# W.E.N.C.H
**(Water Environment Navigable Catering Hydrobot)**

Shuguang Feng
REU 2003
Final Report
August 25, 2003

# Table of Contents

## Abstract

WENCH is an autonomous food service robot. It is designed to function as a water based delivery vehicle. It is capable of receiving voice commands, locating and "tracking" a human master, and eventually delivering its payload. WENCH utilizes ultrasonic rangers, CdS cells, pressure sensors, and voice recognition to achieve it's desired behavior. Its sensor suite is complemented by an Atmel ATmega128 microcontroller that does the main processing and interfaces with a custom built motor driver board and a Voice Direct II voice recognition board.

Although the functionality of her electronic and mechanical systems have exceeded expectations, the "tracking" aspect of her behavior remains inconsistent at best. The added complexity that the fluid dynamics of water adds to the controls challenge proved to be insurmountable given the time available this summer. This particular aspect of the project remains open as an area of possible future exploration.

# Executive Summary

WENCH tackles the task of delivering food and drink to persons relaxing in the pool. It attempts to realize this goal by using a host of relatively inexpensive sensors to mimic the functionality of more expensive camera based vision systems. With this in mind the project set out to create a robot that would act as a vehicle to facilitate research into different areas of machine intelligence and hardware design as well as one that, when completed, would also be a functional and practical robot. Features like custom made food and drink sensors and a voice recognition system from Sensory Inc. were incorporated to add to WENCH's user friendliness.

The Atmel ATmega128 microcontroller serves as the nerve center of WENCH's electrical system. It provides the majority of the processing power and interacts with the onboard voice recognition system as well as the custom built motor driver board in addition to WENCH's sensor array. The sensor suite includes ultrasonic rangers, CdS cells, pressure sensors, and a Voice Direct II board from Sensory Inc.. All these contribute to the robots ability to complete its task. For the most part WENCH relies on her forward facing sonar modules to detect and track its master. In the process the remaining sensors provide data about its environment that allows it to make intelligent decisions regarding which behavior to exihibit and at what time.

With the exception of a reliable tracking mechanism, due to the control issues involved when dealing with motion in water, the remaining project goals were met over the course of WENCH's design cycle.

# Introduction

WENCH is a product of laziness. It was inspired by a desire to create a robotic servant that delivers food to human masters lounging in the pool. Designed as an autonomous air-boat, the goal was to give WENCH the ability to navigate the surface of a swimming pool in search of its master. Another one of the main project goals was to achieve this behavior without the use of expensive camera based vision systems, electing instead to use more economical sensors.

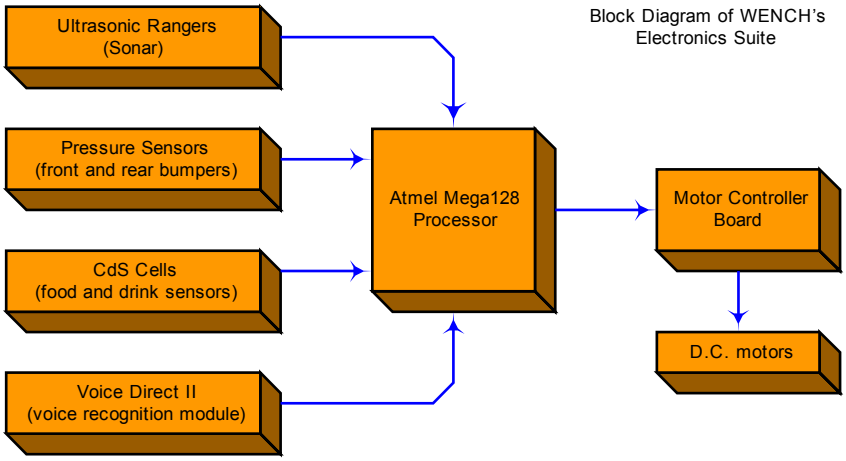
This report documents all salient portions of the project. It provides detailed descriptions of the integrated system, mobile platform, and sensors suite. In addition it records many of the difficulties encountered at each stage of the project and presents possible areas for future research.

# Integrated System

WENCH is powered by an Atmel ATmega128 microprocessor. The microcontroller was purchased from OptiCompo as part of their letATwork II development board. The letATwork II board plugs in to a custom built circuit board designed using Protel 99SE and milled out on IMDL's T-tech Quick Circuit machine. This circuit board greatly reduced the amount of wiring required which in turn reduced the occurrence of error associated with bad connections and faulty cabling. In addition the board also acts as a motherboard of sorts, to which all the individual sensors connect, allowing them to interact with the letATwork daughter board and the ATmega128 processor.

Alongside the microprocessor board WENCH also possesses two additional circuit boards which round out its electronics suite. The first of theses is a Voice Direct II module purchased from Sensory Inc. which supplies WENCH with her voice recognition capabilities and is controlled directly by the ATmega128. The second is another custom designed circuit board which contains the motor drivers, which control the dc motors that drive the propellers, and the voltage regulators that provide power to WENCH's sensor array.

With the information provided by her sensor suite, which includes ultrasonic sonar rangers, pressure sensitive bump sensors, CdS cells, and voice recognition module WENCH is able to intelligently carry out her duties. The data gathered from the sonar modules and the bump sensors provides WENCH with information about its environment as it moves about in search of its master. The voice recognition capability not only allows WENCH to receive verbal commands that dictate its behavior, but also provides a means for it to distinguish between inanimate objects in the pool and the person it is trying to serve.



# Mobile Platform

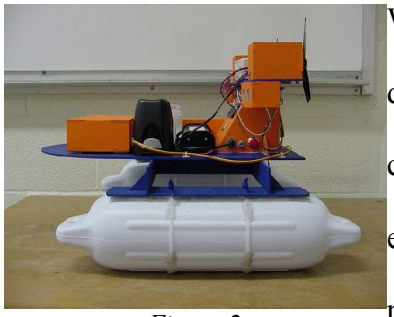


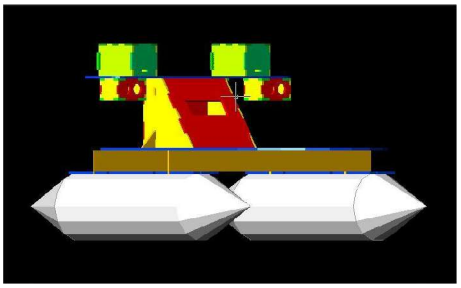
*Figure 2*

WENCH's mobile platform is based on a simple catamaran design. This air-boat style frame was chosen over conventional boat designs to allow for possible future expansion. Without the handicap of having to rely on a propeller and rudder, WENCH can in theory be adapted to function on both land and water as an amphibious delivery system.

WENCH's platform is constructed mainly from 5-ply airplane grade balsa wood. The design of individual components were made in AutoCAD and fabricated using IMDL 's T-tech machine. The pieces were assembled using waterproof wood glue and joints were sealed using silicone caulk. Custom fit cases were also designed to house the electronics and sensors. The entire platform is kept afloat by two PVC boat fenders purchased at WAL-MART. Although wood would not ordinarily lend itself to the construction of water based robots, for the purposes of this project it was sufficient. Keeping in mind that the entire platform sits above the water, an advantage of the catamaran design, and given the budget constraints for WENCH, wood presented a suitable alternative to more expensive materials.

# Actuation

*Movement and Steering*



Two 5" 3-blade push propellers mounted on high speed d.c. motors provide the propulsion. Steering is achieved by running the two motors at different speeds with appropriate correction factors taken into account to compensate for varying motor strengths and torquing caused by having both propellers spinning in the same direction. Interestingly enough the motors chosen were actually surplus 5.25" disk drive motors that happened to be available. The choice of these motors was not only cost effective but also insured that replacements would be available in the event that one, or both, happened to fail during testing.

*Motor Control*

The motors are interfaced with the microprocessor board via a separate motor controller board designed around a DMOS Full-Bridge PWM Motor Driver from Allegro MicroSystems (A3959SB). The drivers provide up to 3A continuous current and can handle up to 6A spikes. These motor driver chips are considerably more sophisticated than others that have been used in the lab and presented quite a few challenges during the design phase, not the least of which was locating an half-ohm resistor.

*Specifications*

The motors used measured 2.5" long and 1.75" in diameter, with a shaft diameter of 0.5". Mounting screwholes with diameters of .125" located .125" from the circumfrence of the

motor were also available and utilized in mounting the motors to WENCH's motor boxes. Since they were surplus motors no data sheets were available and attempts to locate them online via serial numbers proved unfruitful. Although tests were performed to determine steady state current draw and stall current, approximately .125A and 1.5A respectively (at 10V), similar tests to determine torque and rpm's were not feasible.

## Sensors

*Sonar*

Two Devantech SRF04 Ultrasonic Ranger modules from Mark III Robotics provide obstacle detection. The SRF04 interfaced with the letATWork II board's input capture port provides surprisingly consistent distance readings. The effective range is from about 3cm to 3m and is incredibly linear. For future applications an array of sonar transmitters could easily be integrated as a sonar beacon designating the location of a base station.

*Voice Recogntion Module*

A Voice Direct II module from Sensory Inc. provides WENCH with her voice recognition capabilities. The module is activated after sonar detects an object within a certain threshold (currently set in code as 15") and is told to enter into listening mode. The user is then prompted for a “wait” command. If the command is heard the Voice Direct II board sends an external interrupt to the ATmega128, the assumption is made that a human master has been located and WENCH proceeds to make its delivery. Otherwise, the assumption is that sonar has detected a "stationary" object and WENCH will execute object avoidance code.

Interfacing the Voice Direct II with the microcontroller was straightforward and only required the use of a few I/O and a couple external interrupt pins on the ATmega128. However, since the Voice Direct II operates off of 3V ttl signals and the ATmega128 on the letATwork II uses 5V ttl, how to safely send signals back and forth became an interesting issue. Having overlooked this detail initially, the first Voice Direct II board was effectively destroyed within 10 hours of arriving from Sensory, having been connected directly to the ATmega128's 5V I/O pins. With the second board an Octal Line Buffer chip (SN742451) was used to isolate the two voltage levels while still allowing the two boards to send signals back and forth.

*Collision Sensors*

A strip of pressure sensitive resistors from Interlink Electronics is affixed on both the front and rear styrofoam bumpers. With a simple voltage divider circuit and a couple of the ATmega128 A/D ports these sensors are used to detect collisions with stationary objects that managed to circumvent the object avoidance algorithm. In particular, for safety reasons when a rear collision is detected both motors cut off and the entire system goes into standby until the user reactivates it.

One of the difficulties associated with implementing this type of sensor is the need for the sensor to be located relatively close to the surface of the water. This raises significant concerns about waterproofing, which led to the selection of the Interlink sensors,. which comes in a sealed package, over the more affordable micro switches.

*Food and Drink Sensors*

CdS cells proved to be a simple yet effective means of detecting the presence of food and drink. Given that the resistance of CdS cells vary with the amount of light to which they are exposed, implementation consisted only of placing them in a simple voltage divider circuit connected to the ATmega128 A/D ports. With some straightforward software calibration to account for different lighting conditions these CdS sensors are both cheap and reliable.

## Behaviors

*Ideal*

WENCH is designed to remain dormant at a base station until its services are requested. When activated (by placing food in containers), WENCH will move randomly about the pool using sonar to locate a human target. When a target is acquired WENCH attempts to place itself within reach of its master. It will remain stationary until a food item is removed or until it is explicitly told to leave by the master. In the event that there is still food remaining WENCH will move a set distance away and attempt to locate a different master. Once all of the food is removed WENCH will proceed to the base station to “refuel”.

*Achieved*

The biggest hurdle encountered in trying to achive WENCH's ideal behavior was the added complexity that movement in water presented. The fluid dynamics of moving

about in a pool has proven to be a controls nightmare. Even the last minute addition of plexiglass keels down the center of WENCH's platform, although helpful, did not eliminate WENCH's susceptibility to harmful drifting. Although the sonar "tracking" algorithm appears to work well on land, without reasonably precise control of WENCH's movements in water effective tracking in a pool setting has proved elusive. Time and cost restraints have also prevented the construction of a base station. Apart from these setbacks, WENCH's other behaviors have been fully achieved. It is able to receive voice commands that dicate it's actions (ie. "wait" for user to remove food), detect collisions with objects and execute appropriate avoidance code, and tailor it's behavior according to the absence or presence of food detected by the CdS cells.

## Experimental Layout and Results

Experimentation was performed at each stage of this project. WENCH was broken down into standalone subsystems that were designed, prototyped, and tested individually before being incorporated into the final product. This process began with the microprocessor when the letATwork II board arrived from OptiCompo. Next, the motor driver system was designed, then the sensors were chosen and tested, and finally the voice recognition system was implemented. Even after each component had been tested and added to WENCH, testing continued and revisions were occasionally made to improve functionality.

When the different compononents were finally integrated into one system WENCH underwent simple behavioral testing. At first it was simply placed on a table in an empty

classroom while it executed rudimentary tracking software. This gave an idea of whether simple object tracking could be realized using only two forward facing sonar modules. The results were favorable. Both the accuracy of the data and the rate at which it could be reliably aquired were encouraging. Using a running average filter and simple error checking, dependable readings could be made about 2-3 times per second. WENCH's reaction time to input provided from the pressure sensor bumpers and CdS cells was also acceptable given that polling was used since A/D pins cannot generate interrupts.

Naturally, WENCH next underwent testing on water. A small vinyl kiddie pool (8' diameter) was purchased from WALMART and set up outside. Although WENCH appeared to be sufficiently agile, capable of moving from one end of the pool to the other in seconds and turning on command, it was less adept at executing the remaining behavior code and actually tracking a person. Testing continued and eventually was relocated to a full size swimming pool when it became apparent that the kiddie pool no longer afforded adequate room for WENCH to move about. Despite continual modifications made to the tracking algorithm and days of testing, WENCH's ability to find a human target was inconsistent at best. It was obvious that the catamaran platform was very susceptible to drift currents and that after WENCH had built up enough momentum in one direction any attempts to turn it by redirecting the motors would only cause her to rotate and drift sideways. The amount of time and distance required for WENCH to change headings, which given the tracking algorithm based on only sonar was something that needed to be done as often as once every couple of seconds, was unacceptable. Even the addition of keels down the center of WENCH's body, although

considerably dampening some of effects of cross currents, did not offer dramatic improvements to it's turning ability. It seems as if a rudder system, an element purposely omitted during the design phase, may be the only remedy.

## Conclusion

In summary, although WENCH did not completely live up to all expectations, as a whole the project did meet a majority of the goals it set out to achieve. The successful integration of the ATmega128 microcontroller with the assorted sensors on board supplied in and of itself many opportunities to experience the challenges associated with hardware design, among them the mixed blessing of learning Protel. Even the construction of the physical platform in addition to imparting modest AutoCAD proficiency, gave insight into the different design elements that must be considered in creating a robotic platform.

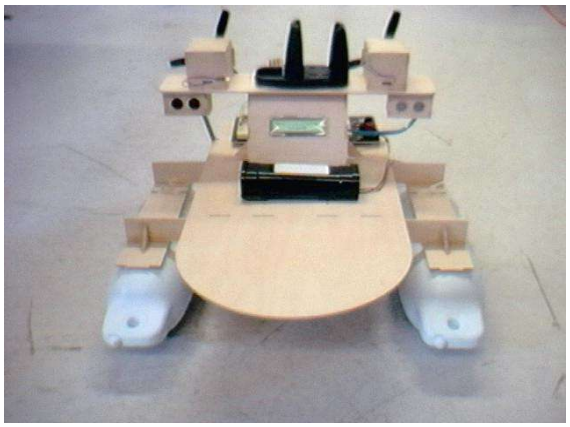
At present all of WENCH's electrical systems meet initial design goals and function as expected. This includes PWM controlled motors via the motor driver board, reliable "tracking" data from the ultrasonic rangers, accurate voice recognition performed by the Voice Direct II (requires a relatively quiet setting), collision detection from the pressure sensors, and lastly food detection using the CdS cells. What remains to be achieved is the high level behavior of actually being able to consistently position itself next to the person it is tracking, which could be realized as mentioned earlier by a rudder system, or even possibly by inserting feedback gathered from devices like an accelerometer into the control loop. Regardless of the solution, this last piece of the puzzle could open doors

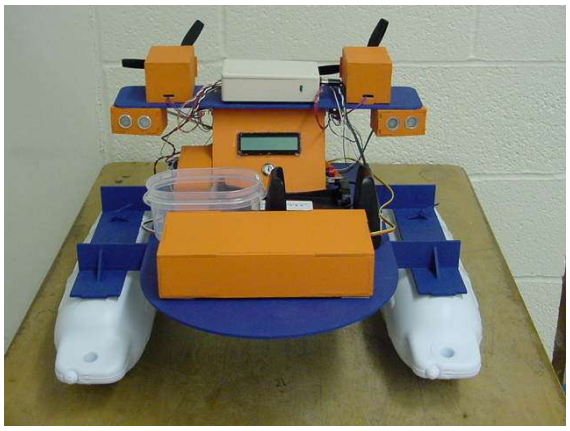
for future exploration, including but not limited to making WENCH amphibious so it can move easily between a deck and the pool or even increasing its size to function in other outdoor environments like a small pond or lake.

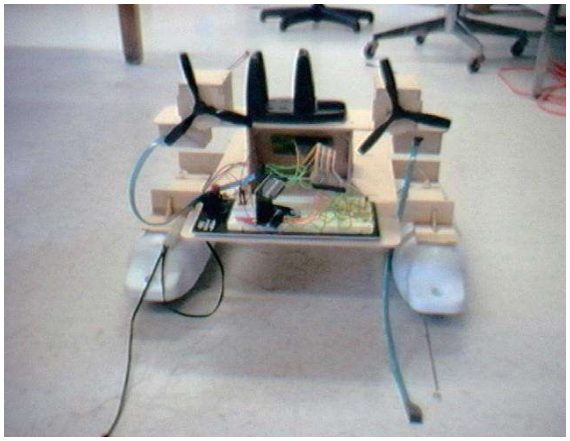
Were I to start this project over part of me wants to say I would have opted to create a land based version of WENCH if not only to avoid the horrors of testing a water based robot. With that said I would also have sped up the design and construction phase, concerning myself less with making the perfect design choices and more with quickly producing a testable prototype. Given the design iterations that almost any robot must endure I feel that I spent an inordinate amount of time debating over trivial issues which later required revisions anyway. Yet, as a whole considering the time and budget constraints I am well pleased with the way WENCH turned out and entertain thoughts of possible future improvements.

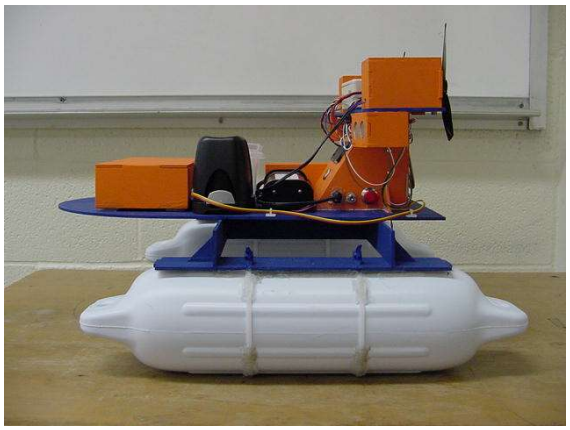
# APPENDIX A

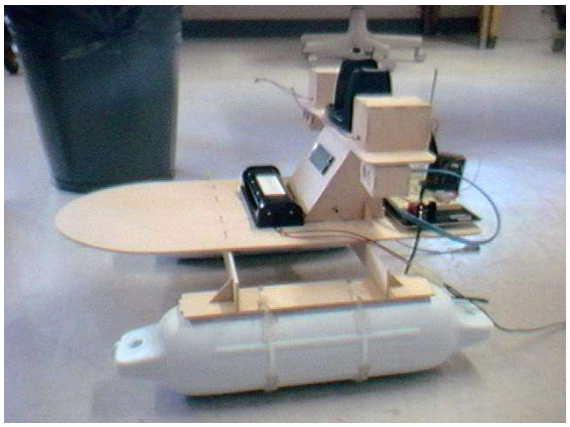
*Pictures of WENCH in development*

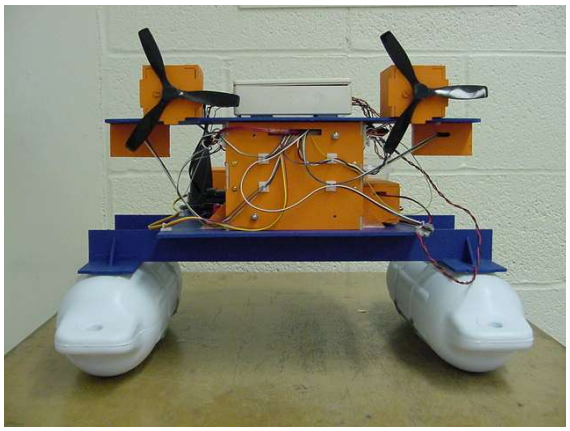






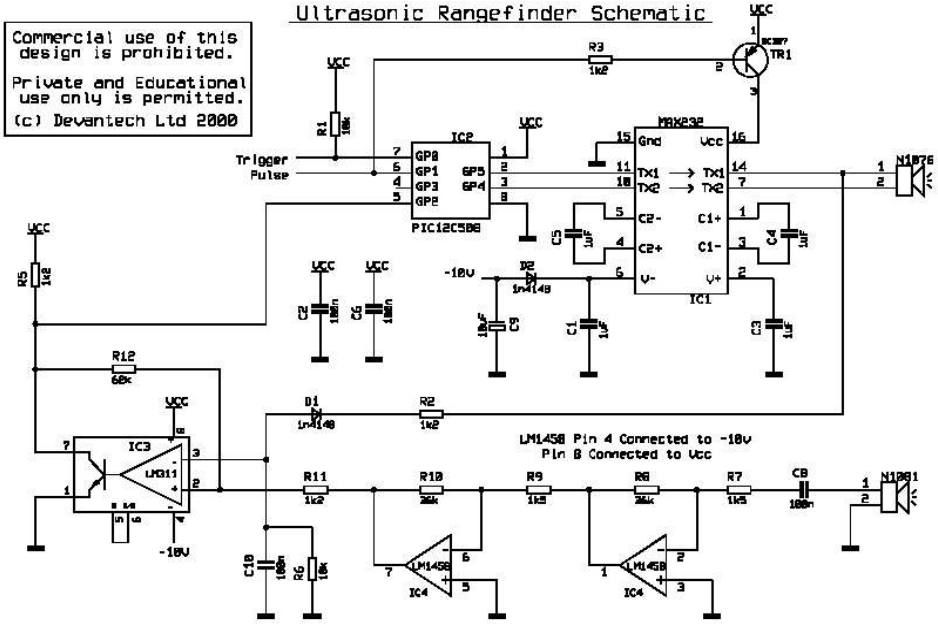


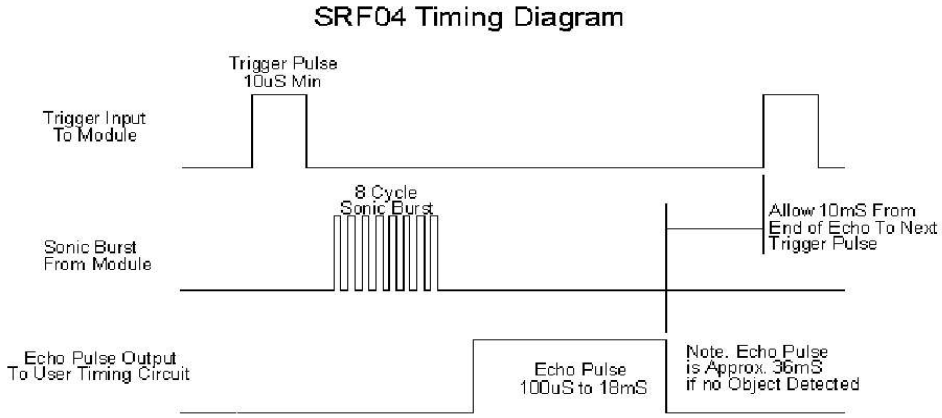




# APPENDIX B

*Sonar documentation (Devantech SRF04)*

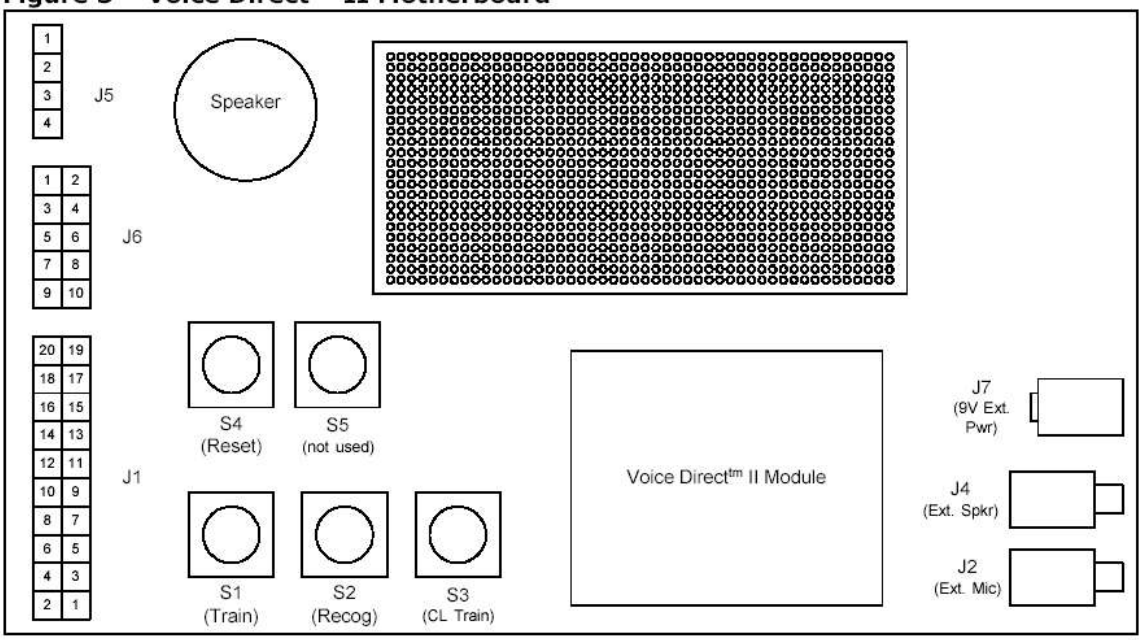


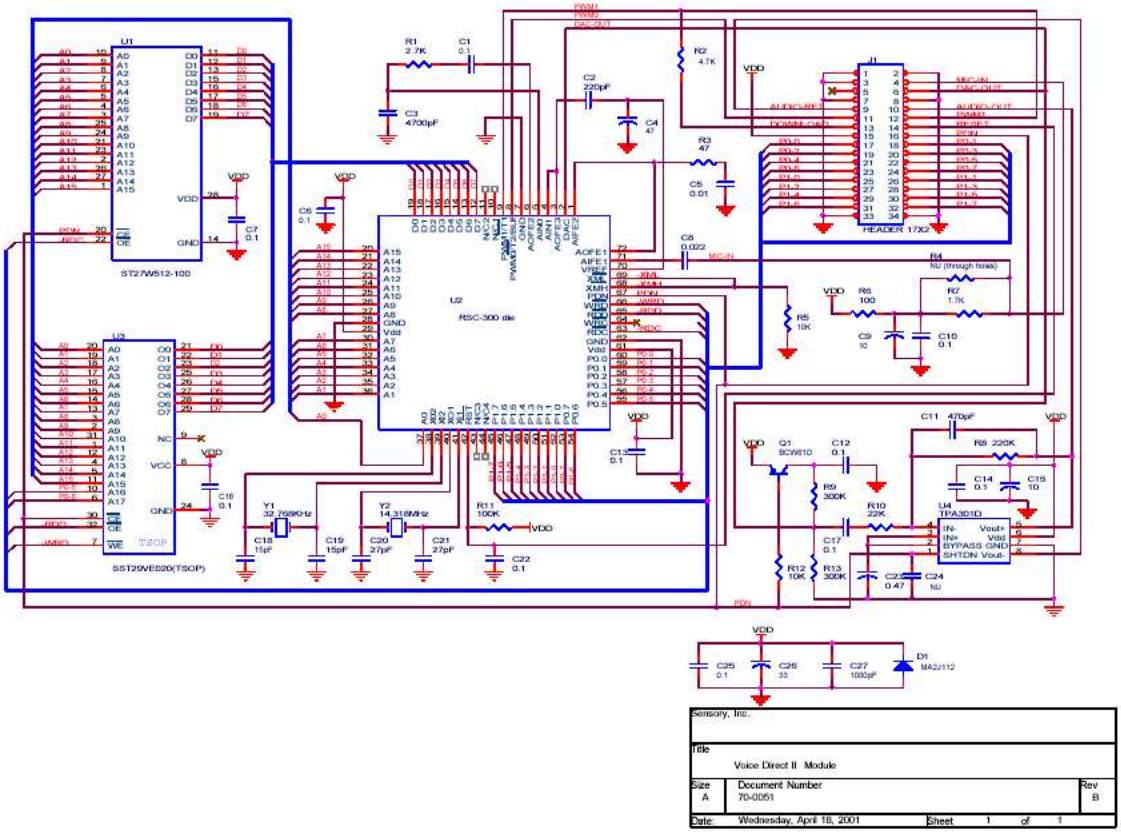



# APPENDIX C

*Voice Direct II documentation*

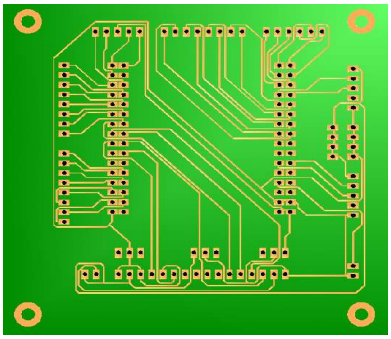
**Figure 3 – Voice Direct™ II Motherboard**



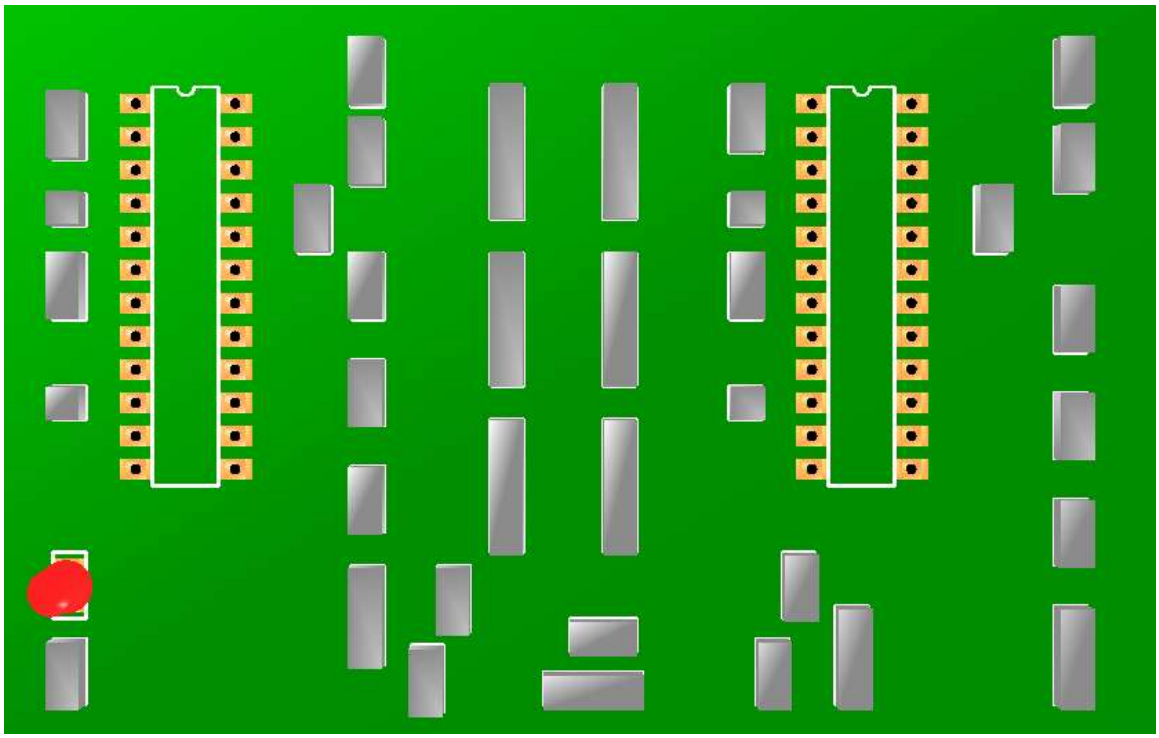



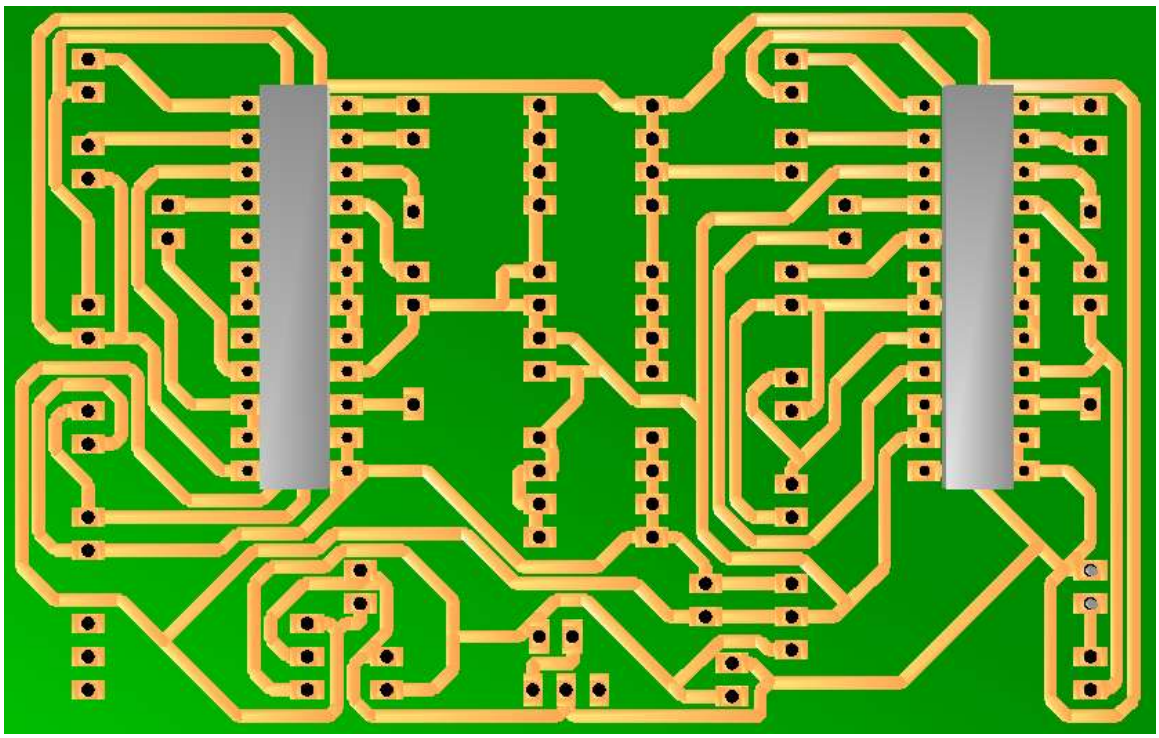
APPENDIX D

*Motherboard design*

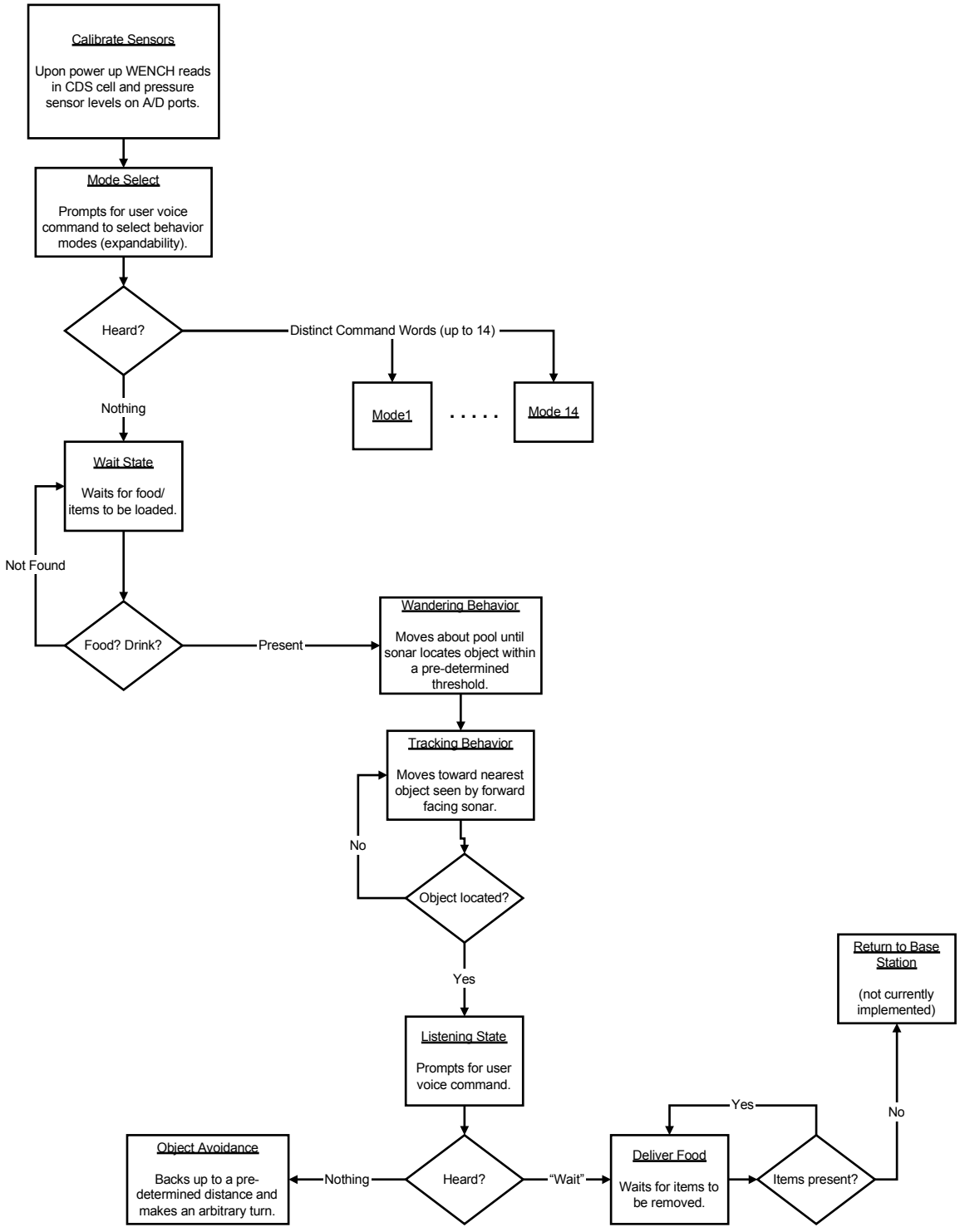
*Motor driver design*





# APPENDIX E

*Block diagram of WENCH's behavior algorithm.*



# APPENDIX F

*Source code*

```
/*
 * Wench.c
 *
 * Author: Shuguang Feng
 *
 * Wench Behavior (PWM, LCD, SONAR, A/D, Tracking, Voice Recognition)
 * 8-5-03
 */

 /********************/
 /**PIN DEFINITIONS**/
 /********************/
 /*
  LCD
                PORTA0 - DB4
                PORTA1 - DB5
                PORTA2 - DB6
                PORTA3 - DB7
                PORTA4 - RS             0-command register, 1-data register
                PORTA5 - EN

  SONAR
                Right Sonar
                PORTD4(IC1) - Echo Output
                PORTD5      - Pulse Trigger Input

                Left Sonar
                PORTE7(IC3) - Echo Output
                PORTE6      - Pulse Trigger Input

  PWM
                PORTB7(OC2)         -       Right motor Enable
                PORTB6              -       Right motor Phase
                PORTB5              -       Left motor Phase
                PORTB4(OC0)         -       Left motor Enable

  A/D
                PORTF0(AD0)         -       Drink sensor
                PORTF1(AD1)         -       Food sensor

  VOICE
                PORTE5(INT5)        -       Motor Demo INT
                PORTE4(INT4)        -       Wait INT
                PORTE3              -       RECOG pin
                PORTE0              -       ~RESET pin
*/


/******************/
/**HEADER FILES**/
```

```
/*****************/
#include <io.h>                 //IO

#include <interrupt.h>          //Sonar includes
#include <stdlib.h>
#include <sig-avr.h>
#include <inttypes.h>           //LCD includes

/*****************************/
/**CONSTANT DEFINITIONS**/
/*****************************/
#define ENABLE 0x20                     //LCD enable pin used to "clock in" instructions

#define OFF 0x00                        //motor speeds
#define SLOWEST 0x2A
#define SLOW 0x55
#define MEDIUM 0xAA
#define FAST 0xCF
#define FORWARD         0               //motor directions
#define REVERSE         1

#define LEFT            0               //object location
#define CENTER          1
#define RIGHT           2

#define CORRECT_R 0x25                  //compensates for torquing caused by both motors spinning
#define CORRECT_L 0x00                  //in the same direction

#define NOTHING         0               //tracking constants
#define PERSON          1
#define OBSTACLE        2

/*****************/
/**PROTOTYPES**/
/*****************/
void init_lcd(void);                    //LCD prototypes
void lcd_delay(void);
void lcd_send_str(char *s);
void lcd_send_byte(uint8_t val);
void lcd_send_command(uint8_t val);
char * int_to_str(int val);
int calc_len(int val);

unsigned int read_sonar_R(void);        //sonar prototypes
unsigned int read_sonar_L(void);
void init_ic(void);
unsigned int calc_distance(unsigned int val);
void check_sonar(void);

void init_pwm(void);                    //pwm prototypes
void pwm_duty_L(unsigned int d);
void pwm_duty_R(unsigned int d);
void change_speed(unsigned int L, unsigned int R);
```

```
void init_ad(void);                         // A/D prototypes
void check_drink(void);
void check_food(void);
void check_fb(void);
void check_rb(void);

void init_voice(void);                      //voice prototypes
void que_voice(void);
void reset_voice(void);

void sonar_test(void);                        //main prototypes
void motor_test(void);
void avoid(void);
void track(void);
void calibrate_AD(void);
void rear_collision(void);
void wander(void);

/***********************/
/**GLOBAL VARIABLES**/
/***********************/
volatile unsigned int risingedge_R = 0;                 //sonar ISR variables
volatile unsigned int pulsewidth_R = 0;
volatile unsigned int risingedge_L = 0;
volatile unsigned int pulsewidth_L = 0;

volatile unsigned int dist_R;                           //sonar readings
volatile unsigned int dist_L;

volatile unsigned int speed_L = 0, speed_R = 0;         //motor speeds
volatile unsigned int dir_L = 0, dir_R=0;               //motor directions

volatile unsigned int obj_loc = CENTER;                 //tracking variables
volatile int        located = NOTHING;
volatile int        task_complete = 0;

volatile int        food;                               // AD
volatile int        drink;
volatile int        front_bumper = 0;
volatile int        rear_bumper = 0;

volatile unsigned long          food_base;
volatile unsigned long          drink_base;
volatile unsigned long          fb_base;
volatile unsigned long          rb_base;

volatile int        demo = 0;                           //mode selection

/********/
/**ISRs**/
/********/
SIGNAL(SIG_INPUT_CAPTURE1) {                            //sonar ISR
                if (TCCR1B & (1<<ICES)) {               //checks for rising edge(when sonar
                risingedge_R = ICR1;                    //recieve circuitry turns on)
```

```
		TCCR1B &= ~(1<<ICES);			//sets input caputre to look for
		} else {				//falling edge
		pulsewidth_R = ICR1 - risingedge_R;	//when falling edge is detected
		TCCR1B |= (1<<ICES);			//the pulsewidth is calculated
		}
}

SIGNAL(SIG_INPUT_CAPTURE3) {			//same as above, but for IC3 (left)
		if (TCCR3B & (1<<ICES)) {
		risingedge_L = ICR3;
		TCCR3B &= ~(1<<ICES);
		} else {
		pulsewidth_L = ICR3 - risingedge_L;
		TCCR3B |= (1<<ICES);
		}
}

SIGNAL(SIG_INTERRUPT4) {			//INT4
		change_speed(OFF,OFF);
		located = PERSON;
}

SIGNAL(SIG_INTERRUPT5) {			//INT5
		demo = 1;
}


/********************/
/**MAIN PROGRAM**/
/********************/
int main(void)
{
		unsigned long wait1=0, wait2=0;
		init_lcd()				//lcd initialization
		init_pwm();				//initializes pwm
		init_ad();				//initializes A/D
		init_ic();				//initializes input capture
		init_voice();				//initializes voice recognition
		sei();					//unmasks interrupts

		int i = 0;
		for(i=20; i > 0; i--)			//calibrates sensor readings
		{
			calibrate_AD();
			for (wait1=0 ; wait1 < 250; wait1++)		//delay loop
			{
				for(wait2=0 ; wait2< 60000; wait2++)
				{
					;
				}
			}
		}

		check_food();			//saves calibration values
```

```
food_base = ADCW;
check_drink();
drink_base = ADCW;
check_fb();
fb_base = ADCW;
check_rb();
rb_base = ADCW;
rear_bumper = 0;
front_bumper = 0;

lcd_send_command(0x01);
lcd_send_str("Values Saved...");
lcd_send_command(0xC0);
lcd_send_str("Run Demo?");

que_voice();                    //prompts for mode select
if(demo == 1)
{
        lcd_send_command(0x01);
        lcd_send_str("Motor test...");
        motor_test();
}

lcd_send_command(0x01);                 //waits for WENCH to be loaded
lcd_send_str("Load WENCH");
check_drink();
check_food();
while(drink != 1 && food !=1)
{
        check_food();
        check_drink();
}

lcd_send_command(0x01);                 //prepares to search
lcd_send_str("Searching in...");
for(i=5; i>0; i--)
{
        lcd_send_command(0x8F);
        lcd_send_str(int_to_str(i));
        for (wait1=0 ; wait1 < 800; wait1++)                    //delay loop
        {

                for(wait2=0 ; wait2< 60000; wait2++)
                {
                        ;
                }
        }
}

while (!task_complete)
{
        wander();
        while(located == NOTHING)               //"tracks" the nearest object
        {
```

```
                    track();

                    if(rear_bumper == 1)
                                        rear_collision();
                    else if(front_bumper == 1)
                    {
                                        change_speed(OFF, OFF);
                                        located = OBSTACLE;
                                        front_bumper = 0;
                    }

                    change_speed(OFF,OFF);
                     lcd_send_command(0x01);

                   if(obj_loc == LEFT)
                    {
                                        lcd_send_str("Compensating R");
                                        change_speed(OFF, OFF);
                                        dir_L = FORWARD;
                                        dir_R = REVERSE;
                                        change_speed(SLOW,SLOW);
                                        change_speed(OFF,OFF);
                                        dir_L = FORWARD;
                                        dir_L = FORWARD;
                    }
                    else if(obj_loc == RIGHT)
                    {
                                        lcd_send_str("Compensating L");
                                        change_speed(OFF, OFF);
                                        dir_L = REVERSE;
                                        dir_R = FORWARD;
                                        change_speed(SLOW, SLOW);
                                        change_speed(OFF,OFF);
                                        dir_L = FORWARD;
                                        dir_L = FORWARD;
                    }

}

lcd_send_command(0x01);
lcd_send_str("Listening...");
que_voice();                        //prompts for user command

if(located == OBSTACLE)
{
                    lcd_send_command(0x01);
                    lcd_send_str("Avoiding object");
                    avoid();
                    located = NOTHING;
}
else if(located == PERSON)
{
                    lcd_send_command(0x01);
                    lcd_send_str("Remove items");
```

```
                                        while(drink)
                                        {
                                                check_drink();
                                        }
                                        while(food)
                                        {
                                                check_food();
                                        }
                                        task_complete = 1;
                                }
                }

                lcd_send_command(0x01);
                lcd_send_str("Task Complete");

                for( ; ; ) {}         //inifinite loop
                return 0;
}

void wander(void)                 //wandering behavior
{
                lcd_send_command(0x01);
                lcd_send_str("Wandering...");
                change_speed(OFF, OFF);
                dir_L = FORWARD;
                dir_R = FORWARD;
                change_speed(MEDIUM,MEDIUM);
                check_sonar();
                while(dist_R > 100 && dist_L > 100)          //moves forward until object is located within
                {                                            //100"
                        check_sonar();
                }
                change_speed(OFF,OFF);
                dir_L = REVERSE;
                dir_R = REVERSE;
                change_speed(MEDIUM, MEDIUM);          //"braking"
                change_speed(OFF,OFF);
                dir_L = FORWARD;
                dir_R = FORWARD;
}

void rear_collision(void)       //shuts down motors after rear collision, resumes after user taps front bumper
{
                unsigned long wait1=0, wait2=0;
                int i=0;
                change_speed(OFF,OFF);
                lcd_send_command(0x01);
                lcd_send_str("****COLLISION****");
                while(front_bumper == 0){ check_fb(); }
                lcd_send_command(0x01);
                lcd_send_str("Resuming in...");
                for(i=5; i>0; i--)
                {
                        lcd_send_command(0x8E);
```

```
                        lcd_send_str(int_to_str(i));
                        for (wait1=0 ; wait1 < 800; wait1++)            //delay loop
                        {
                                for(wait2=0 ; wait2< 60000; wait2++)
                                {
                                        ;
                                }
                        }
                }
                front_bumper = 0;
                rear_bumper = 0;
}


void track(void)                //tracking behavior
{
                check_sonar();

                lcd_send_command(0x01);
                lcd_send_command(0xC0);
                lcd_send_str(int_to_str(dist_R));
                lcd_send_command(0xCD);
                lcd_send_str(int_to_str(dist_L));

                lcd_send_command(0x80);

                if(dist_R > (dist_L+3))         //left sonar sees the closest object
                 {
                        obj_loc = LEFT;
                        lcd_send_str("Turning Left");
                        if(dist_L >= 50)
                        {
                                change_speed(MEDIUM, FAST);
                        }
                        else if(dist_L >= 15)
                        {
                                change_speed(OFF, OFF);
                                dir_L = REVERSE;
                                dir_R = FORWARD;
                                change_speed(SLOW, SLOW);
                        }
                        else if(dist_L < 15)            //object located and engines cut off
                        {
                                 change_speed(OFF, OFF);
                                 located = OBSTACLE;
                        }
                 }
                else if(dist_L > (dist_R+3))    //right sonar sees the closest object
                {
                        obj_loc = RIGHT;
                        lcd_send_str("Turning Right");
                        if(dist_R >= 50)
                        {
                                change_speed(FAST, MEDIUM);
```

```
                        }
                        else if(dist_R >= 15)
                        {
                                change_speed(OFF, OFF);
                                dir_L = FORWARD;
                                dir_R = REVERSE;
                                change_speed(SLOW,SLOW);
                        }
                        else if(dist_R < 15)            //object located and engines cut off
                        {
                                 change_speed(OFF, OFF);
                                 located = OBSTACLE;
                        }
                }
                else            //both sonar see the "same" object
                {
                        obj_loc = CENTER;
                        lcd_send_str("Moving Forward");

                        if(dist_R >= 100 && dist_L >= 100)
                        {
                                change_speed(MEDIUM, MEDIUM);
                        }
                        else if(dist_R >= 15 && dist_L >=15)
                        {
                                change_speed(SLOW, SLOW);
                        }
                        else if(dist_R < 15 || dist_L < 15) //object located and engines cut off
                        {
                                 change_speed(OFF, OFF);
                                 located = OBSTACLE;
                        }
                }
}

void avoid(void)                //object avoidance code
{
                unsigned long wait1=0, wait2=0;

                lcd_send_command(0xC0);
                lcd_send_str("Backing...");

                change_speed(OFF,OFF);      //backs up until object is >30 inches away
                dir_L=REVERSE;
                dir_R=REVERSE;
                change_speed(MEDIUM,MEDIUM);

                while((dist_L < 30 || dist_R < 30) && rear_bumper == 0)
                {
                        check_rb();
                        if(rear_bumper == 1)
                                rear_collision();
                        check_sonar();
                }
```

```
		if(dist_R<dist_L)
		{
			lcd_send_command(0xC0);			//turns left
			lcd_send_str("Turning...Left");
			change_speed(FAST,OFF);
			dir_R = FORWARD;
			change_speed(FAST,FAST);
		}
		else
		{
			lcd_send_command(0xC0);			//turns right
			lcd_send_str("Turning...Right");
			change_speed(OFF,FAST);
			dir_L = FORWARD;
			change_speed(FAST,FAST);
		}

		for (wait1=0 ; wait1 < 500; wait1++)			//delay loop
		{

			for(wait2=0 ; wait2< 60000; wait2++)
			{
				;
			}
		}

		change_speed(OFF,OFF);			//cuts off motors
		dir_L=FORWARD;
		dir_R=FORWARD;
}

void calibrate_AD(void)		//used to set base values for CDS cells
{
		lcd_send_command(0x01);
		lcd_send_str("D:");
		lcd_send_command(0x88);
		lcd_send_str("FB:");
		lcd_send_command(0xC0);
		lcd_send_str("F:");
		lcd_send_command(0xC8);
		lcd_send_str("RB:");

		ADMUX=0x40;				//reads in and displays drink CDS
		ADCSR |= (1<<ADSC);
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}
		lcd_send_command(0x83);
		lcd_send_str(int_to_str(ADCW));

		ADMUX=0x41;				//reads in and displays food CDS
		ADCSR |= (1<<ADSC);
```

```
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}
		lcd_send_command(0xC3);
		lcd_send_str(int_to_str(ADCW));

		ADMUX=0x43;				//reads in and displays front bumper
		ADCSR |= (1<<ADSC);
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}
		lcd_send_command(0x8C);
		lcd_send_str(int_to_str(ADCW));

		ADMUX=0x44;				//reads in and displays rear bumper
		ADCSR |= (1<<ADSC);
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}
		lcd_send_command(0xCC);
		lcd_send_str(int_to_str(ADCW));
}

void sonar_test(void)				//calibrates sonar (debugging use)
{
		unsigned long wait1 = 0;				//delay loop variables
		unsigned long wait2 = 0;

		int iteration=0;

		lcd_send_command(0x01);				//lcd format
		lcd_send_str("Right sonar");
		lcd_send_command(0xC0);
		lcd_send_str("Left sonar");

		for(iteration=0; iteration <=25; iteration++)
		{
			check_sonar();

			lcd_send_command(0x8C);
			lcd_send_str("    ");
			lcd_send_command(0x8C);
			lcd_send_str(int_to_str(dist_R));
			lcd_send_command(0xCB);
			lcd_send_str("    ");
			lcd_send_command(0xCB);
			lcd_send_str(int_to_str(dist_L));

			for (wait1=0 ; wait1 < 125; wait1++)		//delay loop
			{
```

```
                                        for(wait2=0 ; wait2< 60000; wait2++)
                                        {
                                                ;
                                        }
                        }
                }
}

void motor_test(void)           //motor demo
{
                unsigned long wait1=0,wait2=0;

                lcd_send_command(0x01);
                lcd_send_str("Motor test...");
                lcd_send_command(0xC0);
                lcd_send_str("Turning left");

                change_speed(OFF,OFF);
                dir_L = REVERSE;
                dir_R = FORWARD;
                change_speed(FAST,FAST);

                for (wait1=0 ; wait1 < 1500; wait1++)                   //delay loop
                {

                        for(wait2=0 ; wait2< 60000; wait2++)
                        {
                                ;
                        }
                }

                lcd_send_command(0xC0);
                lcd_send_str("Turning right");

                change_speed(OFF, OFF);
                dir_L = FORWARD;
                dir_R = REVERSE;
                change_speed(FAST, FAST);

                for (wait1=0 ; wait1 < 1500; wait1++)                   //delay loop
                {
                        for(wait2=0 ; wait2< 60000; wait2++)
                        {
                                ;
                        }
                }

                lcd_send_command(0xC0);
                lcd_send_str("Moving forward");

                change_speed(OFF, OFF);
                dir_L = FORWARD;
                dir_R = FORWARD;
                change_speed(MEDIUM, MEDIUM);
```

```
		for (wait1=0 ; wait1 < 1500; wait1++)			//delay loop
		{

			for(wait2=0 ; wait2< 60000; wait2++)
			{
				;
			}
		}

		lcd_send_command(0xC0);
		lcd_send_str("Backing up     ");

		change_speed(OFF, OFF);
		dir_L = REVERSE;
		dir_R = REVERSE;
		change_speed(MEDIUM, MEDIUM);

		for (wait1=0 ; wait1 < 1500; wait1++)			//delay loop
		{

			for(wait2=0 ; wait2< 60000; wait2++)
			{
				;
			}
		}

		change_speed(OFF,OFF);
		dir_L = FORWARD;
		dir_R = FORWARD;
}

/************************/
/**VOICE RECOGNITION**/
/************************/
void init_voice(void)
{
		EIMSK = 0x30;
		EICRB = 0x0F;		//sets ISC51=ISC50=ISC41=ISC40=1 for rising edge
					//detection

		DDRE |= (1<<3);		//sets PORTE3 as the RECOG trigger pin.
		DDRE |= (1<<0);		//sets PORTE0 as the ~RESET pin

		PORTE |= (1<<3);
		PORTE |= (1<<0);
}

void que_voice(void)
{
		unsigned long wait1=0, wait2=0;

		PORTE &= ~(1<<3);			//pulls RECOG to gnd and signals
							//voice board to listen for command
```

```
		for (wait1=0 ; wait1 < 25; wait1++)				//delay loop
		{
			for(wait2=0 ; wait2< 64000; wait2++)
			{
				;
			}
		}

		PORTE |= (1<<3);

		for (wait1=0 ; wait1 < 3000; wait1++)				//delay loop
		{
			for(wait2=0 ; wait2< 60000; wait2++)
			{
				;
			}
		}

		reset_voice();
}

void reset_voice(void)
{
		unsigned long wait1=0, wait2=0;

		EIMSK = 0x00;

		PORTE &= ~(1<<0);		//pulls RESET to gnd and resets
						//voice board
		for (wait1=0 ; wait1 < 25; wait1++)			//delay loop
		{
			for(wait2=0 ; wait2< 64000; wait2++)
			{
				;
			}
		}

		PORTE |= (1<<0);

		for (wait1=0 ; wait1 < 1000; wait1++)				//delay loop
		{

			for(wait2=0 ; wait2< 60000; wait2++)
			{
				;
			}
		}

		EIFR |= (1<<INTF5);
		EIFR |= (1<<INTF4);
		EIMSK = 0x30;
}
```

```
/******************/
/**A/D FUNCTIONS**/
/******************/
void init_ad(void)
{
		ADCSR = 0x87;			//AD enabled, prescale=128, single conversion
		ADMUX = 0x40;			//initialized to channel 0
		DDRF = 0x00;
}

void check_drink(void)
{
		ADMUX = 0x40;			//selects A/D channel 0
		ADCSR |= (1<<ADSC);		//sets start conversion bit
		while(!(ADCSR & (1<<ADIF)))  //waits until conversion completes
		{
			;
		}
		if(ADCW > drink_base + 200)			//reads final value
			drink=1;
		else
			drink=0;
}

void check_food(void)				//similar to check_drink
{
		ADMUX = 0x41;			//selects A/D channel1
		ADCSR |= (1<<ADSC);
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}
		if(ADCW > food_base + 200)
			food=1;
		else
			food=0;
}

void check_fb(void)		//checks front bumper
{
		ADMUX=0x43;

		ADCSR |= (1<<ADSC);
		while(!(ADCSR & (1<<ADIF)))
		{
			;
		}

		if(ADCW < (fb_base-50))
		{
			front_bumper = 1;
		}
}
```

```
void check_rb(void)                     //checks rear bumper
{
        ADMUX=0x44;

        ADCSR |= (1<<ADSC);
        while(!(ADCSR & (1<<ADIF)))
        {
                ;
        }

        if(ADCW < (rb_base-50))
        {
                rear_bumper = 1;
        }
}

/*********************/
/**PWM FUNCTIONS**/
/*********************/
void init_pwm(void)
{
        DDRB = 0xF0;            //set OC0 and OC2 to outputs for 8-bit PWM signal, set
                                //PORTB5 and PORTB6 as outputs for direction pins

        TCCR0 = 0x64;           //phase correct PWM, non-inverted, no prescaler (counter0)
        TCCR2 = 0x64;           //phase correct PWM, non-inverted, no prescaler (counter2)

        OCR0 = 0x00;  //initialized to 0% duty cycle.
        OCR2 = 0x00;  //
}

void change_speed(unsigned int L, unsigned int R)
{
        unsigned long wait1 = 0;                                        //delay loop variables
        unsigned long wait2 = 0;                                        //
        unsigned int new_speed_L = 0, new_speed_R = 0;

        check_fb();
        check_rb();

        if(dir_L==FORWARD)                      //change left motor direction
        {
                PORTB &= ~(1<<5);
        }
        if(dir_L==REVERSE)
        {
                PORTB |= (1<<5);
        }

        if(dir_R==FORWARD)                      //change right motor direction
        {
                PORTB &= ~(1<<6);
        }
```

```
		if(dir_R==REVERSE)
		{
			PORTB |= (1<<6);
		}

		if(L == OFF)
			new_speed_L = OFF;
		else
			new_speed_L = L + CORRECT_L;

		if(R == OFF)
			 new_speed_R = OFF;
		else
			new_speed_R = R + CORRECT_R;

		while( ((speed_L != new_speed_L || speed_R != new_speed_R) && rear_bumper == 0 &&
			front_bumper == 0)
		    || ((speed_L != new_speed_L || speed_R != new_speed_R) && new_speed_L ==
			OFF && new_speed_R == OFF) )
		{
			if(new_speed_L>speed_L)
				speed_L++;			//accelerate left motor
			else if(new_speed_L<speed_L)
				speed_L--;			//decelerate left motor
			if(new_speed_R>speed_R)
				speed_R++;			//accelerate right motor
			else if(new_speed_R<speed_R)
				speed_R--;			//decelerate right motor

			for (wait1=0 ; wait1 < 5; wait1++)		//delay loop
			{
				for(wait2=0 ; wait2< 60000; wait2++)
				{
					;
				}
			}
			pwm_duty_L(speed_L);
			pwm_duty_R(speed_R);
			check_fb();
			check_rb();
		}
}

void pwm_duty_L(unsigned int d)
{
		if(d<=255 && d>=0)
		{
			OCR0 = d;	//change duty cycle (left motor)
		}
}

void pwm_duty_R(unsigned int d)
{
		if(d<=255 && d>0)
```

```
		{
			OCR2 = d;	//change duty cycle (right motor)
		}
}

/**********************/
/**SONAR FUNCTIONS**/
/**********************/
void init_ic(void)
{
		DDRD = 0x20;		//initializes PORTD bit5 as output to trigger right sonar
		DDRE |= (1<<6);		//initializes PORTE bit6 as output to trigger left sonar
		TCCR1A = 0x00;		//initializes control registers for input capture
		TCCR1B = 0xC3;		//runs TCNT1 counter at 250kHz
		TCCR3A = 0x00;		//initializes control registers for input capture
		TCCR3B = 0xC3;		//runs TCNT1 counter at 250kHz
		TIMSK |= 1<<TICIE1;	//unmasks input compare interrupt enable for right sonar
		ETIMSK |= 1<<TICIE3;	//unmasks input compare interrupt enable for left sonar
}

unsigned int read_sonar_R(void)
{
		int i = 0;
		//delay loop variable

		TIMSK |= 1<<TICIE1;	//unmasks input compare interrupt enable for right sonar
		TCCR1B |= (1<<ICES);	//sets rising edge triggered input capture
		pulsewidth_R = 0;

		PORTD |= 0x20;		//sends pulse to PORTD5 activate sonar module
		for(i = 0; i < 800; i++) {}	//delay (10us min)
		PORTD &= ~(0x20);	//pulls pin back to low
		while (!pulsewidth_R) {;}	//waits for interrupt routine to execute and

		 //update the value of pulsewidth (listens for echo)


		TIMSK &= ~(1<<TICIE1);	//masks input compare interrupt enable for right sonar
		return (calc_distance(pulsewidth_R));
}

unsigned int read_sonar_L(void)		//same as above
{
		int i = 0;
		ETIMSK |= 1<<TICIE3;	//unmasks input compare interrupt enable for left sonar
		TCCR1B |= (1<<ICES);
		pulsewidth_L = 0;

		PORTE |= 0x40;
		for(i = 0; i < 800; i++) {}
		PORTE &= ~(0x40);
		while (!pulsewidth_L) {;}

		ETIMSK &= ~(1<<TICIE3);	//masks input compare interrupt enable for left sonar
```

```
		return (calc_distance(pulsewidth_L));
}

unsigned int calc_distance(unsigned int val){
		int time_us, distance;

		time_us = val * 2;		//250kHz = 4us period....divide by 2 for echo time
		distance = time_us * .011;	//.9ft/ms = .011 inches/us

		return(distance);
}

void check_sonar(void)
{
		#define LEN 4

		unsigned long wait1=0, wait2=0;
		unsigned long i=0;

		unsigned int R_d1, R_d2, L_d1, L_d2;
		unsigned long avg=0;
		unsigned int valid=0;

		while(!valid)
		{

			avg = 0;
			for(i=0;  i < LEN; i++)
			{
				avg += read_sonar_R();
			}
			R_d1=(avg/LEN);

			for (wait1=0 ; wait1 < 10; wait1++)		//delay loop
			{
				for(wait2=0 ; wait2< 60000; wait2++)
				 {
					;
				 }
			}

			avg = 0;
			for(i=0;  i < LEN; i++)
			{
				avg += read_sonar_L();
			}
			L_d1=(avg/LEN);

			for (wait1=0 ; wait1 < 10; wait1++)		//delay loop
			{
				 for(wait2=0 ; wait2< 60000; wait2++)
				  {
					;
```

```
                                        }
                                }

                                avg = 0;
                                for(i=0;  i < LEN; i++)
                                {
                                        avg += read_sonar_R();
                                }
                                R_d2=(avg/LEN);

                                for (wait1=0 ; wait1 < 10; wait1++)                     //delay loop
                                {
                                         for(wait2=0 ; wait2< 60000; wait2++)
                                         {
                                                 ;
                                         }
                                }

                                avg = 0;
                                for(i=0;  i < LEN; i++)
                                {
                                        avg += read_sonar_L();
                                }
                                L_d2=(avg/LEN);

                                if(R_d1>R_d2+2 || R_d2>R_d1+2 || L_d1>L_d2+2 || L_d2>L_d1+2)
                                {
                                        //checks to see if there are erroneous readings
                                        valid=0;
                                }
                                else
                                {
                                        valid=1;
                                }

                }

                dist_R = R_d2;
                dist_L = L_d2;
}


/********************/
/**LCD FUNCTIONS**/
/********************/
void init_lcd(void)             //LCD init function
{
                DDRA = 0x3f;                                    //initializes PORTA for LCD

                lcd_send_command(0x33);                 //enable 4-bit mode
                lcd_send_command(0x32);

                lcd_send_command(0x2c);                 //enable 2-line mode
```

```
		lcd_send_command(0x0f);			//display cursor blink

		lcd_send_command(0x01);			//clear home

}

void lcd_delay(void)		//LCD delay (4bit mode)
{
		uint16_t time1;
		for (time1 = 0; time1 < 65000; time1++);
}

void lcd_send_str(char *s)			//diplays contents of a string
{
	while (*s) lcd_send_byte(*s++);		//calls lcd_send_byte until a null character is reached
}

void lcd_send_byte(uint8_t val)			//sends individual bytes to LCD microcontroller
{
	uint8_t temp = val;		//temp variable holds value of byte that is passed in

	val >>= 4;			//shifts right 4 bits
	val |= 0x10;			//sets RS pin to select DATA
	PORTA = val;			//sends out upper nibble of original byte

	lcd_delay();

	PORTA |= ENABLE;	//clocks in the upper nibble (rising edge)
	PORTA &= ~ENABLE;	//falling edge

	temp &= 0x0f;			//masks upper nibble of original byte
	temp |= 0x10;			//sets RS pin to select DATA
	PORTA = temp;			//sends out lower nibble of original byte

	lcd_delay();

	PORTA |= ENABLE;	//clocks in the upper nibble (rising edge)
	PORTA &= ~ENABLE;	//falling edge

	lcd_delay();
}

void lcd_send_command(uint8_t val)
{
	uint8_t temp = val;

	val >>= 4;			//shifts right 4 bits
	PORTA = val;			//sends out upper nibble of original byte (RS=0, COMMAND)

	lcd_delay();

	PORTA |= ENABLE;	//clocks in the upper nibble (rising edge)
	PORTA &= ~ENABLE;	//falling edge
```

```
	temp &= 0x0f;		//masks upper nibble of original byte
	PORTA = temp;		//sends out lower nibble of original byte (RS=0, COMMAND)

	lcd_delay();

	PORTA |= ENABLE;	//clocks in the upper nibble (rising edge)
	PORTA &= ~ENABLE;	//falling edge

	lcd_delay();
}

char * int_to_str(int val)
{
		int len = calc_len(val);		//determines length of integer
		char str[len+1];			//creates char array of length len + 1;
		int i;					//loop variable

		for (i = len - 1; i >= 0; i--)		//iterates until all digits are converted
		{
			str[i] = val%10 + 0x30;		//converts to ASCII equivalent by adding 0x30
			val /= 10;			//divides number by 10 to advance to next digit
		}

		str[len] = '\0';	//stores null character in the last element of the array

		return(str);
}

int calc_len(int val)
{
		int done=0;			//loop variable
		int length=1;			//initializes length variable

		while(!done)	//calculates length of integer...cannot be greater than 5
		{		//digits (16bit int)
			if(val>=10)
			{
				length++;
				val=val/10;
			}
			else
			{
				done=1;
			}
		}
		return(length);
}
```