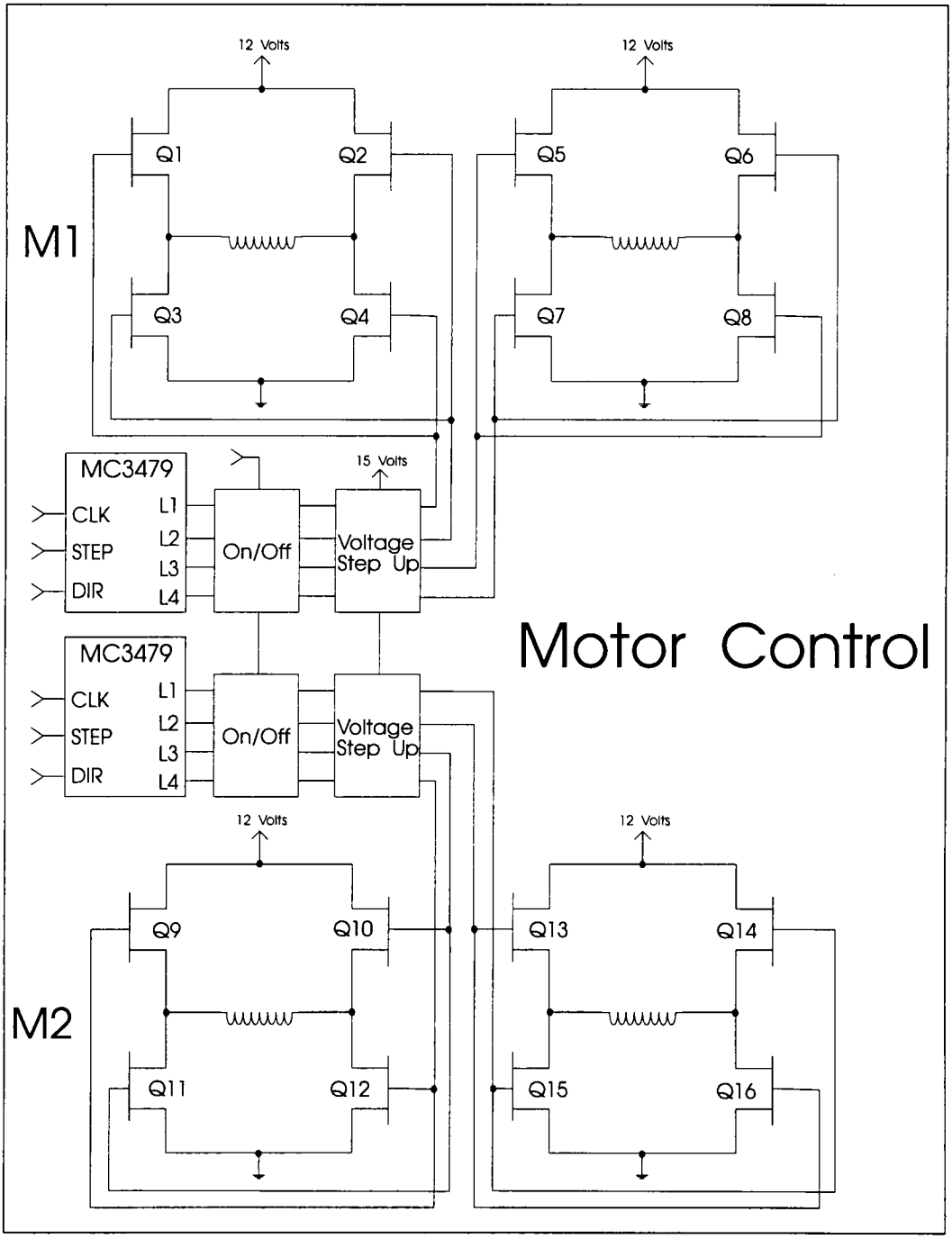

Figure 6.3: *Hardware control scheme for driving two stepper motors.*

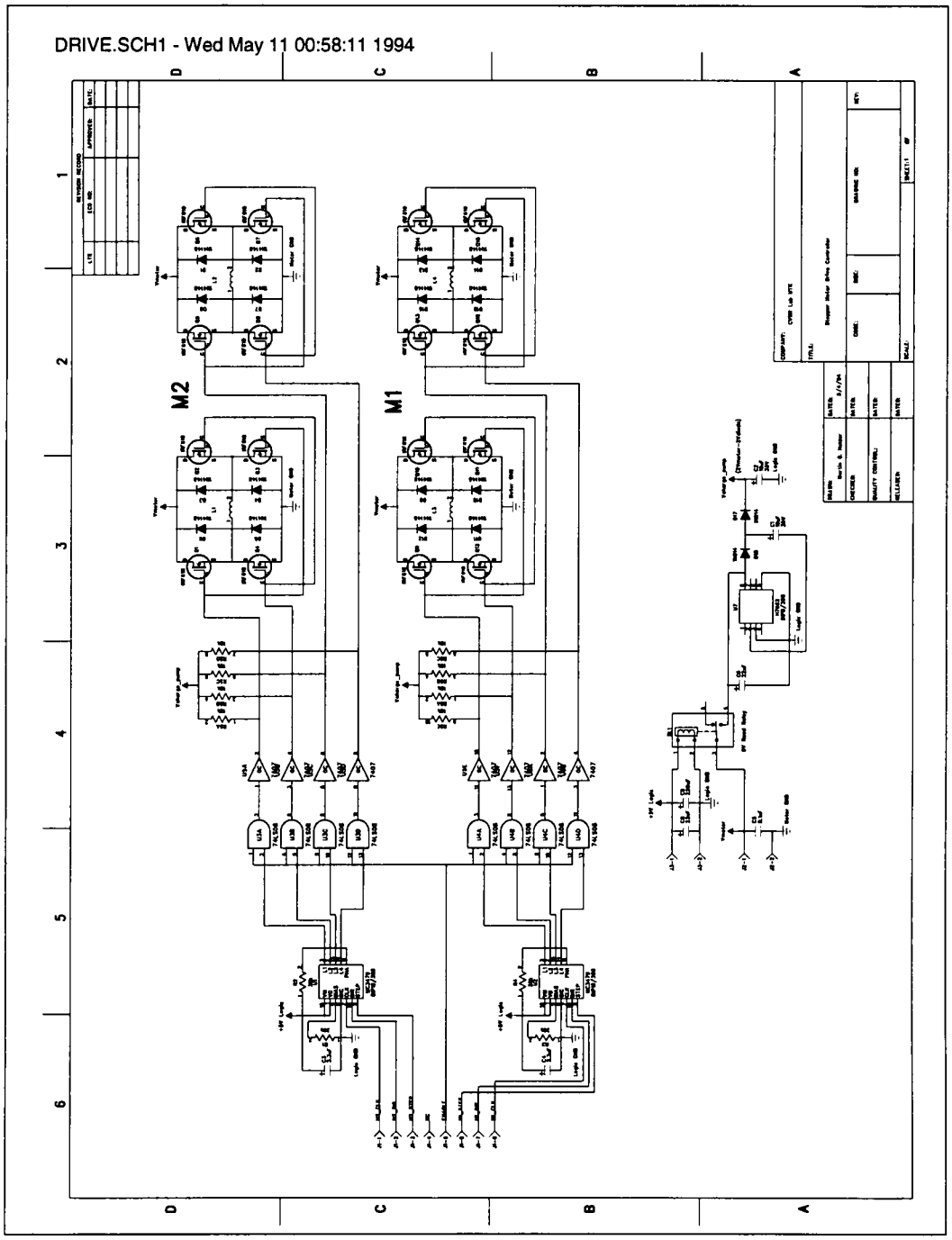

Figure 6.4: *Hardware design for the Stepper Motor Control Subsystem.*

6.2 Assembly Language Programming Specific to Motor Control

A 68HC11EVB provides the necessary control logic for the majority of activities associated with the motor control subsystem. The code was written in Motorola assembly language and placed onto an 8K EPROM. The EPROM was then placed within an available slot on the EVB. Simple jumper connections were configured for program execution to immediately begin with the EPROM code upon power-up.

6.2.1 Motor Control MCU Pin Selection / Control

The MCU is programmed to use a total of three timer interrupts to control all necessary timer related functions within the subsystem. **TOC2** (timer output compare two) is used to generate an interrupt for the pulsing of motor one (right), and **TOC3** generates an interrupt for the pulsing of motor two (left). Sixteen-bit timer registers for the **TOC** interrupts exist. When a particular output compare interrupt is activated, an interrupt is generated when the contents of the **TOC** register match the free-running sixteen bit timer of the MCU. Optional pin actions (programmable within internal registers) can be performed upon a successful compare. Four possibilities of pin action exist: low to high transitions, high to low transitions, toggle, and no action. When a match between the **TOC** value and the free-running counter occurs, an interrupt with no pin action will be made to the 68HC11. The pulsing of the motors are handled within the interrupt's service routine, and the **RTI** (real time interrupt) is used to note time-out conditions for serial transmission and to provide for the accelerating and decelerating of the motors. *Port A* contains the pins associated with the timer related functions. **PA6**, in association with **TOC2**, is connected to the **CLK** input for pulsing motor one. **PA5**,

in association with **TOC3**, is connected to the **CLK** input for pulsing motor two. *Port B* is an eight bit output port and services the **DIRECTION**, **STEP**, and **ENABLE** inputs on the interfacing hardware for both motors.

The interrupts are used according to the mode of operation of the subsystem. Three such modes exist: user-controlled mode, continuous mode, and incremental mode, and all modes of operation require the host system to send serial commands for the changing of direction, step type, and enabling or disabling of the motors.

The user-controlled mode does not require the use of the **TOC** interrupts. The host system will pulse the motors through serial commands sent to the subsystem, and this mode is the default upon initial power up. The RTI is still used for its serial time-out capability, but it does not play part in the acceleration or deceleration of the motors while in this mode.

The continuous mode of operation is probably the most dominant mode of the robot. **TOC2** and **TOC3** interrupts are activated upon entering this mode. Reference should be made to Figures 6.5 and 6.6 for operation. The continuous mode allows the host system to establish the desired speeds of the motors as well as their corresponding rates of accel/decel (acceleration and deceleration). Only one variable for each motor is used for both accelerating and decelerating purposes. Once the desired speeds are established, the subsystem will automatically begin to reach the desired speed from its current speed based on the value of accel/decel for each motor. All timer information for motor speeds is specified in terms of periods, not frequencies, within the MCU's subsystem. Two sixteen-bit internal storage registers contain all pertinent information for the pulsing of each motor. Motor one will be used for explanation purposes, but motor two has the same set of features. The variable *c_spd1* contains the current pulse

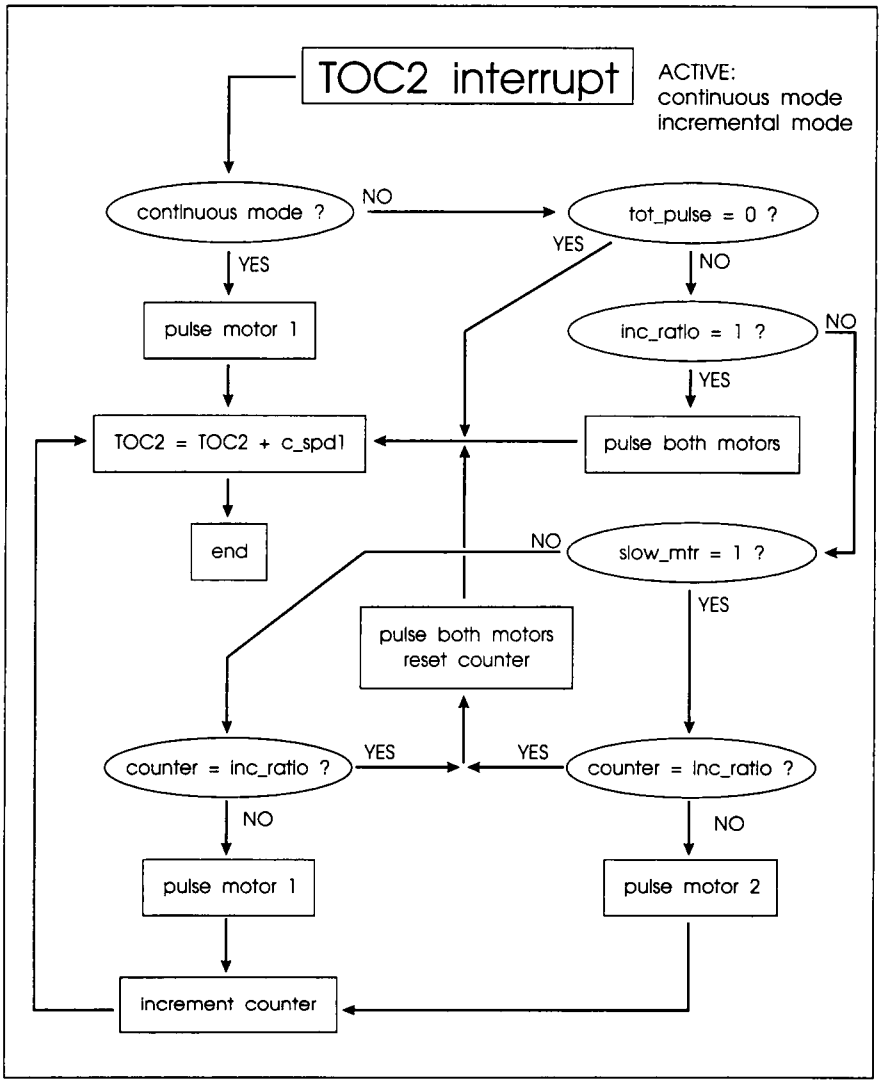

Figure 6.5: *TOC2 Interrupt service routine for the pulsing of the stepper motor(s).*

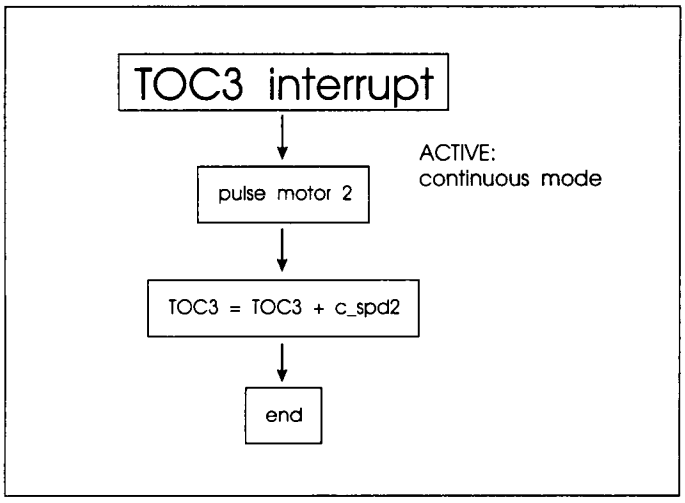

Figure 6.6: *TOC3 Interrupt service routine for the pulsing of the left motor.*

period. Should this current value not match the desired value *d_spd1*, the RTI interrupt service routine will either increment or decrement *c_spd1* by the amount contained in the accel/decel rate variable *d_acc1*. Figure 6.7 shows the block diagram operation of the RTI. Each RTI serviced will only provide one such increment or decrement. After the RTI has occurred a number of times, the speed of the motor will eventually equal the desired speed. Successive RTI's will not change the speed if the desired speed is reached. Since it occurs every 4.1 msec, the RTI provides for constant acceleration/deceleration. The *c_spd1* value is added to the *TOC2* timer value after the **TOC2** interrupt service routine has completed pulsing the motor for next pulse occurrence.

The incremental mode of operation requires the use of a single timer interrupt to pulse both motors, and is used for advancing the robot a given number of pulses before stopping. The host system can send the MCU setup parameters for this mode of operation. Three key variables, which are modifiable in this mode by the host system, are *tot_pulses*, *inc_ratio*, and *slow_mtr*. The variable *tot_pulses* (total pulses) represents the

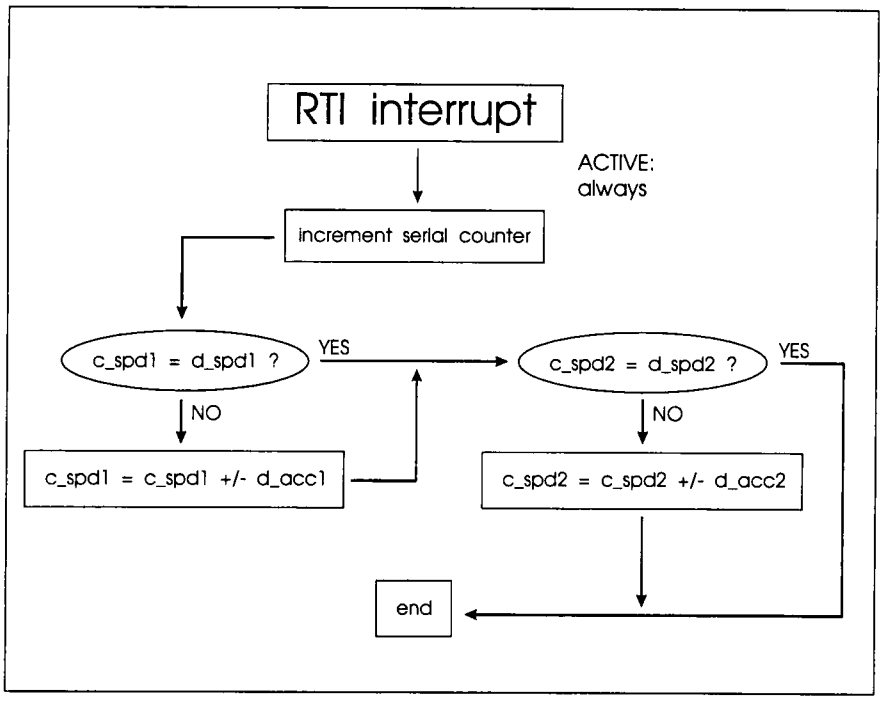

Figure 6.7: *RTI Interrupt Service Routine for Motor Control to monitor serial time-outs and provide acceleration/decelleration of motors.*

number of pulses to be given to the motors before stopping the motors, and the host system can send a serial command, which accesses a flag, to acknowledge whether or not the given number of pulses have been sent. The flag is a variable within the MCU which contains a binary one or zero to indicate completion. Speed and accel/decel rate are inherited from motor one and are applied to both motors in this mode. The *inc_ratio* (incremental ratio) is defined as the ratio of pulses given to one motor before pulsing the other. A default value of 1 indicates both motors are to be pulsed the same number of times. Should this value be changed to 2, one motor will be pulsed twice as many times as the other. The *slow_mtr* (slow motor) variable, which is either a 1 (for motor one) or 2 (for motor two), specifies the slower motor whose relative speed is defined by *inc_ratio*. Thus, *inc_ratio* defines the relative speed of the motors; and *slow_mtr* defines which motor receives the fewer number of pulses. In such a situation, *tot_pulses* refers to the number of pulses to be sent to the faster motor. This arrangement is useful for providing turning capability given a certain number of pulses to be applied.

6.2.2 Motor Control Serial Protocol / Command Set

All serial transmissions received by the MCU are polled in software. The MCU utilizes its **ACIA** (Asynchronous Communications Interface) for such transmissions. Parity, byte size, stop bit, and baud are all software selectable within the MCU's internal control registers. The system is set up for no parity, 8 bit, 1 stop bit, and baud rate of 9600.

A protocol is established for serial communications between the onboard MCU and a connected host computer system. The overall system is set up as a master-slave system. The MCU will react based only on the commands sent by the host computer. At

no time will the subsystem send bytes to the host system without the host system's knowledge of such incoming data. The protocol begins with the host computer system sending a start of transmission byte ASCII 'E'. The next byte sent is the command byte. A total of 28 command bytes are used, and commands are represented by decimal values 1 through 28. The last value sent by the host system is the checksum value which is a single unsigned eight bit value. This value is the unsigned decimal addition of ASCII 'E' and the command byte along with any data bytes sent with the command. If properly sent, the command will be serviced by the MCU; and the MCU will return data if appropriate, along with a checksum. For commands which do not have return data, the MCU will send the received checksum, if valid. Table *6.1* lists the available serial command options.

Continuous Mode of Operation _____ 'E' 1 70

This command will set the motor controller subsystem for continuous mode of operation. The subsystem will set up two interrupts. Each motor has a separate interrupt and interrupt service routine dedicated to it. This mode is used to allow each of the motors to be automatically pulsed in a continuous manner described by each motor's current step and direction values. The frequencies by which each motor is pulsed depends on the current frequency value for each motor. Should the current frequency not match the desired frequency for each motor, the current frequency will approach the desired frequency at a linear rate dependent upon the accel/decel values given for each motor. Should the motors be disabled, the pulsing of the motors will cease. The interrupt service routines for such a mode will continue to be serviced; however, if the motor is disabled, the pulses sent to the hardware are masked out. The pulsing will

Table 6.1: *Stepper Motor Control Serial Command Set.*

Com	Meaning	Com	Meaning
1	Continuous Mode of Operation	15	Set M1 for CW Operation
2	Incremental Mode of Operation	16	Set M1 for CCW Operation
3	Set Incremental Rate and Slow Motor	17	Set M2 for CW Operation
4	User Control Mode of Operation	18	Set M2 for CCW Operation
5	Set Motors for Forward Direction	19	Pulse M1
6	Set Motors for Reverse Direction	20	Pulse M2
7	Set Motors for Pivotal Right Turn	21	Enable Motors
8	Set Motors for Pivotal Left Turn	22	Disable Motors
9	Set Desired Speeds	23	Emergency Stop
10	Set Desired Accel/Decel Rates	24	Pulse both Motors Simultaneously
11	Set M1 for Half Step Operation	25	Get Motor Control Information
12	Set M2 for Half Step Operation	26	Get Task Complete Flag
13	Set M1 for Full Step Operation	27	Set New Encoder Values
14	Set M2 for Full Step Operation	28	Set Incremental Mode Pulse Count

continue at current values when the motors are once again enabled. The only return byte will be the checksum, **70**, if the command is successful.

Incremental Mode of Operation _____'E' 2 71

This command will set the motor controller subsystem for incremental mode of operation. This mode is used to allow the motors to be automatically pulsed for a given number of pulses in a manner described by each motors current step and direction values. The frequencies by which each motor is pulsed depend on the current frequency values for M1 only. Should the current frequency not match the desired frequency for M1, the current frequency will approach the desired frequency at a rate dependent upon the accel/decel rates given for M1. Upon entering such a mode, the pulse counter will be set to zero, thus disabling pulses to be sent to the motors. Serial command **28** will set a counter value for the number of pulses but may be overwritten at any point in time. Should the motors be disabled, the pulsing and counter will stop. When the motors are enabled, the pulsing and counting will resume until the counter has decremented to zero, at which time the motors will stop; and a task complete flag will be set for indication purposes to be gathered by the host system. The only return byte will be the checksum, **71**, if the command is successful.

Set Incremental Rate and Slow Motor ____'E' 3 Rate Slow Checksum

This command is intended to be used in conjunction with the incremental mode of operation. A single interrupt for pulsing the motors is used in the incremental mode. This interrupt is responsible for sending the pulses to each of the motors. The frequency of these pulses is determined by M1's current speed. The incremental **Rate** is described

as a single byte and is the ratio of pulses given to the faster motor before the pulsing of the slower motor. A **Rate** of 1 would indicate that both motors are pulsed at the same time; and the frequency of the pulse depends solely on the current frequency of M1. For this rate, the **Slow Motor** variable has no meaning. A **Rate** of 2 indicates that a motor is to be pulsed twice as many times as the **Slow Motor**. The **Slow Motor** variable is a single byte of value 1 or 2 indicating the motor to be considered the slower motor; and, if the command is successful, the subsystem will return the sent **Checksum**.

User Control Mode of Operation _____ 'E' 4 73

This command will set the motor controller subsystem for user-control mode of operation. All interrupts for motor pulsing will be turned off. The host system must send proper commands for pulsing the motor(s) if and when desired. The only return byte will be the checksum, **73**, if the command is successful.

Set Motors for Forward Direction _____ 'E' 5 74

This command will set each motors direction to correspond to the forward direction of movement. The only return byte will be the checksum, **74**, if the command is successful.

Set Motors for Reverse Direction _____ 'E' 6 75

This command will set each motors direction to correspond to the reverse direction of movement. The only return byte will be the checksum, **75**, if the command is successful.

Set Motors for Pivotal Right Turn _____ 'E' 7 76

This command will set each motors direction to correspond to the pivotal right turn

of movement. This setup will allow each wheel of the robot to be turned in opposite directions of one another (pivot), corresponding to clockwise rotations. The only return byte will be the checksum, **76**, if the command is successful.

Set Motors for Pivotal Left Turn _____'E' 8 77

This command will set each motors direction to correspond to the pivotal left turn. This setup will allow each wheel of the robot to be turned in opposite directions of one another (pivot), corresponding to counter-clockwise rotations. The only return byte will be the checksum, **77**, if the command is successful.

Set Desired Speeds _____'E' 9 M1a M1b M2a M2b Checksum

This command will set the desired speed for each motor. The speed values sent are sixteen bits long each. The 'a' extension refers to the high byte, and the 'b' extension refers to the lower byte. These sixteen-bit unsigned values represent the period at which each motor will be pulsed, and the period is based on a 2MHz bus internal to the MCU, yielding 0.5μsec/count. A maximum period of hex $FE70 is allowable, which corresponds to a minimum frequency of 30.7 Hz. For values greater than $FE70, the MCU will not pulse the particular motor; and the frequency is interpreted as zero. No limits are placed for the minimum period. The only limitations will be the ability of the motor to properly operate at such low periods (high frequencies). The only return byte will be the **checksum** which was sent to the subsystem by the host if the command is successful.

Set Desired Accel/Decel Rates ____'E' 10 M1_rate M2_rate Checksum

This command will set the desired acceleration/deceleration rates for each motor. The rates for both acceleration and deceleration are the same. The rate values are a single unsigned byte for each motor. The range of rates is from 1 to 255. Rates of 0 are automatically adjusted to a value of 1 without giving the host system an error. The greater the rate value, the faster the motors will change from their current speed values to their desired speeds. For maximum rates of 255, the MCU will instantaneously change the current speed to the desired speed when changes in speeds occur. Otherwise, the current speed periods of each motor are changed by their current rate value to reach the desired speed period. This transition occurs in the RTI (Real Time Interrupt) service routine, which occurs every 4.1msec. Each time a RTI is issued, the current speed of each motor will be either decremented or incremented by its corresponding rate (if needed), which is necessary to allow a linear ramping of speeds (constant acceleration) to be given to each motor when reaching the desired speeds. The RTI recognizes whether an increase or decrease in rate is needed (if needed) and adjusts accordingly. The only return byte will be the **checksum**, which is sent to the subsystem by the host if the command is successful.

Set M1 for Half Step Operation _____'E' 11 80

This command is used to set M1 (right front) for half step operation, which corresponds to 0.9°/pulse, or 400 pulses/rev, for motor one, and due to a 2:1 gear ratio of the robot, corresponds to 0.45°/pulse, or 800 pulses/rev, for the right front wheel. The only return byte will be **80** if the command is successful.

Set M2 for Half Step Operation ——— **'E' 12 81**

This command is used to set M2 (left front) for half step operation, which corresponds to 0.9°/pulse, or 400 pulses/rev, for motor two, and due to a 2:1 gear ratio of the robot, corresponds to 0.45°/pulse, or 800 pulses/rev, for the left front wheel. The only return byte will be **81** if the command is successful.

Set M1 for Full Step Operation ——— **'E' 13 82**

This command is used to set M1 (right front) for full step operation, which corresponds to 1.8°/pulse, or 200 pulses/rev, for motor one, and due to a 2:1 gear ratio of the robot, corresponds to 0.9°/pulse, or 400 pulses/rev, for the right front wheel. The only return byte will be **82** if the command is successful.

Set M2 for Full Step Operation ——— **'E' 14 83**

This command is used to set M2 (left front) for full step operation, which corresponds to 1.8°/pulse, or 200 pulses/rev, for motor two, and due to a 2:1 gear ratio of the robot, corresponds to 0.9°/pulse, or 400 pulses/rev, for the left front wheel. The only return byte will be **83** if the command is successful.

Set M1 for CW Operation ——— **'E' 15 84**

This command is used to set M1 (right front) for clockwise operation. This corresponds to forward direction of movement. The only return byte will be **84** if the command is successful.

Set M1 for CCW Operation ——— 'E' 16 85

This command is used to set M1 (right front) for counter-clockwise operation. This corresponds to reverse direction of movement. The only return byte will be **85** if the command is successful.

Set M2 for CW Operation ——— 'E' 17 86

This command is used to set M2 (left front) for clockwise operation. This corresponds to reverse direction of movement. The only return byte will be **86** if the command is successful.

Set M2 for CCW Operation ——— 'E' 18 87

This command is used to set M1 (left front) for counter-clockwise operation. This corresponds to forward direction of movement. The only return byte will be **87** if the command is successful.

Pulse M1 ——— 'E' 19 88

This command is used to pulse M1 (right front). The only return byte will be **88** if the command is successful.

Pulse M2 ——— 'E' 20 89

This command is used to pulse M2 (left front). The only return byte will be **89** if the command is successful.

Enable Motors ——— **'E' 21 90**

This command is used to enable the motors. By enabling the motors, current setup conditions will detail the operation of the motors; and the coils will once again be energized. The only return byte will be **90** if the command is successful.

Disable Motors ——— **'E' 22 91**

This command is used to disable the motors. The disabling of the motors not only stops the motors from being pulsed, but also prevents the coils from being energized. The only return byte will be **91** if the command is successful.

Emergency Stop ——— **'E' 23 92**

This command is used to immediately stop the motors. The coils of the motors will be energized for stopping purposes, yet the motors will not move. This command changes the current mode of operation to the user controlled mode of operation. All interrupts associated with both continuous and incremental modes of operation will be halted before the command call. The only return byte will be **91** if the command is successful.

Pulse Both Motors Simultaneously ——— **'E' 24 93**

This command is used to pulse both M1 (right front) and M2 (left front) simultaneously under each motor's current configuration. The only return byte will be **93** if the command is successful.

Get Motor Control Information ——— **'E' 25 94**

This command is used to gather information regarding the current configurations of

both motor one (M1) and motor two (M2) along with additional mode information. Fifteen bytes of such information are returned to the host system and are as follows:

1. Sticky byte
2. High byte M1 current encoder reading
3. Low byte M1 current encoder reading
4. High byte M1 current speed value
5. Low byte M1 current speed value
6. High byte M1 desired speed value
7. Low byte M1 desired speed value
8. M1 current accel/decel value
9. High byte M2 current encoder reading
10. Low byte M2 current encoder reading
11. High byte M2 current speed value
12. Low byte M2 current speed value
13. High byte M2 desired speed value
14. Low byte M2 desired speed value
15. M2 current accel/decel value

The only byte sent which needs further explanation is the (Sticky byte), which contains embedded information regarding each motor's current step and direction values. Likewise, the *sticky byte* contains information regarding motor enable, slow motor, and motor mode. The bits of the *sticky byte* can be summarized as follows:

- bit 0: M1 direction (0 CW, 1 CCW)
- bit 1: M1 step (0 FULL, 1 HALF)
- bit 2: M2 direction (0 CW, 1 CCW)
- bit 3: M2 step (0 FULL, 1 HALF)
- bit 4: Slow Motor (0 M2, 1 M1)
- bit 5: Motor Enable (0 disabled, 1 enabled)
- bit 6: Current Mode (LSB)
- bit 7: Current Mode (MSB)

After sending the returned bytes, the MCU will send a checksum byte which is the unsigned addition of each returned byte, starting with the current checksum of **94**.

Get Task Complete Flag _____ **'E' 26 95**

This command is used so the host system will know when the given pulse count has been completed during the incremental mode of operation. The subsystem will return a single byte of data and a checksum, which is the unsigned addition of the current checksum value **95** plus the returned data byte. The returned data byte will be 1, if

complete, or 0 if not complete.

Set New Encoder Values _____'E' 27 M1a M1b M2a M2b Checksum

This command will set the desired encoder values for each motor. The encoder values sent are sixteen bits long each. The 'a' extension refers to the high byte; and the 'b' extension refers to the lower byte. These sixteen-bit unsigned values represent the current encoder values to be applied to the subsystem for each motor. The encoder values are software controlled and are incremented or decremented when a motor is pulsed, depending on the motor's current direction. Forward movement of the robot increments both encoder values. Reverse movements decrement both encoder values. The subsystem will return the sent **Checksum** if the command is properly received.

Set Incremental Mode Pulse Count _____'E' 28 Pa Pb Checksum

This command will set the desired encoder values for each motor. The encoder values sent are sixteen bits long each. The 'a' extension refers to the high byte; and the 'b' extension refers to the lower byte. These sixteen bit unsigned values represent the current encoder values to be applied to the subsystem for each motor. These encoder values are software controlled values which, when a motor is pulsed, are incremented or decremented depending on the motor's current direction. Forward movement of the robot increments both encoder values. Reverse movements decrement both encoder values. The subsystem will return the sent **Checksum** if the command is properly received.

6.3 Supplement C Programming Routines Specific to Motor Control

C routines were generated to control all functions of the Stepper Motor Control Subsystem. Simple function calls handle all serial communications for the host system to properly communicate with the subsystems MCU. All routines make the proper serial connections of the host system to the subsystem via a hardware serial multiplexer. The routines are passed a structure pointer of type ROBOT found in *robot.h*, which contains variables pertaining to the subsystem. The variables along with their descriptions can be found in Table *6.2*. These variables are updated each time a routine is called and is successful. Routines which have return values associated with them return zero upon successful operation, or one for failure. The current routines available are as follows:

int m1_cw(ROBOT *rob)

This routine is used to set motor one (right front) for clockwise operation, which corresponds to the right motor moving in forward direction. If the routine is successful, **mtr.m1.dir** will be set to 1.

int m1_ccw(ROBOT *rob)

This routine is used to set motor one (right front) for counter-clockwise operation, which corresponds to the right motor moving in reverse direction. If the routine is successful, **mtr.m1.dir** will be set to -1.

int m1_full(ROBOT *rob)

This routine is used to set motor one (right front) for full step operation. This cor-

Table 6.2: *ROBOT Structure Variables for Motor Subsystem.*

Variable	Description
mtr.port	Multiplexer Port Value
mtr.mode	Current Motor Mode
mtr.enable	Current Motor Enable/Disable
mtr.inc_pulses	Incremental Pulses
mtr.inc_ratio	Incremental Ratio
mtr.slow_mtr	Slow Motor Value
mtr.task_com	Incremental Mode Task Complete Flag
mtr.m1.d_spd_freq	M1 Desired Speed (Frequency)
mtr.m1.c_spd_freq	M1 Current Speed (Frequency)
mtr.m1.d_spd_per	M1 Desired Speed (Period)
mtr.m1.c_spd_per	M1 Current Speed (Period)
mtr.m1.d_acc	M1 Desired Accel/Decel Rate
mtr.m1.dir	M1 Direction Value
mtr.m1.step	M1 Step Value
mtr.m1.enc	M1 Encoder Value
mtr.m2.d_spd_freq	M2 Desired Speed (Frequency)
mtr.m2.c_spd_freq	M2 Current Speed (Frequency)
mtr.m2.d_spd_per	M2 Desired Speed (Period)
mtr.m2.c_spd_per	M2 Current Speed (Period)
mtr.m2.d_acc	M2 Desired Accel/Decel Rate
mtr.m2.dir	M2 Direction Value
mtr.m2.step	M2 Step Value
mtr.m2.enc	M2 Encoder Value

responds to 400 pulses/rev for a 2:1 gear reduction. If the routine is successful, **mtr.m1.step** will be set to 1.

int m1_half(ROBOT *rob)

This routine is used to set motor one (right front) for half step operation. This corresponds to 800 pulses/rev for a 2:1 gear reduction. If the routine is successful, **mtr.m1.step** will be set to 0.5.

int m1_clk(ROBOT *rob)

A pulse will be sent to the hardware for advancing the right stepper motor in its current configuration when this routine is called. **mtr.m1.enc** will be incremented or decremented according to current direction, if the routine is successful. Increments are based on forward motor movements, and decrements are based on reverse motor movements.

int m2_cw(ROBOT *rob)

This routine is used to set motor two (left front) for clockwise operation, which corresponds to left motor moving in reverse direction. If the routine is successful, **mtr.m2.dir** will be set to -1.

int m2_ccw(ROBOT *rob)

This routine is used to set motor two (left front) for counter-clockwise operation, which corresponds to left motor moving in forward direction. If the routine is successful, **mtr.m2.dir** will be set to 1.

int m2_full(ROBOT *rob)

This routine is used to set motor two (left front) for full step operation. This corresponds to 400 pulses/rev for a 2:1 gear reduction. If the routine is successful, **mtr.m2.step** will be set to 1.

int m2_half(ROBOT *rob)

This routine is used to set motor two (left front) for half step operation. This corresponds to 800 pulses/rev for a 2:1 gear reduction. If the routine is successful, **mtr.m2.step** will be set to 0.5.

int m2_clk(ROBOT *rob)

A pulse will be sent to the hardware to advance the left stepper motor when this routine is called. **mtr.m2.enc** will be incremented or decremented according to current direction, if the routine is successful. Increments are based on forward motor movements, and decrements are based on reverse motor movements.

int pulse_both(ROBOT *rob)

This routine pulses both right and left stepper motors simultaneously in their current configurations. Both **mtr.m1.enc** and **mtr.m2.enc** will be incremented or decremented based on their current direction values. Increments are based on forward motor movements, and decrements are based on reverse motor movements.

int set_for(ROBOT *rob)

This routine sets the robot for the forward direction of movement. Both **mtr.m1.dir**

and **mtr.m2.dir** are set to 1 if the routine is successful.

int set_rev(ROBOT *rob)

This routine sets the robot for the reverse direction of movement. Both **mtr.m1.dir** and **mtr.m2.dir** are set to -1 if the routine is successful.

int set_ptr(ROBOT *rob)

This routine sets the robot for a pivotal right turn movement. Both **mtr.m1.dir** and **mtr.m2.dir** are set to -1 and 1 if the routine is successful.

int set_ptl(ROBOT *rob)

This routine sets the robot for a pivotal left turn movement. Both **mtr.m1.dir** and **mtr.m2.dir** are set to 1 and -1 if the routine is successful.

int mtr_en(ROBOT *rob)

This routine enables the motors of the robot. This, in effect, will energize the coils of both motors. **mtr.enable** is set to 1 if the routine is successful.

int mtr_dis(ROBOT *rob)

This routine disables the motors of the robot. The coils of both motors will not be energized, thus saving valuable power. Typically, the routine is called when robot is required to sit stationary for long periods of time. **mtr.enable** is set to 0 if the routine is successful.

int em_stop(ROBOT *rob)

This routine is used to unconditionally stop all pulses from being further applied to the robot in its current mode of operation. The mode of the robot will change to the user-controlled mode of operation if not already in this mode. The coils of the motors will stay energized, providing maximum stopping capability and **mtr.mode** will be set to 0 if the routine is successful.

int set_spds_per(ROBOT *rob, unsigned int d_spd1, unsigned int d_spd2)

This routine allows the robot to change its desired speed based on a period value. This value is based on the MCU's internal 2MHz bus frequency which provides a resolution of 0.5μsec/count. The variables **d_spd1** and **d_spd2** are arguments passed to the routine and correspond to the desired periods for motor one and motor two respectively; and **mtr.m1.d_spd1_per** and **mtr.m2.d_spd2_per** are set to these values if the routine is successful. Conversions exist to provide the correct frequency values for **mtr.m1.d_spd1_freq** and **mtr.m2.d_spd2_freq** to be updated upon the routines success. For periods above hex $FE70, or below frequencies of 30.7Hz, the corresponding motor will be stopped; and no further pulses will be sent. No error will be given for values above $FE70.

int set_spds_freq(ROBOT *rob, float d_spd1, float d_spd2)

This routine allows the robot to change its desired speed based on a frequency value. **d_spd1** and **d_spd2** are arguments passed to the routine and correspond to desired frequencies for motor one and motor two respectively; and **mtr.m1.d_spd1_freq** and

mtr.m2.d_spd2_freq are set to these values if the routine is successful. Conversions, which are based on the MCU's internal 2 MHz bus, change from frequency notation to period notation. The MCU uses the period notation for setting motor speeds. The variables **mtr.m1.d_spd1_per** and **mtr.m2.d_spd2_per** are updated upon success. For frequencies below 30.7Hz or periods above hex $Fe70, the corresponding motor will be stopped; and no further pulses will be sent. No error will be given for values below 30.7Hz.

int set_accs_per(ROBOT *rob, unsigned char d_acc1, unsigned char d_acc2)

This routine allows the robot to change current acceleration/deceleration rates for the individual motors. The single rate for each motor corresponds to the acceleration and deceleration rate of the given motor. The valid range for each rate is 1 to 255. Should a 0 be sent, no error will be given; and the MCU will use a value of 1. Lower values correspond to slower rates of change in speed. A maximum value of 255 will cause all subsequent speed changes to be instantaneous. For values below 255, the change in speed is dependent upon a RTI service routine for the MCU, which occurs every 4.1 msec. When this service routine is entered, the MCU will alter its current period, for a given motor, by the rate value. After a number of such service routines occur, the current speed will eventually reach the desired speed and no further changes will be made until another speed change is made. The MCU automatically recognizes whether it needs to decrement or increment the period for each motor to reach the desired speed. **mtr.m1.d_acc** and **mtr.m2.d_acc** are set to the arguments sent if the routine is successful.

int mode(ROBOT *rob, unsigned char mtr_mode)

This routine is used to alter the mode the robot is to operate in. The robot has a total of three modes of operation which are numbered from 0 to 2. Mode 0 is considered to be the user control mode of operation. In this mode, all interrupts associated with motor control are turned off. The host system must supply serial commands for pulsing the motors if it wishes to do so. Mode 1 is considered to be the continuous mode of operation. This mode will provide automatic pulsing of the motors in their current configurations. Speed changes can be made via serial commands; and based on current acceleration rates for the given motors, the subsystem will seek to obtain such speeds. This mode allows the host system to perform other functions while the robot moves on its own. Mode 2 is considered to be the incremental mode of operation. In this mode, the robot will pulse its motors a selected number of times, which is dependent upon a serial command to the subsystem. This can be useful if the host system wished the robot to move a certain distance and then stop. Another added feature for such a movement is included. The host system can setup an **inc_ratio** (incremental ratio) and **slow_mtr** within the subsystem. The **inc_ratio** represents the number of times one motor will be pulsed before the opposite is pulsed. An **inc_ratio** of 1 indicates both motors are to be pulsed the same number of times; and, for this ratio, **slow_mtr** has no meaning. For an **inc_ratio** of 2, one motor will be pulsed twice as much as the other. In this situation, **slow_mtr** dictates which motor is to be considered the slower motor or the one to be pulsed the least amount of times. The variable **mtr.mode** is set accordingly upon success of the routine.

int set_encoders(ROBOT *rob, unsigned int m1_enc, unsigned int m2_enc)

This routine allows the host system to alter the current readings of the internal software encoders of the MCU to take on the values sent as arguments **m1_enc** and **m2_enc** for motor one and two encoders. The software encoders are either incremented or decremented upon a pulse sent to the motors when enabled. Incrementing occurs for forward directions of motor movements. Decrementing occurs for reverse directions of motor movements, and **mtr.m1.enc** and **mtr.m2.enc** are updated to these values if the routine is successful.

int task_complete(ROBOT *rob)

This routine is used to ask the subsystem if it has completed the given number of pulses in the incremental mode of operation. If the MCU has completed the pulses, the subsystem will return a 1 to the host system and 0 if not. This routine does not return this indication. The routine sets **mtr.task_com** to the returned value from the subsystem if successful.

int motor_info(ROBOT *rob)

This routine is used to collect all of the current parameters of the subsystem. All structure variables for the motor system are updated with such a call!

int set_inc_pulses(ROBOT *rob, unsigned int inc_pulses)

This routine is used while in the incremental mode of operation. This will set the subsystems counter for pulsing the motors a selected number of times to be the argument

inc_pulses passed to the routine. The subsystem's counter value can be overwritten when calling this routine.

int set_inc_ratio(ROBOT *rob, unsigned char inc_ratio, unsigned char slow_mtr)

This routine is to be used with the incremental mode of operation. The **inc_ratio** specifies the ratio of pulsing between the two motors. The **slow_mtr** argument details which motor is to be the slow motor. Slow motor must be of a value of 1 or 2 representing motor one or motor two. Refer to the mode() function for further explanation.

int heading_change(ROBOT *rob, float angle, float max_freq, float ratio, int resume)

For changes in the robots current heading, this routine is used. The **angle** argument specifies the angle to change. Positive values represent angle changes to the left, and negative values represent changes to the right. The **max_freq** represents the maximum allowable frequency of any motor during the change. The **ratio** tells the routine the wheel ratio to use in making the change. A **ratio** of 1 causes the pivotal turn of the robot to change by the given angle amount. Higher **ratio** values represent longer distances over which to change the given angle amount. The **resume** variable tells the routine to either stop the robot or have the robot return to forward movement, and both motor frequencies are set to **max_freq**. If **resume** is 0, the robot will stop. If resume is 1, the robot will proceed forward.

int horizontal_change(ROBOT *rob, float hor_dis, float vert_dis, float

max_freq, int resume)

This routine allows the robot to move in a horizontal direction by an amount indicated in **hor_dis**. Positive values indicate horizontal movement changes to the right and negative to the left. The **ver_dis** argument tells the robot the forward distance it can move in changing such horizontal distances. For larger **ver_dis** values, the robot will perform a "snaking" motion until the robot has moved the correct horizontal distance. For **ver_dis** values of 0, the robot will do a pivotal turn of 90 degrees, move the indicated distance, then perform another pivotal turn of 90 degrees opposite the first turn to finish its movement. After completion of such movements, the robot's heading will be in the same location as when it started the movement.

void motor_control_center(ROBOT *rob)

This routine will provide the host system with an interactive menu-driven environment, enabling the ability to call all functions regarding motor control. Values can be input for certain commands, if applicable.

6.4 Existing Motor Control Problem with Possible Solutions

Once the hardware and corresponding assembly language programming was completed for the motor control subsystem, a problem was noticed in the movements of the robot. The robot would periodically move with subtle jerks, indicating a control problem. The movements were more noticeable when the motors were pulsed at higher frequencies. Further inspection into the cause of this condition revealed the interrupts were not generating the pulses to the motors at the proper points in time. If each mo-

tor's *clk* input would be guaranteed a low to high transition at specific points in time, no control problems would exist. The low to high transition is the only concerning factor; and duty cycle can be altered with insignificant effects on the control issues. Each motor has an individual interrupt associated with it. The control problem was isolated and found to occur only when the motor interrupts would overlap each other. The 68HC11 prioritizes its interrupts and can service only one request at any given point in time. Should the interrupts be set to occur at power-of-two intervals between each other, the control problem does not occur. Such is the case when one motor would operate at half the speed of the other. This is due to one particular interrupt being able to stay ahead of the other in a "leap-frog" fashion; however, should one motor be run at a slightly slower speed than the other, there are time periods in which both interrupts will eventually cross over one another. These time periods are the source of the control problem. Once the interrupts begin to overlap each other, they will continue to overlap each other in subsequent next time pulse occurrences, which continues to occur until the interrupts become far enough apart to operate properly. The number of repeated overlapped time periods is dependent on the current speed deviation between each motor and the length of each interrupt service routine for driving the motors.

6.4.1 Failed Software Attempt One

When the source of the problem was first recognized, a software solution was thought to fix it. The pulsing of the motors occurs within the service routine, and hardware pin control can be applied for the given interrupts by altering software control registers. Each output-compare interrupt has such a software selectable control setup. The pin associated with each output compare interrupt can be set to occur for either high level,

low level, toggle, or no pin action. No pin action is the default value. The associated pins for the given output compare interrupts are used to pulse the motors. The hardware pin control actions, if selected, would detail how the pin functioned when the corresponding **toc** value matched the free-running sixteen-bit counter. The 68HC11 manual states that each output compare pin has its own dedicated sixteen bit comparator. When a comparison is made, the corresponding hardware pin control will take effect, if active.

To further test if hardware pin control would correct the control problem, a separate assembly language code was generated. The code sets the two output-compare interrupts to occur with the hardware pin control set to toggle. The interrupt service routines set the next occurrence of the interrupt. Once the interrupts were set to occur, the code was set to branch in an indefinite loop. No other interrupt sources were active during this test. Each interrupt was set to occur at the same point in time. Subsequent interrupt occurrences were set to occur at the same point in time. If the hardware pin action is guaranteed to occur during a match of its dedicated comparator, an oscilloscope will show transitions of each pin occurring at the same points in time. Should this test be positive, the control problem can be solved by linking the hardware pin actions to the output compare interrupts for pulsing the motors. The oscilloscope showed the two pins toggling, yet not at the same points in time. A noticeable 2.5 msec delay was measured between the toggling points. This indicates the hardware is not granted rights to the pin until its priority is recognized in the interrupt process. This approach will not solve the control problem found.

6.4.2 Failed Software Attempt Two

Another software approach to solving the control problem was attempted. It was thought that a single interrupt could service both motors. The interrupt would be set to occur at a frequency of 4kHz. Counters within the interrupt service routine would be used to pulse both motors. Each motor would have its own corresponding counter. Each time the interrupt service routine is entered, each counter would be incremented. When the counter for a given motor reached some predetermined value, a pulse would be sent to that motor, and the counter would be reset to zero. This method will digitize the number of frequencies the motors can be pulsed. To increase the resolution between frequency changes, the interrupt service routine should be set to occur at faster rates. This method was applied and found to severely cripple the ability of the MCU to accommodate serial communications due to the intensive amount of interrupts generated. Should the interrupt be set to occur at a lower rate, the communications improve, yet the resolution by which motor frequencies can be altered is depleted. Thus, this method proved useless.

6.4.3 Possible Hardware Fix

The only solution left for consideration is further hardware design. The actual hardware was not built but a theoretical design will be discussed. This design will provide a minimum amount of re-wiring needed to accommodate the added hardware on the robot. A single ASIC chip or MACH part can fix the problem associated with the motors. Figure 6.8 details the logical components to be implemented within the chip. The underlying principle is to allow a loadable sixteen-bit counter to maintain the pulsing of a motor. Each counter contains an internal sixteen-bit register from which

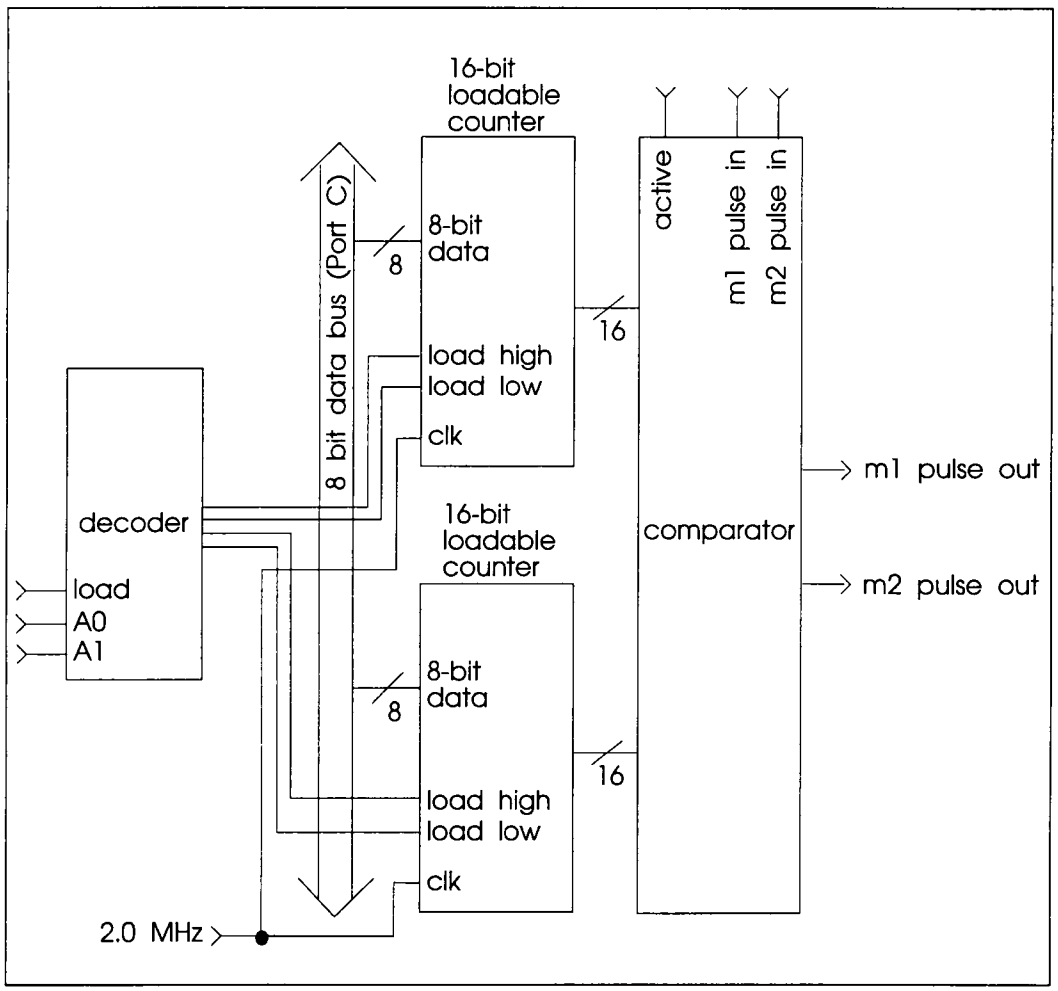

Figure 6.8: *Theoretical additional motor hardware to account for the microcontroller's inability to service two simultaneous interrupts thus possibly degrading the performance of the stepper motors for certain speed settings.*

the counter loads values after each count down to zero. The sixteen-bit register is loadable via MCU in eight-bit segments. Port C on the 68HC11EVB will be used as an eight-bit data bus for loading the sixteen-bit period associated with each motor. This will be done in two steps for low and high byte loads.

A decoder circuit will utilize 3 inputs (**load**, **A0**, and **A1**) to enable the loading of the 8 bit data bus into the corresponding register's low or high byte portion. **A0** and **A1** indicate whether the low or high byte of a particular sixteen-bit register will be loaded. The **load** input, when low, would then load the eight-bit data bus into the proper location. Each counter's internal sixteen-bit register would be loaded when the count-down value is zero. The count would occur with a 2 MHz clock input source, which mimics the onboard free-running sixteen-bit counter of the MCU, thus eliminating the need to modify large portions of the assembly language programming. The alterations to the code would be located within the interrupt service routines. The sixteen-bit value to be placed within the internal sixteen-bit registers is currently within each such service routine. The additional programming would merely set the proper address lines and byte value onto the 8 bit data bus for loading.

A comparator section is included on the hardware for comparisons of each sixteen-bit counter and additional logic for deciding how the output pulses will be handled. Of the three modes of operation, the continuous mode is the only mode which creates the motor problems. This mode uses two interrupts, one for each motor. The other modes either use no interrupts or a single interrupt for both motors. Since the single interrupt is used in the incremental mode of operation, the incremental mode does not allow such differences between motor speeds to occur. The input **active** will be used to differentiate whether or not the robot is in the continuous mode of operation; and,

in this mode, the output pulses for the motors will be based on the sixteen-bit loadable counter outputs. Should **active** not indicate the continuous mode of operation, the given **m1 & m2 pulse in** inputs will be directly fed to the **m1 & m2 pulse out** outputs. Both incremental mode of operation and user controlled operation would provide such conditions.

CHAPTER 7

RF Serial Subsystem

To provide for communication between multiple robots, RF (radio frequency) serial links are used. The wireless modems are produced by Arlan and allow for serial transmission to occur by modulating the serial transmissions using radio frequencies. Thus, no wires are physically connected between the robots for communication purposes. The links operate in the 900MHz band, capable of serial transmissions up to baud rates of 19,200. An on-board EEPROM provides serial communications set up of the device including the changing of the modulation frequency when connected to a terminal and powered-up with a push button engaged. Each robot has such a device connected directly on it. The device need only 12 volts for operation; and, upon initial power up, the device sends a broadcast message to find existing device(s), which once found, allows serial communication to take place and provides error correction capabilities. The wireless modem used for each robot is shown in Figure 7.1.

Two robots were used in the cooperative robot effort. Elvis, a VME-based robot, is placed in cooperation with SMAR-T, a laptop PC-based system. The VME system provides special hardware as seen in Figure 7.2 to operate as a real-time operating system and automatically handling incoming and outgoing serial transmissions for the programmer. Serial buffers, which can be read from and written to at will, for both incoming and outgoing serial data are used within the VME serial system. The receive buffer is automatically filled when incoming serial data exists and can optionally cleared.

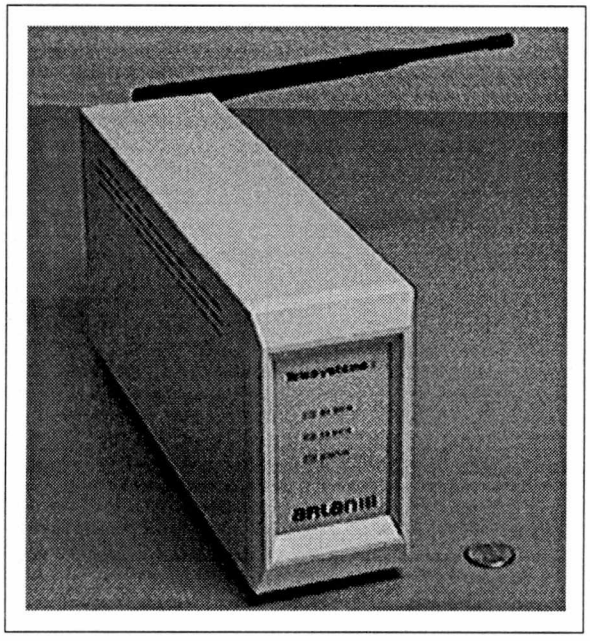

Figure 7.1: *Arlan wireless modem for inter-communication with additional robots.*

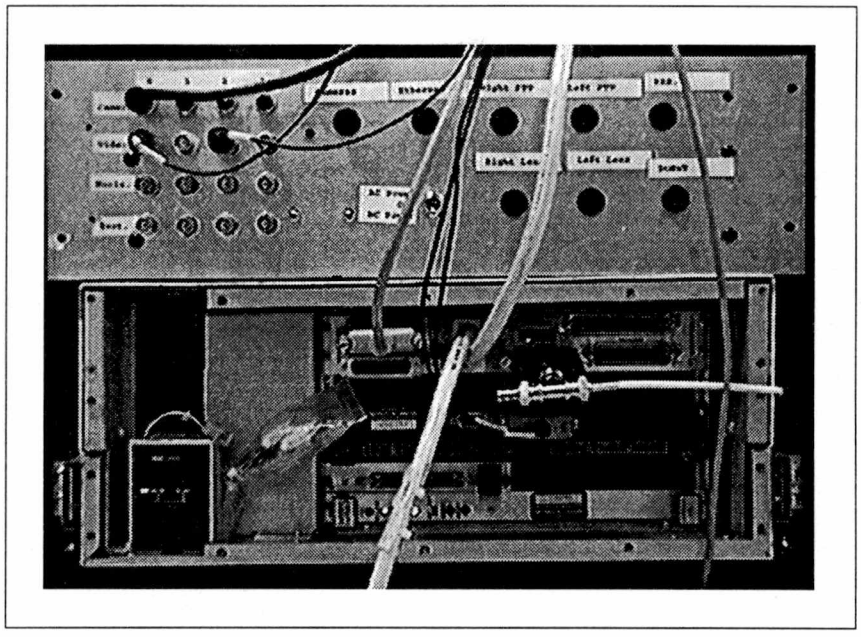

Figure 7.2: *VME rack containing Motorola 68030 processor, Maxvideo 20, and operating under VxWorks a real-time operating system.*

Once it is placed into the write buffer, data can be output onto the serial line. SMAR-T needs such a capability added before cooperation between the two robots can take place. At any point in time, both systems can easily transmit data over the RF serial link; but more important is the ability to capture incoming data when performing other jobs while not monitoring the serial line.

A protocol must be provided for both systems to allow for an understanding of the exchange of data from one robot to the next and should be designed to allow for an easy addition of commands incorporating the identities of the "talking" robots for the cooperative effort.

The description of the RF protocol used and design of SMAR-T's "talking" capabilities will be discussed in this chapter. The VME based system need only adhere to the RF protocol for proper communication between the two robots to take place. The enhanced VME serial hardware on-board Elvis will enable both receive and transmit capabilities without the worry of missing data. SMAR-T's subsystem must be designed to incorporate such a feature.

7.1 General Description of RF subsystem

All subsystems incorporated on SMAR-T are accessed via an RS232 serial link. The host system has only one serial port with which to talk to each of these subsystems. A hardware serial multiplexer has been designed to allow SMAR-T to talk with any of its individual subsystems. The ability to transmit serial data is somewhat a trivial process; however, providing the capability for incoming serial data to be stored until requested by the host system is no trivial matter. At later points in time, when the

robot sees fit to check for incoming data, the RF subsystem should provide the host with such information. If the subsystem were not present, the host system would have to constantly monitor the incoming serial line, which would severely limit the capabilities of the robot. The limitations would be indicative more of a slave robot as opposed to providing a cooperative effort between the two robots.

The separate subsystem for RF serial communications is given a dedicated microprocessor to constantly monitor incoming data and provide the capability of transmitting serial requests from the host system. A 68HC11EVB incorporating two on-board serial port connections is used as the complete subsystem in conjunction with the RF serial link, and no additional hardware needed to be designed to accompany the subsystem. One of the 68HC11EVB's serial ports is used for talking between the subsystem and host system, and the other serial port is used as a direct connection to the RF serial link, and provides an RF serial connection from the subsystem to Elvis.

7.2 Inter Robot Communication Protocol

A protocol is established to allow for communication between various robots. Packets of data are used within the protocol to allow for easy addition of commands without changing the scheme of the protocol, which conforms to the following steps for sending valid packets of data between one robotic system to another. The RF protocol uses no parity, eight bits, and 1 stop bit, and a baud rate of 9600. The host port of the 68HC11EVB is connected directly to the RF serial link, and was set up to provide interrupt-driven receiver capabilities. The transmission of data out of this port is polled in software; and the interrupt-driven receiver input was set up to allow incoming data

to be captured, irregardless of other current processing, which was necessary to ensure that no loss of data for incoming packets would occur.

1. Send start of transmission string "RFLINK".

2. Send single byte containing receiving robots identity (destination).

3. Send single byte containing sending robots identity.

4. Send single byte containing number of bytes in command packet.

5. Send command packet.

6. Send checksum byte.

To begin a transmission, the sending robot would send the ASCII string RFLINK, which establishes the start of a new transmission. The next byte to be sent is considered to be the receiving robot's identity. The robots are given identities (decimal numbers) such that robot 1 could talk with robot 2 without confusing other robots if more than two robots are present. Should the identity be different from that of SMAR-T's, the data is still received (due to the interrupt); but SMAR-T's subsystem disregards the data, knowing the data is intended for another robot. SMAR-T's identity is, by default, decimal 1, but the value can be altered by the host system if desired.

The next sequence in the protocol is to give the identity of the calling robot, which is useful for acknowledgment and task completion purposes. The number of bytes contained in the command packet is the next sequence of the protocol, and this byte is a decimal value. After sending this byte, the command packet is sent. Once the command packet has been sent, the last step in the sending procedure is to send an unsigned

eight-bit checksum containing the unsigned addition of each of the bytes represented in the command packet.

7.3 Intra Robot Communication Protocol / Command Set

An individual protocol was established for communication by the host computer system to the RF serial link subsystem. This protocol allows the ability for the host system to send various commands to the subsystem so the subsystem could react to the given commands. Serial transmissions are setup for no parity, eight-bit, and 1 stop bit; and the baud rate is set for 9600 baud. All serial transmissions have an active timer through which a time-out condition can occur if the host system does not respond; but this condition only occurs when the subsystem is receiving information from the host. The time-out is set for approximately 20.5 msec/character. Should such a time-out condition exist, the subsystem will begin looking for new commands to be sent. All serial transmissions within the subsystem's protocol are polled in software. The protocol operates in the following manner.

1. Send ASCII 'R' start of transmission character.

2. Send single command byte.

3. Send bytes of data if applicable.

4. Send checksum byte.

5. Receive bytes of data if applicable.

6. Receive checksum byte.

The command set is very short and consists of only four commands. The available commands are listed in Table 7.1.

Table 7.1: *RF Serial Link Command Set.*

Command	Meaning
1	RF serial transmit request
2	Query for incoming packets
3	Get last recovered packet
4	Change robots identity

RF serial transmit request _____'R' 1 #Bytes Data Checksum

This command is for transmitting packets of data from the host system to the RF serial link, and the subsystem acts as a middle agent in the process. The **#Bytes** represents the actual decimal number of data bytes to be sent. **Data** are the actual bytes of data which are sent to the RF serial link as soon as they are received from the host system. The subsystem is placed into a loop for the given number of bytes. The checksum byte primarily benefits the host system to ensure proper reception and transmission of data by the 68HC11EVB. If the proper checksum is received by the 68HC11EVB, the MCU will return the checksum to the host system. Should the host system not receive the exact checksum it just sent, an error has occurred; and the host should try sending the command again. The 68HC11EVB has a higher priority for incoming serial data from the RF link as opposed to responding to the host system, which ensures no incoming data from other robots will be lost by the subsystem. If such data is being received, the MCU will revert its attention from the host system to monitor the incoming data.

Should such conditions exist, the subsystem will relax its time-out for the host system to service the incoming data. Once returned from the interrupt service routine, the time-out capability is once again enforced, but is then reset.

Query for incoming packets _____'R' 2 Checksum

This command is requested by the host system when wanting information regarding the number of incoming valid packets found by the subsystem from the RF link. If a packet is intended for the host system, the subsystem will capture such data. This packet, if valid, will increment an internal variable which is within the subsystem and represents the number of current requests by other robots. Only one request by each such robot is capable of incrementing the internal variable. Should the same robot keep sending valid packets intended for the host system, the last valid packet sent is stored in the subsystem's RAM for later gathering by the host system which, in turn, will show only one such valid request present regardless of the previous requests. However, should another robot (different identity) send a valid packet intended for the host system, the internal variable will be incremented once again. Thus, the number of packets present relies solely on the number of robots which have sent valid packets intended for the host system.

Get last recovered packet _____'R' 3 Checksum1 Data Checksum2

This command allows for the host system to acquire the last valid packet received by the subsystem through the RF link. If the proper sequence representing this command is sent to the subsystem (including **Checksum1**), the subsystem will send the last available packet recovered. Once the packet has been sent to the host system, a checksum

representing the sent data within the packet **Checksum2** is sent to the host system. The subsystem then expects the host to send **Checksum2** once again to it, verifying that the host system properly received the packet. If the sent checksum by the host matches **Checksum2**, the subsystem will remove the packet from its internal RAM and decrement its internal variable containing the number of available packets. Should the checksum be different from **Checksum2**, the subsystem will retain the current packet within its RAM and begin searching for new incoming commands. The incoming packets from the RF serial link are available to the host system in a LIFO (Last In First Out) format.

Change robots identity _____'R' 4 Identity Checksum

This command will alter the identity associated with the host system (by default 1) to that of **Identity**. This effects the reception of incoming packets of which the subsystem will store only valid packets intended for its current identity. **Identity** is a decimal value representative of the robots name. Should they not be intended for the current identity of the host system, incoming valid packets are disregarded and are not kept in internal RAM for further gathering by the host system.

7.4 Assembly Language Programming for RF subsystem

Motorola assembly language compiler AS11 was used for the 68HC11EVB. The programming makes use of both serial ports located on the MCU. One serial port made a direct connection to the RF serial link and utilizes the SCI system of the microcontroller. The other serial port is used for connection to the host system and uses the

ACIA for serial transmissions. All serial transmissions are polled via software with the exception of incoming data received by the RF serial link under the SCI system. This exception will generate an interrupt upon which an interrupt service routine is entered for the handling of such an event; and extensive use of RAM is utilized as storage for such data intended for later gathering by the host system.

The ACIA system of the MCU is used for serial transmissions to and from the host system. All serial transmissions intended for the host system are polled in software. The subsystem has a higher priority for incoming serial data from the RF serial link. When such data from the link is present, an interrupt is generated; and the MCU will revert attention to the handling of the incoming data. All polled serial transmissions have a respective time-out capability. When expecting an incoming byte from the host system, a variable **OVERFLOW** is first cleared to zero. The RTI (Real Time Interrupt) capability of the MCU is used to increment **OVERFLOW** when its interrupt service routine is entered, and this routine has been set to occur every 4.1 msec. Should **OVERFLOW** reach a value of five, indicating a wait delay of 20.5 msec before the incoming byte has been received by the MCU, a time-out condition exist. In such a condition, the MCU begins to search for a new command sequence to be given by the host system. Should the MCU service incoming data from the serial link while currently involved in a serial transmission with the host system, the MCU will reset **OVERFLOW** so, when returning from the interrupt, a time-out condition is not reached. The host system has a time-out capability as well; yet the time-out is a much more relaxed value of 1 second, which is more than adequate to accommodate such a condition.

When servicing the incoming data from the link, an interrupt is requested; and an interrupt service routine is entered which will return whenever the packet is complete

or does not conform to the protocol established for it. Should a valid packet exist for the given host system, the service routine will last for the duration of the packet transmission. Once inside the routine, software disables further receiver interrupts from occurring and gathers the incoming data of the packet in a polled fashion, with time-out capabilities enforced and places the data within a temporary storage space. Should it be valid, the received packet is then moved from its temporary storage space to a storage space reserved for the sending robot's identity. This scheme will accommodate a total of five robots which are each given individual buffers of 20 bytes maximum within the internal RAM of the MCU. Six such buffers exist: five are for each robot, and one for the temporary storage space.

Once a valid packet is received, a variable **TOT_PACKS** is incremented acknowledging a valid number of packets exists. Should the same robot send another packet prior to a pending packet being gathered by the host system, the new packet, if valid, will take the place of the old packet; but **TOT_PACKS** will not be incremented in such a condition. The only manner by which **TOT_PACKS** can be incremented is when a robot sends a valid packet, and that robot has no prior pending packet to be gathered by the host system. Should the host system properly gather such a packet from the subsystem, **TOT_PACKS** is decremented; and the storage space for the robot which sent the packet is cleared.

A LIFO (Last In First Out) method of gathering packets is established for the host system. An array **ID_INDEX**, of length five bytes (one for each robot) is key for the MCU to keep track of which robot sent a received packet and the order in which the received packets were sent. Should no packets within the MCU be available to the host system, **TOT_PACKS** will be zero. Once a valid packet is received, **TOT_PACKS** is

incremented. **TOT_PACKS** serves two separate purposes: one the number of available packets to the host system, the other being an offset to allow the sending robots identity to be logged-in **ID_INDEX**. **ID_INDEX** is further used to gather the last packet whose storage is located according to the robots identity which sent it. **LOOK** is a set of 12 bytes which contains the starting addresses for the six individual buffers. The first two bytes of **LOOK** contain the starting address for the temporary storage buffer, and the next two contain the starting address for robot one's buffer. The following bytes are repeated through robot five. This method allows the MCU to find the location of the individual buffers based on the sending robot's identity when gathering packets for the host system. The **ID_INDEX** array contains information leading to the identity of the robot which sent the last packet based on **TOT_PACKS -1** as an offset. The identity value is multiplied by two and serves as an offset to the **LOOK** location for finding the starting address of the buffer, which contains the packet sent by a particular robot. Figure 7.3 shows the LIFO method in action and how the different arrays and variables previously mentioned interact with each other to allow the RF subsystem of robot 1 to point to the proper storage buffer for the gathering of packets by the host system if requested. Figure 7.4 further illustrates the process with a call from robot 3 to robot 1. Robot 2's packet was considered not to have been gathered by the host system and is a pending packet in the subsystem. As can be seen from the illustration, to gather the sent packet by robot 2, the host system must first gather the last packet which was sent by robot 3.

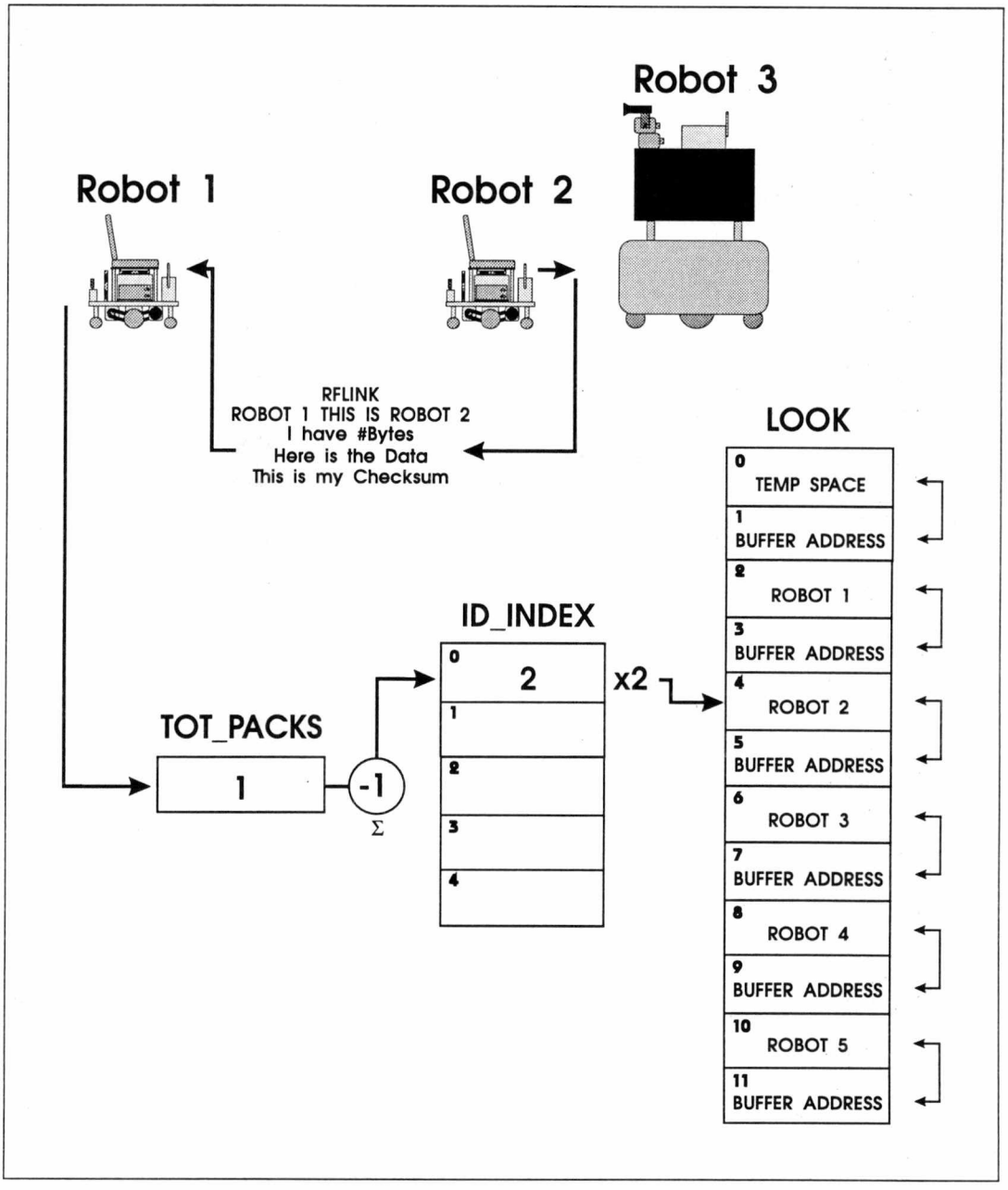

Figure 7.3: *Assembly coded arrays used by the RF Link Subsystem's microcontroller to handle and keep track of incoming broadcasts in order to provide a LIFO means of future packet collection by the host computer system. Robot 2 sends robot 1 a packet which is considered a pending packet in the subsystem not yet collected by the host.*

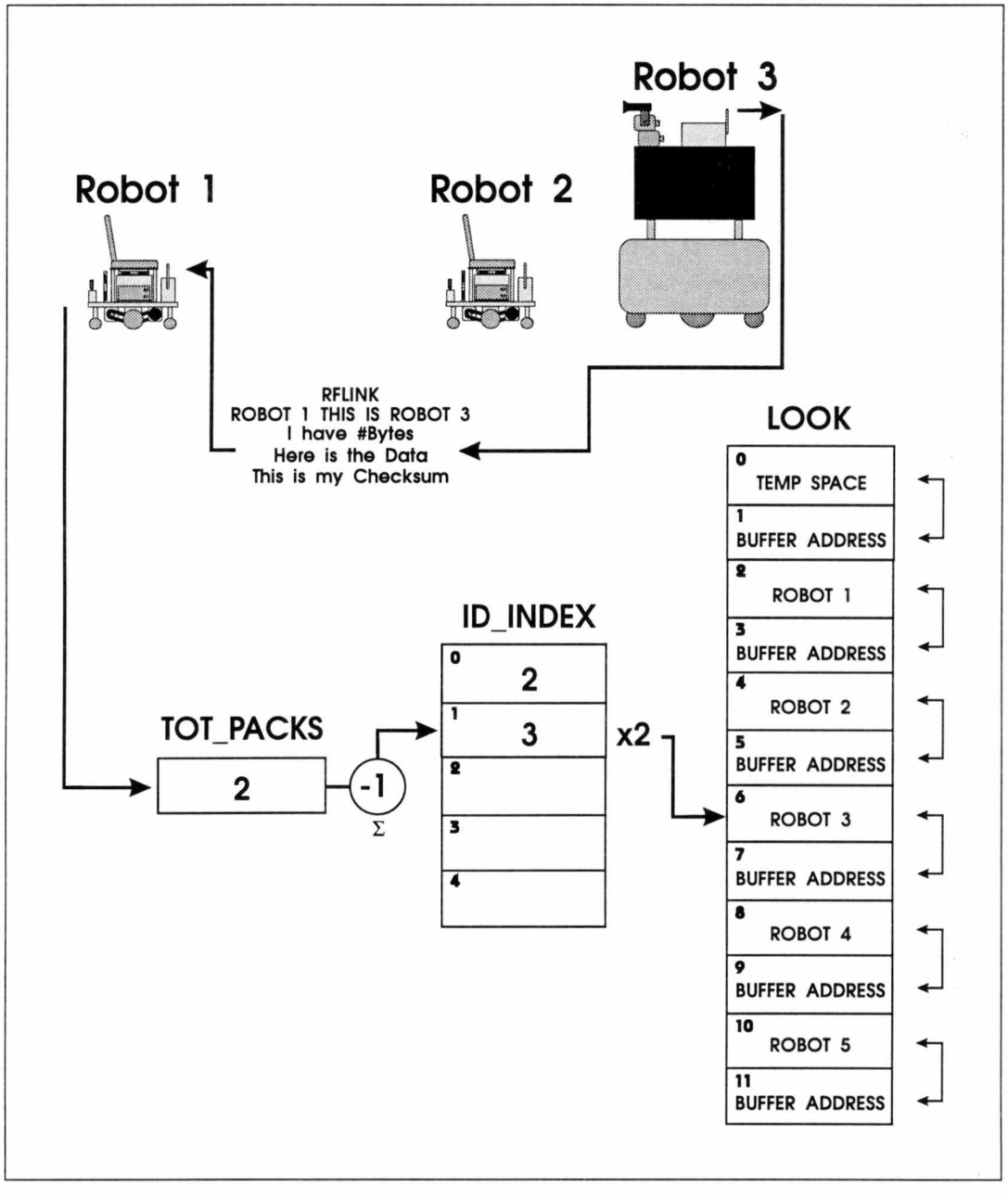

Figure 7.4: *Assembly coded arrays used by the RF Link Subsystem's microcontroller to handle and keep track of incoming broadcasts in order to provide a LIFO means of future packet collection by the host computer system. A prior pending packet sent by robot 2 is still present in the subsystem. After robot 3 broadcasts it's packet to robot 1, the subsystem updates the necessary variables to now point at robot 3 data.*

CHAPTER 8

RS232 Serial Multiplexer

The host computer has only one serial line with which to talk with existing subsystems. The motor control, sonar, and RF link subsystems are currently used, thus requiring serial-driven command sequences for setup and operation. All subsystems are considered to be slave devices and they only respond to the host computer when instructed. At no point in time will the subsystems attempt to broadcast a serial transmission without the host computer's prior acknowledgment. Consequently, multiplexing of the serial lines is feasible. All subsystems require only **GND**, **Tx** and **Rx** for a serial communication to take place. The multiplexing of such lines would only need to alter the **Tx** and **Rx** lines of the host computer to be connected to the subsystem of choice. Since the host computer makes the decision of the subsystem to which it wishes to make a connection, the host provides the selection control. This control is granted through the parallel port of the host system. A single board capable of multiplexing up to four serial lines is shown in Figure 8.1.

8.1 Operation of the MAX309 for Serial Multiplexing

The **MAX309**, manufactured by Maxim, is a dual 4-channel cmos analog multiplexer [Max94]. The duality is used in this design for the switching of both the **Tx** and **Rx** lines. The chip is capable of operating with either a single supply or bipolar supplies while still maintaining cmos logic compatibility. This design required a bipolar supply

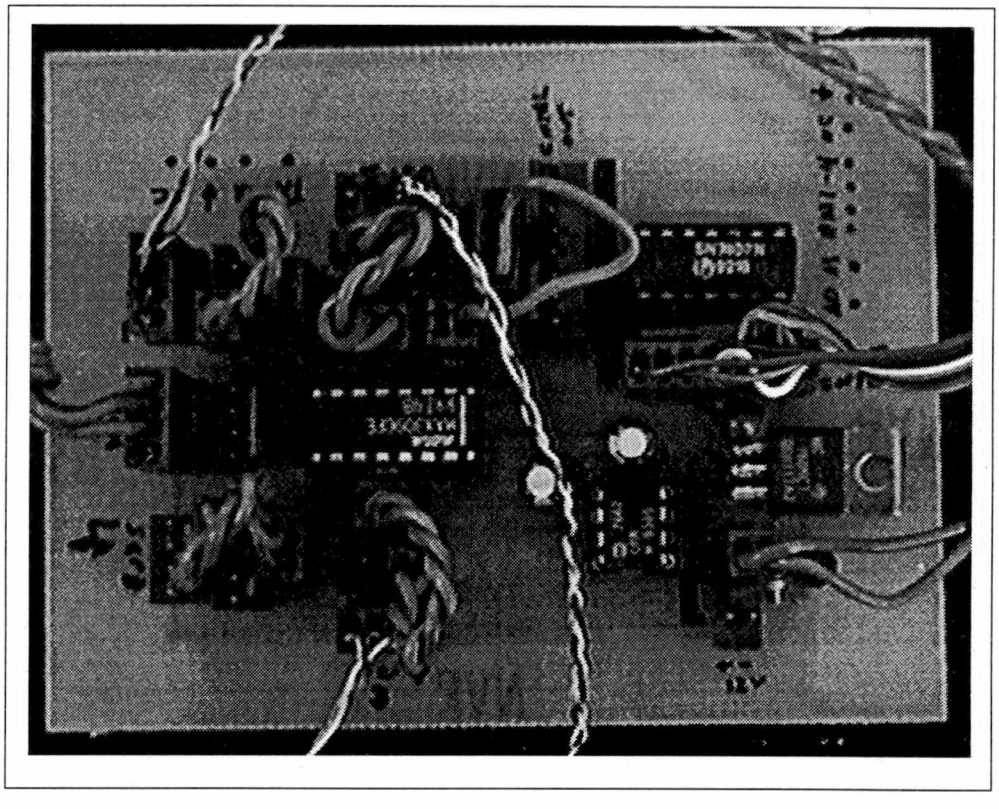

Figure 8.1: *Completed serial multiplexer printed circuit board for communication between the host PC and its dedicated subsystems.*

to enable switching of +/-12 volt RS232 transmission signals. The device can operate with bipolar source voltages from +/-4.5 volts up to +/-20 volts, a range which is well within the limits of the robot. The selection of the channels is made by two address bits, **A0** and **A1**, in conjunction with an enable bit, **EN**. Should the **EN** be asserted low, the device will not make a connection regardless of the address inputs. If **EN** is high, the address inputs will dictate the connection to be made. The host computer system's **Tx** and **Rx** lines are the *common* connections for the mux, while each subsystem is connected to the individual channels. Figure 8.2 shows the switching configuration.

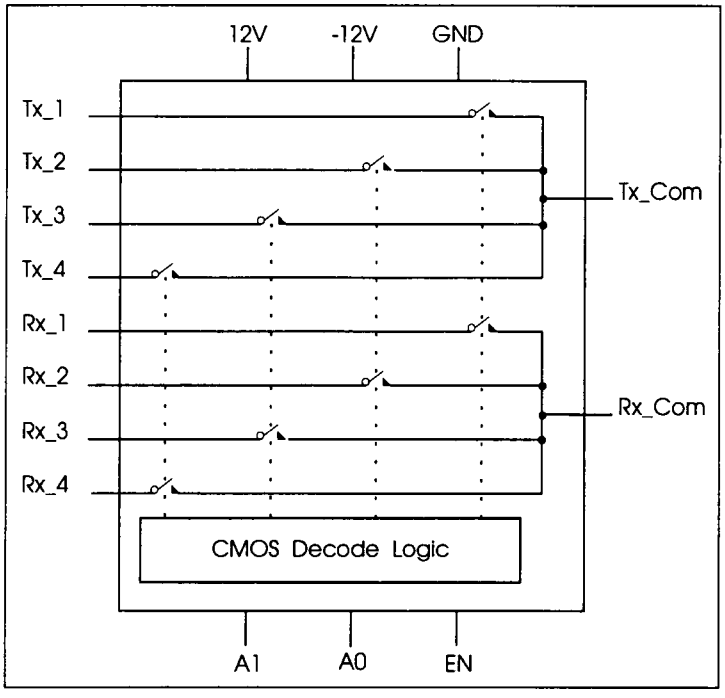

Figure 8.2: *Dual 4-channel analog multiplexer operation for switching the transmit and receive lines from the host PC to its dedicated subsystems.*

8.2 Hardware and Interfacing

A single 12 volt supply is needed for correct operation of the subsystem. The subsystem has an onboard 5 volt regulator, *7805*, for logic and can generate -12 volts from the *ICL7662*, a device produced by the Harris Corporation, which serves as a negative converter requiring the use of two capacitors and two diodes to generate the negative voltage. A minimum of 8 volts is necessary from the source voltage to provide correct operation of the 5 volt logic regulator giving a 3 volt minimum differential between the regulator's input and output.

The overall system is capable of multiplexing up to 16 **Tx** and **Rx** serial connections, which requires the stacking of 4 such boards to accommodate the additional connections and multiplexers needed. Each board contains dual headers to jump necessary lines from one board to another such as selection control, power, common **Tx** and **Rx**. For selection control from the parallel port, the jumping to additional boards occurs after the buffering of the port to minimize drive currents from the PC. Additional jumpers for each external serial device are available on each board to provide the swapping of the individual **Tx** and **Rx** lines of the external device, thus eliminating the need for null modem connectors.

The selection control for the multiplexer is generated from the parallel port of the laptop computer (host system). A total of 25 pins are present on the parallel port, but data bits (d0-d5) and ground are the only connections made in the control scheme with the data bits used as outputs [JG92]. The control of such outputs merely requires the writing of a byte of data to the memory address of the parallel port. Once written, the port will maintain the value until another value overwrites it. The default value from

the port upon power-up of the laptop is zero (all bits low). In consideration for the drive capability of the parallel port, each output bit from the port is buffered. Data bits d0-d1 on the parallel port are used for the address selection; and all boards, regardless of number, use these bits for the multiplexers address lines. The board selection or **EN** (enable of each board's mux) is made by four bits d2-d5 from the parallel port. Only one bit of these four should be logic high at any given point in time, or contentions in the serial lines could result. A jumper on each board is used to extract a dedicated bit from d2-d5 to act as the board's enable. Figure 8.3 shows the jumper connections for the use of four stacked boards. Since the robot has only three serial connections, one for each subsystem, only one board is needed. Bit d2 is selected as the boards enable. The complete schematic of each board is found in figure 8.4.

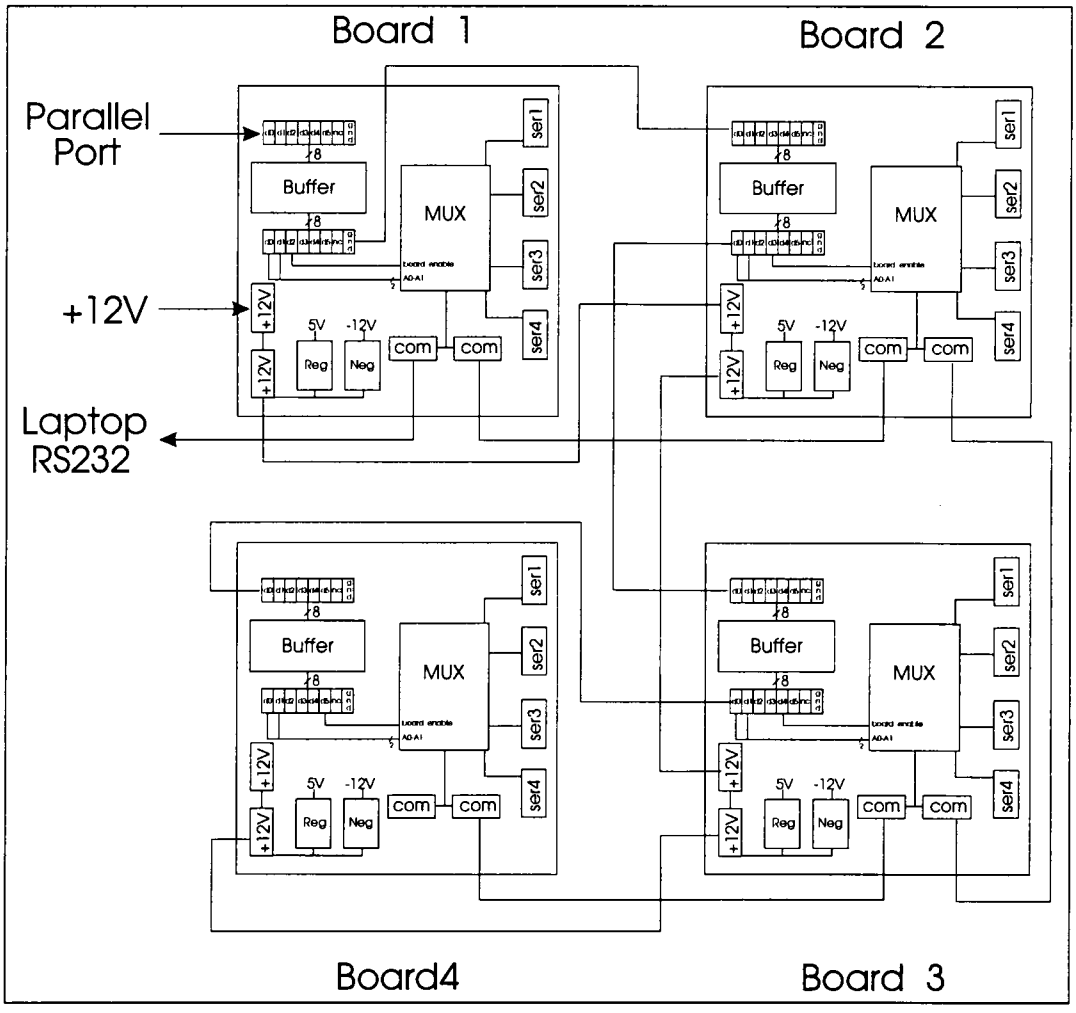

Figure 8.3: *Connections to be made for the stacking of additional serial multiplexer printed circuit boards if more than 4 serial lines are needed. Each printed circuit board supports the switching of up to 4 serial lines.*

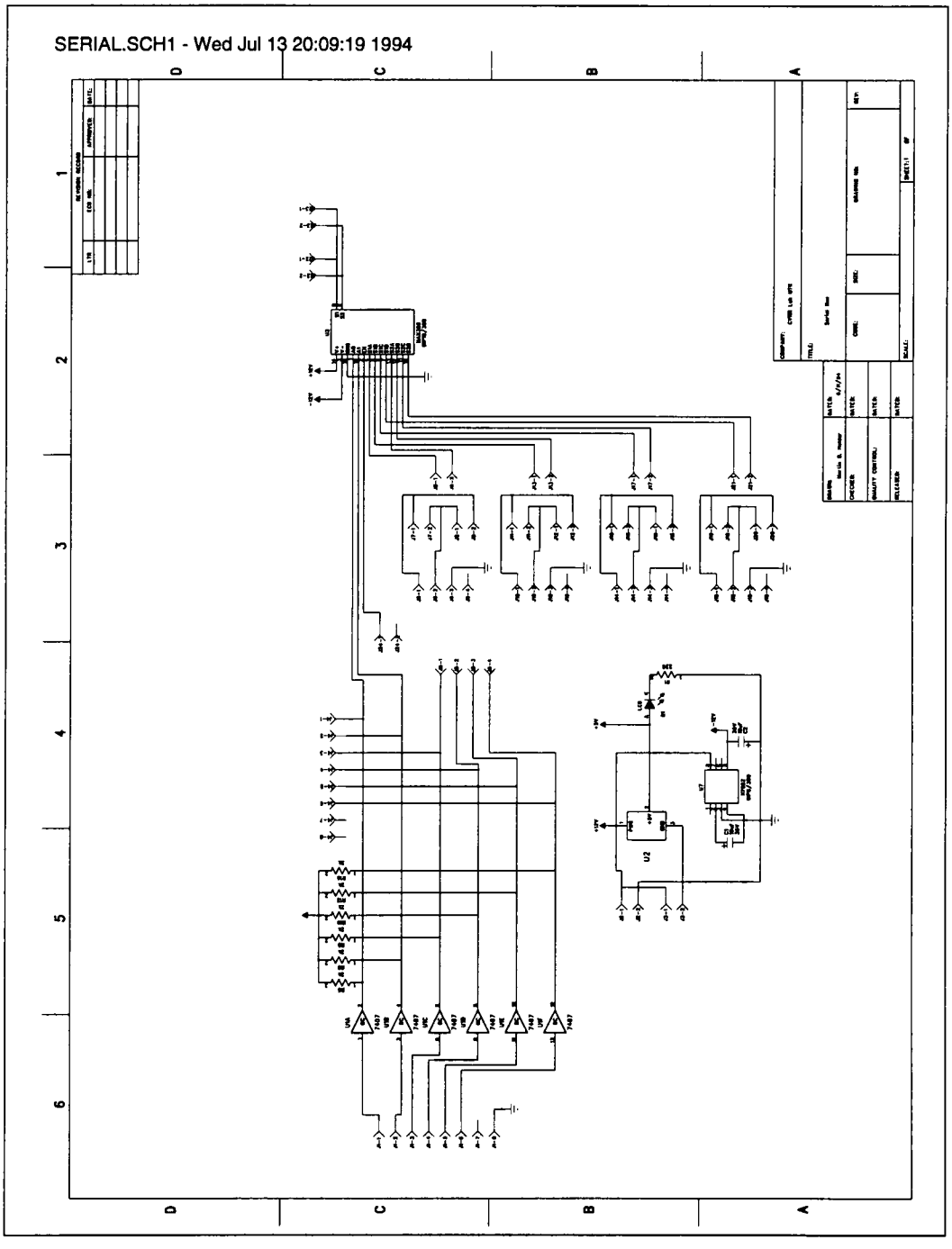

Figure 8.4: *RS232 serial multiplexer hardware for intra-communication between the host PC and its subsystems.*

CHAPTER 9

Robot Behaviors

Two main behaviors have been programmed to show the robot's capability to properly interact with its various subsystems. The robot was given the ability to follow walls, which demonstrates the host system can communicate with its lower level motor and ultrasonic transducer subsystems. The robot was also given the ability to communicate with an existing VME-based mobile robot to perform a type of recovery mission, a behavior which includes the additional use of the robots RF serial subsystem for communication with an additional robot.

9.1 Wall Following

The ability to follow walls sounds simplistic, but is by no means a trivial task for a mobile robot. A multitude of decisions must be made based on current ultrasonic sensor readings. The decisions made will dictate how both motors will be driven to prevent the robot from possible collisions, yet maintain some desired distance from the wall. The sensor readings can be faulty or misleading and can lead to wrong decisions and ultimately poor wall-following ability. The sensor's angular position relative to the given object and reflections of the sound waves from the object play an important role in the inaccuracy of such readings. A means of disregarding such readings improves the robustness of the robot in handling such a task.

The wall-following algorithm utilizes two regions of sensory data to determine the

closeness of surrounding objects. Figure 9.1 illustrates the two regions. *R1* (region one)

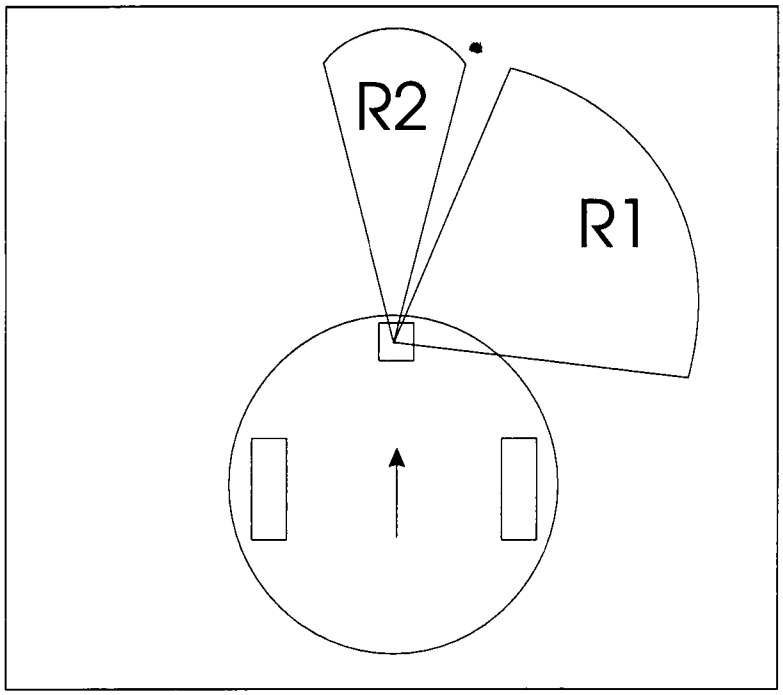

Figure 9.1: *Ultrasonic sensory regions used in providing motor adjustments for a wall-following algorithm.*

includes a total of 13 sonar readings separated by 7 degree increments to cover a total angular area of 70 degrees, and the reading which gives the lowest value is used in the decision process. This method eliminates the bad sensor readings which give values greater than the actual due to the angular position of the sensor relative to the object, and also eliminates bad readings due to reflections. This region provides the ability for the robot to look ahead to avoid upcoming collisions on the robot's right side. Should no collisions be apparent, the robot's lowest reading would most likely be the parallel reading relative to the robot and the given wall structure. *R2* represents the region in which the robot searches for non-avoidable obstacles and is provided by two sonar readings separated by 7 degrees. The lowest reading is the value used in the decision

process.

The two selected current sonar values for both regions govern the current motor movements of the robot. The algorithm for wall following is passed two important parameters, a distance and the maximum motor speed. The distance parameter, in feet, details the closeness by which the robot will attempt to hug the wall. The maximum motor speed parameter restricts the robot's motors from reaching speeds beyond this value. Both parameters are important and affect the ability of the robot to perform its wall following task.

The lower the distance parameter, the more probable it is that the robot will enter various openings along a wall such as open doors. This is due to the 30 degree cone associated with the ultrasonic transducer and the R1 region over which the transducer is scanned. For a larger distance or further wall following approach, a more broad range of sensory data is given, due to the cone. This range could be larger than the opening in question and thus prevent the robot from entering.

The maximum motor speed parameter governs the rate at which each motor can be pulsed. For faster rates more distance will be traveled by the robot before additional sonar scans can take place, and this limits the robot's ability to update its sensory information in time to alter its motor values and can cause the robot to hit nearby objects. Both the distance closeness and the maximum motor speed contribute to an overall effect toward evading possible collisions.

A prioritized approach towards the algorithm attempts to keep the robot from collisions. The top priority is to first monitor any upcoming objects in front of the robot and is represented by the *R2* region. The most probable cause of such a condition would be a corner of a room or hallway, where two walls would most likely meet at a 90 degree

angle. This is considered to be an exception case to the wall following routine. The *R1* region is used for normal evasive maneuvers and to maintain the given distance parameter between the robot and the wall. The minimum reading taken within this region is used to compute a differential distance based on the current reading and the given distance parameter. The differential distance will then alter the speeds of both motors.

For the exception case regarding the *R2* region, the robot will stop when the reading is equal to or less than the given distance parameter. The robot will stop if the distance parameter is less than 1 foot, assuming a wall change is to take place. The robot, after stopping, will begin to turn towards the left in a pivotal turn. Successive readings for the *R2* region will be taken until the reading becomes greater than the given distance parameter. This method has proven more accurate than simply turning a fixed 90 degrees. Once complete, the robot will pivot left an additional 30 degrees and stop to accommodate the sonars inability to see walls at certain angular positions, particularly at 45 degrees, relative to the wall. The robot will then proceed to follow the wall in a normal fashion.

If no objects are within the distance parameter of the *R2* region, the *R1* region provides all necessary information for normal evasive maneuvers and wall-following capabilities; and a differential distance is computed to determine the needed motor speeds. Each motor is first granted half its maximum speed prior to analyzing the differential distance, and the differential distance is then multiplied by a gain of 80. This will then represent a speed deviation in terms of Hertz. The gain of 80 was determined through successive runs of the routine and alters both motor speeds. If the deviation is found to be negative, the robot is too close to the wall and needs to move away from it. Should the deviation be positive, the robot is too far from the wall and needs to move

towards it. Each motor's speed is altered by the speed deviation, one motor will be sped up while the opposite will be slowed down, each by the same amount. The deviation, whether positive or negative, determines which motor's speed is to be increased and which should be decreased.

The method of altering the speeds of the motors based on deviations in distance proved successful. Should the robot's differential distance be zero, no changes in given speed will take place; and this indicates the robot is currently located at the given distance for wall following. Should the robot be slightly off its given distance parameter, it will converge to the parameter by slight motor speed adjustments. For cases in which the robot is following some given wall and the walls reference suddenly disappears, the robot will begin a hard turn towards the right indicating a possible corner of the wall has been reached. Figure 9.2 shows such a scenario and the movement profiles the robot will undergo. In such a condition, the speed deviation is extremely high; and the right wheel of the robot will approach zero while the left wheel will approach maximum speed. This will continue until successive sonar scans reveal the corner of the wall, at which time the motor speeds will once again be adjusted, which continues until the robot once again converges to the distance parameter between itself and the wall.

9.2 Recovery Mission

To substantiate the effectiveness of the RF serial subsystem used in a cooperative effort to talk with additional robots, a recovery mission was programmed. This mission involves the use of the VME-based mobile robot, Elvis who will transmit an S.O.S. signal, indicating its sensory system is out of commission. SMAR-T will then hear

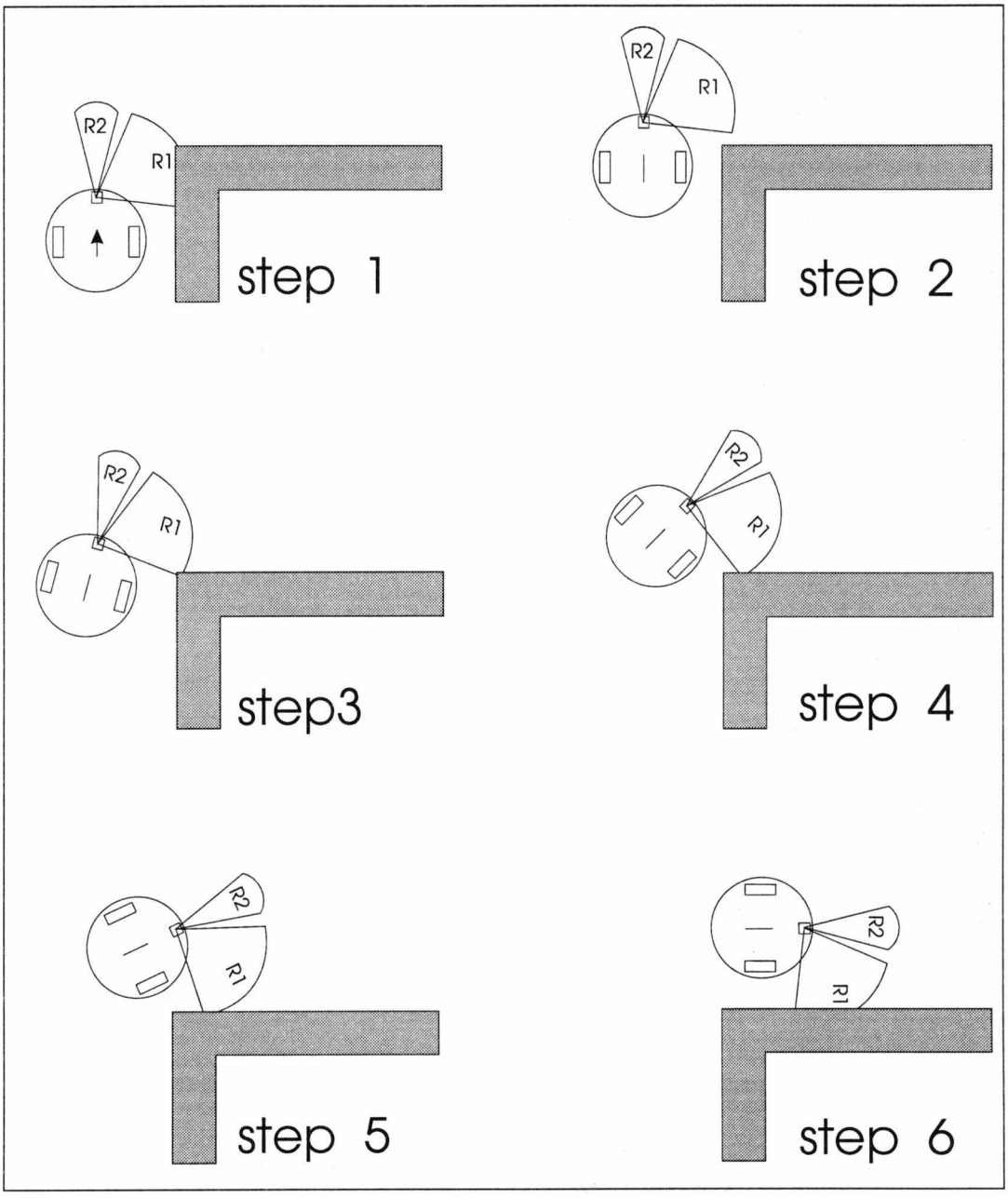

Figure 9.2: *Movement profiles of wall-following showing the robot's ability to turn a corner. Motor adjustments are constantly being made to maintain a particular distance from the wall as seen by the ultrasonic scan regions.*

the *cry for help* and begin to locate Elvis in an attempt to guide the robot to a home position. Figure 9.3 shows the two robots which will be communicating with one other.

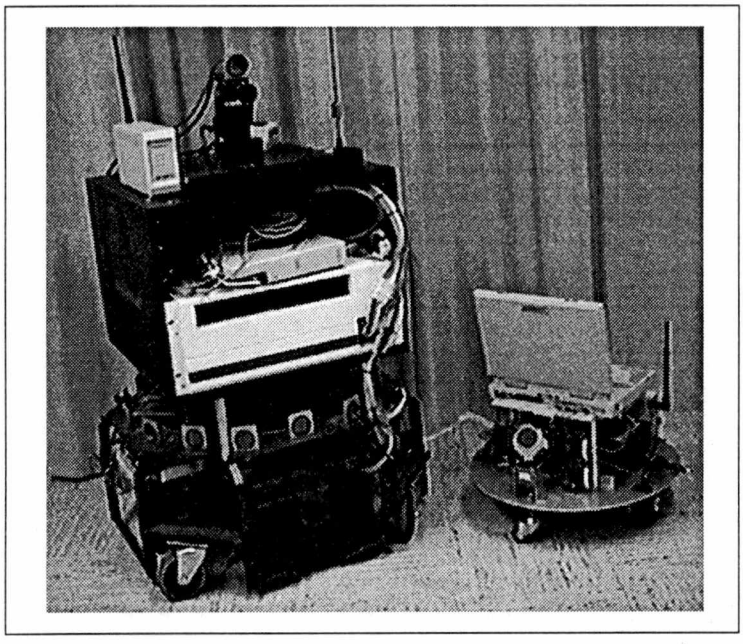

Figure 9.3: *The two cooperating robots: Elvis and SMAR-T.*

9.2.1 Broadcasting Messages

A means of communication between the robot systems must be developed to pass important cooperative messages between the two [GDMO92]. All broadcasted messages to and from the robots conform to the protocol established in the RF serial subsystem. To provide the ability to cooperate with one another for the given task, the robots are given a set of messages described as follows:

1. SOS (Help me get back to the home position)

2. ACK (Acknowledgement, I heard your request!)

3. D (Done, I have completed the movement profile sent)

4. S byte1 byte2 (Set Speed to [byte1][byte2] *mm/sec*)

5. M byte1 byte2 (Move [byte1][byte2] *mm*)

6. R byte1 byte2 (Rotate heading by [byte1].[byte2] *degrees*)

Elvis was given the ability to broadcast messages 1-3, while SMAR-T can broadcast messages 2-6. Confirmation was provided for the sending of such messages to inform the sending robot that its request has been noticed. Table 9.1 shows the confirmation method for sending messages between the two robots. Each message sent by the sending

Table 9.1: *Message Confirmation.*

Sending Robot	Receiving Robot
SOS	ACK
ACK	null
D	ACK
S	ACK, D
M	ACK, D
R	ACK, D

robot requires the receiving robot to respond with an *acknowledgement* message except for the *acknowledge* message itself. For messages 4-6 which involve movement profiles, the receiving robot (Elvis) is to first send an *acknowledgement* message later followed by a *done* message when completed.

Additional messages can be programmed as needed. The decoding scheme, residing in the C programming language, for such messages will need to be altered to accomodate

the added messages. The added messages need only adhere to the protocol through which the RF serial subsystem handles such messages. The message imbedded within the packet of data need not concern the RF subsystem. SMAR-T's decoding method provides various tests for the messages it receives and relates such messages to a numeric value for further processing.

9.2.2 Mission Explanation

Constraints are placed on this mission; and SMAR-T is to be placed at one end of a hallway while Elvis is placed in the center of the hallway somewhere in front of SMAR-T. The orientation of Elvis with respect to SMAR-T should be in one of four possible positions. Elvis can be located facing SMAR-T, 90 degrees to the right, 90 degrees to the left, or 180 degrees away from SMAR-T. All sensory input to Elvis is to be turned off and SMAR-T will provide the necessary instructional movement commands sent as messages to Elvis. No obstacles should lie in the hallway to obstruct the path between SMAR-T and Elvis. Further elaborate means of software control can be added for future, more difficult missions. The imposed constraints aid in alleviating the difficulty involved in finding Elvis since neither robot has a global positioning system on-board, and SMAR-T must rely solely on its ultrasonic transducer as a means of sensory input to find Elvis and guide it to the home position. Current programming realizes the ability to find Elvis and re-orient Elvis if necessary so the front of Elvis will point towards SMAR-T.

Once SMAR-T's recovery mission is activated, it waits for a S.O.S. broadcast from Elvis. SMAR-T's RF serial subsystem is constantly looking for incoming RF serial packets to capture the data. The host system periodically checks for the incoming

packets through RS232 communications between itself and the RF subsystem. Once the subsystem captures the packet and the host system is aware of such data, the host system collects the packet from the subsystem, which is then decoded by the host system, indicating an S.O.S. call from Elvis. SMAR-T will then broadcast an *acknowledgment* to Elvis and proceed to locate Elvis.

Once Elvis has been located, it is instructed to perform a series of movements. The movements are noticed by SMAR-T's ultrasonic transducer. Through collective reasoning, SMAR-T will reorrient Elvis's heading, if necessary, to further deduce where the front of Elvis is located. Once found, Elvis can be relocated to face SMAR-T and thus the home position.

CHAPTER 10

Conclusions

An integrated multi-processor mobile robot (SMAR-T) is designed and implemented to provide self-contained operation and inter-communicate with additional robots for cooperative purposes. To provide such capabilities, two levels of control are realized. A high level of control is furnished through the use of C routines on a laptop PC. The PC is capable of talking to one of three dedicated low level subsystems (motor control, ultrasonic scanner, and inter-robot communications) each of which contains a dedicated 68HC11 micro-processor for performing various task specific commands. The communication between the PC and such subsystems is accomplished through RS232 serial transmissions. A serial multiplexer printed circuit board is developed to provide the PC the ability to connect with a given serial device using the PC's parallel port for selection control. Additional printed circuit boards are developed to handle power regulation, motor control and sensory systems for SMAR-T.

A printed circuit board is designed to provide supply voltages of -12, 5, and 12 volts to the various electrical systems on SMAR-T. Two 12 volt 6 amp/hour lead acid batteries are connected in a parallel fashion as the input power source for the board. The battery source is further distributed to the various components on the board for power regulation and distribution purposes. The -12 volts is generated from a negative converter and is used for the RS232 communications. Two 5 volt regulators are used to supply logic power for all digital systems on-board SMAR-T. The 12 volt battery

source supplies the motor voltage, input voltage for voltage regulation, and RS232 communications. Fuses are used for protection from short circuit conditions. A single triple-throw two position switch is used to provide "on", "off", and "charge" states of operation. An external input to the board is provided to charge the batteries.

An embedded system is developed to detect obstacles around SMAR-T and is controlled by a single 68HC11A1 micro-processor which accesses its programmed code through an external EPROM. A single ultrasonic transducer is mounted to a stepper motor providing the ability to rotate the sensor through various angular positions for scanning purposes. An RS232 serial protocol and command set for the system is developed to handle all dedicated control aspects for obstacle detection. C routines are generated to enable simple function calls at a higher level for dictating the necessary involved low level control actions. Additional C routines are generated to provide a menu driven control set which controls all aspects of the embedded ultrasonic subsystem and optional graphics can be displayed to provide a visual representation of the SMAR-T's surrounding obstacles. Issues regarding the calibration and testing of the subsystem's ability to accurately detect obstacles are discussed.

A subsystem dedicated to controlling two stepper motors is developed for the mobility of SMAR-T. The subsystem utilizes a 68HC11EVB to provide the necessary low level control functions which then interfaces to a custom designed PC board for handling the high current requirements of the motors. The subsystem is sent RS232 serial commands dictating the type of control to be maintained by the subsystem. Three modes of motor operation are made available by the subsystem. Supplement C routines are written to compliment the necessary low level control sequences. A menu driven software control system capable of accessing all commands handled by the motor control subsystem is

generated.

To provide for communication between multiple robots, RF (radio frequency) serial links are utilized in the RF serial subsystem. The wireless modems allow for serial transmission to occur by modulating the serial transmissions using radio frequencies. Thus, no wires are physically connected between the robots for inter-robot communication purposes. The links operate in the 900MHz band, capable of serial transmissions up to baud rates of 19,200 with on-board error correction capabilities. A dedicated 68HC11EVB is programmed to provide interrupt driven serial detection of incoming packets for the communication between the various robots ivolved in the cooperative effort. The dedicated micro-controller can also send packets of data directly from the host system to other robots of interest. Protocols are established through which the packets of data can be sent between the given robots for the cooperative efforts. A maximum of five robots operating in a cooperative effort can be realized by the existing protocols.

Since all dedicated subsystems are controlled through RS232 serial communications and the laptop PC has only one serial port, a serial multiplexer is developed for intra-robot communication purposes. This provides the necessary link between the high level control from the laptop PC to SMAR-T's dedicated low level subsystems. The high level serial control is directed to the dedicated subsystem of interest by providing selection control from the parallel port of the laptop. Each of the three dedicated subsystems are slave devices with the master being the laptop PC. The serial multiplexer's printed circuit board is capable of handling up to a maximum of four external serial lines per board. The ability to stack four additional boards for possible future additions is provided granting serial connection to one of sixteen external devices.

Two main behaviors have been programmed to show SMAR-T's capability to properly interact with its various subsystems. SMAR-T is given the ability to follow walls, which demonstrates the host system can communicate with its lower level motor and ultrasonic transducer subsystems. SMAR-T is also given the ability to communicate with an existing VME-based mobile robot (ELVIS) to perform a type of recovery mission, a behavior which includes the additional use of the robot's RF serial subsystem for communication with an additional robot.

10.1 Future Work

All electrical systems on-board SMAR-T proved to be operating correctly with the exception of a slight problem in SMAR-T's motor control subsystem. Further modification to such a fix will provide for smoother movements and thus a large increase in top speed. Additional high level software can be added to provide added cooperative efforts of a more complex nature.

The motor control for SMAR-T utlizes two stepper motors each of which is serviced by an interrupt through a dedicated 68HC11EVB micro-controller. The 68HC11EVB can service only one such interrupt at any given point in time. Since stepper motors need to be given accurate timed pulses for given motor movements, the ability to service only one interrupt poses a problem for the motor control. In effect, SMAR-T, for some motor speeds, moves with subtle jerks. This is a control problem which needs to be further addressed if the robot is to move at faster rates. This problem does not occur when the given motor speeds are set for the same speed or a multiple of two from one another. A leap-frog effect occurs in such conditions upon which no glitches in motion are noticed.

However, should the motors be set otherwise, over time the interrupts must pass on top of each other thereby resulting in such motion glitches. For slow motor movements, such glitches can be tolerated but faster movements provide undesirable effects. The motors can be thrown out of sync resulting in lost control. Additional hardware is described in this thesis to provide such a fix to this problem. It is by no means complete, however offers the minimal amount of effort for the current motor control methods to provide such a fix.

BIBLIOGRAPHY

BIBLIOGRAPHY

[AR78] Stephen A. Allen and Tony Rossetti. On building a light-seeking robot mechanism. *Byte Magazine*, pages 24–42, August 1978.

[Bro90] Rodney A. Brooks. Elephants don't play chess. In *Proceedings of the 1990 Robotics and Autonomous Systems Conference*, North-Holland, June 1990.

[Con90] Jonathan H. Connell. *Minimalist Mobile Robotics.* Academic Press, Inc., 1990.

[GDMO92] V. Genovese, P. Dario, R. Magni, and L. Odetti. Self organizing behavior and swarm intelligence in a pack of mobile miniature robots in search of pollutants. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1575–1582, Raleigh, NC, July 1992.

[GL93] Steven G. Goodridge and Ren C. Luo. Fuzzy behavior fusion for autonomous mobile robot control. Presented at the Second International Conference on Fuzzy Theory & Technology, October 1993.

[Har93] Harris Semiconductor Corporation. *Harris Semiconductor Linear Circuits Data Book*, 1993.

[Jac93] Ronald M. Jackson. Ultrasonic radar. *Electronics Now Magazine*, pages 31–37, September 1993.

[JG92] Robert Jourdain and The Peter Norton Computing Group. *The Programmer's Problem Solver.* Brady Publishing, A Division of Prentice Hall Computer Publishing, New York, NY, 1992.

[KKR+94] Yasuo Kuniyoshi, Nobuyuki Kita, Sebastien Rougeaux, Shigeyuki Sakane, Makoto Ishii, and Masayoshi Kakikura. Cooperation by observation: The framework and basic task patterns. *IEEE*, 1994.

[Mar92] Fred Martin. The 6.270 robot builder's guide. Distributed as Class Notes for the 6.270 Course at M.I.T., December 1992.

[Mat92] Maja J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, June 1992.

[Max94] Maxim Corporation. *New Releases Data Book Volume III*, 1994.

[Mot88] Motorola Semiconductor Corporation. *Motorola Linear Interface Devices*, 1988.

[Mot89] Motorola. *M68HC11 Reference Manual.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[Sav91] Carpenter Savant, Roden. *Electronic Design: Circuits and Systems.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1991.

[Tex83] Texas Instruments Corporation. *TI Linear Integrated Circuits Data Book*, 1983.

[Tex88] Texas Instruments Corporation. *TI Linear Circuits Data Book*, 1988.

[Tra91] Transitions Research Corporation, Danbury, CT. *Proximity Subsystem User Manual*, 4.6h edition, 1991.

APPENDICES

APPENDIX A

Making Printed Circuit Boards

The use of printed circuit boards makes for nice appearance and increases the durability of a design. All designs requiring hardware were implemented on single sided printed circuit boards. The Ultrasonic Transducer Subsystem is an exception to this and uses wire-wrap techniques for the electrical connections. Different methods exist for making custom printed circuit boards, and software can be bought for this process. The user generates a schematic by placing various components on the screen and connecting the wires to them. Component selection contains size information important for pad creation when placing the actual components on the board. A large selection of various components can be chosen with such software, which gives a schematic form from which the software will generate a netlist. The netlist contains the component and wiring information and is then used by the software to automatically route the actual wires. Unsuccessful routes, if any, are able to be completed by the user after which the user sends the layout file to a company to be placed onto a PC-board. The PC-board can then be stuffed with its components and is ready for operation. This method can be very expensive, especially when only one board is needed, and the software needed to provide this capability is expensive as well. Alternative solutions require a bit more time and effort but nonetheless are very effective. The following information is for creating single sided printed circuit boards without the use of the specialized software packages.

A.1 Masks

A mask must be created to shield the copper traces from the copper. A very crude method is to simply cover the wanted traces with either special stencils, tape, or draw the traces with a permanent marker, which forms a seal or barrier for protection against the etchant; however, this method can be very tedious when dealing with large amounts of circuitry. Another method is to use ultraviolet light to create the mask for the PC-board. Copper boards used with this method have a special layer on top of the copper which when in reaction with ultraviolet light, breaks down the thin layer. This method involves the use of an additional step called developing; and the presynthesized PC-boards for use with such a scheme can come as either positive or negative type. Positive types provide protection for portions of the board which are covered during the exposure to ultraviolet light; and negative types are the reverse process which provide protection for portions unexposed during the exposure to ultraviolet light. The positive type boards are used in SMAR-T's hardware designs.

A computer drawing package allows the user to draw straight lines and easily modify the thickness of such lines. It was necessary to generate templates or pads by which the components could be placed once the layout is complete. This was a trial and error process. The templates were printed and the components were placed onto the paper and tested for fitting. Once the templates were complete, the only necessary step was to provide component layout and connections. The size of the wire connection depends on the current which will run through it. The final layout was then saved as a postscript file to be printed onto a laser printer transparency. A black permanent marker was then used to darken any of the printed areas which did not bond properly to the transparency

because such conditions leave very small holes which can be seen when the transparency is placed against a light. The three layouts used for the designs are given to scale in Figures 1.1, 1.2, and 1.3.

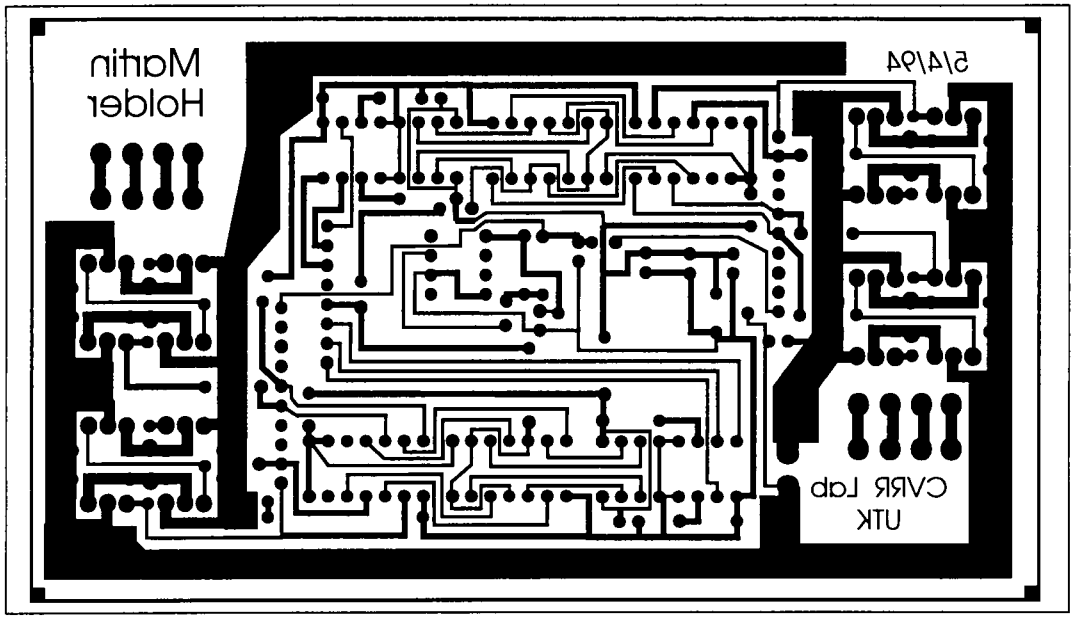

Figure 1.1: *Motor Drive controller's printed circuit board trace layout for controlling two stepper motors.*

The transparency was then placed on top of the PC-board, and a sheet of glass was placed over the transparency. The glass keeps the transparency close to the board and prevents the light from passing through the sides of the darkened areas on the transparency.

There are various sources of ultraviolet light, but a 275 Watt GE sunlamp was used as the ultraviolet light source. The sunlamp gave best results when exposed for approximately 11 minutes at a distance of one foot from the board. Care should be taken when removing the presynthesized PC-board from its dark cover prior to exposure and developing, very dim light should be used.

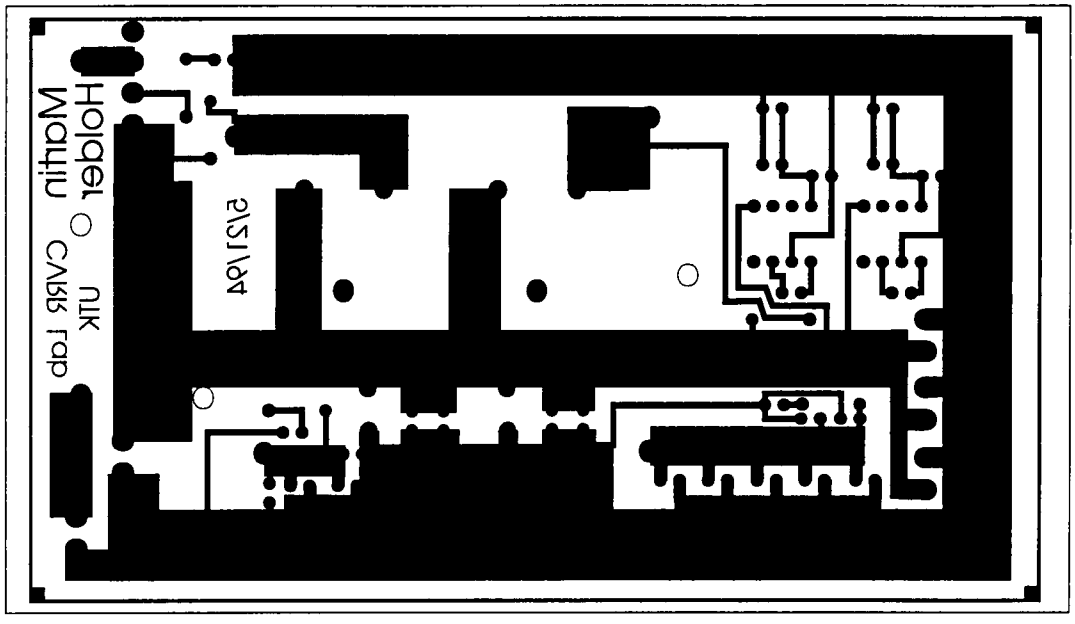

Figure 1.2: *Power Regulation/Distribution printed circuit board's trace layout.*

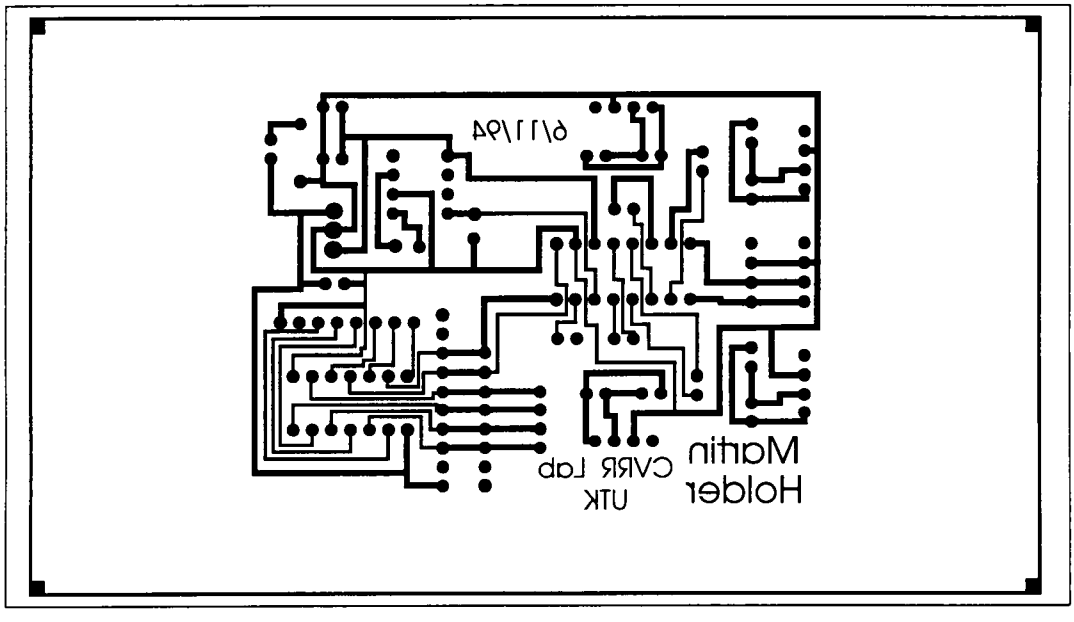

Figure 1.3: *Serial Multiplexer printed circuit board's trace layout for communication between the host PC and its dedicated subsystems.*

A.2 Developer

The presynthesized boards which use the ultraviolet light need to be developed prior to etching because the developing process strips the broken down layer exposing the copper clad beneath. Developers come in both positive and negative type, and one should use the same type as that of the presynthesized PC-board. Once exposed to the ultraviolet light, the board is placed into a solution containing 3 parts water and 1 part developer. While in this solution, the board should be constantly agitated. The layout should appear in approximately 30 seconds. Once the underlying copper clad has become very shiny indicating a removal of the broken down layer, the board should be removed from the solution and placed under cold water. This sets the mask permanently allowing the board to be ready for the etchant.

A.3 Etching

The etching process is a very simple straight forward process, but care should be taken when handling the etchant. As the name implies, etchant removes or etches away copper clad from a PC-board. The etchant is highly reactive to any type of metal, and will also stain certain objects it comes in contact with. Thus, the etchant should be placed in a plastic bowl. The PC-board, when ready for etching, is then placed into the etchant. Many factors determine the time required for the etching process, most important include etchant temperature, and etchant to copper ratio. While etching, the etchant was placed under a 100 Watt light bulb, to keep it warm. Never use a hot-plate or oven to increase the temperature of the etchant. The etchant should be agitated approximately every five minutes. One should notice the PC-board's exposed copper

turning a pinkish color during the process, which is an indication of the chemical reaction taking place on the exposed copper. After being exposed to the etchant an adequate amount of time, the copper clad will disappear and the sealed traces will be left. After the exposed copper is completely removed from the PC-board, the board should be removed and rinsed thoroughly in cold water to clean the board and stop the chemical reaction.

A.4 Removing the Mask

Once the etchant process has been completed, it is necessary to strip the mask from the board before being able to solder to it. For masks generated with simple stencils or tape, this requires the removal of such items. A stripper can be used to remove the mask generated with permanent markers or the ultraviolet light. Alcohol is a valid substitute in this process. Once stripped of the mask, a very fine grain of steel wool can be used to slightly brush over the copper to clean and give it a shiny appearance. The board is now ready for soldering.

A.5 Finishing Touches

After soldering the components to the board and finding the board to be fully operational, a last step process will ensure a long life for the board. The excess solder flux, seen typically as a burned brownish color when soldering, should be removed. Chemicals are available for this. This type of removal used to be done with an aerosol spray but tight restrictions regarding the use of such products are present. Another alternative is to use an alcohol based removal. The alcohol is simply poured over the

soldering and eats away the excess flux. This makes for very clean soldering joints. Once the copper side is shiny and free of excess solder flux, it should be sealed with a polyurethane spray. This prevents the copper from further oxidizing and provides a barrier against grime. Such a barrier likewise prevents short circuits and arcing if applicable. The board is now complete and modifications, if made, should be applied to the layout. Once the circuit is fully operational, a component layout should be made showing the necessary hardware components to be placed on the board for future reference. Figures 1.4, 1.5, and 1.6 shows the component placements for each printed circuit board made.

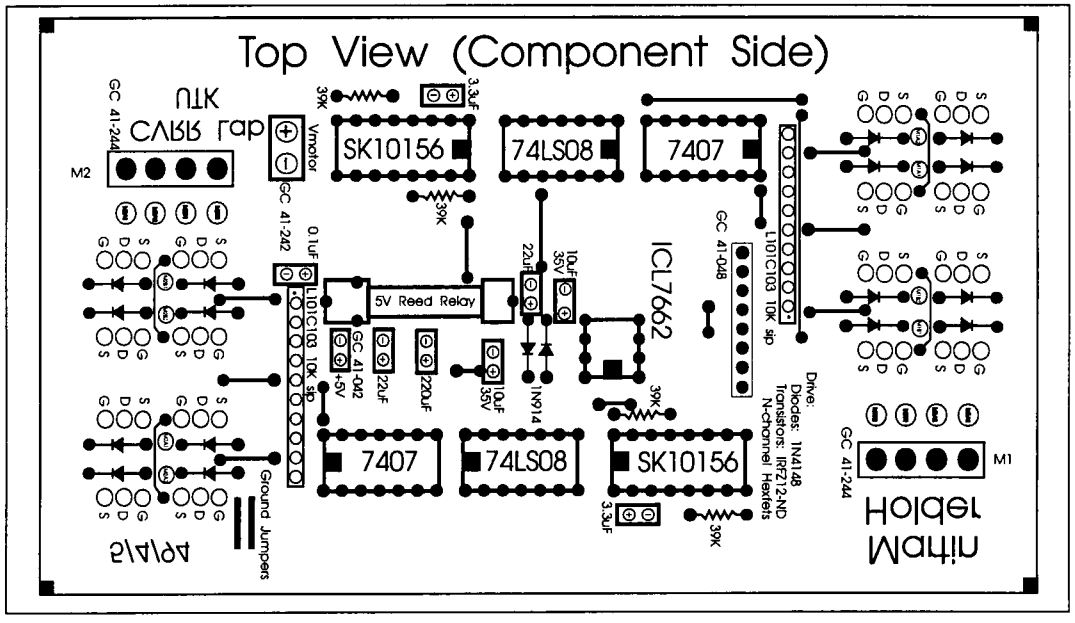

Figure 1.4: *Motor Drive controller's printed circuit board component layout for controlling two stepper motors.*

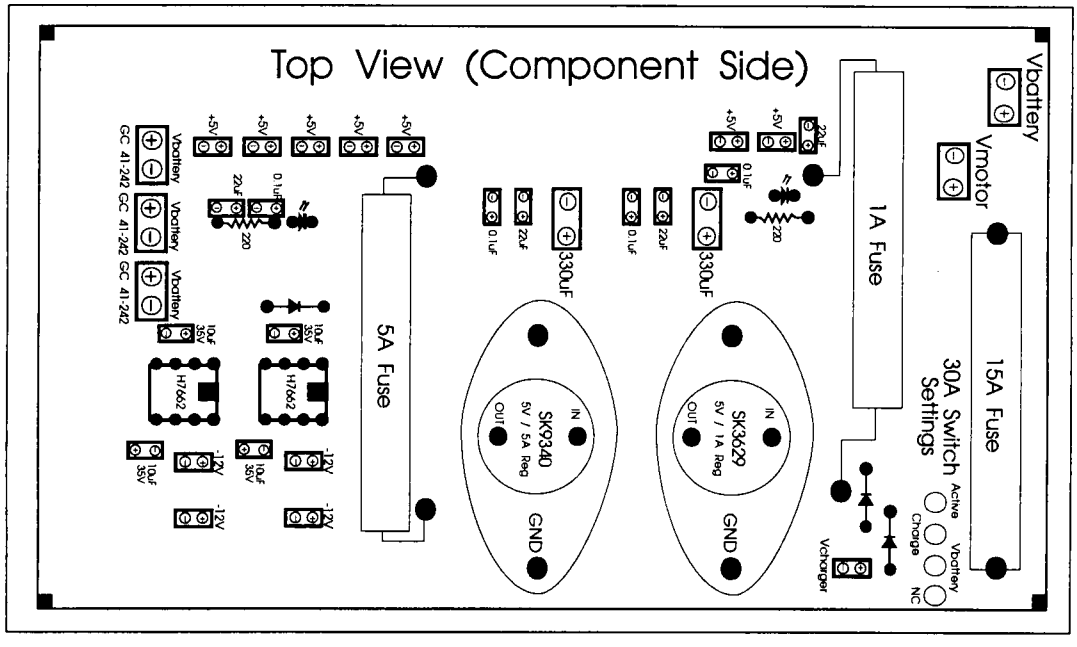

Figure 1.5: *Power Regulation/Distribution's printed circuit board component layout.*

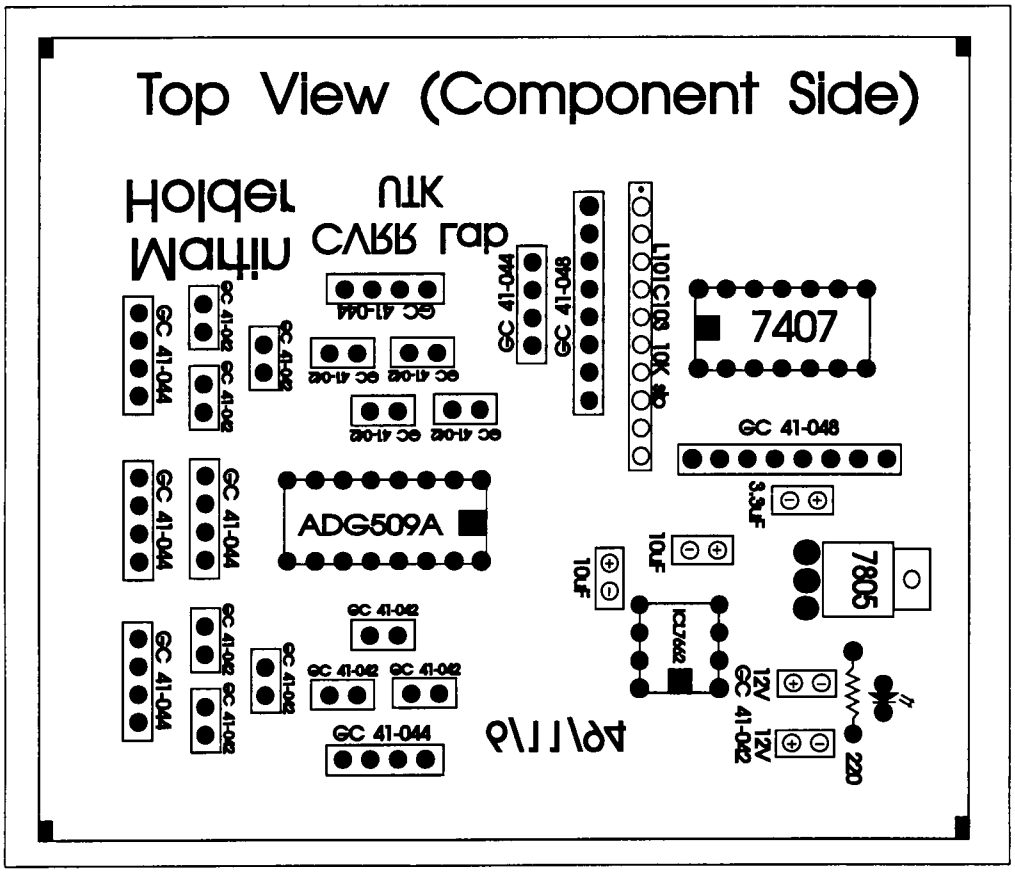

Figure 1.6: *Serial Multiplexer's printed circuit board component layout for intra-communication.*

APPENDIX B

Eprom Programming

The code information for the *68HC11* based subsystems is stored on EPROM (Erasable Programmable Read Only Memory) to provide immediate code execution upon power-up conditions. Some preliminary processes are necessary for successful EPROM implementations. This involves placing addresses in certain locations of the code for RAM variables and main program code locations. The use of the evaluation board provides a means of thoroughly debugging the programmer's code through the use of BUFFALO, an on-board monitor program, which provides a multitude of debugging features. Once proper program execution is achieved, the transformation from RAM downloaded code to permanent EPROM storage is desired. The *S19* record format should first be placed into an equivalent hex format for EPROM programmer purposes. Software code-converters are available to perform this function. The *Inlab 28* programmer and the procedures involved are further discussed to program the EPROMS.

B.1 Assembly Language Code Preparation/Alterations

The existing RAM code to be implemented on EPROM should contain an ORG $C000 mnemonic which serves to compile the code starting at hex address location $C000. The $C000 location should be changed to $6000 which alters the code location from RAM to EPROM memory addresses. It's important to initialize the stack pointer (LDS $DE00) at the beginning of program execution to ensure adequate stack space is

available for subsequent pushes of data onto the stack. Following this command, the assembly programmer should make a subroutine call to Vecinit within Buffalo (JSR $E340) for the checking of interrupt vectors and initialization thereof. *Vecinit* prevents the code from returning, by providing an indefinite loop, if a pseudo-vector is not properly initialized. The ram variables utilized within the code should be addressed with corresponding RAM memory locations starting at location $C000. Placing an *ORG $C000* statement prior to the declaration of the RAM variables realizes RAM addresses for these variables.

After completing the above mentioned changes, the assembly code should be recompiled to generate a new *S19* file providing no errors are found during compilation. The *S19* format should be translated into an equivalent hex format for use with the *Inlab 28* programmer by additional software which can be found on Motorola's bulletin board.

B.2 Inlab 28 Software/Hardware

The *Inlab 28* is used to program a variety of digital parts including PALS, EPROMS, and MACH parts through a combination of dedicated hardware and software. The software configures the hardware for the given manufacturer, chip type, etc. The hardware is connected to a personal computer via RS232 communications. An *8k* EPROM can be used for the 68HC11EVB by selecting the chips manufacturer and type within the software. If the given part type and manufacturer is not listed in the right column of the Inlab software, select a blank space and options for the various manufacturers and types will be given. Simply scroll through the selections with the cursor keys and hit

the return key to select. The memory offset for programming purposes of the EPROM should be adjusted to correspond with the EPROMS starting address in hex notation by typing the letter 'O' and responding to the prompt with the value '$6000' which is the EPROM location for the 68HC11EVB.

The software should now be properly configured for the given *8k* EPROM and the EPROM should be placed into the appropriate slot located on the programmers hardware. To verify the EPROM is blank type the letter 'V'. The software responds by asking for confirmation of the given command and, once complete, the command is executed. Should a blank EPROM exist, the software will respond with *BLANK*; however, if not blank, the software returns a calculated checksum. If a checksum is returned, the EPROM's glass window needs to be exposed to ultraviolet light for approximately 15 minutes to erase the existing data. Once again, verify the device is blank and repeat the process if necessary until the EPROM is found to be blank.

The hardware on the Inlab programmer must now be sent the hex format for the compiled assembly code. The download and upload options within the Inlab software are with respect to the hardware not the personal computer. A request to download should be given by pressing the 'F2' function key. The software will respond with a filename prompt and the user should give the appropriate filename along with the source drive and any required directory structures thereof. An example would be b:\source\test1.hex indicating the file TEST1.HEX is to be found in the SOURCE directory on the B drive. The Inlab software will access the hex file and transfer it to memory residing within the hardware. The 'F6' function key shows the coded hex values for particular memory locations within the programmer. The first memory location should be hex $6000 to correspond with the prior offset given to the Inlab software.

The EPROM is ready for programming once the code has been successfully downloaded and pressing the letter 'P' will initiate the programming mode with a confirmation prompt. Upon confirmation, the hardware begins to program the EPROM broken into three phases of operation: verification, program, and validation. The verification process ensures the EPROM is blank and will halt if not. Otherwise, the hardware will program the EPROM and then provide a means of validation by comparing the hardware's internal memory address with that of the EPROM it just programmed. For valid matches, the software returns a checksum value by responding with '*Programmed XXXX*' where XXXX represents the calculated checksum.

B.3 Final 68HC11EVB Adjustments

Once the EPROM contains the programmed code, it should be placed within the correct socket of the 68HC11EVB. Jumper *J4* is used to indicate whether code execution will begin with the monitor program, BUFFALO, or a jump to EEPROM memory location $B600 will take place. The EEPROM memory location is internal to the 68HC11A8 chip. Buffalo is used to alter the EEPROM memory location $B600 and successive following locations by including a jump to the start of EPROM at location $6000 through BUFFALO's 'MM' (*Memory Modify*) command. Three bytes should be changed at corresponding memory locations $6000, $6001, $6002. Hex value '*7E*' should be placed in memory location $6000 representing the opcode for the jump instruction. The value '*60*' should be placed in memory location $6001 and '*00*' should be placed in $6002 for a jump location of $6000. Buffalo handles the programming of EEPROM onboard the 68HC11 when writes to these locations are executed. Jumper *J4* can now

be changed; and, when powered up or reset, the programmed code should begin to run.

VITA

Martin Holder was born in Murfreesboro Tennessee on November 7, 1969. His father served in the military consequently resulting in the attendance of various schools in different locations from grades one through three. In 1980, his parents returned to Murfreesboro where he completed grades four through twelve. In 1988, he began attending the University of Tennessee Knoxville where he received his B.S. and M.S. in 1992 and 1994. He enjoys digital design with special interest for embedded microprocessor design.