Final Report

Crawford Hampson

"Emerson"

April 20, 2010

University of Florida

Department of Electrical and Computer Engineering

EEL 5666
Intelligent Machines Design Laboratory

Instructors: Dr. A. Antonio Arroyo & Dr. Eric M. Schwartz
TAs: Mike Pridgen & Thomas Vermeer

Table of Contents

Abstract

This paper describes “Emerson,” a robot designed to autonomously find and plug itself into wall outlets. Emerson is built around a circular planform with differential drive wheels. Bump and IR sensors are used for navigation. Battery voltage is monitored, and a number of subsystems control the various operations. Emerson is equipped with a two-axis linear actuator stage which is used to manipulate the plug. While at present plug-finding behavior is intermittent, work is ongoing to make Emerson a reliably self-reliant robot.

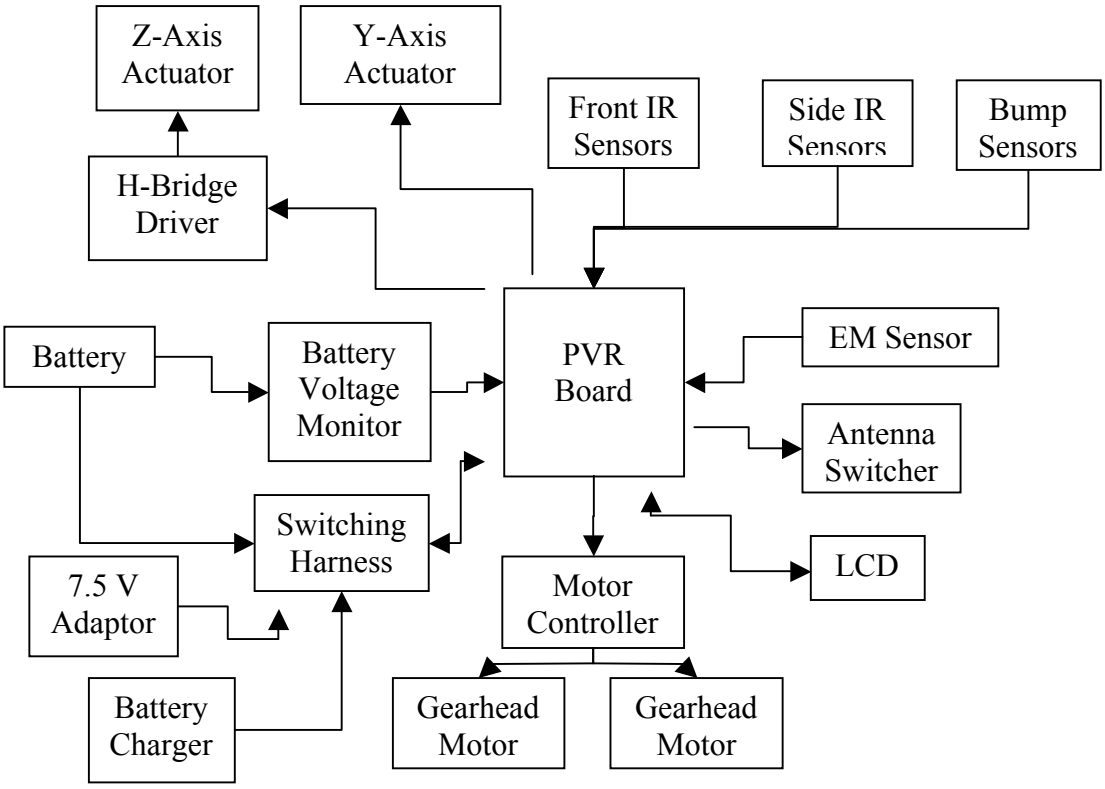
Executive Summary

Robot Name:	"Emerson"
Robot Designer:	Crawford Hampson
Purpose:	Automously find and plug into electrical wall sockets
Microcontroller:	Pridgen-Vermeer Robotics Board
Battery:	1 x 6-Cell Rechargable AA NiMH
Motors:	2 x 24 V Merkle-Korff Gearhead Motors
Motor Drivers:	1 x Sparkfun 1A Dual TB6612FNG 1 x SN754410 H-Bridge Chip
Sensors:	2 x Sharp GP2Y0A21YK Medium-range IR Sensors 2 x Sharp GP2D120 Short-Range IR Sensors 4 x Radio Shack #275-017 "SPDT Switch with 3/4" Roller Lever." Battery Voltage Monitoring Circuit (Voltage Divider) Plug Power Sensor (Optoisolator) Electromagnetic Sensor (Greenlee Noncontact Voltage Detector)
Actuators:	Z-Axis Linear Actuator (Stepper Motor Driven) Y-Axis Linear Actuator (Hacked Servo Driven) Antenna Switching System (Low-Voltage Relay)

Introduction & Purpose

"Emerson," named for the famous author of "Self-Reliance," is designed to do just as its namesake advocated – rely on its own abilities to provide that which robots need most to continue functioning, electrical power. It is intended to seek out electrical wall outlets and plug itself into them. Emerson roams autonomously, avoiding obstacles, until it detects that its battery voltage has dropped below a critical threshold. At this point, the robot finds and begins following a wall until, using an electromagnetic field sensor, it detects a wall outlet. The robot then stops and, using a Y-Z stage actuating a plug, finds the outlet precisely and plugs itself in. This report will describe the mobile platform that supports this functionality, the sensors used, the charging subsystems, the electromagnetic sensor interface, and the software-based behaviors Emerson is currently capable of.

Integrated System



Mobile Platform

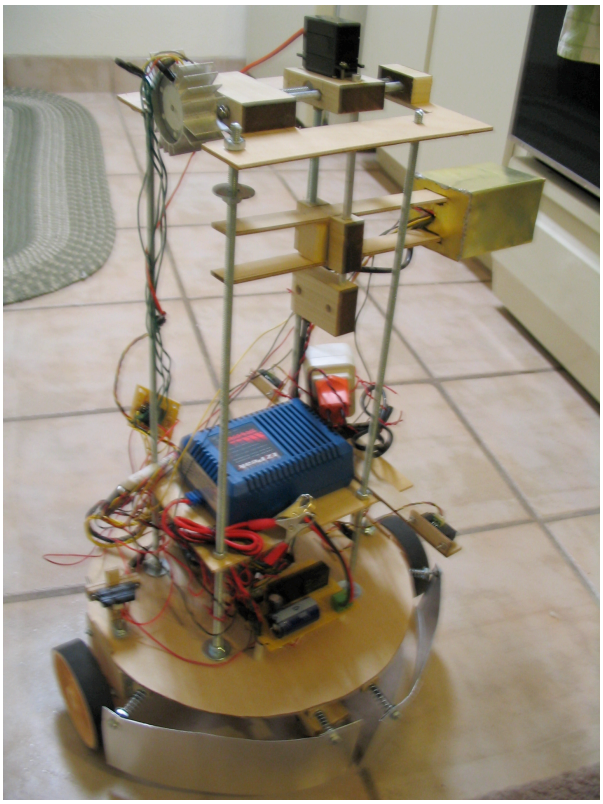


Figure 1

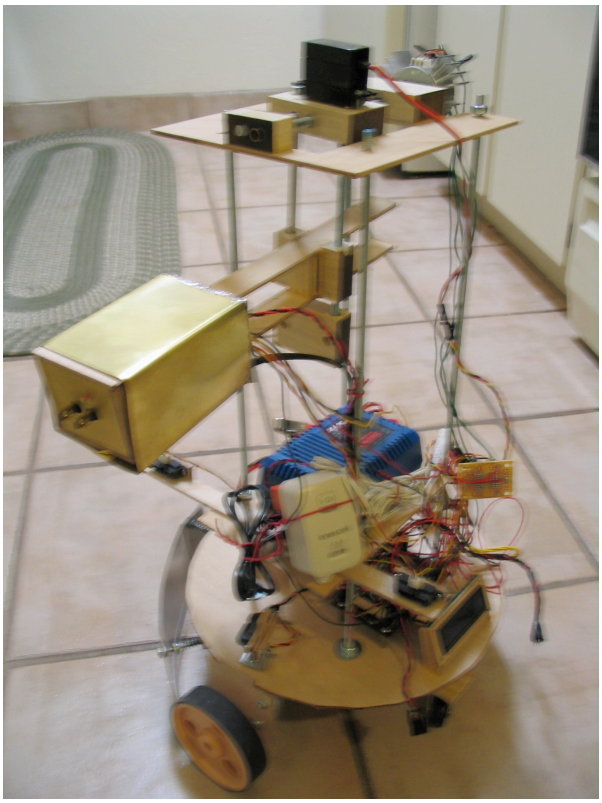


Figure 2

Structure

Emerson's platform is an approximately one-foot diameter circle, with two large gearhead electric motors at either side of the circle's centerline providing locomotion (see Figures 1 & 2). These drive motors are each directly attached to a three inch diameter plastic wheel with rubber traction soles. Two wheel casters along the centerline oriented ninety degrees from the drive motors provide balance. The battery is mounted underneath the main platform, and the PVR board controlling the robot is attached to the platform just aft of center. Two aluminum plates are mounted to the front of the robot, attached to four bump sensor switches. Based on an idea presented by Professor Arroyo in class, these bump sensors are wired into a resistor array and connected to one of the analog inputs of the PVR board; as each is pressed, or as different combinations are pressed simultaneously, different voltages appear on the output line. Two Sharp medium-range infrared sensors are attached to adjustable mounts on either side of the robot, just inboard from the wheels. The battery is connected to a switching harness board which contains a circuit I designed to, upon command from the PVR board, switch the battery pack over to the battery charger and the microcontroller over to a nine volt AC-DC power supply. This is done using two double-pole double-throw relays. A 4700 µF capacitor across the PVR power supply terminals provides constant power to the PVR board during the relays' momentary switching period.

Actuators

In order to both sense the socket while driving past and precisely locate the hot line of the plug for final alignment, Emerson has a switchable antenna system. Two different antenna styles were used:

a flat plate approximately two inches on a side provides large-scale sensing, while an approximately two millimeter metal circle provides high-resolution sensing. The large antenna is affixed to the flat face of the plug, with two holes through the center to allow passage for the plug tines, the bases of which have been insulated. I based this design off of the Intel robot "Marvin."[1] A small hole drilled through the antenna plate directly above the hot tine of the plug allows passage of the small antenna, which is attached to a flexible wire and extends approximately three millimeters past the large antenna. The leads from the small and large antennas go to the normally closed and normally open lines of a small-signal relay, while the wire connected to the detector's antenna input goes to the common. A 3904 BJT transistor controlled by one of the I/O ports on the PVR board switches the flow of current through the relay coil.

To actuate the plug, a Y-Z stage is attached to the top of the tower extending from the mobile platform. X-axis movement is left to the main drive motors. The two axis stage consists of two linear motors using threaded rod and an aluminum guide rod. The Z-axis actuator, which moves the plug into and out of the socket, is driven by a large stepper motor, controlled using an H-bridge chip connected to the PVR board. I used a page from the web site Instructables as a reference while building this.[2] The Y-axis is driven by a hacked servo. The Y-axis actuator, while functional, is not currently implemented, though some placeholder code exists. At the moment Emerson relies on being pre-set to the socket height.

Sensors

The simplest sensors used on this robot are its bump sensors, each of which uses a Radio Shack #275-017 "SPDT Switch with 3/4" Roller Lever."[3] These are connected using a resistor ladder as Professor Arroyo described in class, allowing four bump switches to be detected using a single analog in line on the PVR board.

The primary obstacle avoidance sensor used is the Sharp GP2Y0A21YK.[4] It is the medium range model, with an approximately 80 cm maximum range. Two are mounted on Emerson, both set far enough back on the platform such that the approximately 10 cm dead zone in front of the sensor is taken up by the robot. I initially had great difficulty getting this sensor to work correctly, with a very strange error – it would output a declining voltage as an object moved from approximately 10 cm to approximately 150 cm in front of it, at which point the voltage would begin climbing again until reaching a maximum at approximately 300 cm. I eventually determined, with the help of the TAs and other students, that this was a strange side effect of using a 3.3V supply for the sensor, when it required a 5V supply. Upon properly supplying the sensor, it began operating normally. Additionally, the PVR board's analog inputs needed to be reset to a 0V to 5V range. This is done by opening the PVR.h code and setting the ADCA_REFCTRL variable to 0x10, then wiring analog input 1 to the five volt source on the servo lines.

Power from the plug is split between the AC-input on the hobby battery charger and an adjustable AC to DC adaptor set to 7.5 V. Voltage on the AC adaptor's output is currently sensed with an optoisolator, though this not really necessary given that that voltage is used to power the PVR board when connected to the wall.

The electromagnetic sensor presented the greatest difficulty of any of the sensors used, and ultimately was the element preventing Emerson from being fully operational by Media Day. Initially, I experimented a number of circuits I found on the internet designed to detect wires behind walls, but I did not have a great deal of success. Later, I attempted to use the live wire sensor embedded in a

1. Paper available at http://www.bdm.cc/pubs/plugin.pdf.
2. http://www.instructables.com/id/Drive-a-Stepper-Motor-with-an-AVR-Microprocessor/
3. Available at http://www.radioshack.com/product/index.jsp?productId=2049719.
4. Available at http://www.sparkfun.com/commerce/product_info.php?products_id=242.

Stanley stud finder. While I was able to tap into the IC lines that were triggered when an electromagnetic field was detected, the sensor itself turned out to be insufficiently sensitive, and additionally used an inconvenient voltage. I then attempted to build a custom op amp circuit, following the lead of the group which built Marvin, an Intel robot which can plug itself into walls.[5] While I could detect signals fairly well on the bench, integrating it with the mobile platform proved extremely difficult. The inconsistent power supply and requirement for a virtual ground meant that the available voltage was insufficient to sufficiently amplify the signal. After this attempt failed, I purchased an adjustable electrician's live wire detector,[6] which has proved more successful than the previous methods. However, difficulties with noise initially made detection of the socket nearly impossible. The construction of a Faraday cage around the entire sensor unit and plug assembly and the use of bypass capacitors helped this problem, but the noise problem was not gone, and difficulties with providing a consistent power supply remained.

An additional important part of Emerson's plug-finding behavior is the ability to follow walls. To do this, Emerson is equipped with two Sharp GP2D120 short range IR sensors which face to the left.

Behavior

The software Emerson is currently equipped with allows it to navigate with reasonable success in the environment of an academic building. The infrared sensors are sufficient to detect large obstacles, such as walls and other broad obstacles, and stop the robot before a collision occurs. As there are two such sensors, the robot is able to differentiate between obstacles to its left or right, and turn accordingly. If an obstacle is detected by both sensors, the robot at this point always turns right, but will eventually turn randomly. Because of its circular, symmetrical design, it is capable of rotating three hundred and sixty degrees in place. In order to prevent smaller or lower obstacles from obstructing the robot's progress, two aluminum plates mounted to bump sensor switches are attached to the front of the robot. Additionally, as the motors are fairly strong for its weight, the robot can drive over small obstacles such as door sills. The robot also uses a fuzzy logic speed control system, wherein the robots forward drive speed is controlled by the distances to objects reported by the infrared sensors. Thus, the robot will slow its forward progress if it "sees" something in its path, even if that object is not within the critical stopping distance to trigger its obstacle avoidance behavior.

The power seeking behavior mode consists of switching to the large antenna, activating the wall-following algorithm while operating the bump sensors, and checking for socket fields. The infrared sensors are disabled, as they face slightly outward and tend to be triggered by the proximity of the wall. This should not prove to be too large a problem, as the wall-following speed is significantly below Emerson's normal speed. In order to combat the noise problem, sensor readings from the detector are averaged with the previous measurements weighted more strongly than the current one. When a socket is detected, Emerson stops and enters its plug-alignment routine. After switching to the small antenna, the drive motors are driven forward for very short bursts, stepping forward about a half-centimeter at a time. At each point, a measurement is taken from the electromagnetic field detector. When eight measurements have been taken, the largest measurement is found, and Emerson steps back to that point. The detector is turned on again, and if a field is detected, the Z-axis stepper motor is engaged, constantly checking for power on the plug. When power is detected, the stepper motor is

5. See Note 1
6. Available at http://www.mygreenlee.com/GreenleeDotCom/Products/main.shtml?greenlee_category_id=6&product_category=162&adodb_next_page=1&adodb_next_page=2&portalProcess_2=showGreenleeProductTemplate&upc_number=12727.

stopped and Emerson enters charging mode. At the moment this is simply timed, but eventually the output of the charger status LED will be captured in order to determine directly when the batteries are charged. The switching harness is triggered, and the batteries are switched to the charger, while the microcontroller is switched to the adjustable 7.5 V AC adaptor. The motor driver's power lines are left unconnected, as the main drive motors are not usable until the plug is disengaged. When time is up, the stepper motor is driven backwards until the plug power sensor shows no power on the line. The flag for powerseeking mode is disengaged, and obstacle avoidance mode resumes.

Unfortunately, much of the powerseeking behavior does not work consistently, or occasionally at all. Getting a stable, reliable signal from the detector is difficult, especially when the battery voltage is low, and difficulties with the wall-following algorithm have made socket-finding even more problematic. Hopefully, with more fine-tuning of the code and tweaking of the detector, this can be fixed.

Conclusion & Future Work

While Emerson is not exactly a success at this point, he has a lot of potential. The mobile platform is very solid, and all of the infrastructure for detecting and utilizing wall sockets is present. With more work and refining, full operation seems entirely possible. Eventually, I would like to replace the hacked live wire detector with a sensor circuit designed using some of the experiences gained here as a guide. The key to successful operation here is fine-grained detection of electromagnetic fields, and to do that an integrated, custom sensor is really needed. As an autonomously self-recharging robot platform is a valuable and versatile thing to have, I intend to continue work on Emerson. It could serve as a base for any number of interesting and useful projects, and with a bit more work it will be able to fulfill the promise it has now.

Appendix A: Code

```
#include <avr/io.h>
#include "PVR.h"


/*****************
Left Turn Function
*****************/
void lturn(int speed){
      TCC0_CCA = 5000;                                  //Set motor speed
      TCC0_CCB = 5000;
      PORTB_OUT = 0x16;                                 //Turn left
}


/******************
Right Turn Function
******************/
void rturn(int speed){
      TCC0_CCA = 5000;                                  //Set motor speed
      TCC0_CCB = 5000;
      PORTB_OUT = 0x19;                                 //Turn right
}


/******************
Drive Stop Function
******************/
void drivestop(void){
      PORTB_OUT = 0x10;                                 //Stop drive motors
}


/*********************
Drive Forward Function
*********************/
void driveforward(int speed){
      TCC0_CCA = 0.95*speed;                            //Set speed
      TCC0_CCB = speed;
      PORTB_OUT = 0x1A;                                 //Set direction = forward
}


/*********************
Drive Reverse Function
*********************/
void drivereverse(int speed){
      TCC0_CCA = speed;                                 //Set speed
      TCC0_CCB = speed;
      PORTB_OUT = 0x15;                                 //Set direction = reverse
}


/***********************
Z-Stepper Drive
***********************/
void stepdrive(int steps){
      int stepdex=0;
```

```
	if (steps>0){
		while (stepdex<steps){
			PORTF_OUT = 0x01;
			delay_ms(15);
			PORTF_OUT = 0x05;
			delay_ms(15);
			PORTF_OUT = 0x06;
			delay_ms(15);
			PORTF_OUT = 0x0A;
			delay_ms(15);
			stepdex=stepdex+1;
			int pow = (PORTH_IN & 0x04)>>2;

			if (pow == 1){

				PORTH_OUT |= 0x02;
				lcdData(0x01);                    //Clear LCD
				lcdGoto(0,0);                     //Go to LCD top left
				lcdString("Charging");
				lcdGoto(1,0);
				lcdString("Battery");
				delay_ms(120000);
			}
		}
	}
	if (steps<0){
		while (stepdex>steps){
			PORTF_OUT = 0x0A;
			delay_ms(15);
			PORTF_OUT = 0x06;
			delay_ms(15);
			PORTF_OUT = 0x05;
			delay_ms(15);
			PORTF_OUT = 0x01;
			delay_ms(15);
			stepdex=stepdex-1;
			int pow = (PORTH_IN & 0x04)>>2;

			if (pow == 1){

				PORTH_OUT &= 0xFD;
				stepdex=steps;
				powerseek=0;
			}

			}
	}
}

/************************
Battery Check
************************/
void battcheck(void){
	int battdisp = ADCA7();
	int voltdisp;
	lcdData(0x01);                                //Clear LCD
	lcdGoto(0,0);                                 //Go to LCD top left
	lcdString("Battery Level:");
```

```
	for(int i=0;i<4;i++){
		voltdisp = ADCA7();
		battdisp = (voltdisp+3*battdisp)/4;
	}

	if(battdisp>3180){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("100%");
	}
	if((battdisp>3160) & (battdisp<=3180)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("95%");
	}
	if((battdisp>3140) & (battdisp<=3160)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("90%");
	}
	if((battdisp>3120) & (battdisp<=3140)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("85%");
	}
	if((battdisp>3100) & (battdisp<=3120)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("80%");
	}
	if((battdisp>3080) & (battdisp<=3100)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("75%");
	}
	if((battdisp>3060) & (battdisp<=3080)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("70%");
	}
	if((battdisp>3040) & (battdisp<=3060)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("65%");
	}
	if((battdisp>3020) & (battdisp<=3040)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("60%");
	}
	if((battdisp>3000) & (battdisp<=3020)){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("55%");
	}
	if(battdisp<=3000){
		lcdGoto(1,0);                                   //Go to LCD top left
		lcdString("LOW");
	}
}

/*********************
Plug Finding Algorithm
*********************/
void plugfind(void){
	PORTH_OUT &= 0xFE;
	int xfind=1;
	int yfind=1;
	int emfind;
```

```
int plug=0;
int cnt=0;
while ((xfind=1) & (cnt<8)){
      int xdex=0;
      int xdat[]={0, 0, 0, 0, 0, 0, 0, 0};
      while(xdex<9){
            xdat[xdex] = ADCA4();
            delay_ms(100);
            driveforward(3000);
            delay_ms(30);
            drivestop();
            delay_ms(50);

            xdex=xdex+1;
      }

      int xmax=xdat[0];
      int dex=1;
      xdex=0;

      while(dex<9){
            if(xdat[dex]>xmax){
                  xmax=xdat[dex];
                  xdex=dex;
            }
            dex=dex+1;
      }

      drivereverse(3000);
      delay_ms(30*(7-xdex));
      drivestop();

      emfind = ADCA4();
      if (emfind>1000){
            xfind=0;
            cnt=0;
            plug=1;
            stepdrive(400);
            stepdrive(-200);
      }


      drivereverse(3000);
      delay_ms(30*xdex);
      drivestop();

      cnt=cnt+1;
}
drivestop();


/*while ((yfind=1) & (cnt<8)){
      int ydex=0;
      int ydat=[8];

      while(ydex<9){
            ydat[ydex] = ADCA4();

            ServoD5(-10);
```

```
                  ydex=ydex+1;
            }

            int ymax=ydat[0];
            int dex=1;
            ydex=0;

            while(dex<9){
                  if(ydat[dex]>ymax){
                        ymax=ydat[dex];
                        ydex=dex;
                  }
                  dex=dex+1;
            }

            ServoD5(10*(7-ydex));

            emfind = ADCA4();
            if (emfind>1000){
                  yfind=0;
            }

            ServoD5(10*xdex);

            cnt=cnt+1;
      }
      */


}

/*************************
Main Robot Operation Code:
*************************/

void main(void){

      /****************
      Initialize System
      ****************/

      xmegaInit();                  //setup XMega
      delayInit();                  //setup delay functions
      ServoCInit();                 //setup PORTC Servos
      ServoDInit();                 //setup PORTD Servos
      ADCAInit();                   //setup PORTA analong readings
      lcdInit();                    //setup LCD on PORTK
      lcdString("Emerson");         //display startup text
      lcdGoto(1,0);
      lcdString("Active");
      PORTQ_DIR |= 0x01;            //set Q0 (LED) as output
      PORTH_DIR = 0x03;
      PORTA_DIR = 0x00;             //set Port A as inputs
      PORTB_DIR = 0xFF;       //set Port B as outputs for motor driver control
      PORTF_DIR = 0xFF;       //set Port F as outputs for stepper driver control
      PORTB_OUT &= 0x00;            //clear Port B
```

```
PORTF_OUT &= 0x00;            //clear Port F
int i = 0;                    //declare & zero counter variable
int lir;                      //declare left IR sensor variable
int rir;                      //declare right IR sensor variable
int r_obs = 0;                //declare & clear right obstacle flag
int l_obs = 0;                //declare & clear left obstacle flag
int l = 0;                    //declare & zero left IR averaging variable
int r = 0;                    //declare & zero right IR averaging variable
int powerseek=0;              //declare & clear power seeking flag
int bump = 0;                 //declare bump sensor variable
int em;
int emav=0;
int a = 5000;
int t = 0;
int fwir;
int bwir;
int volt;
int batt;
int pow;
int first=1;
TCC0_CCA = a;                                          //Right
TCC0_CCB = a;                                          //Left
batt = 3500;
PORTH_OUT &= 0xFE;
/***********************************
Initial Waiting Mode
***********************************/

while((bump<2000)& (first=1)){

      bump = ADCA3();

      /************
      Test Area
      ************/

      //powerseek = 0;

      /************
      End Test Area
      ************/


}

battcheck();
delay_ms(3000);

lcdData(0x01);                                         //Clear LCD
lcdGoto(0,0);                                          //Go to LCD top left
lcdString("Obstacle");
lcdGoto(1,0);
lcdString("Avoidance");

delay_ms(1000);

/***********************************
Obstacle Avoidance - Autonomous Mode
***********************************/
```

```
      while(powerseek==0){                //While power seeking flag is clear


            bump = ADCA3();               //Get bump sensor circuit value

            if ((bump>2400) & (bump<3000)){
                                          //If front left bump sensor triggers
                  drivereverse(5000);
                  delay_ms(300);
                  rturn(0.5);
                  delay_ms(1000);
            }

            if (bump>3400){               //If front right bump sensor triggers
                  drivereverse(5000);
                  delay_ms(300);
                  lturn(0.5);
                  delay_ms(1000);
            }

            if ((bump>750) & (bump<950)){      //If side right bump sensor triggers
                  drivereverse(5000);
                  delay_ms(300);
                  lturn(0.5);
                  delay_ms(1000);
            }

            if ((bump>480) & (bump<600)){       //If side left bump sensor triggers
                  drivereverse(5000);
                  delay_ms(300);
                  rturn(0.5);
                  delay_ms(1000);
            }

            lir = ADCA1();                              //Get left IR sensor value

            rir = ADCA2();                              //Get right IR sensor value
            l = (l + lir);                              //Add left sensor value for
averaging
            r = (r + rir);                              //Add right sensor value for
averaging
            i = i + 1;                                  //Increment counter

            if (i>3){                                   //When counter gets to four
                  l = l/4;                              //Average left IR
                  r = r/4;                              //Average right IR
                  if (l > 1200)                         //If obstacle to left
                        l_obs = 1;                      //Set left obstacle flag
                  else                                  //Otherwise
                        l_obs = 0;                      //Clear left obstacle flag
                  if (r > 1200)                         //If obstacle to right
                        r_obs = 1;                      //Set right obstacle flag
                  else                                        //Otherwise
                        r_obs = 0;                            //Clear right obstacle
flag

                  if ((l_obs==0) & (r_obs==0)){             //If no obstacles
                        a = ((14000+(-10)*(l+r)/2)+a)/2;
```

```
                  //Take average of sensor readings, convert to speed
                  driveforward(a);

            }

            if ((l_obs==1) & (r_obs==1)){       //If obstacle on both sides
                  if (l > r)                    //If left obstacles closer
than right
                        r_obs = 0;              //Set left obstacle flag
                  else                          //Otherwise
                        l_obs = 0;              //Set right obstacle flag
            }

            l = 0;                              //Clear sensor variables
            r = 0;
            i = 0;
      }


      if (r_obs==1){                            //If right obstacle flag set
            drivestop();                        //Stop
            delay_ms(100);                      /Wait 100 milliseconds
            lturn(0.5);                         //Turn left half speed
            delay_ms(1000);                     //For 1 second
            drivestop();                        //Stop
            delay_ms(100);                      //Wait 100 miliseconds
            r_obs = 0;                          //Clear obstacle flags
            l_obs = 0;
      }
      if (l_obs==1){                            //If left obstacle flag set
            drivestop();                        //Stop
            delay_ms(100);                      //Wait 100 milliseconds
            rturn(0.5);                         //Turn right half speed
            delay_ms(1000);                     //For 1 second
            drivestop();
            delay_ms(100);                      //Wait 100 milisecond
            r_obs = 0;                          //Clear obstacle flags
            l_obs = 0;
      }


      /********
      Demo Code
      ********

      t = t+1;
      if (t>300){
            powerseek = 1;
            t = 0;
      }

      ************
      End Demo Code
      ************/

      volt = ADCA7();
      batt = (volt+3*batt)/4;

      if (batt < 3000){
```

```
            powerseek = 1;
      }


}


/************************
Power Seeking Mode
************************/
drivestop();

lcdData(0x01);                                          //Clear LCD
lcdGoto(0,0);                                           //Go to LCD top left
lcdString("Battery Low!");
lcdGoto(0,0);
lcdString("Seeking Outlet");


delay_ms(5000);
PORTH_OUT |= 0x01;

while(powerseek==1){

      fwir = ADCA5();                     //Wall Following
      bwir = ADCA6();
      TCC0_CCB = 2000+(2300-fwir)*2-(2300-bwir)*2-(2300-(bwir+fwir)/2);
      TCC0_CCA = 2000-(2300-fwir)*2+(2300-bwir)*2+(2300-(bwir+fwir)/2);

      PORTB_OUT = 0x1A;                   //Set direction = forward

      bump = ADCA3();                     //Get bump sensor circuit value

      if ((bump>2400) & (bump<3000)){//If front left bump sensor triggers
            drivereverse(5000);
            delay_ms(300);
            rturn(0.5);
            delay_ms(1000);
      }

      if (bump>3400){               //If front right bump sensor triggers
            drivereverse(5000);
            delay_ms(300);
            lturn(0.5);
            delay_ms(1000);
      }

      if ((bump>750) & (bump<950)){     //If side right bump sensor triggers
            drivereverse(5000);
            delay_ms(300);
            lturn(0.5);
            delay_ms(1000);
      }

      if ((bump>480) & (bump<600)){       //If side left bump sensor triggers
            drivereverse(5000);
            delay_ms(300);
            rturn(0.5);
            delay_ms(1000);
```

```
}

em = ADCA4();
emav = (6*emav + em)/7;

if(emav>1000){
      drivestop();
      emav = 0;
      i=0;
      while(i<5){
            em = ADCA4();
            emav = (4*emav + em)/5;
            i=i+1;
      }
      if(emav>1000){
            lcdData(0x01);                    //Clear LCD
            lcdGoto(0,0);                     //Go to LCD top left
            lcdString("Socket Found!");
            lcdGoto(1,0);
            lcdString("Aligning...");
            plugfind();
            first=0;
      }



}

volt = ADCA7();

if (volt<1500){
      lcdData(0x01);                          //Clear LCD
      lcdGoto(0,0);                           //Go to LCD top left
      lcdString("Out of Power");
      lcdGoto(1,0);
      lcdString("Please Assist");
      while(1){
      }
}

/********
Demo Code
********

t = t + 1;

if(t>100){

      drivestop();

      lcdData(0x01);                          //Clear LCD
      lcdGoto(0,0);                           //Go to LCD top left
      lcdString("Antenna");
      delay_ms(2000);
      lcdGoto(1,0);
      lcdString("Large");
      delay_ms(500);
      PORTH_OUT = 0x01;
      delay_ms(5000);
```

```
            lcdGoto(1,0);
            lcdString("Small");
            delay_ms(500);
            PORTH_OUT = 0x00;
            delay_ms(7000);


            lcdData(0x01);                              //Clear LCD
            lcdGoto(0,0);                               //Go to LCD top left
            lcdString("Plug Actuator");
            lcdGoto(1,0);
            lcdString("Demonstration");
            delay_ms(3000);

            stepdrive(300);
            delay_ms(1000);
            stepdrive(-300);
            delay_ms(5000);

            ServoD5(-100);
            delay_ms(6000);
            ServoD5(100);
            delay_ms(6000);
            ServoD5(2);


            lcdData(0x01);                              //Clear LCD
            lcdGoto(0,0);                               //Go to LCD top left
            lcdString("Plug In");
            lcdGoto(1,0);
            lcdString("To Charge");

            while(1){


                  pow = (PORTH_IN & 0x04)>>2;

                  if (pow == 1){

                        PORTH_OUT |= 0x02;
                        lcdData(0x01);                  //Clear LCD
                        lcdGoto(0,0);                   //Go to LCD top left
                        lcdString("Charging");
                        lcdGoto(1,0);
                        lcdString("Battery");
                        while(1){
                        }
                  }
            }

      }

       ************
      End Demo Code
      ************/

   }

}
```