A Miniature Robot

Final Project Report
December 9, 1999
E157

Ronalee Lo and Dave Beydler

Abstract

Robots not only aid humans with day-to-day and difficult tasks, but they can also provide people with entertainment. For example, Sony's Aibo is a popular robotic dog. The purpose of this project was to create basic functionality for a wheeled robot that could be applied to either facet of robots. Since robotics combines the disciplines of both mechanics and electronics (and increasingly computer programming), and since this class is aimed for understanding microprocessors, the project is concentrated on the electrical aspects of making robots.

1. Introduction

The purpose of this project was to design and implement a miniature, wheeled robot capable of forward and turning (both left and right) motions based on sensory inputs. Using these degrees of freedom, the robot is able to navigate using the "follow-the-wall" algorithm, if it reaches an immovable object. The main focuses of this project were not only to create this algorithm, but also to write and test the code necessary to handle the I/O of the robot. A prototype was built to apply the code to a physical setting.

2. New Hardware

2.1. Servo Motors

Futaba S148 servo motors were used to move the prototype. Servo motors are normally controlled via a pulse width modulation (PWM) signal. The S148 takes three inputs: power, ground, and a PWM signal. The high-time length for each period, as opposed to the duty cycle, determines the position of the S148. See Fig. 1 for a depiction of a PWM signal.

Since these servo motors were limited to 180 ° of rotation, modifications had to be made to allow for a full, 360 ° range. The internal potentiometer was bypassed by removing both the potentiometer stop and drive plate (see Appendix A for schematics of the Futaba S148 servo motor). When the potentiometer is bypassed, the feedback system that compares the potentiometer readings with the incoming signal continues to run. Thus, a constant high-

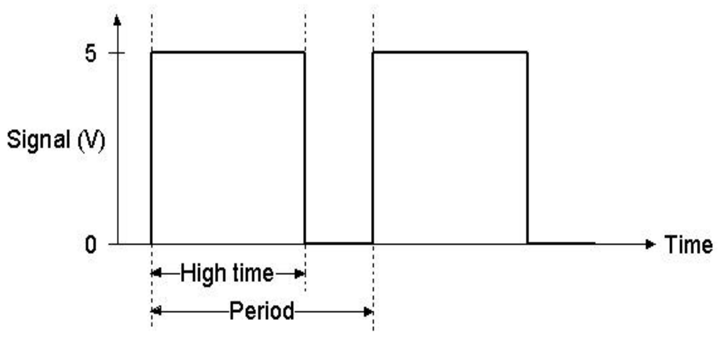

Figure 1. A pulse width modulation signal.

time length results in rotation at a constant speed (the maximum is approximately 60° in 0.22 secs).

3. Schematics

3.1 HC11 I/O Signals

This project was implemented using minimal connections. The code was written in such a way that only two inputs are needed and two outputs are produced. The two sensor inputs are loaded into A/D converters on the HC11. The code calculates the proper response for the motors and sends out the corresponding PWM signal. Most of the connections were made using wire wrap because it made a good connection while allowing for easy modification. The following diagram (Fig. 2) is a schematic of the connections.

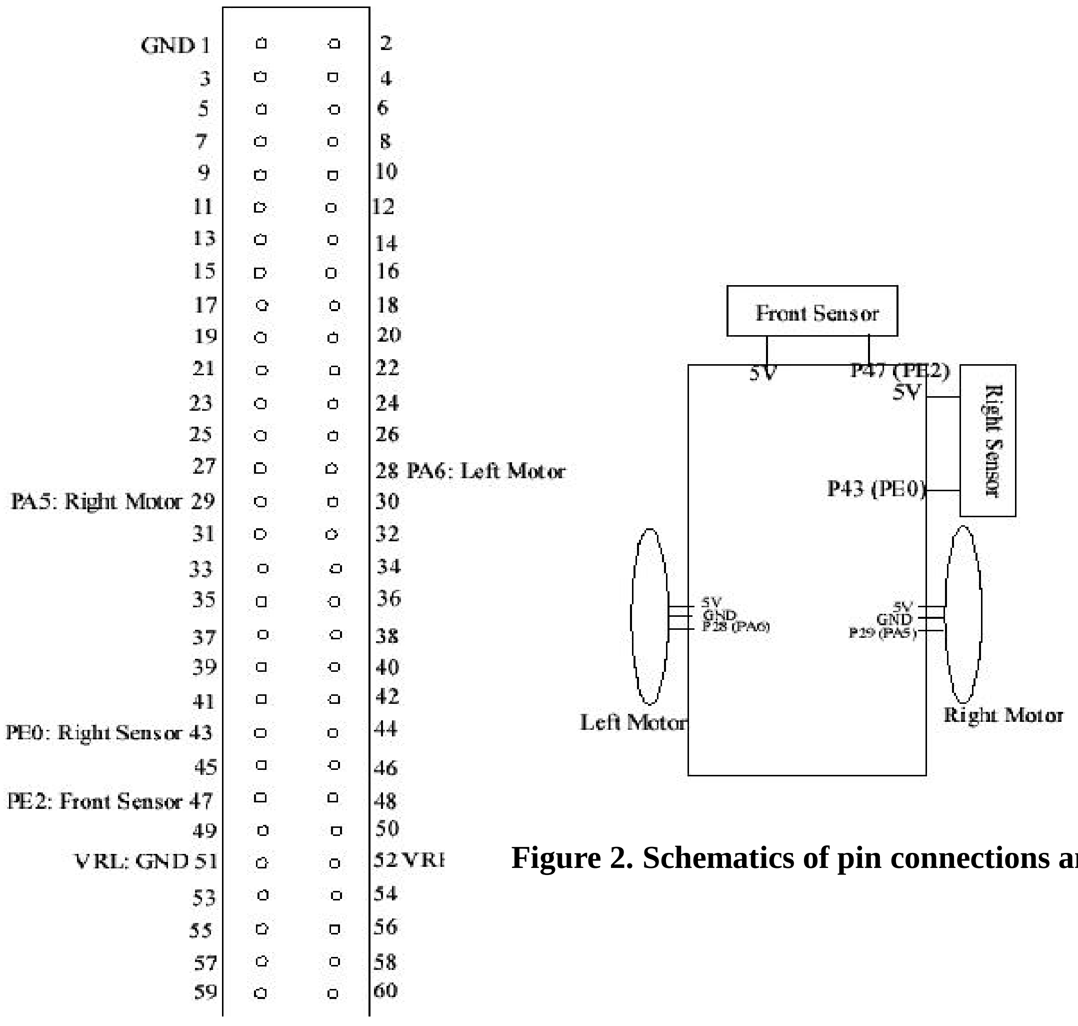

Figure 2. Schematics of pin connections and board layout.

3.2. Power Supply

To run the robot autonomously (untethered) it would have to contain an internal power supply; the clearest way to implement this was using batteries. The prototype was powered using a variable power supply, but this meant that the robot was tethered and could only go as far as the power cords allowed. It was calculated that four AA 1.5V batteries could supply the necessary power. There are three elements that require power the two motors and the evaluation board each requiring 5V. Note that if these elements are in parallel when hooked up to the batteries, a 5V output from the batteries is all that is needed to power all three elements. After mounting the four batteries into a holder, the output voltage was measured to be 6.3707 V. In order to limit this voltage to the necessary 5V the corresponding resistance was ascertained. See Fig. 3 for a schematic of the implemented circuit. However, due to the internal resistance of the batteries and the combined resistance of the elements, the voltage output from the batteries sagged. The solution was to give each element its own power supply; however, time did not permit this solution to be implemented.

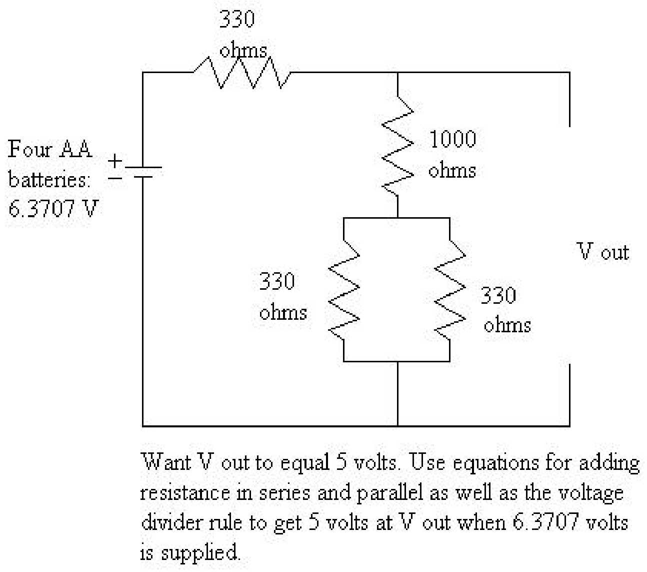
330 ohms
Four AA batteries: 6.3707 V
+
−
1000 ohms
330 ohms
330 ohms
V out
Want V out to equal 5 volts. Use equations for adding resistance in series and parallel as well as the voltage divider rule to get 5 volts at V out when 6.3707 volts is supplied.

Figure 3. Power Supply Circuit.

4. Mechanical Elements

Though the mechanical elements were not the emphasis of this project, a prototype was built to test the control code. The following are descriptions of the elements used to build the prototype.

4.1. Chassis

The original plan was to enclose the circuitry and power supply within a closed chassis. However, due to budget constraints, everything was mounted to a wooden board. This change of plan was actually a good decision since it allowed for an optimal, custom design when attaching all of the elements together.

4.2. Mounting

The motors were secured to the board by first drilling a pair of holes for each motor into the board. The motors were then placed between their respective holes and affixed to the board using cable ties. A perforated board was used to connect the wires necessary for input signals from the sensors and output signals to control the motors. See Fig. 2 for a diagram of these connections.

The front and right sensors were also attached to the wooden board. The front sensor was made by wire wrapping five push-button sensors to a perforated board. The sensors were located in each the four corners and in the middle of the perforated board. A cover made of cardboard was placed over the sensors to create a front bumper. Each sensor was daisy-chained together so only one output line and one input line of 5V went to and from the main circuit board.

The right sensor was a little harder to implement due to the nature of the turns. The right sensor needed to be extended such that if the robot ran into a front wall and turned left 90 °, the right sensor would be activated during the turn. Also, the robot needed to be far enough from the wall such that it would not hit corners upon turning. See Fig. 4 for clarification.

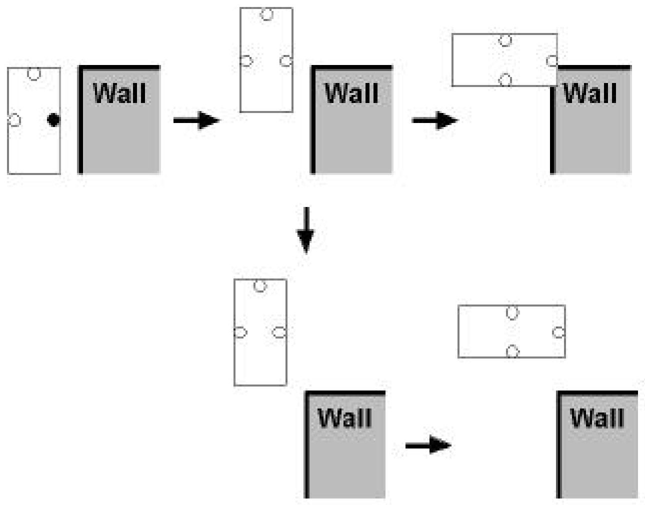

Figure 4. Clipping walls.

To solve this problem, an "arm" was attached to the right side of the robot. A sensor switch was attached to the arm so that when the arm came in contact with the wall it would depress and activate the sensor. The material used to construct the arm needed to fulfill the following criteria:

1. It had to be stiff enough to depress the sensor.
2. The material needed to be flexible enough to follow the contours of a wall.
3. The arm needed to be able to "spring" back when it is not in contact with a wall so that the sensor would not be continuously activated when away from a wall.

After considering these criteria, it seemed that a metal strip could be used. In particular, a copper strip was chosen to make the arm. The strip was creased so that it would bend away from the robot but while allowing the arm to bend back into place when not depressed by a wall. The arm was attached to the wooden board with two screws to prevent the arm from rotating.

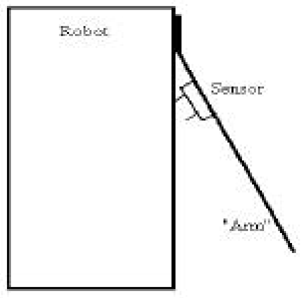

Figure 5. Arm and right sensor placement.

A cylinder was also attached to the end of the arm to minimize friction when following a wall. See Fig. 5 for a top-view of the robot and the right sensor.

4.3. Sensors

A lot of research went into choosing the sensor. There were three main kinds of sensors that were considered. The first was the infrared sensor; however, it did not always provide a clear voltage change when detecting an object. The next kind of sensor considered was a strain-gauge sensor. After wiring it in parallel with a resistor in a voltage divider schematic it produced voltage changes when compressed. However, like the IR sensor, there was not a significant voltage change when compressed. The sensors were finally implemented using a simple switch, as shown in Fig. 6. When the switch is not activated the two lines are kept separate, whereas when the button is pressed the two lines are connected. Consequently, this kind of sensor provides a definite 5V when connected and 0V when not.

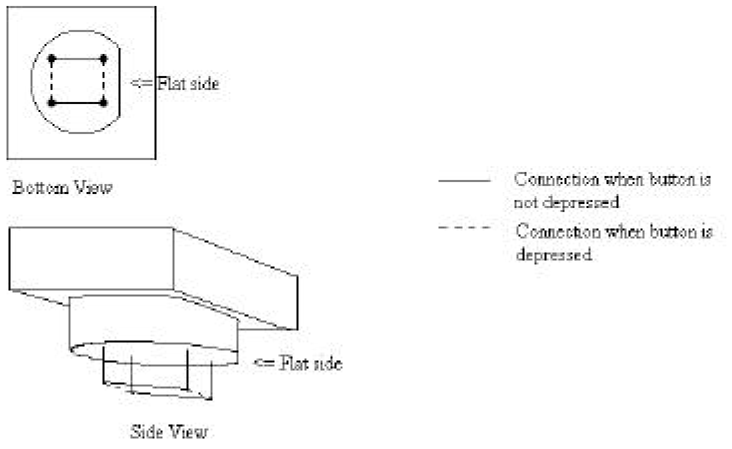
<= Flat side
Bottom View
Connection when button is not depressed
Connection when button is depressed
<= Flat side
Side View

Figure 6. Sensor schematic.

5. Microcontroller Design

5.1. Testing Procedures

During the course of the project many individual elements were tested. The first step was to consolidate the controlling code and the algorithm since different individuals wrote each part. Before the code was written, common variable names and function calls were agreed upon so that merging the two codes would be simplified. Next, the PWM signal used to run the motors were tested by viewing the output wave on an oscilloscope. After the PWM signal was confirmed, each motor was hooked up to understand how they worked and what signal was needed to control them. Once both motors had been modified and tested, the control code needed to be validated. However, in order to test the code and the motors' response, sensor inputs were needed. To expedite testing, a switchboard was constructed to represent the right and front sensors. When the motors were responding to the given stimuli in the way the algorithm proposed, each element was mounted to the chassis. Later, sensors replaced the switchboard so that a physical environment could be used to test the prototype.

5.2. Algorithm

Pseudo-coding the desired motion was the first step towards developing the follow-the-wall algorithm. The code was first written for three sensors: one in front, to the left, and to the right of the robot; see Appendix C for pseudo-code. For simplicity and efficiency, the code was modified for only two sensors. See Appendix B for the main code that was later translated into HC11 assembly language. Calls to move the robot forward or to turn the robot were developed in subroutines and used by this main code. Common subroutines called were:

- FORWARD_C (move forward continuously until either sensor hits a wall)
- FORWARD_CR (move forward until the front sensor sees a wall or the right sensor looses the wall)
- LEFT_TURN (execute a left turn)
- RIGHT_TURN (execute a right turn)

See Appendix D for complete assembly translation of the code.

5.3. Servo Motors

Since the servo motors use PWM, Output Compares 2 and 3 are employed in the code. A single-byte variable is used for each motor, containing the high-time length of the pulses. For the high value, the variable is essentially multiplied by 256 (it is shifted to the left 8 times) to ensure that the free running counter does not run past the Output Compare value before the interrupt is completed. To arrive at the number of clock cycles to pass for the low value, 256 is multiplied by 256 minus the variable. Since the HC11 operates at 2Mhz, a variable with the value 2E hex (46 decimal) would, for example, equate to the following high-time length (HTL):

$$HTL = \left(\frac{46 * 256 cycles}{2000000 \, {}^{cycles}\!/_{\sec}} \right) = 5.89 ms$$

As a temporary setup to determine what values were needed for the PWM signals, HC11 code was written to take as input an analog voltage value and output a PWM signal based on that voltage. The voltage value was on a scale from 0 (0V) to 255 (5V). Table 1 presents the voltages and input values that corresponded with different motor movements.

	Left Motor	Right Motor
Forward	0.313V (10)	0.602V (07)
Stop	0.474V (18)	no value (00)
Backward	0.838V (2A)	0.133V (1E)

Table 1. Voltage values and hex input values (in parentheses).

These values are used by the subroutines that both drive the robot forward and turn it left and right. Note that the right motor has no stopping position, since no voltage value could be found to stabilize the motor. See Appendix D for the HC11 code that performs all of the PWM.

5.4. Sensors

The exact, physical implementation of the sensors was not clear throughout much of the project. Infrared sensors, strain-gauge sensors, and simple switches were all possibilities; hence, the method of input that could handle all sensors was the use of analog-to-digital (A/D) converters. An A/D converter was used for both the front and right sensors. After deciding to use simple switches as the sensors, the only change to the A/D converter code was to change the boundary point between a high and low signal (this boundary was set at 80 hex).

The conversion is triggered by a real-time interrupt that is called approximately every 8ms. At this point, the two variables that store the front and right sensor data are written as zeros if their voltage values are below 80 hex and ones if their voltage values are above 80 hex. Consequently, any part of the code that wishes to view the current state of either sensor will have an accurate reading to within about 8ms.

6. Results

The final project produced a prototype that responded to sensory inputs. The prototype was driven by a variable power supply and controlled by an HC11 evaluation board. The motors were configured so that the robot moved forward and turned in both directions. The forward motion of the prototype was very close to linear except for a slight drift to the left; the veering was hardly noticeable. Each turn was a couple degrees over 90 ° but a left turn would compensate for a right turn and vice versa.

The most difficult part of the project was building a prototype to test the control code. Choosing an appropriate sensor was time consuming and many compromises had to be made, such as settling for a limited range in exchange for a clear, digital signal.

The next problematic section of the project was moving all of the components onto the chassis and freeing the robot from the tethers of the variable power supply and cables to the computer. Moving the existing program from the computer to the available EEPROM space on the evaluation board required some changes to the code and the pin out assignments.

The M68HC11EVB Evaluation Board was used to help debug the code. There was not enough time to use the M68HC11EVBU Universal Evaluation Board, which we had hoped to use, because we thought that the first one did not carry EEPROM. Apparently, though, both evaluation boards contain 512 bytes of EEPROM. Nevertheless, we learned how to program the EEPROM using the following steps:

1. Adjust the code so the origin is at $B600. This is the beginning of EEPROM in the HC11 memory. When a jumper is switched on the evaluation board, the HC11 will start executing commands at this location when powered. Note that variables must be placed after the end of the code, and no other origins can be set (so if you set interrupt vectors using the ORG instruction, you must change this to write the interrupt vector directly).
2. In BUFFALO, type "`mm 1023`", and then "`35`". This sets the baud rate on the HC11 for loading programs over the serial line to 300.
3. Change the baud rate of your Hyperterm (or equivalent) program to 300, disconnect, and reconnect.
4. Type "`load t`" and send your program to the HC11 as a text file. It may take a half a minute to load.

The final outcome of the project differed in some aspects from our proposed project. The backwards motion was replaced by two consecutive turns and then continue forward. Also, variable speed could not be implemented because the motors did not provide a smooth motion driven with a high-time length between moving forward and being stopped. In addition, a custom-made PCB was not used because we felt that a wire-wrapped board was more versatile. Our original proposal did not state that a fully autonomous robot would be constructed, but we nevertheless tried to implement one.

7. References

[1] How to modify a Futaba FP-S148 servo for 360° rotation, http://pws.prserv.net/pebly/futaba360.html

[2] Tower Hobbies, http://www.towerhobbies.com

[3] Futaba Corporation (California), (949) 455-9888

8. Parts List

Part	**Source**	**Vendor Part #**	**Price**
Futaba Servo Motor (2)	Tower Hobbies	S148	$30.00 *
AA Batteries 1.5V (4)	Stock Room	---	---
Wire Wrapping Access.	Stock Room	---	---
Perforated Board	Stock Room	---	---
Battery Case	Professor Harris	---	---
Copper Strip	Sheet Metal Shop	---	---
Wooden Board	Wood Shop	---	---

* Does not include price of shipping/handling or tax.

Appendix A

The following diagram shows the parts of a Futaba S148 servo motor, and highlights the modifications that must be made to allow for a full, 360-degree range.

When the potentiometer stop (#14 in Fig. 7) is removed, the servo motor is able to turn 360 degrees; however, the feedback system does not work properly under these conditions, since it is geared towards a 180 degree turn. Consequently, the drive plate (#7 in Fig. 7) must be removed in order to force the feedback system to read a constant value from the potentiometer. The feedback system subsequently reads a constant position for the servo and, when supplied with a PWM signal, causes the motor to move in the direction of the goal position of this PWM.

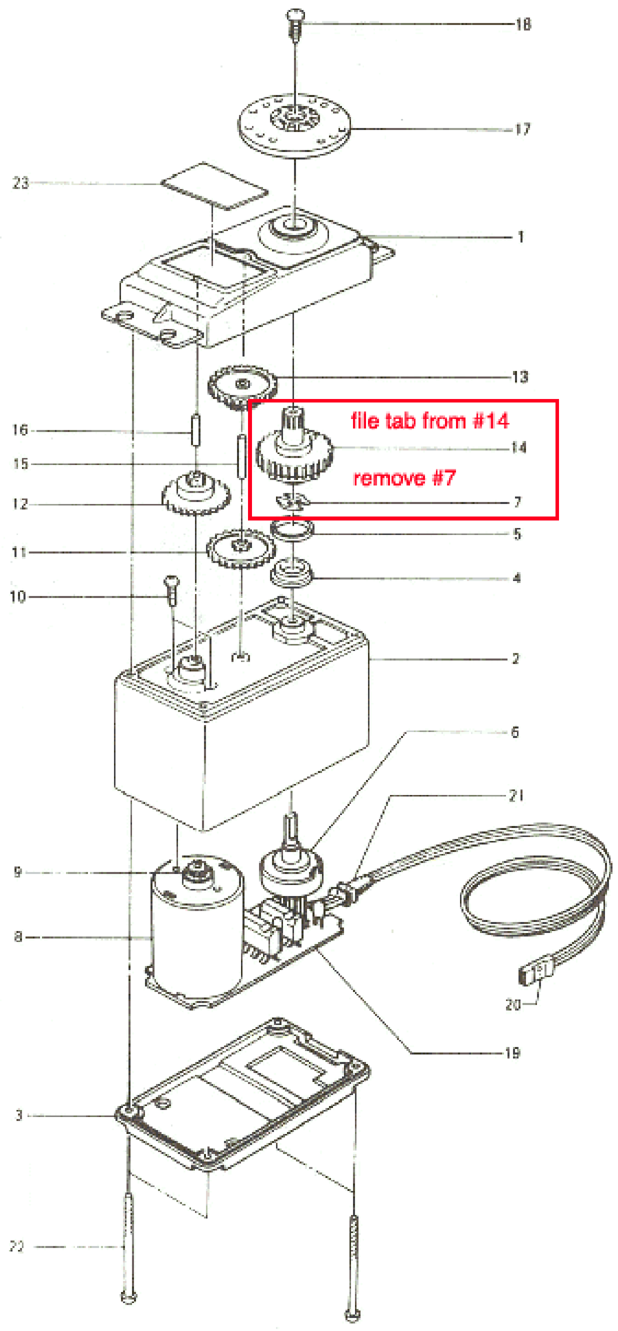

Figure 7. A Futaba S148 servo motor with modifications for 360-degree rotation.

Appendix B

```
//BEGIN
//initializing variables
int wall = 0; // 0 => robot has not found a wall
              // 1 => robot has found a wall
int R = 0;    // 0 => right sensor is not activated
              // 1 => right sensor has found a wall
int F = 0;    // 0 => front sensor is not activated
              // 1 => front sensor has found a wall

//NO_WALL
while (wall == 0)  // cycle here until a sensor detects a wall
{
      if ((F == 0) && (R == 0))
            go forward; //move forward if no wall is detected on either
                        // sensor
      else              // a sensor has hit a wall
            wall = 1;
}

//GET_WALL: move the robot so that the wall it found is to its right
if (F == 1)             // if there is a wall in front of the robot
      turn left 90°;    // no code needed for R == 1 since wall is
                        // already on the right side of the robot

//HAVE_WALL: motions to execute once there is a wall to the right of
//           the robot
while (wall == 1)             // make sure robot has a wall
{
      while (R == 1)          // make sure the right sensor reads wall
      {
            if (F == 0)             // if space in front of robot clear
                  go forward;       // then move forward
            else                    // if object in front of robot
                  turn left 90°;    // then turn left
      } // end while (R == 1) loop


//DOOR: execute this command if when R = 0 and wall is lost (i.e. a
//      door has been found or a corner has been reached

// this command is necessary to prevent the robot clipping the corner
// when it turns, see Fig. 4 for pictorial explanation
      go forward width of robot;
      turn right 90°;               // turn right
      if (( F = = 0) && (R = = 0))  // robot has lost wall
            wall = 0;
      else                    // robot has wall, go back and make
                              // sure the wall is on the robot's right
            branch always to GET_WALL;
}                             // end while (wall == 1) loop
branch always to NO_WALL;     // if this section of code is reached
                              // robot has lost the wall so it will
                              // move forward until it finds one
```

Appendix C

```
Begin
{ // initializing variables
  wall = 1;

No_Wall
  while (wall == 0)
    {
      if ((F == 0) && (R == 0) && (L == 0))
      {
         go forward;      // R = Right Sensor
                          // L = Left Sensor
                          // F = Front Sensor
      }
      else
      {
        wall = 1;
      }

Get_Wall                  // getting a right wall
  if ((F == 1) && (R == 0) && (L == 0))
  {
    turn left 90º;
  }
  else if (((F == 1) && (R == 0) && (L == 1)) ||
           ((F == 1) && (R == 1) && (L == 1)) ||
           ((F == 0) && (R == 0) && (L == 1)))
  {
    turn 180º;
  }

while (wall == 1)
  {
    while (R == 1)     // still have right wall
    {
      if (F == 0)      // room to go forward
      {
        go forward;
      }
      else if (L == 0) // front is blocked, see
      {
        turn left 90º; // if we can turn left
      }
      else             // if front and left is
      {                // blocked, turn around
        turn 180º;
      }

            }  // end while (R == 1)

    turn right 90º;     // this happens when
                        // R == 0, because it
                        // could be a door

            // check to see if any walls are around
            if ((F == 0) && (R == 0) && (L == 0))
            {
                  wall = 0;
            }

            // we have a sensor equal to 1, get right wall
            else
            {
              branch (bra) Get_Wall;
            }
  } // end while (wall == 1)

  branch (bra) No_Wall;

} // end program
```

Appendix D

```
*************************************************************
* Ronalee Lo and Dave Beydler                               *
* rjlo@hmc.edu, dbeydler@hmc.edu                            *
* E157 Final Project                                        *
* November 27, 1999                                         *
* Description: Code for a wheeled robot.                    *
*************************************************************

*************************************************************
************************** DATA ****************************
*************************************************************

***********
* Symbols *
***********
            ORG   $0

REG         EQU   $1000
TOC2        EQU   $1018 * The output compare register
TOC3        EQU   $101A * The output compare register
TCTL1       EQU   $1020 * Timer control register 1
TMSK1       EQU   $1022 * Timer mask register 1
TFLG1       EQU   $1023 * Timer flag register 1

TMSK2       EQU   $1024 * Timer mask register 2
TFLG2       EQU   $1025 * Timer flag register 2
PACTL       EQU   $1026 * Pulse accumulator control register

BIT6        EQU   %01000000  * OL2
BIT4        EQU   %00010000  * OL3
BIT0        EQU   %00000001  * For getting sensor info
OC2F        EQU   %01000000
OC3F        EQU   %00100000

PORTA       EQU   $1000 * control motors and read from sensors

PORTB       EQU   $1004

PORTE       EQU   $100A * where digital rep. of analog signal is read
ADCTL       EQU   $1030 * A/D control register
ADR1        EQU   $1031 * A/D result register 1 (PE0)
ADR2        EQU   $1032 * A/D result register 2 (PE1)
ADR3        EQU   $1033 * A/D result register 3 (PE2)
ADR4        EQU   $1034 * A/D result register 4 (PE3)
OPTION      EQU   $1039 * Hardware option control register

BACK_PWM_R        EQU   $2A   * PWM signal for backwards right motor
NO_MOVE_PWM_R     EQU   $18   * PWM signal for nowhere right motor
FORWARD_PWM_R     EQU   $10   * PWM signal for forwards right motor

BACK_PWM_L        EQU   $07   * PWM signal for backwards left motor
NO_MOVE_PWM_L     EQU   $00   * PWM signal for nowhere left motor
*                               (doesn't work)
FORWARD_PWM_L     EQU   $1E   * PWM signal for forwards left motor

RIGHT_PWM   FCB   $1E   * The PWM signal that the right motor reads
LEFT_PWM    FCB   $1E   * The PWM signal that the left motor reads
```

```
SENSOR_F    FCB   $00   * 1 if front sensor on, 0 if front sensor off
SENSOR_R    FCB   $00   * 1 if right sensor on, 0 if right sensor off

WALL        FCB   $00   * 1 if we have a wall, 0 if we don't

INVALUE     FCB   $00   * Input from port E
PWMLO       FDB   $0100 * How long to wait on low
PWMHI       FDB   $0100 * How long to wait on high


****************************
* Set up interrupt vectors *
****************************
* Note: these will probably change with a different chip
            ORG   $00D9
            JMP   OC3ISR

            ORG   $00DC
            JMP   OC2ISR

            ORG   $00EB
            JMP   RTIISR


************************************************************
*********************** MAIN PROGRAM ***********************
************************************************************

********************
* Initialize stack *
********************
            ORG   $D000
            LDS   #$DFFF      * Define the stack

****************************
* Initialize A/D converter *
****************************
* OPTION: ADPU=1, CSEL=0, IRQE=0, DLY=0, CME=0, 0, CR1=0, CR0=0
            LDAA  #$80
            STAA  OPTION

* Initialize Port E
            CLR   PORTE

* ADCTL: SCAN=1, MULT=1, CD=0, CC=0, CB=x(0), CA=x(0)
*   Since we have MULT=1, CD and CC select the group of four ADRs
*   PE0 = ADR1, PE1 = ADR2, PE2 = ADR3, PE3 = ADR4
            LDAA  #$30
            STAA  ADCTL

******************
* Initialize PWM *
******************
* Initialize OC2 and OC3
            LDAA  #$A0
            STAA  TCTL1
* Enable OC2 and OC3 interrupt
            LDAA  #$60
            STAA  TMSK1
```

```
* Slow down clock to lengthen periods
            LDAA  #$03
            STAA  TMSK2

**********************************
* Initialize Real Time Interrupt *
**********************************
* Set interrupt rate to 8.19 ms
            LDAA  #1
            STAA  PACTL
* Enable RTI interrupt
            LDAA  #BIT6
            STAA  TMSK2


******************
* Stop the robot *
******************
            JSR   STOP_ROBOT

*********************
* Enable interrupts *
*********************
            CLI

*************
* Main loop *
*************

*F refers to front sensor (SENSOR_F)
*R refers to right sensor (SENSOR_R)
NO_WALL
            LDAA  SENSOR_F          *get value of F
            BNE   HIT               *if F != 0, branch to HIT
            LDAB  SENSOR_R          *get value of R
            BNE   HIT               *if R != 0, branch to HIT
            JSR   FORWARD_C         *if R == 0 and F == 0 move forward
                                    *until F or R equals 1

HIT                                 *wall found
            LDAA  #$01              *set wall == 1
            STAA  WALL

GET_WALL                            *move robot so wall is on its right
            LDAA  SENSOR_F          *get value of F
            BEQ   RIGHT_1           *if F == 0, branch to RIGHT_1
            JSR   LEFT_TURN         *else turn left

RIGHT_1
            LDAB  SENSOR_R          *load R
            BEQ   DOOR              *if R == 0, branch to DOOR
            LDAA  SENSOR_F          *load F
            BEQ   CONT_FORWARD      *if F == 0, branch to CONT_FORWARD
            JSR   LEFT_TURN         *if R == 1 and F == 1 turn left
            LDAB  SENSOR_R          *load R
            BNE   RIGHT_1           *if R != 0, branch to RIGHT_1
            BEQ   DOOR              *if R == 0, branch to DOOR

CONT_FORWARD
            JSR   FORWARD_CR        *move forward until F == 1 or R ==0
            LDAB  SENSOR_R          *load R
            BEQ   DOOR              *if R == 0 branch to DOOR

LEFT2                               *if R == 1
            JSR   LEFT_TURN         *turn left
            BRA   RIGHT_1           *branch to RIGHT_1

DOOR                                *door or corner found
            JSR   FORWARD_W         *move forward a length of w
            JSR   RIGHT_TURN        *turn right
            LDAA  SENSOR_F          *load F
            BNE   GET_WALL          *if F != 0 branch to GET_WALL

            LDAB  SENSOR_R          *load R
            BNE   RIGHT_1           *if R != 0 branch to RIGHT_1

* WALL == 0                         *if R == 0 and F == 0 robot is lost
            LDAA  #$00              *set wall == 0
            STAA  WALL
            BRA   NO_WALL           *branch to NO_WALL


***********************************************************************
*********************** INTERRUPT FOR SENSORS ***********************
***********************************************************************

RTIISR
* Did we get an interrupt from the real time interrupt device?
            LDX   #REG
            BRCLR TFLG2-REG,X BIT6 RTRTI  * If not, ignore by returning

* Clear real time interrupt flag (remember: 1 clears, 0 does nothing)
            LDAA  #BIT6
            STAA  TFLG2-REG,X

* Get front sensor data (PE2)
            LDAA  ADR3
            CMPA  #$80
            BHI   SET_SENSOR_F_1
            LDAA  #$00
            BRA   SET_SENSOR_F
SET_SENSOR_F_1
            LDAA  #$01
SET_SENSOR_F
            STAA  SENSOR_F

* Get right sensor data (PE0)
            LDAA  ADR1
            CMPA  #$20
            BHI   SET_SENSOR_R_1
            LDAA  #$00
            BRA   SET_SENSOR_R
SET_SENSOR_R_1
            LDAA  #$01
SET_SENSOR_R
            STAA  SENSOR_R

RTRTI
            RTI                     * Return from the interrupt
```

```
***********************************************************************
*********************** INTERRUPT FOR LEFT MOTOR ********************
***********************************************************************

**********************
* Check that flag set *
**********************
OC2ISR
            LDX   #REG
            BRCLR TFLG1-REG,X OC2F RTOC2

*****************
* Clear the flag *
*****************
            LDAA  #OC2F
            STAA  TFLG1-REG,X       * Acknowledge that interrupt
occurred

            LDAA  LEFT_PWM
            STAA  INVALUE

* Store the analog value into PWMHI
            LDAA  INVALUE
            LDAB  #$00
            STD   PWMHI

* Store $FF minus the analog value into PWMLO
            LDAA  #$FF
            SUBA  INVALUE
            LDAB  #$00
            STD   PWMLO

            BRSET TCTL1-REG,X BIT6 LASTHI1

***************
* Set PWM high *
***************
            BSET  TCTL1-REG,X BIT6
            LDD   TOC2-REG,X
            ADDD  PWMLO
            STD   TOC2-REG,X

            BRA   RTOC2

**************
* Set PWM low *
**************
LASTHI1
            BCLR  TCTL1-REG,X BIT6
            LDD   TOC2-REG,X
            ADDD  PWMHI
            STD   TOC2-REG,X

RTOC2
            RTI                     * Return from the interrupt

***********************************************************************
*********************** INTERRUPT FOR RIGHT MOTOR *******************
***********************************************************************
```

```
***********************
* Check that flag set *
***********************
OC3ISR
            LDX   #REG
            BRCLR TFLG1-REG,X OC3F RTOC3

******************
* Clear the flag *
******************
            LDAA  #OC3F
            STAA  TFLG1-REG,X       * Acknowledge that interrupt
occurred

            LDAA  RIGHT_PWM
            STAA  INVALUE

* Store the analog value into PWMHI
            LDAB  INVALUE
            LDAA  #$00
            LSLD
            LSLD
            LSLD
            LSLD
            STD   PWMHI

* Store $FF minus the analog value into PWMLO
            LDAB  #$FF
            SUBB  INVALUE
            LDAA  #$00
            LSLD
            LSLD
            LSLD
            LSLD
            STD   PWMLO

            BRSET TCTL1-REG,X BIT4 LASTHI2

****************
* Set PWM high *
****************
            BSET  TCTL1-REG,X BIT4
            LDD   TOC3-REG,X
            ADDD  PWMLO
            STD   TOC3-REG,X

            BRA   RTOC3

***************
* Set PWM low *
***************
LASTHI2
            BCLR  TCTL1-REG,X BIT4
            LDD   TOC3-REG,X
            ADDD  PWMHI
            STD   TOC3-REG,X

RTOC3
            RTI                     * Return from the interrupt
```

```
************************************************************
******************* MOVING SUBROUTINES *******************
************************************************************

**************************
* Move forward one cycle *
**************************
FORWARD_W
            PSHA
            LDAA  #FORWARD_PWM_L
            STAA  LEFT_PWM
            LDAA  #FORWARD_PWM_R
            STAA  RIGHT_PWM

* Delay for one second
            LDAA  #$32        * Parameter = # of 20 millisecond delays
            JSR   DELAY

            JSR   STOP_ROBOT

            PULA
            RTS


****************************************************
* Move forward continuously until F == 1 or R == 0 *
****************************************************
FORWARD_CR
            PSHA
            LDAA  #FORWARD_PWM_L
            STAA  LEFT_PWM
            LDAA  #FORWARD_PWM_R
            STAA  RIGHT_PWM

* Poll the sensors to see if we hit something
FORWARD_LOOP1
            TST   SENSOR_F
            BNE   FORWARD_BREAK1
            TST   SENSOR_R
            BEQ   FORWARD_BREAK1
            BRA   FORWARD_LOOP1

FORWARD_BREAK1
            JSR   STOP_ROBOT

            PULA
            RTS


****************************************************
* Move forward continuously until F == 1 or R == 1 *
****************************************************
FORWARD_C
            PSHA
            LDAA  #FORWARD_PWM_L
            STAA  LEFT_PWM
            LDAA  #FORWARD_PWM_R
            STAA  RIGHT_PWM


* Poll the sensors to see if we hit something
FORWARD_LOOP2
            TST   SENSOR_F
            BNE   FORWARD_BREAK2
            TST   SENSOR_R
            BNE   FORWARD_BREAK2
            BRA   FORWARD_LOOP2

FORWARD_BREAK2
            JSR   STOP_ROBOT

            PULA
            RTS


*************************
* Turn right 90 degrees *
*************************
RIGHT_TURN
            PSHA
            LDAA  #FORWARD_PWM_L
            STAA  LEFT_PWM
            LDAA  #BACK_PWM_R
            STAA  RIGHT_PWM

* Delay for one second
            LDAA  #$32        * Parameter = # of 20 millisecond delays
            JSR   DELAY

            JSR   STOP_ROBOT

            PULA
            RTS


************************
* Turn left 90 degrees *
************************
LEFT_TURN
            PSHA
            LDAA  #BACK_PWM_L
            STAA  LEFT_PWM
            LDAA  #FORWARD_PWM_R
            STAA  RIGHT_PWM

* Delay for one second
            LDAA  #$32        * Parameter = # of 20 millisecond delays
            JSR   DELAY

            JSR   STOP_ROBOT

            PULA
            RTS


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

```
* Stop the robot *
******************
STOP_ROBOT
            PSHA
            LDAA  #NO_MOVE_PWM_L
            STAA  LEFT_PWM
            LDAA  #NO_MOVE_PWM_R
            STAA  RIGHT_PWM
            PULA

            RTS


***********************************
* Delay for (A * 20) milliseconds *
***********************************
DELAY
            TSTA
            BEQ   RT
            LDX   #$115C

AGAIN
            BEQ   AHEAD
            DEX
            BRA   AGAIN

AHEAD
            DECA
            BRA   DELAY

RT
            RTS

            END
```