

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

University of Alberta

Design, Construction, and Control of a Teleoperated Mobile Robot
Test-bed

by

L. M. Wyard-Scott



A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Electrical and Computer Engineering
Edmonton, Alberta
Fall 1999



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-47119-5

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: L. M. Wyard-Scott

TITLE OF THESIS: Design, Construction, and Control of a Teleoperated Mobile Robot Test-bed

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1999

Permission is hereby granted to the UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication rights and other rights in association with the copyright of the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed)

L. M. Wyard-Scott

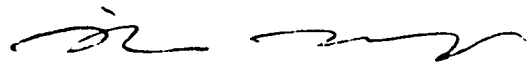
Permanent Address:
24, 2204-118 Street
Edmonton, Alberta
CANADA

Date: 24 September 1999

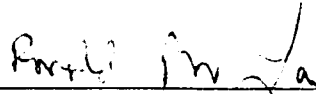
UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

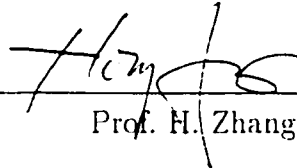
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Design, Construction, and Control of a Teleoperated Mobile Robot Test-bed** submitted by **L. M. Wyard-Scott** in partial fulfillment of the requirements for the degree of Master of Science.



Q.-H. M. Meng (Supervisor)



Prof. R. P. W. Lawson



Prof. H. Zhang

Date: 23 Sept 1999

Abstract

This thesis presents and applies design models for a teleoperated mobile robot test platform. Two models are used. The first is applied to hardware design and is derived from a test-bed's paramount requirement: flexibility. Flexibility manifests itself through modularity of the electronics, standardized connectors, and cross-platform technology selection.

The second design model, applied to software, is referred to as the Robotic Internet Platform (RIP) design paradigm. As with the hardware design model, modularity and flexibility are of utmost importance. This is achieved through selection of a platform-independent workcell software interface implemented using string data types. Code, modularly written, promotes recycling and thereby minimizes development time for research of a selected topic.

The requirements of the test-bed designed here are outlined in terms of active research areas. The reader is lead through the design process which adheres to the design models. A complete, functional, flexible, teleoperated mobile-robot system is presented.

Acknowledgements

I owe gratitude to many people for helping me complete this document and the work associated with it. Dr. Q.-H. (Max) Meng has provided me with the intellectual environment that has made the experience pleasurable. C. Ronald Kube has given me guidance and advice that coerced me into focusing and breaking the daunting task before me into achievable, logical steps. Mike Lepatski has given me unfaltering friendship for many years, and has allowed me to bend his ear many times. This is something that I don't think I could have done without, and I look forward to bending his ear many more times in the future. Keith Lo's occasionally sardonic composure has lead me to many laughs which have made the difficult times easier. Kees den Hartigh, Norman Janz, and Shaju Zacharia, system administrators for the Department of Electrical and Computer Engineering are thanked for putting up with my occasionally uninformed questions, and providing a second-to-none computing environment in which this work was done. Rod Johnson has provided me with a lot of guidance in the workings of the University machine, and on many technical aspects which I needed to conquer. Michelle Lock has helped me through friendship and thorough knowledge of requirements of a graduate student. Glen Walker has made his vast knowledge of C and TCL available to which end the software design became easier. Hien Dang, Fraser Murch, Christy Dolynchuk have all spurred me onward to completion by assisting in tasks which I considered to be unrewarding. Thanks to Jonathan Baldwin whose generosity made my task a little easier. Mike Chen, James Smith, and Rod Frey have provided guidance and occasional pep-talks under the guise of friendship. I am very grateful to Dr. R.P.W. Lawson who never hesitated to share his sensibility and wisdom. Loretta Ludwig has spurred me onward by looking after everyday mundane tasks which the work load occasionally forced me to neglect. She has also helped me realize why this work is so important to me. More satisfaction is drawn from the process by this realization. Last, but not least, is my entire family. Without their support, the entire process would not have been possible.

Contents

1	Introduction	1
1.1	Organization	2
2	Teleoperation Systems	3
2.1	Introduction	3
2.2	A Typical Teleoperation Platform	4
2.3	Telerobotics Research Areas	5
2.3.1	Control	6
2.3.2	Communication	7
2.3.3	Telerobotic System Software	7
2.3.4	System Architecture	8
2.3.5	User-Interface Functionality and Human Factors	9
2.3.6	Sensing Techniques	10
2.4	Summary	11
3	Objectives and System Design Models	12
3.1	Introduction	12
3.2	Functional Specifications	13

3.3	Electronic Hardware Design: The Test-Bed Model	14
3.4	Software Interface Design Model: The RIP Paradigm	16
3.4.1	Selection of the Operating System	18
3.4.2	Design Implications	18
3.4.3	User Interface Development Considerations	19
3.4.4	Workcell Control Engine Design Implications	20
3.5	Summary	21
4	System Overview	22
4.1	Introduction	22
4.2	The Control Station	23
4.2.1	Control Station Operating System	25
4.3	The Workcell PC Motherboard	26
4.3.1	Workcell PC Operating System	27
4.4	Communication Technology	27
4.5	The Client-Server Model	28
4.6	Summary	28
5	Modular Workcell Hardware Design	30
5.1	Introduction	30
5.2	Module Overview	31
5.2.1	Serial Level Conversion PCB	33
5.2.2	Parallel-Port Expansion PCB	34
5.2.3	PC-to-MCU Shared Memory/Serial Network Control PCB . .	34
5.2.4	MC6HC11 Microcontroller (MCU) PCB	35
5.2.5	MCU-to-MCU Shared Memory PCB	38

5.2.6	SONAR Module Interface PCB	39
5.2.7	Near-Range IR Proximity Sensor PCB	40
5.2.8	Bump Sensor PCB	41
5.2.9	Navigation Hardware PCB	42
5.3	Mechanical Structure Overview	42
5.3.1	Structure	43
5.3.2	Sensor Placement	44
5.4	Summary	46
6	Modular Software Design	48
6.1	Introduction	48
6.2	Microcontroller Layer Software	49
6.2.1	Development Platform: Interactive-C	49
6.2.2	Software Operation	51
6.3	Server (Workcell Control Engine) Software	56
6.3.1	Development Platform: Tool Command Language	57
6.3.2	Software Operation	58
6.3.3	Programming Interface	63
6.4	Client (Control-Station) Software and User Interface	64
6.4.1	Development Platform: TCL/TK and C	64
6.4.2	Socket Control Module	65
6.4.3	Navigation Console Module	66
6.4.4	Message Module	67
6.4.5	Sensor Status Display Module	69
6.4.6	Mapper Module	71

6.5	Summary	72
7	Results and Demonstrations	75
7.1	Introduction	75
7.2	Setpoint Movement Test	75
7.3	Bump Sensor Test	78
7.4	Infrared Sensor Test	78
7.5	SONAR Sensor Test	78
7.6	Summary	82
8	Conclusion	83
8.1	Summary of Contribution	83
8.2	Evaluation of the Design and Design Technique	84
	Bibliography	85
	Appendix	
A	MC68HC11 Microcontroller PCB	90
A.1	Introduction	90
A.2	Operation	90
A.2.1	The HC11 Microcontroller IC	91
A.2.2	Control-Line Circuitry	91
A.2.3	Address Latching	92
A.2.4	Static RAM	92
A.2.5	Address Decoding	93

A.3	User-settable Options	93
A.3.1	Reset Switch: SW1	93
A.3.2	CPU Mode Control: J2	94
A.3.3	Power Select: JP1	94
A.4	Connector Descriptions	94
A.4.1	Bus Connector: J1	94
A.4.2	Power Connector: J3	97
A.4.3	Port Connector: J4	97
A.4.4	Piezo Buzzer Connector: J5	98
A.4.5	Serial Data Connector: J6	99
A.5	Future Improvements	99
A.6	Schematic Diagrams and Board Layout	100
A.6.1	Part List	100
B	Serial-Level Conversion PCB	111
B.1	Introduction	111
B.2	Connector Descriptions	111
B.2.1	PC RS-232 Connector: J1	111
B.2.2	HC11 TTL Serial Connector: J2	112
B.3	Future Improvements	112
B.4	Schematic Diagrams and Board Layout	113
B.4.1	Part List	113
C	Parallel Port Expansion PCB	117
C.1	Introduction	117
C.2	Operation	118

C.2.1	Register Descriptions	118
C.3	User-settable Options	121
C.3.1	Data Bus/Output Port Select: JP1	121
C.3.2	Data Bus Input Enable: JP2	121
C.4	Connector Descriptions	122
C.4.1	Power Connector: J1	122
C.4.2	Digital I/O Connector: J2	122
C.4.3	Digital O/P Connector: J3	124
C.4.4	Parallel Port Connector: J4 and J5	124
C.5	Design Limitations	125
C.6	Future Improvements	126
C.7	Schematic Diagrams and Board Layout	127
C.7.1	Part List	127
D	PC-to-MCU Shared Memory PCB	135
D.1	Introduction	135
D.2	Operation	135
D.2.1	Shared Memory Interface	136
D.2.2	Serial Network Control Hardware	136
D.2.3	Logic-Level Conversion	137
D.3	Connector Descriptions	137
D.3.1	Power Connector: J1	137
D.3.2	MCU Bus Connector: J2–J4	138
D.3.3	Digital I/O Connector: J5	138
D.4	Design Limitations	138

D.5	Future Improvements	138
D.6	Schematic Diagrams and Board Layout	139
D.6.1	Part List	139
E	MCU-to-MCU Shared Memory PCB	149
E.1	Introduction	149
E.2	Operation	149
E.3	Connector Descriptions	150
E.3.1	MCU Bus Connectors: J2-J4	150
E.4	Design Limitations	150
E.5	Future Improvements	150
E.6	Schematic Diagrams and Board Layout	151
E.6.1	Part List	151
F	Navigation Hardware PCB	159
F.1	Introduction	159
F.2	Operation	160
F.3	Connector Descriptions	161
F.3.1	Encoder Connectors: J1, J2	161
F.3.2	Battery-level Sensing Connector: J3	161
F.3.3	Motor Power Connector: J4	162
F.3.4	Port Connector: J5	162
F.4	Future Improvements	162
F.5	Schematic Diagrams and Board Layout	162
F.5.1	Part List	163

G	Near-Range IR Proximity Sensor PCB	171
G.1	Introduction	171
G.2	Operation	172
G.2.1	Digital Output (Emitter Control)	172
G.2.2	Analog Input (Detector Reading)	172
G.3	Connector Descriptions	174
G.3.1	Port Connector: J1	174
G.3.2	IR Pair Power Connectors: J2-J5	174
G.3.3	J5-J37: IR Pair Signal Connectors	174
G.4	Design Limitations	175
G.5	Future Improvements	175
G.6	Schematic Diagrams and Board Layout	175
G.6.1	Part List	175
H	Bump Sensor PCB	185
H.1	Introduction	185
H.2	Operation	185
H.3	Connector Descriptions	186
H.3.1	Port Connector: J1	186
H.3.2	Bump Sensor Connectors: J2-J17	187
H.4	Design Limitations	187
H.5	Future Improvements	187
H.6	Schematic Diagrams and Board Layout	188
H.6.1	Part List	188

I	Sonar Module Interface PCB	195
I.1	Introduction	195
I.2	Operation	195
I.3	Connector Descriptions	197
I.3.1	Stepper Motor Connector: J1	197
I.3.2	Polaroid Module Connector: J2, J3	197
I.3.3	Motor Power Connector: J4	199
I.3.4	MCU Port Connector: J5	199
I.3.5	Limit-Switch Connector: J6	199
I.3.6	Photoresistor Connectors: J7–J10	199
I.4	Future Improvements	200
I.5	Schematic Diagrams and Board Layout	200
I.5.1	Part List	200
J	Code Documentation	208
J.1	Microcontroller Code	208
J.1.1	File message.c	208
J.1.2	File memory.c	209
J.1.3	File spi.c	209
J.1.4	Navigation MCU Code	210
J.1.5	Proximity Detector MCU Code	213
J.1.6	SONAR MCU Code	213
J.2	Server (Workcell Control Engine) Code	215
J.2.1	ANSI C Mapping Code	215
J.2.2	TCL Mapping Code	218

J.2.3	ANSI C Parallel Port I/O Code	220
J.2.4	TCL Shared Memory Interface Code	223
J.2.5	Server Module Code	224
J.3	Client (Control Station) Code	225
J.3.1	Socket Control Module Code	226
J.3.2	Navigation Console Module Code	226
J.3.3	Message Module Code	227
J.3.4	Sensor Status Display Module Code	227
J.3.5	Mapper Module Code	228
J.3.6	Functions	228
K	Power-Up Steps	233
K.1	PPP Connection to the Workcell	233
K.2	Downloading the MCU Layer Code	234
K.3	Starting the Workcell Server	235
K.4	Starting the Control Station Client	236
	Glossary	237

List of Tables

Table

6.1	Allocation of the shared memory.	53
A.1	Address decoding (74HC138) ranges.	93
A.2	MCU board J2 jumper position definitions.	94
A.3	MCU board JP1 jumper position definitions.	94
A.4	MCU Board bus connector pinouts (J1).	96
A.5	MCU Board power connector pinouts (J3).	97
A.6	MCU Board port connector pinouts (J4).	98
A.7	MCU Board piezo buzzer connector pinouts (J5).	99
A.8	MCU Board serial data connector pinouts (J6).	99
A.9	Microcontroller board parts list (3 constructed).	100
B.1	Serial-level Conversion PCB connector J1 pinouts.	112
B.2	Serial-level Conversion PCB connector J1 pinouts.	112
B.3	Serial-Level Conversion board parts list.	113
C.1	Parallel port address definitions.	118
C.2	Parallel port expansion PCB register definitions.	119

C.3	Parallel port expansion PCB CTRL register (register 4) bit definitions.	120
C.4	Parallel port expansion PCB UCx register (registers 5–7) bit definitions.	120
C.5	Parallel port expansion PCB connector J1 pinouts.	122
C.6	Parallel port expansion PCB connector J2 pinouts.	124
C.7	Parallel port expansion PCB connector J3 pinouts.	124
C.8	Parallel port expansion PCB connector J4 and J5 pinouts.	125
C.9	Parallel port expansion board parts list.	127
D.1	PC-to-MCU Shared Memory PCB MCU memory base locations and PC selection signal mnemonics.	136
D.2	PC-to-MCU Shared Memory PCB MCU memory base locations and PC selection signal mnemonics.	137
D.3	PC-to-MCU Shared Memory PCB connector J1 pinouts.	138
D.4	PC-to-MCU Shared Memory board parts list.	139
E.1	MCU-to-MCU Shared Memory PCB Memory address bases.	150
E.2	MCU-to-MCU shared memory board parts list.	151
F.1	Motor control signals.	160
F.2	Navigation Control Hardware PCB connector J1, J2 pinouts.	161
F.3	Navigation Control Hardware PCB connector J3 pinouts.	161
F.4	Navigation Control Hardware PCB connector J4 pinouts.	162
F.5	Navigation Control Hardware board parts list.	163
G.1	IR Pair input connector-to-bit correspondence and analog input.	173
G.2	Near-Range Proximity PCB connectors J2–J5: sensor power connections.	174

G.3	Near-Range Proximity PCB connectors J5–J37: sensor signal connections.	174
G.4	Near-range Proximity Detector board parts list.	175
H.1	Digital input connector-to-bit correspondence.	186
H.2	Bump Sensor PCB connectors (J2–J17).	187
H.3	Bump Sensor board parts list.	188
I.1	Sonar Module Interface PCB SPI bit definitions.	196
I.2	Sonar Module Interface PCB stepper motor connector J1 pinouts. . .	197
I.3	Sonar Module Interface PCB Polaroid Module 0 connector J2. . . .	198
I.4	Sonar Module Interface PCB Polaroid Module 1 connector J3. . . .	198
I.5	Sonar Module Interface PCB motor power connector J4 pinouts. . .	199
I.6	Sonar Module Interface PCB Limit-Switch connector J6 pinouts. . .	199
I.7	Sonar Module Interface PCB CdS Cell connector pinouts: J7–J10. . .	200
I.8	Sonar Module Interface board parts list.	201

List of Figures

Figure

2.1	A top-level block diagram of a typical teleoperation system.	4
3.1	A block diagram of a system under the Robotic Internet Platform design paradigm	16
4.1	A block diagram of the test platform with some details filled in under the guidance of the two design models described in Chapter 3.	23
4.2	The telerobotic system layout.	24
5.1	Location of the Serial Conversion PCB in the test-bed.	33
5.2	Block diagram of the Serial Conversion board electronics.	34
5.3	Block diagram of the Parallel-Port Expansion board electronics. . . .	35
5.4	Block diagram of the PC-to-MCU Shared Memory/Serial Conversion board.	36
5.5	Location of the HC11 microcontroller PCBs in the test-bed.	37
5.6	Block diagram of the MCU board electronics.	38
5.7	Block diagram of the MCU-to-MCU Shared Memory board electronics.	39
5.8	Block diagram of the SONAR Module Interface board electronics. . .	40

5.9	Block diagram of the Near-Range Proximity Sensor board electronics.	41
5.10	Block diagram of the Bump Sensor board electronics.	42
5.11	Block diagram of the Navigation Hardware PCB electronics.	43
5.12	Side view of the mobile robot workcell.	44
5.13	Photograph of the mobile robot workcell.	45
5.14	Positions of the workcell sensors.	46
5.15	Graphical depiction of sensing distances.	47
6.1	Graphical depiction of the test bed development platforms.	49
6.2	The slaved-PI velocity control loop for the drive motors.	55
6.3	Graphical depiction of the five user interface modules.	66
6.4	Screenshot of the Socket Control module UI.	67
6.5	Screenshot of the Navigation Console module UI.	68
6.6	Screenshot of the Message module UI.	69
6.7	Screenshot of the Sensor Status Display module UI.	70
6.8	Screenshot of the Mapper module UI.	73
7.1	The setpoint search algorithm in action (1 of 2).	76
7.2	The setpoint search algorithm in action (1 of 2).	77
7.3	Bump sensor test actual and mapped environments.	79
7.4	IR sensor test actual and mapped environments.	80
7.5	SONAR sensor test actual and mapped environments.	81
A.1	Microcontroller board schematic (1 of 7).	101
A.2	Microcontroller board schematic (2 of 7).	102
A.3	Microcontroller board schematic (3 of 7).	103
A.4	Microcontroller board schematic (4 of 7).	104

A.5	Microcontroller board schematic (5 of 7).	105
A.6	Microcontroller board schematic (6 of 7).	106
A.7	Microcontroller board schematic (7 of 7).	107
A.8	Microcontroller board part placement diagram.	108
A.9	Microcontroller board top-layer copper foil pattern. Not to scale. . . .	109
A.10	Microcontroller bottom-layer copper foil pattern (through-board view). Not to scale.	110
B.1	Serial-Level Conversion schematic (1 of 1).	114
B.2	Serial-Level Conversion part placement diagram.	115
B.3	Serial-Level Conversion top-layer copper foil pattern. Not to scale. . .	115
B.4	Serial-Level Conversion bottom-layer copper foil pattern (through-board view). Not to scale.	116
C.1	Parallel port interface card schematic (1 of 4).	128
C.2	Parallel port interface card schematic (2 of 4).	129
C.3	Parallel port interface card schematic (3 of 4).	130
C.4	Parallel port interface card schematic (4 of 4).	131
C.5	Parallel port interface card part placement diagram.	132
C.6	Parallel port interface card top-layer copper foil pattern. Not to scale.	133
C.7	Parallel port interface card bottom-layer copper foil pattern (through- board view). Not to scale.	134
D.1	PC-to-MCU Shared Memory PCB schematic (1 of 6).	140
D.2	PC-to-MCU Shared Memory PCB schematic (2 of 6).	141
D.3	PC-to-MCU Shared Memory PCB schematic (3 of 6).	142
D.4	PC-to-MCU Shared Memory PCB schematic (4 of 6).	143

D.5	PC-to-MCU Shared Memory PCB schematic (5 of 6).	144
D.6	PC-to-MCU Shared Memory PCB schematic (6 of 6).	145
D.7	PC-to-MCU Shared Memory PCB part placement diagram.	146
D.8	PC-to-MCU Shared Memory PCB top-layer copper foil pattern. Not to scale.	147
D.9	PC-to-MCU Shared Memory PCB bottom-layer copper foil pattern (through-board view). Not to scale.	148
E.1	MCU-to-MCU Shared Memory PCB schematic (1 of 4).	152
E.2	MCU-to-MCU Shared Memory PCB schematic (2 of 4).	153
E.3	MCU-to-MCU Shared Memory PCB schematic (3 of 4).	154
E.4	MCU-to-MCU Shared Memory PCB schematic (4 of 4).	155
E.5	MCU-to-MCU Shared Memory PCB part placement diagram.	156
E.6	MCU-to-MCU Shared Memory PCB top-layer copper foil pattern. Not to scale.	157
E.7	MCU-to-MCU Shared Memory PCB bottom-layer copper foil pattern (through-board view). Not to scale.	158
F.1	Navigation Control Hardware schematic (1 of 4).	164
F.2	Navigation Control Hardware schematic (2 of 4).	165
F.3	Navigation Control Hardware schematic (3 of 4).	166
F.4	Navigation Control Hardware schematic (4 of 4).	167
F.5	Navigation Control Hardware part placement diagram.	168
F.6	Navigation Control Hardware top-layer copper foil pattern. Not to scale.	169
F.7	Navigation Control Hardware bottom-layer copper foil pattern (through- board view). Not to scale.	170

G.1	Near-Range Proximity PCB schematic (1 of 6).	176
G.2	Near-Range Proximity PCB schematic (2 of 6).	177
G.3	Near-Range Proximity PCB schematic (3 of 6).	178
G.4	Near-Range Proximity PCB schematic (4 of 6).	179
G.5	Near-Range Proximity PCB schematic (5 of 6).	180
G.6	Near-Range Proximity PCB schematic (6 of 6).	181
G.7	Near-Range Proximity PCB part placement diagram.	182
G.8	Near-Range Proximity PCB top-layer copper foil pattern. Not to scale.	183
G.9	Near-Range Proximity PCB bottom-layer copper foil pattern (through-board view). Not to scale.	184
H.1	Bump Sensor PCB schematic (1 of 3).	189
H.2	Bump Sensor PCB schematic (2 of 3).	190
H.3	Bump Sensor PCB schematic (3 of 3).	191
H.4	Bump Sensor PCB part placement diagram.	192
H.5	Bump Sensor PCB top-layer copper foil pattern. Not to scale.	193
H.6	Bump Sensor PCB bottom-layer copper foil pattern (through-board view). Not to scale.	194
I.1	Sonar Module Interface board schematic (1 of 3).	202
I.2	Sonar Module Interface board schematic (2 of 3).	203
I.3	Sonar Module Interface board schematic (3 of 3).	204
I.4	Sonar Module Interface board part placement diagram.	205
I.5	Sonar Module Interface board top-layer copper foil pattern. Not to scale.	206
I.6	Sonar Module Interface board bottom-layer copper foil pattern (through-board view). Not to scale.	207

Chapter 1

Introduction

The objective of the research presented here is to develop a complete, modifiable teleoperated mobile robot platform for the purpose of control algorithm and interface testing.

The design is carried out under the guidelines of two models: one which outlines necessities for a test-bed, and another (the Robotic Internet Platform (RIP) design paradigm) which dictates the form of software interfaces between the layers of the system.

A comprehensive library of software routines are developed to minimize system development time for future researchers. This software implements the robust, extensible interfaces between the abstract layers of the system in accordance to the second of the two design models.

As a test of the resulting systems capabilities, a complete, functional system is developed: from user-interface to remote workcell actuators. The reader is led through the development of the design models and application of them.

1.1 Organization

The organization of this thesis is as follows. Chapter 2, Teleoperation Systems, is an overview of existing teleoperation platforms and active research areas in teleoperation. This chapter is used to derive the design objectives and system design models of Chapter 3. The next three chapters contain the design of the teleoperation system resulting from the design models: Chapter 4, System Overview, is a top-level description of the system. Chapter 5, Modular Workcell Hardware Design, describes the workcell hardware and the motivation used to achieve it, and Chapter 6, Modular Software Design, is an overview of the system software and the workcell software interface. Upon completion of the system design, Chapter 7, Results and Demonstrations, the test-bed's functionality is demonstrated. Lastly, Chapter 8 evaluates the success of the design models and the system yielded from them.

A set of appendices describe the modular workcell hardware in detail. These are presented to aid future researchers in reconfiguration of the platform. An appendix with software documentation is also presented to aid the discussion in Chapter 6. These appendices, though verbose, are a necessity for an effective test bed platform.

Chapter 2

Teleoperation Systems

2.1 Introduction

In this chapter, the structure of a typical teleoperated system is presented along with the definitions of several terms that are encountered in this very active area of research. There are several specific facets of teleoperation systems that are of paramount interest. These are discussed using a survey of several existing teleoperation systems. There are also many active areas of research in **mobile** robotics. These include Artificial Intelligence (AI) topics, such as subsumption architecture and neural networks, and topics to do with navigation, such as path-planning and localization techniques. Although it is desirable to create a test-bed that will be able to research these topics, they are not covered here.

The next chapter utilizes the information derived here to develop two design models and design objectives for a test-bed system used for research.

2.2 A Typical Teleoperation Platform

Fig. 2.1 shows typical division of a teleoperation system into Control Station and Remote Workcell blocks. The Control Station is where the user interacts through a User Interface (UI) to take control over the Remote Workcell. Communication between the control station and the workcell occurs through a bidirectional channel. Directives are issued from the control station to the workcell (left-to-right data flow), and feedback about the state of the workcell and information about its surrounding environment is related from the workcell back to the control station (right-to-left data flow).

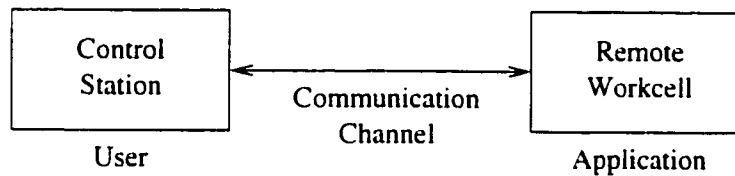


Figure 2.1: A top-level block diagram of a typical teleoperation system. The system is physically divided into a Control Station and a Workcell, operating remotely. Communication between the two occurs through some sort of channel.

The following definitions are commonly encountered in this field of study. The definitions are expressed in terms of a top-level view of the system, shown in Fig. 2.1.

Teleoperation is the process by which control station initiated directives are executed on a remote workcell. The amount of feedback that is returned to the control station from the can vary from nothing (open-loop) to comprehensive (detailed). This definition is apparently context-sensitive. If used when referring to a telepresence system, teleoperation may mean that the user actually performs the task that the workcell is to execute.

Telepresence is the discipline where feedback from the workcell to the control station is comprehensive. The feedback is so thorough that the user “feels” that they are physically located at the remote workcell’s location. The actions that the user takes at the control station are somehow mimicked by the workcell.

Telerobotics has varying definitions. For instance, [PPA94] states that a telerobotics system is one wherein the user at the control station actually performs the action taken by the workcell. Other literature, such as [JP94] or [LMR94] assume that telerobotics is the field of study which is a superset of teleoperation, telepresence, and teleprogramming: any system which has a control station, communication link, and a remote workcell. This second definition is the one used throughout this document.

Teleprogramming is the act of changing the operation of a workcell from the control station through transmission of a program. Programming may occur at run-time, or when the workcell is in an idle mode. However the programming is performed, the syntax that the program is defined with is referred to as the teleprogramming language.

Distinction between the terms is difficult and depends upon the specific research area or system being addressed.

2.3 Telerobotics Research Areas

This section is an overview of telerobotic research areas. The goal is to be able to design a test-bed that can be used for research in any of these areas. This analysis of existing work will be used to develop the system design models and functional

objectives presented in the next chapter.

Dividing the research into these areas is synthetic since they are all intertwined. For instance, the form of a user interface could be classified as control research.

2.3.1 Control

A workcell is often given reflexive behaviours to preserve itself. For instance, a mobile workcell is typically not given the option to drive over a cliff, even if the user (via the control station) tells it to do so. This area addresses the topic of where control-loops are closed: within the workcell itself (local control), or through the control-station (remote control).

Certain applications may require that control be transferred from the control-station (remote) to the workcell (local), or vice versa. This type of transfer is not necessarily trivial. Often, cases will arise where the workcell may behave unpredictably when control is transferred.

If a workcell is performing an application-critical task under control from the control station and the communication link is lost, some technique needs to be in place to rationalize the workcell's actions. Since the control-loop incorporating the control station is effectively broken, the workcell could become unstable.

Teleautonomous guidance [BK90] is a scheme in which the workcell has collision-avoidance (or similar) control loops closed on the workcell and an external loop closed by the user's perception and action at the control station. When activated, the collision-avoidance loop is closed, otherwise remaining open. This approach can also be seen in [TBGX94], [HM94a] (approached through a teleprogramming paradigm), and [BAXTJ96]. "Shared autonomy" [MA94] is a similar method, but assumes that both control loops are closed simultaneously. The sharing of autonomy with the

workcell makes the system safer, dependable, and reduces interaction required of the user at the control station [SJ94].

2.3.2 Communication

Research focusing on communication examines the operation of a system when there is transmission delay between the control station and the workcell.

Communication between the control station and the workcell can be achieved in several different ways: umbilical, wireless modem, wireless Ethernet, infrared, microwave, and satellite, among others. Establishing such a link may be simple, or involve multiple “hops” as in [Leo95] or [BFKS94].

Control-loops behave poorly under delay. The more accurate the control needs to be, the higher the required bandwidth. If a control-loop is closed via the control station, the bandwidth and transmission delay of the communication channel are of interest.

To compensate for a communication delay, predictive models can be used as in [HM94b], [Say96], [HLF94] and [BBZ98].

2.3.3 Telerobotic System Software

Communication between control station and workcell must follow a protocol. The structure of the “language” that is used to communicate between the two components of the system is referred to as a **teleoperation language**. This language can be embedded into an operating system, such as CHIMERA, developed at Carnegie-Mellon and utilized by [NX94] to control a manipulator used in Space Station Freedom. The syntax of the communication may also be implemented in another language. See, for instance, Steele and Backes [SB94].

It can be argued that a teleoperation language is simply a teleprogramming language in its most basic form. Teleprogramming typically involves specifying a task or a mission that the workcell is to complete.

In task-level teleprogramming [PPA94], in addition to a description of the task that the workcell is to perform, a program can be written to include information that the workcell is unable to obtain itself (perhaps a-priori knowledge of the workcell's environment).

Programming and Interpreted Languages Of actions for Telerobotics (PILOT) [LMR94], is a high-level interpreted visual language which is able to accommodate event-driven actions. Its object-oriented design links three software layers (mission, control-task, and servo-control) using standard language primitives (sequential, iterative, conditional, and parallel). PILOT is not intended to process data from sensors: operations (other than comparisons) are not supported. This would be a disadvantage for a test bed application since research into areas such as sensor fusion would not be supported.

2.3.4 System Architecture

Telerobotic system architecture refers to how the software is written, or how the hardware is designed. Since hardware and software are co-dependent, descriptions typically address both.

Typically, there is a physical distinction between a telerobotic system's control station and workcell. However, from a software perspective a division between the two is not so obvious. For instance, all of the system software could be implemented on the workcell; only the UI is displayed on the "empty" control station, perhaps using a web-browser or X-window display commands. Where the software division is

made depends not only upon the application, but also upon the other technologies (such as the communication medium) that the system employs.

The NASA/NIST Standard Reference Model (NASREM) [Lum94] suggests that a telerobot system should have a parallel-processing structure to meet real-time performance requirements. The software architecture consists of six levels which range from a synchronous servo level to an asynchronous mission-defining level. NASREM does not provide a methodology for designing the system hardware.

Another architecture project in [JP94] outlines the need for a standard for industrial remanufacturing telerobotic systems. Backes et al. [BBL⁺94] implement a work-cell controller system entitled Modular Telerobot Task Execution System (MOTES). It is an interpreted language which offers concurrent control over control modules. A three-layer structure is used in [Cro89]: the top layer is an interpreter which handles asynchronous commands, the middle layer is a "virtual" workcell, and the bottom layer is the interface to the actual vehicle. The European Space Agency follows the guidelines of a unified control architecture for planetary rovers. Two projects which make use of this system are [RVR98] and [SLMV98].

Regardless of the architectures, there is one undeniable fact: there is movement towards decoupling and modularizing of telerobotic system hardware and software to increase flexibility and promote re-use of code and hardware.

2.3.5 User-Interface Functionality and Human Factors

For effective control over the remote workcell, the interface to the user at the control station will need to be effective. Poor interfaces will lead to poor control. This research explores the design of effective UIs and the methods used to evaluate them.

UIs are not only used to provide a means for the user to control the workcell, but also to provide a medium for the feedback from the workcell to be delivered to the user. Effectiveness of the UI can be evaluated by the number of successful task completions [CRKW96] or even through the EMG signals of the system user [RCMT94].

To help a user maintain control of the workcell, the UI can present the feedback in varying manners. Lloyd et al. [LBPL97] and McMaster et al. [MNBF94] present the workcell and its environment as a computer-generated graphical image. This approach is useful in situations where the communication bandwidth is insufficient to transmit, or the environmental conditions are too poor to capture, quality video signals. Another approach is to super-impose computer-generated graphics over a potentially low-quality video stream as in [BFKS94] or [MCC92].

Another possibility is to have the feedback to the UI completely simulated. Although this is no longer a telerobotic system, it may be useful for operator training [MS94]. To this end, unrealistic worlds could be shown to research perception and how it applies to teleoperation systems [RKSG94].

UIs can be delivered on different platforms: through a WWW browser, application-level programs in a Windowing environment, or perhaps even a text-only shell environment.

2.3.6 Sensing Techniques

How a workcell senses the environment around it is important for effective control. In telepresence applications, sensing is important as it is through it that the remote world is represented to the user at the control station.

Even to implement autonomous behaviours (to close the workcell control loop on the workcell itself), there is need for sensors to relay information about the envi-

ronment back to the control architecture. This sensing can be as simple as a bump switch or as complex as a multi-camera video system.

One may be inclined to think that more sensor information is better. However, the limitations of the CPU's speed or the communication channel's bandwidth will quickly become apparent. The process of sensor fusion [LK89] [DW87] can help the system reduce the amount of data to process by by algorithmically isolating the relevant portions of sensor data by linking together data from more than one (or more than one type) of sensor. The algorithms may involve simple modeling of sensor response [WSM96] or involve the operator in the process [YMP+94].

2.4 Summary

There is a lot of activity in the field of telerobotics research. This chapter has outlined basic terms found in, and examined research directions in the field of telerobotics. Control techniques, communication, system software (teleprogramming and teleoperation languages), system (software) architecture, user interfaces, and sensing techniques are all areas that a test bed should be able to research. The next chapter presents design models which emphasize the flexibility and modularity that are required of a system in order to achieve this goal.

Chapter 3

Objectives and System Design Models

3.1 Introduction

This chapter provides a “specification” for the desired operation of the test-bed. In order to meet these specifications and to obtain a system that can successfully be used to research the areas outlined in the last chapter, it is necessary to concentrate on the fact that the system is a test-bed. Since it is impossible to predict with certainty the areas in which a test-bed will be applied, flexibility becomes the key issue.

Flexibility is required not only in the hardware design but also in the software. The model used for this design has been coined the Robotic Internet Platform (RIP) design paradigm and is presented in Section 3.4.

3.2 Functional Specifications

The teleoperated mobile-robot test-bed is to have the ability to research the areas outlined in Chapter 2. These areas are restated here in terms of high-level specifications. The resulting system is to have the following features:

- (1) Ability to implement autonomous or reflexive behaviours on the workcell (control level research);
- (2) Ability to transfer control between the workcell (autonomous control) and the control station (user-initiated control) (control transfer research);
- (3) Interruptable communication channel (communication research);
- (4) Delayable communication channel (communication research);
- (5) Changeable communication medium (communication research);
- (6) Flexible languages for UI implementation (UI functionality and human factors research);
- (7) Multiple UI delivery mechanisms (web- or application-based UI implementation) (UI interface research);
- (8) Support for teleoperation languages of different structures and syntax (teleoperation language research);
- (9) Flexible teleprogramming interface (teleprogramming research);
- (10) Ability to move the workcell/control station division easily (system software research); and

(11) Workcell sensors (sensor fusion and sensing technique research).

Two design models, loosely categorized by hardware and software, are used to achieve these design goals. The remainder of this chapter outlines these models.

3.3 Electronic Hardware Design: The Test-Bed Model

Telerobotic systems have been developed for use in submarine [LB94a] [LMR94], outer space [WL94] [BBL⁺94], and hazardous environments [BHW⁺94]. This thesis documents the design of a teleoperated mobile robot system intended for **research** in the areas outlined in the last section and Chapter 2. Unlike the projects mentioned above, the development of a test bed requires a great deal of flexibility to accommodate future research directions.

In typical modern systems, some flexibility is offered through software modifications. This concept is pursued in this project. In addition, flexibility with regard to electronic hardware structure is required. This requirement is approached by designing the electronic systems with a high degree of **modularity** to accommodate future changes. Fortunately, modularity also accommodates the most restrictive design constraint: financial limitations. Without a modular approach, cost would increase with the effort to accommodate new research areas.

Therefore, in addition to the features outlined in the last section, the following test bed features are required: design:

- modularity;
- flexibility; and

- safety of operation.

Modularity promotes the reuse of individual software and hardware blocks, and reduces system set-up time. Wherever possible, each module should be made as a stand-alone device, which aids in debugging and expands its usefulness. When designing hardware modules, the technology used should be consistent. For instance, a single type of microcontroller should be used throughout the system, or connector types should be standardized. Again, this promotes recycling, not only of code, but also of schematic libraries. Additionally, when researchers use the test-bed, the amount of information they need to know to reconfigure the setup is minimized.

The design is to be **flexible** so that both hardware and software can be reconfigured to suit an area of research. Reconfiguration may involve leaving a specific module entirely out of the system.

Safe operation requires the system to be reliable. If a CPU crashes the workcell should not behave unpredictably. In order to achieve this, it is suggested that a distributed-processor model be used to give the system some redundancy.

Start-up of systems typically present transient operation which needs to be accounted for. Bus contention needs to be avoided by ensuring that devices are disabled on power-up.

Since the test-bed is intended for use in a wide range of research applications, several requirements need to be accommodated:

- Multiple users will need access to the system, perhaps simultaneously;
- Development time of software needs to be minimized;
- Operation needs to be straightforward, as the researchers may receive minimal training on the test-bed; and

- Documentation needs to be concise and comprehensive.

In short, the more familiar the technology used in the system, the more effective the system will be from the researchers' perspective.

3.4 Software Interface Design Model: The RIP Paradigm

Design of telerobotic system software can be a daunting task. Design approaches can quickly become cluttered as more detail is established. To make the system software design process achievable, an alternate view of the system is required. This view is represented in Fig. 3.1, differing slightly from Fig. 2.1.

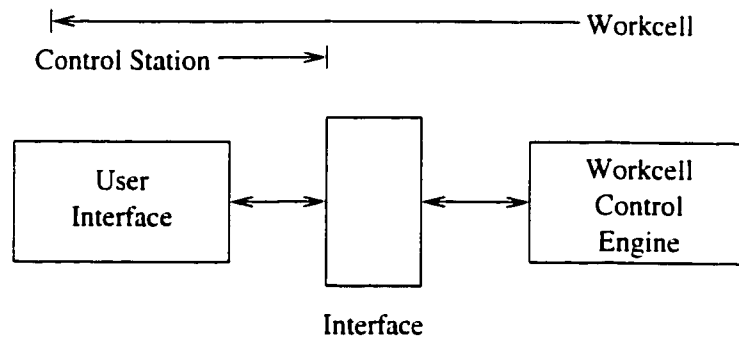


Figure 3.1: A top-level block diagram of a telerobotic application under the Robotic Internet Platform design paradigm. The User Interface is the component that the operator interacts with. The Workcell Control Engine, typically running on the workcell itself, is software responsible for control of the workcell movement. The software interface between the two needs to be well documented, flexible, and based upon the string data type.

This view is referred to as the “Robotic Internet Platform” (RIP) paradigm. Under the RIP paradigm, it is assumed that the communication channel (referring to Fig. 2.1) consists of an Internet connection and both the control station and the

workcell are Internet machines capable of utilizing this connection. Inherent to the design paradigm is the ability to develop telerobotic systems that have more than one user and/or more than one workcell.

Essentially, the remote workcell can be viewed as an Internet server that has the ability to move, whether in respect to some global coordinate frame, or through movement of manipulators, etc. The addition of motion to an Internet server (or conversely, Internet connectivity added to a robot) opens up an area that is just starting to be explored.

A primary goal of this model is to separate the user interface (UI) from the workcell control engine, a separation which manifests itself in the telerobotic system software. The separation of the control station and the workcell is no longer solidly defined: the bounds of where specific features of the system are implemented depends upon:

- the method chosen to implement the UI; and
- the required nature of the workcell control engine.

The workcell control engine is responsible for low-level I/O with sensors and actuators. Depending upon the application, it may also be responsible for control algorithms requiring reasonably fast execution speed. It is assumed here that the workcell control engine will be entirely implemented on the physical workcell.

The primary goal of the RIP design model is to develop a standardized workcell control engine that can be utilized by any telerobotic system UI. Conversely, this will allow any UI developer to access the features of the workcell through a standardized interface; something which is essential in a multi-user environment. Such a separation will minimize development time for researchers and developers.

3.4.1 Selection of the Operating System

The operating system (OS) upon which the user interface is implemented can be different than that selected for the workstation control engine.

For quick development time it is suggested that the selected operating systems on the workcell and the control station(s) be identical. This allows direct porting of code common to the control station and the workcell.¹ However, depending upon the application being addressed, common OSs may not be possible. For instance, in design of a system which is intended to be accessed by a wide range of users, it may not be possible to specify a consistent control station OS.

As time progresses and the difference in capabilities of operating systems becomes smaller (reflecting the apparent trend in OS development), this stage of the design process will become simpler. In essence, the choice of the operating system will depend upon the selected user interface and workstation control methods, outlined shortly.

3.4.2 Design Implications

Changing how a telerobotic system is viewed as is suggested under the RIP paradigm changes the design process. A great deal of the design effort needs to be spent on creation of the software interface between the UI and the workcell control engine.

The fact that the Internet is utilized as the communication medium between the control station and the workcell makes this software interface a necessity. To make the telerobotic system usable by the widest range of users it will be necessary to

¹ Code porting can also be addressed by the development language. The selection of the development language is addressed in Chapter 6.

accommodate various UIs written on various development platforms: thus, the need for a strictly defined (yet flexible) software interface arises. The fact that multiple users may be in communication with the robot simultaneously also requires a great deal of planning in order to maintain system integrity.

The interface should receive and issue nothing but ASCII characters. The primary reason for this is that strings can encapsulate other data types (by use of their string representations) and thereby offer a sense of global portability. Almost all programming languages deal with strings in an identical way.

In essence, the RIP paradigm emphasizes the need for modularity of the two major system software components - the UI and the workcell control engine. This decoupling is commonplace in software design, and it is suggested that the lessons learned there should be applied to telerobotic systems. When decoupled, telerobotic systems used for research and development may be adopted by institutions with few, if any, changes to the workcell control engine.

3.4.3 User Interface Development Considerations

UIs can be developed in many different programming languages. Bearing in mind the telerobotic application and its test-bed nature, the following factors are of interest in selection of the UI development platform:

- hardware independence/portability;
- ease and speed of development; and
- bandwidth/hardware requirements.

Hardware independence describes the ability to quickly port software to new workcell configurations as well as the ability to run the same user-end software on mul-

multiple workstation types. Ease and speed of development is especially important in the experimental phase when the developer needs to make rapid changes to the software and quickly simulate various environmental conditions. Bandwidth and hardware requirements may be an issue when power or circuit real estate is at a premium, such as in small mobile robots.

3.4.4 Workcell Control Engine Design Implications

The workcell control engine has several roles:

- (1) to provide access to the robot sensors and actuators (low-level I/O);
- (2) to provide feedback control of the actuators (if required by the application);
and
- (3) to provide the software interface to the UI.

The typical method of providing low-level I/O to the robot sensors and actuators is through routines written in C or Assembler. The interface to the workcell control engine needs to conform to one major restriction: the data that is initiated by, and destined for, the UI needs to be compatible with data types supported by the control engine development platform. Since one of the major motivations of the RIP paradigm is to allow development of a UI on almost any platform, this interface needs to cover as many bases as possible. This requirement points to the use of ASCII characters (or sequences of characters) as the data type of choice, outlined earlier.

Certain levels of autonomous operation, survival reflexes, etc., may be implemented within the workcell control engine, depending upon the application being addressed. The issue of how complex (how high-level) the workcell control engine should be is still a topic of discussion (see, for instance, [Lum94] or [LMR94]).

The manner in which this is addressed indirectly points to the syntax of the teleoperation language.

The implementation and type of control-loops that are embedded into the workcell control engine may place requirements on where it is implemented. For instance, a servo-loop that requires a fast sampling rate would be best implemented directly on the workcell in order to avoid any delay presented by the communication medium.

3.5 Summary

This chapter has outlined two models which should govern the design process of a teleoperation system. These models are a hardware design model (the test-bed design model) and a software design model (the RIP paradigm), although there is a certain amount of interplay between the roles of the models.

The test-bed model dictates that the design process should be modular and flexible to minimize reconfiguration for research in a specific area of telerobotics.

The RIP paradigm suggests that the workcell, viewed from a software perspective, be considered an Internet server that has the ability to move. The interface between the UI and the workcell control engine should be string-oriented to allow for development of either component in any language. However, the UI and the workcell control engine should be written with portability in mind, allowing for implementation on an arbitrary operating system.

Chapter 4

System Overview

4.1 Introduction

This chapter contains a high-level description of the system. The discussion of the design is presented in light of the models presented in Chapter 3.

The following components of the design receive attention here:

- (1) the control station hardware;
- (2) the control station operating system;
- (3) the workcell PC hardware (motherboard);
- (4) the workcell PC operating system;
- (5) the communication technology; and
- (6) the client-server relationship.

Chapter 5 details the workcell hardware. Chapter 6 describes the software (and the development platforms) for the components of the system residing on the

microcontroller (sensor/actuator) layer, the workcell (server) layer, and the control-station (client) layer.

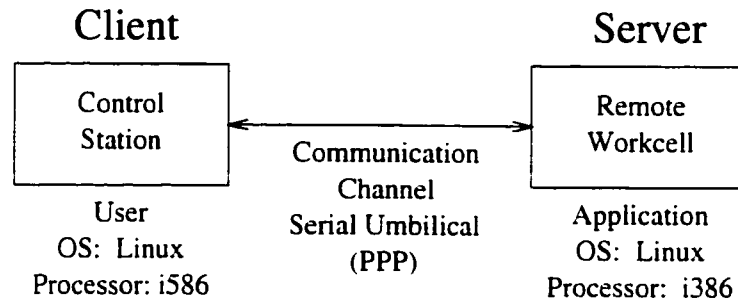


Figure 4.1: A block diagram of the test platform with some details filled in under the guidance of the two design models described in Chapter 3.

Fig. 4.1 is the typical view of a telerobotic system with some details of the design filled-in. Fig. 4.2 shows the system at a more detailed level than Fig. 4.1.

4.2 The Control Station

The Control Station is the component of the system upon which the User Interface runs. Commands are sent to, and feedback is received from, the workcell PC motherboard through the communication link.

In the system presented here, the entire workcell control engine is implemented directly on the workcell. Conversely, the UI is implemented on a dedicated PC. This Cyrix 586 motherboard has sufficient computational ability to present UIs to the different users of the system:

- The programmer.
- The user.
- The builder (designer).

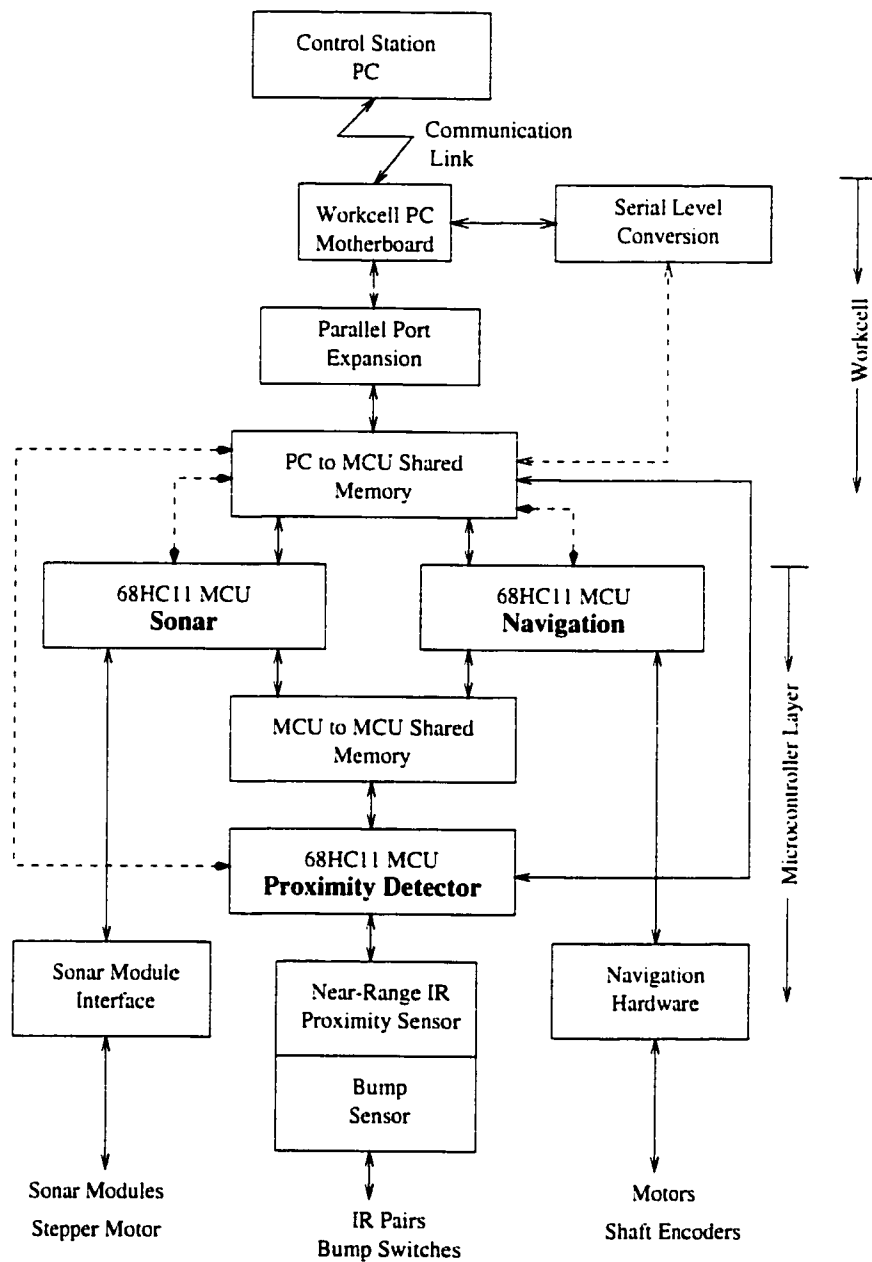


Figure 4.2: A block diagram of the system layout. Dashed lines indicate serial communication. The system is divided into three "layers": the control station layer, consisting of the Control Station PC, the workcell PC layer, consisting of the Workcell PC Motherboard, the Serial Level Conversion, Parallel Port Expansion and PC to MCU Shared Memory blocks, and the microcontroller layer.

- The maintainer.

A PC platform was selected due to the wide range of communication peripherals that are available for it: wireless Ethernet, wireless modems, etc.

In addition to the motherboard, the control station PC contains storage devices, external Ethernet connection to the Internet, a 17" monitor, and mouse.

4.2.1 Control Station Operating System

The operating system on the Control Station PC is Linux, a PC derivative of the UNIX operating system. The Linux (RedHat 6.0) platform was selected for a number of reasons:

- It is free:
- It is a truly pre-emptive multitasking operating system:
- The kernel is open-source and modifications can be made to the operating system:
- It is well documented:
- There is a large knowledge-base in the Internet community which freely communicates regarding Linux issues:
- It offers a comprehensive set of development tools, such as C-compilers, debuggers, and text editors:
- It offers the X-windows interface, ideal for GUI development and presentation:
and
- It has full TCP/IP support.

Multitasking is required as the system may have multiple users connected to the system simultaneously, and the event-driven nature of GUIs needs to be accommodated. Good documentation and a healthy knowledge base are advantages for a good test-bed system. The development tools, often written by programmers for programmers, are very flexible, varied, and thorough. Many programming languages have been ported to Linux. The X-windows system is not required, but in addition to providing a nice GUI environment it also offers a novel display technique which allows applications to display their UIs on other X-window compliant systems connected through a network. TCP/IP support is required under the RIP paradigm.

As mentioned in the previous chapter, any operating system could have been selected if it could meet the UI requirements. This is made possible by the separation of the control station from the workcell control engine under the RIP paradigm.

The wide range of languages available in Linux meets the design objectives regarding UI interface medium research, and UI functionality and aesthetic research, outlined in the last chapter.

4.3 The Workcell PC Motherboard

The Workcell PC Motherboard resides on the workcell. It is responsible for implementing the server-side of the software interface. The Workcell Control Engine, implemented on this device, has ultimate responsibility over workcell activity.

An Intel 386 motherboard is the major processing element on the workcell. The selected motherboard, chosen for its compact size and availability, conveniently offers integral video, disk controller, serial communication, and parallel (printer) port circuitry.

The computational ability of the i386 system is sufficient to allow autonomous or reflexive behaviour creation. This meets the control level research, autonomous control, control transfer, and user-initiated control research design objectives, as outlined in the last chapter.

4.3.1 Workcell PC Operating System

Again, the Linux operating system is chosen for the workcell PC. Consistency of this operating system with that of the control station PC is suggested under the RIP paradigm. Rationale for selection of Linux is the same as that described in the last section.

4.4 Communication Technology

The communication link connects the Control Station PC and the Workcell PC Motherboard. It is through this link that all directives from the control station, and feedback from the workcell are sent.

The communication channel is an umbilical serial line. The signals are RS-232C format; the communication protocol used is the Internet Protocol (IP) at 115200 baud. Linux offers pppd, the Point to Point Protocol (PPP) daemon, which controls the connection.

The serial umbilical can be replaced with a wireless modem pair (one at the control station, one at the workcell) with little reconfiguration of the PPP connection. The Linux operating system offers kernel support for wireless Ethernet adaptors, providing smooth transition into a more robust, faster communication interface.

The communication channel research design objective outlined in the last chap-

ter is therefore met.

4.5 The Client-Server Model

This section applies the client-server model to the design of a teleoperation system.

Many telerobotic systems take the perspective that the control station is the master (server) and the workcell is the slave (client). See, for instance, [LB94b], [LF94], [STS⁺95], [LB94a], and [BBZ98].

Under the RIP paradigm, a client-server model can be used to describe the system structure. Since it is likely that several researchers will be utilizing the system simultaneously (although in different capacities), it is necessary that the workcell is the server. This provides the most flexible platform in order to perform research in many areas.

4.6 Summary

In this chapter, many of the design objectives outlined in Chapter 3 have been met. This was conveyed through a top-level abstraction of the system design. At either end of a flexible communication medium, the Linux operating system is running on x86 processors. Linux was chosen since it is free, well supported by the Internet community, is open-source, and contains a suite of development platforms ideal for telerobotic system development. These choices are guided by the design models presented in the last chapter. When a client-server model is applied under the RIP paradigm, it is necessary that the workcell control engine is the server, and

the control-station is the client.

Chapter 5

Modular Workcell Hardware Design

5.1 Introduction

This chapter is a description of the electronic and mechanical hardware developed for the remote workcell. Additional information, including schematics, part-lists, and board layouts can be found in the Appendices.

The components referred to here are the result of a design approach whose generalized goals are to:

- (1) achieve a high degree of electronic modularity;
- (2) use simple, generalized hardware module interconnection;
- (3) keep hardware complexity minimal; and,
- (4) where possible, design the modules such that their usefulness outside of the test-bed design will be maximized.

The next section describes how each of the blocks fit into the system as a whole includes overviews of the modules. The last section describes the mechanical structure of the workcell in its current configuration.

5.2 Module Overview

Nine modules were designed for the workcell electronics. As mentioned previously, each of the modules receives detailed attention in the Appendix.

The nine modules (and their purposes) used in normal workcell operation are listed here:

- (1) Serial level conversion board – converts RS-232C signals to TTL levels, and vice versa.
- (2) Parallel-port expansion board – expands the capabilities of a standard PC parallel port to interface with the microcontroller boards and the shared memory interface.
- (3) PC-to-MCU Shared-memory/serial network control board – implements a shared-memory model for communication between the workcell PC motherboard (through the parallel-port expansion board) and the microcontrollers.
- (4) MC68HC11 Microcontroller (MCU) board – implements a single-board embedded system which is used to interface to sensors and actuators. Three are used, allowing for a distributed-processing architecture: “SONAR”, “Proximity Detector”, and “Navigation” MCUs.
- (5) MCU-to-MCU Shared-memory board – implements inter-microcontroller com-

munication through a shared-memory model. Again, this is used to implement a distributed-processing architecture.

- (6) SONAR module interface board – interfaces the “SONAR” MCU with Polaroid SONAR modules and a stepper motor.
- (7) Near-range proximity sensor board (IR pairs) – interfaces the “Proximity Detector” MCU with up to 32 infrared pairs.
- (8) Bump sensor board – interfaces the “Proximity Detector” MCU with up to 16 bump switches.
- (9) Navigation hardware board – interfaces the “Navigation” MCU with the workcell drive motors and their infrared shaft-encoder feedback.

Fig. 4.2 shows how the system hardware is normally configured. Sufficient detail is provided in the subsequent subsections and appendices so that a system developer should be able to see how the hardware can be interconnected for different purposes (for instance, a simplified setup for testing of MCU software).

The system is intended for a distributed-processing “kernel”. As mentioned in Chapter 3, selection of this model makes the system robust to, for instance, incorrect operation of a computational element. As an example, software could be written such that the workcell would still operate (although with reduced functionality) if the on-board PC was rendered inoperative. This is necessary for safe operation since the workcell should remain under control even if a failure occurs—a requirement of the test-bed design model.

To implement the distributed-processing structure, a shared memory-model was chosen due to its flexible nature and fast data transfer. Simplicity, however, is also

an important trait.

Motivation for selection of the technologies and techniques used for the individual modules is addressed in the corresponding subsections, below.

5.2.1 Serial Level Conversion PCB

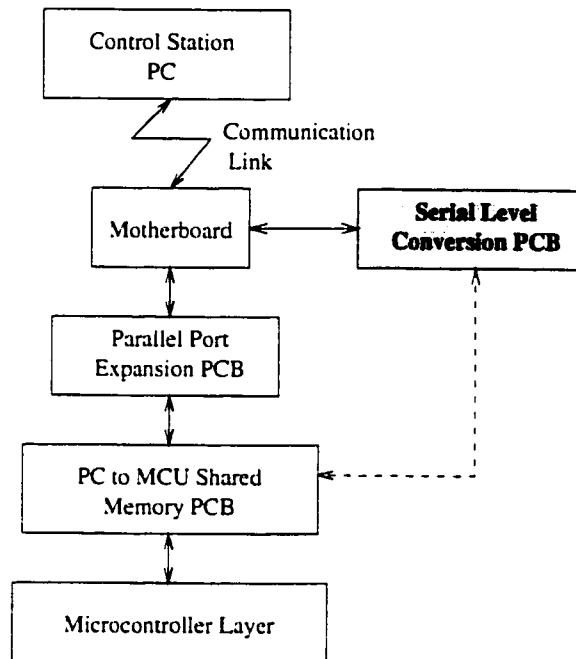


Figure 5.1: Location of the Serial Conversion PCB in the test-bed.

The simplest of all the modules, the Serial Level Conversion PCB is connected between the workcell PC serial port and the MCU boards (via the PC-to-MCU Shared-Memory/Serial Network Control PCB). PC RS-232C serial voltage levels (typically ± 10 V) are converted to TTL-levels (0, 5V) which are used by the microcontrollers. Conversion is also performed in the opposite direction by this single-IC board. Fig. 5.1 shows where this hardware module fits into the overall system. A block diagram of the board electronics are shown in Fig. 5.2. For more information

on this module, refer to Appendix B.

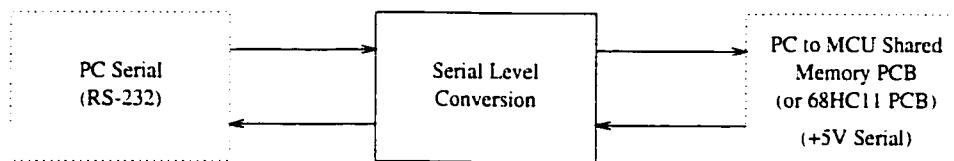


Figure 5.2: A block diagram of the Serial Conversion board electronics, used to convert Workcell PC Motherboard RS-232C level serial levels to TTL-level serial for use by the microcontrollers.

5.2.2 Parallel-Port Expansion PCB

This board is used to connect the Workcell PC Motherboard to the memory shared with each of the MCUs, and to also provide PC-initiated control-signals.

The Parallel-Port Expansion PCB connects to the parallel port of the workcell PC motherboard. Using a data-latching scheme, the card provides 64-bits of digital output, or 56-bits of digital output and an 8-bit bidirectional data bus for connecting to 8-bit peripherals such as shared-memory. A block diagram of board electronics is shown in Fig. 5.3. For more information on this module, refer to Appendix C.

5.2.3 PC-to-MCU Shared Memory/Serial Network Control PCB

This module serves as the communication medium between the Workcell PC Motherboard and the microcontrollers.

This PCB performs three specific functions:

- (1) to implement 4K of shared memory between the workcell PC and each of the three MCUs;

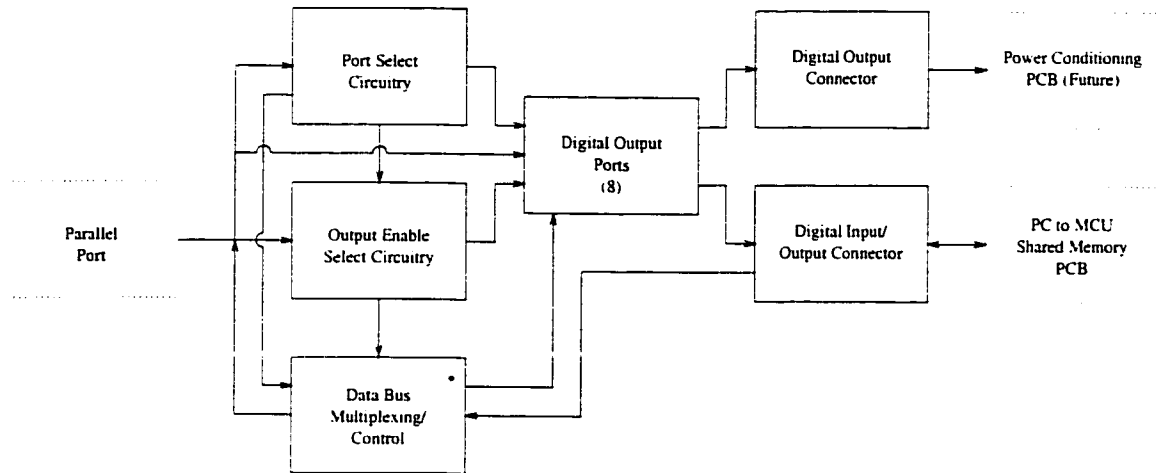


Figure 5.3: A block diagram of the Parallel-Port Expansion board electronics. Dashed blocks are external to the PCB. Eight digital output ports, controlled through output-enable and port select circuitry are used to control the system MCUs and provide address and data bus interfaces to shared memory.

- (2) to route TTL-level serial signals between the PC (via the Serial-Level Conversion PCB) and the MCUs: and
- (3) to provide low-power Schottky (LS) to high-density CMOS (HC) TTL level conversion.

A block diagram of board electronics is shown in Fig. 5.4. For more information about this module, refer to Appendix D.

5.2.4 MC6HC11 Microcontroller (MCU) PCB

This board consists of a Motorola 68HC11 8-bit microcontroller, 32K external battery-backed static RAM, chip select circuitry, and various support hardware. The microcontroller boards are used to interface directly to the system actuators and sensors. Three of these devices are used: SONAR, Proximity Detector, and Navigation MCUs for specific functional control of the workcell. This structure allows for parallel

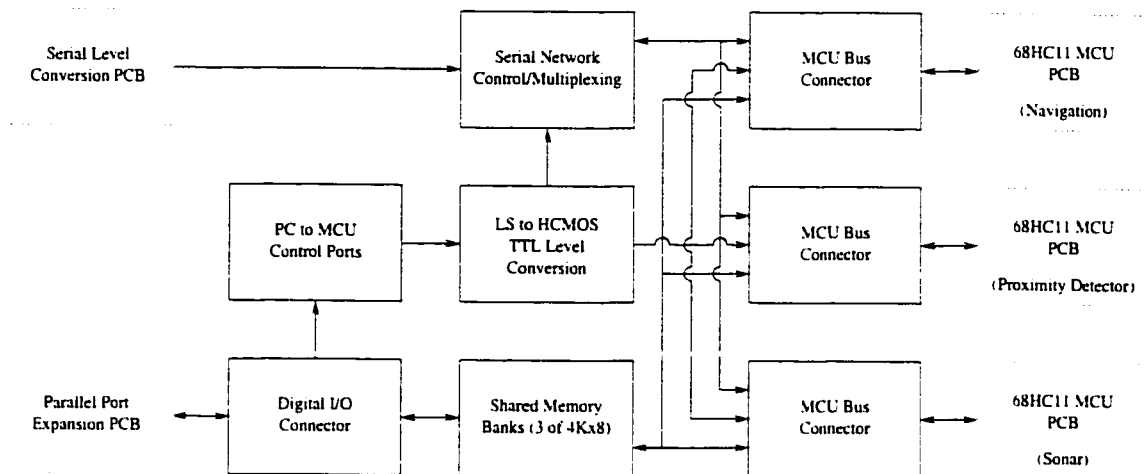


Figure 5.4: A block diagram of the PC-to-MCU Shared Memory/Serial Level Conversion board. The LS-to-HCMOS TTL Conversion block translates motherboard-initiated control signals to levels usable by the MCUs. A Serial Network Control block allows connection of a single serial channel to three microcontrollers. Shared-memory is used as the run-time communication medium between the Workcell PC (via the Parallel-port Expansion board) and the MCUs.

processing, redundant computation, and fail-safe operation of the workcell. Fig. 5.5 shows where these hardware modules fit into the overall system.

Fig 5.6 shows a block diagram of the circuitry located on this PCB. Appendix A gives a complete description of the connector pin-outs and the board operation.

The Motorola 68HC11 was chosen for several reasons:

- (1) it is well-supported by the robotics community;
- (2) it is inexpensive;
- (3) it is very dependable (stable);
- (4) there is an abundance of free development products available for it; and
- (5) availability.

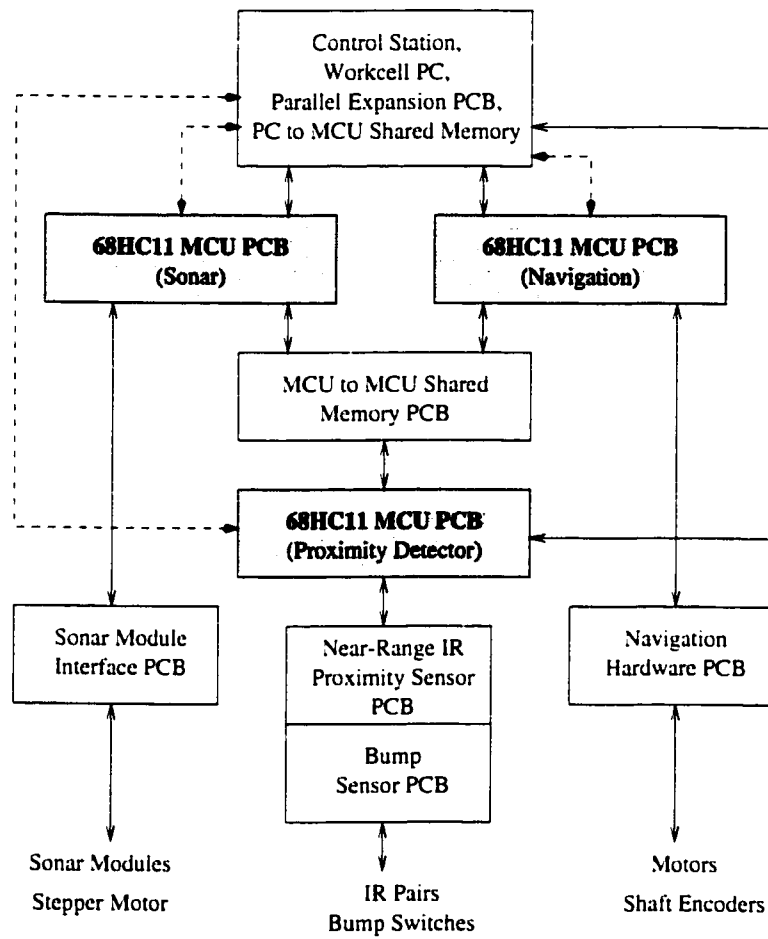


Figure 5.5: Location of the HC11 microcontroller PCBs in the test-bed.

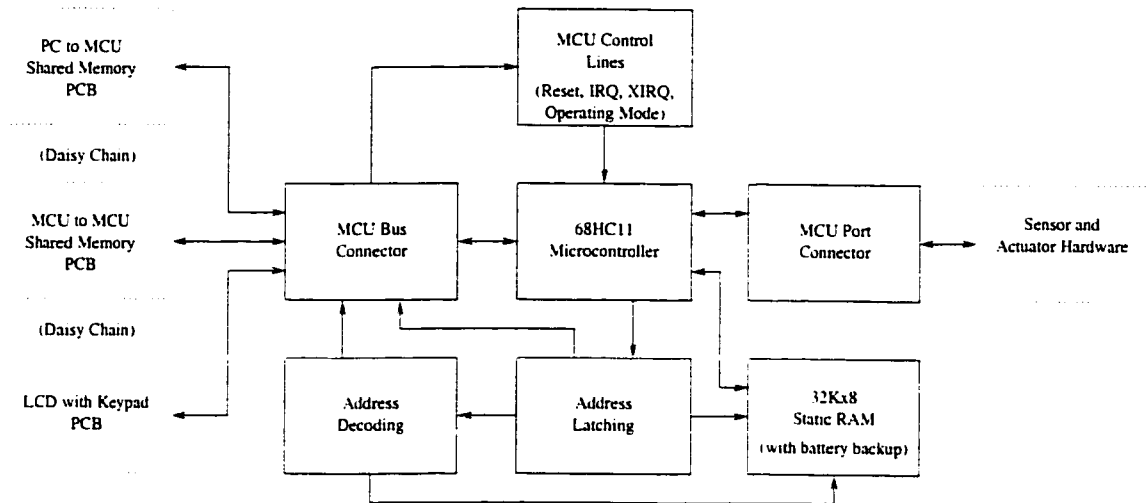


Figure 5.6: A block diagram of the MCU board electronics. The blocks create a fully-functional single-board computer: address decoding and latching, the microcontroller itself, and battery-backed RAM.

Using three of these devices in the system is overkill for the example application described later in this document. However, the logical separation of the workcell sensory/actuator subsystems into three categories (navigation, proximity detector, and SONAR) has proven to be a definite advantage in that the system developer is forced to define the nature of the interface to each logical block. Software may be developed in a tight, modular manner which promotes “recycling” of code when a different teleoperation application is addressed.

Another advantage of using three HC11 microcontroller boards is that it provides resources for future implementation of more complex sensor-fusion algorithms and autonomous behaviours.

5.2.5 MCU-to-MCU Shared Memory PCB

This board serves as an inter-MCU communication medium, useful for distributed-processing structure.

In order to achieve MCU-to-MCU communication without involvement of the PC, this board provides three banks of 4K dual-port static RAM. This supports the distributed-processing architecture of the workcell hardware. A block diagram of board electronics is shown in Fig. 5.7. For more information about this module, refer to Appendix E.

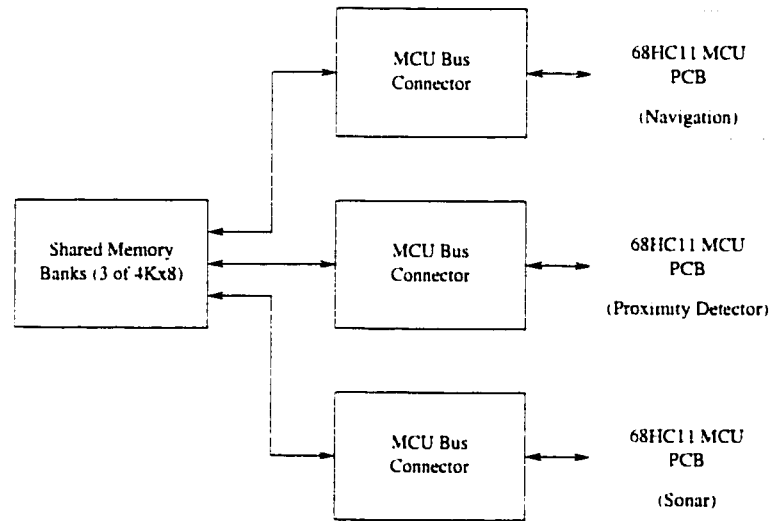


Figure 5.7: A block diagram of the MCU-to-MCU Shared Memory board electronics. Inter-MCU communication is achieved through 3 banks of 4k dual-port static RAM.

5.2.6 SONAR Module Interface PCB

This board is used to interface to two Polaroid SONAR modules and a stepper motor used to rotate the SONAR transducers.

A block diagram of the board electronics is shown in Fig. 5.8. This board uses the SPI facility and two HC595 serial-to-parallel converters to derive digital outputs. These digital outputs are used to interface with Polaroid SONAR modules and the stepper motor driver. As with the navigation control hardware, the backbone of the motor driving circuitry is an SGS Thomson L298N dual H-bridge driver. In

addition to the SPI-derived digital outputs, four signals from the connected MCU's Port A are interfaced to the SONAR modules. Port A, associated with the HC11's timer facilities, was used in order to simplify the software responsible for driving the modules.

For more information about this module, refer to Appendix I.

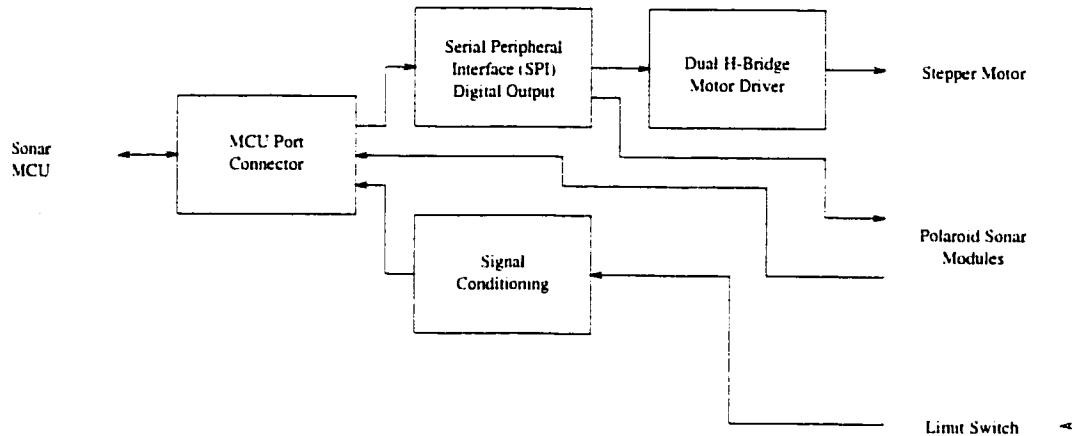


Figure 5.8: A block diagram of the SONAR Module Interface board electronics. A stepper motor is driven by a dual H-bridge motor driver IC, which is in turn controlled by the SONAR MCU through its Serial Peripheral Interface (SPI). Connections to two Polaroid SONAR modules are made through the SPI and directly to the MCU. A limit-switch's signal, relating information about the position of the SONAR array, is passed to the MCU after signal conditioning.

5.2.7 Near-Range IR Proximity Sensor PCB

This board is used to interface the Proximity MCU to 32 near-range infrared pairs.

A block diagram of board electronics is shown in Fig. 5.9. Facility is provided to apply power to the infrared LEDs, and an analog multiplexing scheme is used to select which "bank" of infrared phototransistors are connected to the HC11 A/D

converter. Using this scheme, each IR LED can be individually turned on or off, necessary for reading ambient-light levels and sensor data fusion.

For more information on this module, refer to Appendix G.

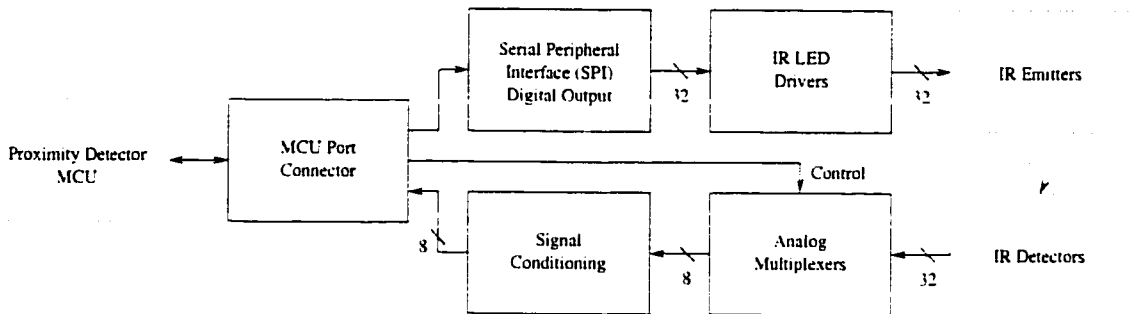


Figure 5.9: A block diagram of the Near-Range Proximity Sensor board electronics. The SPI is used to control the power applied to the infrared emitters through a set of LED drivers. Feedback from the IR detectors is passed through a set of analog multiplexers, controlled by the Proximity Detector MCU. After signal conditioning, the analog levels of the IR detectors are passed back to the MCU analog ports.

5.2.8 Bump Sensor PCB

The purpose of this board is to interface the Proximity Detector MCU to the many bump-sensors on the robotic workcell.

A block diagram of the board electronics is shown in Fig. 5.10.

The backbone of this board is the use of 'HC165 parallel-to-serial converters. Interfaced to the Motorola's Serial Peripheral Interface (SPI), two of these devices allow for 16 digital inputs (8 each). Each bump-switch connector has a single 4.7k pull-up resistor, allowing connection of more than one device in a wired-or configuration. Although this interconnection scheme is possible, only single bump-switches are connected to a given input in this project.

For more information about this module, refer to Appendix H.

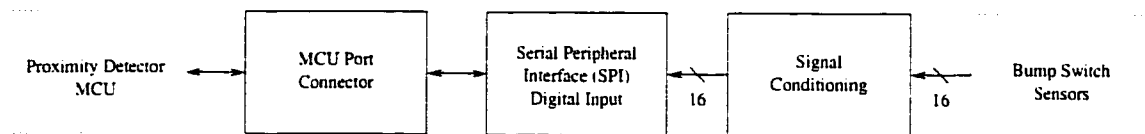


Figure 5.10: A block diagram of the Bump Sensor board electronics. Serial Peripheral Interface (SPI) inputs are used to read the conditioned state of up to 16 bump switches.

5.2.9 Navigation Hardware PCB

This board is used to interface the Navigation MCU to the DC servo motors that drive the robot. This interface consists of driver circuitry (an L298N dual H-bridge motor driver), and connections to, and signal conditioning for, optical encoders.

A block diagram of board electronics is shown in Fig. 5.11. The main component of this board is the SGS Thomson L298 dual H-bridge motor driver. This device can control two motors requiring up to 2 Amps each at 46 Volts.

For more information about this module, refer to Appendix F.

5.3 Mechanical Structure Overview

The workcell consists of an inexpensive, modifiable chassis which incorporates flexible positioning of the electronics modules and sensors. Changes to the robot's chassis structure to accommodate operability in a different environment than that originally intended would require redesign. However, the modularity of the electronic systems, as described previously, would smooth this transition. This section describes the chassis design and sensor placement for the mobile robot system component configured for indoor applications.

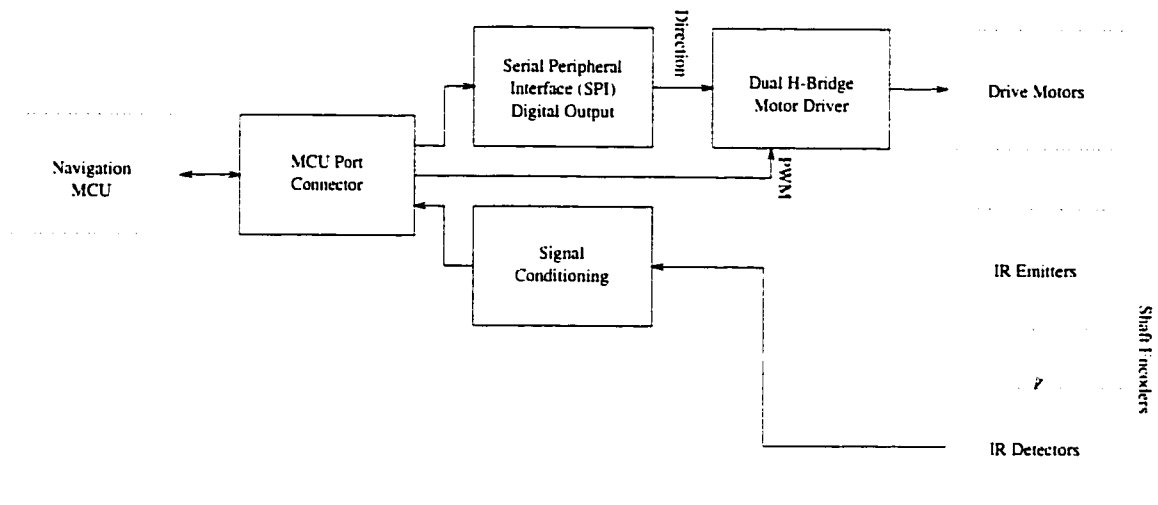


Figure 5.11: A block diagram of the Navigation Hardware PCB electronics. An H-bridge motor driver, controlled via the SPI and direct connections to the MCU drives the two workcell motors. Feedback from optical shaft encoders on the motor is conditioned and sent back to the Navigation MCU.

5.3.1 Structure

The structure of the chassis was inspired by Ron Kube's¹ design of a mobile robot used in telepresence research at the University of Alberta [BBZ98] in the Department of Computing Science. Fig. 5.12 shows a side view of the mobile robot. Each "level" is a 17" diameter circle of 5/8" medium-density fibreboard (MDF). MDF is inexpensive and easily allows sensor mounting at arbitrary positions. Each level is held in place by nut and washers on the four 3/8" threaded rods. This method offers flexibility in level spacing and provides a surprising amount of rigidity. A photograph of the workcell is shown in Fig. 5.13.

Two Polaroid SONAR transducers are mounted at the top of the mobile robot upon a stepping motor with a step resolution of 1.8°. Rotation of the SONAR array

¹ Ron Kube was at the time a Ph.D. student in the Department of Computing Science at the University of Alberta.

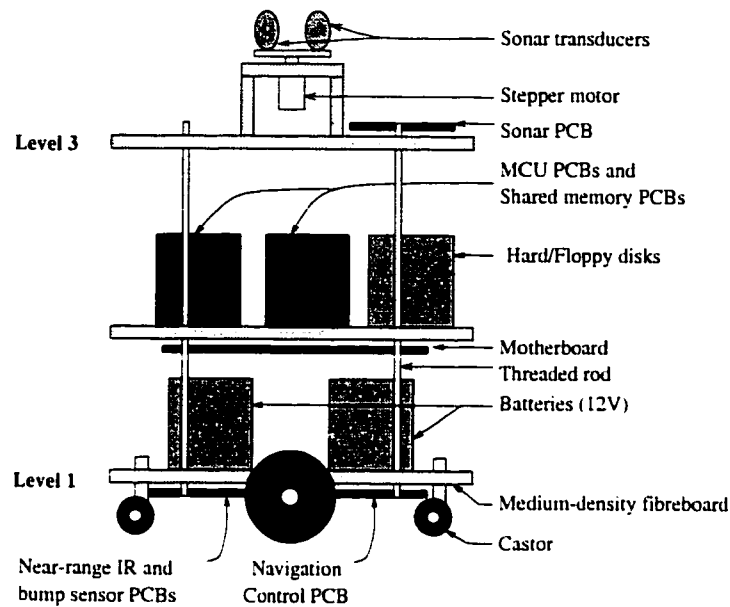


Figure 5.12: Side view of the mobile robot workcell.

in this manner makes it possible to obtain satisfactory results using only two of the modules.

Two 24 Volt gear-head motors provide locomotion via differentially steered drive wheels. Encoders with a resolution of 50 pulses per wheel revolution are attached directly to the output shaft and provide the feedback necessary for closed-loop control of the mobile robot's speed. These are also used for dead-reckoning calculation of position.

5.3.2 Sensor Placement

Fig. 5.14 shows the locations of the many proximity sensors used in the system (with the exception of the SONAR modules, which are shown in Fig. 5.12). Near-range proximity sensors include 26 infrared (IR) pairs and 16 bump switches mounted at multiple levels. The redundancy provided by placing the sensors at equivalent po-

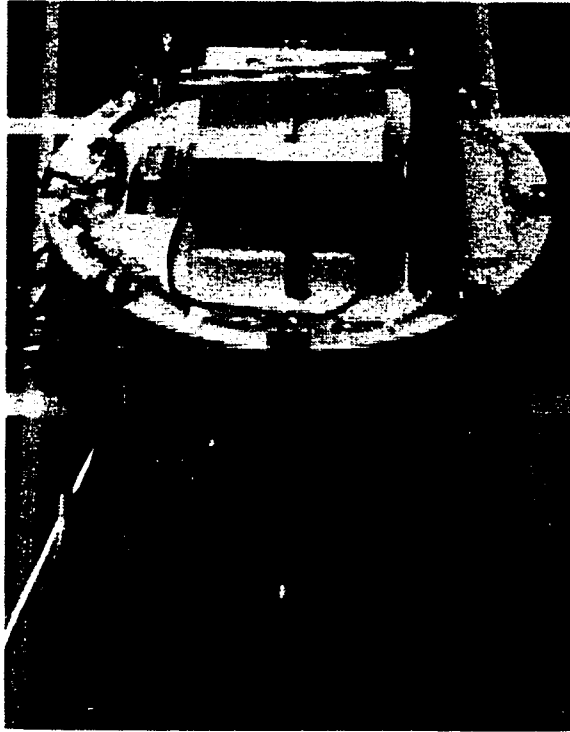


Figure 5.13: Photograph of the mobile robot workcell.

sitions on multiple levels is intended to assist in accurately sensing situations that will inevitably be present in unknown environments. Around the drive wheels and castors, sensors are arranged to detect obstacles in the direction of travel and irregularities in the traveling surface.

Fig. 5.15 shows the sensing ranges of the three sensor types. The range of bump sensors is limited, but also of the utmost importance: if a bump is detected, action needs to be taken immediately due to the proximity of the obstacle and its potential effects on the workcell. Infrared sensors have a range of up to 1 metre. Their data, sometimes unreliable due to ambient light levels, does not need to be acted upon immediately. SONAR, by far the longest range sensor, provides reliable data up to approximately 10 metres. The continuous nature of a SONAR “sweep”, outlined

in the next few chapters, makes SONAR data ideal for constructing a map of the environment. Although reconfigurable, the presented system configuration allows for research in sensor fusion technique, thus meeting another of the design objectives.

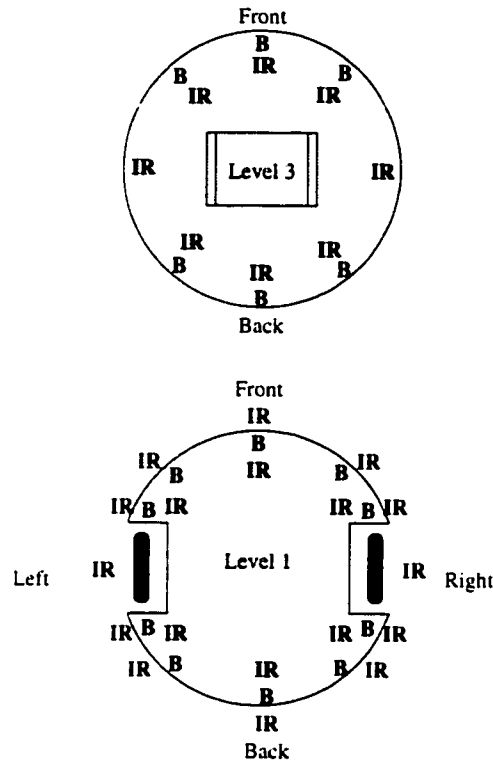


Figure 5.14: A top view of the positions of the workcell sensors. **B** – Bump switches. **IR** – Infrared pairs.

5.4 Summary

This chapter has presented the modular structure of the custom-designed work-cell electronics. Each of the nine modules can operate to some capacity as a stand-alone device which makes debugging and testing a simpler task: an important trait of a test bed. Placed in a system together, the modules form a distributed-processing

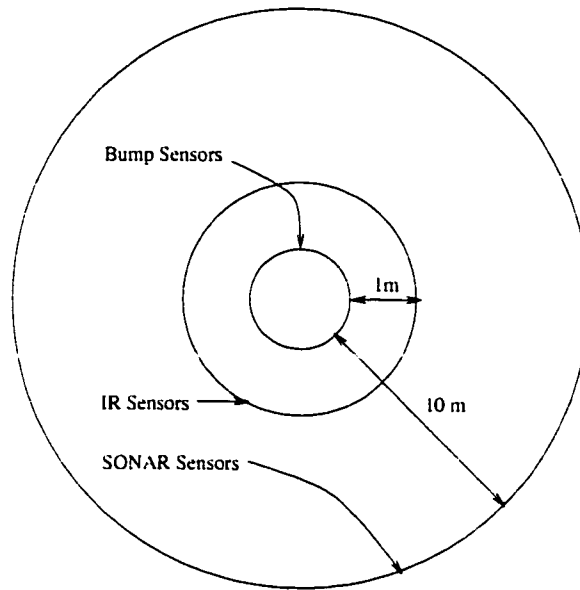


Figure 5.15: A graphical depiction of the sensing distances of the three sensor types present on the chassis: bump, infrared, and SONAR. This configuration allows for advance warning of approaching obstacles and allows the software to attach priorities to data from a particular sensor type.

platform which allows development of a robust, reliable system. The mechanical chassis and sensor placement is reconfigurable, allowing for research in different areas, and operation in varying environments.

Chapter 6

Modular Software Design

6.1 Introduction

This chapter is a description of the software developed for both the mobile-robot platform and the control station. The mobile-robot platform has four computational elements: the Workcell 80386 motherboard (PC), the proximity detector microcontroller, the navigation microcontroller, and the SONAR microcontroller. The control station has one computational element: a Pentium-compatible processor.

Details about specific procedures and functions are located in Appendix J. Please see this documentation for more information.

To aid in discussion, Fig. 6.1 depicts the languages used for implementing the various layers of the robotic system.

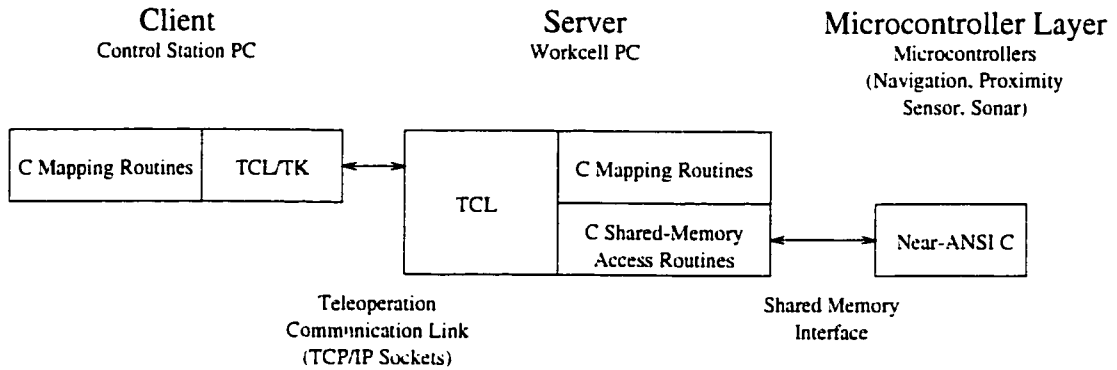


Figure 6.1: A diagram indicating the development languages used for the various components of the robotic system. The two primary languages are C and TCL/TK.

6.2 Microcontroller Layer Software

This section describes the software (and the development platform upon which it is developed) that runs on the microcontroller units. These MCUs are connected directly to the sensors and actuators of the workcell, as well as to the PC on the workcell.

Discussion begins with a description of Interactive-C, a real-time near ANSI-compliant C interpreter that runs on the MCUs. A description of how the interface to the MCU layer is performed is followed by description of the software which runs on the three MCUs.

6.2.1 Development Platform: Interactive-C

Interactive-C (IC) is a near-ANSI C compliant language that offers certain benefits to the microcontroller programmer, particularly in the field of robotics. It is termed “interactive” because it includes a rudimentary operating system that allows immediate processing of C-statements when typed in at a command prompt.

IC will run on any Motorola HC11 microcontroller that has 32K of external

RAM in the upper-half of its address space, a requirement which influenced the selection of the hardware memory devices. The user makes connection to the serial port on the microcontroller, and first downloads a pseudo-code (P-Code) interpreter into the external RAM. For steps used to send the P-code interpreter (and the application code) to the microcontrollers, refer to Appendix K.

Once up and running, IC has several advantages that are apparent:

- Floating-point routines are built-in (something that even several commercial C compilers intended for the HC11 are missing).
- The rudimentary operating system is a truly (time-sliced) multitasking environment.
- There are several free libraries of C routines that are specific to the HC11, and particularly to mobile robotics. For instance, pulse-width modulation routines are included with the system.
- IC's source-code is available, making it possible to modify its operation to better suit the application.

There are also commands for loading and unloading C-programs into memory. If there is a function called `main()` resident in the microcontroller memory, then the operating system takes a back seat to execution of the code contained therein when a reset is experienced.

Overall, Interactive-C running on the microcontrollers has performed quite well. However, there are many other platforms that could have been used to achieve the same functionality. Most notably, traditional assembly with the MCX11 multitasking kernel [BA89] would have worked well.

6.2.2 Software Operation

An overview of microcontroller software operation is presented here. In order to understand how the microcontrollers communicate with each other and with the workcell PC motherboard, the shared memory allocation is first described. Following this description, overview is provided for the software running on the Navigation, Proximity Detector, and SONAR microcontrollers.

6.2.2.1 The Shared Memory Allocation

There are four memory banks which are accessible by any one of the microcontrollers. There is the 32K RAM bank which it calls its own, two 4K RAM banks which are shared with the other two microcontrollers, and another 4K that is shared with the mobile robot PC. Although it would be possible to use the serial connections of all the microcontrollers (and the PC) to communicate data, it was chosen that these would be reserved for programming.¹

For the sake of simplicity, it was chosen that each 4K block of shared memory in the workcell would have the exact same memory map. Although a location may remain unused in a specific memory, 4K has proven to be far more than required. Several memory locations have been allocated for future expansion. Table 6.1 is shows the shared-memory allocation.

¹ This decision is based on the fact that shared-memory is far more flexible and the fact that debugging Interactive-C programs that use a serial interface is difficult.

Allocation of Shared Memory		
Mnemonic	Address (decimal)	Purpose
Computer Operating Properly Registers (Unused)		
NAV_COP	1	Computer operating properly (Navigation MCU).
PROX_COP	2	Computer operating properly (Proximity MCU).
SON_COP	3	Computer operating properly (SONAR MCU).
PC_COP	4	Computer operating properly (Workcell PC).
Command Registers (Unused)		
NAV_CMD	5	Command byte with operands (to Navigation MCU).
PROX_CMD	25	Command byte with operands (to Proximity MCU).
SON_CMD	45	Command byte with operands (to SONAR MCU).
PC_CMD	65	Command byte with operands (to Workcell PC).
Memory Access Semaphore Registers (Unused)		
NAVPROX_SEM	85	Navigation to Proximity MCU semaphore register.
PROXSON_SEM	87	Proximity to SONAR MCU semaphore register.
SONNAV_SEM	89	SONAR to Navigation MCU semaphore register.
PCNAV_SEM	91	Workcell PC to Navigation MCU semaphore register.
PCPROX_SEM	93	Workcell PC to Proximity MCU semaphore register.
PCSON_SEM	95	Workcell PC to SONAR MCU semaphore register.
Messaging Registers (overlapping fields)		
NAV_MSG_FLAG	97	Navigation message flag.
NAV_MSG	98	Navigation message.
PROX_MSG_FLAG	97	Proximity message flag.
PROX_MSG	98	Proximity message.
SON_MSG_FLAG	97	SONAR message flag.
SON_MSG	98	SONAR message.
MSG_FLAG	97	Generic message flag.
MSG_STRING	98	Generic message offset.
Navigation MCU Interface Registers		
(continued)		

Allocation of Shared Memory (continued)		
Mnemonic	Address (decimal)	Purpose
NAV_DIRECTIVE	366	Navigation directive.
SPEED_SETPOINT	367	Speed setpoint in metres per second.
DISTANCE_SETPOINT	371	Distance setpoint in metres.
ANGLE_SETPOINT	375	Angular (rotational) setpoint in radians.
COORD_REFRESH	379	Requested coordinate refresh rate in seconds.
X.LOCATION	397	Workcell current x-coordinate in metres.
Y.LOCATION	401	Workcell current y-coordinate in metres.
ORIENTATION	405	Workcell current orientation in radians.
Sonar MCU Interface		
SON_COMMAND	1537	SONAR sweep command register.
SON_RANGE1	1538	Range to obstacles from SONAR module 1.
SON_RANGE2	1938	Range to obstacle from SONAR module 2.
Proximity MCU Interface		
BUMP_DATA	3542	Bit-aligned bump sensor data.
IR_DATA	3544	IR sensor levels (A/D levels).
BUMP_FORWARD	3612	Forward bump sensors activated (flag).
IR_FORWARD	3613	IR sensors activated forward (flag).
PROX_REFRESH_RATE	3614	Proximity sensor polling rate (seconds).

Table 6.1: Allocation of the shared memory.

6.2.2.2 Shared MCU Software

The MCUs make use of shared library routines written in C. This is not only good practice, but also required under the test bed design model. There are three specific shared modules:

- (1) A messaging system module;
- (2) A shared memory access module; and
- (3) A Serial Peripheral Interface (SPI) module.

The first allows string information to be sent to the Control Station (via the Workcell PC) from the microcontrollers. This facility is useful for transmission of debugging information to the system developer.

Although shared memory can be accessed directly by the MCUs without any intervention, all access is performed through a shared memory access module. This was done so that shared memory access semaphores could be implemented in software in the future.

All three of the MCUs make use of the HC11's SPI facility. This module contains the code to initialize and perform input and output using the SPI.

6.2.2.3 The Navigation MCU Software

The Navigation MCU, responsible for all aspects of the workcell's movement has three processes running concurrently:

Slaved PI control (proportional-integral control) used to control the speed of the two drive motors. A graphical depiction of this algorithm is shown in Fig. 6.2 and derived from [JF93]. Along with this control-loop, the distance the workcell has traveled is updated using the information gathered from the encoders.

Position update uses the information updated in the PI control-loop to update the global position of the workcell in shared memory. The calculations are based on the drive motor encoder information.

Navigation Interpreter is a loop that reads navigation directives placed in shared memory by the Workcell PC. The interpreter acknowledges these commands and takes control over the PI loop to carry out the directive.

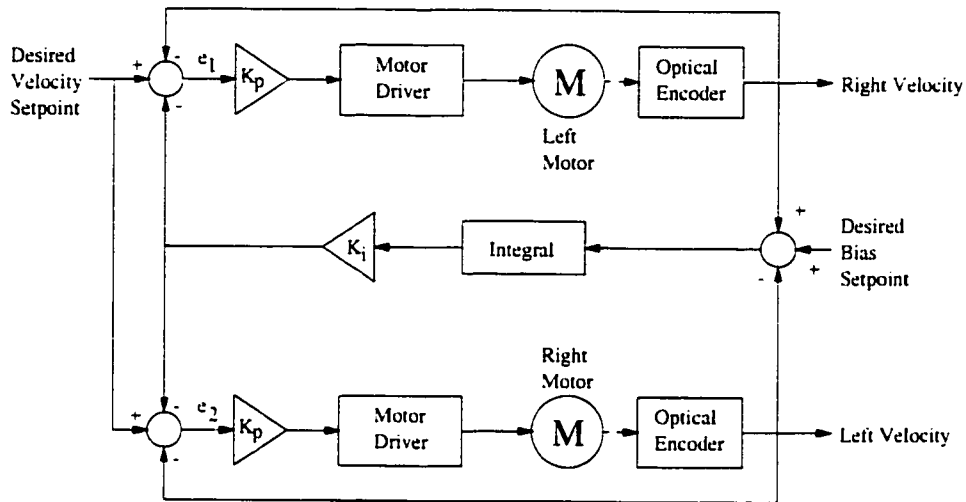


Figure 6.2: This slaved-PI velocity control loop is used to “link” the independent drive-wheels of the workcell together to, for instance, make them rotate at the same speed to drive in a straight line. The desired translational velocity and rotational (bias) velocity setpoints are applied to the control-loop. The common feedback for both of the motors, implemented using an integral and an integral gain, K_i achieve the link in software. The proportional gain, K_p in the feed-forward portions of each control loop are adjusted for suitable transient response of the corresponding motors. This control loop is executed 4 times per second.

6.2.2.4 The Proximity Detector MCU Software

The Proximity Detector MCU interfaces to the workcell’s bump and near-range IR sensors. Both types of sensors are connected to the SPI. In order to avoid a conflict between concurrent processes accessing the same piece of hardware, a single process reads the state of both sensor blocks:

Sensor Polling utilizes the SPI to first read the state of all bump switches and

update the shared memory. Next, individual “banks” of IR sensors are activated, and using the analog multiplexing scheme, their levels are read and related to the workcell PC motherboard.

The location of detected obstacles with respect to the global coordinate frame are calculated in the server software running on the workcell PC.

6.2.2.5 The SONAR MCU Software

This microcontroller is responsible for moving the SONAR sensor array and gathering data from the Polaroid SONAR modules. Only one process runs on the microcontroller:

Sweep Command Interpreter polls the state of the SON_COMMAND shared memory location to determine when a SONAR sweep is to occur. When a command is received, the sweep is performed, the shared memory updated, and the command is acknowledged.

A “sweep” consists of stepping the two SONAR transducers 180 degrees in steps of 1.8 degrees. At each step, both modules are “pinged” and the time to echo is captured (using the HC11 timer facility). Using the speed of sound, the distance to the obstacles is calculated.

6.3 Server (Workcell Control Engine) Software

This section is a summary of the software that runs on the system server: the workcell PC.

The workcell control engine has several roles:

- (1) to provide access to the robot sensors and actuators (low-level I/O);
- (2) to provide feedback control of the actuators (if required by the application);
and
- (3) to provide the software interface to the UI.

The method of providing low-level I/O is through routines written in C. The client-side interface to the workcell control engine needs to use the string data type as dictated by the RIP paradigm.

First, a discussion of the Tool Command Language (TCL) development platform is presented. TCL proves to be well-suited for implementation of the server side of the client-server relationship between the workcell and the control station. A description of the server software is then followed by techniques that can be used to program the server, perhaps remotely.

6.3.1 Development Platform: Tool Command Language

The server development platform is string-based TCL with C-based mapping routines and parallel-port I/O routines. TCL is simple to set up as a TCP/IP server, and has the ability to pass commands received through a communication channel to its interpreter.

Implementation of the workcell control engine utilizing TCL and low-level C routines offers a few unique advantages. First, TCL's primary data type is the string which allows natural adherence to the primary requirement placed on data-types by the RIP paradigm. Second, interfacing with TCL's C Application Program Interface (API) can be more-or-less automated with the use of the "Simplified Wrapper and

Interface Generator” (SWIG) [Bea97].²

The interpretive nature of TCL, and the ability to call the interpreter to execute commands passed across a socket connection in a sense makes TCL a teleoperation language in its own right.

Other languages, such as Java, are also suitable for system development. See, for instance [WSFM98]. However, for fast development time, TCL/TK is chosen.

6.3.2 Software Operation

The server (workcell control engine) software and how it operates is presented here. Some of the information presented here will be referred to in the section describing the client software. Specifically, the modules to do with mapping are almost identical on both the workcell and the control station.

6.3.2.1 Mapping Module Software

A mapping module consists of two parts: a C portion, interfaced to TCL’s C API, and a TCL portion.

A map of the workcell’s environment is represented using a Moravec occupancy grid [Mor86]. technique. This representation, implemented entirely in C, has re-configurable resolution and size. For a graphical depiction of the map, see Fig. 6.8. Each map “cell” has an 8-bit value associated with it that is used to store bit-aligned obstacle information. The levels are associated with User, Sonar, Bump, Infrared, and Algorithmic data. Sonar, Bump, and Infrared layers are used to store obstacles by the corresponding sensor type. The User layer is included to allow the user (at

² SWIG also has the ability to automate this process with script languages other than TCL/TK, such as PERL.

the control station) to interactively add their own obstacles. An Algorithmic layer is included to allow for a pattern-recognition algorithm to place virtual obstacles in the map.

The TCL portion of the mapping software provides a higher-level interface to the C map structure. Through the routines provided, maps can be transferred to- and from- the control-station, cleared, and synchronized.

6.3.2.2 Parallel Port Module Software

Interface with the parallel port expansion module, described in the last chapter, is achieved through use of low-level ANSI-C code. Also included in this module is higher-level TCL code which interfaces to the microcontroller layer through the shared memory.

The C routines are used to control the parallel port expansion board. This control allows reading from- and writing to- shared memory, and changing the state of the MCU control lines. As with the C mapping routines, the code is made callable from TCL through use of SWIG.

The higher level TCL routines are used to:

- Poll the state of the bump and infrared sensors:
- Control SONAR sensor sweeps and read the data returned by a sweep:
- Issue navigation directives to the Navigation MCU: and
- Read the position and orientation of the workcell.

Reading of the state of bump and infrared sensors is performed in a paced-loop, the period of which is established by the control station. If an obstacle is detected,

the corresponding layer in the C map is updated (and, if the obstacle is new, the control-station map is updated). The server-side “volunteers” this information once the sensor polling routines have been instigated (by opening the appropriate socket, described below). When a specific sensor has been activated a vector from the center of the workcell chassis to the obstacle is determined, based upon the specific sensor detecting the obstacle, and the detection range of the sensor. For instance, a bump switch on the rear of the chassis has a vector of magnitude equal to the workcell chassis radius at an angle of 180 degrees. This vector is used to calculate the location of the obstacle with respect to the global coordinate frame. When the map data is updated, these coordinates are used. Therefore, when the map resolution is quite high, it is possible to retain accurate representation of the environment.

SONAR sweeps, initiated by the user at the control station, are achieved by signalling the SONAR MCU that a sweep has been requested. When a sweep is completed, the MCU notifies the server software through shared memory. The obstacles, as detected by the SONAR array, are placed into the map.

Navigation directives, which can be initiated either by the user at the control station or by other software modules running on the server, are passed to the Navigation MCU through shared memory. To allow for emergency stopping, the Navigation MCU code is written so that a new directive will over-ride an older one. Also included in the server code is a “setpoint navigation” search scheme which will automatically drive the workcell to a specified coordinate in the map. This again demonstrates that the system is able to be used for research in control-level and autonomous behaviour research.

6.3.2.3 Server Module Software

This software module contains routines which open and maintain TCP/IP socket connections for connection to the control station. It is also responsible for executing commands (and responding to these commands) issued by the client.

It is within this module that TCL's interpretive nature is exploited. Any incoming communication from the control station is passed directly to the TCL interpreter running the server software. In this way, very flexible control is given over every facet of the workcell system software. It is even possible to send a new procedure from the control station to the server, where it will be immediately made available for execution. In this way, the TCL language effectively becomes the teleoperation language.

When executed, the server module implements a standard input handler which treats typed commands the same as incoming socket commands: they are passed to the TCL interpreter. This technique has proven to be very effective for debugging and providing an interface to aspects of the workcell operation that may not always be of interest and therefore not receive a dedicated UI component on the control station.

To be passed to the TCL interpreter, received commands pass through event handlers. Within these procedures, it is possible to implement communication delay and interruption whereby the design objectives regarding these research areas are met. It is also possible to change the syntax of the commands passed between the control station and the workcell through these handlers. This allows for research to occur in the area of teleoperation languages: another design objective.

Four TCP/IP sockets are used to make a full connection between the control station and the server. These sockets are used to transmit information that is logically

grouped together into the following categories:

- (1) A **Navigation** socket is used to communicate navigation directives (such as “GO forward”) to the workcell. Feedback such as workcell position and orientation is transmitted back to the control station.
- (2) A **Mapping** socket is used to relate all information about the map: barrier information, user-created virtual obstacle creation, etc.
- (3) A **Sensor** socket is used to relate information about the state of individual sensors. Through this socket, it is possible to determine which specific sensor has detected an obstacle.
- (4) A **Message** socket is used to transmit strings received by the server from the microcontroller layer to the control station. It is also used by the server software itself to display messages to the user. This is useful for debugging and informing the user in text form what the workcell is doing.

This socket division has allowed the code on the control station to be written in a modular form. As will be seen, specific UI components are directly related to an individual socket. Another advantage of this separation is the fact that it is possible to terminate control-loops running on one socket, and yet still receive test data back from the workcell. For instance, if a control-loop is implemented on the Navigation socket, its operation could still be evaluated through the Mapping socket. This allows for research in the field of control transfer.

When a socket of a specific type is opened, routines which update the UI are started and run in a periodic manner. This does not, however, mean that the control-station cannot explicitly request information from the workcell control engine.

6.3.3 Programming Interface

To program the workcell, the Linux operating system provides several options: direct programming, through the use of standard UNIX commands (perhaps through the X-window system), or through the server software itself.

The most obvious method of programming the workcell is to connect a monitor and keyboard directly to the workcell PC motherboard. This will provide standard interaction with the Linux operating system.

Since the workcell is connected to the control station using serial PPP, connection can be made by using standard UNIX commands such as `rlogin`, `telnet`, and `ftp`. Using a combination of these, new workcell programs can be developed. Both the control station and the workcell support the X-window environment. It is therefore possible to run the development platform on the workcell, but display it on the control station (or any other Internet machine). This is achieved by setting the `DISPLAY` environment variable on the workcell to display on the desired screen.

As mentioned previously, it is also possible to interact with the server as it is running. Since TCL is an interpreter, it is possible to define new procedures at run-time. This is a rudimentary form of programming as it expands the capability of the server. Additionally, since commands are sent from the control station to the workcell and executed the same way as if they were directly entered at the server, users at the client end can send new commands through one of the sockets opened for communication. These programming techniques can be expanded on. Through this expansion, the teleprogramming research objective is achieved.

6.4 Client (Control-Station) Software and User Interface

This section is a summary of the software that runs on the system control station. There are five modules which will be discussed:

- (1) A **Socket Control module**, used to connect and disconnect the workcell-control station communication link using four different TCP/IP sockets;
- (2) A **Navigation Console module**, used to issue user-instigated control directives to the workcell so the user can drive the robot;
- (3) A **Message module**, which relates debugging and status information to the user about the state of the server and microcontroller software.
- (4) A **Sensor Status Display module**, which graphically depicts the state of the infrared and bump sensors; and
- (5) A **Mapping module**, which provides a graphical display of the workcell's environment and the workcell's position within it.

Prior to presentation of these modules, a discussion of the Tool Command Language (TCL) and its window Tool Kit (TK) as it pertains to the client-side development is presented. As with the workcell (server), TCL proves to be well-suited for implementation of the client side of the client-server relationship.

6.4.1 Development Platform: TCL/TK and C

TCL/TK is a modified version of the TCL interpreter which includes commands used for graphical user interfaces. It has proven to be simple to use and ideal for fast

development. Since the TCL interpreter is integral to TCL/TK all discussion about TCL in the last section also applies here.

Use of TCL on the server side and TCL/TK on the client side of the teleoperation system allows the system software to be written seamlessly. Many routines which are written for the server are also used on the client, again making for fast development time.

The TCL/TK code written for each of the modules is divided into two separate files: one creates the UI, and the other includes procedures used to support the UI.

Each module (except the Socket Control module) receives communication from the workcell on a specific socket. As described earlier, this was done so that each module indeed stands alone and allows simulated interruption of control-loops to be made without cutting off feedback that may be of interest.

A web-browser plugin for TCL/TK exists. The UI components could be modified to become "tclets", usable by the browser. This provides another medium for delivery of the UI. Additionally, this also allows the entire teleoperation system (except for the browser) to be implemented on the workcell. The workcell would include a web-server which would deliver the UI to the client browser.

Fig. 6.3 shows the five individual components that form the user interface.

6.4.2 Socket Control Module

The Socket Control module is responsible for giving the user control over which sockets (Navigation, Message, Mapping, or Sensor) are active. Fig. 6.4 is a screenshot of the socket control module.

The code behind the UI contains procedures which send commands through the sockets, and executes commands received from the workcell (again, passing them to

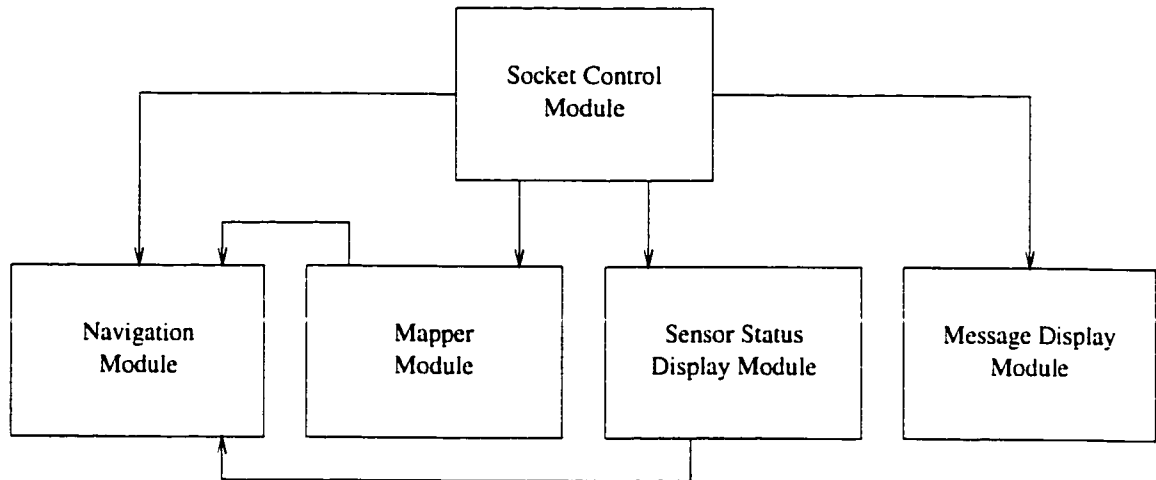


Figure 6.3: The five components that form the user interface. The arrows indicate how data flows between the modules. Each of the four lower modules communicate with the workcell through a dedicated socket.

the TCL interpreter). These routines are similar to their counterparts in the workcell's server software.

6.4.3 Navigation Console Module

The Navigation Console module provides the user with an interface to drive the mobile workcell around. Fig. 6.5 is a screenshot of the socket control module.

Movement on the workcell is limited to translational (forward or backward) and rotational (pivoting left or right). Simplified obstacle feedback is provided so the user knows when an obstacle has been detected, rather than having to look at the Sensor Status Display or Map modules.

Movement setpoints are established through this UI (distance, rotation angle, and speed setpoints) giving the user control over how the navigation directives are achieved.

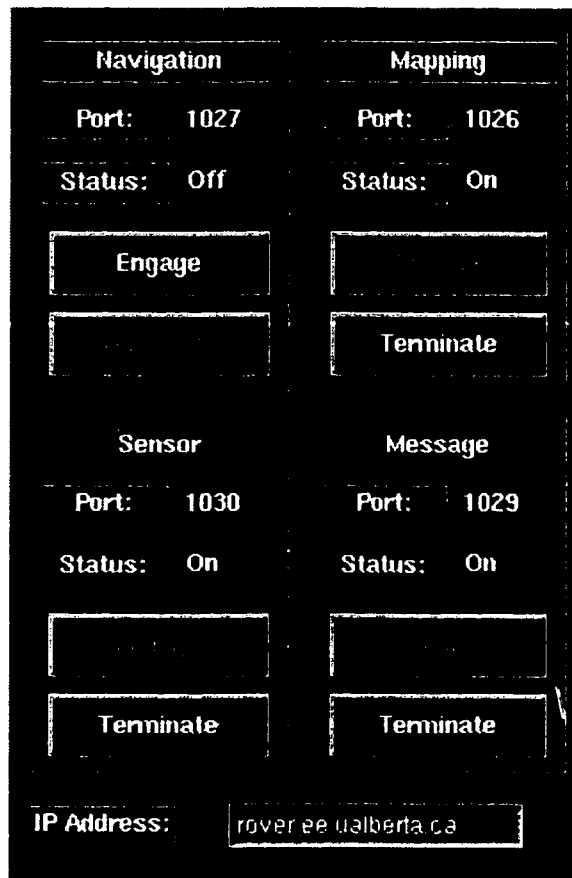


Figure 6.4: A screenshot of the Socket Control GUI module. The user has the ability to terminate or engage specific modules which are used to transfer data of a specific type.

6.4.4 Message Module

The Message module presents debugging and status information from the workcell microcontrollers and server software in text form. Fig. 6.6 is a screenshot of the socket control module.

Four list boxes are used to display information from the following workcell computational elements:

- (1) the Navigation MCU;

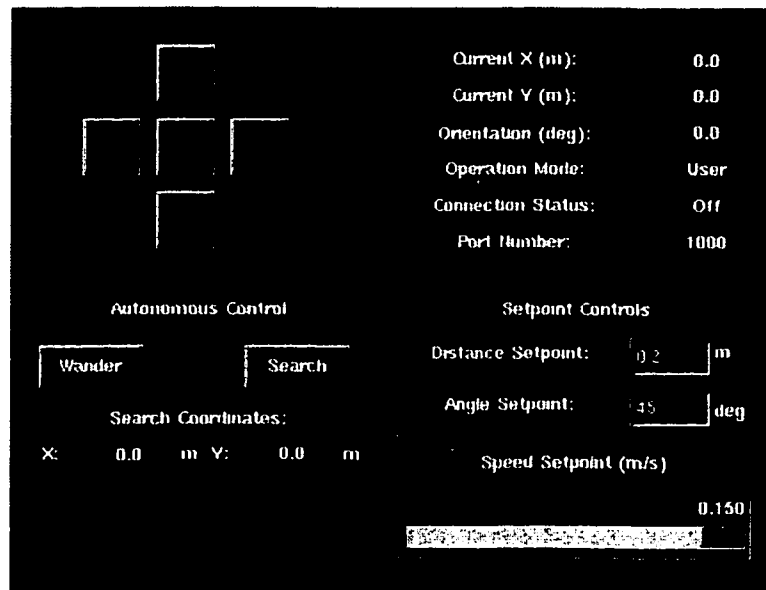


Figure 6.5: A screenshot of the Navigation Console GUI component. The user has the ability to drive the workcell and receives position and simplified barrier feedback from the workcell. Provision is made for execution of autonomous behaviours: Search and Wander. Movement setpoints (speed, distance, and turn angle) are also established through this UI.

- (2) the Proximity MCU;
- (3) the SONAR MCU; and
- (4) the workcell PC motherboard.

Status information that does not warrant a dedicated UI component can still be related to the user in an efficient manner using the Message module.

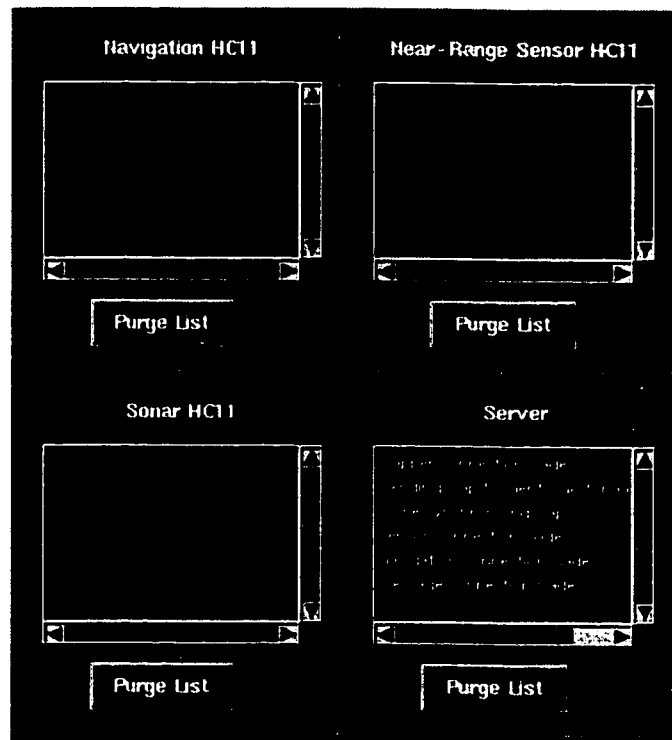


Figure 6.6: A screenshot of the Message module. Each microcontroller and the workcell server itself can display messages to the user. This aids in debugging and allows information to be displayed that may not warrant a dedicated UI component.

6.4.5 Sensor Status Display Module

The Sensor Status Display module displays a graphical rendition of the workcell upon which the status of individual infrared and bump sensors is related. Fig. 6.7 is

a screenshot of the sensor status module.

This type of display has proven to be convenient for obtaining more information about the location of an obstacle than the Mapper module can relate. For instance, the approximate height of an obstacle can be determined by determining which specific sensor has been activated. This module has also proven handy for verification and calibration of specific sensors.

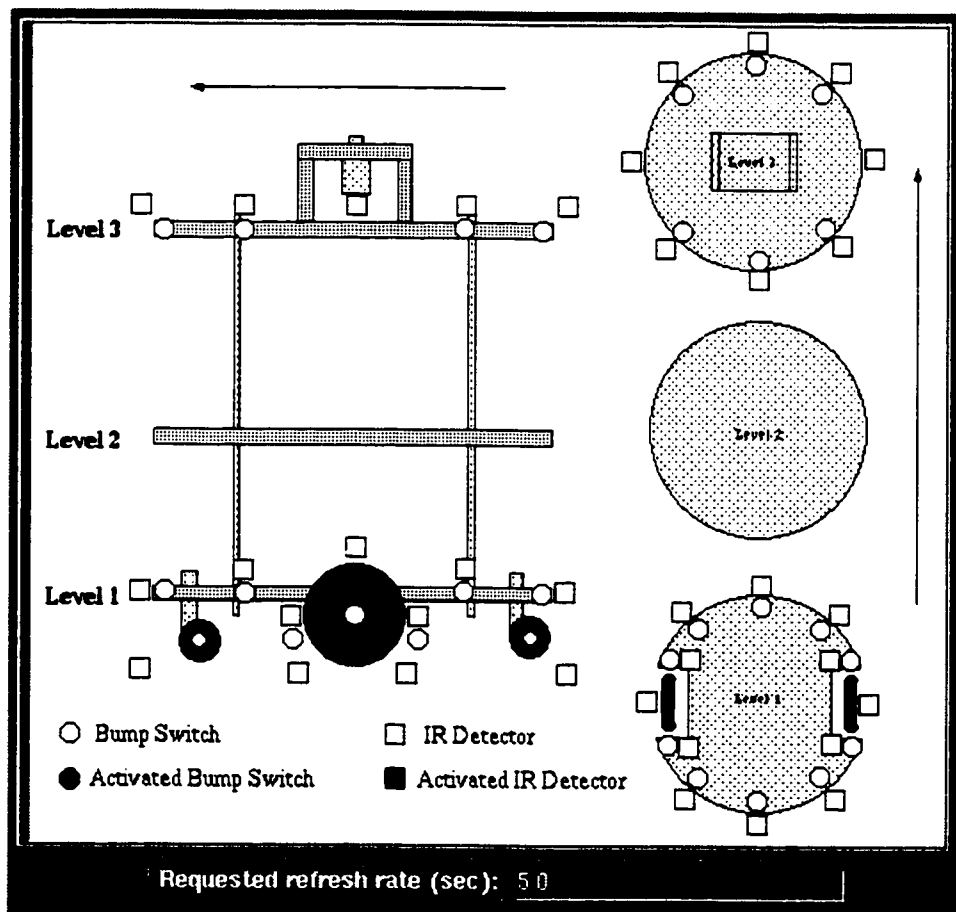


Figure 6.7: A screenshot of the Sensor Status Display module UI. The states of individual bump and infrared sensors is related to the user. If a square is filled, the corresponding bump sensor on the workcell has detected an obstacle. Correspondingly, a filled circle indicates an activated IR sensor.

6.4.6 Mapper Module

The Mapper module presents the user with a flexible view of the workcell's map of the environment. Fig. 6.8 is a screenshot of the Mapper module.

By far the largest module, the Mapper module contains a component in addition to the UI and TCL support code: the C mapping routines. As described earlier, these routines are also used by the workcell server to create and maintain a C representation of mapped environment.

The map displays the obstacle information using occupancy grids. Each "layer" of the map, attributed to a specific sensor type, is displayed in its own colour. Individual points of SONAR obstacle data are displayed after a sweep is initiated. In this way, the user is provided with the information that is necessary to interpret the data.

Through this UI, the user can perform the following actions:

- Clear the map;
- Resize the map;
- Set the map grid resolution (limited only by the accuracy of double-precision floating-point numbers, and the amount of memory available to store the resulting map);
- Display the map at various scales;
- Upload a map to the workcell;
- Download a map from the workcell;
- Load a map from a file;

- Save a map to a file:
- Print a map to an encapsulated postscript file:
- Add, delete, and move text:
- Add “virtual” obstacles to a map:
- Remove obstacles from a map:
- Display the map to show any one of the layers on top:
- Initiate a SONAR sweep:
- Clear SONAR ping data: and
- Select a cell as the goal for autonomous searching.

Obstacle data is volunteered by the workcell after the mapping socket has been opened. SONAR sweeps, however, are instigated by the user.

6.5 Summary

This chapter has outlined the software which runs on the three “layers” of the test bed: the microcontroller layer, the workcell PC (server), and the control station (client). The software was written in a modular manner to promote re-use of code for quick development.

The development platforms include C, the Tool Command Language (TCL) and its windows tool kit, TK. The C routines are compiled for calling from the TCL interpreters. Through the modularity of the software and the nature of TCL,

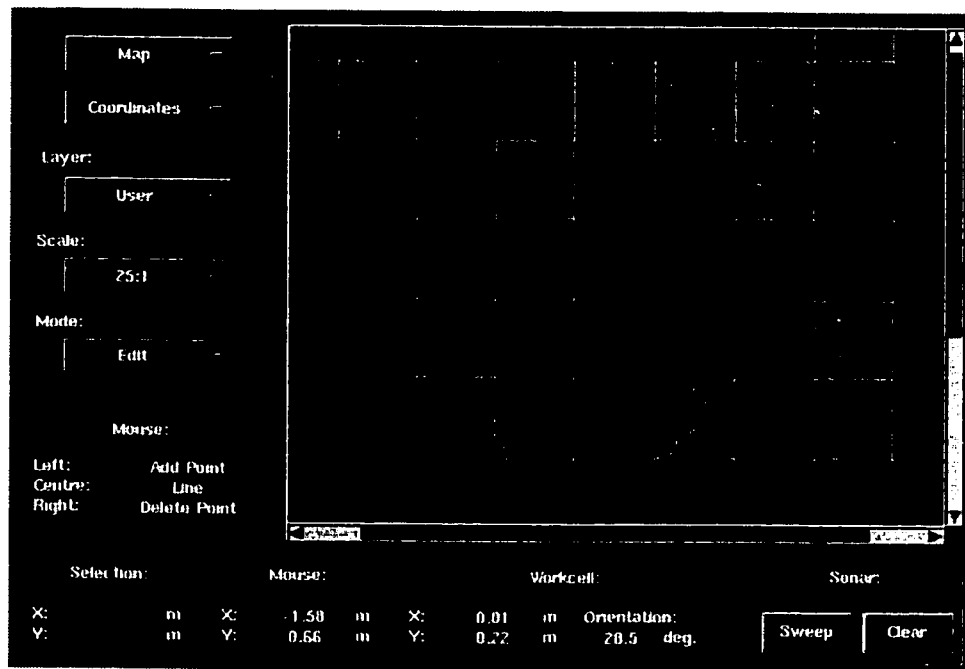


Figure 6.8: A screenshot of the Mapper module UI. All aspects of the map representation and how it is related to the user is controllable through the menu structure.

the rest of the design objectives (not met by hardware design) are achieved. Particularly, research into communication delay and interruption, the syntax of teleoperation languages, control transfer and software division have been addressed.

Information passed back and forth between the workcell and the control station is all string-based, meeting the primary requirement of the RIP paradigm. Some data (such as sensor and workcell coordinate information) is periodically sent to the control station once the socket dedicated to that data-type has been opened. Although the workcell is considered the server, commands are sent to it by the user. When the user presses a button (to go forward, for instance), a string command is sent to the workcell PC. The workcell server software passes this string to the TCL interpreter upon which it, itself, is running. This command then somehow modifies the memory shared with the microcontrollers. If, for instance, the command was to “go forward”, the command modifies the memory shared with the microcontroller responsible for controlling the workcell drive actuators. The MCU acts upon this command.

Chapter 7

Results and Demonstrations

7.1 Introduction

This chapter demonstrates the effectiveness of the system through use of figures showing the actual and mapped environments during workcell movement and sensing. Several tests are used which test specific components of the workcell design.

7.2 Setpoint Movement Test

This section tests the setpoint movement algorithm. The user selects a destination location on the control station's map and instigates the "Search" algorithm which runs on the workcell. This is a functional test of the system's ability to implement autonomous behaviour and its ability to determine its position through odometry data. It is a functional test of the control station Navigation module, the Workcell PC shared-memory interface, Navigation MCU code and navigation hardware: the workcell motors, H-bridge driver, and optical encoder feedback.

Fig. 7.1 shows the starting position and goal location. Fig. 7.2 shows the work-cell enroute to the goal.

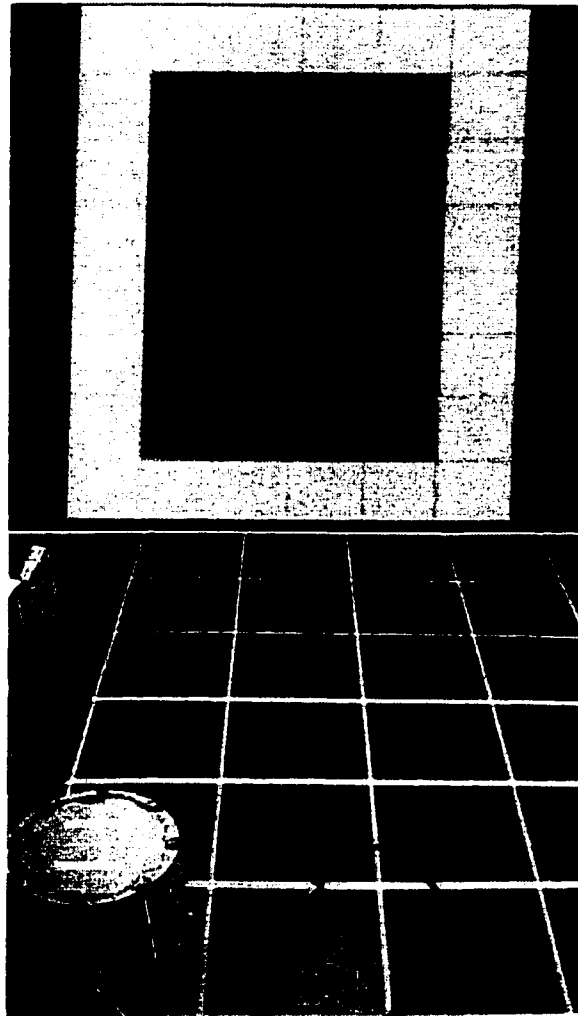


Figure 7.1: Actual and mapped environments of the setpoint search algorithm starting position. The destination, selected by the user, is highlighted on the UI map. The corresponding location in the actual environment is the grid cell one row down, and one column to the left from the top right corner.

As can be seen, correlation between the resulting mapped position and the actual position is high. The search routine successfully located the goal position. Error is due to the dead-reckoning scheme used to calculate workcell position.

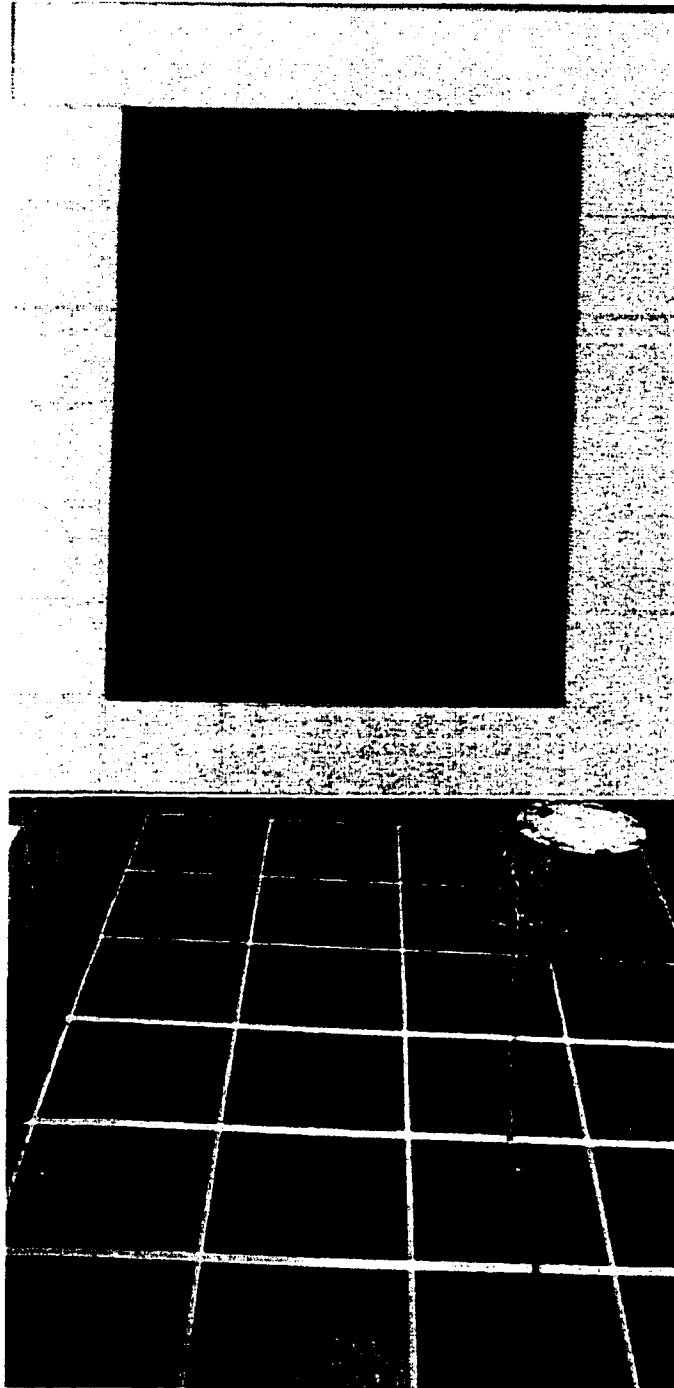


Figure 7.2: Actual and mapped environments of the setpoint search algorithm during execution. The workcell first rotates so that it can drive straight to the destination. It then drives straight until the goal location, or a point close to it is achieved. The workcell is shown attempting to get to the destination with more accuracy during the second pass of the algorithm.

7.3 Bump Sensor Test

This test verifies bump sensor operation. These sensors require direct contact with the obstacle to be activated. This is a functional test of the Proximity Detector MCU Software, the Bump Sensor board, and the bump switches themselves.

Fig. 7.3 shows the actual and mapped environments after the workcell has been driven by the user. The obstacles have been detected using only the bump sensors.

7.4 Infrared Sensor Test

This test verifies near-IR sensor data. The primary advantage of IR sensors over bump sensors is that obstacles do not need to make contact with the workcell to be detected. This is a functional test of the Proximity Detector MCU software, the IR Sensor Board, and the IR sensors themselves.

Fig. 7.4 shows the actual and mapped environments after the workcell has been driven into the “box canyon”.

7.5 SONAR Sensor Test

This test verifies SONAR data. SONAR sensors have the longest detection range of all the sensors on the workcell. This is a functional test of the SONAR MCU software, the Polaroid SONAR modules, the interface board, and the transducers.

Fig. 7.5 shows the actual and mapped environments after a sweep is performed by the SONAR array. Individual data points are related to the user for their inspection. Map cells are filled in when an obstacle is detected within them.

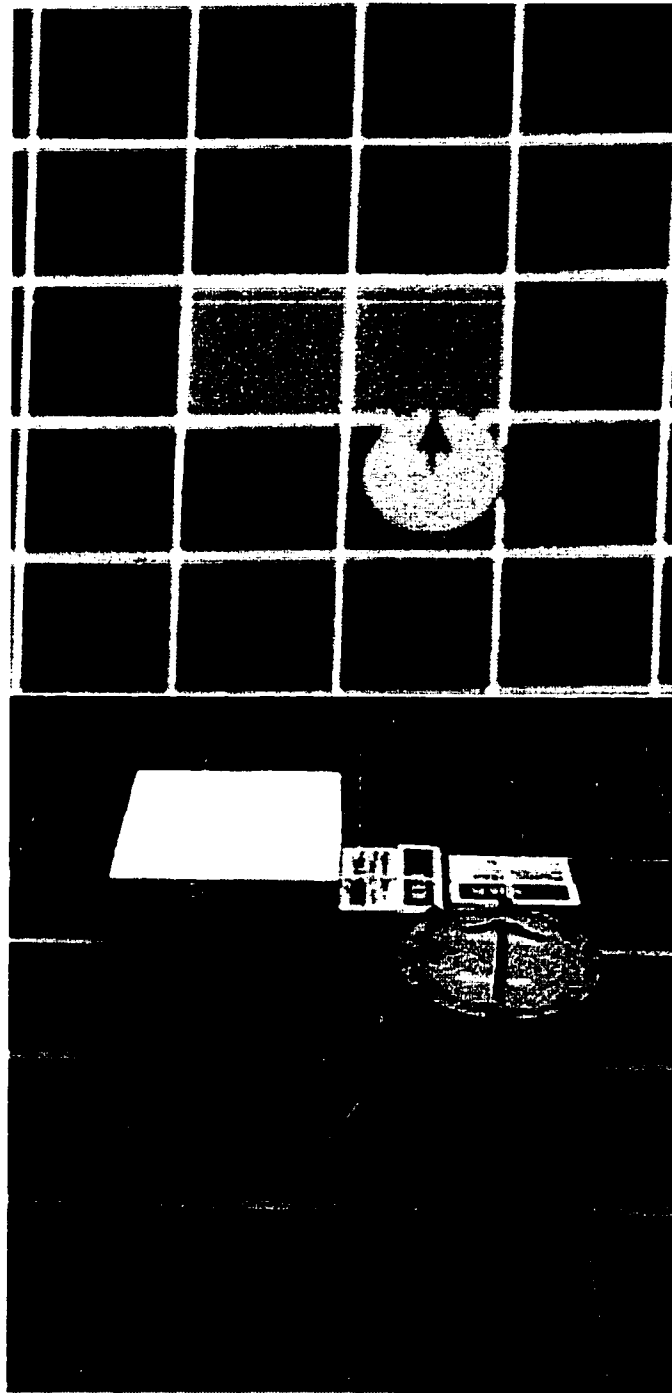


Figure 7.3: Bump sensor test actual and mapped environments. The user has moved the workcell around the environment and detected the obstacles using the bump sensors.

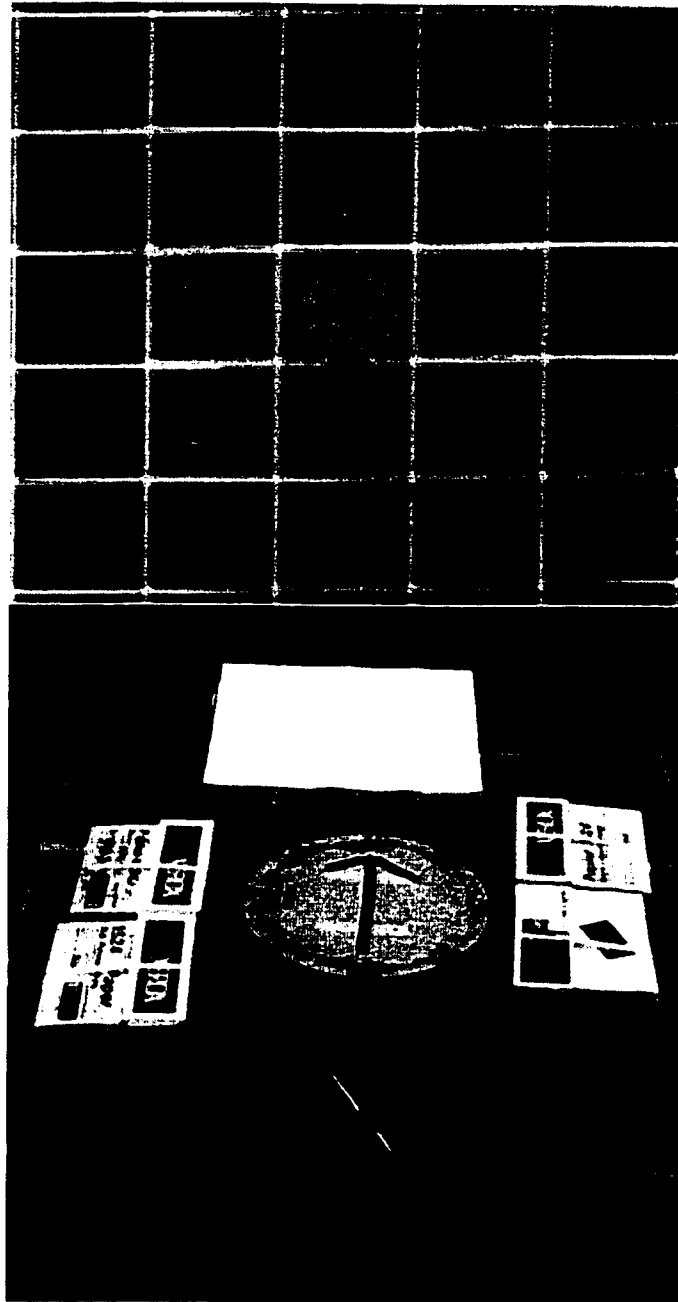


Figure 7.4: IR sensor test actual and mapped environments. The user has moved the workcell into the “box canyon”. The obstacles are detected using IR sensors.

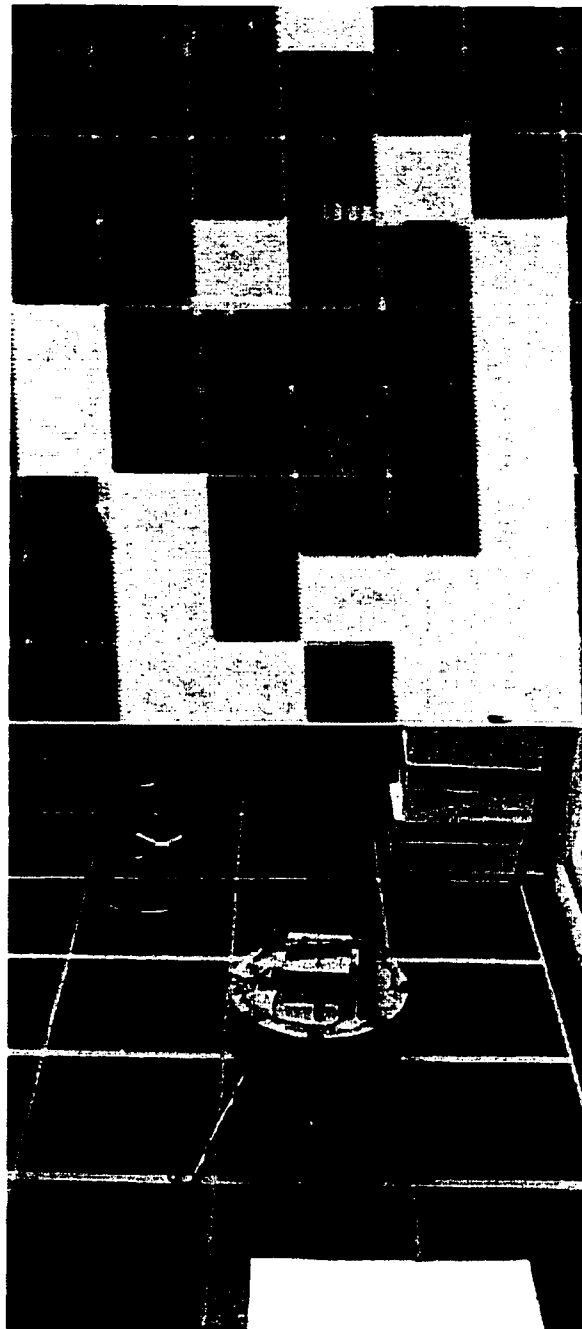


Figure 7.5: SONAR sensor test actual and mapped environments. The user has initiated a sweep of the SONAR sensor array. The map is updated with individual points of data collected from the sweep, and the map cells are updated if they contain an obstacle. The stool and the boxes (ahead and behind) are all successfully detected.

7.6 Summary

This chapter presented tests that demonstrated the abilities of the test-bed: from teleoperated navigation, to building of maps of the environment based on data from the workcell's sensors. All aspects of the system are functional. The suite of sensors and the way the data from it is presented to the user has proven to be effective. Indeed, the user can drive around an environment and build a map that shows the obstacles within it.

Since the entire system has been developed with modularity and flexibility in mind, the tests presented here can be expanded with little effort to research other areas in the field of telerobotics.

Chapter 8

Conclusion

8.1 Summary of Contribution

This project has contributed to the field of telerobotics by presenting the development of an entire telerobotic system and the design models with which it was conceived. From the beginning, the goal has been to provide researchers with the ability to run experiments that provide insight into almost any of the active research areas of the field.

Other architecture projects such as NASREM make the wise move of attempting to decouple a telerobotic system's hardware from the structure of its software. The Robotic Internet Platform design model, presented here, also emphasized the need for decoupling by proclaiming the importance of the workcell-control station interface. Indeed, the system presented here could be made NASREM-compliant with modification to the existing software layers.

It is hoped that to have an entire thoroughly-documented design within the covers of this thesis that telerobotic system **developers** can learn from its faults and

take from it the strengths. It is hoped that **researchers** will find the design flexible enough to accommodate their research with speed and efficiency.

8.2 Evaluation of the Design and Design Technique

The functional specifications outlined in Chapter 3 have been met. The objective was to be able to perform experiments in:

- (1) control research;
- (2) communication research;
- (3) UI functionality and human factor research;
- (4) teleoper robotic software research; and
- (5) sensor fusion and sensing technique research.

The test bed does accommodate these research areas. Application of the test bed design model has resulted in system hardware and software that has modularity, flexibility, and safe operation. Application of the Robotic Internet Platform design paradigm ensured that the resulting design is more-or-less platform-independent, that the control station and workcell are decoupled, and that their string-based interface is well-documented. It is believed that application of the models in parallel has resulted in a researcher-friendly system.

Bibliography

- [BA89] A. T. Barret and Associates. MCX11 – Microcontroller eXecutive for the HC11 version 1.5. Motorola, Inc., 1989.
- [BAXTJ96] Kevin J. Brady, Robert J. Anderson, Ning Xi, and Tarn Tzyh-Jong. Modular controller architecture for multi-arm telerobotic systems. In Proceedings of the IEEE International Conference on Robotics and Automation, volume 2, pages 1024–1029. IEEE, 1996.
- [BBL⁺94] Paul G. Backes, John Beahan, Mark K. Long, Robert D. Steele, Bruce Bon, and Wayne Zimmerman. A prototype ground-remote telerobot control system. In Robotica, volume 12, pages 481–490. Cambridge University Press, 1994.
- [BBZ98] Jonathan Baldwin, Anup Basu, and Hong Zhang. Predictive windows for delay compensation in telepresence applications. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 2884–2889, Leuven, Belgium, May 1998.
- [Bea97] David. M. Beazley. SWIG Users Manual Version 1.1. University of Utah and the Regents of the University of California, March 1997.
- [BFKS94] Antal K. Bejczy, Paolo Fiorini, Won Soo Kim, and Pal Schenker. Toward integrated operator interface for advanced teleoperation under time-delay. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, volume 1, pages 563–570, 1994.
- [BHW⁺94] J. Benner, J. Hanseemann, A. Weber, H. Haffner, and W. Till. Telerobotic control system development in the CEC project TELEMAN-INGRID. In Proceedings of the Fifth World Conference on Robotics Research, pages 5–65–5–78, 1994.

- [BK90] J. Borenstein and Y. Koren. Teleautonomous guidance for mobile robots. IEEE Transactions on Systems, Man, and Cybernetics, 20(6):1437-1443, November/December 1990.
- [CRKW96] Darwin G. Caldwell, K. Reddy, O. Kocak, and A. Wardle. Sensory requirements and performance assessment of tele-presence controlled robots. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 1375-1380, 1996.
- [Cro89] J. Crowley. Asynchronous control of orientation and displacement in a robot vehicle. In Proceedings of the IEEE Conference on Robotics and Automation, pages 1277-1282, 1989.
- [DW87] H. Durrant-Whyte. Consistent integration and propagation of disparate sensor observations. International Journal of Robotics Research, 6(3), 1987.
- [HLF94] G. Hirzinger, K. Landzettel, and Ch. Fagerer. Telerobotics with large time delays - the ROTEX experience. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, volume 1, pages 571-578, 1994.
- [HM94a] D.D. Haule and A. S. Malowany. Real-time teleprogramming of remote robotic workcells. In Proceedings of the 5th World Conference on Robotics Research, pages 5-31-5-49, 1994.
- [HM94b] D.D. Haule and A.S. Malowany. Teleprogramming control paradigm for remote robotic workcells. In Proceedings of the Canadian Conference on Electrical and Computer Engineering, volume 2, pages 802-805, 1994.
- [JF93] Joseph L. Jones and Anita M. Flynn. Mobile Robots - Inspiration to Implementation. A K Peters, 1993.
- [JP94] M.B. Leahy Jr. and S.B. Petroski. Unified telerobotic architecture project program overview. In Proceedings of the International Conference on Intelligent Robots and Systems, volume 1, pages 240-244, 1994.
- [LB94a] Donald G. Langrock and David R. Broome. Advanced telerobotic controller. In IEEE Oceans Conference Record, volume 2, pages 157-162, 1994.
- [LB94b] Trevor Larkum and David Broome. Advanced controller for an underwater manipulator. In Proceedings of the IEEE Conference on Control Applications, volume 2, pages 1081-1086, 1994.

- [LBPL97] John E. Lloyd, Jeffrey S. Beis, Dinesh K. Pai, and David G. Lowe. Model-based telerobotics with vision. In Proceedings of the 1997 IEEE International Conference on Robotics and Automation, pages 1297–1304. Albuquerque, New Mexico, 1997.
- [Leo95] Mark Leon. Establishing a real-time video link to Antarctica. IEEE Network, 9(2):8–15, March/April 1995.
- [LF94] G.M.H. Leung and B.A. Francis. Robust nonlinear control of bilateral teleoperators. In Proceedings of the American Control Conference, pages 2119–2123, 1994.
- [LK89] Ren C. Luo and Michael G. Kay. Multisensor integration and fusion in intelligent systems. IEEE Transactions on Systems, Man, and Cybernetics, 19(5):901–931, September/October 1989.
- [LMR94] E. Le Rest, L. Marcé, and V. Rigaud. VORTEX-PILOT: a top-down approach for AUVs mission telerobotics language. In IEEE Oceans Conference Record, volume 2, pages 102–107, 1994.
- [Lum94] Ronald Lumia. Using NASREM for telerobot control system development. Robotica, volume 12:505–512, 1994.
- [MA94] Paul Michelman and Peter Allen. Shared autonomy in a robot hand teleoperation system. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, volume 1, pages 253–259. IEEE, 1994.
- [MCC92] M. M Mallem, F. Chavand, and E. Colle. Computer-assisted visual perception in teleoperated robotics. Robotica, 10:93–103, 1992.
- [MNBF94] R.S. McMaster, J.H. Nixon, B.G. Boyle, and D. Fouchier. Task enhancement of an underwater robotic arm by graphical simulation techniques. In Oceans Conference Record, volume 2, pages 163–167, 1994.
- [Mor86] H.P. Moravec. Certainty grids for mobile robots. Technical report, Carnegie-Mellon University, 1986. Preprint.
- [Mot91] Motorola, Inc. M68HC11 Reference Manual, 3rd edition, 1991.
- [MS94] Nadine E. Miner and Sharon A. Stansfield. An interactive virtual reality simulation system for robot control and operator training. In Proceedings of the IEEE International Conference on Robotics and Automation, volume 2, pages 1428–1435. IEEE, 1994.

- [NX94] M. C. Nechyba and Y. Xu. SM² for new space station structure: Autonomous locomotion and teleoperation control. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, volume 2, pages 1765–1770. 1994.
- [PPA94] Jerome Perret, Christophe Proust, and Rachid Alami. How to teleprogram a remote intelligent robot. In Proceedings of the International Conference on Intelligent Robots and Systems, pages 397–404. 1994.
- [RCMT94] A. Rovetta, F. Cosmi, L. Molinari Tosatti, and L. Termite. Evaluation of human control in telerobotics by means of EMG. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, volume 1, pages 268–272. 1994.
- [RKSG94] Catherine Richards, Larry Korba, Chris Shaw, and Mark Green. Virtual reality and virtual bodies. In Proceedings of SPIE – The International Society for Optical Engineering, volume 2177, pages 386–396. 1994.
- [RVR98] Reinhard Roll, Michel Van Winnendael, and Lutz Richter. The development of a European micro-robot for planetary surface exploration. In Proceedings of the Conference and Exposition on Robotics for Challenging Environments, pages 174–180. American Society of Civil Engineers. 1998.
- [Say96] Craig P. Sayers. Intelligent image fragmentation for teleoperation over delayed low-bandwidth links. In Proceedings of the IEEE International Conference on Robotics and Automation, volume 2, pages 1363–1368. Minneapolis, Minnesota. 1996.
- [SB94] Robert Steele and Paul Backes. Ada and real-time robotics: Lessons learned. Computer, 27(4):49–54. April 1994.
- [SJ94] Philip F. Spelt and Sammy L. Jones. Operator-centered control of a semi-autonomous industrial telerobot. In Proceedings of the Human Factors and Ergonomics Society, volume 2. 1994.
- [SLMV98] Roland Siegwart, Michel Lauria, Pierre-Alain Mausli, and Michel Van Winnendael. Design and implementation of an innovative micro-rover. In Proceedings of the Conference and Exposition on Robotics for Challenging Environments, pages 154–160. American Society of Civil Engineers. 1998.
- [STS⁺95] M. Saitoh, Y. Takahashi, A. Sankaranarayanan, H. Ohmachi, and K. Marukawa. A mobile robot testbed with manipulator for security

- guard application. In Proceedings of 1995 IEEE International Conference on Robotics and Automation, volume 2, pages 2518–2523. IEEE, 1995.
- [TBGX94] Tzyh-Jong Tarn, Antal K. Bejczy, Chuanfan Guo, and Ning Xi. Intelligent planning and control for telerobotic operations. In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, volume 1, pages 389–396. IEEE, 1994.
- [Wel95] Brent B. Welch. Practical Programming in TCL and TK. Prentice Hall PTR, 1995.
- [WL94] C.R. Weisbin and D. Lavery. NASA Rover and Telerobotics Technology Program. In IEEE Robotics & Automation Magazine, volume 1, number 4, pages 14–21. IEEE, Dec. 1994.
- [WSFM98] L. Wyard-Scott, R. Frey, and Q. H. Meng. A robotic internet workstation design paradigm. In Proceedings of the Conference and Exposition on Robotics for Challenging Environments, pages 251–257. American Society of Civil Engineers, 1998.
- [WSM96] L. Wyard-Scott and Q.-H. M. Meng. Use of image warping techniques to correct and interpret infrared sensor data for navigation. In Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, pages 931–934, 1996.
- [YMP⁺94] S.K. Yeung, W.S. McMath, E.M. Petriu, N. Trif, and C. Gal. Teleoperator-aided multi-sensor data fusion for mobile robot navigation. In Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pages 470–476, 1994.

Appendix A

MC68HC11 Microcontroller PCB

A.1 Introduction

An outline of the operation, features, and user-settable options of the the 68HC11 Microcontroller (MCU) board is given here. This board is intended to be connected to the workcell PC via the MCU-to-MCU shared-memory PCBs, although they are also designed to work as stand-alone devices.

The MCUs are connected directly to the sensor and actuator PCBs and therefore allow for software-directed signal conditioning. The HC11 has also proven to be powerful enough to allow for reasonably complex sensor-fusion algorithms and implementation of autonomous (and reflexive) behaviours.

A block diagram of the board electronics is shown in Fig. 5.6. The design is a complete, stand-alone, single-board computer. Once an application is coded, it can be run on the MCU board with no external interface (an embedded system).

Figs. A.1 through A.7, contained later in this chapter, show the schematic of the MCU board. Fig. A.8 is a diagram showing the part placement. Figs. A.9 and A.10 show the top and bottom copper foil patterns, respectively.

A.2 Operation

As mentioned previously, the HC11 MCU board is a single-board computer. It consists of seven logical blocks, each of which are discussed in more detail below:

- (1) the HC11 microcontroller IC;
- (2) control-line circuitry;
- (3) address latching;
- (4) static RAM (SRAM);

- (5) address decoding;
- (6) MCU bus connector; and
- (7) MCU port connector.

A.2.1 The HC11 Microcontroller IC

The HC11 microcontroller is the MC68HC11A1FN, which is one of the first HC11s that entered the market. It utilizes a multiplexed address/data bus if used in the 'expanded multiplexed mode' (the normal mode of operation for the teleoperation system). It contains 512 bytes of EEPROM and 256 bytes of internal RAM. The address space of the MCU (once demultiplexed) is 64K, which is more than sufficient for the example application.

There is a variety of support circuitry required to make the IC operate as desired, most of which is discussed in the next subsection. The HC11 contains an on-board oscillator that utilizes an external crystal. The frequency of the crystal is 8.000 MHz, which is the frequency expected by Interactive-C. Interactive-C utilizes this crystal frequency to set, among other things, the serial communications baudrate, and the multitasking executive 'tick' time.

The HC11 has many subsystems implemented directly on the IC, some of which are listed here:

- (1) SCI (asynchronous serial interface);
- (2) SPI (synchronous serial interface);
- (3) Multi-function timer;
- (4) Input capture;
- (5) Output compare; and
- (6) Analog to digital converters.

All of these systems are used in the teleoperation system.

A.2.2 Control-Line Circuitry

The HC11 makes use of several control lines to configure its mode of operation. The first is a RESET* signal which is driven by a MC34064 Low-Voltage Inhibit (LVI). This IC is responsible for placing the HC11 in the known 'reset state' when the supply voltage is unstable. This instability occurs during voltage ramp-up during application of power and also during power-down. This 'reset state' is essentially an inert mode which will not alter the contents of RAM. Since the MC34064 drives the

RESET* pin with an open-drain output, other sources of reset can be attached in a wired-or configuration: a push-button switch, or a PC control line.

Maskable and nonmaskable interrupt pins (IRQ* and XIRQ*, respectively) are made available off-board through two connectors (discussed below). Again, these signals are configured to be driven by open-drain (or open-collector) outputs, and thus have pull-up resistors connected.

Lastly, one control line, DWNLD*, is used to configure the operating mode of the HC11. When pulled low, the HC11 is placed in the 'Special Bootstrap' mode at which point a small program is expected to be uploaded (usually this program is a more comprehensive downloader). This mode is used to upload the Interactive-C pseudo-code interpreter. When DWNLD* is high, MODA and MODB are high, placing the HC11 in 'Expanded-Multiplexed Mode' which enables the external multiplexed address and data busses (via HC11 ports C and D). As with the other control signals, the DWNLD* signal may be driven by multiple sources. In the system presented here, the workcell PC controls the line, although provision is made to allow connection of a switch.

A.2.3 Address Latching

Address demultiplexing is performed via an 'HC573 transparent latch. The output of this latch forms the least-significant byte of the 16-bit address word. The HC11, intended for this type of demultiplexing, controls the latch via an address strobe (AS) signal.

A.2.4 Static RAM

A 62256 (32Kx8) static RAM is the only memory device required on the board. The discretely-derived address decoding places it in the upper-half of the memory map (0x8000-0xFFFF). As the HC11 is not a terribly fast MCU, slow versions of this IC can be utilized.

The memory IC is backed-up using battery power. This is apparent in the power connection to the IC, and to the 'HC10 which performs the address decoding. For the memory to be selected, all three of the inputs to the 'HC10 gate need to be high: A15, LVI, and E. The E-clock is used to 'E-qualify' the chip-enable signal: a requirement for the HC11 to successfully use its external address and data bus. The need arises due to timing requirements placed upon memory devices because of the dual-purpose address/data lines (AD0-AD7). Half the time (while E is low), AD0-7 act as the low-byte of the address, and the other half, they form the data bus. The LVI (low-voltage inhibit) signal originates from the MC34064 IC and goes low when the logic voltage is low enough that digital devices (the MCU, in particular) do not operate predictably. Inclusion of this signal in the chip-enable circuitry protects the

RAM against potentially having its memory corrupted as the MCU board is being powered-down.

A.2.5 Address Decoding

Eight chip-select lines are derived on-board using an 'HC138 decoder. These outputs evenly divide the lower-half of the HC11's logical address space into 4K blocks. As with the SRAM, the chip-select lines are E-qualified for successful interface to devices that may be connected.

The following table outlines the memory ranges decoded by specific 'HC138 outputs.

Output	Mnemonic	Decoded Range
Y0	UC_CS0*	0x0000-0x0FFF
Y1	UC_CS1*	0x1000-0x1FFF
Y2	UC_CS2*	0x2000-0x2FFF
Y3	UC_CS3*	0x3000-0x3FFF
Y4	UC_CS4*	0x4000-0x4FFF
Y5	UC_CS5*	0x5000-0x5FFF
Y6	UC_CS6*	0x6000-0x6FFF
Y7	UC_CS7*	0x7000-0x7FFF

Table A.1: Address decoding ('HC138) ranges.

A.3 User-settable Options

The user has the ability to configure or control the MCU board in the following capacities:

- (1) Reset the HC11 via a push-button switch (SW1):
- (2) Set the operating mode or manually interrupt the CPU via J2: and
- (3) Select the logic power source via JP1.

Each of these features receives brief attention in a dedicated subsection.

A.3.1 Reset Switch: SW1

When pressed, the reset switch places the HC11 in a known start-up condition. The CPU fetches a reset vector, the location of which depends upon the selected operating mode, outlined below. For more information, see [Mot91]. Note that the RESET* signal can also be accessed off-board via either the Port or Bus connectors (J4 and J1, respectively).

A.3.2 CPU Mode Control: J2

This connector may serve either as a connector to external switches or as a jumper-block, depending on the application. The following table outlines the purpose of the connector positions.

Position	Purpose
1-2	Place MCU into reset when closed.
3-4	Interrupt the MCU via the IRQ* pin when closed.
5-6	Place the HC11 in 'download' mode when closed.

Table A.2: MCU board J2 jumper position definitions.

These signals are all brought to the edge of the board via J1 and J4.

A.3.3 Power Select: JP1

The user has the ability to select where the logic power for the MCU board originates. This selection, performed via a jumper, was included so that system power cabling requirements would be less stringent. The following table outlines the use of JP1.

Position	Purpose
1-2	Utilize power from J1 (off-board power).
3-4	Utilize power from J3 (on-board power).

Table A.3: MCU board JP1 jumper position definitions.

A.4 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the microcontroller board.

A.4.1 Bus Connector: J1

Connector J1 is used to interface to other boards which make use of the HC11 external address and data buses. This is also the connector which accepts control

signals from the workcell PC via the Parallel Expansion PCB. The following table uses the mnemonics SPI and SCI, defined here:

SPI Serial Peripheral Interface (synchronous serial system)

SCI Serial Communications Interface (asynchronous serial system)

Bus Connector – J1			
Number	Mnemonic	Direction	Purpose
1	PC5V	In	+5V logic power (selected via J2).
2	PC5V	In	+5V logic power (selected via J2).
3	+12V	In	+12V power (unused).
4	-12V	In	-12V power (unused).
5	GND	Out	DC power return.
6	GND	Out	DC power return.
7	-5V	In	-5V power (unused).
8	N.C.	-	No connection.
9	N.C.	-	No connection.
10	UC_PA3	Out	Output from HC11 port A3 (Piezo OC).
11	UC_R/W*	Out	RAM access Read/Write signal.
12	UC_E	Out	HC11 E-clock (muxed bus address timing).
13	UC_RESET*	In	Connection to HC11 RESET*.
14	UC_DWNLD	In	HC11 mode control.
15	UC_IRQ*	In	Connection to HC11 IRQ*.
16	UC_XIRQ*	In	Connection to HC11 XIRQ*.
17	UC_PC4	In	Unused line from Parallel Expansion PCB.
18	UC_PC5	In	Unused line from Parallel Expansion PCB.
19	UC_PC6	In	Unused line from Parallel Expansion PCB.
20	UC_PC7	In	Unused line from Parallel Expansion PCB.
21	UC_PD0	I/O	Connection to Port D0 (SCI).
22	UC_PD1	I/O	Connection to Port D1 (SCI).
23	UC_PD2	I/O	Connection to Port D2 (SPI).
24	UC_PD3	I/O	Connection to Port D3 (SPI).
25	UC_PD4	I/O	Connection to Port D4 (SPI).
26	UC_PD5	I/O	Connection to Port D5 (SPI).
(continued)			

Bus Connector – J1 (continued)			
Number	Mnemonic	Direction	Purpose
27	UC_AD0	I/O	Multiplexed address/data (LSB).
28	UC_AD1	I/O	Multiplexed address/data.
29	UC_AD2	I/O	Multiplexed address/data.
30	UC_AD3	I/O	Multiplexed address/data.
31	UC_AD4	I/O	Multiplexed address/data.
32	UC_AD5	I/O	Multiplexed address/data.
33	UC_AD6	I/O	Multiplexed address/data.
34	UC_AD7	I/O	Multiplexed address/data (MSB).
35	N.C.	-	No connection.
36	N.C.	-	No connection.
37	UC_A7	Out	Demultiplexed address.
38	UC_A6	Out	Demultiplexed address.
39	UC_A5	Out	Demultiplexed address.
40	UC_A4	Out	Demultiplexed address.
41	UC_A3	Out	Demultiplexed address.
42	UC_A2	Out	Demultiplexed address.
43	UC_A1	Out	Demultiplexed address.
44	UC_A0	Out	Demultiplexed address (LSB).
45	UC_A8	Out	Demultiplexed address.
46	UC_A9	Out	Demultiplexed address.
47	UC_A10	Out	Demultiplexed address.
48	UC_A11	Out	Demultiplexed address.
49	UC_A12	Out	Demultiplexed address.
50	UC_A13	Out	Demultiplexed address.
51	UC_A14	Out	Demultiplexed address.
52	UC_A15	Out	Demultiplexed address (MSB).
53	N.C.	-	No connection.
54	N.C.	-	No connection.
55	UC_CS7*	Out	Chip select (0x7000–0x7FFF).
56	UC_CS6*	Out	Chip select (0x6000–0x6FFF).
57	UC_CS5*	Out	Chip select (0x5000–0x5FFF).
58	UC_CS4*	Out	Chip select (0x4000–0x4FFF).
59	UC_CS3*	Out	Chip select (0x3000–0x3FFF).
60	UC_CS2*	Out	Chip select (0x2000–0x2FFF).

Table A.4: MCU Board bus connector pinouts (J1).

A.4.2 Power Connector: J3

If JP1 is set accordingly, this screw-terminal connector is used to provide +5V power to the MCU board.

Power Connector – J3			
Number	Mnemonic	Direction	Purpose
1	Vcc	In	+5V logic power.
2	GND	In	Ground return.

Table A.5: MCU Board power connector pinouts (J3).

A.4.3 Port Connector: J4

This connector is used to connect to peripherals that require interface to the specialized ports of the HC11. The following table uses the mnemonics IC, OC, and PA when describing the purpose of signals. Definitions for these mnemonics are provided here for convenience.

IC Input Capture

OC Output Compare

PA Pulse Accumulator

For more information on these HC11 resources, see [Mot91].

Port Connector – J4			
Number	Mnemonic	Direction	Purpose
1	Vcc	Out	+5V logic power.
2	Vcc	Out	+5V logic power.
3	+12V	In	+12V power (unused).
4	-12V	In	-12V power (unused).
5	GND	Out	DC power return.
6	GND	Out	DC power return.
7	-5V	In	-5V power (unused).
8	N.C.	-	No connection.
9	N.C.	-	No connection.
10	N.C.	-	No connection.
(continued)			

Port Connector – J4 (continued)			
Number	Mnemonic	Direction	Purpose
11	UC_PE7	In	Input to HC11 A/D converter.
12	UC_PE6	In	Input to HC11 A/D converter.
13	UC_PE5	In	Input to HC11 A/D converter.
14	UC_PE4	In	Input to HC11 A/D converter.
15	UC_PE3	In	Input to HC11 A/D converter.
16	UC_PE2	In	Input to HC11 A/D converter.
17	UC_PE1	In	Input to HC11 A/D converter.
18	UC_PE0	In	Input to HC11 A/D converter.
19	N.C.	-	No connection.
20	N.C.	-	No connection.
21	UC_PA0	In	Input to HC11 Port A0 (IC).
22	UC_PA1	In	Input to HC11 Port A1 (IC).
23	UC_PA2	In	Input to HC11 Port A2 (IC).
24	UC_PA3	Out	Output from HC11 Port A3 (Piezo OC).
25	UC_PA4	Out	Output from HC11 Port A4 (OC).
26	UC_PA5	Out	Output from HC11 Port A5 (OC).
27	UC_PA6	Out	Output from HC11 Port A6 (OC).
28	UC_PA7	I/O	I/O connected to HC11 Port A7 (PA).
29	N.C.	-	No connection.
30	N.C.	-	No connection.
31	UC_PD5	I/O	Connection to Port D5 (SPI).
32	UC_PD4	I/O	Connection to Port D4 (SPI).
33	UC_PD3	I/O	Connection to Port D3 (SPI).
34	UC_PD2	I/O	Connection to Port D2 (SPI).
35	N.C.	-	No connection.
36	UC_XIRQ*	In	Connection to HC11 XIRQ*.
37	UC_IRQ*	In	Connection to HC11 IRQ*.
38	UC_RESET*	In	Connection to HC11 RESET*.
39	UC_R/W*	Out	RAM access Read/Write signal.
40	UC_E	Out	HC11 E-clock (muxed bus address timing).

Table A.6: MCU Board port connector pinouts (J4).

A.4.4 Piezo Buzzer Connector: J5

Interactive-C includes libraries for connection of a piezo buzzer to an output compare pin of the HC11. This connector is provided specifically for this peripheral.

Piezo Buzzer Connector – J5			
Number	Mnemonic	Direction	Purpose
1	UC_PA3	Out	Output-compare signal to piezo buzzer.
2	GND	Out	Ground return.

Table A.7: MCU Board piezo buzzer connector pinouts (J5).

A.4.5 Serial Data Connector: J6

TTL serial signals are sent to, and received from, the Serial-Level Conversion PCB through this connector.

Serial Data Connector – J6			
Number	Mnemonic	Direction	Purpose
1	UC_PD1 (UC_RX_DATA)	In	TTL serial data input.
2	Vcc	Out	+5V to Serial-Level Conversion PCB.
3	GND	Out	Ground return.
4	UC_PD0 (UC_TX_DATA)	Out	TTL serial data output.

Table A.8: MCU Board serial data connector pinouts (J6).

A.5 Future Improvements

The following are a list of potential future improvements of this microcontroller board.

- Due to a PCB footprint error, all MCUs are mounted on the wrong side of the board.
- Include a socket for either EPROM or EEPROM.
- Include on-board power regulation.
- Replace low-voltage inhibit circuitry for the SRAM and reset circuitry with a Maxim Semiconductor IC, capable of performing both functions.
- Place RS-232 level conversion directly on the PCB, rather than including it as another module.
- Make the board surface-mount to reduce space and power requirements.

A.6 Schematic Diagrams and Board Layout

Figures A.1 through A.10 comprise the schematic and board layout of the Motorola 68HC11 microcontroller board.

A.6.1 Part List

Microcontroller Board Parts			
	Number Used	Part Type	Designators
1	1	0.01uF	C2
2	5	0.1uF	C6-C10
3	2	1N4004 Diode	D1 D2
4	1	1uF	C3
5	1	3.6V Lithium Battery	B1
6	1	4.7K	RP1
7	1	8.000 MHz Crystal	XTAL1
8	1	10M	R1
9	1	10uF	C1
10	2	27pF	C4 C5
11	1	47uF	C11
12	1	62256-100ns SRAM	U5
13	1	2-contact screw terminal	J3
14	1	2-pin Header	J5
15	1	3x2-pin Header	J2
16	1	4-pin Header	JP1
17	1	20x2-pin Header	J4
18	1	30x2-pin Header	J1
19	1	MC68HC11A1FN MCU	U2
20	1	MC74HC10	U6
21	1	MC74HC138A	U4
22	1	MC74HC573	U3
23	1	MC34064	U1
24	1	4-pin Molex header	J6
25	1	Push-button switch	SW1

Table A.9: Microcontroller board parts list (3 constructed).

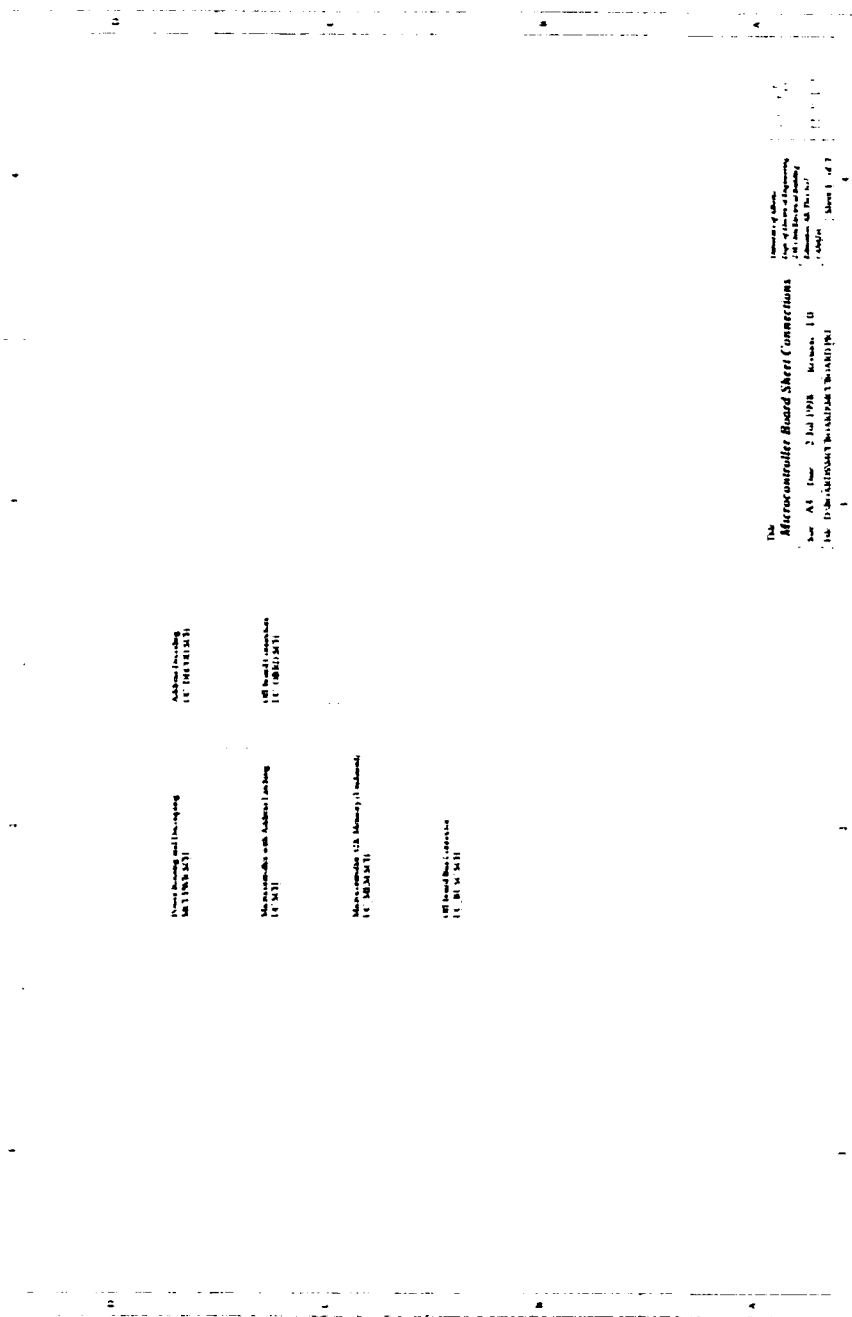


Figure A.1: Microcontroller board schematic (1 of 7).

Figure A.2: Microcontroller board schematic (2 of 7).

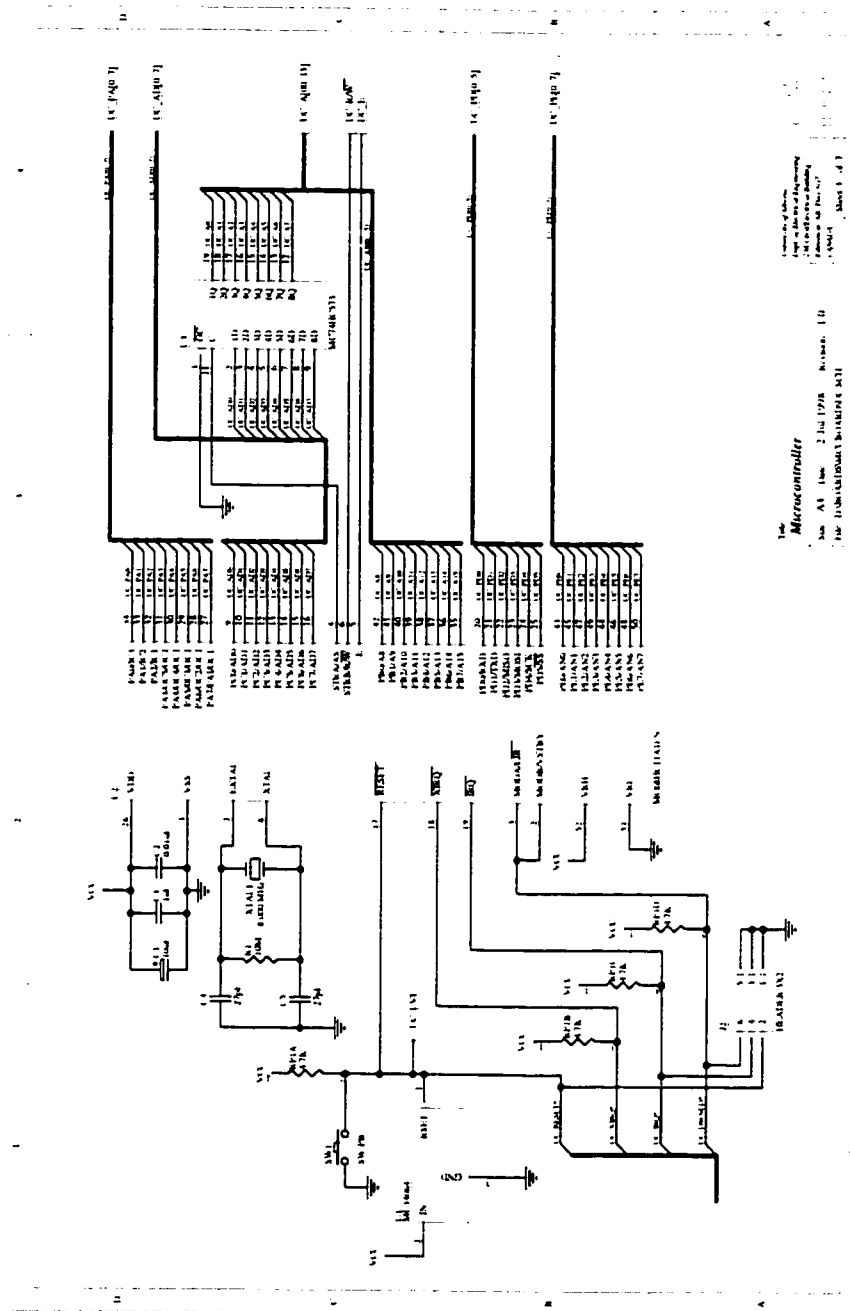
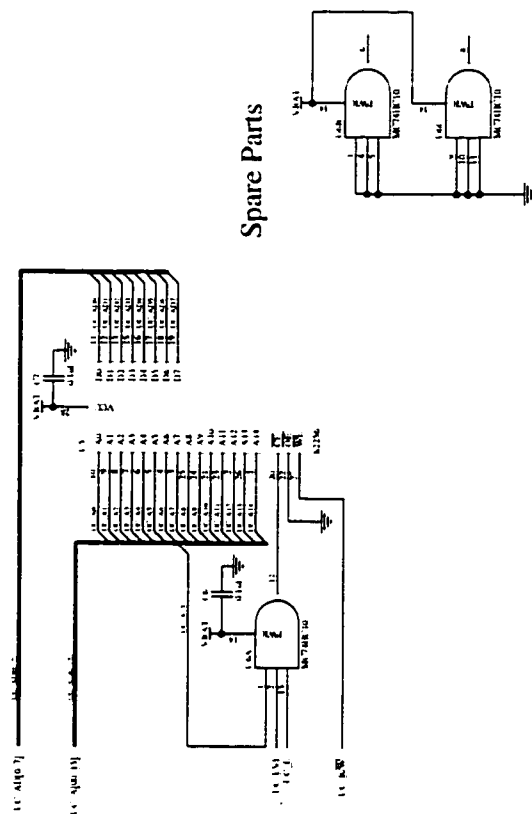
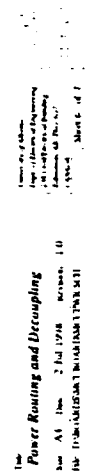


Figure A.3: Microcontroller board schematic (3 of 7).

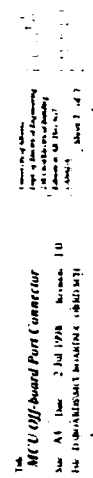


Spare Parts

1. Microcontroller Memory (Unshared)
 2. 74VHC125
 3. 74VHC123
 4. 10k
 5. 100nF
 6. 5V
 7. 10k
 8. 100nF
 9. 5V
 10. 10k
 11. 100nF
 12. 5V
 13. 10k
 14. 100nF
 15. 5V
 16. 10k
 17. 100nF
 18. 5V
 19. 10k
 20. 100nF
 21. 5V
 22. 10k
 23. 100nF
 24. 5V
 25. 10k
 26. 100nF
 27. 5V
 28. 10k
 29. 100nF
 30. 5V
 31. 10k
 32. 100nF
 33. 5V
 34. 10k
 35. 100nF
 36. 5V
 37. 10k
 38. 100nF
 39. 5V
 40. 10k
 41. 100nF
 42. 5V
 43. 10k
 44. 100nF
 45. 5V
 46. 10k
 47. 100nF
 48. 5V
 49. 10k
 50. 100nF
 51. 5V
 52. 10k
 53. 100nF
 54. 5V
 55. 10k
 56. 100nF
 57. 5V
 58. 10k
 59. 100nF
 60. 5V
 61. 10k
 62. 100nF
 63. 5V
 64. 10k
 65. 100nF
 66. 5V
 67. 10k
 68. 100nF
 69. 5V
 70. 10k
 71. 100nF
 72. 5V
 73. 10k
 74. 100nF
 75. 5V
 76. 10k
 77. 100nF
 78. 5V
 79. 10k
 80. 100nF
 81. 5V
 82. 10k
 83. 100nF
 84. 5V
 85. 10k
 86. 100nF
 87. 5V
 88. 10k
 89. 100nF
 90. 5V
 91. 10k
 92. 100nF
 93. 5V
 94. 10k
 95. 100nF
 96. 5V
 97. 10k
 98. 100nF
 99. 5V
 100. 10k
 101. 100nF
 102. 5V
 103. 10k
 104. 100nF
 105. 5V
 106. 10k
 107. 100nF
 108. 5V
 109. 10k
 110. 100nF
 111. 5V
 112. 10k
 113. 100nF
 114. 5V
 115. 10k
 116. 100nF
 117. 5V
 118. 10k
 119. 100nF
 120. 5V
 121. 10k
 122. 100nF
 123. 5V
 124. 10k
 125. 100nF
 126. 5V
 127. 10k
 128. 100nF
 129. 5V
 130. 10k
 131. 100nF
 132. 5V
 133. 10k
 134. 100nF
 135. 5V
 136. 10k
 137. 100nF
 138. 5V
 139. 10k
 140. 100nF
 141. 5V
 142. 10k
 143. 100nF
 144. 5V
 145. 10k
 146. 100nF
 147. 5V
 148. 10k
 149. 100nF
 150. 5V
 151. 10k
 152. 100nF
 153. 5V
 154. 10k
 155. 100nF
 156. 5V
 157. 10k
 158. 100nF
 159. 5V
 160. 10k
 161. 100nF
 162. 5V
 163. 10k
 164. 100nF
 165. 5V
 166. 10k
 167. 100nF
 168. 5V
 169. 10k
 170. 100nF
 171. 5V
 172. 10k
 173. 100nF
 174. 5V
 175. 10k
 176. 100nF
 177. 5V
 178. 10k
 179. 100nF
 180. 5V
 181. 10k
 182. 100nF
 183. 5V
 184. 10k
 185. 100nF
 186. 5V
 187. 10k
 188. 100nF
 189. 5V
 190. 10k
 191. 100nF
 192. 5V
 193. 10k
 194. 100nF
 195. 5V
 196. 10k
 197. 100nF
 198. 5V
 199. 10k
 200. 100nF
 201. 5V
 202. 10k
 203. 100nF
 204. 5V
 205. 10k
 206. 100nF
 207. 5V
 208. 10k
 209. 100nF
 210. 5V
 211. 10k
 212. 100nF
 213. 5V
 214. 10k
 215. 100nF
 216. 5V
 217. 10k
 218. 100nF
 219. 5V
 220. 10k
 221. 100nF
 222. 5V
 223. 10k
 224. 100nF
 225. 5V
 226. 10k
 227. 100nF
 228. 5V
 229. 10k
 230. 100nF
 231. 5V
 232. 10k
 233. 100nF
 234. 5V
 235. 10k
 236. 100nF
 237. 5V
 238. 10k
 239. 100nF
 240. 5V
 241. 10k
 242. 100nF
 243. 5V
 244. 10k
 245. 100nF
 246. 5V
 247. 10k
 248. 100nF
 249. 5V
 250. 10k
 251. 100nF
 252. 5V
 253. 10k
 254. 100nF
 255. 5V
 256. 10k
 257. 100nF
 258. 5V
 259. 10k
 260. 100nF
 261. 5V
 262. 10k
 263. 100nF
 264. 5V
 265. 10k
 266. 100nF
 267. 5V
 268. 10k
 269. 100nF
 270. 5V
 271. 10k
 272. 100nF
 273. 5V
 274. 10k
 275. 100nF
 276. 5V
 277. 10k
 278. 100nF
 279. 5V
 280. 10k
 281. 100nF
 282. 5V
 283. 10k
 284. 100nF
 285. 5V
 286. 10k
 287. 100nF
 288. 5V
 289. 10k
 290. 100nF
 291. 5V
 292. 10k
 293. 100nF
 294. 5V
 295. 10k
 296. 100nF
 297. 5V
 298. 10k
 299. 100nF
 300. 5V
 301. 10k
 302. 100nF
 303. 5V
 304. 10k
 305. 100nF
 306. 5V
 307. 10k
 308. 100nF
 309. 5V
 310. 10k
 311. 100nF
 312. 5V
 313. 10k
 314. 100nF
 315. 5V
 316. 10k
 317. 100nF
 318. 5V
 319. 10k
 320. 100nF
 321. 5V
 322. 10k
 323. 100nF
 324. 5V
 325. 10k
 326. 100nF
 327. 5V
 328. 10k
 329. 100nF
 330. 5V
 331. 10k
 332. 100nF
 333. 5V
 334. 10k
 335. 100nF
 336. 5V
 337. 10k
 338. 100nF
 339. 5V
 340. 10k
 341. 100nF
 342. 5V
 343. 10k
 344. 100nF
 345. 5V
 346. 10k
 347. 100nF
 348. 5V
 349. 10k
 350. 100nF
 351. 5V
 352. 10k
 353. 100nF
 354. 5V
 355. 10k
 356. 100nF
 357. 5V
 358. 10k
 359. 100nF
 360. 5V
 361. 10k
 362. 100nF
 363. 5V
 364. 10k
 365. 100nF
 366. 5V
 367. 10k
 368. 100nF
 369. 5V
 370. 10k
 371. 100nF
 372. 5V
 373. 10k
 374. 100nF
 375. 5V
 376. 10k
 377. 100nF
 378. 5V
 379. 10k
 380. 100nF
 381. 5V
 382. 10k
 383. 100nF
 384. 5V
 385. 10k
 386. 100nF
 387. 5V
 388. 10k
 389. 100nF
 390. 5V
 391. 10k
 392. 100nF
 393. 5V
 394. 10k
 395. 100nF
 396. 5V
 397. 10k
 398. 100nF
 399. 5V
 400. 10k
 401. 100nF
 402. 5V
 403. 10k
 404. 100nF
 405. 5V
 406. 10k
 407. 100nF
 408. 5V
 409. 10k
 410. 100nF
 411. 5V
 412. 10k
 413. 100nF
 414. 5V
 415. 10k
 416. 100nF
 417. 5V
 418. 10k
 419. 100nF
 420. 5V
 421. 10k
 422. 100nF
 423. 5V
 424. 10k
 425. 100nF
 426. 5V
 427. 10k
 428. 100nF
 429. 5V
 430. 10k
 431. 100nF
 432. 5V
 433. 10k
 434. 100nF
 435. 5V
 436. 10k
 437. 100nF
 438. 5V
 439. 10k
 440. 100nF
 441. 5V
 442. 10k
 443. 100nF
 444. 5V
 445. 10k
 446. 100nF
 447. 5V
 448. 10k
 449. 100nF
 450. 5V
 451. 10k
 452. 100nF
 453. 5V
 454. 10k
 455. 100nF
 456. 5V
 457. 10k
 458. 100nF
 459. 5V
 460. 10k
 461. 100nF
 462. 5V
 463. 10k
 464. 100nF
 465. 5V
 466. 10k
 467. 100nF
 468. 5V
 469. 10k
 470. 100nF
 471. 5V
 472. 10k
 473. 100nF
 474. 5V
 475. 10k
 476. 100nF
 477. 5V
 478. 10k
 479. 100nF
 480. 5V
 481. 10k
 482. 100nF
 483. 5V
 484. 10k
 485. 100nF
 486. 5V
 487. 10k
 488. 100nF
 489. 5V
 490. 10k
 491. 100nF
 492. 5V
 493. 10k
 494. 100nF
 495. 5V
 496. 10k
 497. 100nF
 498. 5V
 499. 10k
 500. 100nF
 501. 5V
 502. 10k
 503. 100nF
 504. 5V
 505. 10k
 506. 100nF
 507. 5V
 508. 10k
 509. 100nF
 510. 5V
 511. 10k
 512. 100nF
 513. 5V
 514. 10k
 515. 100nF
 516. 5V
 517. 10k
 518. 100nF
 519. 5V
 520. 10k
 521. 100nF
 522. 5V
 523. 10k
 524. 100nF
 525. 5V
 526. 10k
 527. 100nF
 528. 5V
 529. 10k
 530. 100nF
 531. 5V
 532. 10k
 533. 100nF
 534. 5V
 535. 10k
 536. 100nF
 537. 5V
 538. 10k
 539. 100nF
 540. 5V
 541. 10k
 542. 100nF
 543. 5V
 544. 10k
 545. 100nF
 546. 5V
 547. 10k
 548. 100nF
 549. 5V
 550. 10k
 551. 100nF
 552. 5V
 553. 10k
 554. 100nF
 555. 5V
 556. 10k
 557. 100nF
 558. 5V
 559. 10k
 560. 100nF
 561. 5V
 562. 10k
 563. 100nF
 564. 5V
 565. 10k
 566. 100nF
 567. 5V
 568. 10k
 569. 100nF
 570. 5V
 571. 10k
 572. 100nF
 573. 5V
 574. 10k
 575. 100nF
 576. 5V
 577. 10k
 578. 100nF
 579. 5V
 580. 10k
 581. 100nF
 582. 5V
 583. 10k
 584. 100nF
 585. 5V
 586. 10k
 587. 100nF
 588. 5V
 589. 10k
 590. 100nF
 591. 5V
 592. 10k
 593. 100nF
 594. 5V
 595. 10k
 596. 100nF
 597. 5V
 598. 10k
 599. 100nF
 600. 5V
 601. 10k
 602. 100nF
 603. 5V
 604. 10k
 605. 100nF
 606. 5V
 607. 10k
 608. 100nF
 609. 5V
 610. 10k
 611. 100nF
 612. 5V
 613. 10k
 614. 100nF
 615. 5V
 616. 10k
 617. 100nF
 618. 5V
 619. 10k
 620. 100nF
 621. 5V
 622. 10k
 623. 100nF
 624. 5V
 625. 10k
 626. 100nF
 627. 5V
 628. 10k
 629. 100nF
 630. 5V
 631. 10k
 632. 100nF
 633. 5V
 634. 10k
 635. 100nF
 636. 5V
 637. 10k
 638. 100nF
 639. 5V
 640. 10k
 641. 100nF
 642. 5V
 643. 10k
 644. 100nF
 645. 5V
 646. 10k
 647. 100nF
 648. 5V
 649. 10k
 650. 100nF
 651. 5V
 652. 10k
 653. 100nF
 654. 5V
 655. 10k
 656. 100nF
 657. 5V
 658. 10k
 659. 100nF
 660. 5V
 661. 10k
 662. 100nF
 663. 5V
 664. 10k
 665. 100nF
 666. 5V
 667. 10k
 668. 100nF
 669. 5V
 670. 10k
 671. 100nF
 672. 5V
 673. 10k
 674. 100nF
 675. 5V
 676. 10k
 677. 100nF
 678. 5V
 679. 10k
 680. 100nF
 681. 5V
 682. 10k
 683. 100nF
 684. 5V
 685. 10k
 686. 100nF
 687. 5V
 688. 10k
 689. 100nF
 690. 5V
 691. 10k
 692. 100nF
 693. 5V
 694. 10k
 695. 100nF
 696. 5V
 697. 10k
 698. 100nF
 699. 5V
 700. 10k
 701. 100nF
 702. 5V
 703. 10k
 704. 100nF
 705. 5V
 706. 10k
 707. 100nF
 708. 5V
 709. 10k
 710. 100nF
 711. 5V
 712. 10k
 713. 100nF
 714. 5V
 715. 10k
 716. 100nF
 717. 5V
 718. 10k
 719. 100nF
 720. 5V
 721. 10k
 722. 100nF
 723. 5V
 724. 10k
 725. 100nF
 726. 5V
 727. 10k
 728. 100nF
 729. 5V
 730. 10k
 731. 100nF
 732. 5V
 733. 10k
 734. 100nF
 735. 5V
 736. 10k
 737. 100nF
 738. 5V
 739. 10k
 740. 100nF
 741. 5V
 742. 10k
 743. 100nF
 744. 5V
 745. 10k
 746. 100nF
 747. 5V
 748. 10k
 749. 100nF
 750. 5V
 751. 10k
 752. 100nF
 753. 5V
 754. 10k
 755. 100nF
 756. 5V
 757. 10k
 758. 100nF
 759. 5V
 760. 10k
 761. 100nF
 762. 5V
 763. 10k
 764. 100nF
 765. 5V
 766. 10k
 767. 100nF
 768. 5V
 769. 10k
 770. 100nF
 771. 5V
 772. 10k
 773. 100nF
 774. 5V
 775. 10k
 776. 100nF
 777. 5V
 778. 10k
 779. 100nF
 780. 5V
 781. 10k
 782. 100nF
 783. 5V
 784. 10k
 785. 100nF
 786. 5V
 787. 10k
 788. 100nF
 789. 5V
 790. 10k
 791. 100nF
 792. 5V
 793. 10k
 794. 100nF
 795. 5V
 796. 10k
 797. 100nF
 798. 5V
 799. 10k
 800. 100nF
 801. 5V
 802. 10k
 803. 100nF
 804. 5V
 805. 10k
 806. 100nF
 807. 5V
 808. 10k
 809. 100nF
 810. 5V
 811. 10k
 812. 100nF
 813. 5V
 814. 10k
 815. 100nF
 816. 5V
 817. 10k
 818. 100nF
 819. 5V
 820. 10k
 821. 100nF
 822. 5V
 823. 10k
 824. 100nF
 825. 5V
 826. 10k
 827. 100nF
 828. 5V
 829. 10k
 830. 100nF
 831. 5V
 832. 10k
 833. 100nF
 834. 5V
 835. 10k
 836. 100nF
 837. 5V
 838. 10k
 839. 100nF
 840. 5V
 841. 10k
 842. 100nF
 843. 5V
 844. 10k
 845. 100nF
 846. 5V
 847. 10k
 848. 100nF
 849. 5V
 850. 10k
 851. 100nF
 852. 5V
 853. 10k
 854. 100nF
 855. 5V
 856. 10k
 857. 100nF
 858. 5V
 859. 10k
 860. 100nF
 861. 5V
 862. 10k
 863. 100nF
 864. 5V
 865. 10k
 866. 100nF
 867. 5V
 868. 10k
 869. 100nF
 870. 5V
 871. 10k
 872. 100nF
 873. 5V
 874. 10k
 875. 100nF
 876. 5V
 877. 10k
 878. 100nF
 879. 5V
 880. 10k
 881. 100nF
 882. 5V
 883. 10k
 884. 100nF
 885. 5V
 886. 10k
 887. 100nF
 888. 5V
 889. 10k
 890. 100nF
 891. 5V
 892. 10k
 893. 100nF
 894. 5V
 895. 10k
 896. 100nF



106



107

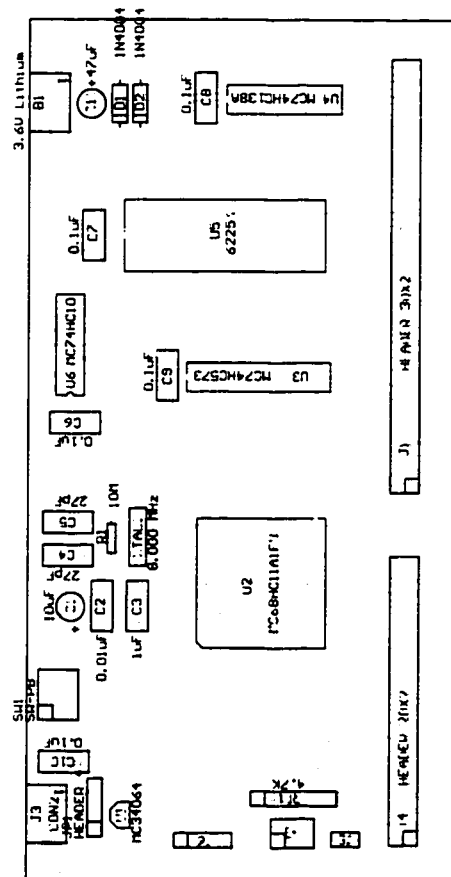


Figure A.8: Microcontroller board part placement diagram.

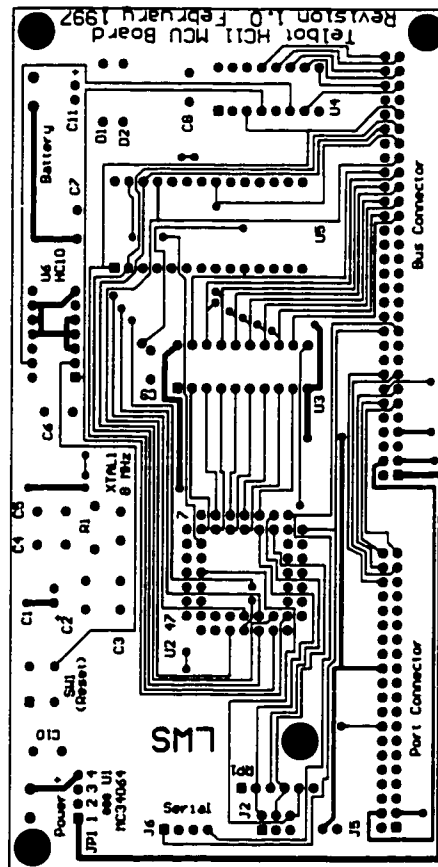


Figure A.9: Microcontroller board top-layer copper foil pattern. Not to scale.

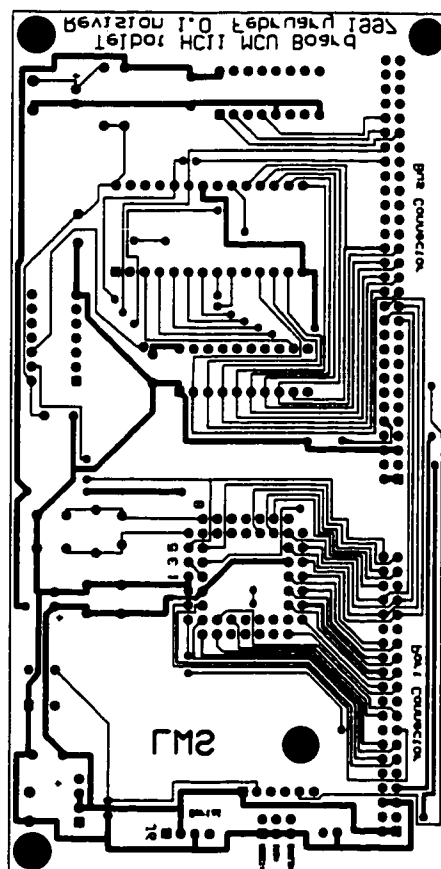


Figure A.10: Microcontroller bottom-layer copper foil pattern (through-board view).
Not to scale.

Appendix B

Serial-Level Conversion PCB

B.1 Introduction

An outline of the operation of the Serial-level conversion PCB. is given here. The purpose of this board is simple: to convert RS-232C signal voltage levels to TTL levels (usable by the HC11 Microcontroller boards) and vice-versa.

A block diagram of the board electronics is shown in Fig. 5.2.

The functionality of this board lies in the use of a MAX232 IC. In fact, the only additional components on this PCB are the connectors used to connect to the RS-232C (PC) serial line, and to the TTL-level serial line (to the HC11 board). For more information, refer to a MAX232 data sheet.

Figs. B.1, contained later in this chapter, shows the schematic of the serial-level conversion board. Fig. B.2 is a diagram showing the part placement. Figs. B.3 and B.4 show the top and bottom copper foil patterns, respectively.

B.2 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the serial conversion board.

B.2.1 PC RS-232 Connector: J1

J1 – PC RS-232 Connector			
Pin	Mnemonic	Direction	Description
1	T1_OUT	Out	RS-232 Output (to PC)
2	Vcc	-	+5V D.C. power.
3	GND	-	Ground (return).
(continued)			

J1 – PC RS-232 Connector (continued)			
Pin	Mnemonic	Direction	Description
4	RI_IN	In	RS-232 Input (from PC)

Table B.1: Serial-level Conversion PCB connector J1 pinouts.

B.2.2 HC11 TTL Serial Connector: J2

J2 – HC11 TTL Serial Connector			
Pin	Mnemonic	Direction	Description
1	UC_PD1	In	UC Tx Data (from HC11)
2	Vcc	-	+5V D.C. power.
3	GND	-	Ground (return).
4	UC_PD0	Out	UC Rx Data (to HC11)

Table B.2: Serial-level Conversion PCB connector J1 pinouts.

B.3 Future Improvements

Future improvements would include

- Changing the RS-232C connector to the PC to a standard DB9 which would simplify cabling; and
- Including this circuitry directly on the HC11 MCU boards. This change would require redesign of the serial network multiplexing included on the PC-to-MCU shared memory PCB to accommodate RS-232 levels.

B.4 Schematic Diagrams and Board Layout

Figures B.1 through B.4 comprise the schematic and board layout of the Serial-Level Conversion board.

B.4.1 Part List

Serial-Level Conversion Board Parts			
	Number Used	Part Type	Designators
1	2	4-pin polarized Molex header	J1 J2
2	1	MAX233	U1

Table B.3: Serial-Level Conversion board parts list.

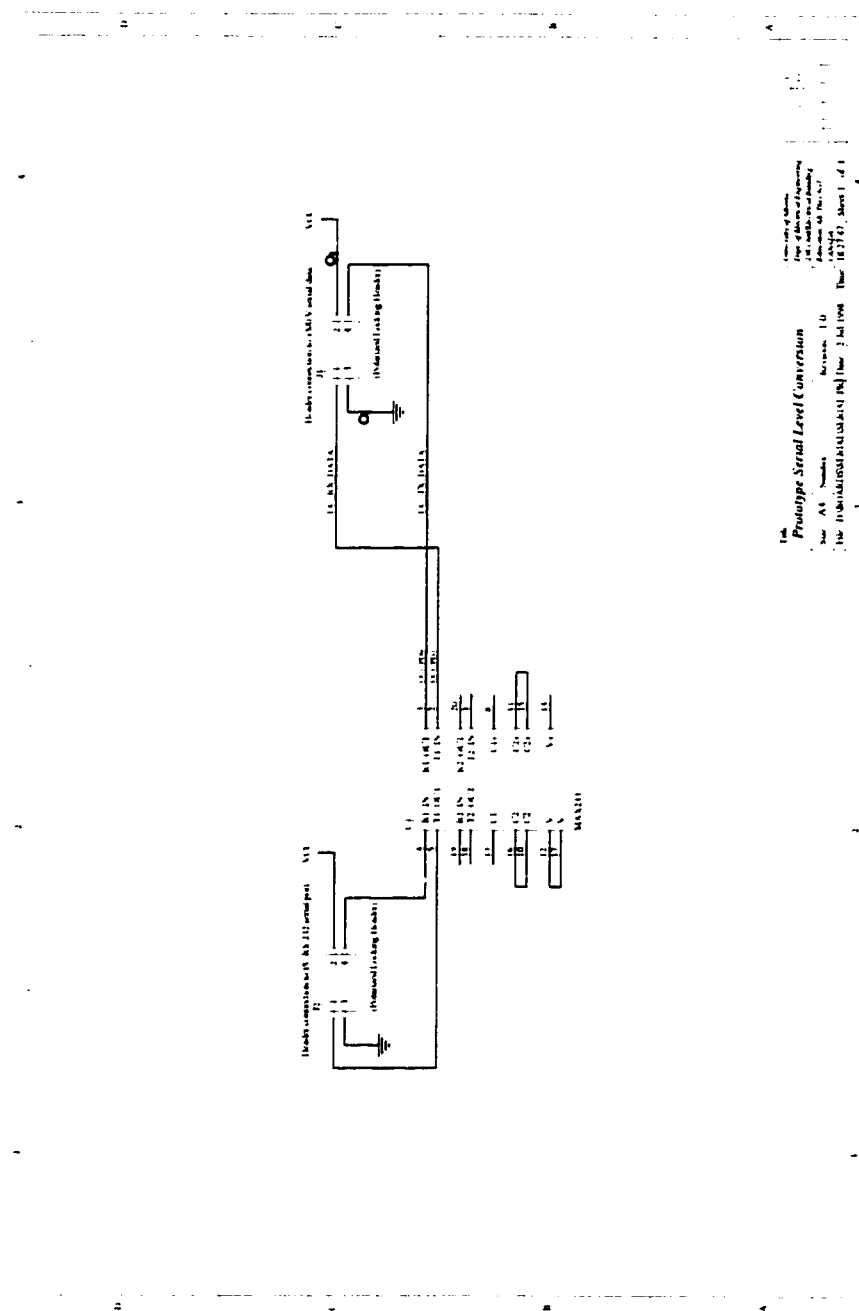


Figure B.1: Serial-Level Conversion schematic (1 of 1).

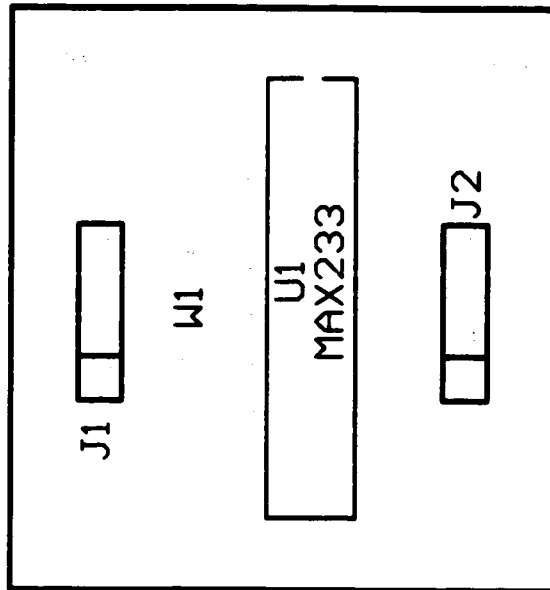


Figure B.2: Serial-Level Conversion part placement diagram.

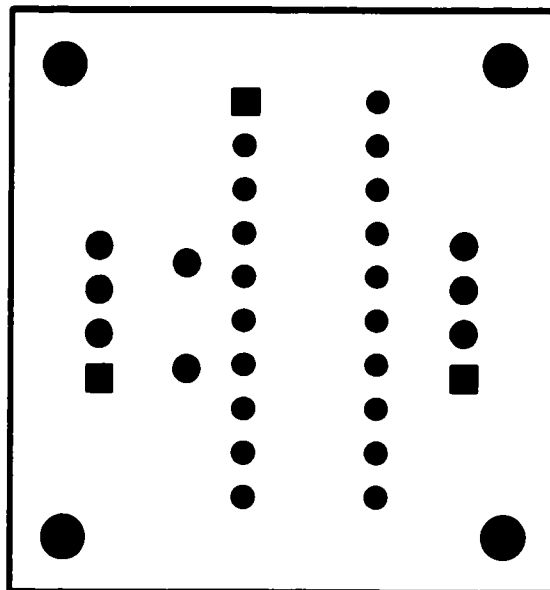


Figure B.3: Serial-Level Conversion top-layer copper foil pattern. Not to scale.

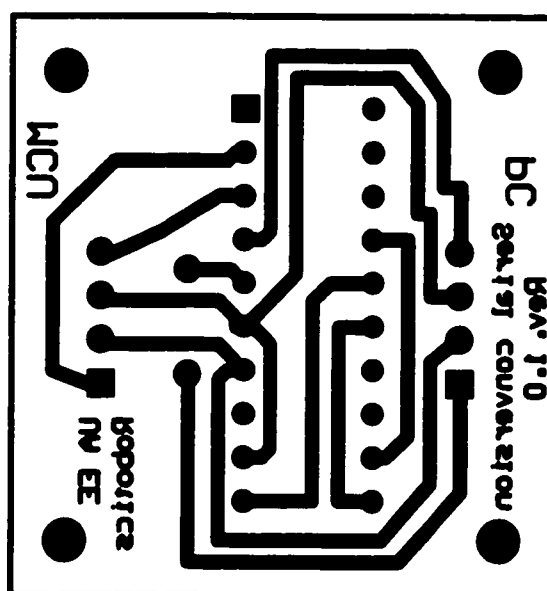


Figure B.4: Serial-Level Conversion bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix C

Parallel Port Expansion PCB

C.1 Introduction

An outline of the operation, features, and user-settable options of the Parallel Port Expansion PCB (referred to as the ‘expansion board’) is given here. This board is connected between the PC and the PC-to-MCU Shared Memory PCB. It provides digital input/output with the PC in order to achieve the interface to this memory. In addition, several digital outputs, indirectly routed to the MCUs, are used for controlling their operating modes.

A block diagram of the board electronics is shown in Fig. 5.3. This PCB expands the capability of a parallel printer port to give 64 digital outputs (arranged in 8 banks of 8 bits each) and an 8-bit digital input port. Each bank of digital outputs has its own output-enable control, a feature which has proven effective in suppressing undesired transient power-up characteristics.

Since the board is designed to be interfaced to banks of 8-bit SRAMs, one of the output banks is designated as a ‘data bus’ and it must be controlled such that its outputs are not enabled when any SRAM is placing data onto the bus. The user is given a choice of two different operating modes to achieve this operation: either an automatic bus control-line implementation, or through explicit programming of the output enable bit. This jumper-selectable choice was provided to make the board suitable for a wider range of applications. However, for the purposes of this application, the control is performed through the ‘Software’ option.

The reasons that the parallel port was chosen as the prime means of communication with the microcontrollers is two-fold: cross-platform bus-architecture independence and simplicity. A good deal of system testing was performed on a notebook computer which has a vendor-specific internal bus architecture. Interface with the system bus would have proven inefficient and costly.

One disadvantage of using the parallel port is the communication speed with the shared memory, primarily due to the roundabout manner in which a specific

memory location needs to be accessed. However, this is a small disadvantage as the required bandwidth of the telerobotic application is quite small. Future improvements such as the addition of video cameras to the test-bed would be done through the PC system bus, and therefore no greater bandwidth requirement would be placed upon this component.

Figs. C.1 through C.4, contained later in this chapter, show the schematic of the expansion board. Fig. C.5 is a diagram showing the parts placement. Figs. C.6 and C.7 show the top and bottom copper foil patterns, respectively.

C.2 Operation

The expansion board operates by providing a multiplexing scheme operating upon the standard ability of a parallel printer port. The usual address range of the printer port on IBM compatible computers is 0x378-0x37A (hexadecimal) (1pt1:).

Address	Bits	Description
0x378	0-7	Expansion board output data to be latched.
0x379	0-3	Register select lines (used to latch data).
0x37A	4-7 ¹	Input data nibble (4 bits).

Table C.1: Parallel port address definitions.

For digital output, the basic operation is to write 8-bits of data to location 0x378, and then to 'strobe' the destination register's clock line via the 4 least significant bits of address 0x379. For digital input, one nibble (4 bits) of the input 8-bit data bus must be read at a time into the high nibble of location 0x37A. Which nibble is read in is determined by the register select lines: if register 9 is currently selected, the low byte is read², otherwise the high byte is read.

C.2.1 Register Descriptions

The following table outlines the purpose of each register as selected by the value written to 0x37A.

¹ Bit 4 (the most significant bit of the nibble) needs to be inverted by software after reading.

² For Revision 1.0 of the board (used in the test-bed hardware), bits 2 and 3 of the low nibble need to be swapped due to a design bug.

Register Descriptions – (0x37A)			
Number	Mnemonic	IC	Function
0	OEx	U6	Register output enable lines (mandatory).
1	MEM_SDx	U5	External data bus output lines.
2	MEM_SAx	U4	Address bus (bits 0–7).
3	MEM_SAx	U3	Address bus (bits 8–15).
4	CTRL	U2	Chip select and control lines.
5	UC1	U7	Microcontroller 1 control register.
6	UC2	U8	Microcontroller 2 control register.
7	UC3	U9	Microcontroller 3 control register.
8	PWR	U10	Power routing control register.
9	Nibble	U1	Input data bus nibble select (mandatory).
10–15	-	-	Unimplemented.

Table C.2: Parallel port expansion PCB register definitions.

Register 0 (OEx) is utilized to enable or disable the 8 output registers. Bit 0 of this register corresponds to register 1, through to bit 7, corresponding to register 8. When a '0' is written to the corresponding bit, the register will be enabled. If a '1' is the output value, the register will be in a high-impedance state.

The output of register 1 (MEM_SDx) is connected directly to the peripheral data bus. It should drive the data bus only when there will be no contention for it. As addressed in the next section, the method by which this register is enabled can be selected by the user.

Registers 2 and 3 (MEM_SAx) are two general-purpose digital outputs that may be used as address lines for an 8-bit peripheral (such as externally implemented memory).

Register 4 is a digital output register that is used to implement peripheral (memory) select and control lines. The suggested use of the individual bits is as shown in the following table.

Register 4 Bit Definitions		
Bit Number	Mnemonic	Function
0	IOR*	Peripheral read (active low).
1	IOW*	Peripheral write (active low).
2	CS_UC1_MEM*	UC1 shared memory chip select.
3	CS_UC2_MEM*	UC2 shared memory chip select.
4	CS_UC3_MEM*	UC3 shared memory chip select.
5	SPARE4	Unused digital output.
6	SPARE5	Unused digital output.
7	SPARE6	Unused digital output.

Table C.3: Parallel port expansion PCB CTRL register (register 4) bit definitions.

The user has the option of connecting bit 1 directly to the output enable line of the data bus output IC (register 1).

Registers 5, 6, and 7 are used as control registers for various aspects of the three system microcontrollers, UC1, UC2, and UC3, respectively. The following table outlines the purpose of each bit.

Register 5–7 Bit Definitions		
Bit Number	Mnemonic	Function
0	UCx_RESET	MCU reset.
1	UCx_DWNLD	MCU download mode.
2	UCx_IRQ	MCU interrupt request.
3	UCx_XIRQ	MCU non-maskable interrupt request.
4	UCx_PCLS4	Unused digital output.
5	UCx_PCLS5	Unused digital output.
6	UCx_SER_RX*	MCU serial receive.
7	UCx_SER_TX*	MCU serial transmit.

Table C.4: Parallel port expansion PCB UCx register (registers 5–7) bit definitions.

Due to subsequent level-conversion on the PC-to-MCU shared-memory board, the signals issued by the PC are inverted from those expected by the microcontrollers. The last two bits (UCx_SER_RX* and UCx_SER_TX*) are active-low signals which determine if MCUx is connected to the serial network. To avoid collision of serial data being transmitted by the MCUs, only one MCU at a time should have UCx_SER_TX

at a low level.

Register 8 is reserved for interface to power-routing circuitry. Currently, the Power Conditioning and Routing PCB is not implemented. Future modifications include further PC-directed control of power routing and regulation.

‘Register 9’ is not really a register. The select line is used to select which nibble of the data bus is read at address 0x379 as previously mentioned.

C.3 User-settable Options

The user has the ability to configure the expansion board in two capacities:

- (1) to select manual or automatic enabling of register 1 which drives the external data bus; and
- (2) to enable (or disable) the input data bus.

C.3.1 Data Bus/Output Port Select: JP1

Via positioning of the jumper on JP1, the user can select whether the register connected to the external data bus (register 0) is enabled automatically or manually.

‘Automatic mode’ is entered by placing the jumper across pins 1 and 2. This selection connects bit 1 of register 4 (IOW*) directly to the output enable line of register 1. This function has not been tested, and future modifications should take steps to ensure that bus contention will not result.

‘Manual mode’ implies that the user will manually toggle the output enable line of register 1 via register 0 (the OEx lines). The mode is selected by placing the jumper across pins 2 and 3. This mode is the recommended configuration as it allows the user to ensure through software that no contention problems will be encountered. The test-bed software assumes that the ‘Manual mode’ is used.

C.3.2 Data Bus Input Enable: JP2

This jumper-selectable option allows the user to enable (jumper from pin 1 to 2) or disable (jumper from pin 2 to 3) the ability to read digital data in from the expansion board.

This feature is offered so that the expansion board can remain in the system when the printer is used. Although this is not likely in the test-bed, it is a possibility should the board be used outside the context of this project. There are two DB25 connectors provided, one which can be used as a loop for connection to the printer. However, nothing other than the PC and printer should be connected to the expansion PCB when the printer is used (and the data bus input is disabled). It is probable that the ‘disable’ feature will rarely be used.

C.4 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the expansion board.

C.4.1 Power Connector: J1

J1 – Power Connector			
Pin	Mnemonic	Direction	Description
1	Vcc	-	+5V D.C. power.
2	GND	-	Ground (return).

Table C.5: Parallel port expansion PCB connector J1 pin-outs.

C.4.2 Digital I/O Connector: J2

J2 – Digital I/O Connector			
Pin	Mnemonic	Direction	Description
1	MEM_SD0	I/O	External data bus bit 0.
2	MEM_SD1	I/O	External data bus bit 1.
3	MEM_SD2	I/O	External data bus bit 2.
4	MEM_SD3	I/O	External data bus bit 3.
5	MEM_SD4	I/O	External data bus bit 4.
6	MEM_SD5	I/O	External data bus bit 5.
7	MEM_SD6	I/O	External data bus bit 6.
8	MEM_SD7	I/O	External data bus bit 7.
9	Vcc	-	+5V D.C. power.
10	Vcc	-	+5V D.C. power.
11	MEM_SA0	O	External address bus bit 0.
12	MEM_SA1	O	External address bus bit 1.
13	MEM_SA2	O	External address bus bit 2.
14	MEM_SA3	O	External address bus bit 3.
15	MEM_SA4	O	External address bus bit 4.
16	MEM_SA5	O	External address bus bit 5.
17	MEM_SA6	O	External address bus bit 6.
18	MEM_SA7	O	External address bus bit 7.
19	MEM_SA8	O	External address bus bit 8.
(continued)			

J2 – Digital I/O Connector (continued)			
Pin	Mnemonic	Direction	Description
20	MEM_SA9	O	External address bus bit 9.
21	MEM_SA10	O	External address bus bit 10.
22	MEM_SA11	O	External address bus bit 11.
23	MEM_SA12	O	External address bus bit 12.
24	MEM_SA13	O	External address bus bit 13.
25	MEM_SA14	O	External address bus bit 14.
26	MEM_SA15	O	External address bus bit 15.
27	IOR*	O	Peripheral read control line (active low).
28	IOW*	O	Peripheral write control line (active low).
29	CS_UC1_MEM*	O	UC1 shared memory chip select.
30	CS_UC1_MEM*	O	UC2 shared memory chip select.
31	CS_UC1_MEM*	O	UC4 shared memory chip select.
32	SPARE4	O	Unused digital output.
33	SPARE5	O	Unused digital output.
34	SPARE6	O	Unused digital output.
35	GND	-	Ground (return).
36	GND	-	Ground (return).
37	UC1_RESET	O	MCU 1 reset.
38	UC1_DWNLD	O	MCU 1 download mode.
39	UC1_IRQ	O	MCU 1 interrupt request.
40	UC1_XIRQ	O	MCU 1 non-maskable IRQ.
41	UC1_PCLS4	O	Unused digital output.
42	UC1_PCLS5	O	Unused digital output.
43	UC1_SER_RX*	O	MCU 1 serial receive.
44	UC1_SER_TX*	O	MCU 1 serial transmit.
45	UC2_RESET	O	MCU 2 reset.
46	UC2_DWNLD	O	MCU 2 download mode.
47	UC2_IRQ	O	MCU 2 interrupt request.
48	UC2_XIRQ	O	MCU 2 non-maskable IRQ.
49	UC2_PCLS4	O	Unused digital output.
50	UC2_PCLS5	O	Unused digital output.
51	UC2_SER_RX*	O	MCU 2 serial receive.
52	UC2_SER_TX*	O	MCU 2 serial transmit.
53	UC3_RESET	O	MCU 3 reset.
54	UC3_DWNLD	O	MCU 3 download mode.
55	UC3_IRQ	O	MCU 3 interrupt request.
56	UC3_XIRQ	O	MCU 3 non-maskable IRQ.
57	UC3_PCLS4	O	Unused digital output.
(continued)			

J2 – Digital I/O Connector (continued)			
Pin	Mnemonic	Direction	Description
58	UC3_PCLS5	O	Unused digital output.
59	UC3_SER_RX*	O	MCU 3 serial receive.
60	UC3_SER_TX*	O	MCU 3 serial transmit.

Table C.6: Parallel port expansion PCB connector J2 pin-outs.

C.4.3 Digital O/P Connector: J3

J3 – Digital O/P Connector			
Pin	Mnemonic	Direction	Description
1	Vcc	-	+5V D.C. power.
2	GND	-	Ground (return).
3	PWR0	O	Navigation actuator power control.
4	PWR1	O	Power control register (reserved).
5	PWR2	O	Power control register (reserved).
6	PWR3	O	Power control register (reserved).
7	PWR4	O	Power control register (reserved).
8	PWR5	O	Power control register (reserved).
9	PWR6	O	Power control register (reserved).
10	PWR7	O	Power control register (reserved).

Table C.7: Parallel port expansion PCB connector J3 pin-outs.

C.4.4 Parallel Port Connector: J4 and J5

The directions presented here are with respect to the expansion board. For more information about the parallel port interface, please refer to an IBM technical reference.

J4 and J5 – Parallel Port Connector			
Pin	Mnemonic	Direction	Description
1	SEL0*	I	Select line 0 (active low).
2	OUT0	I	Expansion board register data bus bit 0.
3	OUT1	I	Expansion board register data bus bit 1.
(continued)			

J4 and J5 – Parallel Port Connector (continued)			
Pin	Mnemonic	Direction	Description
4	OUT2	I	Expansion board register data bus bit 2.
5	OUT3	I	Expansion board register data bus bit 3.
6	OUT4	I	Expansion board register data bus bit 4.
7	OUT5	I	Expansion board register data bus bit 5.
8	OUT6	I	Expansion board register data bus bit 6.
9	OUT7	I	Expansion board register data bus bit 7.
10	IN2	O	Data bus nibble bit 2 (PC read).
11	IN3	O	Data bus nibble bit 3 (PC read).
12	IN1	O	Data bus nibble bit 1 (PC read).
13	IN0	O	Data bus nibble bit 0 (PC read).
14	SEL1*	I	Select line 1 (active low).
15	N/C	-	No connection.
16	SEL2	I	Select line 2.
17	SEL3*	I	Select line 3 (active low).
18	GND	-	Ground (return).
19	GND	-	Ground (return).
20	GND	-	Ground (return).
21	GND	-	Ground (return).
22	GND	-	Ground (return).
23	GND	-	Ground (return).
24	GND	-	Ground (return).
25	GND	-	Ground (return).

Table C.8: Parallel port expansion PCB connector J4 and J5 pinouts.

The SELx lines are inverted where appropriate in order to accommodate the inversion that occurs internally to the computer to which the board is connected. Thus, although some signals are active-low at the connector, the programmer still initiates active-high signals through software. In other words, to select register 3, a ‘3’ is still written to memory location 0x37A.

C.5 Design Limitations

There are two shortcomings of the parallel port expansion PCB. The first is the fact that the ‘automatic’ output enable feature of the data bus output register (register 5) has not been fully tested. It is suspected that timing requirements required for writing to 8-bit peripherals (such as a memory bank) may not be met. This potential shortcoming which could arise in bus contention or failure of data write sequences

can be overcome by utilizing the 'manual' mode of operation in which the software provides the timing of all control signals. This is done in the test-bed software.

The second shortcoming is with respect to power-up states of the register outputs. Typically, the output of an 'LS374 is low upon application of power. This will result in the majority of output signals being enabled (not high-impedance) and low. This is of particular interest with regard to signals which are active low, such as any chip select circuitry that the outputs may drive. Thus, the suggested role of some outputs may need to be inspected. With this knowledge, any circuitry connected to the expansion board can be designed in order to accommodate the power-up state. To accommodate this shortcoming, a switch has been added to the OE* line of U6 – the OE register. It can be toggled between +5V and ground which gives control over all of the digital outputs. When switched to +5V, all registers are placed in a high-impedance state.

C.6 Future Improvements

Potential future improvements include:

- hardware inversion of the single bit of the input nibble without interfering with the possible connection (loop) to a printer (when set to this mode); and
- inclusion of power-up reset circuitry to establish appropriate signal levels (such as having all digital outputs in a high-impedance state). This should likely include user-selectable settings to accommodate a wide variety of connected equipment.
- swapping of the MEM_SD2 and MEM_SD3 bits on U1. This would rid the need for swapping the bits in software.
- the 'LS42, used to decode the register select lines, creates data 'spikes' during its decoding process. To compensate for this, a timing patch was prototyped but not included on the PCB. The schematic for this patch is included with the board's schematics.

C.7 Schematic Diagrams and Board Layout

Figures C.1 through C.7 comprise the schematic and board layout of the parallel port expansion board.

C.7.1 Part List

Parallel Port Expansion Board Parts			
	Number Used	Part Type	Designators
1	12	0.1uF	C1 - C12
2	1	47uF	C13
3	2	DB25	J4 J5
4	1	DM74LS04	U12
5	1	DM74LS42	U17
6	1	DM74LS157	U1
7	11	DM74LS374	U2 - U10, U18, U19
8	2	HEADER 3	JP1 JP2
9	1	HEADER 5X2	J3
10	1	HEADER 30X2	J2
11	1	Molex 2-Pin Header	J1
12	1	12MHz TTL Oscillator	OSC1
13	1	SN7495A	U13
14	1	SN74LS00	U15
15	1	SN74LS85	U14
16	1	DIP16 Socket	U11

Table C.9: Parallel port expansion board parts list.

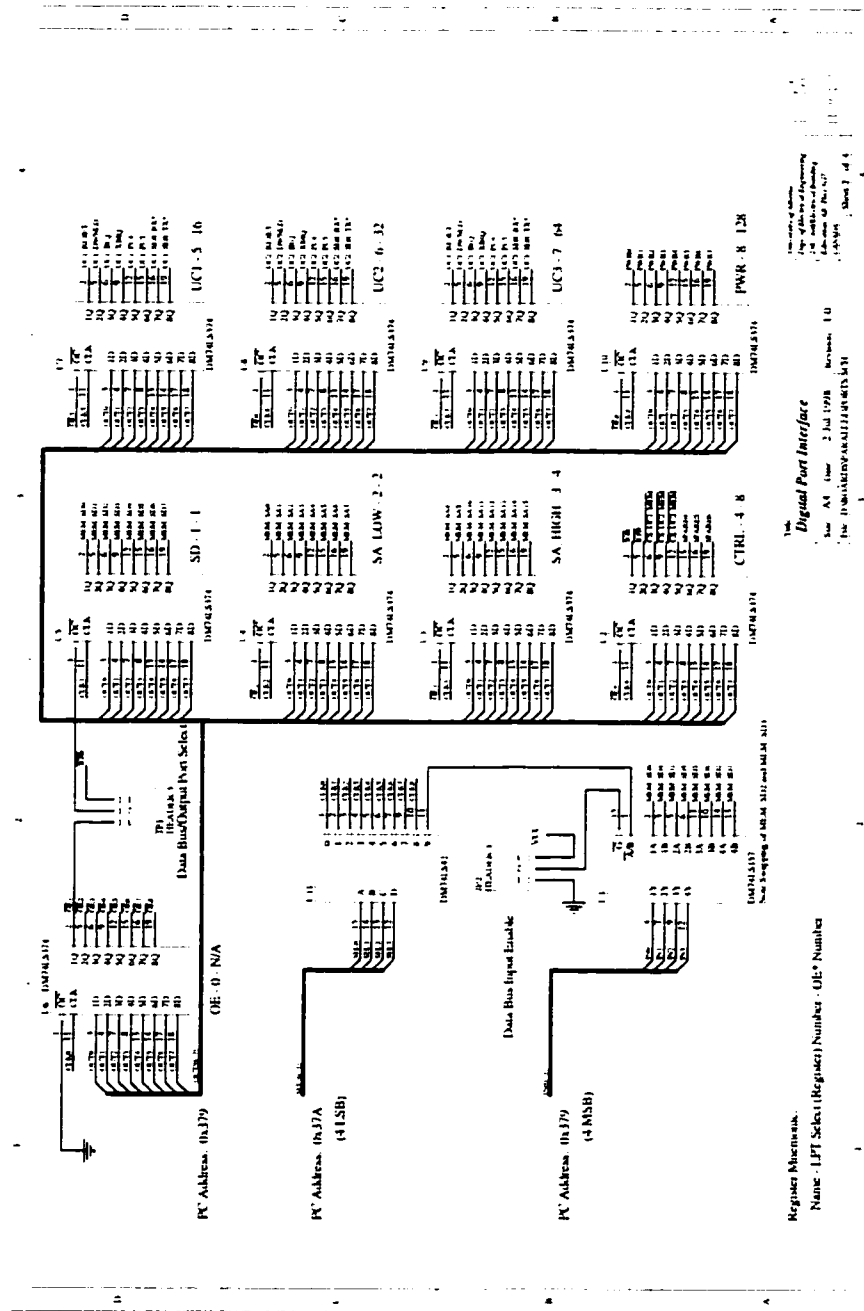


Figure C.2: Parallel port interface card schematic (2 of 4).

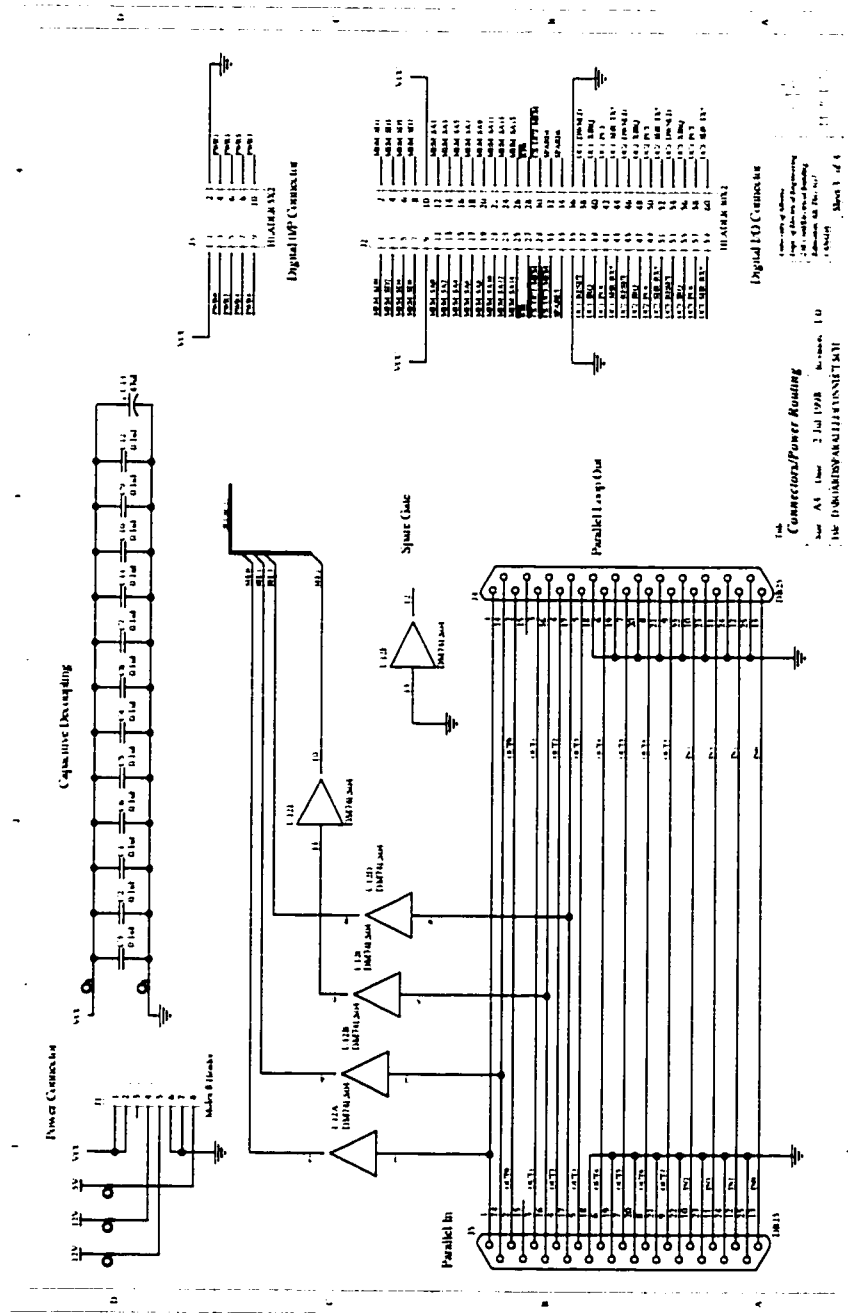


Figure C.3: Parallel port interface card schematic (3 of 4).

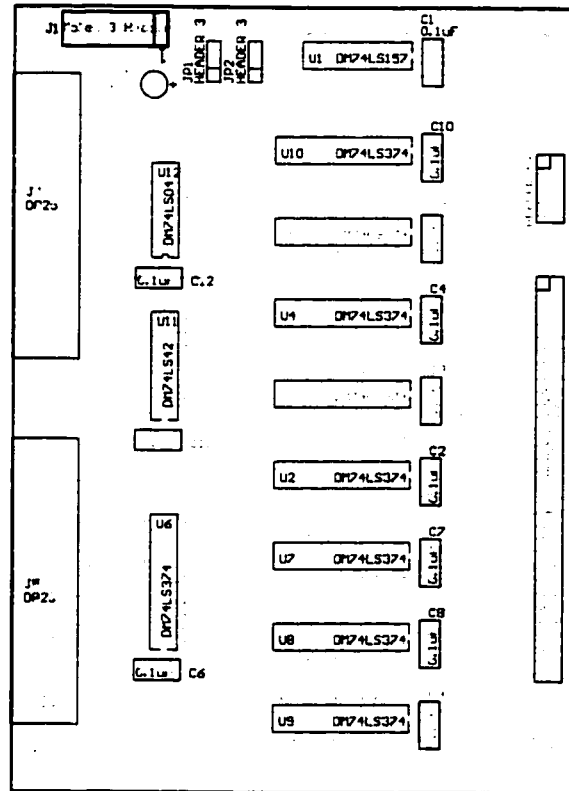


Figure C.5: Parallel port interface card part placement diagram.

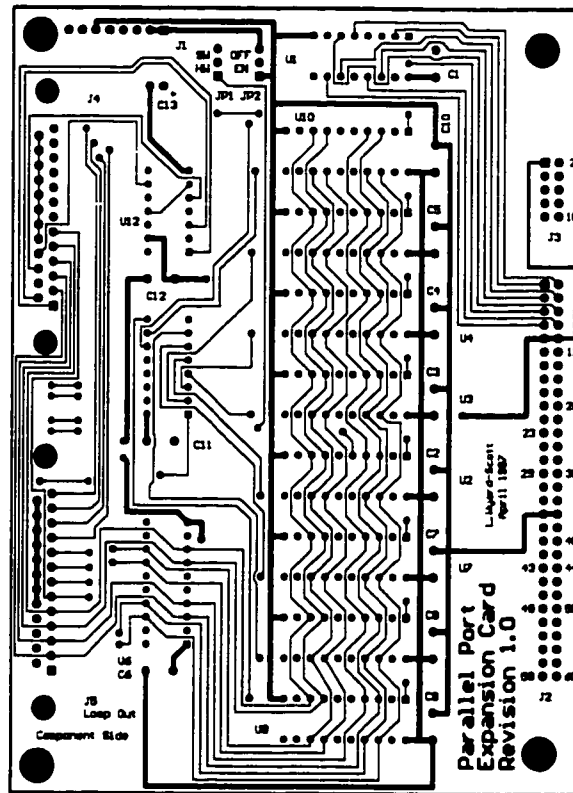


Figure C.6: Parallel port interface card top-layer copper foil pattern. Not to scale.

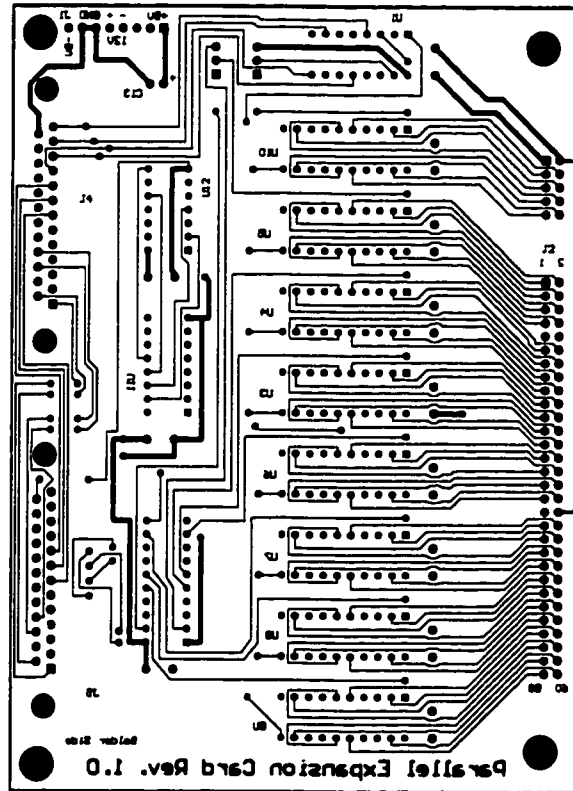


Figure C.7: Parallel port interface card bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix D

PC-to-MCU Shared Memory/Serial Network Control PCB

D.1 Introduction

An outline of the operation and features of the PC-to-MCU Shared Memory PCB is given here. This board serves as a medium for communication between three connected microcontrollers and the workcell PC (via the Parallel Expansion PCB). This communication is implemented through the use of shared memory. In addition, this board also contains hardware to control the flow of serial information to, and from, the PC (serial network control) and performs logic-level conversion.

A block diagram of the board electronics is shown in Fig. 5.4.

Figs. D.1 through D.6, contained later in this chapter, show the schematic of the PC-to-MCU Shared Memory board. Fig. D.7 is a diagram showing the part placement. Figs. D.8 and D.9 show the top and bottom copper foil patterns, respectively.

D.2 Operation

The PC-to-MCU Shared Memory board serves three specific purposes:

- (1) Achieve a shared-memory interface;
- (2) Control the RS-232 serial 'network'; and
- (3) Perform logic-level conversion.

Accordingly, each of these facilities is the topic of a dedicated subsection.

D.2.1 Shared Memory Interface

The implementation of the shared-memory interface is similar to that found on the MCU-to-MCU shared-memory PCB, described in Appendix E.

The shared memory consists of 3 ICs: Cypress Semiconductor CY7C135 4Kx8 Dual-Port Static RAMs. These devices are no different than normal static RAM except for the fact that there are two sets of address and control lines used to access a common memory array.

The location at which a specific memory device appears in the connected MCU's logical address space is determined by the chip-select line originating from the 'HC138 decoder IC located on the MCU board. The following table outlines the base MCU memory locations at which the memory appears and the Parallel Expansion PCB signal mnemonic connected to the active-low chip-enable input. Note that since a dedicated digital output on the parallel expansion PCB is used to select a memory device (along with dedicated active-low read and write signals), to give a PC-side 'address' would be meaningless.

Shared Memory Information			
Description	IC	MCU Location (Hexadecimal)	PC Mnemonic
PC to MCU1	U5	0x4000	CS_UC1_MEM*
PC to MCU2	U2	0x4000	CS_UC2_MEM*
PC to MCU3	U3	0x4000	CS_UC3_MEM*

Table D.1: PC-to-MCU Shared Memory PCB MCU memory base locations and PC selection signal mnemonics.

For presentation of the shared-memory usage (and the memory map), see Chapter 6.

D.2.2 Serial Network Control Hardware

The serial-network control hardware connects a single serial port (originating from, and destined for, the PC) to one of the three system microcontrollers. These serial signals are at TTL levels and since they are passed through the Serial-Level Conversion PCB prior to connection to the PC serial port.

To control which of three microcontrollers are connected to the PC receive or transmit signal, six control lines are used. These signals originate from the Parallel Expansion PCB. Their mnemonic and function is described in tabular form, below.

Serial Network Control Signals	
Signal Mnemonic	Purpose
UC1.SER_RX	Connects PC transmit to UC1 receive circuitry.
UC1.SER_TX	Connects PC receive to UC1 transmit circuitry.
UC2.SER_RX	Connects PC transmit to UC2 receive circuitry.
UC2.SER_TX	Connects PC receive to UC2 transmit circuitry.
UC3.SER_RX	Connects PC transmit to UC3 receive circuitry.
UC3.SER_TX	Connects PC receive to UC3 transmit circuitry.

Table D.2: PC-to-MCU Shared Memory PCB MCU memory base locations and PC selection signal mnemonics.

This control is achieved through use of 'HC08 AND gates. An 'HC03 (with open-drain outputs) is used on the PC receive circuitry to allow the controlled serial outputs from the microcontrollers to be connected together (wired-OR connection).

Although it is possible to permit flow from all three MCUs to the PC simultaneously, data collision will occur. Accordingly, only one UCx.SER_TX signal should be brought low at a given time.

D.2.3 Logic-Level Conversion

As Low-power Schottky (LS) and High-Density CMOS (HC) series TTL levels are not compatible, it is necessary to provide circuitry to interface the two. Conversion is not necessary for connection to the Cypress shared-memory ICs as they are compatible with both levels. However, MCU control lines which originate from the PC (via the Parallel Expansion PCB) such as RESET*, DWNLD*, etc., require the level-conversion. To achieve this, 'LS05 open-collector inverters are used (U1-U4). With use of a pull-up resistor, the output of these devices can drive HC-TTL loads.

D.3 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the PC-to-MCU Shared Memory board.

D.3.1 Power Connector: J1

This connector is used provide power to the electronics. However, it is possible to apply power to this PCB through the Parallel Expansion PCB (described in Appendix C) via connector J5.

J1 – Power Connector			
Pin	Mnemonic	Direction	Description
1	Vcc	-	+5V D.C. power.
2	Vcc	-	+5V D.C. power.
3	-	-	No connection.
4	-12V	-	-12V D.C. power.
5	+12V	-	+12V D.C. power.
6	GND	-	Ground (return).
7	GND	-	Ground (return).
8	-5V	-	-5V D.C. power.

Table D.3: PC-to-MCU Shared Memory PCB connector J1 pinouts.

D.3.2 MCU Bus Connector: J2–J4

For a description of the signals available from the three microcontrollers connected to this board, refer to the MCU Bus Connector description in Appendix A. Signal directions for this connector are opposite to those described in that table.

D.3.3 Digital I/O Connector: J5

This connector is used to interface to the Parallel Expansion PCB, detailed in Appendix C. For more information on J5, refer to this appendix, noting that the signal directions detailed there are opposite to those at this connector.

D.4 Design Limitations

For a discussion of design limitations with respect to the shared-memory portion of this PCB, please refer to the corresponding section in Appendix E. MCU-to-MCU Shared Memory PCB.

D.5 Future Improvements

The portion of the serial network control hardware including U11 (7HC03) was not implemented on the PCB. It is prototyped using a small wire-wrapped board, connecting to the pins of J6. This portion of the circuitry should be included on the next revision of the board.

Additionally, for more flexible design, it may be worthwhile examining a form of communication which utilizes a multi-drop serial approach rather than point-to-point

RS-232C connection. This would complicate the design and require a specialized microcontroller programming interface running on the PC workcell, but the gained modularity would be an advantage.

The requirement for TTL level conversion (the 'LS05 ICs) could be avoided by replacing the registered outputs on the Parallel Expansion PCB with technology that can drive the HC inputs directly. This would save a great deal of space and reduce power consumption.

For future improvements regarding the shared-memory, please see Appendix E, a description of the MCU-to-MCU Shared Memory PCB.

D.6 Schematic Diagrams and Board Layout

Figures D.1 through D.9 comprise the schematic and board layout of the PC-to-MCU Shared Memory board.

D.6.1 Part List

PC-to-MCU Shared Memory Board Parts			
	Number Used	Part Type	Designators
1	1	Molex 4-Pin Locking Header	J7
2	10	0.1uF	C1-C10
3	2	4.7K	R1 RP1
4	1	47uF	C11
5	3	CY7B135-35JC	U5 U6 U7
6	4	DM74LS05	U1 U2 U3 U4
7	1	HEADER 6X2	J6
8	4	HEADER 30X2	J2 J3 J4 J5
9	2	MC74HC08A	U9 U10
10	1	Molex 8-pin Locking Header	J1
11	1	SN74HC03	U11

Table D.4: PC-to-MCU Shared Memory board parts list.

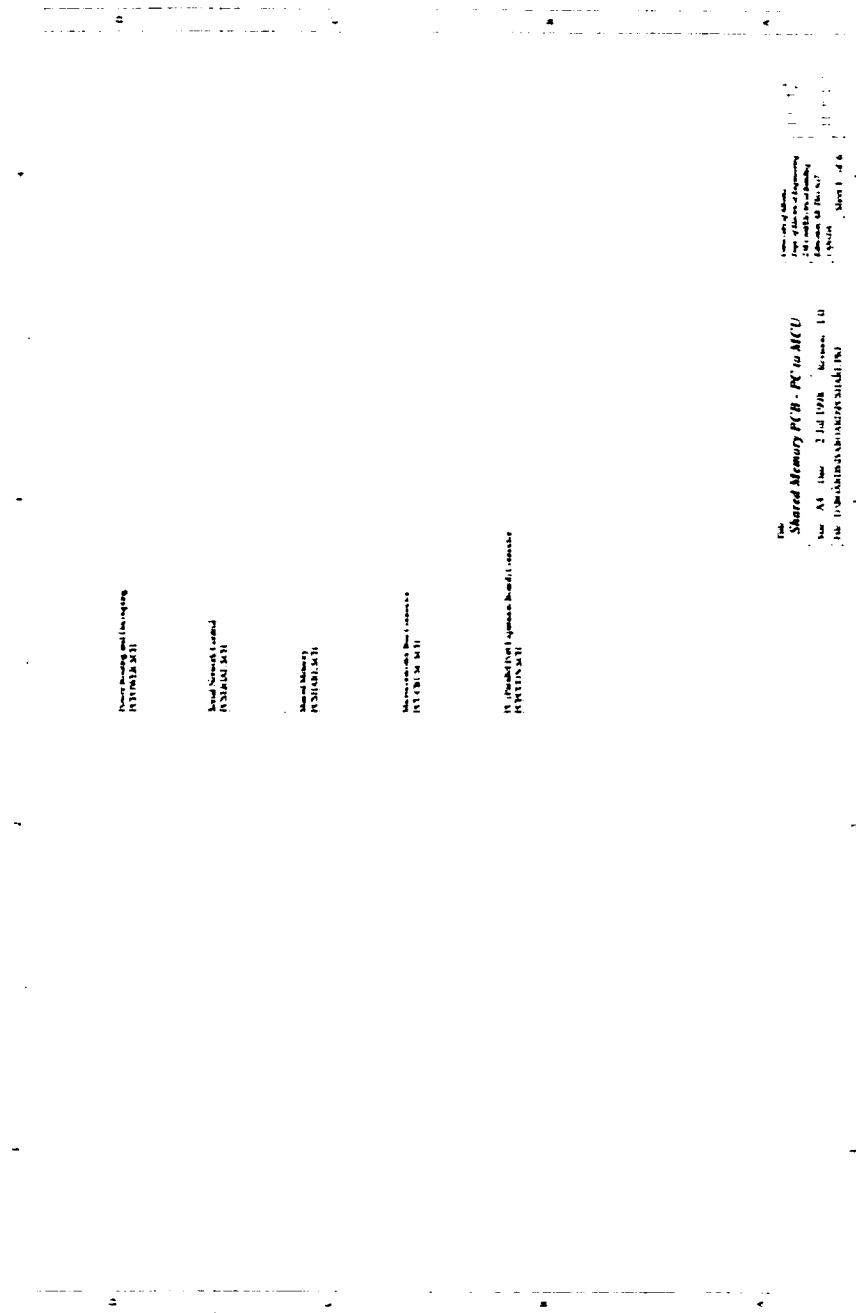


Figure D.1: PC-to-MCU Shared Memory PCB schematic (1 of 6).

Figure D.2: PC-to-MCU Shared Memory PCB schematic (2 of 6).

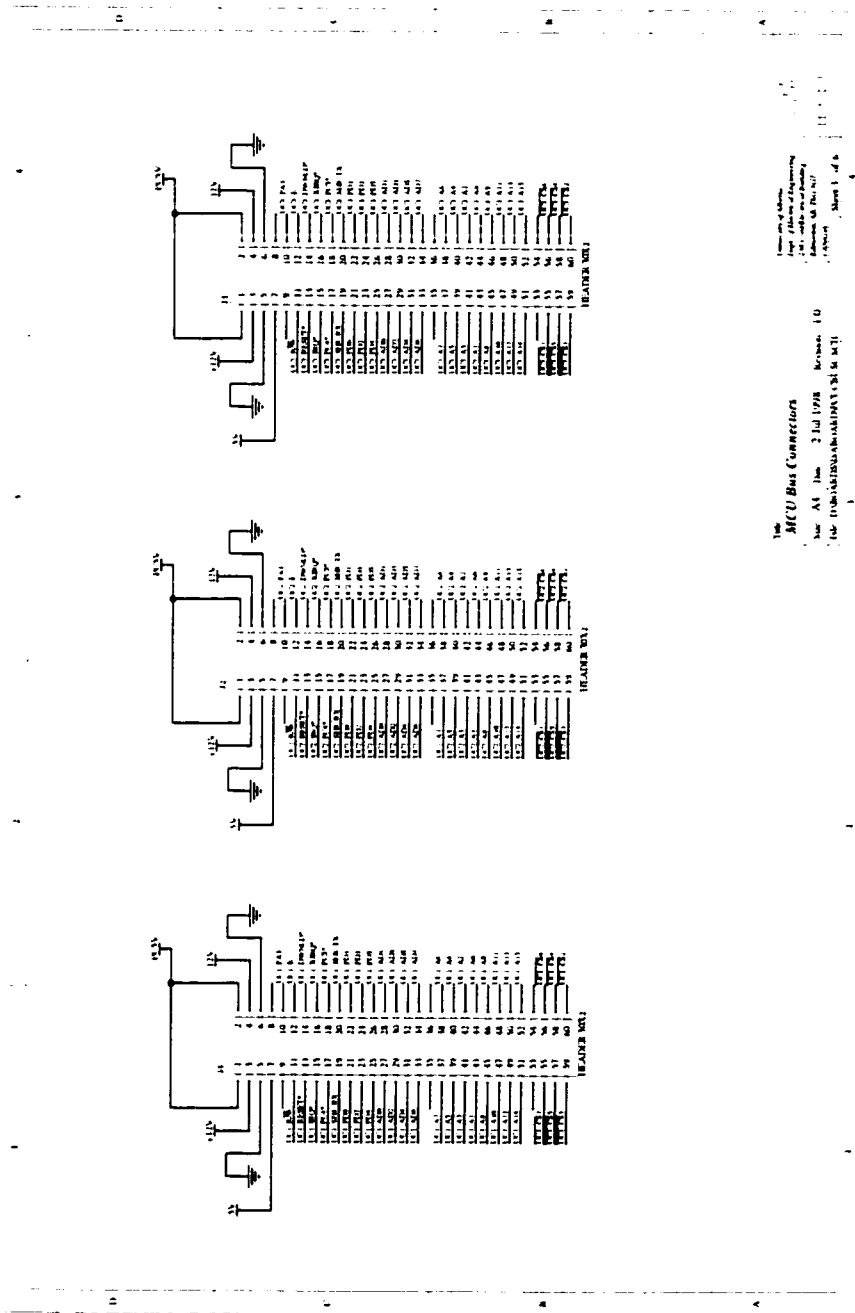


Figure D.3: PC-to-MCU Shared Memory PCB schematic (3 of 6).

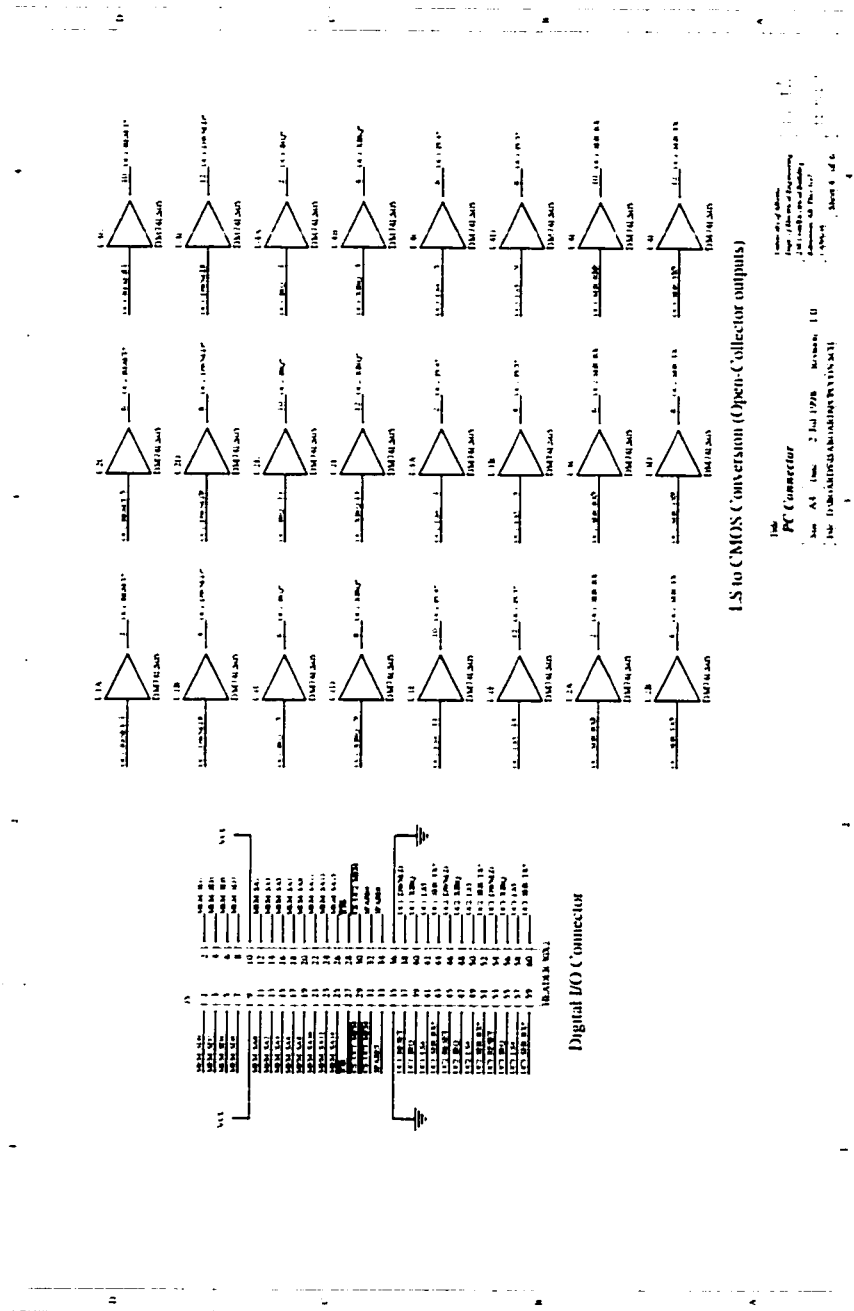


Figure D.4: PC-to-MCU Shared Memory PCB schematic (4 of 6).

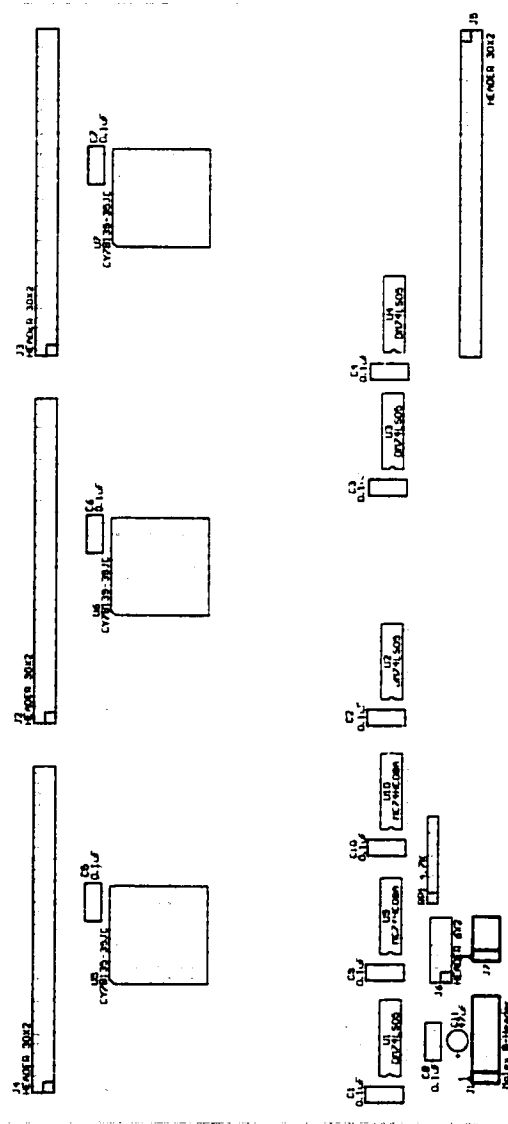


Figure D.7: PC-to-MCU Shared Memory PCB part placement diagram.

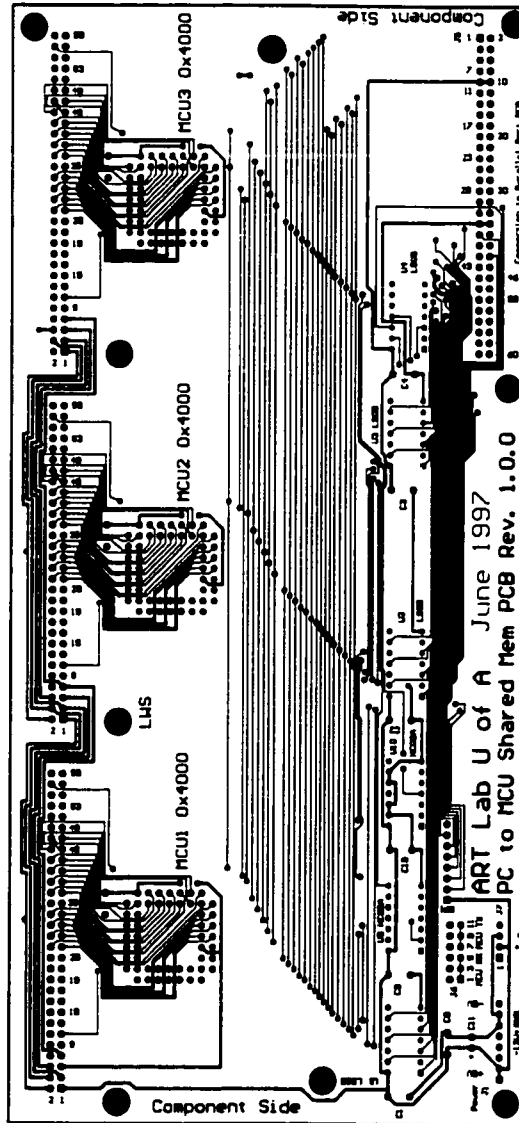


Figure D.8: PC-to-MCU Shared Memory PCB top-layer copper foil pattern. Not to scale.

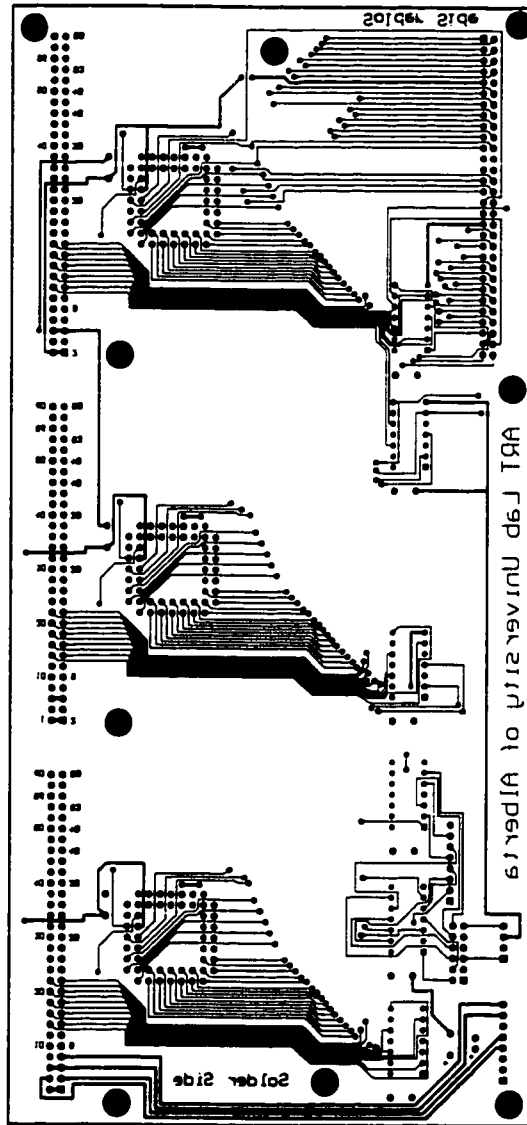


Figure D.9: PC-to-MCU Shared Memory PCB bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix E

MCU-to-MCU Shared Memory PCB

E.1 Introduction

An outline of the operation and features of the MCU-to-MCU Shared Memory PCB is given here. This board serves as a medium for communication between three connected microcontroller boards. Communication is implemented through use of shared-memory (dual port static RAM).

A block diagram of the board electronics is shown in Fig. 5.7.

The board is relatively simple and contains only three integrated circuits: Cypress Semiconductor CY7C135 4Kx8 Dual-Port Static RAMs. These devices are quite expensive (approximately \$35.00 per unit, when individual units can be found).

Figs. E.1 through E.4, contained later in this chapter, show the schematics of the MCU-to-MCU shared memory board. Fig. E.5 is a diagram showing the part placement. Figs. E.6 and E.7 show the top and bottom copper foil patterns, respectively.

E.2 Operation

This PCB implements shared memory. Access to the on-board memory is no different than accessing other memory. The location at which a given memory device appears in the connected MCU's logical address space is determined by the chip-select line originating from the 'HC138 decoder IC located on the MCU board. The following table outlines the base memory locations at which the memory appears:

Base Memory Locations		
Connected MCUs	IC	Location (Hexadecimal)
MCU1 to MCU2	U1	MCU1: 0x2000, MCU2: 0x2000
MCU1 to MCU3	U2	MCU1: 0x3000, MCU3: 0x2000
MCU2 to MCU3	U3	MCU2: 0x3000, MCU3: 0x3000

Table E.1: MCU-to-MCU Shared Memory PCB Memory address bases.

For presentation of the shared-memory usage (and the memory map), see Chapter 6.

E.3 Connector Descriptions

E.3.1 MCU Bus Connectors: J2–J4

For a description of the signals available from the three microcontrollers connected to this board, refer to the MCU Bus Connector description in Appendix A. Directions for this board are opposite to those described at that location.

E.4 Design Limitations

A peculiarity of the design is that the address lines to each side of a given shared-memory IC cannot be the same for successful reads and writes. In other words, a lack of memory-access semaphore makes itself evident by reading and writing erroneous data. If all address lines of one side of the device are at 0 (addressing memory location 0), then a read from the other side will read a data value of 0.

To account for this characteristic, it was chosen that addresses on one side of the device would be made to ‘idle’ at an unused memory location: an address of 0.

E.5 Future Improvements

Although circuitry and access to the memory would be more involved, all shared memory devices should be replaced with Cypress Semiconductor CY7C1432 4Kx8 Dual-Port Static RAM with **Semaphores**. These devices include circuitry on-chip to help with access arbitration to specific memory locations. This would avoid the design limitation described above.

Another feature which should be added is the ability to change the memory location at which the SRAMs appear, perhaps via a jumper block.

E.6 Schematic Diagrams and Board Layout

Figures E.2 through E.7 comprise the schematic and board layout of the MCU-to-MCU shared memory board.

E.6.1 Part List

MCU-to-MCU shared memory Board Parts			
	Number Used	Part Type	Designators
1	3	0.1uF	C1 C2 C3
2	1	47uF	C4
3	3	CY7B135-35JC	U1 U2 U3
4	3	HEADER 30X2	J2 J3 J4

Table E.2: MCU-to-MCU shared memory board parts list.

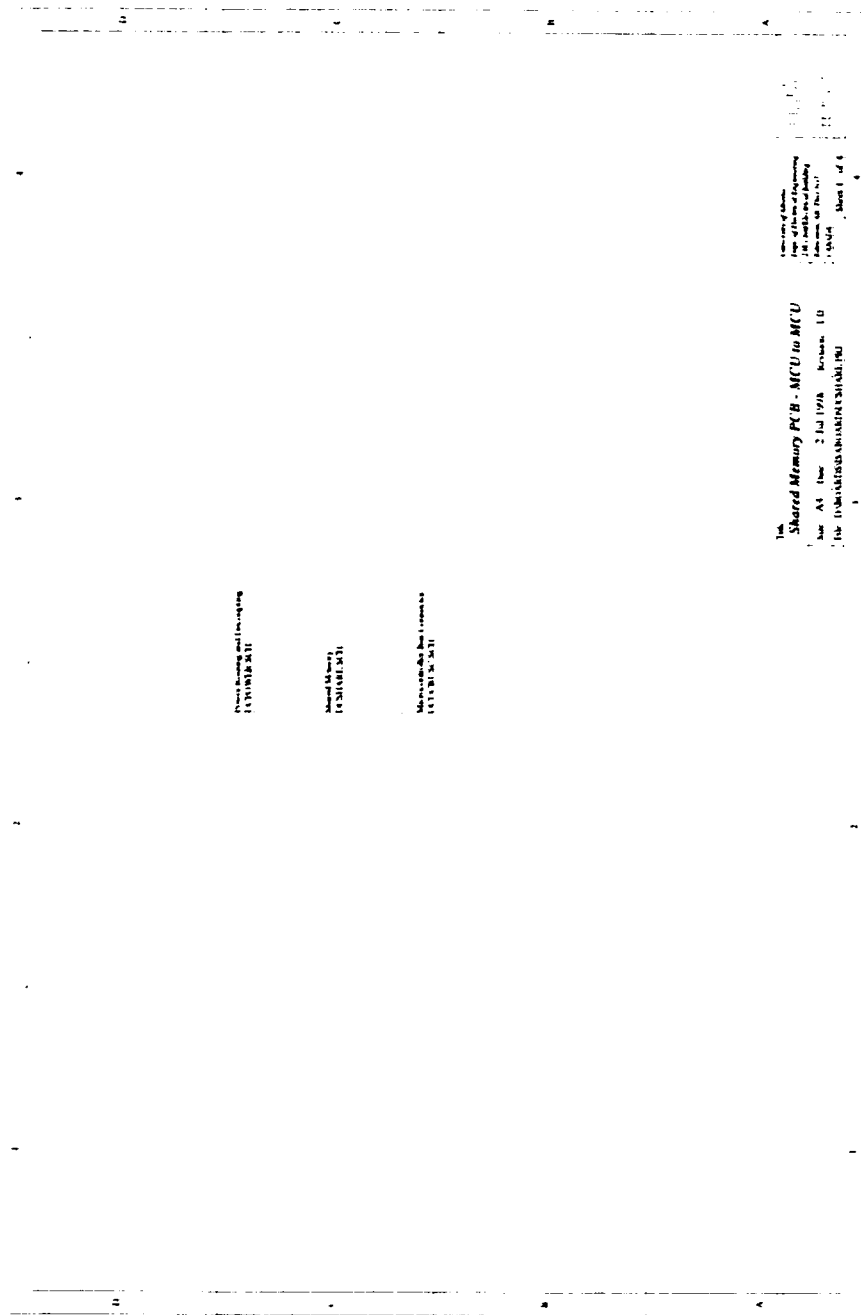


Figure E.1: MCU-to-MCU Shared Memory PCB schematic (1 of 4).

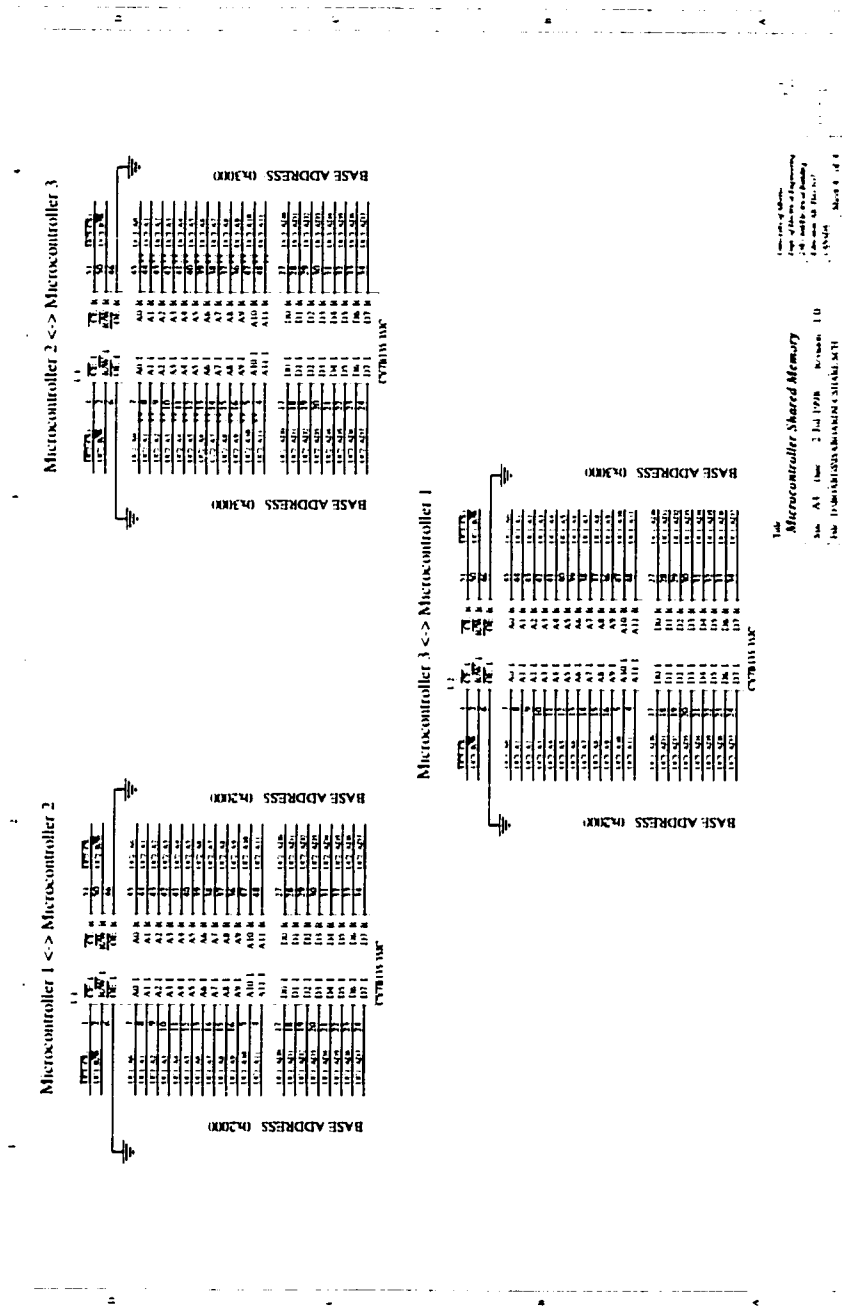


Figure E.4: MCU-to-MCU Shared Memory PCB schematic (4 of 4).

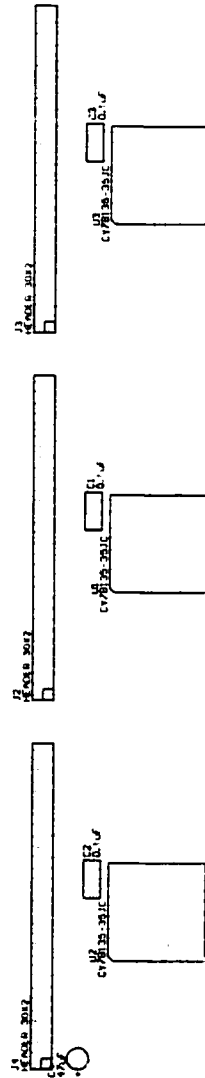


Figure E.5: MCU-to-MCU Shared Memory PCB part placement diagram.

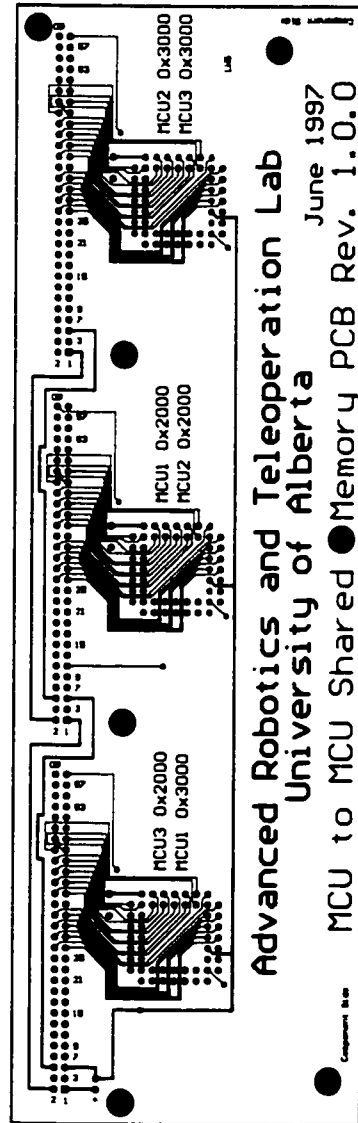


Figure E.6: MCU-to-MCU Shared Memory PCB top-layer copper foil pattern. Not to scale.

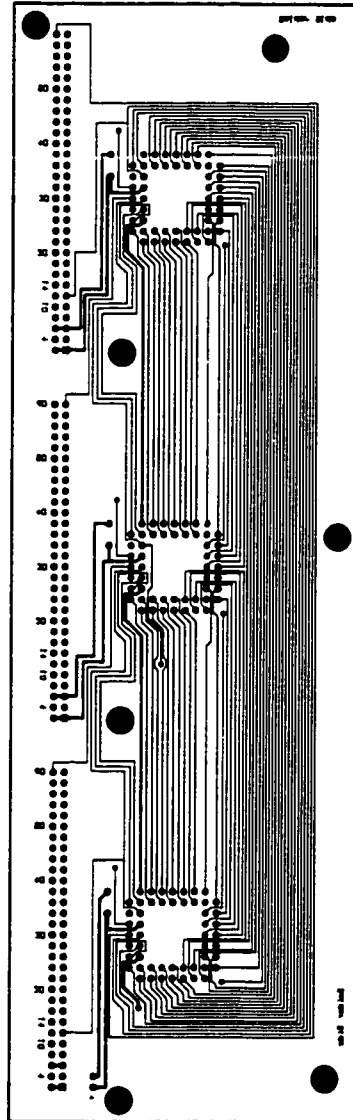


Figure E.7: MCU-to-MCU Shared Memory PCB bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix F

Navigation Hardware PCB

F.1 Introduction

An outline of the operation, features, and user-settable options of the Navigation Control Hardware PCB is given here. This board is used to interface to the DC servo motors that drive the robot. This interface consists of driver circuitry (an L298N dual H-bridge motor driver), and connections to, and signal conditioning for, optical encoders.

A block diagram of the board electronics is shown in Fig. 5.11.

The main component of this board is the SGS Thomson L298 dual H-bridge motor driver. This device can control two motors requiring up to 2 Amps each at 46 Volts. The only signals required to achieve control over a single motor are a direction bit (to select the polarity of the voltage applied by the H-bridge), and an active-high enable signal. Interface to the IR pairs used to read the optical disks on the drive shafts consists of appropriate selection of LED current-limiting resistor and phototransistor emitter resistance. The signal-conditioning is passing of the emitter signal through a Schmitt-trigger inverter (74C14A) to reduce noise. The output of these inverters are passed directly to the HC11 responsible for the motor control algorithm.

The primary reason that the L298 was chosen for driving the motors is the simplicity with which they are used (due to on-board circuitry such as generation of FET-switching voltages). One large disadvantage is the fact that the 2 Amp limit imposed on current-draw from the H-bridges is constrictive. Many motors which could be used in a robotics platform would potentially draw more current. For a solution to this problem, see the section describing future improvements.

Figs. F.1 through F.4, contained later in this chapter, show the schematic of the Navigation Control Hardware board. Fig. F.5 is a diagram showing the part placement. Figs. F.6 and F.7 show the top and bottom copper foil patterns, respectively.

F.2 Operation

The motor-driver circuitry, as mentioned previously, requires only two bits per motor for control purposes: a direction bit and an enable bit. The direction bits are derived from an 'HC595A serial-to-parallel converter which is connected to the HC11 SPI (synchronous serial peripheral interface). The enable bits are connected to HC11 output-compare pins on the HC11. The output-compare facility allows for generation of pulse-width modulated signals using an interrupt approach.

The following table describes the HC11 signals which are used to control the drive motors:

Motor Control Signals		
Purpose	Signal (Motor 1)	Signal (Motor 2)
Direction	SPI Byte 0 Bit 0	SPI Byte 0 Bit 1
Enable	PA6 (OC2)	PA5 (OC3)
Current Sense	PE0	PE1
Encoder Pulses	PA7	PA0

Table F.1: Motor control signals.

The feedback from the optical encoders connected to the driven shafts of the motor are fed into either PA7 (motor 0) or PA0 (motor 1), the first of which is associated with a pulse accumulator. The pulse accumulator can be used to count the number of pulses applied. If this count is read periodically, the rotational velocity of the output shaft is known. Although there is no pulse accumulator associated with PA0, the same functionality is implemented using an interrupt-service routine. This method of determining velocity (and absolute position) is used as the basis for the motor control-loops and the dead-reckoning to yield the global coordinates and orientation of the workcell.

F.3 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the Navigation Control Hardware board.

F.3.1 Encoder Connectors: J1, J2

J1, J2 – Encoder Connectors			
Pin	Mnemonic	Direction	Description
1	ENCPWRx	Out	Current-limited IR LED anode connection.
2	GND	-	Ground return (from IR LED cathode).
3	Vcc	-	+5V to IR phototransistor collector.
4	ENCDETx	In	Analog IR phototransistor emitter level.

Table F.2: Navigation Control Hardware PCB connector J1, J2 pinouts.

F.3.2 Battery-level Sensing Connector: J3

This connector is unused in the current implementation of the workcell but is included for thoroughness. These connections are nothing more than a direct connection to the A/D facility on the connected HC11 board.

J3 – Battery-level Sensing Connector			
Pin	Mnemonic	Direction	Description
1	PE4	In	Connection to MCU A/D converter.
2	PE5	In	Connection to MCU A/D converter.
3	PE6	In	Connection to MCU A/D converter.
4	PE7	In	Connection to MCU A/D converter.
5	NC	-	Unused.
6	NC	-	Unused.
7	Vcc	-	+5V
8	GND	-	Ground.

Table F.3: Navigation Control Hardware PCB connector J3 pinouts.

F.3.3 Motor Power Connector: J4

J4 – Motor Power Connector			
Pin	Mnemonic	Direction	Description
1	VMOT	In	Motor Power (46V maximum).
2	GND	-	Ground return.

Table F.4: Navigation Control Hardware PCB connector J4 pinouts.

F.3.4 Port Connector: J5

Details of connector J5, used to interface with an HC11 MCU board, is located in the Appendix A. Signal directions relative to this board are opposite to those documented there.

F.4 Future Improvements

Future improvements to this board would include:

- Adding a general-purpose digital output connector for the unused bits on the HC595A serial-to-parallel converter (SPI).
- Using discrete H-bridge drivers to allow connection to motors with higher current demands.
- Addition of opto-isolators to reduce noise and improve resilience to voltage fluctuations.
- Addition of circuitry to allow for decoding of quadrature signals which are commonly used as feedback for brushless DC servo motors (perhaps Hewlett Packard HCTL-20XX quadrature decoders/counters).

F.5 Schematic Diagrams and Board Layout

Figures F.1 through F.7 comprise the schematic and board layout of the Navigation Control Hardware board.

F.5.1 Part List

Navigation Control Hardware Board Parts			
	Number Used	Part Type	Designators
1	4	0.1nF	C1-C4
2	8	1N914	D1-D8
3	2	9.53K	R2 R4
4	2	10 Ohm, 5W	R5 R6
5	2	47uF	C5 C6
6	2	100 Ohm	R1 R3
7	2	100nF	C7 C8
8	3	2-pin power connector	J4 M1 M2
9	1	HEADER 20X2	J5
10	1	L298N	U2
11	1	MC74HC14A	U1
12	1	MC74HC595A	U3
13	2	Molex 4-Pin Header	J1 J2
14	1	Molex 8-Pin Header	J3

Table F.5: Navigation Control Hardware board parts list.

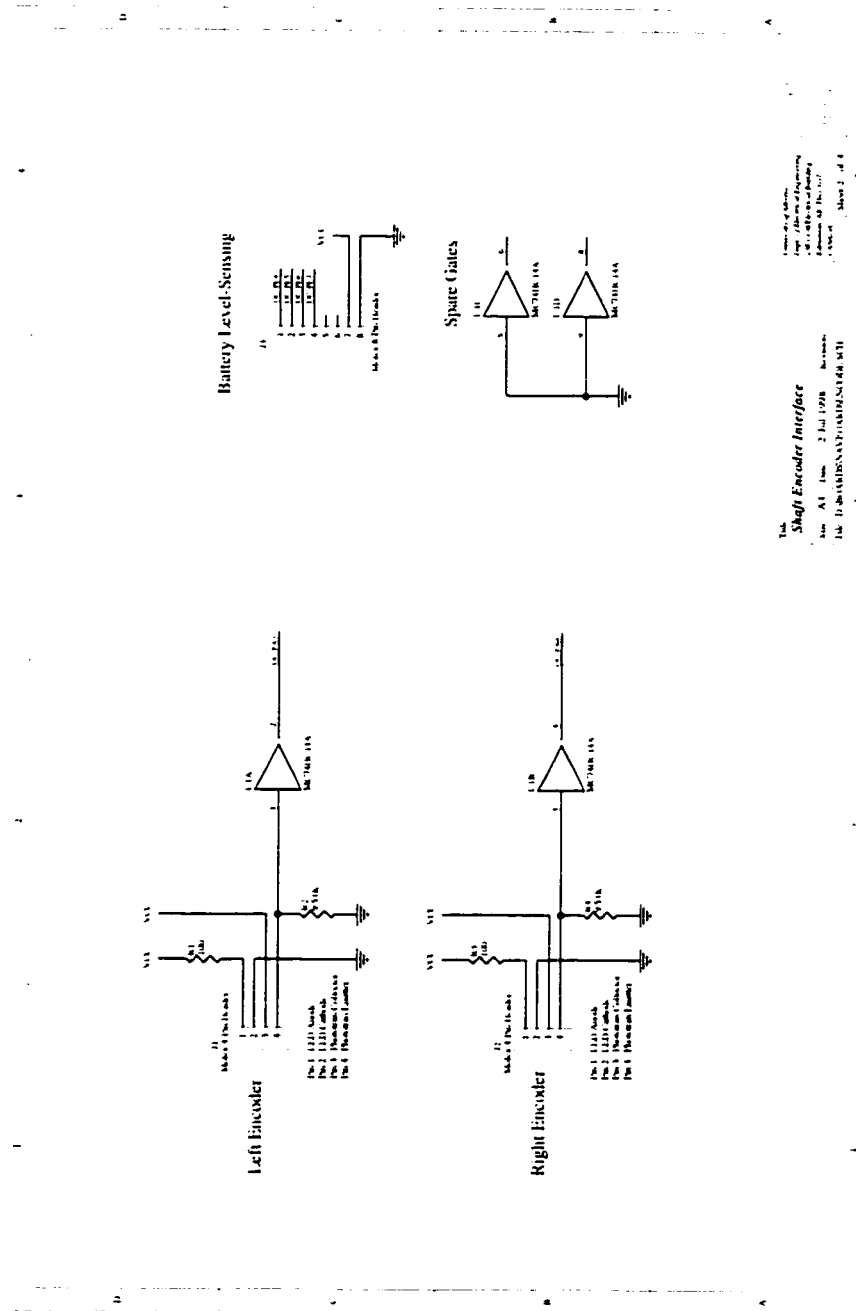


Figure F.2: Navigation Control Hardware schematic (2 of 4).

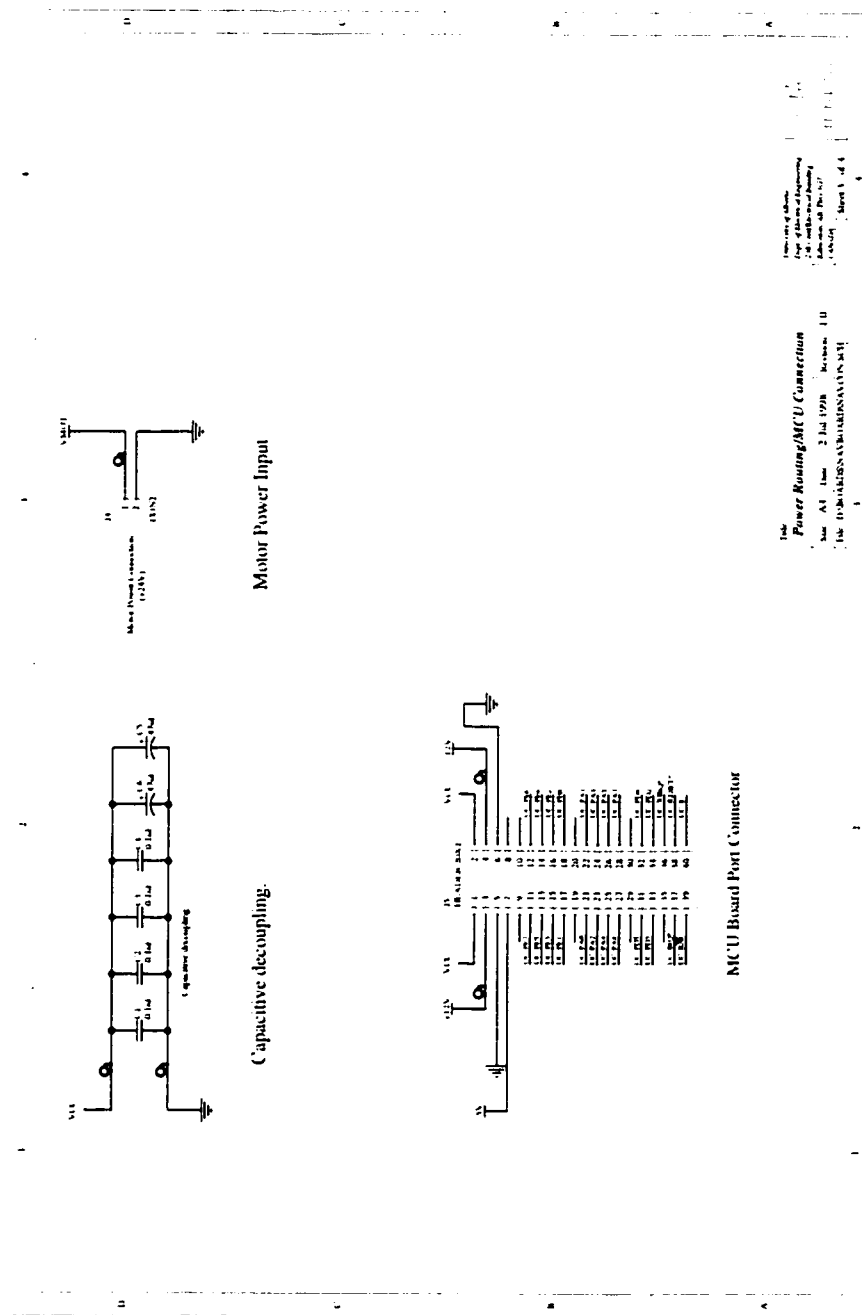


Figure F.3: Navigation Control Hardware schematic (3 of 4).

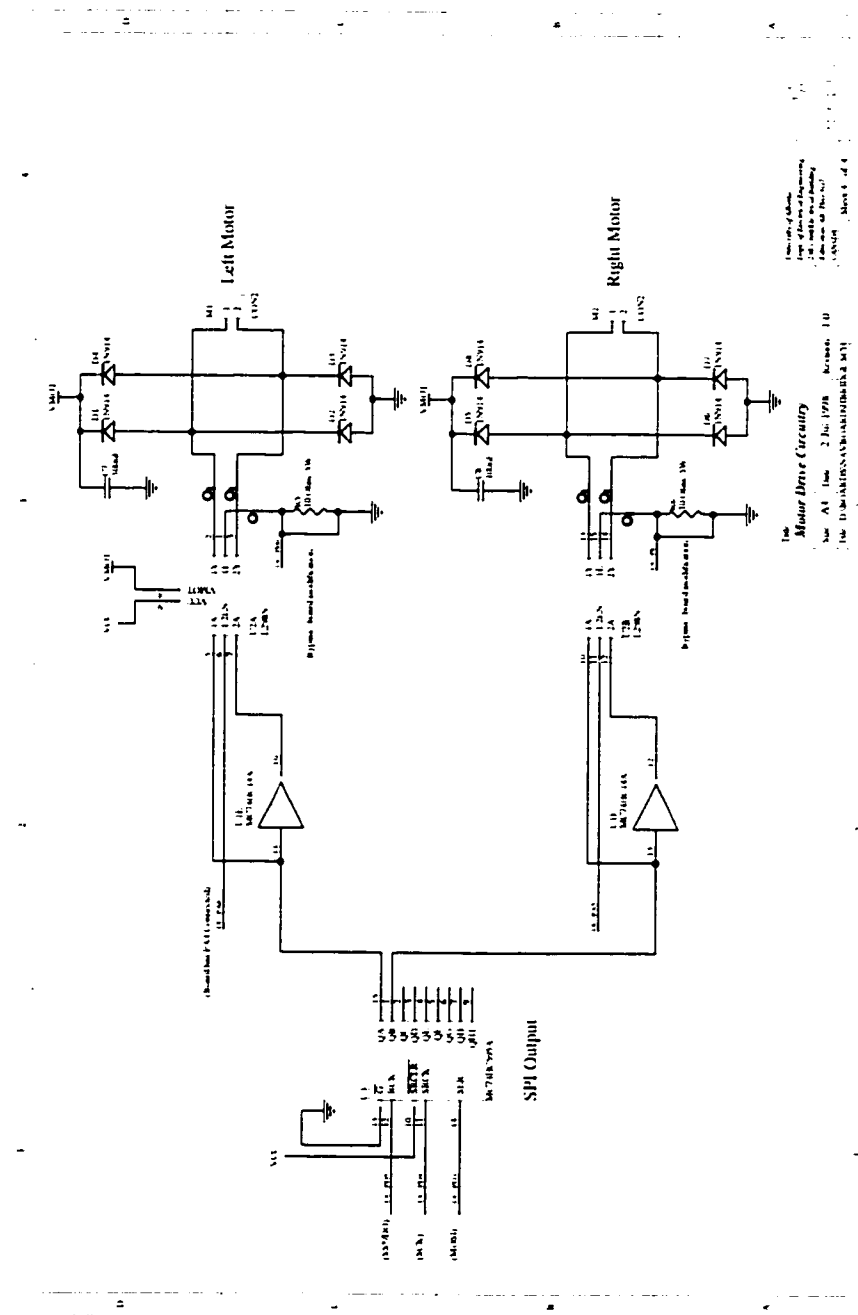


Figure F.4: Navigation Control Hardware schematic (4 of 4).

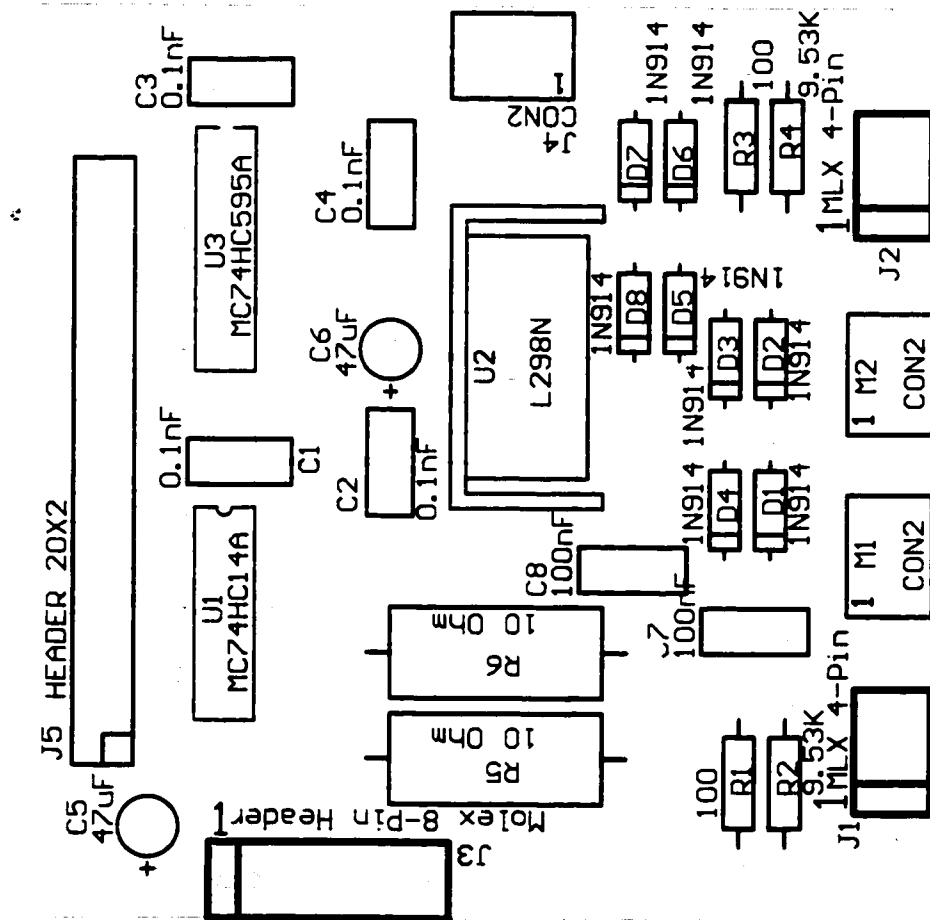


Figure F.5: Navigation Control Hardware part placement diagram.

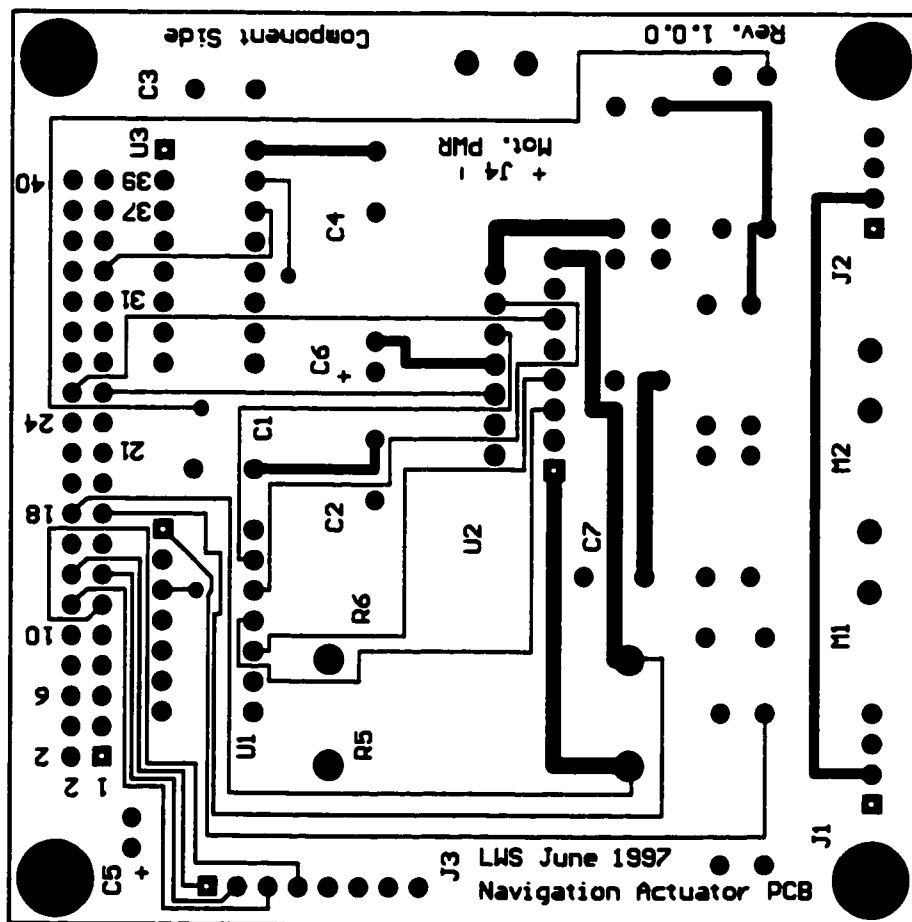


Figure F.6: Navigation Control Hardware top-layer copper foil pattern. Not to scale.

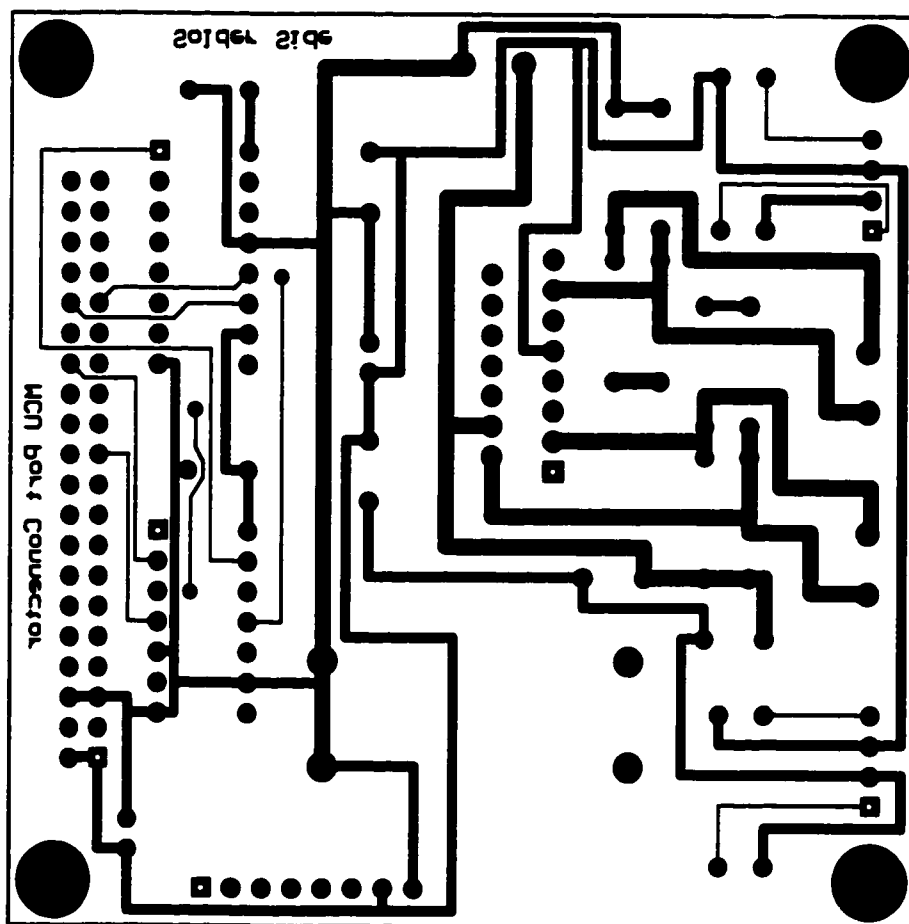


Figure F.7: Navigation Control Hardware bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix G

Near-Range IR Proximity Sensor PCB

G.1 Introduction

An outline of the operation, features, and user-settable options of the Near-range proximity sensor PCB is given here. This board, also referred to as the near-range IR, or IR board, is used to control and interface up to 32 near-range infrared pairs. Facility is provided to apply power to the infrared LED, and an analog multiplexing scheme is used to select which 'bank' of infrared phototransistors are connected to the HC11 A/D converter.

A block diagram of the board electronics is shown in Fig. 5.9.

This board does not necessarily need to be used to interface with IR pairs. Instead, it could simply be used for digital output (32 signals) and analog input (32 signals multiplexed to 8 signals). Four cascaded 74HC595A serial-to-parallel converters are used to provide the 32 digital outputs. Used in conjunction with Motorola's Serial Peripheral Interface (SPI), the use of the '595As in this manner is simple and efficient. The digital outputs are not able to provide sufficient current to the IR LEDs directly, and therefore NPN switches are used (2N2222). The collector resistances are selected to give appropriate current to the Knight Lites KIE-6304 GaAs IR LEDs. The 32 phototransistor outputs are multiplexed by Motorola 74HC4052 dual 4-to-1 analog multiplexer ICs. Control signals for the multiplexers (to determine which of the 4 inputs is connected to the output) are taken from the HC11 MCU board (pins A5 and A6) via the port connector. The output from the multiplexers are shunted to ground via 6.2V Zener diodes. These devices were experimentally chosen as appropriate for providing the necessary impedance to 'drain' the A/D inputs on the HC11. Without these diodes (or some other impedance to ground), the Successive-Approximation A/D converters will not lose charge after a sample has been made, yielding erroneous results.

As with the bump-sensor PCB (described in Appendix H), the primary advantage of using the SPI is the simplicity with which the interface to digital outputs can be achieved. There is no address-decoding required since 'HC595A devices can be cascaded. This number of digital outputs is required since the current-draw of a single IR LED is large (100 mA). To have all devices permanently 'on' would drain the power source quickly. In addition, having control over specific IR pairs allows use of novel ambient-light filtration algorithms and sensor fusion algorithms in software.

Figs. G.1 through G.6, contained later in this chapter, show the schematic of the Near-range IR Detector board. Fig. G.7 is a diagram showing the part placement. Figs. G.8 and G.9 show the top and bottom copper foil patterns, respectively.

G.2 Operation

This section describes operation of the near-range IR PCB.

G.2.1 Digital Output (Emitter Control)

Once the SPI is configured, transfer is initiated by writing a byte to the SPDR (SPI Data Register). The SPI operates as a 'ring' buffer. Whatever the byte placed in the SPDR, a transfer is initiated, and the byte is pushed out to whatever device is connected to the serial output: in this case, A 74HC595A serial-to-parallel converter. Simultaneously, data from any input devices connected to the serial input is shifted in. In order to 'latch' the serial data into the 'HC595A output registers, a register clock line must be strobed. This could be performed by a memory-mapped chip-select circuit, but a normal digital output is used (microcontroller signal PD5). In addition, an active-low output enable pin is connected to another general-purpose digital output (PA4) to allow another form of control over the digital outputs: when the enable pin is high, all the digital outputs will remain in high-impedance mode effectively disabling all IR emitters.

G.2.2 Analog Input (Detector Reading)

Reading of analog levels from the IR phototransistors takes two distinct steps:

- (1) Select the 'bank' that the the sensor resides in via PA5 and PA6.
- (2) Sample the voltage presented to the Port E input.

The following table describes the position of the digital outputs and the IR LED output which they drive. Note that PA4 of the driving HC11 must be low in order for the level of the bit to be driven. This table also indicates the bank in which they reside, and the Port E input to which they are connected when the bank is selected.

IR Pair	SPI Output Bit	Connector	Analog I/P
Bank 0 (PA5 = 0, PA6 = 0)			
1	Byte 0 Bit 0	J6	PE0
2	Bit 1	J7	PE1
3	Bit 2	J8	PE2
4	Bit 3	J9	PE3
5	Bit 4	J10	PE4
6	Bit 5	J11	PE5
7	Bit 6	J12	PE6
8	Bit 7	J13	PE7
Bank 1 (PA5 = 0, PA6 = 1)			
9	Byte 1 Bit 0	J14	PE0
10	Bit 1	J15	PE1
11	Bit 2	J16	PE2
12	Bit 3	J17	PE3
13	Bit 4	J18	PE4
14	Bit 5	J19	PE5
15	Bit 6	J20	PE6
16	Bit 7	J21	PE7
Bank 2 (PA5 = 1, PA6 = 0)			
17	Byte 2 Bit 0	J22	PE0
18	Bit 1	J23	PE1
19	Bit 2	J24	PE2
20	Bit 3	J25	PE3
21	Bit 4	J26	PE4
22	Bit 5	J27	PE5
23	Bit 6	J28	PE6
24	Bit 7	J29	PE7
Bank 3 (PA5 = 1, PA6 = 1)			
25	Byte 2 Bit 0	J30	PE0
26	Bit 1	J31	PE1
27	Bit 2	J32	PE2
28	Bit 3	J33	PE3
29	Bit 4	J34	PE4
30	Bit 5	J35	PE5
31	Bit 6	J36	PE6
32	Bit 7	J37	PE7

Table G.1: IR Pair input connector-to-bit correspondence and analog input.

G.3 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the Near-range Proximity PCB.

G.3.1 Port Connector: J1

Details of connector J1, used to interface with an HC11 MCU board, is located in the Appendix A. Signal directions relative to this board are opposite to those documented there.

G.3.2 IR Pair Power Connectors: J2–J5

These 2-pin power connectors are used to provide ground return for the Infrared LEDs and power to the IR phototransistors. Using a daisy-chain approach, power signals to all the IR pairs is efficient with regard to pin-count, but connection convenience is compromised, as outlined in the section on future improvements.

J2–J5: IR Pair Power Connectors			
Pin	Mnemonic	Direction	Description
1	Vcc	O	Power output.
2	GND	-	Ground return.

Table G.2: Near-Range Proximity PCB connectors J2–J5: sensor power connections.

G.3.3 J5–J37: IR Pair Signal Connectors

These 2-pin connectors are used to provide the signal interface to specific IR pairs (power connection is made through connectors J2–J5).

J5–J37: IR Pair Signal Connectors			
Pin	Mnemonic	Direction	Description
1	LEDPWRQx	O	IR LED power output (transistor driven).
2	PTx	I	IR Phototransistor input.

Table G.3: Near-Range Proximity PCB connectors J5–J37: sensor signal connections.

G.4 Design Limitations

No performance-hindering limitations are known at this time.

G.5 Future Improvements

Future designs should accommodate 4-pin connectors for specific IR pairs, rather than providing power and ground through daisy-chain connectors. Although cabling will become bulky, flexibility of sensor positioning will greatly increase.

Interactive-C makes use of the output-compare circuitry connected to PA4, and therefore does not allow automatic (interrupt-driven) toggling of the state of PA4. If Interactive-C is retained for future designs, another output-compare pin should be used to allow use of this feature to 'strobe' the enable pins of the 'HC595A. This kind of modulation could be used to aid in ambient light rejection.

G.6 Schematic Diagrams and Board Layout

Figures G.2 through G.9 comprise the schematic and board layout of the Near-IR Proximity Detector board.

G.6.1 Part List

Near-range Proximity Detector Board Parts			
	Number Used	Part Type	Designators
1	8	0.1nF	C1-C8
2	32	1K	R41-R72
3	8	1N4735ATOR. 6.2V	D33-D40
4	32	2N2222	Q1-Q32
5	32	39 Ohm. 1/2 W	R1-R32
6	1	47uF	C9
7	1	HEADER 20X2	J1
8	4	MC74HC595A	U1-U4
9	4	MC74HC4052	U5-U8
10	36	Molex 2-Pin Header	J2-J37

Table G.4: Near-range Proximity Detector board parts list.

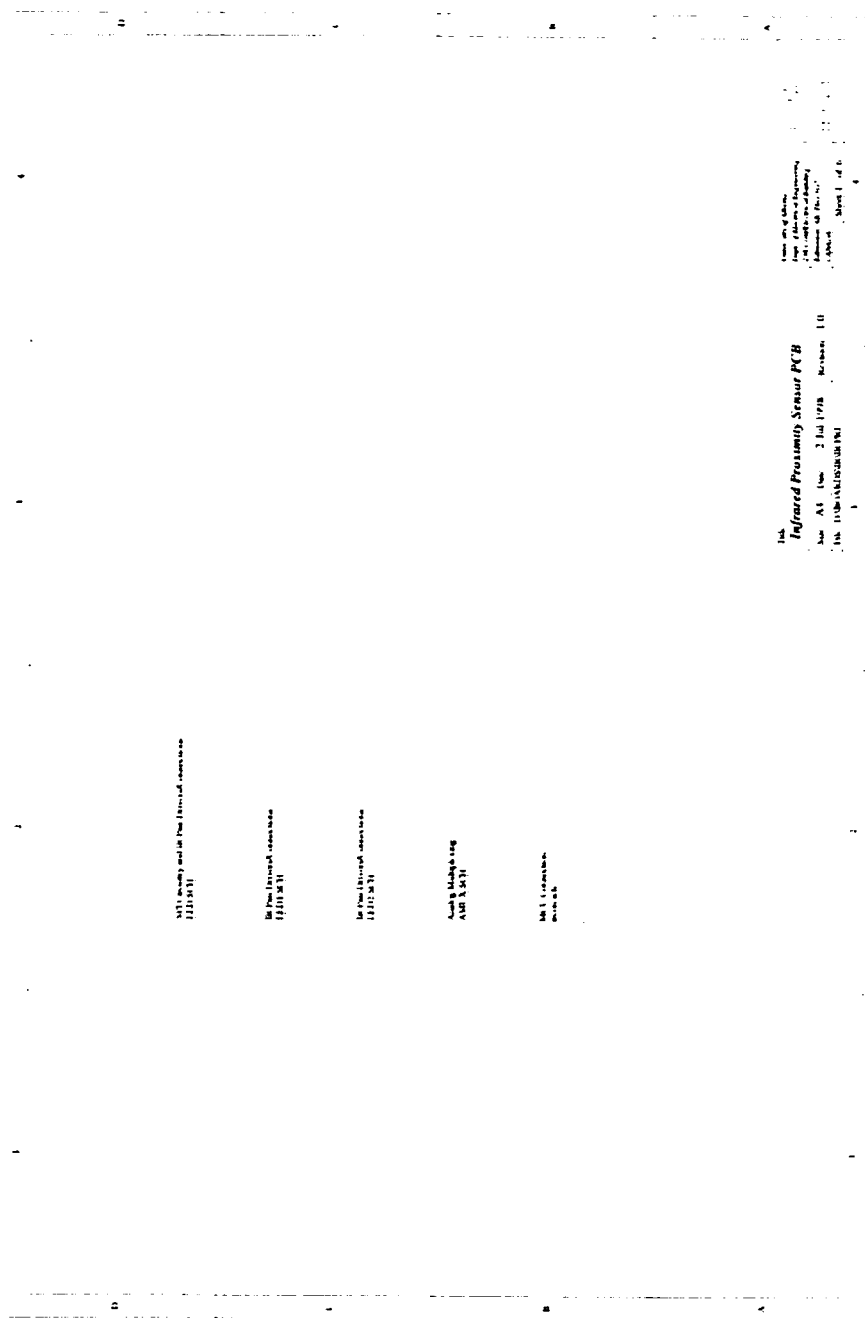
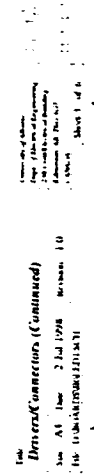
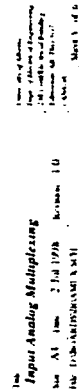


Figure G.1: Near-Range Proximity PCB schematic (1 of 6).



178



180

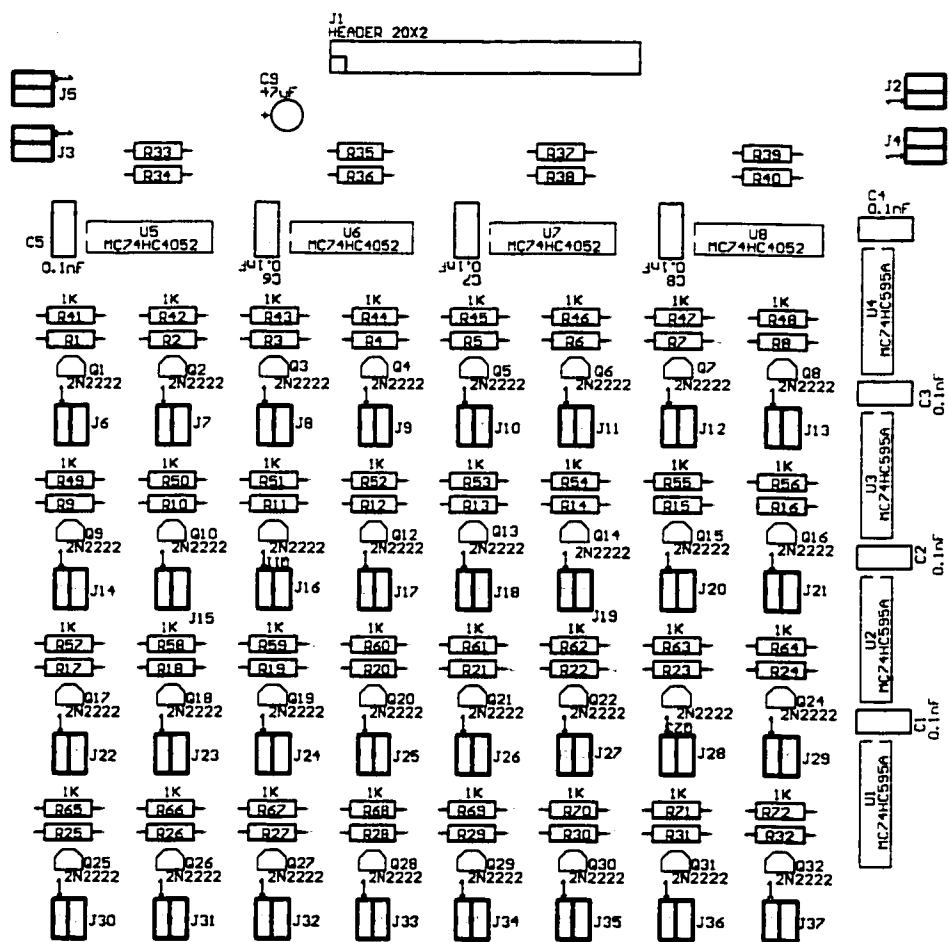


Figure G.7: Near-Range Proximity PCB part placement diagram.

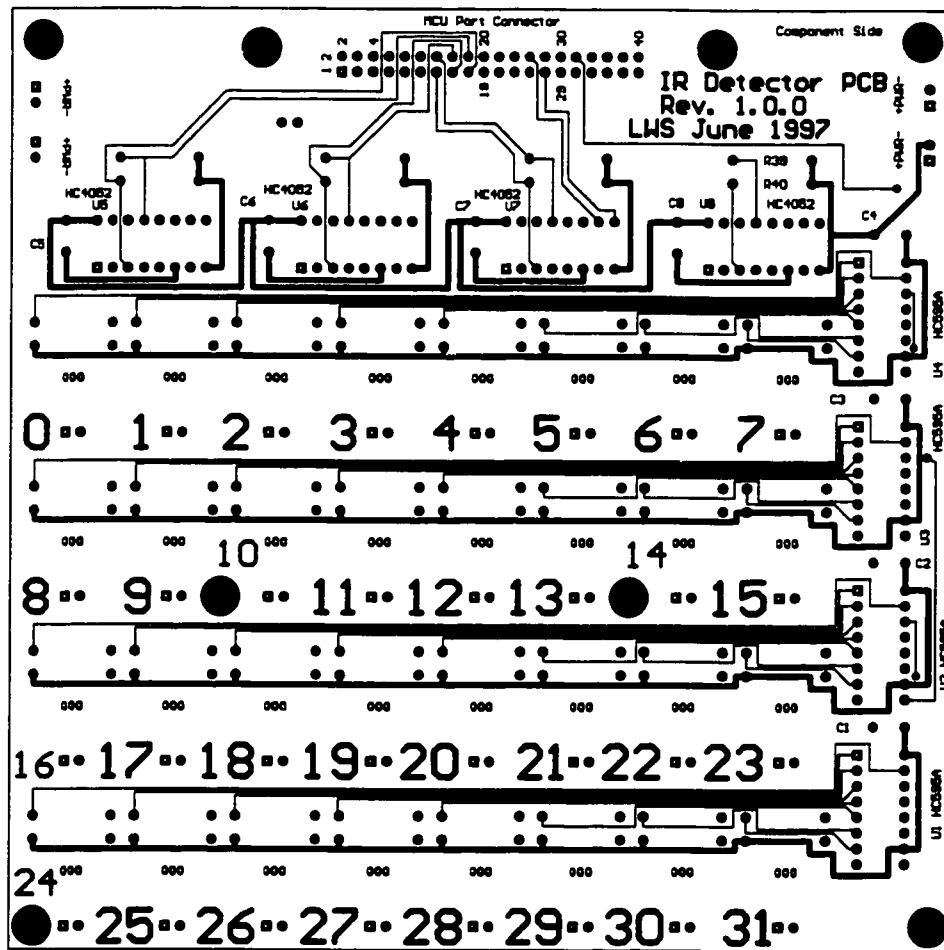


Figure G.8: Near-Range Proximity PCB top-layer copper foil pattern. Not to scale.

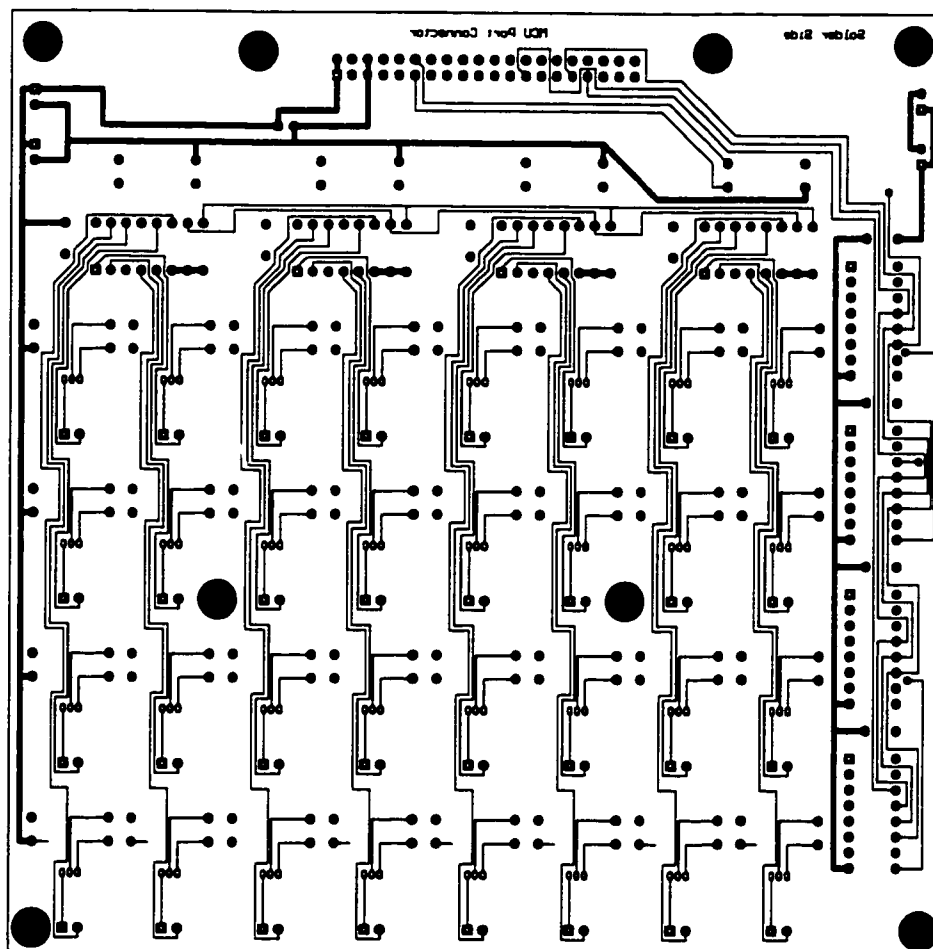


Figure G.9: Near-Range Proximity PCB bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix H

Bump Sensor PCB

H.1 Introduction

An outline of the operation, features, and user-settable options of the Bump Sensor PCB is given here. The purpose of this board is to interface to the many bump-sensors on the robotic workcell. It is connected to the Proximity Detector HC11 board via the 'port' connector. Connection to individual bump-switches is made via an array of 2-pin Molex connectors.

A block diagram of the board electronics is shown in Fig. 5.10.

The backbone of this board is the use of 'HC165 parallel-to-serial converters. Interfaced to the Motorola's Serial Peripheral Interface (SPI), two of these devices allow for 16 digital inputs (8 each). Each bump-switch connector has a single 4.7k pull-up resistor, allowing connection of more than one device in a wired-or configuration. Although this interconnection scheme is possible, only single bump-switches are connected to a given input in this project. Conceivably, this board could be renamed '16-bit Digital Input PCB'. For particulars of the SPI facility, refer to the Motorola 68HC11 documentation.

The primary advantage of using the SPI is the simplicity with which the interface to digital inputs can be achieved. There is no address-decoding required since 'HC165 devices can be cascaded serially.

Figs. H.1 through H.3, contained later in this chapter, show the schematic of the Bump Sensor PCB. Fig. H.4 is a diagram showing the part placement. Figs. H.5 and H.6 show the top and bottom copper foil patterns, respectively.

H.2 Operation

Once the SPI is configured, transfer is initiated by writing a byte to the SPDR (SPI Data Register). A transfer is initiated, and the byte is shifted out of the micro-

controller. Simultaneously, data from input devices is shifted in. Since this PCB has two 'HC165s cascaded, two bytes must be read in to get the data from all 16 digital inputs. In order to 'latch' the input data into the 'HC165 serial shift registers, their clock pin must be strobed. This could be performed by a memory-mapped chip-select circuit, but a normal digital output is used (PD5).

The following table describes the two bytes as they relate to the digital inputs (the bump sensors). Note that due to the pull-up resistors, the switch is open if a logic '1' is read by the microcontroller at its corresponding bit.

Bit	Switch Connector
Byte 0 Bit 0	J2
Bit 1	J3
Bit 2	J4
Bit 3	J5
Bit 4	J6
Bit 5	J7
Bit 6	J8
Bit 7	J9
Byte 1 Bit 0	J10
Bit 1	J11
Bit 2	J12
Bit 3	J13
Bit 4	J14
Bit 5	J15
Bit 6	J16
Bit 7	J17

Table H.1: Digital input connector-to-bit correspondence.

H.3 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the Bump Sensor board.

H.3.1 Port Connector: J1

Details of connector J1, used to interface with the Proximity Detector HC11 board, is located in the Appendix A. Signal directions relative to this board are opposite to those documented there.

H.3.2 Bump Sensor Connectors: J2–J17

As mentioned previously, the input from each of these connectors is connected to a 4.7K pull-up resistor. As such, if a connected switch is open, the corresponding bit is high.

J2–J17: Bump Sensor Connectors			
Pin	Mnemonic	Direction	Description
1	BUMPx	I	Bump switch input.
2	GND	-	Signal ground.

Table H.2: Bump Sensor PCB connectors (J2–J17).

H.4 Design Limitations

None are known at this time.

H.5 Future Improvements

One of the wonderful strengths of SPI-compatible devices is the fact that it is possible to completely interface using so few signals (in this case three), and power and ground. Future versions of this board should make use of this strength by replacing the IDC ‘Port’ connector (J1) with a connector arrangement that is less expensive and simpler to work with.

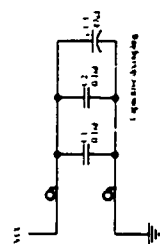
H.6 Schematic Diagrams and Board Layout

Figures H.2 through H.6 comprise the schematic and board layout of the Bump Sensor board.

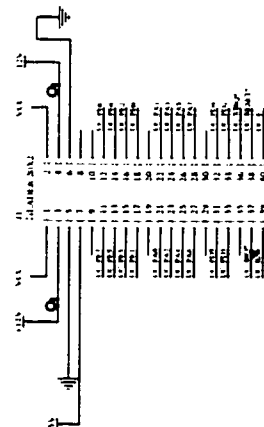
H.6.1 Part List

Bump Sensor Board Parts			
	Number Used	Part Type	Designators
1	2	0.1nF	C1 C2
2	3	4.7K (Bussed Resistor Pack (7))	RP1 RP2 RP3
3	1	47uF	C3
4	1	HEADER 20X2	J1
5	2	MC74HC165	U1 U2
6	16	Molex 2-Pin Header	J2-J17

Table H.3: Bump Sensor board parts list.



Capacitive decoupling.



MICU Board Port Connector

144
Power Routing/Off-board Connectors
Rev. A1 10m 2 Jul 1978 Revision 1.0
144 Distribution: 144MAY1978

Figure H.2: Bump Sensor PCB schematic (2 of 3).

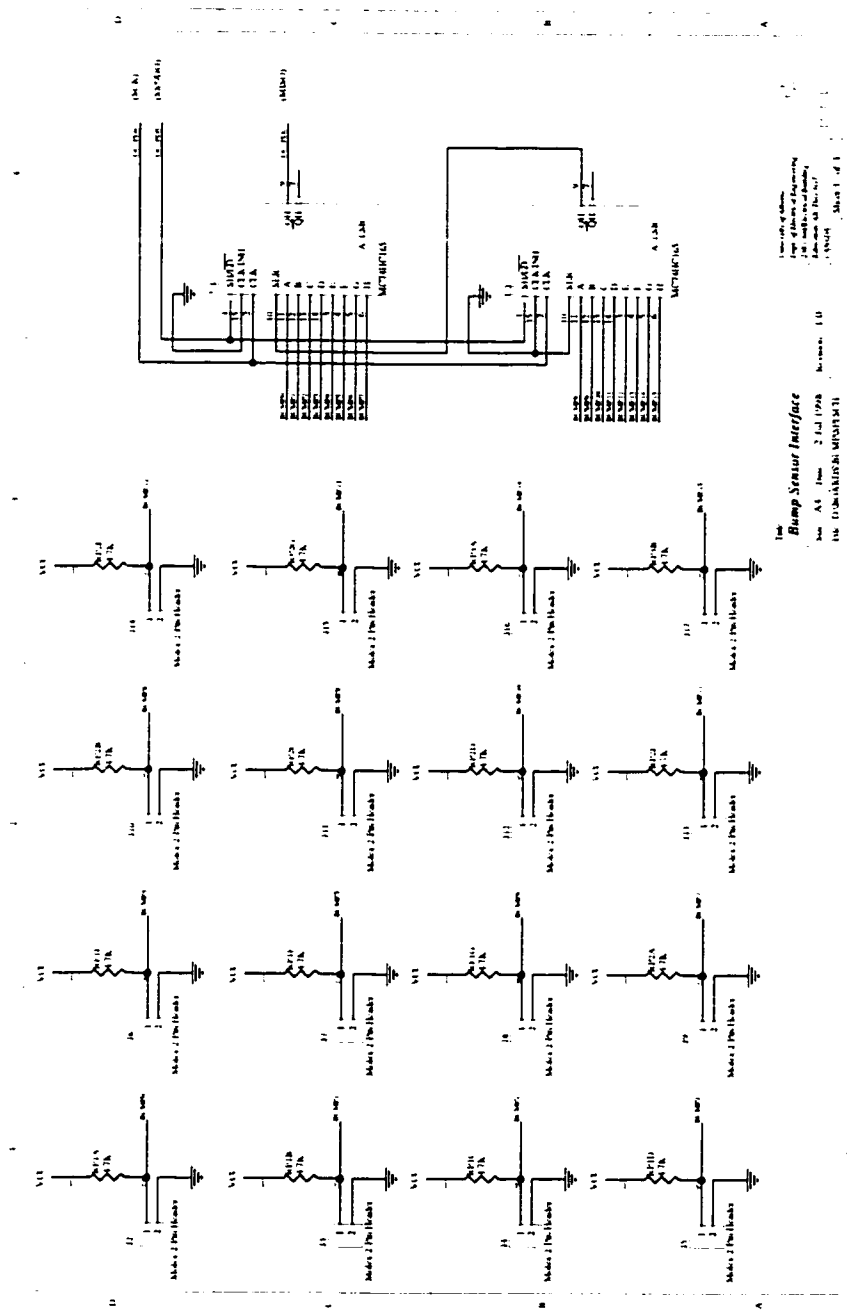


Figure H.3: Bump Sensor PCB schematic (3 of 3).

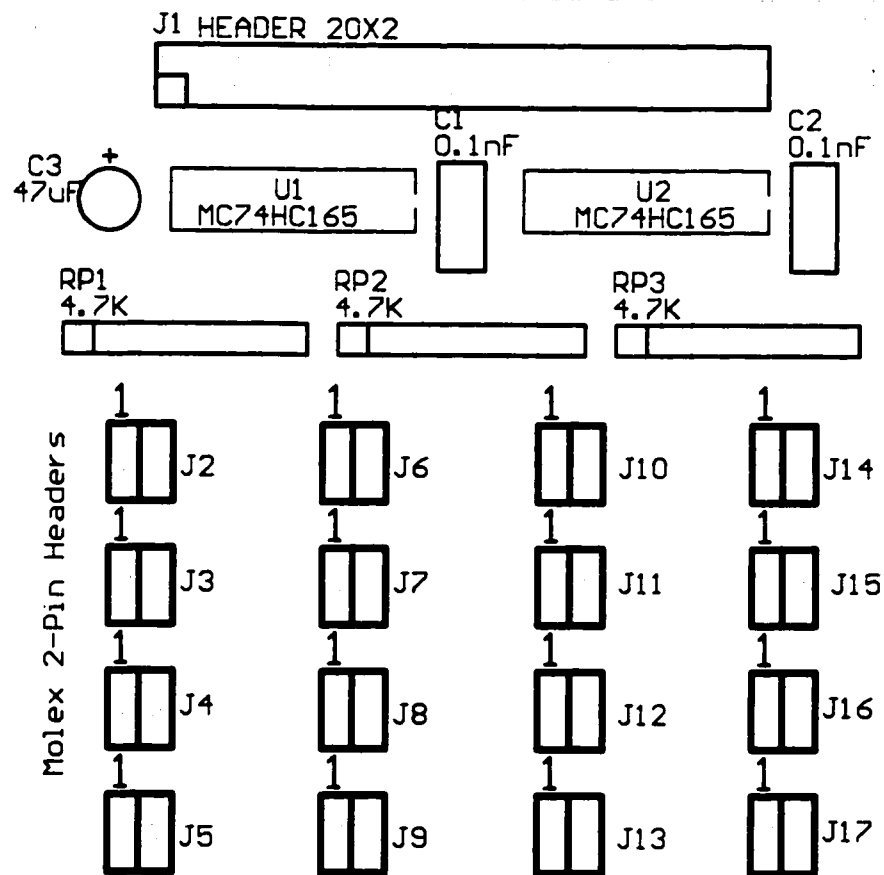


Figure H.4: Bump Sensor PCB part placement diagram.

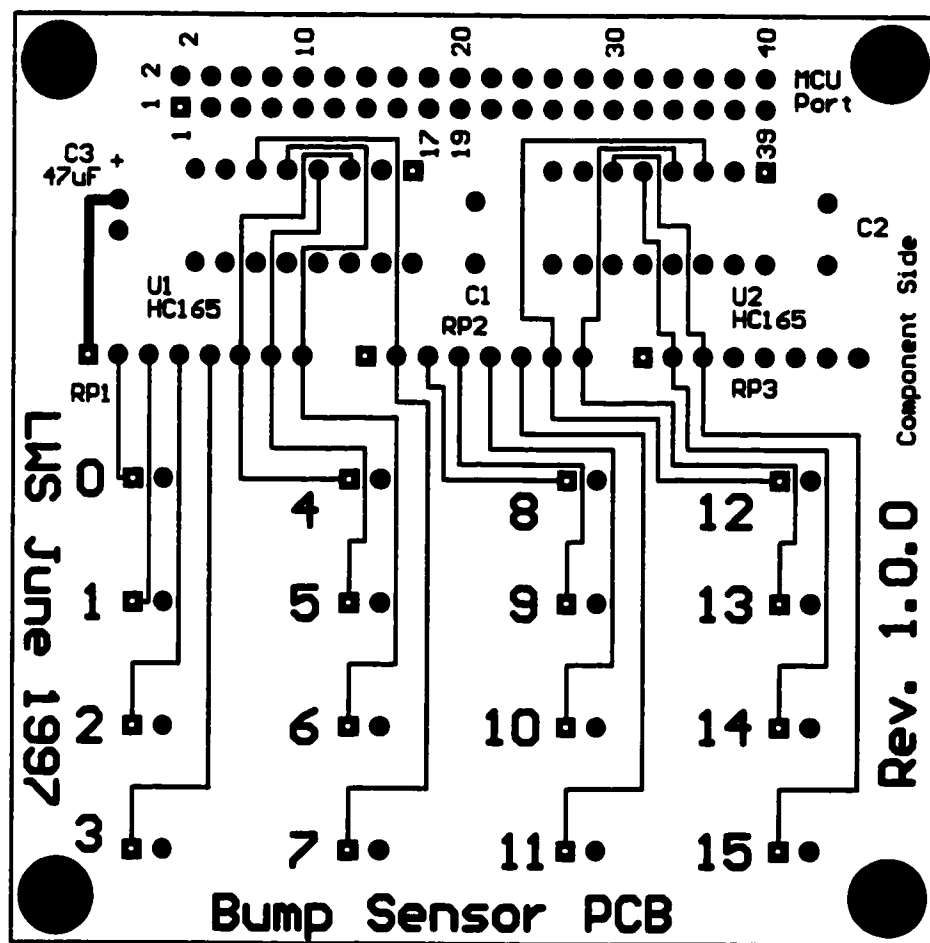


Figure H.5: Bump Sensor PCB top-layer copper foil pattern. Not to scale.

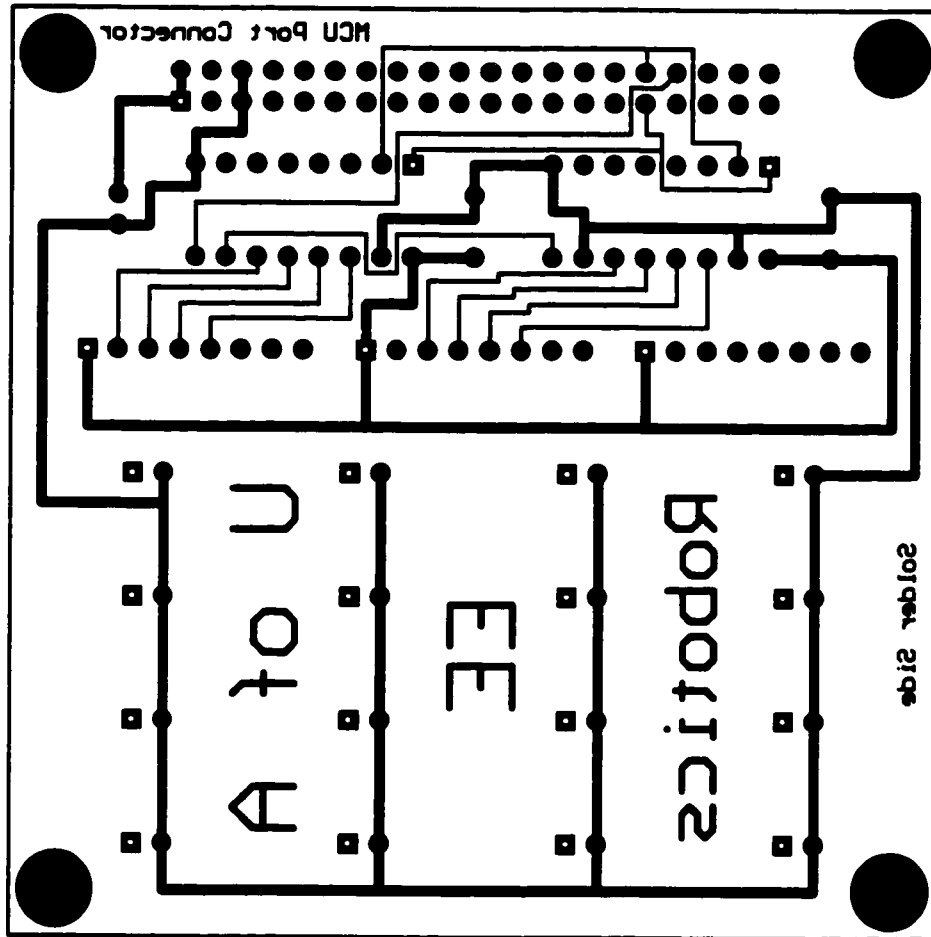


Figure H.6: Bump Sensor PCB bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix I

Sonar Module Interface PCB

I.1 Introduction

An outline of the operation, features, and user-settable options of the Sonar Module Interface board is given here. This board is used to interface to two Polaroid sonar modules and a stepping motor which is used to rotate the sonar transducers.

A block diagram of the board electronics is shown in Fig. 5.8. This board makes use of the connected MCU's SPI facility and two 'HC595 serial-to-parallel converters to derive digital outputs. These digital outputs are used to interface with Polaroid sonar modules as well as the stepping-motor driver. As with the navigation control hardware, the backbone of the motor driving circuitry is an SGS Thomson L298N dual H-bridge driver. In addition to the SPI-derived digital outputs, four signals from the connected MCU's Port A are interfaced to the sonar modules. Port A, associated with the HC11's timer facilities was used in order to simplify the software responsible for driving the modules.

Although it would have been possible to create the sonar modules themselves, Polaroid's modules are well-documented, reliable, and offer certain features which would be difficult to duplicate.

Figs. I.1 through I.3, contained later in this chapter, show the schematic of the Sonar Module Interface board. Fig. I.4 is a diagram showing the part placement. Figs. I.5 and I.6 show the top and bottom copper foil patterns, respectively.

I.2 Operation

The following table describes the position of the digital SPI outputs and their function.

SPI Output Bit	Mnemonic	Purpose
Byte 0 Bit 0	BLNK0	Sonar module 0 Blanking input.
Bit 1	BINH0	Sonar module 0 blanking inhibit.
Bit 2	-	Unused.
Bit 3	-	Unused.
Bit 4	BLNK1	Sonar module 1 Blanking input.
Bit 5	BINH1	Sonar module 1 blanking inhibit.
Bit 6	-	Unused.
Bit 7	-	Unused.
Byte 1 Bit 0	PHASEA	Stepper Phase A.
Bit 1	PHASEB	Stepper Phase B.
Bit 2	PHASEC	Stepper Phase C.
Bit 3	PHASED	Stepper Phase D.
Bit 4	STEP_ENABLE	Stepper enable.
Bit 5	LED0	Indicator LED.
Bit 6	LED1	Indicator LED.
Bit 7	LED2	Indicator LED.

Table I.1: Sonar Module Interface PCB SPI bit definitions.

PHASEA through PHASED provide the stepper driver with the appropriate signals to initiate rotation. The **STEP_ENABLE** signal can be used to disable the drivers to reduce current consumption. For instance, once the motor has been placed at the desired location, power can be removed by bringing **STEP_ENABLE** low. No current is drawn by the motor (up to 2 Amps per phase) when in this state. Both halves of the L298N dual H-bridge driver need to be used to interface to a stepper in the manner shown. Fly-back diodes are used to prevent voltage spikes from destroying the driver IC, or any other device connected to the supply.

Three general-purpose LED indicators are controlled by the signals LED0 through LED2. These indicators may be used for whatever purpose the developer desires.

The next byte of digital outputs is used to control the two sonar modules. The **BLNKx** signals are used to 'desensitize' the sonar modules to received echos while it is asserted. This is often used when there exist obstacles at a close range that we are not interested in. By asserting **BLNKx** at the time when this obstacle would be returning an echo, we can avoid assertion of a returned **ECHO** signal. The **BINHx** signals are used to **stop** internal blanking from occurring within the sonar modules. This may be desired, for instance, if obstacles are very near the module.

Typical use of the sonar modules through this interface board is as follows:

- (1) Initiate a 'ping' by asserting **INIT**.
- (2) Wait for a response via the module's assertion of the **ECHO** signal.

(3) Deassert INIT until another 'ping' is issued.

See the description of J2 and J3 for connection of the INIT and ECHO signals.

Modification of the Polaroid Sonar modules was required in order to decouple their supplies from the interface board logic supply. This was achieved by placing a large capacitance on the Polaroid module between the supply and ground. This was made necessary due to the large current-drain (up to 2 Amps) when issuing a series of ultrasonic pulses. For more information, refer to the Polaroid Sonar Module information.

I.3 Connector Descriptions

The following information is a list of connector pin-outs. All signal directions specified are with respect to the Sonar Module Interface board.

I.3.1 Stepper Motor Connector: J1

This 6-pin polarized header is used to connect to the stepper used to rotate the sonar transducers.

J1 – Stepper Motor Connector			
Pin	Mnemonic	Direction	Description
1	VMOT	Out	Motor power supply.
2	PhaseA	Out	Motor Phase A.
3	PhaseB	Out	Motor Phase B.
4	PhaseC	Out	Motor Phase C.
5	PhaseD	Out	Motor Phase D.
6	GND	-	Motor power ground return.

Table I.2: Sonar Module Interface PCB stepper motor connector J1 pinouts.

I.3.2 Polaroid Module Connector: J2, J3

These specialized Burndy connectors are used to connect to the Polaroid sonar modules via propriety flexible cables.

J2 – Polaroid Module 0 Connector			
Pin	Mnemonic	Direction	Description
1	GND	-	DC ground return.
2	BLNK0	Out	Blanking (echo mask) output.
3	N.C.	-	Not connected.
4	INIT0 (UC_PA6)	Out	Pulse initiate output
5	N.C.	-	Not connected.
6	N.C.	-	Not connected.
7	ECHO0 (UC_PA0)	In	Echo received input.
8	BINH0	Out	Blanking inhibit output.
9	Vcc	Out	+5V supply.

Table I.3: Sonar Module Interface PCB Polaroid Module 0 connector J2.

J3 – Polaroid Module 1 Connector			
Pin	Mnemonic	Direction	Description
1	GND	-	DC ground return.
2	BLNK1	Out	Blanking (echo mask) output.
3	N.C.	-	Not connected.
4	INIT1 (UC_PA5)	Out	Pulse initiate output
5	N.C.	-	Not connected.
6	N.C.	-	Not connected.
7	ECHO1 (UC_PA1)	In	Echo received input.
8	BINH1	Out	Blanking inhibit output.
9	Vcc	Out	+5V supply.

Table I.4: Sonar Module Interface PCB Polaroid Module 1 connector J3.

I.3.3 Motor Power Connector: J4

This connector is used to provide a decoupled supply for the stepper motor.

J4 – Motor Power Connector			
Pin	Mnemonic	Direction	Description
1	VMOT	In	Motor power supply.
2	GND	-	Ground return.

Table I.5: Sonar Module Interface PCB motor power connector J4 pinouts.

I.3.4 MCU Port Connector: J5

This connector is used to interface to the MCU responsible for the sonar subsystem. For information regarding the MCU Port Connector, refer to its description in Appendix A, the MC68HC11 Microcontroller PCB.

I.3.5 Limit-Switch Connector: J6

A Molex 2-pin polarized header is used to connect to a limit switch used for moving the sonar transducer assembly to a known position. The input signal is pulled-up through a 4.7K resistor and tied to an A/D converter port on the MCU.

J6 – Limit Switch Connector			
Pin	Mnemonic	Direction	Description
1	UC_PE0	In	Limit switch input. 0=closed.
2	GND	-	Signal ground return.

Table I.6: Sonar Module Interface PCB Limit-Switch connector J6 pinouts.

I.3.6 Photoresistor Connectors: J7–J10

Provision has been made for connection of 4 Cadmium Sulphide (CdS) photo-cells. These inputs can be used for arbitrary connection to the MCU A/D facility. The only caveat is that the inputs are pulled-up through 4.7K resistors which may not be required for some applications.

The input signals listed here are for J7 through J10, respectively.

J7 to J10 – Photoresistor Connectors			
Pin	Mnemonic	Direction	Description
1	UC_PE4,5,6,7	In	CdS Analog input.
2	GND	-	Signal ground return.

Table I.7: Sonar Module Interface PCB CdS Cell connector pinouts: J7–J10.

I.4 Future Improvements

As with other boards in this project that utilize the SPI, future revisions could make connection to the MCU simpler by using fewer connections. However, since various Port A and Port E signals are used, the simplicity gained here may be minimal.

I.5 Schematic Diagrams and Board Layout

Figures I.1 through I.6 comprise the schematic and board layout of the Sonar Module Interface board.

I.5.1 Part List

Sonar Module Interface Board Parts			
	Number Used	Part Type	Designators
1	3	0.1nF	C3 C4 C5
2	3	1K	R1 R2 R3
3	8	1N914	D1–D8
4	3	4.7K	R4 R5 RP1
5	3	47uF	C6 C7 C8
6	2	100nF	C1 C2
7	2	Burndy SLP9S-2	J2 J3
8	1	2-contact terminal block	J4
9	1	HEADER 20X2	J5
10	1	L298N	U1
11	3	LED	D9 D10 D11
12	2	MC74HC595A	U2 U3
(continued)			

Sonar Module Interface Board Parts (continued)			
	Number Used	Part Type	Designators
13	5	Molex 2-Pin Header	J6 J7 J8 J9 J10
14	1	Molex 6-Pin Header	J1

Table I.8: Sonar Module Interface board parts list.

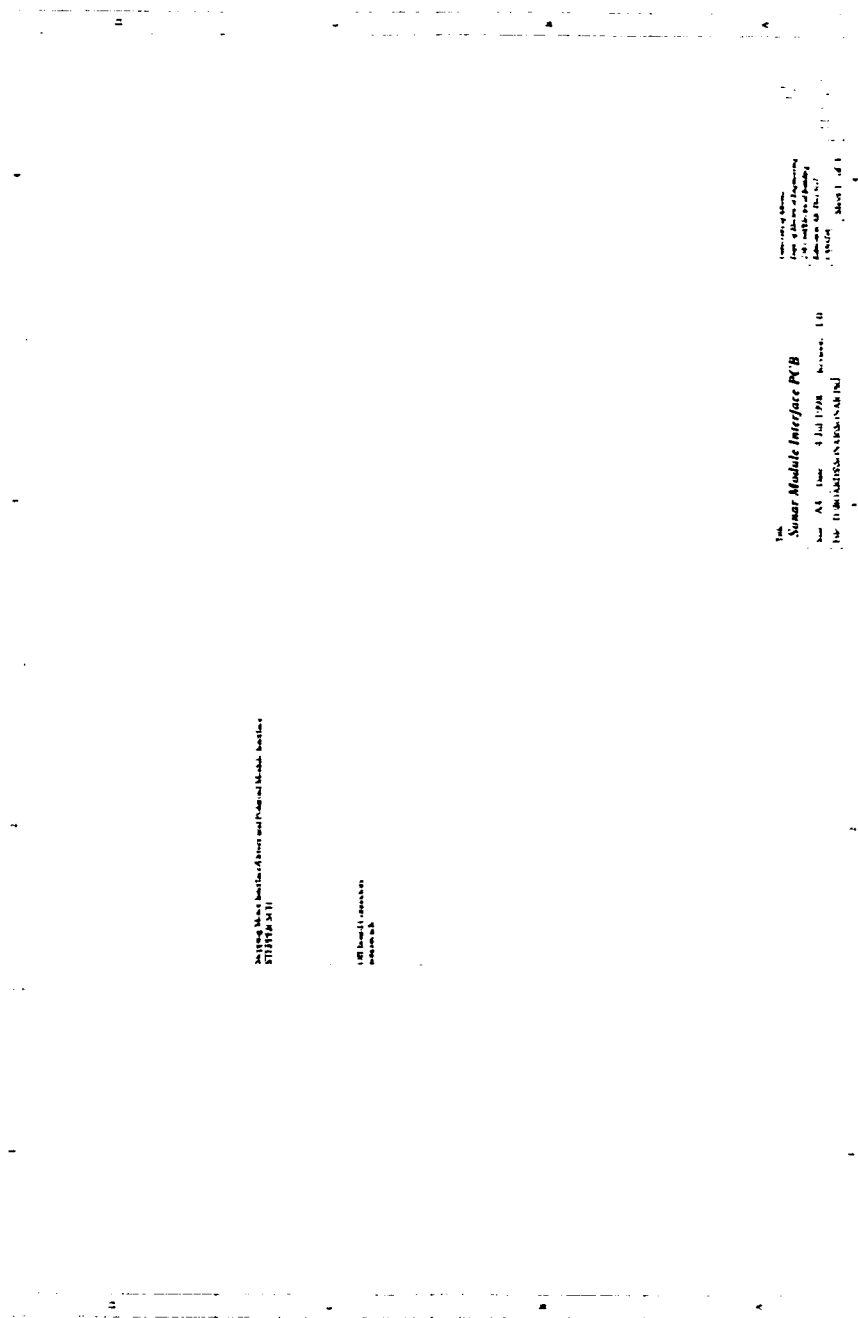


Figure I.1: Sonar Module Interface board schematic (1 of 3).

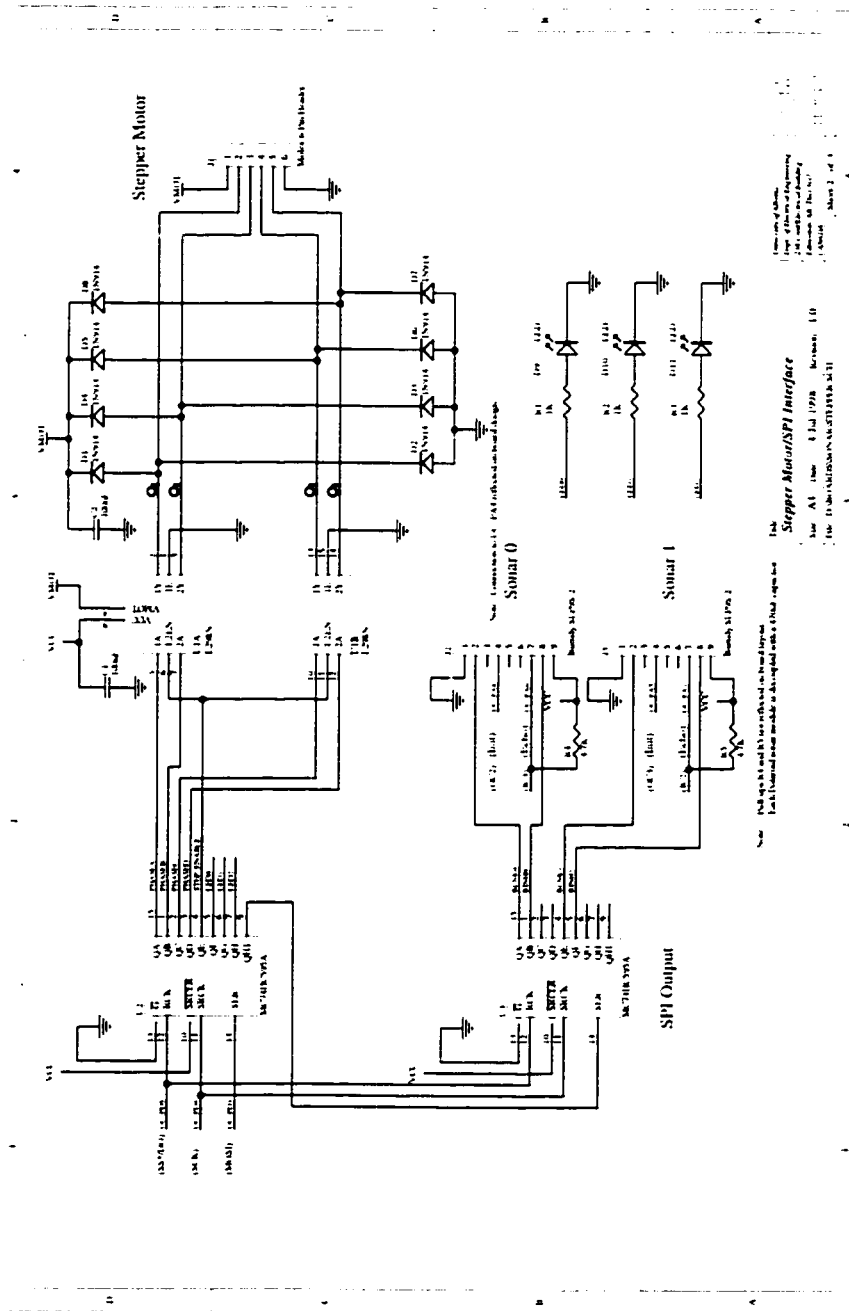


Figure I.2: Sonar Module Interface board schematic (2 of 3).

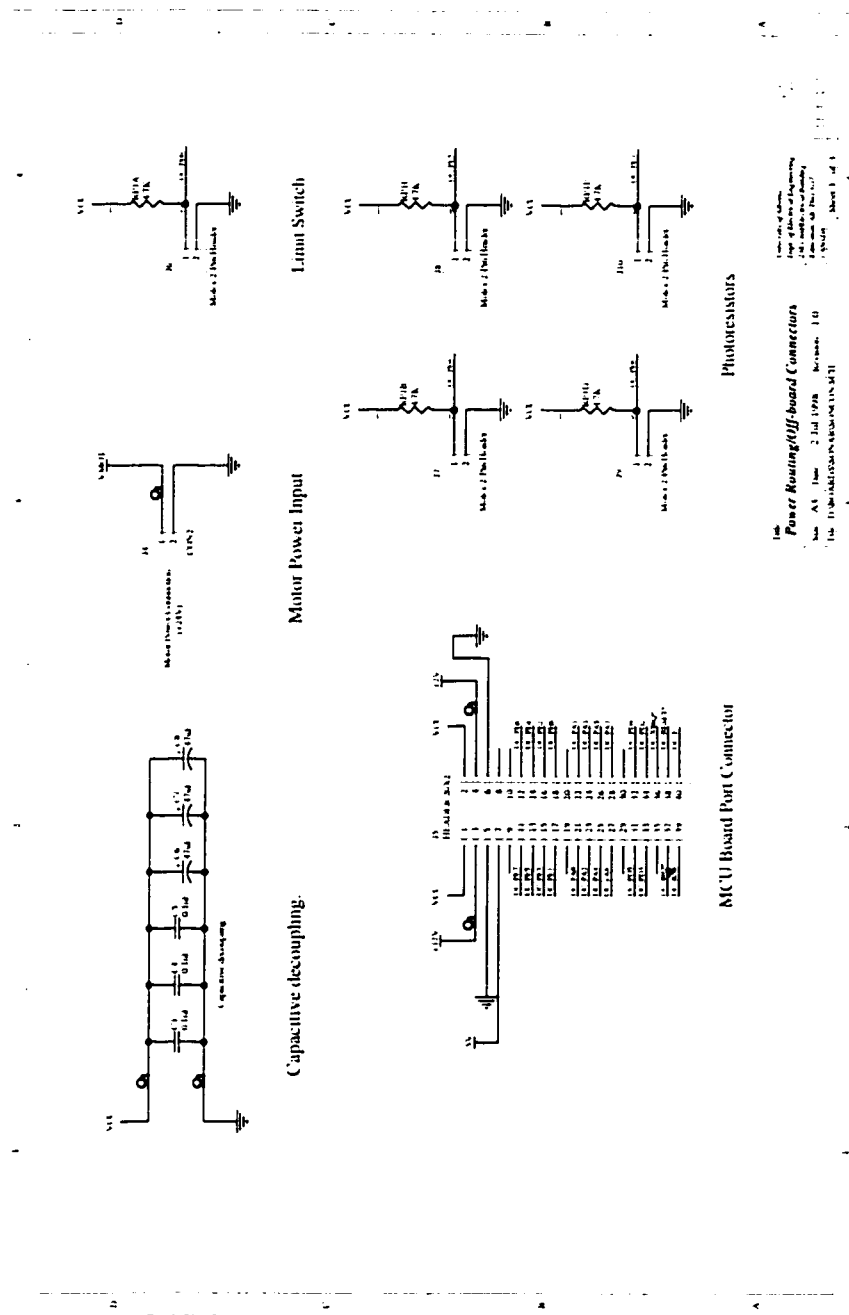


Figure I.3: Sonar Module Interface board schematic (3 of 3).

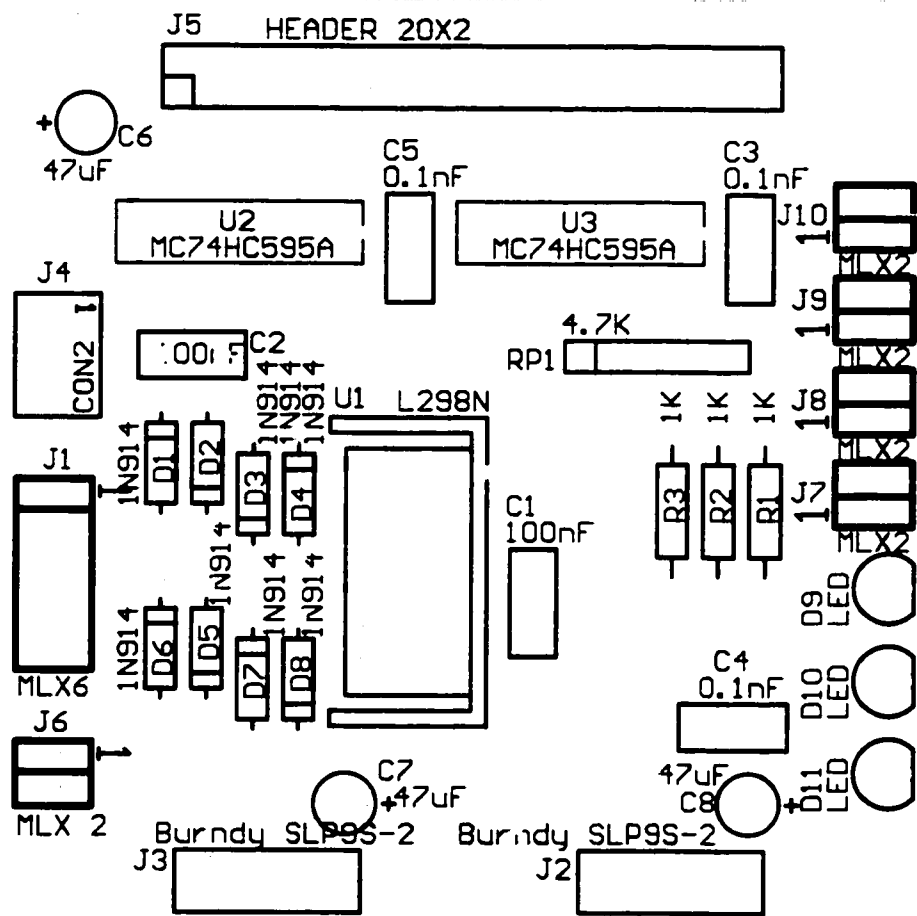


Figure I.4: Sonar Module Interface board part placement diagram.

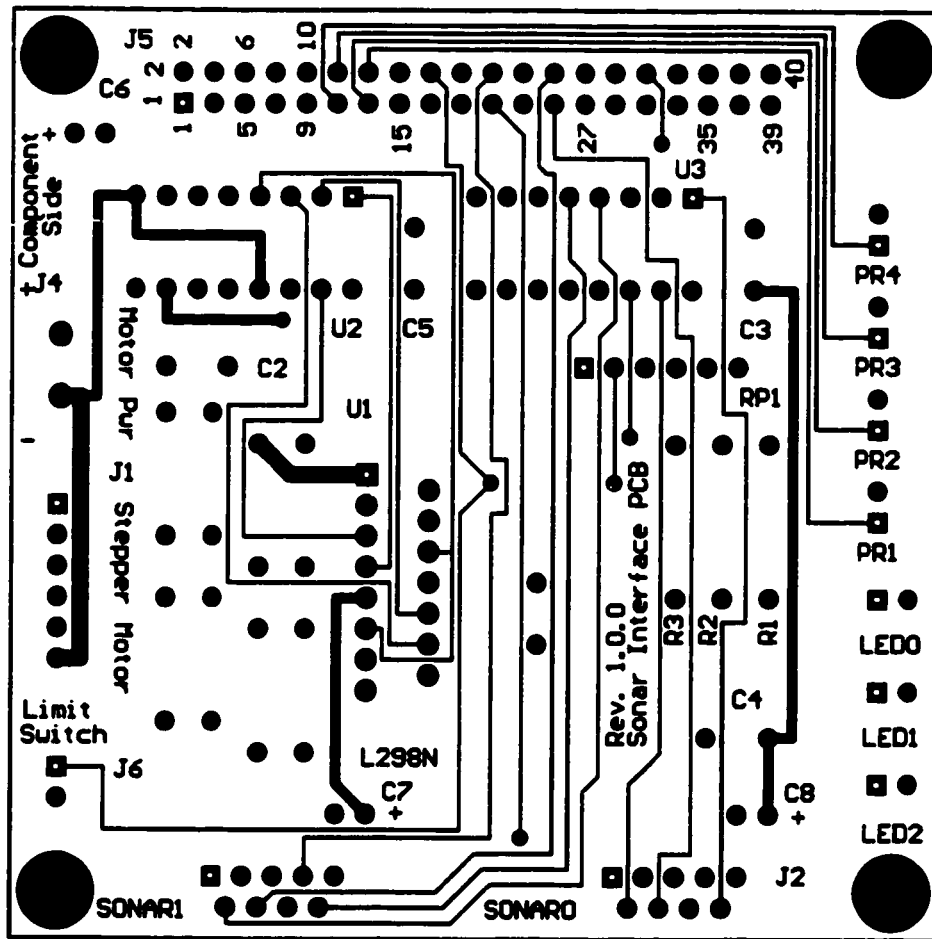


Figure I.5: Sonar Module Interface board top-layer copper foil pattern. Not to scale.

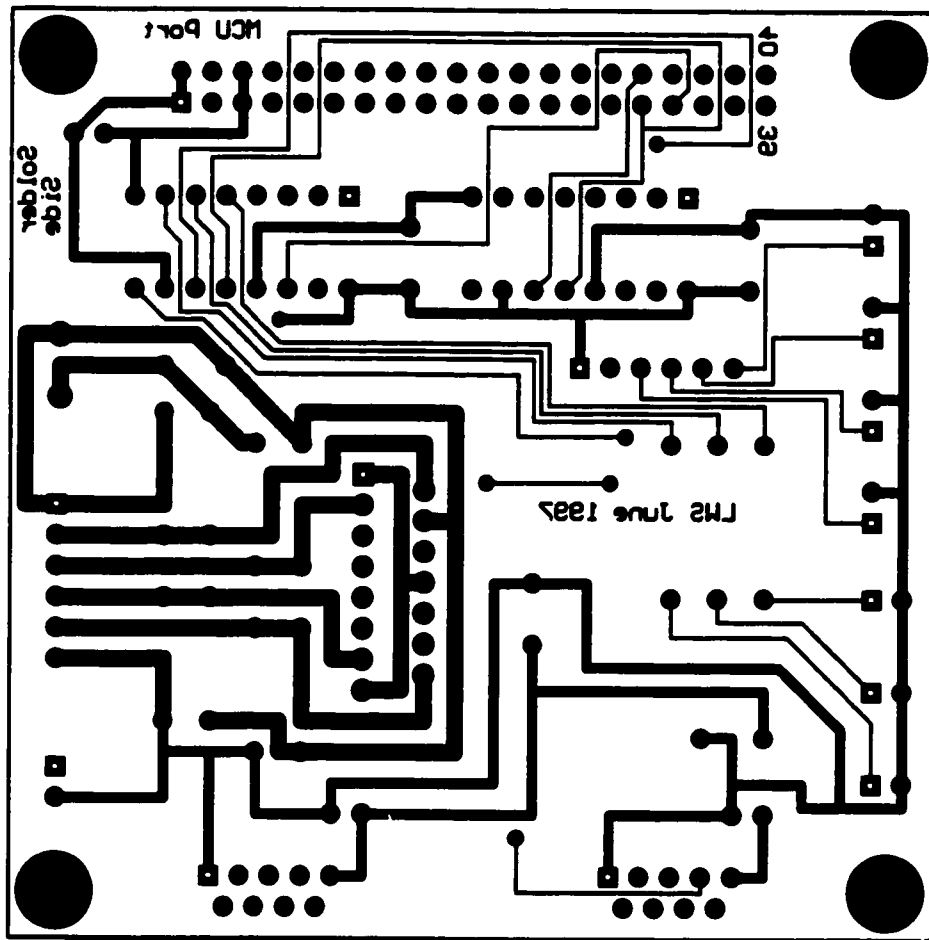


Figure I.6: Sonar Module Interface board bottom-layer copper foil pattern (through-board view). Not to scale.

Appendix J

Code Documentation

Many lines of code have been written for the teleoperated mobile-robot test bed. This appendix provides documentation for the procedures and functions. Actual code is found on the CDROM distributed with this document.

J.1 Microcontroller Code

The HC11 microcontroller code is written in near-ANSI compliant C. There are two shared code modules for messaging and shared-memory access, as well as the code specific to each of the three microcontrollers: Navigation, Proximity and SONAR. Each of these modules receives attention in a dedicated subsection.

J.1.1 File `message.c`

This module implements the messaging system allowing the microcontrollers to display information to the user at the control-station (via the workcell PC).

- `void msg2PC(char msg[])` Routine to send a string message to the PC via the shared memory. Sets the message flag. Transfers the string to the appropriate location (`TO_PC + MSG_STRING` in `memory.h`). Note that the string should be NULL terminated.
- `int digit2char(int number)` Convert a decimal digit to its ASCII equivalent.
- `void strcpy(char source[], char dest[], int number)` Copies `number` characters from `source` to `dest`. Places a NULL (0) in `dest[number]`.
- `int int2str(int number, char dest[], int start)` Converts `number` into a string placed in `dest`. The string **must** have enough room in it for to hold

the number (a maximum of 6 characters). Integer variable **start** gives the starting place at which the string equivalent of the integer is to be placed. The function returns the index of the array just after the last character that was added.

- **int float2str(float number, char dest[], int start)** Similar to **int2str**, but a floating-point number is converted into a string. Scientific notation with three decimal places is used. Thus, the character array **MUST** have at least **-x.xxxe-xx** (10) characters available.

J.1.2 File memory.c

This suite of routines are 'wrappers' to access shared memory. These are written so that software-implemented access semaphores can be included.

- **void write_shared_mem(int base, int offset, int data)** This routine writes a byte of data to the memory bank defined by **base**. The memory byte is stored in the location specified by **offset**. If a **base** of 0 is passed, then **all** three shared memory banks are written to.
- **int read_shared_mem(int base, int offset)** This routine returns a byte of data (**data**) read from the memory bank defined by (**base**). The specific location is determined from the (**offset**) value. This 'wrapper' is used to allow implementation of a shared memory access semaphore.
- **int write_shared_float(int base, int offset, float value)** Routine to write a floating-point value into shared memory. This cannot be achieved using pointers within Interactive-C.
- **float read_shared_float(int base, int offset)** Routine to read a floating-point value from shared memory.
- **void clear_shared_mem(int base)** Routine to clear a 4K block of shared memory starting at **base**.

J.1.3 File spi.c

This module contains code to initialize and utilize the Serial Peripheral Interface (SPI) on the Motorola 68HC11. All microcontrollers in the system have this interface.

- **int spi_size** This is a variable which contains the maximum byte-size of all serially-cascaded devices on the SPI, be they input or output devices.
- **int spi_input[]** This array of size **spi_size** contains the information read in from the SPI facility after a call to **spi()**.

- `int spi_output[]` This array of size `spi_size` contains the information read to be output to the SPI output devices. Actual output is performed through a call to `spi()`.
- `void init_spi()` Routine to prepare the SPI port as Master.
- `void spi()` This writes out the contents of the global array `spi_output` and fills the global array `spi_input` with the values read in. Each array will have `spi_size` entries, some of which may be garbage, depending on the hardware.

J.1.4 Navigation MCU Code

The Navigation MCU is responsible for controlling the workcell drive motors, and calculating the global position of the workcell based upon odometry data fed back from the motors. Three files are used to achieve this.

J.1.4.1 File `encoder.asm`

This file contains an interrupt service routine (ISR) which utilizes one of the HC11's Port A pins to count the number of encoder pulses presented to it. Interface to the C source, outlined shortly, is through a global variable, `left_clicks`. Once the ISR is running, this variable contains the number of encoder pulses read since the last time it was cleared.

J.1.4.2 File `motor.c`

- `float control_interval` This variable is the period (in seconds) of the PI control-loop execution.
- `float des_vel` A variable which contains the desired speed of the workcell in encoder clicks per interval.
- `float des_bias` Desired bias in encoder clicks per interval.
- `float integral` The integral of the velocity error.
- `float k_i` Control-loop integral (I) gain.
- `float k_p` Control-loop proportional (P) gain.
- `float power[]` Array of motor powers (given as Pulse-Width-Modulation percentage duty cycle).
- `float count[]` Array of encoder counts of the two motors.

- `int nav_pid` Process identification of the directive currently being executed.
- `int init_motors()` Initialize the HC11 for Port A pulse-width modulation.
- `void stop()` Stop both drive motors.
- `void motor(int index, float vel)` Establishes a speed of `vel` (-100% to 100%) of the motor given by `index` (0 is left, 1 is right).
- `void drive(float trans_vel, float rot_vel)` Use drive to control motion of the robot. A positive `rot_vel` makes the robot turn left.
- `void init_velocity(void)` Initialize the pulse accumulator, and PA0 (Input Capture 3) interrupts.
- `float get_sign(float arg)` Routine to return the sign of an argument. Returns +1.0 if the argument was positive (or zero). -1.0 if negative.
- `float get_right_vel(void)` Get and return the number of pulses accumulated by the accumulator (PA7) and return. Clears the count after reading.
- `float get_left_vel(void)` Get and return the number of pulses counted by the interrupt routine associated with PA0. Clears the count after reading.
- `float limit_range(float val, float low, float high)` Function to limit the range of a value.
- `float integrate(float left_vel, float right_vel, float bias)` Simple rectangular numerical integral technique.
- `void alter_power(float error, int motor_index)` Routine to adjust the motor powers after control loop calculations have been performed.
- `void speed_control(void)` Procedure to be called as a process. Implements the PI control algorithm of the drive motors.
- `void set_velocity(float speed_ms, float bias_ms)` Procedure to set the speed setpoint and the wheel bias. Speed arguments are to be in metres per second.
- `int start_speed_control(void)` Starts the motor control algorithm.
- `void directive_interp()` Procedure to interpret navigation commands coming from shared memory. This is to be called as a process.

- **void translate(int direction)** Routine to navigate either forward or backward. This is bang-bang control over the internal slaved-PI velocity control loop. This function is called as a process and is controlled by `directive_interp()`.
- **void rotate(int direction)** Routine to navigate either right or left. This is bang-bang control over the internal slaved-PI velocity control loop. This function is called as a process and is controlled by `directive_interp()`.
- **void stop_pi()** Routine to stop the workcell while under PI control. The PI process is first killed, and the motors are stopped. After a delay, the coordinates are updated, and the integral is cleared. Then the PI process is started again so navigation can continue.

J.1.4.3 File nav.c

- **float Xcoord** The global workcell position in the x-direction (metres).
- **float Ycoord = 0.0** The global workcell position in the y-direction (metres).
- **float Bearing** Bearing of the workcell with respect to the global coordinate frame (radians).
- **int pipid** PI algorithm process ID.
- **void main(void)** The program which runs out of reset.
- **void go(void)** Main calling routine for navigation control.
- **void update_coords(void)** Routine to update the workcell according to the workcell odometry (dead-reckoning data). This is performed via calls to `update_cartesian()`, which updates the X and Y coordinates of the workcell, and `update_bearing()`, which updates the workcell's orientation. Note that this routine should be called at a reasonably fast rate to avoid errors potentially created by trans-rotational movement. It is assumed here (and made as true as possible by the navigation directives) that movement is EITHER translational or rotational; not both. The distance traveled is calculated by averaging the distance traveled by each wheel. This should help eliminate transient errors present in the slaved-PI motor speed control algorithm.
- **void update_coords_periodic()** Wrapper routine to call `update_coords()` periodically. The update rate is grabbed from shared memory. This routine is intended to be called as a process.

- `void update_cartesian(float distance_m)` Routine to update the X and Y location of the workcell (stored in memory shared with the PC).
- `void update_bearing(float arc_length_m)` Updates the workcell's orientation in shared memory based upon the distance traveled by the wheels (in the opposite direction).

J.1.5 Proximity Detector MCU Code

Two files are used to poll the state of the workcell bump and IR sensors. The first, `sensor.h`, contains preprocessor definitions regarding the position and array assignments of individual sensors. The other, `prox.c`, detailed here, is responsible for reading sensor states and relating them to the workcell PC via shared memory.

J.1.5.1 File `prox.c`

- `int ir_threshold[]` An array of 8-bit thresholds for the A/D levels read from the IR detectors.
- `int ir_levels[]` An array of the 8-bit A/D levels of the IR detectors.
- `void main(void)` The program which runs when the MCU comes out of reset.
- `void go(void)` Runs the routines. This is the main program.
- `void poll_sensors(void)` Infinite loop to periodically update the bump sensor data in the `spi_input` array, and the IR sensor information in shared memory.
- `void init_porta(void)` Establish Port A as digital-only mode (no timer functions). Port A is used to implement the analog MUX switching lines.
- `void init_ir_thresholds(void)` Routine to establish the default ir detector analog level thresholds. The default threshold is 128 of 255.
- `void set_amux_lines(int bank)` Set the port A lines PA5, PA6 to select which of the four banks of IR sensors are connected to Port E.

J.1.6 SONAR MCU Code

The operation and movement of the SONAR sensor array is done through three files, the contents of which are detailed here.

J.1.6.1 File son.c

This file is the main file which brings together the contents of `stepper.c` and `polaroid.c` to perform the sensor sweep.

- `void main()` The program which runs out of reset. This is simply a wrapper for function `go()`.
- `void go()` The main calling routine for the sonar board routines.
- `void sweep(void)` Routine to step the two SONAR transducers through 180 degrees each, gathering echo data every 1.8 degrees.

J.1.6.2 File polaroid.c

This module contains all the functions necessary to initialize and use the Polaroid sonar modules.

- `void init_sonar(void)` Initializes the SPI output port to set BLNK and BINH of both sonar modules to 0.
- `float ping0(void)` A ping routine which relies on the HC11 internal timer counter. Pings only sonar unit 0. The distance (in metres) to the obstacle which provided the echo is returned.
- `float ping1(void)` A ping routine which relies on the HC11 internal timer counter. Pings only sonar unit 1. The distance (in metres) to the obstacle which provided the echo is returned.

J.1.6.3 File stepper.c

This module is used to provide the software interface to the stepper motor atop which the SONAR array is mounted.

- `void step_motor(int direction)` Routine to pulse the motor in the specified direction. Direction of 1 is clockwise. -1 is counter-clockwise.
- `void enable_stepper(int value)` If passed a non-zero argument, this routine uses the SPI port to enable the stepper motor driver circuitry. If passed a zero argument, the driver circuitry is disabled. This is generally used in order to reduce current-draw of the stepper actuator.
- `void home_stepper()` Routine to rotate the motor until the limit switch (connected to PE0) is closed. The limit switch signal is active-low.

J.2 Server (Workcell Control Engine) Code

The server code is written in the Tool Command Language (TCL), but also includes C mapping and parallel port I/O routines. The C routines are called from TCL by compiling them to use TCL's C API, a process automated by the Simplified Wrapper Interface Generator (SWIG).

J.2.1 ANSI C Mapping Code

The mapping routines are written in ANSI-C. The prototypes presented here are identical to those used for map manipulation in the client side of the system. There are two files in the ANSI-C portion of the mapping routines: map.h and map.c. The first contains a definition of the map data type. These files are compiled using the 'Simplified Wrapper Interface Generator' (SWIG), and an interface file (map.i) which defines the functions in map.c that are to be callable from TCL.

J.2.1.1 File map.h

This file contains the definition of the map structure.

```
typedef struct map {
    double resolution; /* Map resolution in metres. */
    double xmin; /* Minimum x coordinate in metres. */
    double xmax; /* Maximum x coordinate in metres. */
    double ymin; /* Minimum y coordinate in metres. */
    double ymax; /* Maximum y coordinate in metres. */
    int x_bytes; /* Number of bytes in the x-direction. */
    int y_bytes; /* Number of bytes in the y-direction. */
    unsigned char *data; /* Actual map data. */
} Map;
```

This structure is used to encapsulate all information that a map contains. The individual pieces of map data are unsigned characters (8 bits) and are used to store 8 individual map 'layers', as described in Chapter 6.

J.2.1.2 File map.c

This file contains a suite of routines used to create and manipulate the Map data type.

- **Map map1, map2** Variables of the Map data type. Two are used: map1 is the 'current' map, and map2 is a 'modified' map.
- **Map current = &map1** A pointer to the 'current' map.

- `Map modified = &map2` A pointer to the 'modified' map.
- `Map *get_map_current_pointer()` Returns the pointer to the current map.
- `Map *get_map_modified_pointer()` Returns the pointer to the modified map.
- `double get_xmax(Map *pmap)` Wrapper which returns the maximum x-coordinate of the map.
- `void set_xmax(double x, Map *pmap)` Wrapper which sets the maximum x-coordinate of the map.
- `double get_ymax(Map *pmap)` Wrapper which returns the maximum y-coordinate of the map.
- `void set_ymax(double y, Map *pmap)` Wrapper which sets the maximum y-coordinate of the map.
- `double get_xmin(Map *pmap)` Wrapper which returns the minimum x-coordinate of the map.
- `void set_xmin(double x, Map *pmap)` Wrapper which sets the minimum x-coordinate of the map.
- `double get_ymin(Map *pmap)` Wrapper which returns the minimum y-coordinate of the map.
- `void set_ymin(double y, Map *pmap)` Wrapper which sets the minimum y-coordinate of the map.
- `double get_resolution(Map *pmap)` Wrapper which returns the resolution of the map.
- `void set_resolution(double res, Map *pmap)` Wrapper which sets the resolution of the map.
- `int allocate_map_data(Map *pmap)` Allocates memory for map data in the map structure specified by the Map pointer. Data regarding range and resolution should already be set in the map structure. The size of the arrays should be set in `x_bytes`, `y_bytes`. Returns 1 if the allocation is successful. 0 if not.
- `int free_map_data(Map *pmap)` Frees the memory associated with 'data' field in the map structure specified by the Map pointer. This function is no longer required since implementation of `realloc` over `malloc`. Nonetheless, it is left in for the purpose of debugging.

- `int quantize_x(double x_m, Map *pmap)` Returns the array index of the cell that contains the x coordinate specified by `x_m`. Returns -1 if the specified `x_m` coordinate is out of range of the specified map.
- `double find_x_centre(double x_m, Map *pmap)` Returns the x coordinate (metres) of the centre of the cell that contains the coordinate. `x_m`.
- `int quantize_y(double y_m, Map *pmap)` Returns the array index of the cell that contains the y coordinate specified by `y_m`. If the `y_m` coordinate is out of range of the specified map, then -1 is returned.
- `double find_y_centre(double y_m, Map *pmap)` Returns the y coordinate (metres) of the centre of the cell that contains the coordinate. `y_m`.
- `int calculate_size(Map *pmap)` Calculates the array size required to store map data ranging from `xmin` to `xmax`, `ymin` to `ymax` at resolution. with the origin at the **centre** of a cell. All this information should be previously established in the Map structure pointed to. If the origin is not within the range, then the minimum coordinate is assumed to define the lower **EDGE** of one of the grid cells. The ranges are adjusted to coincide with grid cell edges.
- `int get_index(int x, int y, Map *pmap)` A routine to calculate the 1-D array size from the 2-D array coordinates for the map pointed to. Returns -1 if the specified x and y index coordinates are out of range for the specified map.
- `void set_obstacle(double x_m, double y_m, int layer, Map *pmap)` Sets the appropriate layer bit in the map cell which contains the location (specified in metres). The layer numbering is detailed in the TCL calling script.
- `void set_obstacle_byte(double x_m, double y_m, int value, Map *pmap)` Sets the entire byte in the map cell which contains the location (specified in metres).
- `void clear_obstacle(double x_m, double y_m, int layer, Map *pmap)` Clears the appropriate layer bit in the map cell which contains the location (specified in metres).
- `void clear_obstacle_byte(double x_m, double y_m, Map *pmap)` Clears the entire byte in the map cell which contains the location (specified in metres).
- `unsigned char get_obstacle_byte(double x_m, double y_m, Map *pmap)` Returns the entire byte associated with the specified location: 1 is User, 2 is Sonar, 3 is User **and** Sonar, etc.

- `unsigned char get_obstacle(double x_m, double y_m, int layer, Map *pmap)` Returns 0 if there is no obstacle present at the specified location on the specified layer.
- `int redo_map(double xmin, double xmax, double ymin, double ymax, double resolution)` Resizes or changes resolution of the current map according to the new information passed to it. No obstacle data is lost in the transformation. Returns 1 if successful. 0 if not. This function assumes that the current map is sized correctly (that the function `calculate_size` has already been called for it).
- `int copy_map(Map *source, Map *destination)` Copies a map structure, and all of its data from source to destination. The source is left untouched. Returns 1 if successful. 0 if not.
- `int new_map(double xmin, double xmax, double ymin, double ymax, double resolution)` Deletes the current map and allocates resources that conform to the newly specified parameters. Returns 1 if successful. 0 if not.
- `void clear_map_array(Map *pmap)` Function to clear each element of the specified map.
- `int read_map_file(char *filename)` Reads a map from binary file named by filename. Returns non-zero if successful, zero if not. Map data is placed in the current map structure.
- `int write_map_file(char *filename)` Writes a map to binary file named by filename. Returns non-zero if successful, zero if not. Map data is taken from the current map structure.
- `void print_map_info(Map *pmap)` Debugging routine to print contents of the map structure pointed to by pmap.
- `void print_map_array(Map *pmap)` Routine to display a rough outline of what the map looks like.

J.2.2 TCL Mapping Code

File `map.tcl` contains code used to interface to the C representation of the map. The routines shown here are almost identical to those used to control the Mapping Module UI on the control station.

J.2.2.1 File `map.tcl`

- `map(ptr)` Pointer to C map structure.
- `map(resolution)` Map resolution in metres.
- `map(xmin)` Minimum map x-coordinate (metres).
- `map(xmax)` Maximum map x-coordinate (metres).
- `map(ymin)` Minimum map y-coordinate (metres).
- `map(ymax)` Maximum map y-coordinate (metres).
- `map(refresh)` Map refresh rate (milliseconds)
- `map(xintervals)` Grid size in the x-direction.
- `map(yintervals)` Grid size in the y-direction.
- `proc create_new_map { }` This procedure is responsible for creating a new map using settings passed through the global `map()` array. A default map is created when the server is started.
- `proc move_workcell { x_m y_m bearing }` Move the workcell objects to match the specified coordinates and bearing.
- `proc synchronize_maps { {workcellx 0.0} {workcelly 0.0} }` Procedure to synchronize the C and remote (client) maps. The optional workcell coordinate arguments default to 0.
- `proc clear_map { }` A procedure to clear all obstacles from a map. The map size and resolution are retained.
- `proc upload_map { }` Routine to receive the map from the client.
- `proc download_map { }` Routine to send all map information to the client: size, resolution, obstacle information, etc.
- `proc layer2number { layer }` A procedure which converts a named layer into its layer number used in the C representation of the map.
- `proc create_obstacle {metre_coordinates tag}` Creates an 'obstacle' at the grid specified by the dual-value `metre_coordinates` argument. Transmits an update to the workcell if this obstacle on this layer has not been set. The layer tag specified may any of those described in procedure `layer2number`.

J.2.3 ANSI C Parallel Port I/O Code

The routines here are used to interface with the microcontrollers and shared-memory. The file, `ports.c` is written in such a way that it can be compiled to simulate connection to the microcontrollers and shared memory (useful for debugging), run as a stand-alone application with menu interface (compiled as `mcuset`), or interfaced with TCL via SWIG.

J.2.3.1 File `ports.c`

- `void main()` Main function to loop stand-alone application until done.
- `void static_screen(void)` Display the static command status screen for the debugging interface.
- `void reset_mcu()` Resets a specific MCU (menu-driven operation).
- `void dwnld_mcu(void)` Places a specific MCU into download mode (menu-driven operation).
- `void active_mcu(void)` Makes a specific MCU active (menu-driven operation).
- `void read_mem(void)` Reads memory shared with a specific MCU (menu-driven operation).
- `void write_mem(void)` Writes memory shared with a specific MCU (menu-driven operation).
- `void write_mem_float(void)` Writes a float to memory shared with a specific MCU (menu-driven operation).
- `char getcmd(void)` Reads a single letter from stdin and returns it.
- `int gethex(int num_chars)` Routine to limit entry to values in the hex range. It is passed the number of digits to convert before exiting.
- `void initialize_parallel_card()` Sets starting values for all the registers on the parallel port expansion PCB.
- `void synchronize_card(void)` This routine ensures that the data in the Port structure. 'card', is synchronized with what is actually present in the hardware. The OE register is the last to be written to minimize transient data effects.
- `void open_port()` Get access to the printer port. Requires root permission.

- `void close_port()` Terminate access to the printer port.
- `void write_byte(unsigned char block, unsigned char value)` Write a byte to one of the blocks (registers) on the parallel expansion board.
- `unsigned char read_byte(void)` Read a byte from the so-called 'data bus'. This must happen in two stages to read individual nibbles. Includes a patch for the REV1.0 board out-of order low nibble.
- `unsigned char read_nibble()` Function which reads the nibble that is presently accessible to LPT1:. Returns the nibble in the 4MSBs of the unsigned char.
- `unsigned char read_shared_mem(int mcu, unsigned short location)` This function returns the byte read from the dual-port static RAM connected to microcontroller mcu at the address specified by location. The value is read several times (as defined in by `READ_VERIFICATION_COUNT`). If the number not the same for all these instances, then reading continues until it is.
- `void write_shared_mem(int mcu, unsigned short location, unsigned char value)` This function writes the byte value to the dual-port static RAM connected to microcontroller mcu at the address specified. Calls itself recursively until the data is successfully written.
- `int get_oe(int bit)` Routine that returns the value of the OE bit specified. If `bit` is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_sd(int bit)` Routine that returns the value of the SD bit specified. If `bit` is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_sa_low(int bit)` Routine that returns the value of the SA_LOW bit specified. If `bit` is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_sa_high(int bit)` Routine that returns the value of the SA_HIGH bit specified. If `bit` is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_ctrl(int bit)` Routine that returns the value of the CTRL bit specified. If `bit` is 8, then the entire byte value is returned rather than just the specified bit (0-7).

- `int get_uc1(int bit)` Routine that returns the value of the UC1 bit specified. If bit is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_uc2(int bit)` Routine that returns the value of the UC2 bit specified. If bit is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_uc3(int bit)` Routine that returns the value of the UC3 bit specified. If bit is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `int get_pwr(int bit)` Routine that returns the value of the PWR bit specified. If bit is 8, then the entire byte value is returned rather than just the specified bit (0-7).
- `void set_uc1(int bit, int value)` Routine to set the specified bit (0-7) of the UC1 register. If bit is 8, then the entire byte value is assumed to be passed in value.
- `void set_uc2(int bit, int value)` Routine to set the specified bit (0-7) of the UC2 register. If bit is 8, then the entire byte value is assumed to be passed in value.
- `void set_uc3(int bit, int value)` Routine to set the specified bit (0-7) of the UC3 register. If bit is 8, then the entire byte value is assumed to be passed in value.
- `void set_pwr(int bit, int value)` Routine to set the specified bit (0-7) of the PWR register. If bit is 8, then the entire byte value is assumed to be passed in value.
- `void write_float(int mcu, unsigned short location, double value)` Writes a floating point number to the specified location in shared memory.
- `void read_float(int mcu, unsigned short location)` Reads 4 bytes from shared memory and converts them to a floating-point number. Places the floating-point value in the global `float_result` which must be read after this function is called.
- `void write_string(int mcu, unsigned short location, char *string)` Writes a NULL-terminated string to shared memory starting at the specified location. It is up to the user to ensure that the string is not too long.
- `char *read_string(int mcu, unsigned short location)` Reads a NULL-terminated string from shared memory starting at the specified location.

J.2.4 TCL Shared Memory Interface Code

This file contains the routines which interface with the microcontrollers and the shared memory. It includes routines used for polling of sensor states, reading messages originating from the MCUs, and issuing movement directives to the Navigation MCU.

J.2.4.1 File `memif.tcl`

- `proc GO { direction }` Procedure to initiate movement in a specific direction. Argument `direction` can be: `forward`, `backward`, `right`, `left`, or `stop`.
- `proc establish_setpoint { type value }` Procedure to set setpoint type `speedsetpoint`, `distancesetpoint`, or `anglesetpoint` to `value`.
- `proc check_for_message { mcu }` Procedure to check for a message from the specified `mcu` (`nav`, `prox`, `son`, or `all`). If a message is waiting, then receipt is acknowledged (to the MCU) and the message is sent via the `message` socket. If `all` is specified, this procedure runs recursively at a period specified in the array member `scheduler(message)`.
- `proc init_shared_memory { }` Procedure to initialize shared memory to 0.
- `proc bump_update { }` Procedure to read the status of the active-low bump switches. Updates the map with the barrier info. If the status of a specific switch is different than that previously read, the control base is informed of the change through the sensor socket so the sensor display update can be set. This is recursive procedure which calls itself using the `after` command and the time specified in `scheduler(bump)`. Note that the active-low nature of the bump signals is transformed within the microcontroller C routine to be active high.
- `proc Sensor2Coordinates {vector}` Procedure to take the vector of the sensor relative to the workcell coordinate frame and transform them into global x and y metre coordinates, returned as a list.
- `proc position_update { }` Procedure to read the coordinates (and bearing) from shared memory and return them to the control station. This is a recursive procedure that sends the data at a rate specified in `scheduler(position)`. Recall that function `read_float` (written in C), returns the float value through a global variable, `float_result`. This was the only way that the routine would work correctly.

- `proc Search {searchcoordx searchcoordy}` Procedure to search out the coordinates specified. Takes control over the workcell using the "GO" command. Global variable `searchstage` is used to determine what "level" the search algorithm is working in.
- `proc ir_update { }` Procedure to read the status of the near-IR pairs, apply a threshold to those values, and determines which sensors are activated. Updates the map with the barrier info. This is recursive procedure which calls itself using the `after` command and the time specified in `scheduler(bump)`.
- `proc sonar_update { }` Procedure to read the shared memory from the Sonar MCU. Reads obstacle information, translates it into global coordinates and, if necessary, updates the map through the mapper socket. Also sends a series of 'ping dots' to the workcell so the user can interpret the map, if desired.

J.2.5 Server Module Code

This software module contains routines which open and maintain TCP/IP socket connections for connection to the control station. It is also responsible for executing commands (and responding to these commands) issued by the client.

J.2.5.1 File `server.tcl`

- `start_port(navigation)` Starting navigation port.
- `start_port(mapper)` Starting mapper port.
- `start_port(sensor)` Starting sensor port.
- `start_port(message)` Starting messenger port.
- `navmode` Workcell mode variable. Possible values are Idle, User, Wander, and Search.
- `proc doService {name cmd}` The actual (incoming) socket service routine.
- `proc svcHandler { name }` Handles the input from the client and client shutdown. Passes control (by name of the socket) to `doService` if the client has not terminated connection.
- `proc accept {name wsock addr port}` Connection acceptance handler for the server. Called when the client makes a connection to the server. Passed the channel we're to communicate with the client on, the address of the client and the port we're using, and the name (alias) of the socket.

- `proc startServer { name }` Create a server socket on the port specified by the name. The name can be `navigation`, `mapper`, `sensor`, or `message`. Calls `accept` when a client attempts a connection.
- `proc open_sockets { }` Routine to open the four sockets for client connection.
- `proc stdinHandler { }` Event-handling procedure used to accept entries from `stdin` and to pass them to the interpreter. Useful for debugging.
- `proc close_socket { name }` Closes the named socket. The argument name can be `navigation`, `mapper`, `sensor`, `message`, or `all`. If `all` is specified, then this procedure is called recursively.
- `proc server_exit { }` Procedure to terminate the server.
- `proc send_command { name cmd }` Procedure to send a command (or data) through the named socket (`navigation`, `mapper`, `sensor`, or `message`). If the socket is not active, then this procedure simply exits. If a command is to be sent through all sockets, then the name `all` can be passed. For a description of the named sockets and what they are used to transfer, see file `memif.tcl`. Interestingly enough, if transmission of commands occurs too quickly, the socket will be closed on the other end.

J.3 Client (Control Station) Code

Control station consists of five modules:

- (1) A **Socket Control Module**, used to connect and disconnect the workcell-control station communication link using four different TCP/IP sockets;
- (2) A **Navigation Console Module**, used to issue user-instigated control directives to the workcell so the user can drive the robot;
- (3) A **Message Module**, which relates debugging and status information to the user about the state of the server and microcontroller software.
- (4) A **Sensor Status Display Module**, which graphically depicts the state of the infrared and bump sensors; and
- (5) A **Mapping Module**, which provides a graphical display of the workcell's environment and the workcell's position within it.

Each module consists of a file specifically used to create the UI, and another file which contains the TCL support code behind it. The UI files are lengthy and not relevant and are not included in the descriptions here. For screen-shots of the UIs, refer to Chapter 6.

The main program is `tr.tcl`, a file which starts all the desired modules and includes a standard input event handler, much as the Server Module in the workcell.

J.3.1 Socket Control Module Code

The Socket Control Module is responsible for giving the user control over which sockets (Navigation, Message, Mapping, or Sensor) are active.

J.3.1.1 File `sock.tcl`

- `proc open_all_sockets { }` Initializes variables, etc. and displays them in the UI.
- `proc read_sock {name}` Read data from a channel and execute it as a TCL command. This would be a good place to insert named socket-specific routing routines, etc. For instance, it may be handy to dump all commands sent through the mapper socket to a text widget. Also, some security measures could be implemented here. Note that there is no corresponding `write_sock` routine since the `puts` command supports direct writing if the socket handle is passed as the first argument.
- `proc terminate {name}` Closing (termination) of the named socket and modification of the socket status UI variables. The `name` argument can be one of `navigation`, `mapper`, `sensor`, `message`, or `all`. In the last case, this procedure is called recursively.
- `proc engage {name}` Procedure to establish socket connections bound to the specified name.
- `proc send_command { name cmd }` Procedure to send a command (or data) through the named socket (`navigation`, `mapper`, `sensor`, or `message`). If the socket is not active, then this procedure simply exits. If a command is to be sent through all sockets, then the name `all` can be passed. For a description of the named sockets and what they are used to transfer, see file `memif.tcl`.

J.3.2 Navigation Console Module Code

The Navigation Console module provides the user with an interface to drive the mobile workcell around. This consists of buttons for moving forward and back-

ward, and pivoting left or right. It also allows instigation of autonomous behaviours, establishing movement setpoints, and displays simplified obstacle information.

J.3.2.1 File nav.tcl

- `proc nav_directive { direction }` Issue a navigation directive to the workcell. The specified direction can be: forward, backward, right, left, or stop.
- `proc send_setpoint { type }` This procedure sends the setpoint of the specified type to the workcell. Argument `type` can be `angle`, `distance`, or `speed`. The actual value is grabbed from the global variables which are attached to the specific widgets.
- `proc set_mode { mode }` A procedure to set the remote workcell operating mode. This mode can be `User`, `Wander`, or `Search`.

J.3.3 Message Module Code

The Message module presents debugging and status information from the workcell microcontrollers and server software in text form. It is a rather simple module and most of the operation is embedded in the UI itself. As such, only one short file is required to interface with the UI.

J.3.3.1 File msg.tcl

- `proc purge { list_name }` This routine purges all entries out of the named list.
- `proc add_list_entry { list_name entry }` Adds the entry to the named list. Ensures that the listbox focus is on the new entry, which is appended to the end of the list.

J.3.4 Sensor Status Display Module Code

J.3.4.1 File sen.tcl

- `proc init_chassis { w }` This procedure loads the bitmap image of the workcell chassis and creates a bunch of objects to represent sensor states. Each object (indicator) is given a 'tag' or 'mnemonic' which uniquely identifies it. In cases where a viewing angle dictates, the indicator is given multiple names.

- `proc sensor_state { state mnemonics }` Procedure to turn the sensor named by the mnemonic (tag) on or off. More than one sensor can be specified if the `mnemonic` argument is a list.
- `proc find_all_sensors { mnemonic w }` A routine to find **all** the widget canvas items (the sensor indicators) which correspond to the given tag (mnemonic).
- `proc send_senreqrate { period }` Procedure to send an updated sensor request rate. This is a binding to the requested sensor update rate entry widget.

J.3.5 Mapper Module Code

The code used for maintaining the maps consists of three components: C mapping routines, the UI, and TCL/TK procedures used to support the UI and interface with C code. The C mapping routines are exactly the same as those outlined earlier for the server code. The UI code, not presented here, results in the UI shown in Fig. 6.8.

The TCL/TK procedures are contained in file `map.tcl`, documented here. Several of these routines are the same as those run on the server.

J.3.5.1 File `map.tcl`

J.3.6 Functions

- `proc starting_settings { }` This procedure is responsible for initializing widgets and opening the default map settings. Although this is called when the client is started, the server will update the map when the mapping socket connection is opened.
- `proc DispMsg {message {title Error!}}` A generic warning window. User must enter OK.
- `proc GetValue { prompt_text }` A generic text entry widget. It returns the text that was typed in, or nothing if the operation was canceled.
- `proc create_workcell { }` Create the workcell canvas objects at the origin with 0 degree bearing.
- `proc move_workcell { x_m y_m bearing }` Move the workcell objects to match the specified coordinates and bearing.

- `proc move_workcell_start { }` When a new map is loaded, or upon initialization, this procedure is called to place the workcell at either the origin, or at the minimum location of the map.
- `proc scroll_map_canvas {x_m y_m}` Procedure to scroll the map to show contents of the location at the specified map (metre) coordinates.
- `proc animation_test { }` A test of the workcell movement routine: animation.
- `proc synchronize_maps { {workcellx 0.0} {workcelly 0.0} }` Procedure to synchronize the client C and TCL maps. The optional workcell coordinate arguments default to 0.
- `proc load_map { }` Function to load a map from a prompt-entered file-name.
- `proc save_map { }` Procedure to save the current map as a binary file. The user is prompted to enter the name of the file. If the named file already exists, then it will be overwritten.
- `proc clear_map { }` A procedure to clear all obstacles from a map. The settings of size and resolution are retained.
- `proc upload_map { }` Routine to send the map at the control base (client) to the workcell (server).
- `proc download_map { }` Routine to download the map from the workcell.
- `proc print_map { }` Procedure to print an encapsulated postscript version of the map to a file. The resulting postscript map is 16 cm wide.
- `proc jump_workcell { }` This procedure is provided to the user so that the workcell can be moved around the map without physical movement. It provides a means for position corrections required, perhaps, due to the inaccuracy of dead-reckoning.
- `proc display_layers { w }` Reorder the display order of the map layers according to the global variable, `toplayer`. Possible `toplayer` settings are: Sonar, Infrared, Bump, Algorithmic, User. Text is always displayed last (on top), along with the workcell.
- `proc workcell_layer {level}` Procedure to move the workcell up or down in the display list so that when searching for a cell near the workcell, the workcell itself will not be found. Argument `level` may be top or bottom.

- `proc configure_resolution { {res_value 0} }` Configure the map resolution. If no value is passed, the resolution defaults to zero indicating that it is necessary to prompt the user to enter the value.
- `proc configure_size { direction { range {0 0} } }` Routine to specify the size of the map in the x or y direction. If a range of 0 0 is encountered (i.e. if no range has been specified, then the user is prompted to enter the range.
- `proc configure_refresh { {refresh_time 0} }` Establishes the requested mapper refresh rate in milliseconds.
- `proc configure_update { }` This procedure is initiated when map attributes have been edited to the user's (or server's) satisfaction.
- `proc get_canvas2m { w }` Get the canvas2m conversion constant in the X direction. Resulting units: pixels physical m.
- `proc clear_canvas { w }` Delete all objects from the map canvas.
- `proc set_mode_bindings { modename }` Establish the canvas-related mouse bindings for specific modes.
- `proc edit_add_point { x_screen y_screen w }` Add a user-defined map barrier to the cell closest to the screen coordinates passed. This is a Button-1 mouse binding for the Edit operating mode.
- `proc set_obstacle_tcl {x_m y_m cell_object tag w {editflag 0}}` This procedure is responsible for changing the colour of cell_object in w to reflect the colour associated with tag, and also to add the corresponding tag to the cell. Argument editflag needs to be non-zero if this function was called from a manual edit of the map. This allows the user to, for instance, load a file (which eventually calls this function) without sending it to the server. Sending the map to the server should be called by procedure `upload_map`.
- `proc delete_obstacle {metre_coordinates tag w}` Procedure to delete an obstacle on the specified level (tag) at the specified x,y coordinates. The C representation of the map is updated.
- `proc create_obstacle {metre_coordinates tag w}` Creates an 'obstacle' at the grid specified by the dual-value metre_coordinates argument. Calls procedure `set_obstacle_tcl` which attaches 'tag' to the cell and sets the colour accordingly.

- `proc layer2number { layer }` A procedure which converts a named layer into its layer number used in the C representation of the map.
- `proc edit_start_line { x_screen y_screen w }` Binding procedure: edit mode, start (mark) line.
- `proc edit_draw_line { x_screen y_screen w }` Binding procedure: edit mode, draw line on mouse motion with B2.
- `proc edit_finish_line { x_screen y_screen w }` Binding procedure: edit mode, drawing of barrier line complete. The bounding box (rectangular) of the line item is determined, and cells that are found in this region are set with the User tag.
- `proc edit_delete_point { x_screen y_screen w }` Delete defined map barrier to the cell closest to the screen coordinates passed. This is a Button-3 mouse binding for the Edit operating mode.
- `proc clear_obstacle_tcl { cell_object w }` Reset cell colour and delete all tags except 'cell'.
- `proc add_text { x y w }` Binding procedure: text mode, add overlay text.
- `proc delete_text { x y w }` Binding procedure: text mode, delete overlay text.
- `proc move_text { x y w }` Binding procedure: text mode, move marked text.
- `proc mark_text { x y w }` Binding procedure: text mode, select text for movement.
- `proc select_cell { x_screen y_screen w }` Procedure to select a map cell when in Select mode.
- `proc display_selection { coords }` Display the metre coordinates in the selection status area.
- `proc cell_to_coordinates { object w }` Convert a selected cell into the coordinates specifying it's midpoint (in metres).
- `proc m_to_canvas { metre_coord }` Convert from map metres to canvas coordinates.
- `proc canvas_to_m { canvas_coord }` Convert from canvas coordinates to metres.

- `proc screen_to_m { x_screen y_screen w }` Procedure to convert screen coordinates to metres.
- `proc m_to_cm { coord_m }` Convert a physical map distance to a canvas distance in cm. A 'c' is appended onto the result allowing direct use in canvas calls.
- `proc set_grid { xrange yrange w }` Display a grid in the given range specified in map metres. The range of the grid is adjusted so that the origin is at the centre of a grid point. Each cell has (by default) a 'cell' tag.
- `proc set_scroll_region { xrange_m yrange_m w }` Set the map scroll region (canvas w) based on the range in metres.
- `proc disp_mouse_coords { x_screen y_screen w }` Binding to mouse motion in map canvas. Displays the mouse position in the status frame after conversion to metres has occurred.
- `proc scale_map { ratio }` This procedure is called when the user changes the map scale.
- `proc nav_workcell_idle { }` Routine to place the workcell in idle mode: motors stopped, sensor update relay, and so on, terminated.
- `proc add_ping_dot { metre_coordinates }` Routine to add a dot where sonar coordinates gave a reading.
- `proc sonar_sweep { }` Routine to initiate a workcell SONAR sweep.
- `proc clear_ping_dots { }` Procedure to remove all the circles placed on the map by `add_ping_dot`.

Appendix K

Power-Up Steps

This appendix is an outline of the steps required to activate the workcell. Dedicated sections are provided which outline the steps needed to start the PPP connection to the workcell, download the code to the microcontrollers, start the workcell server, and the control station client.

K.1 PPP Connection to the Workcell

Connection to the workcell using PPP is required. This step does not need to be implemented every time: only if an Internet connection between the workcell and the control-station is not already made. To establish a PPP connection, on the control station, follow these steps:

- (1) Start the X-windows system on the control station.
- (2) In a shell window, gain root access using the `su` command. You will be performing the following steps as root in this window.
- (3) Execute `minicom -s robot` to make normal serial connection to the workcell. The workcell is configured to start the `getty` program on its serial port connected to the control station when it comes out of the boot process. Exit the minicom configuration screen and a login prompt for `rover.ee.ualberta.ca` should be seen.
- (4) Login to the workcell as `robot`.
- (5) On the workcell, gain root access using the `su` command.
- (6) Start PPP on the workcell by executing `/home/robot/connect`, a PPP configuration script.

- (7) Exit minicom (`Ctrl-a q`) on the control station.
- (8) Start PPP on the control station by executing `/home/robot/connect`.

This should establish a PPP connection between the control station and the workcell. Test the connection using `ping rover`. Normally, the PPP connection to the workcell is left running.

K.2 Downloading the MCU Layer Code

The following steps are used to download and the microcontroller layer code.

- (1) On the control station, open two windows. One will be used for sending information to the microcontroller, the other will be used to interface with the parallel port expansion interface to the microcontrollers.
- (2) In both windows, login to the workcell using `rlogin rover -l robot`.
- (3) In one window, gain root access on the workcell using the `su` command. This will be referred to as the 'control window'.
- (4) In the 'control window', execute `/home/robot/code/mcu/program/mcuset`. This is a program which interfaces to the MCU control lines, and to the shared memory.
- (5) Once `mcuset` is running, apply power to the parallel expansion board.
- (6) Enable the outputs on the parallel expansion PCB by toggling the switch on the board to 'On'.
- (7) Within the `mcuset` program, place Microcontroller 1 (the navigation MCU) into download mode `D 1`. This will prevent the drive motors from acting unpredictably when power is applied to the MCU layer.
- (8) Apply power to the MCUs. If the SONAR units 'ping' at this point, then place Microcontroller 3 (the SONAR MCU) into download mode through `mcuset D 3`.
- (9) Download the Interactive-C pseudo-code interpreter to all three of the microcontrollers. Perform the following steps for each microcontroller:
 - (a) Using `mcuset` in the control window, place the microcontroller into download mode by typing `D x` where `x` is the microcontroller number (1, 2, or 3).

- (b) In the other window (logged into the workcell as user robot), execute `dl -bs_ignore pcodeTel.s19`. This sends the pseudo-code interpreter to the MCU which is in download mode.
- (10) Once the pseudo-code interpreter is downloaded to all MCUs, the application code needs to be sent and executed:
- (a) Make MCU 1 (the navigation MCU) active by typing `A 1 R 1` (which also resets the microcontroller).
 - (b) In the other window, change to directory `/home/robot/code/mcu/navigation`.
 - (c) Execute `ic navIC.lis` to load the Interactive-C list file onto the microcontroller.
 - (d) Exit the ic program by typing `exit`.
 - (e) Make MCU 2 (the proximity MCU) active by typing `A 2 R 2`.
 - (f) In the other window, change to directory `/home/robot/code/mcu/proximity`.
 - (g) Execute `ic proxIC.c`.
 - (h) Exit the ic program by typing `exit`.
 - (i) Make MCU 3 (the SONAR MCU) active by typing `A 3 R 3`.
 - (j) In the other window, change to directory `/home/robot/code/mcu/sonar`.
 - (k) Execute `ic sonIC.c`.
 - (l) Exit the ic program by typing `exit`.
- (11) If you are intending on starting the workcell server, leave the `mcuset` program running.

K.3 Starting the Workcell Server

With the microcontroller-layer code resident on the MCUs, the server and client can be started. To start the workcell server, the following steps are used:

- (1) Start the program `mcuset` on the workcell, if it is not already running. See the steps in the last section to achieve this. The window in which `mcuset` is executing will be referred to as the 'control window'.
- (2) In another window, log in to the workcell using the command `rlogin rover -l robot`. This window will be referred to as the 'server window'.
- (3) In the server window, gain root access using the command `su`.

- (4) Again, in the server window, change to directory `/home/robot/code/workcell`.
- (5) Execute `./server.tcl` to start the workcell server. This program starts by clearing the memory shared with the microcontrollers which takes a few moments. When the TCL interpreter shows the prompt `Server 0.0%`, proceed to the next step.
- (6) In the control window, reset all microcontrollers by typing `R A`. This starts executing the microcontroller- resident code.

Note that if at any time the workcell operates unpredictably, in the control window (in which `mcuset` is executing) reset all the MCUs.

K.4 Starting the Control Station Client

With the workcell server and the microcontroller software running, the control station client can be started. Starting the client requires only a few steps:

- (1) Open a window on the control station (the X interface needs to be running).
- (2) Change to directory `/home/robot/code/base`.
- (3) Execute `./tr.tcl`. This starts the TCL/TK client interpreter and brings up the user interface.

Glossary

API Application Program Interface. A software interface to an application.

CMOS Complementary Metal Oxide Semiconductor.

HC MOS High-density CMOS

IC Integrated Circuit

IR Infrared.

MCU Microcontroller Unit.

PC Personal Computer. Usually an IBM-compatible machine.

PCB Printed Circuit Board.

SCI Serial Communications Interface. A facility offered on the HC11 microcontroller which is used for asynchronous serial communication.

SPI Serial Peripheral Interface. A facility offered on the HC11 microcontroller which is used for synchronous serial communication.

TCL Tool Command Language. An interpreted (script) language.

TK Tool Kit. The window extension for TCL.

TTL Transistor-Transistor Logic. A technology used to implement digital logic.

UI User Interface. The component of a software application with which the user of the program interacts.