TABLE OF CONTENTS

ABSTRACT

The four main systems that make the robot, Thomas, capable of accomplishing the ultimate goal of Landmark Navigation are --

1) Expanded 68HC11 EVBU Board.

2) DC Motor Actuation with Motor Encoders.

3) Infra-Red Emitter/Detector Array.

4) Pan/Tilt Servo Head with Sonar Detector.

Thomas' design makes it rugged, reliable, and esthetically pleasing. Although the Landmark Navigation software is not yet written, all of the necessary hardware is now in place. The software included in Appendix A is an object avoidance program that utilizes all of Thomas' systems and shows that Thomas is ready to begin the next step of landmark navigation software development.

EXECUTIVE SUMMARY

Two specific goals have driven the design of the robot Thomas. The first goal is to build a robot with all the capabilities of another Machine Intelligence Laboratory robot, Riker. The second is to build this new robot much more robust electrically, mechanically and esthetically than its predecessor.

To accomplish the goal of Landmark Navigation, Riker's designer Steven Seed equipped Riker with IR sensors for short distance object detection, motor encoders for measuring distance traveled, a sonar sensor for long distance object measurements, and an expanded 68HC11 EVBU micro controller to control the systems.

To accomplish the first goal, Thomas has been equipped with each of Riker's features, each of which is a significant upgrade both in technology and accuracy over its older cousin. Thomas also has a feature never incorporated into Riker, Thomas has a 2-axis Pan/Tilt Servo head that enables it to point it's sonar array in any direction in the upper hemisphere. To accomplish the second goal, Thomas' design is electrically and mechanically robust. Each piece of Thomas is modular in design to facilitate the installation of new parts and circuit upgrades. The wire routing and part placement have also been designed to keep Thomas as neat as possible.

The balance of this paper details how Thomas has met his design goals by showing how each system has been constructed and by giving an example of the software that has been written to drive the systems. Although the Landmark Navigation software is not yet written, Thomas has the necessary hardware to accomplish this goal also. This will be the next step in Thomas' development.

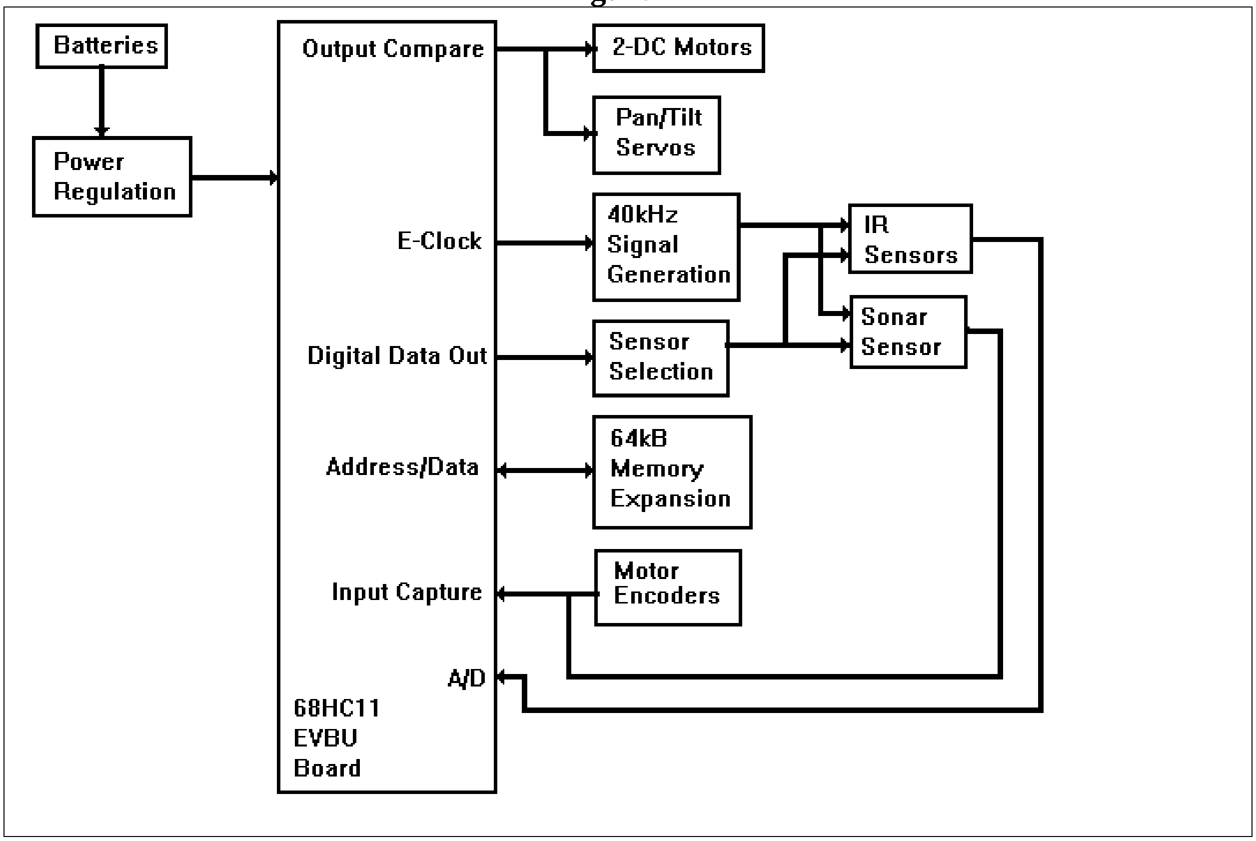
INTRODUCTION

The robot, Thomas' design goal is to duplicate the functions of the robot, Riker. Riker is a robot, developed for the Machine Intelligence Laboratory. Built by Steven Seed as a Masters project in the spring of 1994, Riker was developed to perform Landmark Navigation. Thomas' design incorporates all of Riker's capabilities in a more rugged and esthetically pleasing package.

SYSTEM OVERVIEW

To achieve Thomas' overall goal, it's design uses a variety of sensors and system upgrades. Figure #1 shows a system level block diagram of Thomas.

Figure #1



As shown in Figure #1, Thomas has the following systems, and/or subsystems:

1) Power Regulation System

2) Memory Expansion

3) DC Motor Actuation

4) Motor Encoder System

5) Pan/Tilt Servo Head

6) Infrared Sensor System

7) Sonar Sensor System

When combined, these systems equip Thomas with every capability possessed by the robot Riker, and, in several instances, Thomas surpasses Riker's capabilities.

MAIN BODY LAYOUT

Thomas' main body is a box constructed of sheet aluminum, slightly rounded at the back. The main axle for the two front wheels is pop-rivitted to the bottom as is the back castor. Figure #2 shows the interior of the main body, as well as Thomas' dimensions.

The main body box contains the following elements:

1) Motors

2) Motor Driver Circuitry

3) Motor Encoder Circuitry

4)Batteries

5) Power Regulation Circuitry

6) Five of the 7 IR-LEDs, these are mounted through the sides of the main body box.

The metal body acts as a Faraday Cage to electromagneticly isolate Thomas' electronics from any noise transmitted from the motors. All signals passed into or out of the main body box pass through one of two 9-pin feed-through connectors. The back of the main body box can be removed to provide access to the battery pack. Access to the interior of the main body box is provided through the top which is secured with screws. The mount for the Pan/Tilt Servo head is screwed onto the top plate of the main body box. Figure #3 shows the Pan/Tilt mounting bracket and head configuration

Figure #2

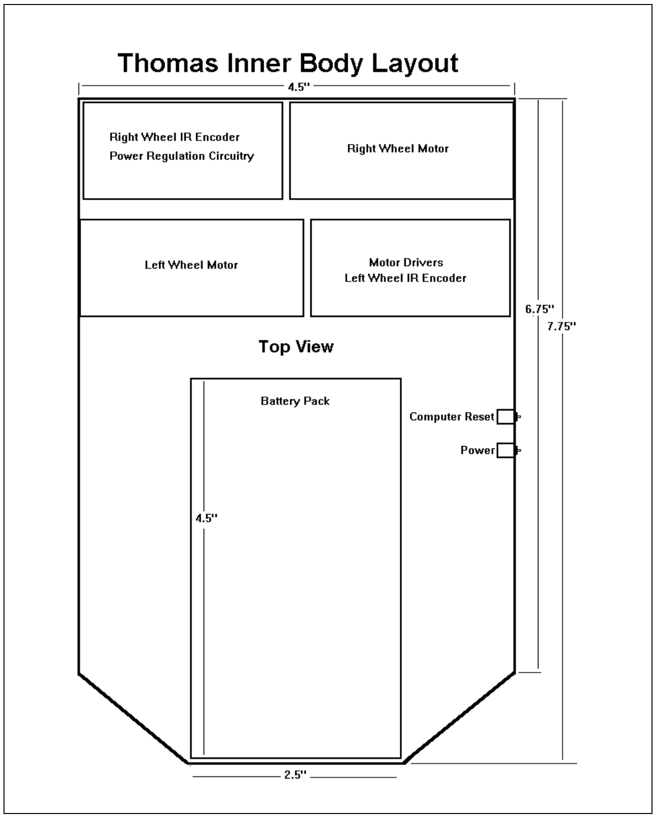
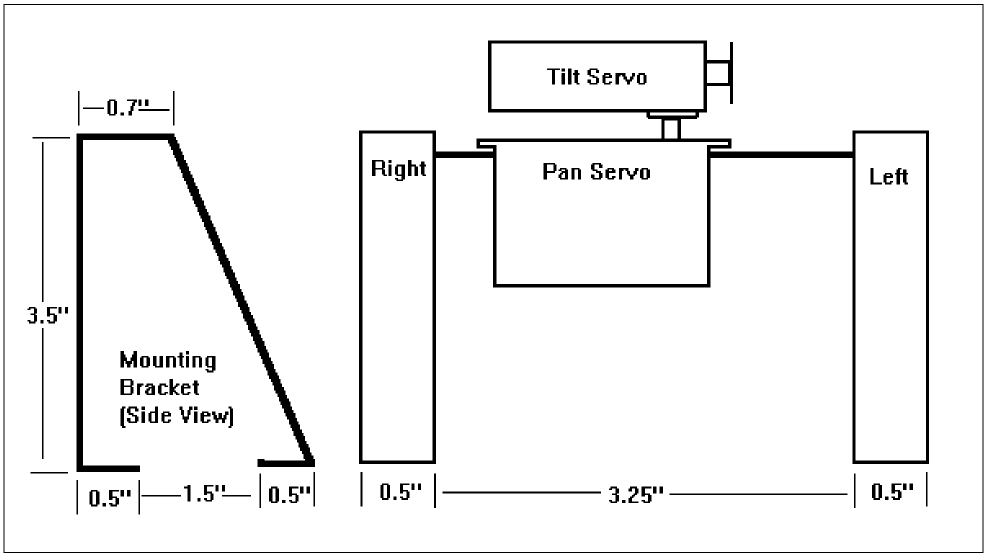


Figure #3

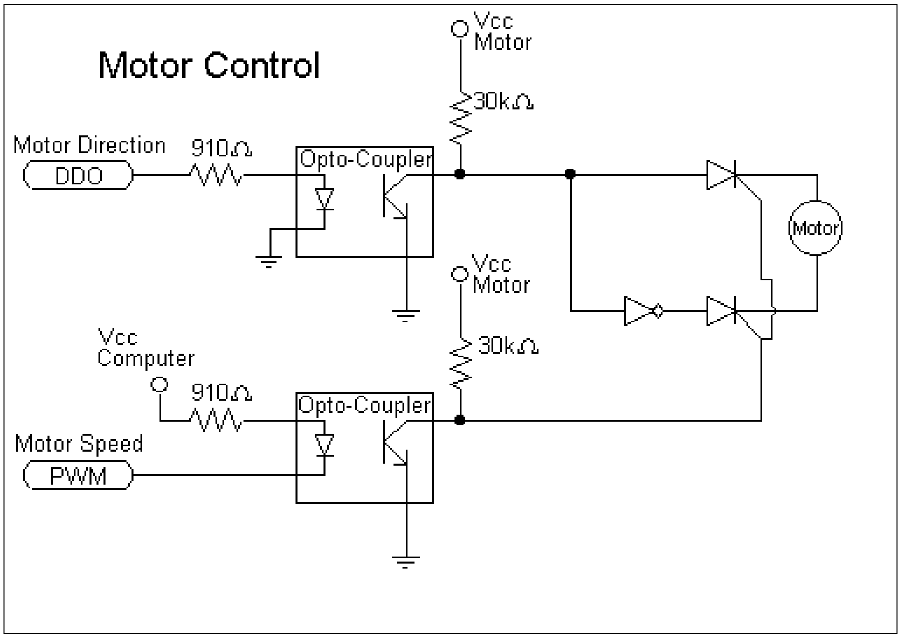


ROBOT MOVEMENT

Drive Motors

Separate DC motors drive the front wheels. These motors contain their own gear reduction system and apply their power to the front wheels through a single 14/82 gear pair. The motors are very efficient, which is useful in reducing power consumption, but can also cause problems. The largest problem connected with the drive motors is Thomas' momentum. When Thomas is moving it will not stop immediately, but coasts to a stop over a distance of approximately 2 inches. This causes problems in wall following and distance measuring behaviors. Thomas uses Pulse Width Modulation [PWM] and a direction bit to control it's two drive motors. OC2 and OC3 on 68HC11 generate the PWM signals. The direction bits come from the DDO (Digital Data Out) memory mapped I/O as shown in Figure #4. Both the direction and PWM signals are sent through a TI9310 Optocoupler to maintain electrical isolation of the motors from the electronics. Next, the signals are fed into a SN754410NE Motor driver chip that drives the motors.

Figure #4



The motor direction bits of DDO are Bits 0 and 1.

1 = Forward, 0 = Backward.

The percentage of the PWM signal driving the motors controls the motor speed, the higher the percentage, the faster the motor runs.

Pan/Tilt Servo Head

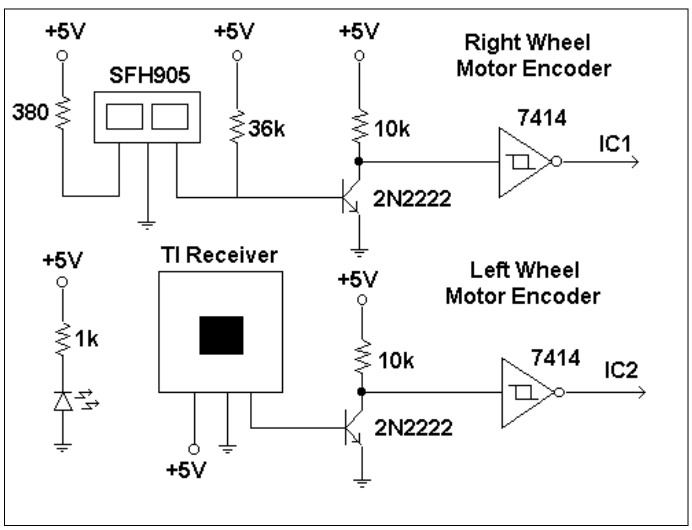
The Pan/Tilt Servo Head mounted on top of Thomas uses standard Futaba servos to derive its actuation. As shown in Figure #3, the 190^{o} of servo movement allows accurate positioning of the servo head in any direction in the upper hemisphere. The PWM signals developed in OC4 and OC5 control servo direction (See Appendix A Memory Location 14B4). Servos derive their position data from the width of the positive part of the 55Hz PWM Signal. The position of the servo is controlled by controlling the width of the pulse (See Appendix A, Memory Location 1555).

SENSORS

Motor Encoder

The motor encoder system utilizes both older and newer technologies to perform its function. Figure #5 shows the circuitry involved with the motor encoder system. During a modification of the Power Regulation system, the circuit board containing the IR emmiter/detector for the right wheel motor was replaced with the new SFH905 integrated IR emmiter/detector. The IR. emmiter/detector for the left wheel motor remains the older style. The motor encoder signals from both encoders are amplified to TTL levels by 2N2222 common emitter amplifier circuits. The rising and falling edges of the outputs of these amplifiers are squared up by 7414 Schmitt Trigger inverters.

Figure #5



The outputs of these inverters are sent to IC1 and IC2 for counting (See Appendix A, memory Location 171E).

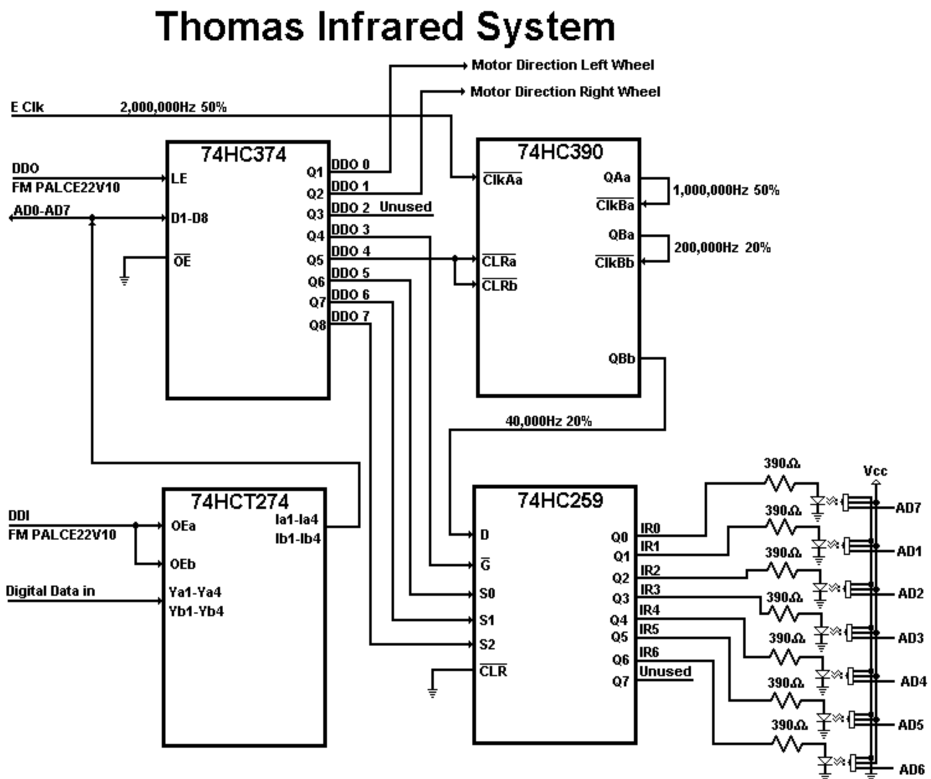
IR Sensors

Thomas uses 7 infrared emmiter/detector pairs to sense the outside world. Figure #6 is a diagram of the IR sensing system. The IR sensing system employed in Thomas has two important features**.** The first of these is the generation of the 40kHz signal that drives the IR emitters. This is accomplished by the use of a 74HC390 Dual 4 Stage Ripple Counter. The 40kHz is derived from the E-Clock by dividing the E-Clock in the following manner --

This method produces a stable 40kHz signal with a duty cycle of 20%. By dividing the E-Clock in a different order it is possible to produce a 50% 40kHz, but a 20% signal is more efficient because of the lower power consumption.

The second important feature of the IR Sensing System is its ability to illuminate and read a single emmiter/detector pair. Using the 74HC259 8-Bit Addressable Latch, the 40kHz signal is routed to any one of the 7 IR emitters. The corresponding IR Detector can then be measured. It is also possible to take readings without turning any IR emitter on. This allows "Dark" readings to be taken to help eliminate background IR radiation.

Figure #6



The IR system is controlled through the memory mapped I/O DDO (Digital Data Out). This Digital Output word is mapped to memory location $0200 - $021F. When data is written to this location in memory a latching pulse is generated on the DDO line of the PALCE22V10. This latching pulse latches the data in to a 74HC374 8-Bit Latch. From the latch the data is used to control the IR System and to control Motor Direction. The DDO data word is defined as follows:

DDO

$0200 -

$021F

7	6	5	4	3	2	1	0
IR# Bit 2	IR# Bit 1	IR# Bit 0	40kHz 0=ON	Demux 0=ON	Not Used	Right Wheel	Left Wheel

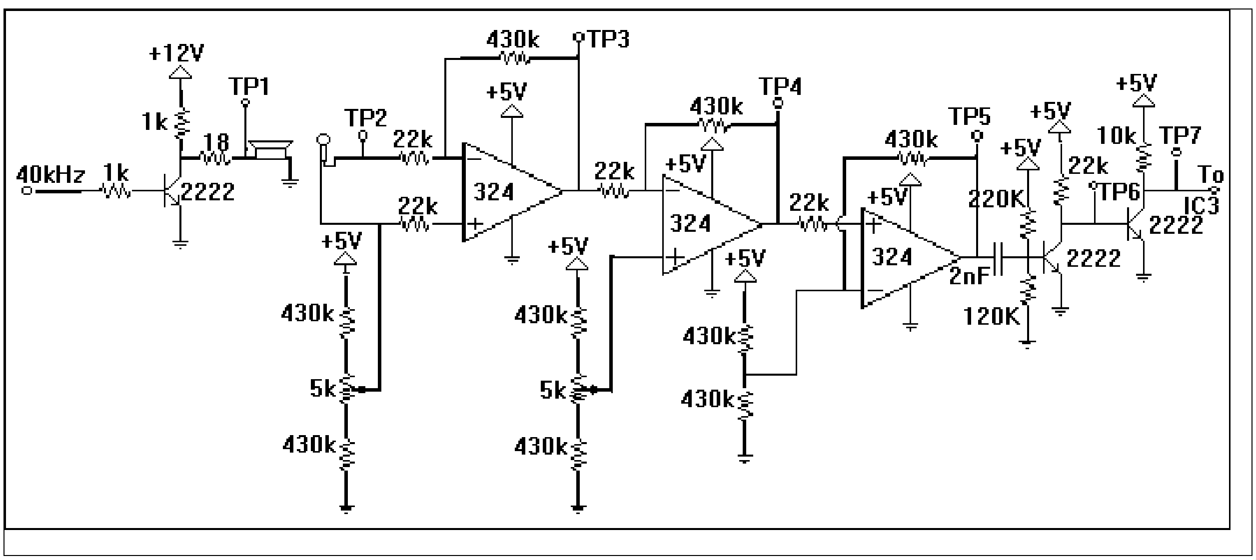
			1=OFF	1=OFF		0=Reverse 1=Forward	0=Reverse 1=Forward

The IR. Detectors on Thomas have received the "Arial" modification. This allows the detector to produce an analog output that is fed directly into the 68HC11 A/D system. A characterization of the IR Sensing System is provided in the Experimental Layout and Result section of this report.

Sonar

The Sonar system consists of a 40kHz speaker and it's corresponding microphone receiver. Figure #7 shows the circuit used in the sonar system. The circuit consists of a common emitter amplifier circuit for amplifying the 40kHz signal that is used to drive the speaker.

Figure #7



This increases the output signal power, and hence increases the range. The returned signal is amplified through 3 op-amp stages, each stage having a gain of 20. The resulting signal is shaped to TTL levels using a Darlington pair of 2222 transistors. The return signal is then sent to IC3 where it is used to determine the time of flight (See Appendix A, Memory Location 137A).

BEHAVIORS

Four major behaviors have been realized to date. These four behaviors are contained in the code of the program listed in Appendix A. The behaviors are:

1) Straight Line -- Adjust motor PWM until both motor encoder counts are equal (See Appendix A, Memory Location 1566).

2) Object Avoidance -- Use IR sensors to detect objects and avoid them (See Appendix A, Memory Location 125A).

3) Sonar Scanning -- Use Sonar to scan 180^{o} in front of Thomas and return Pan angle to closest object (See Appendix A, Memory Location 11A9).

4) Sonar Turn -- Use sonar scanning pan angle to turn Thomas toward scanned object (See Appendix A, Memory Location 111B).

SENSOR CHARACTERIZATION AND RESULTS

IR Sensors

The characterization of the IR sensors was accomplished by taking readings from the seven IR sensors at 1 inch intervals from a light colored piece of card board and then a dark colored piece of cardboard. Figure #8 shows a plot of this collected data. The graph shows the IR sensor's sensitivity and characteristic response. Although the response curve is not linear, the center section from 2 inches to 12 inches is very close to linear. The graph also shows how close the seven IR sensors are in their response. The graph indicates a good repeatability from sensor to sensor.

Figure #8

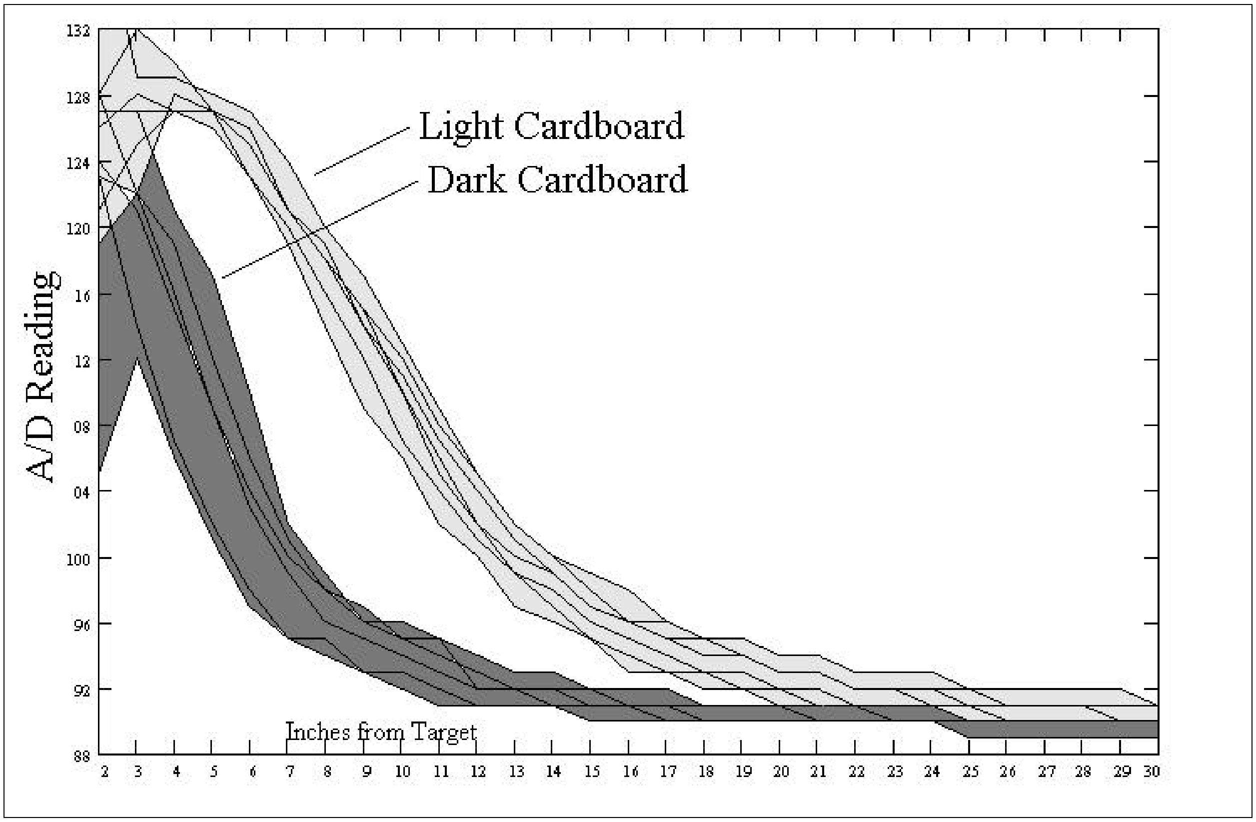


Figure #9 contains the data used to create Figure #8. The data has been compressed to fit it onto a single page. The first column, headed by the variable "n", contains the distance from target data in inches. The following columns contain the data for the respective IR's. $IR0L_n$ means IR #0, Light, at n inches, $IR0D_n$ means IR #0, Dark, at n inches, and so on.

Figure

#9

n	IR0L_n :=	IR0D_n :=	IR1L_n :=	IR1D_n :=	IR2L_n :=	IR2D_n :=	IR3L_n :=	IR3D_n :=	IR4L_n :=	IR4D_n :=	IR5L_n :=	IR5D_n :=	IR6L_n :=	IR6D_n :=
30	5Ah	5Ah	5Ah	59h	5Bh	5Ah	5Ah	5Ah	5Ah	5Ah	5Bh	5Bh	5Ah	5Ah
29	5Ah	5Ah	5Ah	59h	5Bh	5Ah	5Ah	5Ah	5Ah	5Ah	5Ch	5Bh	5Ah	5Ah
28	5Bh	5Ah	5Ah	59h	5Bh	5Ah	5Ah	5Ah	5Ah	5Ah	5Ch	5Bh	5Bh	5Ah
27	5Bh	5Ah	5Ah	59h	5Bh	5Ah	5Ah	5Ah	5Ah	5Ah	5Ch	5Bh	5Bh	5Ah
26	5Bh	5Ah	5Ah	59h	5Bh	5Ah	5Ah	5Ah	5Ah	5Ah	5Ch	5Bh	5Bh	5Ah
25	5Bh	5Ah	5Ah	59h	5Ch	5Ah	5Ah	5Ah	5Bh	5Ah	5Ch	5Bh	5Bh	5Ah
24	5Ch	5Ah	5Bh	5Ah	5Ch	5Ah	5Bh	5Ah	5Bh	5Ah	5Dh	5Bh	5Bh	5Ah
23	5Ch	5Ah	5Bh	5Ah	5Ch	5Ah	5Bh	5Ah	5Bh	5Ah	5Dh	5Bh	5Ch	5Ah
22	5Ch	5Ah	5Bh	5Ah	5Ch	5Bh	5Bh	5Ah	5Bh	5Ah	5Dh	5Bh	5Ch	5Ah
21	5Dh	5Ah	5Bh	5Ah	5Dh	5Bh	5Bh	5Ah	5Ch	5Ah	5Eh	5Bh	5Dh	5Ah
20	5Dh	5Ah	5Ch	5Ah	5Dh	5Bh	5Bh	5Ah	5Ch	5Ah	5Eh	5Bh	5Dh	5Bh
19	5Eh	5Ah	5Ch	5Ah	5Eh	5Bh	5Ch	5Ah	5Dh	5Ah	5Fh	5Bh	5Eh	5Bh
18	5Eh	5Ah	5Dh	5Ah	5Fh	5Bh	5Ch	5Ah	5Dh	5Ah	5Fh	5Bh	5Fh	5Bh
17	5Fh	5Ah	5Dh	5Ah	60h	5Ch	5Dh	5Ah	5Eh	5Bh	60h	5Bh	5Fh	5Bh
16	60h	5Bh	5Eh	5Ah	60h	5Ch	5Dh	5Bh	5Fh	5Bh	62h	5Bh	60h	5Bh
15	61h	5Bh	5Fh	5Ah	61h	5Ch	5Fh	5Bh	60h	5Bh	63h	5Bh	62h	5Ch
14	63h	5Ch	61h	5Bh	63h	5Dh	60h	5Bh	62h	5Bh	64h	5Bh	64h	5Ch
13	64h	5Ch	63h	5Bh	65h	5Dh	61h	5Bh	63h	5Bh	66h	5Ch	66h	5Ch
12	66h	5Dh	65h	5Bh	68h	5Eh	64h	5Bh	66h	5Bh	69h	5Ch	69h	5Ch
11	69h	5Eh	68h	5Bh	6Bh	5Fh	66h	5Bh	6Ah	5Ch	6Ch	5Fh	6Dh	5Dh
10	6Eh	5Fh	6Bh	5Ch	6Fh	5Fh	6Ah	5Ch	6Eh	5Dh	70h	60h	71h	5Eh
9	73h	60h	70h	5Dh	72h	61h	6Dh	5Ch	72h	5Dh	73h	60h	75h	5Fh
8	76h	63h	74h	5Eh	77h	62h	72h	5Dh	76h	5Fh	76h	62h	78h	60h
7	79h	66h	78h	5Fh	79h	64h	77h	5Fh	79h	5Fh	79h	65h	7Ch	63h
6	7Eh	6Eh	7Bh	61h	7Dh	68h	7Bh	61h	7Dh	62h	7Dh	6Ah	7Fh	67h
5	7Fh	75h	7Eh	65h	7Fh	6Dh	7Fh	62h	7Fh	66h	7Fh	70h	80h	6Dh
4	80h	79h	7Fh	6Ah	7Fh	73h	82h	67h	7Fh	6Bh	7Fh	77h	81h	74h
3	7Ah	7Fh	7Fh	70h	80h	79h	84h	6Eh	7Dh	72h	7Dh	7Ah	81h	7Ah
2	7Bh	7Fh	7Fh	69h	7Eh	7Ch	80h	76h	79h	7Bh	79h	77h	8Ch	80h

Sonar

The sonar system was characterized and the data plotted. The graph of the data shows how accurate, linear, and reliable the sonar system is. The characterization of the sonar system was conducted in the following manner. The system was tested against a hard smooth wall (white board) to obtain a maximum returned signal. Data points were taken from the maximum return distance (100 inches) to the minimum return distance (8 inches) in 2 inch increments. Figure #10 is a graph of the data shown in Figure #11. It should be noted that the software that drives the sonar system sends only a single pulse, counts the number of E-Clocks until the return pulse and displays that number. The software then waits for a key-press the take the next reading. Therefore, no averaging is being taken of multiple readings. However, of the 16 bits returned, Bit 15 is disregarded because it is always 0, Bit 14 - Bit 7 are stable, and Bit 6 - Bit 0 are unstable. This gives the readings 7 bits of significance.

Figure #10

Figure #11 contains the data used to create Figure #10. The "Time of Flight" column is only a very rough estimate calculated by multiplying the # of E-Cycles by the time for 1 E-Cycle of 1/2,000,000. The actual E-Clock cycle was never measured. The speed of sound was calculated in the following manner -- Two data points, 30 inches apart, were used for each calculation. Using the following calculation -- These various calculated speed of sounds were averaged to produce a final result. The speed of sound given in a standard college physics book is . The experimentally derived speed of sound is this difference yields an error of 3.1%, assuming that the speed of sound in the lab at the time of the experiment was actually . A returned value of 0000 indicates the reading was too close to be accurate. A returned value of ffff indicates that no return pulse was detected.

Figure #11

Distance [inch]	Returned Value [Hex]	[Decimal]	Time of Flight [mS]	Speed of Sound [Mi/Hr]	Avreage Speed of Sound [Mi/Hr]
8	0	0	0		
10	0bcb	3019	1.5095	760.957792207792	754.855384089956
12	0e2c	3628	1.814	757.828367031435	
14	1092	4242	2.121	757.744145163572	
16	12e1	4833	2.4165	771.113076021468	
18	1536	5430	2.715	761.63782598099	
20	17a1	6049	3.0245	760.194204279387	
22	19e8	6632	3.316	759.516744812501	
24	1c05	7173	3.5865	746.870612135154	
26	1e23	7715	3.8575	734.876246840032	
28	2099	8345	4.1725	737.978332956145	
30	2374	9076	4.538	740.865132911205	
32	24fa	9466	4.733	740.462838638338	
34	27c3	10179	5.0895	745.808555915754	
36	29ff	10751	5.3755	753.057413097175	
38	2cb1	11441	5.7205	760.194204279387	
40	2ecb	11979	5.9895	753.890072775522	
42	3151	12625	6.3125	754.641042410827	

44	33b8	13240	6.62	752.641772621903
46	356b	13675	6.8375	742.155417239776
48	382e	14382	7.191	742.155417239776
50	3aaa	15018	7.509	740.221671716624
52	3cf9	15609	7.8045	741.509713777251
54	3fae	16302	8.151	755.644665652424
56	4261	16993	8.4965	779.933861608536
58	44b0	17584	8.792	768.33241133444
60	4767	18279	9.1395	782.350179940541
62	48f2	18674	9.337	759.432147269082
64	4b79	19321	9.6605	768.852257350228
66	4d5d	19805	9.9025	759.940015401451
68	4fba	20410	10.205	
70	521f	21023	10.5115	
72	549c	21660	10.83	
74	571b	22299	11.1495	
76	594e	22862	11.431	
78	5c11	23569	11.7845	
80	5ea5	24229	12.1145	
82	60e4	24804	12.402	

84	62ed	25325	12.6625
86	6487	25735	12.8675
88	675a	26458	13.229
90	6972	26994	13.497
92	6c04	27652	13.826
94	6e1d	28189	14.0945
96	7069	28777	14.3885
98	ffff	65535	32.7675
100	ffff	65535	32.7675

CONCLUSION

All of Thomas' systems are functioning and operating well. While there remains room for improvement in some areas, Thomas' design goals have been met. Thomas' main body construction is rugged and clean. The 68HC11 EVBU board memory upgrade and port reconstruction are compact and fully functional. The DC motor actuation and motor encoder systems are fully functional and efficient. The short range IR sensing system is working well and senses objects up to 2 feet away. The Pan/Tilt servo head is operational, but remains somewhat jerky and prone to oscillation. This is most probably due to the fact that the servos are receiving unregulated power. Installing a separate 5Vdc regulated power supply for the servos should eliminate this problem. The Sonar sensing system is fully operational and very accurate up to 8 feet, however, the sonar system becomes unusable if the servos are moving when the sonar reading is taken. Solving the servo problem will help improve sonar accuracy. All of the software drivers for all of Thomas' systems have been written and are compatible with each other. The next step in the development of Thomas is the writing the Landmark Navigation software. This will most probably prove to be a significant software challenge. Hopefully some higher level language than Assembly can be found to simplify the task of writing the landmark navigation algorithms.

CLASS COMMENTS

The class was entirely enjoyable, and I look forward to continuing my work with Thomas. However, there were times when I became completely frustrated with the lack of equipment in the IMDL lab. I eventually decided to do all of my work at home. Several other students expressed similar frustrations to me over the semester. I realize that this class is new, and it is my hope and belief that this problem will be rectified by the fall semester.

REFERENCES

The 6.270 Robot Builders Guide, by Fred G. Martin, MIT, 1992

Technical Document for the Riker Robot, by Steven Seed, Machine Intelligence Laboratory, Department of Electrical Engineering, University of Florida, 1994

Microcomputer Engineering, by Gene H. Miller, GMI Engineering & Management Institute, Flint, Michigan, 1993

HC11 M68HC11 Reference Manual, Motorola Inc., 1991

HC11 M68HC11 E Series Programming Reference Guide, Motorola Inc., 1991

Effective Professional and Technical Writing, by Michael L. Keene, Second Edition, D.C. Heath and Company, Massachusetts.

Mobile Robots: Inspiration to Implementation, by Joseph L Jones, and Anita M. Flynn

APPENDIX A -- DEMO1.LIS

```
DEMO1.ASM              Assembled with IASM   04/29/1995  20:51  PAGE 1


                       1  * Programmer : Kelly Snow
```

```
     * Description: This is Simple behavior program . In this version Thomas will
     *  go forward until it sees an object ahead, it will then check it's side
     *  sensors and turn toward which ever is free. This version is designed to
     *  utilize the modified IR. sensor.
     *
     * This version uses the motor encoders to adjust the pulse width when the
     *  robot is going strait. The PWM is adjusted to make the motor counts equal.
     *
     * This version also uses the new Servo routine that utilizes an adjustable
     *  step size to position the servos
     *
     * The program periodically scans with the sonar head and turns Thomas toward
     *   the closes object it detects
     *
     * The IR. routine utilizes a differential (Dark vs. Light) algorithm
     *
     *       NOTE *** IR0 is Connected to PE7 do to Buffalo constraints on PE0. ****
     ***************************************
     * Register Address Definition
     ***************************************
0000 REGBAS  EQU $1000       ; Base address for register block
0000 CFORC   EQU $0B ; used to force the OC registers
0000 OC1M    EQU $C  ; Register used by OC1 to determine
```

```
                                                      which OC pins to
                    *                 ;  control
0000                OC1D    EQU $D  ; Register used by OC1 to determine
                                                       what to set the
                    *                 ;  affected OC pins to.
0000                TCNT    EQU $0E ; T-Count Register
0000                TIC3    EQU $14 ; TCNT value when IC3 triggers
0000                TOC1    EQU $16 ; Timer Output Compare Register for
                                                                  OC1
0000                TOC2    EQU $18 ; Timer Output Compare Register for
                                                                  OC2
0000                TOC3    EQU $1A ; Timer Output Compare Register for
                                                                  OC3
0000                TOC4    EQU $1C ; Timer Output Compare Register for
                                                                  OC4
0000                TOC5    EQU $1E ; Timer Output Compare Register for
                                                                  OC5
0000                TCTL1   EQU $20 ; OC2 - OC5 Control Register
0000                TCTL2   EQU $21 ; Select edge for capture for IC1 -
                                                                    3
0000                TMSK1   EQU $22 ; OC1 - OC5 Interrupt Mask Register
0000                TFLG1   EQU $23 ; OC1 - OC5 Flag Register
0000                TMSK2   EQU $24 ; Used to enable RTI interrupts
0000                TFLG2   EQU $25 ; BIT 7 -> TCNT FLAG
0000                PACTL   EQU $26 ; bit #2 determines if IC4 or OC5
                                                            is active
0000                ADCTL   EQU $30 ; A/D Control Register
0000                ADR1    EQU $31 ; A/D Result Register #1
0000                ADR2    EQU $32 ; A/D Result Register #2
0000                ADR3    EQU $33 ; A/D Result Register #3
0000                ADR4    EQU $34 ; A/D Result Register #4
0000                OPTION  EQU $39 ; OPTION Register used to turn on
                                                              the A/D
0000                DDO     EQU $0200       ; Digital Data Out
```

```
***************************************
* Subroutines in this program
*
* Servoini --- Initializes OC4 and OC5 to produce the PWM for the servos
* OC4ISR ----- ISR for OC4
* OC5ISR ----- ISR for OC5
* MoveHead --- Moves the Pan/Tilt head to the desired position
* ServoCalc -- Used by MoveHead to calculate PWM High time
* Sonar -- Takes sonar reading and stores it in SonVal
* ADSetup -- Setup A/D system
* IRRead --- Scan IR Sensors and update IR. Variables
* OC1Setup -- Setup OC1, OC2, OC3 for PWM
* CALOFF ---- Calculates the necessary offset for the waveform generation
* OC1ISR ---- OC1 Interrupt Service Routine
* AdjPWM ---- Use PWMDC1 and PWMDC2 to establish offsets for PWM
* ICSetup --- Setup IC
* SVIC1  ---- IC1 Interrupt Service Routine
* SVIC2  ---- IC2 Interrupt Service Routine
* SonarTurn -- Turns Thomas towards the detected object
* Scan -- Scans the sonar head through the 180 degrees in front of Thomas and remembers the
*              shortest distance
* Av10Sonar -- Takes 10 sonar readings and averages them
* RTIISR -- RTI Interrupt Service Routine
```

```
                    * RTIINI -- RTI Initialization Routine
                    * Avoid -- Use IRs to avoid all objects encountered
                    * Rotate -- Rotates the Servo Head through 180
                                         degrees in 10 degree increments
                    * Sonar -- Take a single sonar reading
                    * Strait -- Uses Motor Encoders to adjust the PWM
                                   to the motors until the motor encoder
                    *                    counts are equal.
                    * PWMSetup -- Sets up OC1 to control OC2 and 3 for
                                                    motor PWM generation.
                    ***************************************
00DF                        ORG $00DF       ; Pseudo Vector for OC1
                                                                Interrupt
00DF [03] 7E1702            JMP OC1ISR      ; Jump to OC1ISR

00E5                        ORG $00E5       ; Pseudo Vector for IC2
                                                          (IC1 is at $E8)
00E5 [03] 7E172C            JMP SVIC2       ; Jump to SVIC2
00E8 [03] 7E171E            JMP SVIC1       ; Jump to SVIC1

00D3                        ORG $00D3       ; Pseudo Vector for OC5
00D3 [03] 7E14B4            JMP OC5ISR

00D6                        ORG $00D6       ; Pseudo Vector for OC4
00D6 [03] 7E147A            JMP OC4ISR

00EB                        org $00EB       ; Pseudo Vector for RTI
00EB [03] 7E1240            JMP RTIISR

1040                        ORG $1040       ; Variables are at $1040
1040 [03] 7E1092            JMP MAIN        ; Branch to MAIN
                    ****************************************
                    * Define Variables stored in RAM

```

```
1043    14          PAN     FCB !20         ; Pan Servo position (<=SvSteps)
1044    5F          TILT    FCB !95         ; Tilt Servo position (<=SvSteps)
1045    14          NewPan  FCB !20
1046    5F          NewTilt FCB !95
1047    14          HiPan   FCB !20
1048    5F          HiTilt  FCB !95

1049    00BE        SvSteps FDB !190        ; # of Steps for full Servo motion
                                            ; SvSteps Must be => 16

                    * Used by Servoini, OC4ISR, OC5ISR

104B    00          int4    FCB !0          ; Used to determine if OC4 will be Low/High
104C    00          int5    FCB !0          ; Used to determine is OC5 will be Low/High
104D    0240        high4   FDB !576        ; Used by OC4ISR to determine PWM High time
104F    06B8        high5   FDB !1720       ; Used by OC5ISR to determine PWM High time
1051    020E        NewHigh4 FDB !526       ; # of e-clocks for up-time at servo destination
1053    06B8        NewHigh5 FDB !1720      ; # of e-clocks for up-time at servo destination

                    * Variables used by sonar

1055    1B          DDOV    db %00011011    ; Variable used to store bit pattern that is written to
                    *                       ;  DDO Register

```

```
1056      0000          ICnt    FDB $0          ; Holds Initial value of
                                                                  T-Cnt
1058      00            oflow   FCB $0          ; Used to indicate if the
                                                TCNT overflows while waiting
                        *                       ;  for return signal
1059      0000          SonVal  FDB $0          ; Final Sonar Value
105B      0000          HiSonVal FDB $0
105D      0000          ASonVal FDB $0
105F      00            resultf FCB $0          ; Sonar result soft register

                        * Variables used by Rotate

1060      08121C26      fr db   !8,!18,!28,!38,!48,!58,!68,!78,!88,!98,!108,
                                                             !118,!128,!138
          303A444E
          58626C76
          808A
106E      949EA8B2      fr1 db  !148,!158,!168,!178,!188,$ff
          BCFF
1074      00            Times   FCB !0          ; Used to count how Rotate
                                                                  calls
1075      0000          PosPtr  FDB !0          ; Position Pointer
1077      00            EndR    FCB !0          ; Used to indicate the end
                                                          of rotation cycle


1078      00            IR0     db 0    ; Left Back IR.
1079      00            IR1     db 0    ; Left Front IR.
107A      00            IR2     db 0    ; Front Left IR.
107B      00            IR3     db 0    ; Front Center IR.
107C      00            IR4     db 0    ; Front Right IR.
107D      00            IR5     db 0    ; Right Front IR.
107E      00            IR6     db 0    ; Right Back IR.
```

```
107F       63           PWMDC1  FCB !99          ; OC2 % duty cycle RIGHT
                                                                    MOTOR
1080       63           PWMDC2  FCB !99          ; OC3 % duty cycle LEFT
                                                                    MOTOR
1081       B5           PWMP1P  FCB !181         ; Contains # E-cycles for
                                                       0.5% of period (8 bit)
1082       0E34         PWMPER  FDB !3636        ; Contains # E-cycles for
                                                              period (16 bit)
1084       63           STRR    FCB !99          ; Value for strait movement
                                                                    Right
1085       63           STRL    FCB !99          ; Value for strait movement
                                                                     Left
1086       05           IRL     db $5            ; IR. Set point
1087       0000         RMCnt   FDB $0000        ; Right Motor Count
1089       0000         LMCnt   FDB $0000        ; Left Motor Count
108B       00           Dark    db !0            ; Used to store dark IR.
                                                                  reading
108C       0000         RTIs    FDB !0           ; Used by RTIISR to store
                                                            the # OF RTI's
108E       00           AV10LC  FCB !0           ; Used by AV10Sonar to
                                                              store loops
108F       02           DegMult FCB !2           ; Used by SonarTurn
1090       0000         TargetCnt FDB !0         ; Used by SonarTurn for
                                                       Target motor count
                        ***************************************
1092 [03] 8E0FFF        MAIN    LDS #$0fff       ; Initialize Stack
```

```
                     * Setup Variables

1095 [06] 7F1078             CLR IR0
1098 [06] 7F1079             CLR IR1
109B [06] 7F107A             CLR IR2
109E [06] 7F107B             CLR IR3
10A1 [06] 7F107C             CLR IR4
10A4 [06] 7F107D             CLR IR5
10A7 [06] 7F107E             CLR IR6
10AA [02] 861B               LDAA #%00011011
10AC [04] B71055             Staa DDOV
10AF [04] B61084             LDAA STRR
10B2 [04] B7107F             STAA PWMDC1
10B5 [04] B61085             LDAA STRL
10B8 [04] B71080             STAA PWMDC2
10BB [03] CC0E34             LDD #!3636
10BE [05] FD1082             STD PWMPER
10C1 [02] 8605               LDAA #$5
10C3 [04] B71086             STAA IRL
10C6 [03] CC0000             LDD #!0
10C9 [05] FD108C             STD RTIs

                     *       JSR ONSCI

                     * Setup RTI

10CC [06] BD124E             jsr RTIINI

                     * Setup A/D

10CF [06] BD15F9             jSR ADSetup

                     * Setup PWM
```

```
10D2 [06] BD16B6            JSR PWMSetup
                     * Institute PWM
10D5 [06] BD167B            JSR AdjPWM
                     * Initialize Input capture
10D8 [06] BD16F1            JSR ICSetup
                     * Initialize Servos
10DB [06] BD1456            JSR ServoINI
10DE [02] 0E                CLI             ; Enable Global Interrupts
10DF [06] BD14EE     MainLoop JSR MoveHead
10E2 [06] BD125A            JSR Avoid
                     * Check how long we have been doing avoid
10E5 [05] FC108C            LDD RTIs
                     *       CPD #!14500
10E8 [05] 1A831C52          CPD #!7250
10EC [03] 2C03              BGE rot
10EE [03] 7E10DF            JMP MainLoop
10F1 [06] 7F107F     rot    CLR PWMDC1
10F4 [06] 7F1080            CLR PWMDC2
10F7 [06] BD167B            JSR AdjPwm
10FA [06] BD11A9            JSR Scan
10FD [06] BD111B            JSR SonarTurn
1100 [03] CC0000            LDD #!0
1103 [05] FD108C            STD RTIs
1106 [03] 7E10DF            JMP MainLoop
```

```
* GOTO Buffalo

1109 [06] 7F107F     Buf      CLR PWMDC1
110C [06] 7F1080              CLR PWMDC2
110F [06] BD167B              JSR AdjPwm
1112 [03] CC0000              LDD #!0
1115 [05] FD108C              STD RTIs
1118 [03] 7EE0BF              JMP $E0BF        ; goto buffalo
                     ***************************************
                     * SonarTurn Subroutine
                     *
                     * This subroutine turns the robot to point toward the object it has
                     *  detected. This uses PAN to determine the angle
                     *
                     * Input    : None
                     * Output   : None
                     * Calls    :
                     * Destroys : None
                     ****************************************
111B [03] 36         SonarTurn PSHA
111C [03] 37                  PSHB
111D [04] 3C                  PSHX

                     * Determine if Thomas has to turn Right or Left

111E [04] B61047              LDAA HiPAN
1121 [02] 8155                CMPA #$55       ; Value For Strait ahead
1123 [03] 253B                BLO TurnRight   ; if Pan < $55 Turn Right
1125 [03] 2775                BEQ STDone      ; if Pan = $55 go straight ahead
1127 [02] 8055                SUBA #$55       ; if Pan > $55 Turn Left, A = # of degrees to turn
```

```
                     * Calculate final motor count reading
1129 [04] F6108F             LDAB DegMult    ; B = Degree Multiplier
112C [10] 3D                 MUL             ; D = # of Degrees X Degree multiplier
112D [05] FD1090             STD TargetCnt   ; TargetCnt = D
                     * if turning Left, Clr Right Wheel Count and turn right wheel the
                     *   appropriate # of counts
1130 [03] CC0000             LDD #!0         ;
1133 [05] FD1087             STD RMCnt       ; reset right motor count
                     * Setup to turn Left
1136 [04] B61055             LDAA DDOV       ; A = DDOV
1139 [02] 84FC               ANDA #%11111100 ; Mask Motor Direction Bits
113B [02] 8B01               ADDA #%00000001 ; Turn toward Left
113D [04] B71055             STAA DDOV
1140 [04] B70200             STAA DDO
                     * Give power to the motors
1143 [04] B61084             LDAA STRR
1146 [02] 44                 LSRA
1147 [04] B7107F             STAA PWMDC1
114A [04] B61085             LDAA STRL
114D [02] 44                 LSRA
114E [04] B71080             STAA PWMDC2     ; Half Speed
1151 [06] BD167B             JSR AdjPwm
                     * Read motor count and compare until done
```

```
1154 [05] FC1087     CHKLLoop LDD RMCnt
1157 [07] 1AB31090            CPD TargetCnt   ; RMCnt - TargetCnt = ?
115B [03] 2DF7                BLT ChkLLoop     ; If RMCnt < TargetCnt Loop
115D [03] 7E119C              JMP STDone      ; if RMCnt => TargetCnt
                                                                 goto STDone

                     * Turn Right

1160 [02] 8655       TurnRight LDAA #$55       ; if Pan < $55 Turn Right,
1162 [04] B01043              SUBA Pan        ; A = # of degrees to turn

                     * Calculate final motor count reading

1165 [04] F6108F              LDAB DegMult    ; B = Degree Multiplier
1168 [10] 3D                  MUL             ; D = # of Degrees X Degree
                                                                  multiplier
1169 [05] FD1090              STD TargetCnt   ; TargetCnt = D

                     * if turning Right, Clr Right Wheel Count and turn
                                                             right wheel the
                     *   appropriate # of counts

116C [03] CC0000              LDD #!0         ;
116F [05] FD1087              STD RMCnt       ; reset right motor count

                     * Setup to turn Right

1172 [04] B61055              LDAA DDOV       ; A = DDOV
1175 [02] 84FC                ANDA #%11111100 ; Mask Motor Direction Bits
1177 [02] 8A02                ORAA #%00000010 ; Turn toward Right
1179 [04] B71055              STAA DDOV
117C [04] B70200              STAA DDO

                     * Give power to the motors
```

```
117F [04] B61084                LDAA STRR
1182 [02] 44                    LSRA
1183 [04] B7107F                STAA PWMDC1
1186 [04] B61085                LDAA STRL
1189 [02] 44                    LSRA
118A [04] B71080                STAA PWMDC2     ; Half Speed
118D [06] BD167B                JSR AdjPWM

                        * Read motor count and compare until done

1190 [05] FC1087        CHKRLoop LDD RMCnt
1193 [07] 1AB31090              CPD TargetCnt   ; RMCnt - TargetCnt = ?
1197 [03] 2DF7                  BLT ChkRLoop     ; If RMCnt < TargetCnt Loop
1199 [03] 7E119C                JMP STDone      ; if RMCnt => TargetCnt
                                                                 goto STDone


119C [06] 7F107F        STDone  CLR PWMDC1
119F [06] 7F1080                CLR PWMDC2
11A2 [06] BD167B                JSR AdjPwm      ; Stop motors
11A5 [05] 38                    PULX
11A6 [04] 33                    PULB
11A7 [04] 32                    PULA
11A8 [05] 39                    RTS

                        ***************************************
                        * Scan Subroutine
                        *
                        * This subroutine uses the servo head and the sonar
                                                     to scan the area around
                        *  the robot and find the most open space.
                        *
                        * Input    : None
```

```
                       * Output   : update HiPan and HiTilt
                       * Destroys : None
                       * Calls    :
                       ***************************************
11A9 [03] 36           Scan    PSHA
11AA [03] 37                   PSHB
11AB [04] 3C                   PSHX

11AC [04] DE00                 LDX !0
11AE [05] FF105B               STX HiSonVal    ; HiSonVal=0

11B1 [06] BD132A       SMLoop   JSR Rotate     ; Rotate to next position

11B4 [02] C603                 LDAB #$3
11B6 [03] CEFFFF       rw1     LDX #$ffff
11B9 [03] 09           RW      DEX
11BA [03] 26FD                 BNE RW
11BC [02] 5A                   DECB
11BD [03] 26F7                 BNE RW1         ; Wait for servos to settle

11BF [04] B61077               LDAA EndR
11C2 [02] 81FF                 CMPA #$ff
11C4 [03] 2733                 BEQ SDone

                       * Take 10 Sonar Readings and average them and
                                                    determine if it is high

11C6 [06] BD11FD               JSR AV10Sonar

11C9 [05] FE105D       er      LDX ASonVal
11CC [05] FF1059               STX SonVal      ; SonVal = ASonVal

11CF [05] FC1059               LDD SonVal
11D2 [05] 1A83FFFF             CPD #$ffff
```

```
11D6 [03] 27D9              BEQ SMLoop      ; If Sonval = FFFF Continue loop
11D8 [05] 1A830000          cpd #$0000
11DC [03] 27D3              BEQ SMLoop     ; If SonVal = 0000 Continue Loop
11DE [07] 1AB3105B          CPD HiSonVal
11E2 [03] 2503              BLO NewHi      ; if SonVal > HiSonVal goto New Hi
11E4 [03] 7E11B1            JMP SMloop      ; If SonVal <= HiSonVal loop

     * Create new hi value

11E7 [05] FD105B     NewHi  STD HiSonVal
11EA [04] B61045            LDAA NewPAN
11ED [04] B71047            STAA HiPan
11F0 [04] B61046            LDAA newTilt
11F3 [04] B71048            STAA HiTilt
11F6 [03] 7E11B1            JMP SMLoop


11F9 [05] 38         SDone  PULX
11FA [04] 33                PULB
11FB [04] 32                PULA
11FC [05] 39                RTS
     ***************************************
     * Av10Sonar Subroutine
     *
     * This subroutine takes ten sonar readings and averages them together
     *
     * Input    : None
     * Output   : Update ASonVal
     * Calls    : Sonar
```

```
                        * Destroys : None
                        ***************************************
11FD [03] 36            Av10Sonar PSHA
11FE [03] 37                    PSHB
11FF [04] 3C                    PSHX

1200 [03] CE0000                LDX #!0
1203 [05] FF105D                STX ASonVal     ; ASonVal = 0
1206 [02] C60B                  LDAB #!11       ; Loop Count = 11
1208 [04] F7108E                STAB AV10LC     ; AV10LC = 11
120B [04] F6108E        aloop   LDAB AV10LC     ; B = AV10LC
120E [02] 5A                    DECB            ; Loop Count = Loop Count -1
120F [03] 272B                  BEQ Av10Done    ; if Loop Count = 0 go to
                                                                AV10Done
1211 [04] F7108E                STAB AV10LC     ; AV10LC = AV10LC - 1
1214 [06] BD137A                JSR Sonar       ; if Loop Count <> 0 take
                                                            sonar reading

1217 [05] FE1059                LDX Sonval      ; X = Sonar Reading
121A [04] 8C0000                CPX #$0000
121D [03] 27EC                  BEQ aloop       ; If Sonar Reading = 0000
                                                                    Loop
121F [04] 8CFFFF                CPX #$ffff
1222 [03] 27E7                  BEQ aloop       ; If Sonar reading = ffff
                                                                    Loop
1224 [05] FC105D                LDD ASonval     ; If Valid Sonar reading
1227 [06] F31059                ADDD Sonval     ;   D = Sonval + ASonVal
122A [03] 2409                  BCC noc         ; If No Carry go to noc
122C [03] 04                    LSRD            ; If carry, D = (Sonval +
                                                              ASonVal)/2
122D [02] 8A40                  ORAA #%1000000  ; Account for carry
122F [05] FD105D                STD ASonVal     ; Save new average
1232 [03] 7E120B                JMP aloop       ; Loop
1235 [03] 04            noc      LSRD           ; If no carry, D = (SonVla
```

```
                                              + ASonVal)/2
1236 [05] FD105D          STD ASonVal     ; Save new average
1239 [03] 7E120B          JMP aloop       ; Loop

123C [05] 38      AV10Done PULX
123D [04] 33              PULB
123E [04] 32              PULA
123F [05] 39              RTS

                  ***************************************
                  * RTIISR RTI Interrupt Service Routine
                  *
                  * This routine increments a software flag (RTIs)
                                              which is used to time
                  *  how long a particular routine has been running.
                  *
                  * Input    : None
                  * Output   : None
                  * Calls    : None
                  * Destroys : None
                  ***************************************
1240 [05] FE108C  RTIISR  LDX RTIs
1243 [03] 08              INX
1244 [05] FF108C          STX RTIs        ; Increment RTIs

1247 [03] CE1000          LDX #RegBas
124A [07] 1D25BF          BCLR TFLG2,X,%10111111 ; Clear RTI Flag
124D [12] 3B              RTI

                  ***************************************
                  *  RTIINI  RTI Initialization Subroutine
                  *
                  * This Routine initializes the RTI for use.
                  *
```

```
                     * Input    : None
                     * Output   : None
                     * Calls    : None
                     * Destroys : None
                     ***************************************
124E [04] 3C         RTIINI  PSHX

124F [03] CE1000             LDX #RegBas
1252 [07] 1C2440             BSET TMSK2,X,%01000000
1255 [07] 1D25BF             BCLR TFLG2,X,%10111111

1258 [05] 38                 PULX
1259 [05] 39                 RTS

                     ***************************************
                     * Avoid Behavior Subroutine
                     *
                     * This is the Avoid Obstacles Subroutine
                     *  This routine uses the IR. sensors to avoid the
                                                  obstacles it encounters
                     *
                     * Input    : None
                     * Output   : None
                     * Destroys : None
                     * Calls    : AdjPwm, Strait, MoveHead, IRread
                     ***************************************
125A [03] 36         Avoid   PSHA
125B [03] 37                 PSHB
125C [04] 3C                 PSHX
125D [05] 183C               PSHY

                     * Straight

125F [06] BD1566             JSR Strait
```

```
* Read IR. and test forward Sensors (IR2, IR3, IR4)
*   Test IR3 ( Front Center )

1262 [02] C603              LDAB #!3
1264 [06] BD1612            JSR IRRead      ; Read IR3
1267 [04] B6107B            LDAA IR3        ; A = IR3
126A [04] B11086            CMPA IRL        ; Compare A to IRL
126D [03] 2226              BHI Turn        ; If IR3 > IRL goto Turn
126F [06] BD1566            JSR Strait

*   Test IR2 (Front Left)

1272 [02] C602              LDAB #!2
1274 [06] BD1612            JSR IRRead      ; Read IR2
1277 [04] B6107A            LDAA IR2        ; A = IR2
127A [04] B11086            CMPA IRL        ; Compare A to IRL
127D [03] 2244              BHI TL          ; If IR2 > IRL Turn Left
127F [06] BD1566            JSR Strait

*   Test IR4 (Front Right)

1282 [02] C604              LDAB #!4
1284 [06] BD1612            JSR IRRead      ; Read IR4
1287 [04] B6107C            LDAA IR4        ; A = IR4
128A [04] B11086            CMPA IRL        ; Compare A to IRL
128D [03] 2503              BLO TLDone      ; If IR4 < IRL Loop
128F [03] 7E12ED            JMP TR          ; else turn right
1292 [03] 7E1324    TLDone  JMP AvDone

* If one of the test fails Turn Thomas

1295 [02] 8650      Turn    LDAA #$50
1297 [04] B7107F            STAA PWMDC1
```

```
129A [04] B71080              STAA PWMDC2
129D [06] BD167B              JSR AdjPWM      ; Slow Down For Turns

12A0 [02] C601                LDAB #!1
12A2 [06] BD1612              JSR IRRead
12A5 [04] B61079              LDAA IR1        ; A = IR1 (Left Front)
12A8 [04] B11086              CMPA IRL        ; Compare IR1 to IRL
12AB [03] 2240                BHI TR          ; if Left Blocked, Turn
                                                                   Right

12AD [02] C605                LDAB #!5
12AF [06] BD1612              JSR IRRead
12B2 [04] B6107D              ldaa IR5        ; if Left Clear, Check
                                              right, A = IR5 (Right Front)
12B5 [04] B11086              CMPA IRL
12B8 [03] 2209                BHI TL          ; if Left Clear & Right
                                                        Blocked, Turn Left
12BA [03] CE1000              LDX #REGBAS     ; If Left and right Clear,
                                                                 flip coin
12BD [05] EC0E                LDD TCNT,X
12BF [02] C188                cmpb #$88
12C1 [03] 2C2A                Bge TR          ; if MSB of TCNT in Lower
                                                           half turn right
                         *                    ;  Else Turn left.

                         *    If IR1 is clear turn until front is clear

12C3 [02] 860F           TL   LDAA #!15
12C5 [04] B71045              STAA NewPAN
12C8 [02] 8609                LDAA #!9
12CA [04] B71046              STAA NewTilt
12CD [06] BD14EE              JSR MoveHead

12D0 [04] B61055              LDAA DDOV       ; A = DDOV
```

```
12D3 [02] 84FC       589          ANDA #%11111100 ; Mask Motor Direction Bits
12D5 [02] 8B01       590          ADDA #%00000001 ; Turn toward Left
12D7 [04] B71055     591          STAA DDOV
12DA [04] B70200     592          STAA DDO
                     593
12DD [02] C602       594  LLoop   LDAB #!2
12DF [06] BD1612     595          JSR IRRead
12E2 [04] B6107A     596          LDAA IR2        ; A = IR2 (Last IR. to
                                                                    Clear)
12E5 [04] B11086     597          CMPA IRL
12E8 [03] 2AF3       598          BPL  LLoop      ; If IR4 still blocked
                                                               continue turn
                     599
12EA [03] 7E1324     600          JMP AvDone      ; Ir IR4 clear go strait
                     601
                     602  * Turn Right
                     603
12ED [02] 86A5       604  TR      LDAA #!165
12EF [04] B71045     605          STAA NewPAN
12F2 [02] 8609       606          LDAA #!9
12F4 [04] B71046     607          STAA NewTilt
12F7 [06] BD14EE     608          JSR MoveHead
                     609
12FA [04] B61055     610          LDAA DDOV       ; A = DDOV
12FD [02] 84FC       611          ANDA #%11111100 ; Mask Motor Direction Bits
12FF [02] 8B02       612          ADDA #%00000010 ; Turn toward Right
1301 [04] B71055     613          STAA DDOV
1304 [04] B70200     614          STAA DDO
                     615
1307 [02] C604       616  RLoop   LDAB #!4
1309 [06] BD1612     617          JSR IRRead
130C [04] B6107C     618          LDAA IR4        ; A = IR4 (Last IR. to
                                                                    Clear)
130F [04] B11086     619          CMPA IRL
```

```
1312 [03] 22F3             BHI  RLoop      ; If IR2 still blocked
                                                       continue turn

1314 [02] C601             LDAB #!1
1316 [06] BD1612           JSR IRRead
1319 [04] B61079           LDAA IR1        ; A = IR1
131C [04] B1108B           CMPA IRL+5
131F [03] 22E6             BHI RLoop       ; Move away from side wall
1321 [03] 7E1324           JMP AvDone      ; IF IR2 clear go strait

1324 [06] 1838     AvDone  PULY
1326 [05] 38               PULX
1327 [04] 33               PULB
1328 [04] 32               PULA
1329 [05] 39               RTS

                   ***************************************
                   * Rotate Subroutine
                   *
                   * This subroutine rotates the servo head through
                                                 360 degrees in 10 degree
                   *   increments. Rotates to next position each time
                                                             it is called
                   *
                   * Input     : None
                   * Output    : Variable EndR = ff when done rotation
                   * Distroyes : None
                   * Calls     : MoveHead
                   ****************************************
132A [03] 36       Rotate  PSHA
132B [03] 37               PSHB
132C [04] 3C               PSHX

                   * increment rotate counter
```

```
132D [04] B61074           ldaa Times
1330 [02] 4C               inca
1331 [04] B71074           STAA Times
1334 [02] 8101             CMPA #!1
1336 [03] 260E             BNE MRLoop      ; If not 1st time skip initialization

                   * Move tilt into initial position

1338 [02] 8609             ldaa #$09
133A [04] B71046           STAA newtilt    ; Tilt position for right-side-up level
133D [03] CE1060           LDX #fr
1340 [05] FF1075           STX PosPtr      ; Store Initial Position Pointer
1343 [06] 7F1077           CLR Endr

                   * Determine next pan position

1346 [05] FE1075   MRLoop  ldx PosPtr      ; X=address for fr (Pan Front coordinate)
1349 [04] A600             ldaa 0,X        ; A = pan position indicated by X
134B [02] 81FF             CMPA #$ff       ;
134D [03] 2716             BEQ TRDone         ; if A=ff goto test for end
134F [04] B71045           staa NewPan     ; if A<>ff store new pan
1352 [03] 08               INX             ; increment table pointer
1353 [05] FF1075           STX PosPtr      ; Store Next Position
1356 [03] 7E1373           JMP rdone       ; goto inner loop

1359 [04] B61046   TFE     LDAA NewTilt
135C [02] 81BE             CMPA #$be       ;
```

```
135E [03] 2705                BEQ TRDone      ; GOTO totally Done, End of rotation
1360 [02] 86BE                LDAA #$be
1362 [04] B71046              STAA NewTilt    ; Tilt position for upside down level

1365 [03] CE1060      TRDone  LDX #fr
1368 [05] FF1075              STX PosPtr      ; Store Initial Position Pointer
136B [02] 86FF                LDAA #$ff
136D [04] B71077              STAA ENDR       ; Indicate end of rotation
1370 [06] 7F1074              CLR Times       ; Clear Rotate times counter

1373 [06] BD14EE      RDone   JSR MoveHead
1376 [05] 38                  PULX
1377 [04] 33                  PULB
1378 [04] 32                  pula
1379 [05] 39                  RTS
                      ***************************************
                      *       Sonar Subroutine
                      *
                      * This subroutine reads the sonar and updates the sonar variable
                      *
                      * Input    : None
                      * Output   : SonVal
                      * Destroys : None
                      * Calls    : None
                      ***************************************

137A [03] 36          Sonar   PSHA
137B [03] 37                  PSHB
137C [04] 3C                  PSHX
137D [05] 183C                PSHY
```

```
* Initialize

137F [06] 7F1058          clr oflow
1382 [06] 7F105F          CLR resultf
1385 [03] CE0000          LDX #$0
1388 [05] FF1056          STX ICnt
138B [05] FF1059          STX SonVal
138E [03] CE1000          LDX #REGBAS

* Enable IC3

1391 [07] 1D2201          BCLR tmsk1,x,%00000001

* Enable IC3 to trigger on any edge

1394 [07] 1C2103          BSET tctl2,x,%00000011

* Place current value of TCNT into ICnt

1397 [03] CE1000          LDX #REGBAS
139A [05] EC0E            LDD TCNT,X       ; D=TCNT
139C [05] FD1056          STD ICnt         ; TCNT->ICnt

* turn on 40kHz

139F [04] B61055          LDAA DDOV
13A2 [02] 8AE0            ORAA #%11100000 ; Set bits 7,6,5 to address
                                                              of sonar
13A4 [02] 84E7            ANDA #%11100111 ; Clear Bits 4,3 to enable
                                                                40 kHz
13A6 [04] B71055          STAA DDOV
13A9 [04] B70200          STAA DDO
```

```
                         * Wait while pulse is sent

13AC [02] C64F                   LDAB #$004f
13AE [02] 5A             SL1     DECB
13AF [03] 2EFD                   BGT SL1

                         *  Turn off 40kHz

13B1 [04] B61055                 LDAA DDOV
13B4 [02] 841F                   ANDA #%00011111 ; Clear bits 7,6,5
13B6 [04] B71055                 STAA DDOV       ; Turn off all IR.
13B9 [04] B70200                 STAA DDO

                         * Wait for the amplifier to settle

13BC [02] C6FF                   ldab #$00ff
13BE [02] 5A             SL2     DECB
13BF [03] 2EFD                   BGT SL2         ; Wait for circuit to settle

                         * Clear IC3 and TCNT Flags

13C1 [07] 1D23FE                 BCLR TFLG1,X,%11111110
13C4 [07] 1D257F                 BCLR TFLG2,X,%01111111

                         * Look for return flag (IC3) or wait for 2 TCNT
                                                                    Flags

13C7 [02] C602                   LDAB #$2        ; TCNT loop Count = 2
13C9 [04] A623           here    LDAA TFLG1,X    ; A = TFLG1 (Look for
                                                                 Capture)
13CB [02] 46                     RORA            ; Move Bit 0 into the Carry
13CC [03] 251A                   BCS Capture     ; if TFLG1 bit 0 = 1 goto
                                                                  Capture
                         *                       ; if no capture, test for
```

```
                                                                   time-out
13CE [04] A625        772          LDAA TFLG2,X    ; A = TFLG2
13D0 [03] 2AF7        773          BPL here        ; if Bit 7 = 0 continue
                                                                    looking
13D2 [07] 1D257F      774          BCLR TFLG2,X,%01111111 ;if Bit 7 = 1, clear
                                                                  TCNT Flag
                      775
13D5 [02] 5A          776          DECB            ; Decrement Loop count
13D6 [03] 2705        777          BEQ Timeout     ; If loop count = 0 assume
                                                        no return, goto Loop
13D8 [06] 7C1058      778          INC oflow       ; if loop count <> 0
                                                             Increment oflow
13DB [03] 20EC        779          BRA here        ; Continue waiting
                      780
13DD [04] B6105F      781  Timeout ldaa resultf    ; A=Result Flag variable
13E0 [02] 8A01        782          ORAA #%00000001 ; Set time out flag
13E2 [04] B7105F      783          STAA resultf    ; Store resultf
13E5 [03] 7E142C      784          jmp End
                      785
                      786  * If return signal is captured, test for TCNT
                                                                    o-flow.
                      787
13E8 [06] 7D1058      788  Capture tst oflow
13EB [03] 270D        789          BEQ noflow      ; if oflow=0 calculate time
                                                           with no over flow
13ED [03] CCFFFF      790          LDD #$ffff      ; if oflwo<>0 calculate
                                                         time with over flow
                      791
13F0 [06] B31056      792          SUBD ICnt       ; FFFF - ICnt -> D
13F3 [06] E314        793          ADDD TIC3,X     ; D = (FFFF-ICnt)+TIC3
13F5 [05] FD1056      794          STD ICnt
13F8 [03] 200A        795          BRA DBL
                      796
13FA [05] EC14        797  noflow  LDD TIC3,X      ; Load value of TCNT When
```

```
                                                      IC3 Triggered
13FC [06] B31056          SUBD ICnt       ; TIC3 - ICnt -> D
13FF [05] FD1056          STD ICnt        ; ICnt = Timer Difference
                                                i.e. Time of Flight
1402 [03] 2000            BRA DBL

     * Check if reading is to close to be accurate

1404 [05] FC1056     DBL     LDD ICnt        ; D=ICnt
1407 [05] 1A830A00           CPD #$0A00      ; Closest allowable reading
                                                             is 0A00
140B [03] 2C0B               BGE AVGR        ; if ICnt => 0A00 goto AVGR
140D [04] B6105F             LDAA resultf    ; if ICnt < 0A00 set result
                                                                flag
1410 [02] 8A02               ORAA #%00000010 ; set To-Close Flag
1412 [04] B7105F             STAA resultf    ; Store result flag
1415 [03] 7E142C             JMP End        ; continue BigL

     * Average the old and new readings

1418 [05] FC1059     AVGR    LDD SonVal      ; D=Sonval
141B [06] F31056             ADDD ICnt       ; D=sonVal+ICnt
141E [03] 2506               BCS addsome     ; if Carry = 1 the addition
                                                 had a carry that must
     *                                       ;  be accounted for
1420 [05] FD1059             STD SonVal
1423 [03] 7E142C             JMP End

1426 [02] 01         addsome NOP             ;LSRD
1427 [02] 8A80               ORAA #%10000000 ; Set D bit 15 to account
                                                           for carry
1429 [05] FD1059             STD SonVal

     * determine final output
```

```
142C [05] FC1059     End     LDD SonVal
142F [03] 2E1F               BGT OUT         ; If final average > 0 you have a valid reading
1431 [04] B6105F             LDAA resultf    ; if final average = 0 look at result Flags
1434 [02] 44                 LSRA            ; Move To-Far bit into Carry
1435 [03] 2408               BCC TTC         ; If To-Far bit = 0 goto Test To Close
1437 [03] CCFFFF             LDD #$ffff      ; if To-Far bit = 1 D=ffff
143A [05] FD1059             STD SonVal      ; SonVal = ffff to indicate To-Far
143D [03] 2011               BRA OUT

143F [02] 44         TTC     LSRA            ; Move To-Close bit into Carry
1440 [03] 2408               BCC UNK         ; If To-Close bit = 0 goto UNKnown
1442 [03] CC0000             LDD #$0000      ; If To-Close bit = 1 D=0000
1445 [05] FD1059             STD SonVal      ; SonVal = 0000 to indicate To-Close
1448 [03] 2006               BRA OUT

144A [03] CCABCD     UNK     LDD #$ABCD      ; This is the unknown case
144D [05] FD1059             STD SonVal      ; ABCD indicates unknown problem


1450 [06] 1838       OUT     PULY
1452 [05] 38                 PULX
1453 [04] 33                 PULB
1454 [04] 32                 PULA
1455 [05] 39                 rts
                     ***************************************
```

```
                    * Servoini Subroutine
                    *
                    *   Setup OC4 and OC5 for servo control
                    *
                    * Input    : None
                    * Output   : None
                    * Destroys : None
                    * Calls    : None
                    ***************************************
1456 [04] 3C        Servoini PSHX

1457 [03] CE1000            LDX #REGBAS

145A [07] 1D2604            BCLR pactl,X,%00000100  ; Enable OC5

                    * Set OC4 & OC5 to go low

145D [07] 1C200A            BSET TCTL1,X,%00001010
1460 [07] 1D2005            BCLR TCTL1,X,%00000101

                    * Force OC4 & OC5

1463 [07] 1C0B18            BSET CFORC,X,%00011000

                    * Setup OC4 and OC5 to toggle

1466 [07] 1C2005            BSET TCTL1,X,%00000101
1469 [07] 1D200A            BCLR TCTL1,X,%00001010  ; Set OC4 and OC5
                                                            to toggle

                    * Turn on OC4 and OC5 Interrupts

146C [07] 1C2218            BSET TMSK1,X,%00011000

```

```
                  * Clear any old OC4 or OC5 Interrupt flag
146F [07] 1D23E7          BCLR TFLG1,X,%11100111
                  * Clear int4 & int5
1472 [06] 7F104B          CLR int4
1475 [06] 7F104C          CLR int5
1478 [05] 38              PULX
1479 [05] 39              RTS
                  ***************************************
                  *  OC4ISR
                  *
                  *   This ISR Controls OC4 and produces the PWM
                                              signal needed to drive a
                  *      Standard Futaba Servo
                  *
                  * Input    : None
                  * Output   : None
                  * Destroys : None
                  * Calls    : None
                  ***************************************
147A [03] CE1000  OC4ISR  LDX     #REGBAS
147D [04] B6104B          LDAA    int4    ; int4 indicates if OC4 is
                                                     to go low or high
1480 [03] 2623            BNE     off4    ; if int4 <> 0 calculate
                                                              off time
1482 [05] FC104D          LDD     high4   ; if int4 = 0 calculate on
                                               time, D=#-f cycles high
1485 [05] 1A8311F8        cpd     #!4600  ; Highest acceptable value
1489 [03] 2F06            BLE     TLO4    ; if High4<=4300 goto TLO4
148B [03] CC11F8          LDD     #!4600  ; if High4>4300, use 4300
```

```
148E [03] 7E149A            JMP    cont4
1491 [05] 1A830208   TLO4   CPD    #!520   ; Lowest acceptable value
1495 [03] 2C03              bge    cont4   ; if 400<=High4=>4300 the
                                                 number is valid
1497 [03] CC0208            LDD    #!520   ; if High4<400 use 400
149A [06] E31C       cont4  addd   toc4,X  ; D=high4+toc4
149C [05] ED1C              std    toc4,X  ; toc4=high4+toc4
149E [02] 8601              ldaa   #!1     ;
14A0 [04] B7104B            staa   int4    ; int4 <> 0
14A3 [03] 200B              bra    done4
14A5 [03] CC9538     off4   LDD    #$9538  ; D=Standard off time
14A8 [06] E31C              ADDD   toc4,X  ; D= off time + toc4
14AA [05] ED1C              STD    toc4,X  ; toc4=toc4+off time
14AC [02] 4F                CLRA
14AD [04] B7104B            STAA   int4    ; int4 = 0
14B0 [07] 1D23EF     done4  bclr   tflg1,X,%11101111       ; reset OC4
                                                       interrupt
14B3 [12] 3B                RTI
                     ***************************************
                     *  OC5ISR
                     *
                     *   This ISR Controls OC5 and produces the PWM
                                                signal needed to drive a
                     *      Standard Futaba Servo
                     *
                     * Input    : None
                     * Output   : None
                     * Destroys : None
                     * Calls    : None
                     ***************************************
14B4 [03] CE1000     OC5ISR LDX    #REGBAS
14B7 [04] B6104C            LDAA   int5    ; int5 indicates if OC5 is
                                                   to go low or high
14BA [03] 2623              BNE    off5    ; if int5 <> 0 calculate
```

```
                                                      off time
14BC [05] FC104F          LDD     high5   ; if int5 = 0 calculate on
                                               time, D=#-f cycles high
14BF [05] 1A8311F8        cpd     #!4600  ; Highest acceptable value
14C3 [03] 2F06            BLE     TLO5    ; if High5<=4300 goto TLO5
14C5 [03] CC11F8          LDD     #!4600  ; if High5>4300, use 4300
14C8 [03] 7E14D4          JMP     cont5
14CB [05] 1A830208  TLO5  CPD     #!520   ; Lowest acceptable value
14CF [03] 2C03            bge     cont5   ; if 400<=High5=>4300 the
                                                   number is valid
14D1 [03] CC0208          LDD     #!520   ; if High5<400 use 400
14D4 [06] E31E      cont5 addd    toc5,X  ; D=high5+toc5
14D6 [05] ED1E            std     toc5,X  ; toc5=high5+toc5
14D8 [02] 8601            ldaa    #!1     ;
14DA [04] B7104C          staa    int5    ; int5 <> 0
14DD [03] 200B            bra     done5
14DF [03] CC9538    off5  LDD     #$9538  ; D=Standard off time
14E2 [06] E31E            ADDD    toc5,X  ; D= off time + toc5
14E4 [05] ED1E            STD     toc5,X  ; toc5=toc5+off time
14E6 [02] 4F              CLRA
14E7 [04] B7104C          STAA    int5    ; int5 = 0
14EA [07] 1D23F7    done5 bclr    tflg1,X,%11110111       ; reset OC5
                                                         interrupt
14ED [12] 3B              RTI

                    ***************************************
                    * MoveHead Subroutine
                    *
                    *  This subroutine moves the Pan/Tilt servo head to
                                                               its
                    *   new position SMOOTHLY
                    *
                    * Input: New head position is stored in NewPan and
                                                           NewTilt
```

```
                 * Output: The servo head is moved and the final
                                              position is stored in
                 *          PAN and TILT
                 * Destroys: The old values in PAN and TILT are
                                           replaced with the values
                 *            in NewPan and NewTilt
                 * Calls: ServoCalc
                 ***************************************
14EE [03] 36     MoveHead PSHA
14EF [03] 37              PSHB
14F0 [04] 3C              PSHX

                 * Disable IC Interrupts

14F1 [03] CE1000          LDX #REGBAS

14F4 [07] 1D2207          BCLR tmsk1,x,%00000111

                 * Calculate New Servo position High times

14F7 [04] B61045          LDAA NewPAN
14FA [06] BD1555          JSR ServoCalc
14FD [05] FD1051          STD NewHigh4
1500 [04] B61046          ldaa NewTILT
1503 [06] BD1555          JSR ServoCalc
1506 [05] FD1053          STD NewHigh5

                 * Wait

1509 [03] CE0100 MHLoop  LDX #$0100
150C [03] 09     MHL1    DEX
150D [03] 26FD           BNE MHL1

                 * Adjust Pan High Time
```

```
150F [05] FC104D             LDD High4
1512 [07] 1AB31051           CPD NewHigh4    ; High4-NewHigh4=?
1516 [03] 270F               BEQ TTilt       ; if High4=NewHigh4 goto
                                                                  TTilt
1518 [03] 2D08               BLT Pandn       ; if High4>NewHigh4 goto
                                                                  pandn
151A [03] 8F                 XGDX            ; if High4<NewHigh4 X=High4
151B [03] 09                 DEX             ; X=High4-1
151C [05] FF104D             STX High4
151F [03] 7E1527             JMP TTilt
1522 [03] 8F         PanDn   XGDX            ; X=High4
1523 [03] 08                 INX             ; X=High4+1
1524 [05] FF104D             STX High4
1527 [05] FC104F     TTilt   LDD High5
152A [07] 1AB31053           CPD NewHigh5    ; High5-NewHigh5=?
152E [03] 2712               BEQ TDone       ; if High5=NewHigh5 goto
                                                                  TDone
1530 [03] 2D08               BLT Tiltdn      ; if High5>NewHigh5 goto
                                                                  pandn
1532 [03] 8F                 XGDX            ; if High5<NewHigh5 X=High5
1533 [03] 09                 DEX             ; X=High5-1
1534 [05] FF104F             STX High5
1537 [03] 7E1509             JMP MHLoop
153A [03] 8F         TiltDn  XGDX            ; X=High5
153B [03] 08                 INX             ; X=High5+1
153C [05] FF104F             STX High5
153F [03] 7E1509             JMP MHLoop
1542 [05] FC104D     TDone   LDD High4
1545 [07] 1AB31051           CPD NewHigh4
1549 [03] 26BE               BNE MHLoop
```

```
                       * enable IC Interrupts

154B [03] CE1000       TCDone  LDX #REGBAS

154E [07] 1C2206               BSET tmsk1,x,%00000110

1551 [05] 38                   PULX
1552 [04] 33                   PULB
1553 [04] 32                   PULA
1554 [05] 39                   RTS


                       ***************************************
                       * ServoCalc
                       *
                       * This subroutine calculates the offset necessary for correct servo
                       *  placement
                       *
                       * NOTE *** This routine is only valid for SvSteps => 16
                       *
                       * Input:    A = Desired servo position
                       * Output:   D = Total Hi-Time offset for servo position
                       * Destroys: B
                       * Calls:    None
                       ***************************************

1555 [04] 3C           ServoCalc PSHX
1556 [03] 36                   PSHA            ; A = Desired Servo position

1557 [03] CC1130               LDD #!4400      ; D=# of E-cycles for complete Servo Movement
```

```
155A [05] FE1049          LDX SvSteps     ; X=# of Steps for complete
                                                      Servo Movement
155D [41] 02              IDIV            ; X=# of E-Cycles per Step
155E [03] 8F              XGDX            ; B=# of E-Cycles per Step
155F [04] 32              PULA            ; A=Desired Servo Position
1560 [10] 3D              MUL             ; D=Calculated offset for
                                                      servo position
1561 [04] C30208          ADDD #!520      ; D=Total Hitime offset for
                                                      servo position
1564 [05] 38              PULX
1565 [05] 39              RTS
                  ***************************************
                  * Strait Subroutine
                  *
                  * This subroutine is called when Thomas is going
                                              straight.  It uses the
                  *  motor encoders to determine if Thomas is going
                                                  straight and makes
                  *  adjustments accordingly.
                  ***************************************
1566 [03] 36      Strait  PSHA
1567 [03] 37              PSHB
1568 [04] 3C              PSHX
1569 [04] B61084          LDAA STRR
156C [04] B7107F          STAA PWMDC1
156F [04] B61085          LDAA STRL
1572 [04] B71080          STAA PWMDC2     ; Full Speed Ahead!
1575 [02] 8609            LDAA #$09
1577 [04] B71046          STAA NewTilt
```

```
157A [02] 8655              ldaa #$55
157C [04] B71045            STAA NewPAN
157F [06] BD14EE            JSR MoveHead    ; look strait ahead

1582 [04] B61055            LDAA DDOV       ; A = DDOV
1585 [02] 84FC              ANDA #%11111100 ; Mask Motor Direction
1587 [02] 8A03              ORAA #%00000011 ; Go Strait
1589 [04] B71055            STAA DDOV
158C [04] B70200            STAA DDO        ; Go Strait

                      * Compare RMCnt with LMCnt

158F [05] FC1087            LDD RMCnt
1592 [07] 1AB31089          CPD LMCnt
1596 [03] 2734              BEQ RL100
1598 [03] 2D19              BLT DecRt


                      * If RMCnt > LMCnt decrement STRR & increment STRL


159A [06] 7C1085            INC STRL
159D [02] 8664              ldaa #!100
159F [04] B11085            cmpa STRL
15A2 [03] 2E28              BGT RL100       ; If 100 > STRL continue
15A4 [02] 8663              LDAA #!99       ; else set STRL = 99
15A6 [04] B71085            STAA STRL

15A9 [06] 7A1084            DEC STRR
15AC [03] 2E1E              BGT RL100       ; If STRR >0 continue
15AE [02] 8601              LDAA #!1        ; else set STRR = 1
15B0 [04] B71084            STAA STRR

                      * If RMCnt < LMCnt increment STRR & decrement STRL
```

```
15B3 [06] 7C1084    DecRt   INC STRR
15B6 [02] 8664              ldaa #!100
15B8 [04] B11084            cmpa STRR
15BB [03] 2E0F              BGT RL100       ; If 100 > STRR continue
15BD [02] 8663              LDAA #!99       ; else set STRR = 99
15BF [04] B71084            STAA STRR

15C2 [06] 7A1085            DEC STRL
15C5 [03] 2E05              BGT RL100       ; If STRL >0 continue
15C7 [02] 8601              LDAA #!1        ; else set STRL = 1
15C9 [04] B71085            STAA STRL

                    * Check if both motors are below 99%, if so
                                                       increment both.

15CC [02] 8699      RL100   ldaa #%99
15CE [04] B11084            CMPA STRR
15D1 [03] 2F0B              BLE CStr        ; If STRR => 99 continue
                                                                strait
15D3 [04] B11085            CMPA STRL
15D6 [03] 2F06              BLE CStr        ; If STRL => 99 continue
                                                                strait
15D8 [06] 7C1084            INC STRR
15DB [06] 7C1085            INC STRL

                    * Clear motor counts and re-enable local interrupts

15DE [02] 4F        CStr        CLRA
15DF [02] 5F                CLRB
15E0 [05] FD1087            STD RMCnt
15E3 [05] FD1089            STD LMCnt
```

```
                    * Implement changed parameters

15E6 [04] B61084            LDAA STRR
15E9 [04] B7107F            STAA PWMDC1
15EC [04] B61085            LDAA STRL
15EF [04] B71080            STAA PWMDC2
15F2 [06] BD167B            JSR AdjPWM      ; Full Speed Ahead!

15F5 [05] 38        StrEnd  PULX
15F6 [04] 33                PULB
15F7 [04] 32                PULA
15F8 [05] 39                RTS
                    ***************************************
                    * ADSetup Subroutine
                    *
                    * This subroutine sets up the A/D system
                    ***************************************
15F9 [03] 36        ADSetup PSHA
15FA [04] 3C                PSHX

15FB [03] CE1000            LDX #REGBAS     ; X = Register Offset
15FE [07] 1C3980            BSET OPTION,X,%10000000 ; Power up A/D

                    * Delay for 32.77mS

1601 [07] 1D257F            BCLR TFLG2,X,%01111111 ; Clear Timer
                                                          Overflow Flag
1604 [04] E625      AFset1  LDab TFLG2,X    ; Wait for TFLG2 to set
1606 [03] 2AFC              BPL AFset1      ; IF TFLG2 IS POSITIVE, BIT
                                                         7=0, GOTO Fset
                    *                       ;  IF TFLG2 IS NEGATIVE,
                                                      BIT 7=1, CONTINUE
1608 [07] 1D257F            BCLR TFLG2,X,%01111111 ; Clear Timer
                                                          Overflow Flag
```

```
160B [04] E625    AFset2  LDAB TFLG2,X    ; Wait for TFLG2 to set
160D [03] 2AFC            BPL AFset2      ; IF TFLG2 IS POSITIVE, BIT 7=0, GOTO Fset

160F [05] 38              PULX
1610 [04] 32              PULA
1611 [05] 39              RTS


                  ***************************************
                  * IRRead Subroutine
                  *
                  * This subroutine takes readings from the IR# that was passed to
                  *  it in B.  It reads the corresponding sensor and updates the
                  *  associated IR. Variable.
                  *
                  * This routine takes a differential reading, i.e. It takes a dark
                  *  reading, then it takes a light reading, then it subtracts the
                  *  two, and returns the difference.
                  *
                  * Input    : B=IR#
                  * Output   : IR# variable is updated
                  * Destroys : None
                  * Calls    : None
                  ***************************************
1612 [03] 36      IRRead  PSHA
1613 [04] 3C                      PSHX
1614 [05] 183C                    PSHY
1616 [03] 37                      PSHB

```

```
1617 [03] CE1000    1218          LDX #REGBAS     ; X = Register Offset
                    1219
                    1220  * Take Dark A/D Reading
                    1221  *   IF Reading IR0 Setup to read AD7.
                    1222
161A [03] 2602      1223          BNE ReadAD      ; if IR# <> 0 Read A/D
161C [02] C607      1224          LDAB #!7        ; if IR# = 0 Setup to read
                                                                      A/D 7
                    1225
161E [04] E730      1226  ReadAD  STAB ADCTL,X    ; Start A/D on IR.
                    1227
                    1228
1620 [04] A630      1229  ADLoop1 LDAA ADCTL,X    ; Look for CCF
                    1230
1622 [03] 2AFC      1231          BPL  ADLoop1    ; if CCF = 0 Loop
1624 [04] A631      1232          LDAA ADR1,X     ; Load Reading into A
1626 [04] B7108B    1233          STAA Dark       ; Store dark reading in Dark
                    1234
                    1235  * turn on Appropriate IR. Emitter
                    1236
1629 [02] 58        1237          lslb
162A [02] 58        1238          lslb
162B [02] 58        1239          lslb
162C [02] 58        1240          lslb
162D [02] 58        1241          lslb            ; Move Bit # into position
162E [04] B61055    1242          LDAA DDOV       ; A = DDO Variable
1631 [02] 8403      1243          ANDA #%00000011 ; Mask IR. Control Bits,
                                                              BIT 4=0 (TURNS ON
                    1244  *                                       40kHz)
1633 [02] 1B        1245          ABA             ; A+B=A = New DDO Variable
1634 [04] B71055    1246          STAA DDOV       ; Store New DDO Variable to
                                                                       DDOV
                    1247
                    1248  * Delay for 32.77mS
```

```
1637 [07] 1D257F            BCLR TFLG2,X,%01111111
163A [04] E625      Fset1   LDab TFLG2,X    ; Wait for TFLG2 to set
163C [03] 2AFC              BPL Fset1       ; IF TFLG2 IS POSITIVE, BIT
                                                              7=0, GOTO Fset
                    *                       ;  IF TFLG2 IS NEGATIVE,
                                                           BIT 7=1, CONTINUE
163E [04] B70200            STAA DDO        ; turn on 8-line demux
1641 [07] 1D257F            BCLR TFLG2,X,%01111111
1644 [04] E625      Fset2   LDAB TFLG2,X    ; Wait for TFLG2 to set
1646 [03] 2AFC              BPL Fset2       ; IF TFLG2 IS POSITIVE, BIT
                                                              7=0, GOTO Fset

                    * Take Light A/D Reading

1648 [04] 33                PULB    ; B=IR#
1649 [03] 37                PSHB

                    *   IF Reading IR0 Setup to read AD7.

164A [03] 2602              BNE ReadADD      ; if IR# <> 0 Read A/D
164C [02] C607              LDAB #!7        ; if IR# = 0 Setup to read
                                                                       A/D 7

164E [04] E730      ReadADD  STAB ADCTL,X    ; Start A/D on IR.

1650 [04] A630      ADLoop2 LDAA ADCTL,X    ; Look for CCF

1652 [03] 2AFC              BPL  ADLoop2    ; if CCF = 0 Loop
1654 [04] A631              LDAA ADR1,X     ; Load Reading into A

                    * Turn off IR.

1656 [04] F61055            LDAB DDOV       ; B = DDO Variable
```

```
1659 [02] C403              ANDB #%00000011 ; Mask IR. address and
                                                  motor Control Bits
165B [02] CA18              ORAB #%00011000 ; SET BIT3=1 & Bit 4=1
                                                    (Demux->Clear,
                    *                                 40kHz->off)
165D [04] F71055            STAB DDOV       ; B -> DDOV
1660 [04] F70200            STAB DDO        ; Turn off all IR.

                    * Store Reading into Corresponding Variable

1663 [03] CE1078            LDX #IR0        ; X = Address of IR0
1666 [04] 33                PULB            ; B = IR#
1667 [02] 5D        ADLoop3 TSTB            ; B = IR. #
1668 [03] 2704              BEQ ADone       ; If B = 0 GOTO ADone
166A [03] 08                INX             ; if B <> 0 IncX
166B [02] 5A                DECB            ; DEC B
166C [03] 20F9              BRA ADLoop3     ; Loop

166E [04] B0108B    ADone   SUBA Dark       ; A = Light - Dark
1671 [03] 2A01              BPL ok          ; if Light - Dark => 0 goto
                                                                ok
1673 [02] 4F                CLRA            ; if Light - Dark < 0 it = 0
1674 [04] A700      ok      STAA 0,X        ; A -> Appropriate IR.
                                                          Variable

1676 [06] 1838              PULY
1678 [05] 38                PULX
1679 [04] 32                PULA
167A [05] 39                RTS

                    ***************************************
                    * AdjPWM Subroutine
                    *
                    * This subroutine takes the %duty cycle values
```

```
                                   stored in PWMDC1, PWMDC2,
                 *  PAN, and TILT and determines the offset in
                                E-Cycles for each one. It then
                 *  institutes the change in the PWM system.
                 *
                 * Input    : None
                 * Output   : None
                 * Destroys : None
                 * Calls    : None
                 ***************************************
167B [03] 36     AdjPWM  PSHA
167C [03] 37             PSHB
167D [04] 3C             PSHX

167E [03] CE1000         LDX #REGBAS

                 * Adjust for duty cycles > 50%

1681 [02] 5F             CLRB            ; B=0
1682 [04] B6107F         LDAA PWMDC1     ; A=OC2 % of duty cycle
1685 [02] 8132           CMPA #!50       ; See if % of duty cycle >
                                                                50%
1687 [03] 2302           BLS ARNZ61      ; If % of duty cycle <= 50%
                                           branch to OC3 duty cycle
1689 [02] CB40           ADDB #%01000000 ; If % of duty cycle > 50%
                                                       B=0+01000000
168B [04] B61080 ARNZ61  LDAA PWMDC2     ; A=OC3 % of duty cycle
168E [02] 8132           CMPA #!50       ; See if % of duty cycle >
                                                                50%
1690 [03] 2302           BLS ARNZ62      ; If % of duty cycle <= 50%
                                                           continue
1692 [02] CB20           ADDB #%00100000 ; If % of duty cycle > 50%
                                                       B=B+00100000
```

```
1694 [04] E70D      ARNZ62  STAB OC1D,X     ; B->OC1D set the bits that are to go high at next
                    *                       ;  interrupt.

                    * Determine the length of 1 period in E-Cycles and store in PWMPER and TOC1

1696 [04] B61081            LDAA PWMP1P     ; A=the number of cycles for 1/2 of 1% of the period
1699 [02] C664              LDAB #!100      ; B=100
169B [10] 3D                MUL             ; D=AXB D=1/2 the number of cycles for 1 period
169C [03] 05                LSLD            ; Multiply by 2, D=number of cycles for 1 period
169D [05] FD1082            STD PWMPER      ; D->PWMPER
16A0 [05] ED16              STD TOC1,X      ; D->TOC1

                    * Determine the Offset count for OC2, OC3, and store in TOC2, TOC3
                    *  respectively.

16A2 [04] B6107F            LDAA PWMDC1     ; A=% duty cycle for OC2
16A5 [06] BD16DD            JSR CALOFF      ; Change % duty cycle (A) to Offset Cnt (D)
16A8 [05] ED18              STD TOC2,X      ; D->TOC2 TOC2=Offset Cnt for OC2
16AA [04] B61080            LDAA PWMDC2     ; A=% duty cycle for OC3
16AD [06] BD16DD            JSR CALOFF      ; Change % duty cycle (A) to Offset Cnt (D)
16B0 [05] ED1A              STD TOC3,X      ; D->TOC3 TOC3=Offset Cnt for OC3

16B2 [05] 38                PULX
16B3 [04] 33                PULB
```

```
16B4 [04] 32               PULA
16B5 [05] 39               RTS

                   ***************************************
                   * PWMSetup Subroutine
                   *
                   * This subroutine sets up OC1, OC2, OC3 for use as Pulse Width
                   *  modulation.  OC1 controls OC2 - OC3.
                   * Input    : None
                   * Output   : None
                   * Destroys : None
                   * Calls    : None
                   ***************************************
16B6 [03] 36       PWMSetup PSHA
16B7 [04] 3C               PSHX

16B8 [03] CE1000           LDX #REGBAS             ; X=Register offset

                   * Setup OC1, OC2, OC3,

16BB [07] 1C2050           BSET TCTL1,X,%01010000
16BE [07] 1D20A0           BCLR TCTL1,X,%10100000  ; Set OC2 & OC3 to toggle

16C1 [07] 1C0CE0           BSET OC1M,X,%11100000
16C4 [07] 1D0C1F           BCLR OC1M,X,%00011111

                   * Enable interrupts for OC1

16C7 [07] 1D237F           BCLR TFLG1,X,%01111111  ;Clear local OC1 Interrupt

16CA [07] 1C2280           BSET TMSK1,X,%10000000  ; Enable local OC1
```

```
                                                          interrupts

16CD [04] B61055          ldaa DDOV               ; A = DDOV Bit
                                                             pattern
16D0 [02] 84FC            ANDA #%11111100         ; Mask out the
                                                 Motor directions bits
16D2 [02] 8A03            ORAA #%00000011         ; Set bits to make
                                                 both wheels go forward
16D4 [04] B71055          STAA DDOV               ; A -> DDOV
                                             (Digital Data Out Variable)
16D7 [04] B70200          STAA DDO                ; A -> DDO (Digital
                                                             Data Out)

16DA [05] 38              PULX
16DB [04] 32              PULA
16DC [05] 39              RTS

                    ***************************************
                    * CALOFF - Subroutine
                    * Description: This subroutine changes the duty
                                                 cycle to offset count.
                    * If duty < 50% ($32) change to 100-duty
                    * If duty > 100% ($64) force to 100% ($64)
                    * multiply by 1% of period
                    *
                    * Input    : A=%Duty Cycle
                    * Output   : D=Offset Count
                    * Destroys : B
                    * Calls    : None
                    ***************************************
                    * Determine if A is between 50 and 100, if A<50,
                                               A=100-A, if A>100, A=100

16DD [02] 8132      CALOFF  CMPA #!50       ; Compare A to 50
```

```
16DF [03] 2404            BHS ARN6A       ; If A=>50 branch
16E1 [02] 16              TAB             ; If A<50 A->B, (B is lost)
16E2 [02] 8664            LDAA #!100      ; A=100
16E4 [02] 10              SBA             ; A=A-B
16E5 [02] 8164      ARN6A CMPA #!100      ; Compare A to 100
16E7 [03] 2302            BLS ARN6B       ; If A<=100 branch
16E9 [02] 8664            LDAA #!100      ; If A>100, A=100

                    * Calculate the number of cycles until a state
                                                  change is needed.

16EB [04] F61081    ARN6B LDAB PWMP1P     ; B=# cycles for 1/2 of 1%
                                                          of period
16EE [10] 3D              MUL             ; D=1/2 # cycles until
                                                       state change
16EF [03] 05              LSLD            ; D=D*2 D=# cycles until
                                                       state change
16F0 [05] 39              RTS

                    **************************************************
                    * ICSetup Subroutine
                    *
                    * This subroutine sets up the input capture to
                                            catch the pulses being sent
                    *  to it from the motor encoders.
                    **************************************************
16F1 [03] 36        ICSetup PSHA
16F2 [04] 3C              PSHX

16F3 [03] CE1000          LDX #REGBAS     ; X=Register offset

                    * Setup IC1, IC2 for keeping track of Left and
                                                   Right Motor Data

```

```
16F6 [07] 1C213C            BSET TCTL2,X,%00111100  ; Select input
                                                     capture of both edges
                    *                                ;   for IC1 & IC2

                    * Enable Local interrupts for  IC1, IC2

16F9 [07] 1D23F9            BCLR TFLG1,X,%11111001  ; Clear local IC1,
                                                             IC2 Interrupt

16FC [07] 1C2206            bset TMSK1,X,%00000110  ; Enable local IC1,
                                                            IC2 interrupts

16FF [05] 38                PULX
1700 [04] 32                PULA
1701 [05] 39                RTS

                    ***************************************
                    * OC1ISR - OUTPUT COMPARE 1 SERVICE ROUTINE
                    * Description: The OC1 Interrupt service Routine
                                                        does the following
                    *  a) Calculate next compare value for OC1 and
                                                             store in TOC1
                    *  b) Calculate next compare value for OC2 and
                                                             store in TOC2
                    *  c) Calculate next compare value for OC3 and
                                                             store in TOC3
                    *  e) Clear Flag
                    * Input    : None
                    * Output   : None
                    * Destroys : None
                    * Calls    : None
                    ***************************************

1702 [03] CE1000    OC1ISR  LDX #REGBAS      ; X=Register offset
```

```

* Calculate Next value for TOC1

1705 [05] EC16             LDD TOC1,X      ; D=TOC1 (last OC1 Compare Value)
1707 [06] F31082           ADDD PWMPER     ; D=(last OC1 compare value) + (# cycles for 1 period)
170A [05] ED16             STD TOC1,X      ; D->TOC1 (Next OC1 compare value)

* Calculate Next value for TOC2

170C [05] EC18             LDD TOC2,X      ; D=TOC2 (last OC2 Compare Value)
170E [06] F31082           ADDD PWMPER     ; D=(last OC2 Compare Value) + (# cycles for 1 period)
1711 [05] ED18             STD TOC2,X      ; D->TOC2 (Next OC2 Compare Value)

* Calculate Next value for TOC3

1713 [05] EC1A             LDD TOC3,X      ; D=TOC3 (last OC3 Compare Value)
1715 [06] F31082           ADDD PWMPER     ; D=(last OC3 compare value) + (# cycles for 1 period)
1718 [05] ED1A             STD TOC3,X      ; D->TOC3 (Next OC3 Compare Value)

171A [07] 1D237F           BCLR TFLG1,X,%01111111  ; Clear OC1 Interrupt
171D [12] 3B               RTI

     ****************************************************
     ***************************
```

```
                     * SVIC1 - IC1 Interrupt Service Routine
                     *
                     * Description: This Service Routine keeps track of Right Motor Data
                     *
                     * Input    : None
                     * Output   : None
                     * Destroys : None
                     * Calls    : None
                     ****************************************************************************
171E [05] FE1087     SVIC1   LDX RMCnt
1721 [03] 08                 INX
1722 [05] FF1087             STX RMCnt

                     * Clear interrupts for IC1

1725 [03] CE1000             LDX #REGBAS
1728 [07] 1D23FB             BCLR TFLG1,X,%11111011  ; Clear local IC1 Interrupt

172B [12] 3B                 RTI

                     ****************************************************************************
                     * SVIC2 - IC2 Interrupt Service Routine
                     *
                     * Description: This Service Routine keeps track of Left Motor Data
                     *
                     * Input    : None
                     * Output   : None
                     * Destroys : None
                     * Calls    : None
```

```
                        *****************************************************************************
172C [05] FE1089        SVIC2   LDX LMCnt
172F [03] 08                    INX
1730 [05] FF1089                STX LMCnt

                        * Clear interrupts for IC2

1733 [03] CE1000                LDX #REGBAS
1736 [07] 1D23FD                BCLR TFLG1,X,%11111101  ; Clear local IC2 Interrupt

1739 [12] 3B                    RTI
```

Symbol Table

ADCTL	0030
ADDSOME	1426
ADJPWM	167B
ADLOOP1	1620
ADLOOP2	1650
ADLOOP3	1667
ADONE	166E
ADR1	0031
ADR2	0032
ADR3	0033
ADR4	0034
ADSETUP	15F9
AFSET1	1604

AFSET2	160B
ALOOP	120B
ARN6A	16E5
ARN6B	16EB
ARNZ61	168B
ARNZ62	1694
ASONVAL	105D
AV10DONE	123C
AV10LC	108E
AV10SONAR	11FD
AVDONE	1324
AVGR	1418
AVOID	125A
BUF	1109
CALOFF	16DD
CAPTURE	13E8
CFORC	000B
CHKLLOOP	1154
CHKRLOOP	1190
CONT4	149A
CONT5	14D4
CSTR	15DE
DARK	108B
DBL	1404
DDO	0200
DDOV	1055
DECRT	15B3
DEGMULT	108F
DONE4	14B0
DONE5	14EA
END	142C
ENDR	1077
ER	11C9
FR	1060

FR1	106E
FSET1	163A
FSET2	1644
HERE	13C9
HIGH4	104D
HIGH5	104F
HIPAN	1047
HISONVAL	105B
HITILT	1048
ICNT	1056
ICSETUP	16F1
INT4	104B
INT5	104C
IR0	1078
IR1	1079
IR2	107A
IR3	107B
IR4	107C
IR5	107D
IR6	107E
IRL	1086
IRREAD	1612
LLOOP	12DD
LMCNT	1089
MAIN	1092
MAINLOOP	10DF
MHL1	150C
MHLOOP	1509
MOVEHEAD	14EE
MRLOOP	1346
NEWHI	11E7
NEWHIGH4	1051
NEWHIGH5	1053
NEWPAN	1045

NEWTILT	1046
NOC	1235
NOFLOW	13FA
OC1D	000D
OC1ISR	1702
OC1M	000C
OC4ISR	147A
OC5ISR	14B4
OFF4	14A5
OFF5	14DF
OFLOW	1058
OK	1674
OPTION	0039
OUT	1450
PACTL	0026
PAN	1043
PANDN	1522
POSPTR	1075
PWMDC1	107F
PWMDC2	1080
PWMP1P	1081
PWMPER	1082
PWMSETUP	16B6
RDONE	1373
READAD	161E
READADD	164E
REGBAS	1000
RESULTF	105F
RL100	15CC
RLOOP	1307
RMCNT	1087
ROT	10F1
ROTATE	132A
RTIINI	124E

RTIISR	1240
RTIS	108C
RW	11B9
RW1	11B6
SCAN	11A9
SDONE	11F9
SERVOCALC	1555
SERVOINI	1456
SL1	13AE
SL2	13BE
SMLOOP	11B1
SONAR	137A
SONARTURN	111B
SONVAL	1059
STDONE	119C
STRAIT	1566
STREND	15F5
STRL	1085
STRR	1084
SVIC1	171E
SVIC2	172C
SVSTEPS	1049
TARGETCNT	1090
TCDONE	154B
TCNT	000E
TCTL1	0020
TCTL2	0021
TDONE	1542
TFE	1359
TFLG1	0023
TFLG2	0025
TIC3	0014
TILT	1044
TILTDN	153A

TIMEOUT	13DD
TIMES	1074
TL	12C3
TLDONE	1292
TLO4	1491
TLO5	14CB
TMSK1	0022
TMSK2	0024
TOC1	0016
TOC2	0018
TOC3	001A
TOC4	001C
TOC5	001E
TR	12ED
TRDONE	1365
TTC	143F
TTILT	1527
TURN	1295
TURNRIGHT	1160
UNK	144A