ButlerBot

Erik Sjolander

Final Report
EEL5666: IMDL
Fall 2002
Professor: Dr. Arroyo
TAs: Jason Plew, Uriel Rodriguez

Table of Contents

Abstract

ButlerBot is an autonomous beverage-dispensing robot. It is designed to be an automated bartender, finding and filling empty glasses without any human intervention. The basic platform consists of a fairly standard drive system (2 driven wheels with a rear caster) with an onboard beverage container. The beverage is dispensed using a small 12V pump.

Introduction

Keeping all the glasses full at a packed bar is a tough job for any bartender. To help bartenders everywhere keep the patrons happy, I've developed ButlerBot, an autonomous beverage-dispensing system. ButlerBot lives to keep your glass full. Traveling down the bar, it checks the liquid level in each glass it finds, fills if needed, and continues on its way. An on-board beverage container makes ButlerBot truly autonomous, requiring only periodic refilling. A smart line-following system makes getting your glass refilled a simple process – just place your glass on the line, and ButlerBot will stop to fill it the next time it comes by.

Integrated System

The robot consists of several interlinked systems, principally the platform and drive systems; computing hardware and electronics; and sensors. In addition there was a significant amount of software, which is discussed in more depth in a later section.

Platform and Drive Systems

Platform

The platform of ButlerBot required careful design due to the constraints placed on it. Having an onboard beverage container means the robot will have to be able to support a fairly significant amount of weight (around 2lbs for the size of container chosen). However, ButlerBot must be small enough to maneuver on the top of a bar or table. A 2-driven-wheel plus caster design was chosen for its simplicity and relatively small size. Due to the weight of the robot, it was necessary to use a wheeled caster. A caster similar to those on wheeled chairs was chosen for its quality construction and high load rating. Designing the platform in AutoCad and cutting it from model aircraft plywood on the T-Tech was selected as the best construction method to get both strength and small size. A careful design provides the strength required, and the tight tolerances allow the platform to be as small as possible.

The choice of beverage container was influenced by much the same factors that influenced platform design. While a container as large as possible is obviously desirable, the constraints placed on the robot's size force a compromise. A 1-liter Rubbermaid container was chosen as the best compromise, and the platform was designed around this container.

Drive System

The drive system on ButlerBot had to be robust enough to handle the weight placed on it by the beverage container, platform, and battery, and have the power to drive the robot when fully loaded. These needs couldn't be met by hacked servos, so the decision was made to use 12V DC gearhead motors. There are a number of 12V motors available, making the selection more difficult. The final decision was to use motors (from herbach.com) which came with a wheel attached. A 5A-hr 12V lead-acid battery was selected as the power source. While a large battery provides many hours of running time, it adds another 3.5lbs to the robot weight.

Dispensing System

The initial design of ButlerBot called for a gravity-feed system to dispense the beverage. This design required an electronically operated valve. After some research, no suitable valves were found. Rather surprisingly, several suitable pumps were found for very reasonable prices (under $10). Using a pump-based dispensing system has several advantages, including allowing the beverage container to be mounted lower in the platform (for better stability) and much better control of the flow rate when dispensing.

A 12V DC pump (designed as a car windshield-washer pump) was selected. This pump had to be mounted at the bottom of the beverage container, since it can not generate much suction at its inlet. It also doesn't have a check valve built in, which causes the beverage to be siphoned through the outlet if the end of the dispensing tube is lower than the level of liquid in the beverage container. This problem was avoided by positioning the dispensing tube relatively high, and not overfilling the beverage container.

The pump draws about 2.5A when running, which makes designing a circuit to drive it somewhat complex (see the electronics section).

Computing Hardware and Electronics

Microcontroller

The Atmel Mega323 microcontroller on a Progressive Resources MegaAVR-Dev board was chosen for its highly integrated board and relatively powerful processer. In particular, the 2 high-resolution PWM channels proved useful for driving the motors.

Motor Drivers

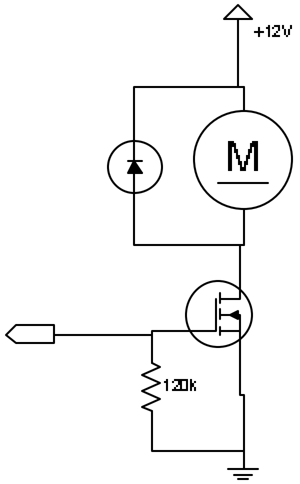
Initially the Allegro 3959 full-bridge DMOS motor driver was my choice. After getting these working (two are required, as they only drive one motor each), I broke one of them. Static electricity is the most likely culprit, but I'm still not completely sure that was the problem. The 3959 requires 6 (or more) external components per chip, which made debugging very difficult. Instead of replacing the broken chip, I decided to switch to a simpler part which requires fewer external components. The motors I purchased only draw about 500mA peak (100mA operating) which made the 3A continuous/6A peak current of the 3959 a bit excessive. The TI L293D was my choice for a replacement, as it will drive two motors, and does not require any external components. It also lists static-protection in its features list, something the Allegro parts don't do. Installing the TI part was much easier, and it has worked flawlessly.

I ran the motor drivers off the 10-bit PWMs in the Mega323. The motors start turning at about a 50% duty-cycle PWM, which made the software a little more difficult.

Pump Driver

The pump I'm using draws about 2.5A continuous, and pumps water quite a bit faster than is needed to fill glasses. I decided to use one of the other PWM channels to control the pump speed. Using a PWM on the pump meant that a relay would be a bad choice, as the mechanical contacts might not switch fast enough. A power FET was the obvious choice, so I went out and bought an IRF510 power MOSFET. After finding and studying the data sheet for this part, I found that it was not guaranteed to deliver the current I needed with a 5V gate drive voltage. Doing research on the web, I noticed that Fairchild

Semiconductor has a free samples program that includes some of their power FETs. After searching through the catalog for a suitable part which they offer samples of, I settled on the HUF76107P3. This is a 20A, 30V power FET which guarantees to be fully "on" with a logic-level gate drive (it also has an impressively low on-resistance of 0.052O). I used a Radio Shack 3A, 400V diode across the motor to handle inductive transients in the circuits, and a 120kO resistor from gate to source to guarantee that the pump stays off if the gate is left open-circuit. This circuit has performed flawlessly in driving the pump. The pump turns on when given a PWM with a duty cycle of more than about 30-40%, and I found that a 50% duty cycle gives a good flow rate for filling glasses.



Sensors

Every robot needs to know what's going on in the world around it, so sensors are an essential part of every design. My robot will use several types of sensors. IR sensors provide short- and intermediate-range object detection, plus line-following abilities. Sonar provides long-range detection, and detection of clear objects. Bump switches serve as a last-resort collision detection system. Liquid level sensors detect when the glass is full when dispensing beverage. These sensors allow the robot to correctly respond to a number of situations.

IR Sensors

In my robot, I use two types of IR sensors. One is a simple IR LED and phototransistor mounted in a convergent housing (the Fairchild QRB1134). The other is a long-range analog IR sensor that includes its own electronics (the Sharp GP2D12).

Fairchild IR

The Fairchild QRB1134 is a relatively simple IR sensor, consisting only of an IR LED and a phototransistor mounted in a plastic case so the focal length where the diode and transistors active regions coincide is about .25" from the front of the lens.
I connected a 150 Ohm resistor in series with the diode to give a nominal diode current of about 30mA, and a 3.3k resistor in series with the collector of the phototransistor, which

gives a reading for currents from 0mA to 1mA (the transistor is specified to sink 0.6mA in its on-state).

I collected data on the operation of these sensors using the 10-bit A/D converter on my Atmel Mega323. The analog reference was tied to +5V with a bypass capacitor.

Below is the table of the data I collected, a graph of the data (white target data in blue, black target data in green), and the corresponding normalized current vs. distance plot from the QRB1134 spec sheet.

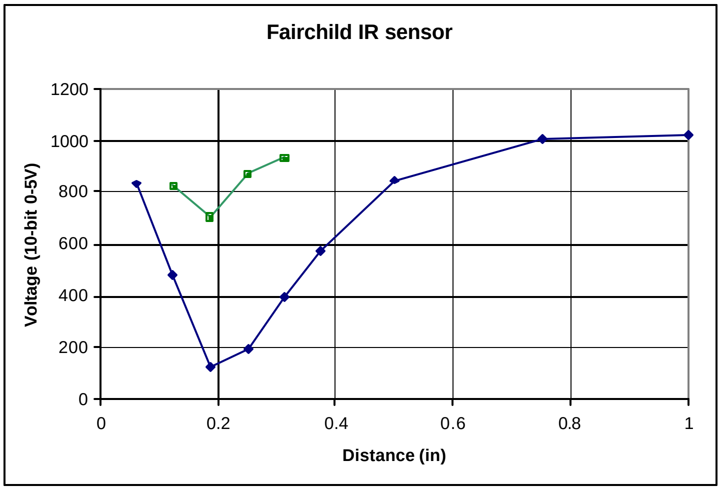
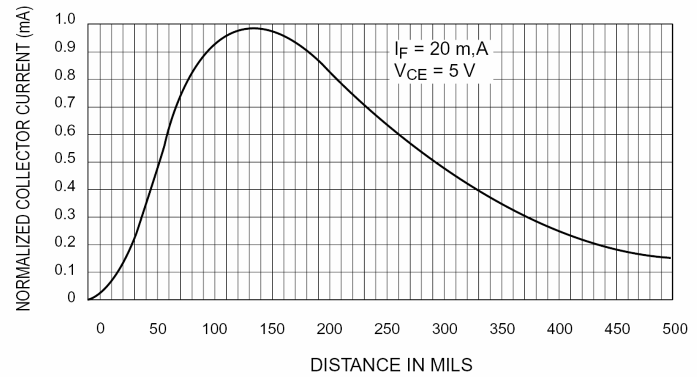


Fig. 5 Normalized Collector Current vs. Distance



The data for this sensor agree well with the theoretical data presented in the spec sheet (note that my graph is inverted since I'm using a pull-up resistor, so higher current corresponds to lower output voltage). This sensor provides good sensitivity over a range of about ¼" to ¾". It would be dangerous for an object to get closer that ¼" to the sensor, since the output voltage for very close distances is indistinguishable from that of long distances. This can be avoided by placing the sensor so that isn't possible.

This sensor is also quite sensitive to color variations in the target. I compared a white sheet of paper to black electric tape (see plot). The variation in the A/D reading was over 500 at 3/16", corresponding to a voltage difference of more than 2.5V. Since they are so sensitive to color variations, these sensors work very well for line-following, and turned out to be fairly tolerant of different surface conditions.

This sensor is easily confused by looking more or less directly at a strong IR source (e.g. my halogen desk light). It gave a reading as low as 4-5 when pointed directly at the light, and gave noticeably lower readings when the target was directly lit by the light. This sensor will probably have serious problems if used outside, but should work well inside at short ranges.

I plan to use three of these sensors to implement line following, and three more to detect the edge of the table so I don't fall off.

Sharp IR

The other IR sensor I'm using is the Sharp GP2D12 long-range analog IR sensor. This sensor has a measuring range of 10-80 cm (about 4-32 inches) according to its specifications. It is very simple to use, only requiring +5V and ground, and giving an analog voltage output on its third pin.

Like the Fairchild sensor, I tested this sensor on my board with +5V as the reference voltage, using the full 10 bits of precision. Below is the table of the raw data I collected, a plot of the data (with the light target in blue and the dark target in pink), and the V_{OUT} vs. distance plot from the GP2D12 spec sheet (note that the units in the spec sheet are cm, while my graph is in inches).

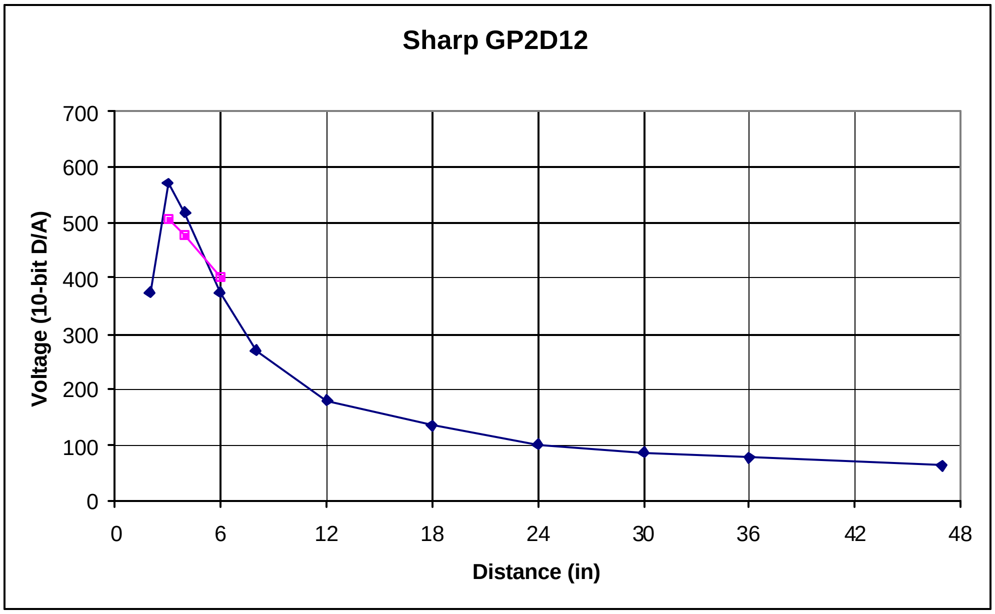
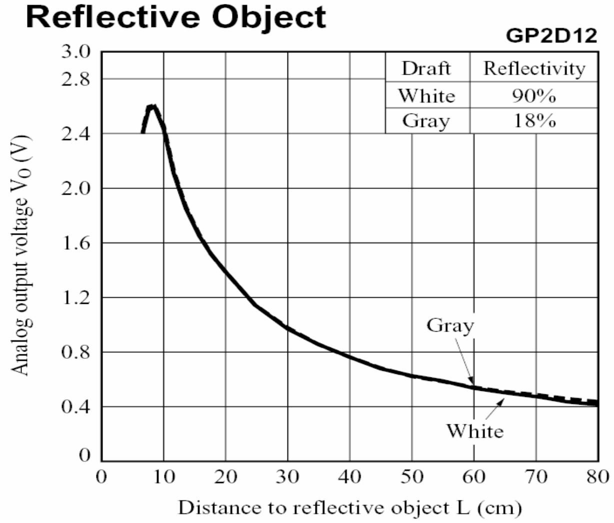


Fig.6 Analog Output Voltage vs.Distance to Reflective Object



This sensor behaves very similarly to the Fairchild, but scaled up by a factor of about 10. It has the same problem with the output voltage dropping if the detected object is too close, so it is important that nothing get closer than about 2” to the sensor.

This sensor shows much nicer, more consistent behavior than the Fairchild. The output does not depend much on the color of the target, or on the angle of incidence. I detected a slight dependence on the color of the target, but the difference is almost within the measurement error as my technique was not that exact. The beam is quite tight, about 2” across at 1’ distance, with sharp edges (about ½” from full-on to full-off). It is also not sensitive to other IR sources. I couldn’t get my halogen desk light to have any significant effect on it. Direct sunlight might cause problems, but in normal indoor lighting interference shouldn’t be a problem.

I plan to use two of these sensors in a cross-eyed configuration to implement object detection and possibly wall-following.

Bump Switches

ButlerBot uses digital ports with internal pull-up resistors to read the values of the switches. Using digital ports means that there is no need for any external components and the data is already in the digital form I need for use in my code. I chose to implement the switches in a wired-or network, with all the left-side switches wired to one pin, and the right-side to another. This allows me to run 4 bump switches on 2 digital ports. Not knowing exactly which of the right-side bump switches was pressed turned out not to be much of a problem, since these switches are a backup to the main obstacle-avoidance system.

Sonar

I purchased a Devantech SRF08 sonar module. This communicates with the controller via the TWI/I2C interface. It has a range of about 2” – 18’, with a resolution of 1cm/1”. Please see my sensor report for more information about the sonar.

Liquid Detector

I need a sensor to detect when the glass I'm filling is full, and when the on-board container is empty. I considered several techniques to measure the liquid level, including sonar, IR and a capacitive sensor. I decided against sonar because of the cost (about $50 for another sonar module) and complexity. IR does not to work reliably on clear liquids. I'm not sure I could design a capacitive sensor that works properly, so I decided to try to find something simpler. In the best KISS tradition, I decided to try using two probe wires in a voltage-divider with a large (1M or 10M) resistor. If the probe wires are not in contact with anything, the pull-up resistor gives +5V on the output. If the probe is in contact with something (say, a beverage) the output voltage is the output of a voltage divider formed by the pull-up resistor and the resistance between the probes. The circuit will never have more than 5uA of current with a 1M resistor, which is small enough not to pose any danger.

In testing, I found that the resistance between the probes was much less than even the 1M resistor. When the probes were in contact with water (filtered tap water), I never got a reading higher than 1V. The was consistently 5V when the probes were not in contact with the liquid. This performance is quite good, and works well for my purposes.

Using a 10M resistor caused some problems, as it would take the circuit almost a second to rise from the low value up to the open-circuit value. I expect that this is due to the capacitance in the circuit, possibly in combination with the 10M input resistance of the analog port. The 1M provides good differentiation between on and off, without these problems, so it is my choice.

Software

The software in ButlerBot was written in C for the gcc compiler. This was chosen for its low price (free) and good support on avrfreaks.net. The software consists of a behavior system based on a simplified version of the system presented by Arroyo and Harrelson in their "Programming Behaviors" PDF available on the IMDL website. The behaviors used a variety of low-level routines to do sensor input and motor control. Line following, glass filling, obstacle avoidance and bump switch/table edge detection were the four behaviors implemented. Each behaviors returns a structure which includes its desired values for the three controls (left motor, right motor, and acceleration) and a confidence number which is used to select the dominant behavior. The selection of a behavior is done by finding the max of the confidence numbers. In real life, this system was found to work quite well. Having acceleration control on the motors prevented rapid speed changes from causing jerky motion with rapid forward/backward changes, and allowed the obstacle avoidance code to appear quite smooth, though it uses a simple off/slow/fast speed system.

The main file of the program is "demo.c", with each behavior and all the low-level routines implemented in separate files. As compiled, the final version of the code was about 4kB in size, which easily fits in the 32kB of flash-rom on the Mega323.

Component Sources

Motors: Integrated wheel/motor purchased from Herbach&Rademan (www.herbach.com) part # TM99MTR3278. I also purchased my battery here.

Pump: 12V DC pump (car windshield washer pump) purchased from All Electronics, (www.allelectronics.com) part # PMP-2

Proto-boards: Circuit Specialists (www.web-tronics.com) part # PC462906. I also purchased my soldering iron her, the SL-10 model.

Microcontroller: The MegaAVR-Dev from Progressive Resources, LLC (www.prllc.com)

Motor Drivers: I got samples of the L293D from TI (www.ti.com) and the 3959 from Allegro (www.allegromicro.com). TI also has I^2C port expanders and analog muxes.

Power MOSFET: Free sample from Fairchild (www.fairchildsemi.com), part # HUF76107P3

Sonar: The Devantech SRF08 from Acroname (www.acroname.com). I also got the plastic housing for the sonar from Acroname.

IR sensors: Sharp GP2D12 and Fairchild sensors are both from Mark III robotics (http://www.junun.org/MarkIII/)

Appendix A: Source Code

Main Code

demo.h

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//demo.h
///////////////////////////////////////////////////
#ifndef DEMO_H
#define DEMO_H
#include "global.h"

#define right_eye          0
#define left_eye           1
#define right_table        2
#define left_table         3
#define glass_detector     4
#define left_line_sensor   5
#define center_line_sensor 6
#define right_line_sensor  7

#define behavior_array_length 4

//function prototypes
void delay(u16);

//structure definition
struct behavior
{
    s16 left_motor;
    s16 right_motor;
    s08 confidence;
    s08 acceleration;
};

//variable definitions
extern s16 ad_data[];
extern s16 ad_data_scaled[];


#endif
```

demo.c

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//demo.c
//The main file for running my robot.
///////////////////////////////////////////////////

#include <io.h>
#include <interrupt.h>

#include "global.h"
#include "motor.h"
#include "a_d.h"
```

```
#include "sonar.h"
#include "red_button.h"

#include "demo.h"

#include "obstacle_avoidance.h"
#include "line_following.h"
#include "bump_switches.h"
#include "glass_filling.h"

//#include "uart.h"

s16 ad_data[8];
s16 ad_offset[8];
s16 ad_data_scaled[8];
u16 sonar;

void delay(u16 delay_time) {
    do {
      u08 i=0;
      do {
      asm volatile("nop\n\t"
          "nop\n\t"
          "nop\n\t"
          "nop\n\t"
          ::);
      } while(--i);
    } while(--delay_time);
}


void get_ad_scaled(void)
{
    get_all_ad_avg(ad_data);
    u08 i;
    for(i=0; i<8; i++) {
        ad_data_scaled[i] = ad_data[i]-ad_offset[i];
    }

}

void set_ad_scale(void)
{
    get_all_ad_avg(ad_offset);
}


/*
//doesn't work - I don't know why
u08 get_max_index(struct behavior in[])
{
    u08 i;
    s08 confidence=-100;
    u08 index = 0;
    for(i=0;i<behavior_array_length; i++) {
        if(in[i].confidence > confidence) {
            confidence = in[i].confidence;
            index = i;
        }
    }

    return index;
}
*/


int main(void)
{
    DDRA = 0;
    DDRB = 0;
    DDRC = 0;
    DDRD = 0;

    PORTC = 0x30; //turn off the LEDs
    DDRC  = 0x30;  //Turn on high portC led's

    /*
    char out[32];
    uart_init();
    */
    motor_init();
    ad_init();
    sonar_init();
    red_button_init();


    glass_filling_init();
    obstacle_avoidance_init();
    line_following_init();
    bump_switches_init();

    sei();


    struct behavior main_behavior[behavior_array_length];

    wait_for_button();
    set_ad_scale();

    wait_for_button();

    sonar_start(); //get sonar reading for first time through loop

    //u08 portc_led = 0x01;

    u08 index;


    for (;;)                        // loop forever
    {
        sonar = sonar_closest(); //get the sonar data
        sonar_start();        //start a new sonar ping
        get_ad_scaled();

        line_following(&main_behavior[0]);
        obstacle_avoidance(&main_behavior[1]);
        bump_switches(&main_behavior[2]);
        glass_filling(&main_behavior[3]);

        //index = get_max_index(main_behavior);
        u08 i;
        index =  0;
        s08 confidence = -100;
        for(i=0; i<behavior_array_length; i++) {
           if(main_behavior[i].confidence > confidence) {
               confidence = main_behavior[i].confidence;
               index = i;
           }
        }

        motor_speed(main_behavior[index].left_motor, main_behavior[index].right_motor);
        set_acceleration(main_behavior[index].acceleration);
        //portc_led++;
        //PORTC = ((~portc_led)&0xF0) | (PORTC & 0x0F);
        PORTC = ( (~(index<<4)) & 0x30 ) | (PORTC & 0x0F);
        delay(175); //about 1/18 sec.

        /*
        sprintf(out,"line: left: %i\tr:%i\tc:%i",
              main_behavior[0].left_motor, main_behavior[0].right_motor,
main_behavior[0].confidence);;
        uart_putstr(out);
        sprintf(out,"obst: left: %i\tr:%i\tc:%i",
              main_behavior[1].left_motor, main_behavior[1].right_motor,
main_behavior[1].confidence);;
        uart_putstr(out);
        delay(0x2000);
        */

    }


}
```

Behaviors

bump_switches.h

```
////////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//bump_switches.h
////////////////////////////////////////////////////
#ifndef BUMP_SWITCHES_H
#define BUMP_SWITCHES_H

void bump_switches_init(void);
void bump_switches(struct behavior* out);

#endif
```

bump_switches.c

```
////////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//bump_switches.c
//Behavior code that handles bump switches and
//edge-of-table sensors.
////////////////////////////////////////////////////
#include "demo.h"
#include "global.h"
#include <io.h>
#include "motor.h"
#include "bump_switches.h"

void bump_switches_init()
{
    DDRC  &= ~(_BV(3)|_BV(2));  //set these pins as inputs
    PORTC |=   _BV(2)|_BV(3);    //enable pull-up resistors
}

void bump_switches(struct behavior* out)
{
    s16 left_motor=MOTOR_STOP, right_motor=MOTOR_STOP;
    s08 confidence=-1;
    if(!(PINC & _BV(3))) {
        left_motor = MOTOR_BACKWARD_SLOW;
        confidence =100;
    }
    if(!(PINC & _BV(2))) {
        right_motor = MOTOR_BACKWARD_SLOW;
        confidence = 100;
    }
    if(ad_data[right_table] > 1010 || ad_data[left_table] > 1010) {
        left_motor = MOTOR_STOP;
        right_motor = MOTOR_STOP;
        confidence = 100;
    }


    out->confidence = confidence;
    out->left_motor = left_motor;
    out->right_motor = right_motor;
    out->acceleration = ACCELERATION_FAST;
}
```

glass_filling.h

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//glass_filling.h
///////////////////////////////////////////////////
#ifndef GLASS_FILLING_H
#define GLASS_FILLING_H

void glass_filling_init(void);
void glass_filling(struct behavior* out);

#define PWM1_TOP 255
#define PUMP_START 32 //lowest PWM value that (might) start the motors.
#define PUMP_RUN   128 //value to run the pump at

#define PUMP_OCR      OCR0


#endif
```

glass_filling.c

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//glass_filling.c
//Code to fill a glass, when it is found.  The only
//of the 4 behaviors that does not return (this
//behavior takes over the robot until it completes).
///////////////////////////////////////////////////
#include "demo.h"
#include "global.h"
#include <io.h>
#include "motor.h"
#include "glass_filling.h"


void glass_filling_init(void)
{

    DDRB  |= _BV(3);                        //Set OC0 as output
    PORTB &= ~_BV(3);                       //Turn off OC0

    OCR0  = 0x00;                           //Make sure pin stays off
    TCCR0 = _BV(PWM0)|_BV(COM01)|_BV(CS01);    //enable timer 0 as PWM
}

void glass_filling(struct behavior* out)
{
    s16 left_motor = MOTOR_STOP, right_motor = MOTOR_STOP;
    s08 confidence = -1;
    //static s08 count = 0;
    if(ad_data[glass_detector] < 900) {
        //confidence = 60;
        set_acceleration(ACCELERATION_FAST);
        motor_speed(MOTOR_STOP, MOTOR_STOP);

        //count++;
        //if(count >1 ) {
            //fill the glass.,.
        if(PIND & _BV(7)) {
            PUMP_OCR = PUMP_RUN; //turns on pump
            while(PIND & _BV(7)) {;} //wait for glass to fill
            PUMP_OCR = 0; //turn off pump
        }

            //turn left to go around the glass
        //left_motor = MOTOR_BACKWARD_SLOW;
        //right_motor = MOTOR_FORWARD_SLOW;
        motor_speed(MOTOR_BACKWARD_SLOW, MOTOR_STOP);
        delay(0x1000);
        motor_speed(MOTOR_BACKWARD_SLOW, MOTOR_FORWARD_SLOW);
        //motor_speed(left_motor, right_motor);
        delay(0x2000);
        motor_speed(MOTOR_FORWARD_SLOW, MOTOR_FORWARD_SLOW);
        delay(0x1000);
        //}
        //do the glass filling here.........
        //remember the motors don't turn off until AFTER
        //the routine has run once

    } else {
        //no glass found;
    }

    out->confidence = confidence;
    out->left_motor = left_motor;
    out->right_motor = right_motor;
    out->acceleration = ACCELERATION_FAST;

}
```

line_following.h

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//line_following.h
///////////////////////////////////////////////////
#ifndef LINE_FOLLOWING_H
#define LINE_FOLLOWING_H

void line_following_init(void);
void line_following(struct behavior* out);

#endif
```

line_following.c

```
///////////////////////////////////////////////////
```

```
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//line_following.c
//A line-following algorithm, based on 3 analog
//sensors at the front of my robot.
///////////////////////////////////////////////////
#include "demo.h"
#include "global.h"
#include <io.h>
#include "motor.h"
#include "line_following.h"

s16 line_difference(void)
{
    return (ad_data[left_line_sensor] - ad_data[right_line_sensor]);

}

void line_following_init()
{
    ;
}

void line_following(struct behavior* out)
{
    static s16 integral_error=0;
    static u08 have_line = 0;
    s16 left_tmp, right_tmp;
    s16 line_delta;
    s08 confidence;

    line_delta = line_difference();

    //diff = abs_val(left_tmp - right_tmp); //current wheel speed difference
    if(have_line) {
        integral_error += line_delta/32;
        left_tmp = MOTOR_FORWARD_SLOW - integral_error - line_delta;
        right_tmp = MOTOR_FORWARD_SLOW + integral_error + line_delta;
    } else {
        left_tmp = MOTOR_FORWARD_SLOW - line_delta;
        right_tmp = MOTOR_FORWARD_SLOW + line_delta;
    }

    if(left_tmp > MOTOR_TOP) {
        left_tmp = MOTOR_TOP;
    }else if(left_tmp < -MOTOR_TOP) {
        left_tmp = -MOTOR_TOP;
    }
    if(right_tmp>MOTOR_TOP) {
        right_tmp = MOTOR_TOP;
    } else if(right_tmp < -MOTOR_TOP) {
        right_tmp = -MOTOR_TOP;
    }

    if( (ad_data[center_line_sensor]-ad_data[left_line_sensor] > 200) &&
          (ad_data[center_line_sensor]-ad_data[right_line_sensor] > 200)) {
        confidence = 25; //we *know* we're on the line
        have_line=1;
    } else if(ad_data_scaled[center_line_sensor] > (200 - 150*have_line)) {
        confidence = 10; //maybe it's a line - center sensor is dark
        have_line = 1;
        //confidence += have_line * 10; //more important to stay on line
        //integral_error = integral_error*have_line;
    } else {
        confidence = 0;  //no line :-(
        integral_error = 0;
        have_line = 0;
    }

    out->left_motor = left_tmp;
    out->right_motor = right_tmp;
    out->confidence = confidence;
    out->acceleration = ACCELERATION_FAST;

}
```

obstacle_avoidance.h

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//obstacle_avoidance.h
///////////////////////////////////////////////////
#ifndef OBSTACLE_AVOIDANCE_H
#define OBSTACLE_AVOIDANCE_H

void obstacle_avoidance_init(void);
void obstacle_avoidance(struct behavior* out);
extern u16 sonar;

#endif
```

obstacle_avoidance.c

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//obstacle_avoidance.c
//Basic obstacle avoidance code.  Assumes the presence
//of variables set in demo.c
///////////////////////////////////////////////////
#include "demo.h"
#include "global.h"
#include <io.h>
#include "motor.h"
#include "obstacle_avoidance.h"

#define LEFT 1
#define RIGHT 0

s16 max(s16 a, s16 b)
{
    return a>b?a:b;
}

void obstacle_avoidance_init()
{
   ;
}

u08 random(void)
{
    u08 rand = TCNT1L ^ TCNT0; //the two counters I'm using
    rand *= 71; //multiply by a prime to mix the bits around
    return rand;
}
/*
void turn(u08 direction, u08 time)
{
    if((direction & 0x1) == right) {
        //turn right
    } else {
        //turn left
    }
}
*/
void obstacle_avoidance(struct behavior* out)
{
    s16 right_motor, left_motor;
    s08 confidence;
    s16 right = ad_data[right_eye];
    s16 left = ad_data[left_eye];
    static s08 count=0;
    static u08 direction;

    if(count == 0) {
        if(right<=150 && left<=150 && sonar>=17) {
            //don't see anything
            if(sonar > 24) {
                right_motor = MOTOR_FORWARD_FAST;
                left_motor = MOTOR_FORWARD_FAST;
            } else {
                right_motor = MOTOR_FORWARD_SLOW;
                left_motor = MOTOR_FORWARD_SLOW;
            }
            confidence = 1;
        } else {
            //there's something out there...
            if(right > (left + left/4) ) {
                //see something in right eye (to left of bot)
                right_motor = MOTOR_BACKWARD_SLOW;
                left_motor = MOTOR_FORWARD_SLOW;
                direction = RIGHT;
            } else if(left > (right + right/4)) {
                // see something to the right of the bot
                right_motor = MOTOR_FORWARD_SLOW;
                left_motor = MOTOR_BACKWARD_SLOW;
                direction = LEFT;
            } else {
                //pretty much in front of us
                //turn in a random direction
                if(random() & 0x01) {
                    right_motor = MOTOR_BACKWARD_SLOW;
                    left_motor = MOTOR_FORWARD_SLOW;
                    direction = RIGHT;
                } else {
                    right_motor = MOTOR_FORWARD_SLOW;
                    left_motor = MOTOR_BACKWARD_SLOW;
                    direction = LEFT;
                }
            }
            confidence = 11;
            count = ((random()&0x03) + 1)*16;
        }

    } else {
        //we're in the middle of turning
        count--;
        if(direction == RIGHT) {
            right_motor = MOTOR_BACKWARD_SLOW;
            left_motor = MOTOR_FORWARD_SLOW;
        } else {
            right_motor = MOTOR_FORWARD_SLOW;
            left_motor = MOTOR_BACKWARD_SLOW;
        }
        confidence = 11;
    }

    //nothing
    // eyes <= 150
    // sonar >= 17

    //kinda see something
    // "eyes" > 150
    // sonar < 17

    //really need to get out of here
    // "eyes" > 250
    // sonar < 10

/*
    if(ad_data[right_eye] < 100) {
        right_motor = MOTOR_FORWARD_FAST;
        confidence = 1;
    } else if(ad_data[right_eye] < 150) {
        confidence = 5;
        right_motor = MOTOR_FORWARD_SLOW;
    } else if(ad_data[right_eye] < 300) {
        right_motor = MOTOR_STOP;
        confidence = 11;
    } else {
        right_motor = MOTOR_BACKWARD_SLOW;
        confidence = 50;
    }

    if(ad_data[left_eye] < 100) {
        left_motor = MOTOR_FORWARD_FAST;
    } else if(ad_data[left_eye] < 150) {
        left_motor = MOTOR_FORWARD_SLOW;
        confidence = max(confidence, 5);
    } else if(ad_data[left_eye] < 300) {
        left_motor = MOTOR_STOP;
        confidence = max(confidence, 11);
    } else {
        left_motor = MOTOR_BACKWARD_SLOW;
        confidence = max(confidence, 50);
    }
*/
    /*
    if(left_motor == MOTOR_STOP && right_motor == MOTOR_STOP) {
        left_motor = MOTOR_FORWARD_SLOW;
        right_motor = MOTOR_BACKWARD_SLOW;
    }
    */
    out->right_motor = right_motor;
    out->left_motor = left_motor;
    out->confidence = confidence;
    out->acceleration = ACCELERATION_SLOW;

}
```

Support Code

red_button.h

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//red_button.h
///////////////////////////////////////////////////
#ifndef RED_BUTTON_H
#define RED_BUTTON_H

void red_button_init(void);
void wait_for_button(void);
```

```
#endif
```

red_button.c

```
//////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//red_button.c
//Implements a pause routine, and wait-until-button-
//pressed, using a switch on PD2 (INT0).
//////////////////////////////////////////////////
#include <interrupt.h>
#include <sig-avr.h>
#include <io.h>
#include <inttypes.h>
#include "global.h"
#include "red_button.h"

//pause/unpause
//wait_for_button

volatile s08 waiting;

//delay for about 200ms
void short_delay(void)
{
    int32_t tmp = 400000;
    do{;}while(--tmp);
}

//Pause function: press once to pause,
//press again to unpause
SIGNAL(SIG_INTERRUPT0)
{
    PORTB &= ~(_BV(1)); // turn on D9 diode
    if(waiting) {
        //we are waiting for a button press
        //make sure button is low
        short_delay();
        //wait for release of button
        loop_until_bit_is_set(PIND, 2);
        //wait for pin to stabilize
        short_delay();
        waiting = 0;
    } else {
        //act as pause function
        s16 ocr1a_value = OCR1A;
        s16 ocr1b_value = OCR1B;
        OCR1A = 0;
        OCR1B = 0;
        s08 ocr0_value = OCR0;
        OCR0 = 0;

        //wait for button release
        short_delay();
        loop_until_bit_is_set(PIND, 2);
        short_delay();

        //wait for button press
        loop_until_bit_is_clear(PIND,2);
        short_delay();
        //wait for button release
        loop_until_bit_is_set(PIND, 2);
        short_delay();

        //restore everything
        OCR1A = ocr1a_value;
        OCR1B = ocr1b_value;
        OCR0  = ocr0_value;
    }
    PORTB |= _BV(1);  //turn off D9 diode
}


void red_button_init(void)
{
    waiting = 0;

    SFIOR &= ~(_BV(PUD)); // enable pull-up resistors
    DDRD &= ~(_BV(2)); // pin2 is input
    PORTD |= _BV(2);   //pull-up resistor is on

    PORTB |= _BV(1);   //use D9 diode as pause status light
    DDRB |= _BV(1);


    MCUCR &= ~(_BV(ISC11)|_BV(ISC10));  //INT0 is low-level triggered
    GIFR |= _BV(INTF0); // clear the flag
    //PORTB &= ~(_BV(1));
    short_delay();  //make sure everything has stabilized
    //PORTB |= _BV(1);
    GICR |= _BV(INT0); //enable the interrupt
}

void wait_for_button(void)
{
    waiting = 1;
    do{;}while(waiting);
}
```

global.h

```
///////////////////////////////////////////////////
//based on code provided in the avr-gcc 3.2 package
//frow www.avrfreaks.net.  Code originally written
//by Volker Oth.
//
//global.h
//global defines
///////////////////////////////////////////////////
#ifndef GLOBAL_H
#define GLOBAL_H

#define MAX_U16  65535
#define MAX_S16  32767

#define IN       0
#define OUT      1

#define LOW      0
#define HIGH     1

#define FALLING  0
#define RISING   1

typedef unsigned char  u08;
typedef          char  s08;
typedef unsigned short u16;
typedef          short s16;
typedef u08            bool;

#define DDR(x) ((x)-1)    /* address of data direction register of port x */
#define PIN(x) ((x)-2)    /* address of input register of port x */

#endif
```

a_d.h

```
//////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//a_d.h
//////////////////////////////////////////////////
#ifndef A_D_H
#define A_D_H

#define ad_average_length 4
/* prototypes */
extern void ad_init(void);
extern s16 get_sample(u08 channel);
extern void get_all_ad(s16* ad_array);
extern void get_all_ad_avg(s16* ad_array);

#endif
```

a_d.c

```
//////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//a_d.c
//a/d routines.  Includes 4-sample averaging.
//////////////////////////////////////////////////

#include <io.h>
#include "global.h"
#include "a_d.h"

void ad_init(void)
{
    DDRA  = 0x00;  //all A ports are inputs
    ADMUX = 0x00;  //select AD0, external Vref, not left adjusted
    ADCSR = _BV(ADEN)|_BV(ADIF)|_BV(ADPS2)|_BV(ADPS0);  //enable, clear int. flag, set ad
clock to CK/32
}


s16 get_sample(u08 channel)
{
    u08 temp = ADMUX & 0xE0;  //keep top 3 bits (REFS1:0, ADLAR)
    channel &= 0x07;  //throw away top bits
    ADMUX = channel | temp;  // put two parts back together
    while(ADCSR & _BV(ADSC)){;} //wait for bit to clear - should never be set
    ADCSR |= _BV(ADSC);  //start conversion
    while(ADCSR & _BV(ADSC)){;} //wait for conversion to complete
    s16 retval = ADCW;
    return retval;
}

void get_all_ad(s16* ad_array)
{
    u08 i;

    for(i=0;i<8;i++)
    {
        ad_array[i] = get_sample(i);
    }

}
```

```
void get_all_ad_avg(s16* ad_array)
{
    u08 i,j;
    get_all_ad(ad_array);
    for(i=1;i<ad_average_length;i++) {
        for(j=0;j<8;j++) {
            ad_array[j] += get_sample(j);
        }
    }
    for(i=0;i<8;i++) {
        ad_array[i] /= ad_average_length;
    }
}
/*
void delay(u16 delay_time) {
    do {
      u08 i=0;
      do {
      asm volatile("nop\n\t"
          "nop\n\t"
          "nop\n\t"
          "nop\n\t"
          ::);
      } while(--i);
    } while(--delay_time);
}

int main(void)
{
    DDRA = 0;    //all ports are inputs
    DDRB = 0;
    DDRC = 0;
    DDRD = 0;

    uart_init();
    ad_init();

    //s16 analog7;
    char out[32];
    s16 ad_values[8];
    sei();
    //uart_putstr("AD conversion started...\r\n");


    for(;;)
    {
        get_all_ad(ad_values);

        u08 i;
        for(i=0; i<8; i++)
        {
            //ad_values[i] = get_sample(i);
            sprintf(out, "AD%i:\t%i\r\n", i, ad_values[i]);
            uart_putstr(out);
        }
        //analog7 = get_sample(7);
        //sprintf(out, "AD7:\t%i\r\n", analog7);
        //uart_putstr(out);
        delay(0x1FFF);
        uart_nl();
    }
}
*/
```

motor.h

```
//////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
```

```
//notice is preserved.  Any other use requires my
//permission.
//
//motor.h
///////////////////////////////////////////////////
#ifndef MOTOR_H
#define MOTOR_H

#define MOTOR_TOP   511


//#define MOTOR_MAX             0x3FFF
#define MOTOR_FORWARD_FAST    511
#define MOTOR_FORWARD_SLOW    256
#define MOTOR_STOP              0
#define MOTOR_BACKWARD_SLOW  -256
#define MOTOR_BACKWARD_FAST  -511

#define ACCELERATION_SLOW     6
#define ACCELERATION_FAST     1


/* prototypes */
extern void motor_init(void);
extern void motor_speed(s16 left_motor, s16 right_motor);
extern void set_acceleration(s08 _acceleration);
#endif
```

motor.c

```
///////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//motor.c
//Motor driver routines.  Uses the Timer1 to do
//PWM.  External TI L293D chip used.  Acceleration
//limiting is implemented.
///////////////////////////////////////////////////
#define RIGHT_BACKWARD _BV(4)
#define RIGHT_FORWARD  _BV(5)
#define RIGHT_OCR      OCR1B

#define LEFT_BACKWARD _BV(6)
#define LEFT_FORWARD  _BV(7)
#define LEFT_OCR      OCR1A


#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

#include "global.h"
#include "motor.h"

#define PWM1_TOP 1023
#define MOTOR_START 512 //lowest PWM value that (might) start the motors.
//#define LEFT_MOTOR_OFFSET -10
//#define RIGHT_MOTOR_OFFSET -50


// left motor=OC1A
// right motor = OC1B

//range from -511 to 511
//set these to change motor speed
volatile s16 left_motor_speed;
volatile s16 right_motor_speed;
volatile s08 acceleration;

SIGNAL(SIG_OVERFLOW1)
{
    static s08 count = 6;
    static s16 left_motor = 0;
    static s16 right_motor = 0;
    if(--count == 0) {
        if(left_motor != left_motor_speed) {
            if(left_motor < left_motor_speed) {
                //speed up the motor
                left_motor++;
            } else {
                //must need to slow down the motor
                left_motor--;
            }
            s16 left_tmp;
            if(left_motor <0) {
                left_tmp = (-left_motor) + MOTOR_START;
            } else if(left_motor > 0) {
                left_tmp = left_motor + MOTOR_START;
            }
            LEFT_OCR = left_tmp;
        }
        if(right_motor != right_motor_speed) {
            if(right_motor < right_motor_speed) {
                //speed up the motor
                right_motor++;
            } else if(right_motor > right_motor_speed) {
                //slow down the motor
                right_motor--;
            }
            s16 right_tmp;
            if(right_motor<0) {
                right_tmp = (-right_motor) + MOTOR_START;
            } else if(right_motor>0) {
                right_tmp = right_motor + MOTOR_START;
            }
            RIGHT_OCR = right_tmp;
        }

        u08 control_vals = 0;
        if(right_motor < 0) {
            control_vals |= RIGHT_BACKWARD;
        } else if(right_motor > 0) {
            control_vals |= RIGHT_FORWARD;
        }

        if(left_motor < 0) {
            control_vals |= LEFT_BACKWARD;
        } else if(left_motor > 0) {
            control_vals |= LEFT_FORWARD;
        }
        u08 tmp = PORTB & 0x0F; //throw away top bits - these are the ones used for motor
direction
        PORTB = control_vals | tmp;
        count = acceleration;             //start the counter over
    }
}

void motor_init(void)
{    PORTC = 0x30; //turn off the LEDs    PORTC = 0x30; //turn off the LEDs
    DDRD  |= _BV(5);                         //Set OC1A as output
    PORTD &= ~_BV(5);                        //Turn off OC1A
    DDRD  |= _BV(4);                         //OC1B is output
    PORTD &= ~_BV(4);

    OCR1A  = 0x0000;                         //Make sure pin stays off
    OCR1B  = 0x0000;

    TCCR1A = _BV(PWM11)|_BV(PWM10)|_BV(COM1A1)|_BV(COM1B1); //10-bit PWM, non-inverted
    TCCR1B = _BV(CS10);                             //Divide clock by 1

    DDRB   |= 0xF0;                      //use high bits of portB to control direction
    PORTB  &= ~0xF0;                     //Turn off the outputs.

    left_motor_speed = 0;
    right_motor_speed = 0;
    acceleration = ACCELERATION_SLOW;

    TIFR |= _BV(TOV1); //clear TCNT1 overflow flag
    TIMSK |= _BV(TOIE1);  //enable TCNT1 overflow interrupt
}


void set_acceleration(s08 _acceleration)
{
    acceleration = _acceleration;
}

//speeds are neg. for reverse
//range from -511 to 511.  0 is stopped.
void motor_speed(s16 left_motor, s16 right_motor)
{
    if(left_motor > MOTOR_TOP) {
        left_motor = MOTOR_TOP;
    } else if(left_motor < -MOTOR_TOP) {
        left_motor = -MOTOR_TOP;
    }

    if(right_motor > MOTOR_TOP) {
        right_motor = MOTOR_TOP;
    } else if(right_motor < -MOTOR_TOP) {
        right_motor = -MOTOR_TOP;
    }

    left_motor_speed = left_motor;
    right_motor_speed = right_motor;
/*
    u08 temp = PORTB;
    temp &= 0x0F;  // throw away the top bits -
    u08 control_vals=0;


    if(left_motor==0) {
        LEFT_OCR = 0;
    }
    else {
        if(left_motor <0) {
            control_vals |= LEFT_BACKWARD;
            left_motor = -left_motor;
        }
        else {
            control_vals |= LEFT_FORWARD;
        }
        left_motor += MOTOR_START;
        //left_motor += LEFT_MOTOR_OFFSET; //correct for different motor speeds
        if(left_motor > PWM1_TOP) {
             left_motor = PWM1_TOP;
        }
        LEFT_OCR = left_motor;
    }

    if(right_motor==0) {
        RIGHT_OCR = 0;
    }
    else {
        if(right_motor<0) {
            control_vals |= RIGHT_BACKWARD;
            right_motor = -right_motor;
        }
        else {
            control_vals |= RIGHT_FORWARD;
        }
        right_motor += MOTOR_START;
        //right_motor += RIGHT_MOTOR_OFFSET;
        if(right_motor > PWM1_TOP) {
            right_motor = PWM1_TOP;
        }
        RIGHT_OCR = right_motor;
    }

    PORTB = control_vals | temp;
*/
}
```

sonar.h

```
//sonar.h
#ifndef SONAR_H
#define SONAR_H

/* prototypes */
void sonar_init(void);
s08 sonar_start(void);
s08 sonar_data(u16* ranges, u08* light_sensor);
u16 sonar_get_closest(u16* ranges);
u16 sonar_closest(void);

#endif
```

sonar.c

```
///////////////////////////////////////////////////
//Based on code provided by Steven Theriault, and
//the Atmel TWI sample code.
//
//sonar.c
///////////////////////////////////////////////////
#include <io.h>
#include "global.h"
#include <twi.h>

#define TWI_RATE_SELECT 21   //100kHz clock @ 6MHz fck

#define SONAR_ADDRESS 0xE0
#define READ 1
#define WRITE 0

#define SONAR_COMMAND_REGISTER 0
#define SONAR_COMMAND 80 //result in inches
#define SONAR_MAX_ADDRESS 35 //last byte available (36 bytes 0:35)



void sonar_init(void)
{
    PORTC &= ~(_BV(1)|_BV(0));  //SDA, SCL are outputs,
    DDRC |= _BV(1)|_BV(0);

    TWBR = TWI_RATE_SELECT;          //set clock
    TWCR = _BV(TWEA) | _BV(TWEN) | _BV(TWINT); //acknoledge, enable, clear IRQ
    TWAR = 0x01;        // Don't care, won't be in Slave mode
}

//start the sonar "ping"
//note that it takes at least 65ms for the data to
//be available (so it's best to do something else
//while the sonar is processing)
s08 sonar_start(void)
{
    loop_until_bit_is_clear(TWCR, TWSTO);  //check no transmission in progress
    //based on demo code on p. 112 in mega323 manual
    TWCR = _BV(TWSTA) | _BV(TWEN); //send start condition
    while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
    if(TWSR != TW_START) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -1;  //error occured - should have been start
    }

    TWDR = SONAR_ADDRESS | WRITE;  //send sonar address
    TWCR = _BV(TWINT) | _BV(TWEN); //clear interrupt to start transmission
    while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
    if(TWSR != TW_MT_SLA_ACK) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -2;
    }

    TWDR = SONAR_COMMAND_REGISTER;  //send the address on the sonar to write to
    TWCR = _BV(TWINT) | _BV(TWEN); //clear IRQ, continue transmission
    while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
    if(TWSR != TW_MT_DATA_ACK) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -3;
    }

    TWDR = SONAR_COMMAND;          //send the sonar "start ranging" command
    TWCR = _BV(TWINT) | _BV(TWEN);
    while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
    if(TWSR != TW_MT_DATA_ACK){
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -4;
    }

    TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); //send stop condition

    return 0;

}

//data should be 36 long
s08 sonar_get_data(u08* data)
{
    //char out[32];
    //u08* ranges = (u08*) _ranges;
    loop_until_bit_is_clear(TWCR, TWSTO); //wait for last transmission to end

    TWCR = _BV(TWINT)|_BV(TWSTA)|_BV(TWEN); //send start condition
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_START) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        //sprintf(out, "1: %x", TWSR);
        //uart_putstr(out);
        return -1;
        //return TWSR;
    }

    TWDR = SONAR_ADDRESS | WRITE;         //send sonar address
    TWCR = _BV(TWINT)|_BV(TWEN);
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_MT_SLA_ACK) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        //sprintf(out, "2: %x", TWSR);
        //uart_putstr(out);
        return 1;            //expected error when sonar is busy
    }

    TWDR = 0;    //send address to start reading at (register 0)
    TWCR = _BV(TWINT)|_BV(TWEN);
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_MT_DATA_ACK) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        //sprintf(out, "3: %x", TWSR);
        //uart_putstr(out);
        return 1;                  //expected error when sonar is busy
    }

    TWCR = _BV(TWINT)|_BV(TWSTA)|_BV(TWEN);  //send repeated start
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_REP_START) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        //sprintf(out, "4: %x", TWSR);
        //uart_putstr(out);
        return -4;
    }

    TWDR = SONAR_ADDRESS | READ;            //send sonar address for read
    TWCR = _BV(TWINT)|_BV(TWEA)|_BV(TWEN);
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_MR_SLA_ACK) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -5;
    }

    u08 i;
    for(i=0; i<SONAR_MAX_ADDRESS; i++) {
        TWCR = _BV(TWINT)|_BV(TWEA)|_BV(TWEN);
        loop_until_bit_is_set(TWCR, TWINT);
        if(TWSR != TW_MR_DATA_ACK) {
            TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
            return -6;
        }
        *data = TWDR;
        data++;
    }

    TWCR = _BV(TWINT)|_BV(TWEN);   //TWEA not set - to indicate this is last byte
    loop_until_bit_is_set(TWCR, TWINT);
    if(TWSR != TW_MR_DATA_NACK) {   //get NACK since we didn't set TWEA
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -7;
    }
    *data = TWDR;  //get last byte

    TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);  //send stop condition

    return 0;
}


//data is input from sonar_get_data, ranges and light_sensor are output
//data[36], ranges[17], light_sensor[1];
void convert_data(u16* ranges, u08* light_sensor, u08* data)
{
    data++;
    *light_sensor = *data; //1st element is light sensor
    data++;         //first range data byte (high byte)
    u08 i;
    //step through the 17 range locations
    for(i=0; i<17; i++) {
        *ranges = (*data)<<8 | *(data+1);
        data+=2;
        ranges++;
    }
}

//returns the light sensor value,
//gets the ranges in the provided array
//which must be 17 elements long
s08 sonar_data(u16* ranges, u08* light_sensor)
{
    u08 data[36];
    s08 error;
    do {
        error = sonar_get_data(data);
    } while(error==1);
    if(error == 0) {
        convert_data(ranges, light_sensor, data);
        return 0;
    } else {
        return -1;
    }
}

u16 sonar_get_closest(u16* ranges)
{
    //things closer than 6inches are probably reflections
    //off the front of the robot
    while(*ranges <= 6 && *ranges!=0) {
        ranges++;
    }
    return *ranges;
}

//returns the closest item to the sonar
//this is the easiest way to use the sonar -
//just call sonar_start(); then call sonar_closest();
//to get the closest reading.
//returns 0 if an error occurred.
u16 sonar_closest(void)
{
    u16 ranges[17];
    u08 temp;
    s08 error;
    error = sonar_data(ranges, &temp);
    if(error == 0) {
        return sonar_get_closest(ranges);
    } else {
        return 0;
    }

}


/*
int main(void)
{
    DDRA = 0;    //all ports are inputs
    DDRB = 0;
    DDRC = 0;
    DDRD = 0;

    //u08 data[36]; //sonar data
    u16 ranges[17];
    u08 light_sensor=0;
    char out[32];

    delay(0x2000); //give bootloader a second to reset

    uart_init();
    sonar_init();
    sei();

    s08 error1, error2;
    u08 i;

    for(;;) {
        error1 = sonar_start();
        error2 = sonar_data(ranges, &light_sensor);

        if(error1==0 && error2==0) {
            for(i=0; i< 17; i++) {
                if(i%8 == 0) {
                    uart_nl();
                }
                sprintf(out, " %i:%i ",i,ranges[i]);
                uart_putstr(out);
            }
            uart_nl();
        }
        sprintf(out, "light: %i\r\nerror1: %i error2: %i", light_sensor, error1, error2);
        uart_putstr(out);
        uart_nl();
        delay(0x2000);
    }

}
*/
```

uart.h

```
//uart.h

#ifndef UART_H
#define UART_H

/* set baud rate here */
#define UART_BAUD_RATE 19200
#define ESC            0x1b
#define UART_BUF_SIZE    16
#define SCRATCH 16
#define F_CPU        6000000              /* 6MHz processor */


/* automatically calcuate baud register value */
#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*16l)-1)


/* prototypes */
extern void uart_init(void);
extern void uart_clr(void);
extern void uart_nl(void);
extern bool uart_putchar(u08 c);
extern s16  uart_getchar(void);
extern bool uart_putstr(u08 s[]);
extern bool uart_putstr_fast(u08 s[]);
extern int sprintf(u08 *, const u08 *, ...);

#endif
```

uart.c

```
//UART library functions.
//based on sample code by Volker Oth in
//the gcctest9 package of avr-gcc 3.2
//from www.avrfreaks.net
/*
    Title:    UART library
    Author:   Volker Oth
    Date:     5/1999
    Purpose:  Sending a block of data to uart.
    needed
    Software: AVR-GCC to compile, AVA to assemble and link
    needed
    Hardware: ATS90S8515 on STK200 board
    Note:     To contact me, mail to
                  volkeroth@gmx.de
              You might find more AVR related stuff at my homepage:
                  http://members.xoom.com/volkeroth
*/

#include <stdarg.h>
#include <string-avr.h>
#include <io.h>
```

```
#include <sig-avr.h>
#include <interrupt.h>
#include "global.h"
//#include "timer.h"
#include "uart.h"


/* uart globals */
volatile u08 uart_txd_buf_cnt;
volatile u08 uart_rxd_buf_cnt;
u08 *uart_txd_in_ptr, *uart_txd_out_ptr;
u08 *uart_rxd_in_ptr, *uart_rxd_out_ptr;
u08 UART_CLR[] = {ESC, '[','H', ESC, '[', '2', 'J',0};
u08 UART_NL[] = {0x0d,0x0a,0};
u08 uart_txd_buffer[UART_BUF_SIZE];
u08 uart_rxd_buffer[UART_BUF_SIZE];

void uart_init(void)
// initialize uart
{
    UBRRH = 0x00;
    UBRRL = UART_BAUD_SELECT;
    UCSRB = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);
    UCSRC = _BV(URSEL)|_BV(UCSZ1)|_BV(UCSZ0);
    UCSRA = 0x00;

    uart_txd_in_ptr  = uart_txd_out_ptr = uart_txd_buffer;
    uart_rxd_in_ptr  = uart_rxd_out_ptr = uart_rxd_buffer;
    uart_txd_buf_cnt = 0;
    uart_rxd_buf_cnt = 0;
}


SIGNAL(SIG_UART_DATA)
/* signal handler for uart data buffer empty interrupt */
{
    if (uart_txd_buf_cnt > 0) {
        outp(*uart_txd_out_ptr, UDR);           /* write byte to data buffer */
        if (++uart_txd_out_ptr >= uart_txd_buffer + UART_BUF_SIZE) /* Pointer wrapping */
            uart_txd_out_ptr = uart_txd_buffer;
        if(--uart_txd_buf_cnt == 0)             /* if buffer is empty: */
            cbi(UCSRB, UDRIE);                    /* disable UDRIE int */
    }
}


SIGNAL(SIG_UART_RECV)
/* signal handler for receive complete interrupt */
{
    *uart_rxd_in_ptr = inp(UDR);            /* read byte from receive register */
    uart_rxd_buf_cnt++;
    if (++uart_rxd_in_ptr >= uart_rxd_buffer + UART_BUF_SIZE) /* Pointer wrapping */
        uart_rxd_in_ptr = uart_rxd_buffer;
}


s16 uart_getchar(void)
{
    u08 c;

    if (uart_rxd_buf_cnt>0) {
        cli();
        uart_rxd_buf_cnt--;
        c = *uart_rxd_out_ptr;              /* get character from buffer */
        if (++uart_rxd_out_ptr >= uart_rxd_buffer + UART_BUF_SIZE) /* pointer wrapping */
            uart_rxd_out_ptr = uart_rxd_buffer;
        sei();
        return c;
    }
    else
        return -1;                          /* buffer is empty */
```

```
}


bool uart_putchar(u08 c)
{
    if (uart_txd_buf_cnt<UART_BUF_SIZE) {
        cli();
        uart_txd_buf_cnt++;
        *uart_txd_in_ptr = c;               /* put character into buffer */
        if (++uart_txd_in_ptr >= uart_txd_buffer + UART_BUF_SIZE) /* pointer wrapping */
            uart_txd_in_ptr = uart_txd_buffer;
        sbi(UCSRB, UDRIE);                    /* enable UDRIE int */
        sei();
        return 1;
    }
    else
        return 0;                           /* buffer is full */
}


bool uart_putstr(u08 s[])
{
    char *c = s;

    while (*c)
        if (uart_putchar(*c))
            c++;

    return 1;
}

bool uart_putstr_fast(u08 s[])
/*puts as much of the string as possible into the buffer*/
{
    char *c = s;
    bool success=1;

    while(*c && success)
    {
        success=uart_putchar(*c);
        c++;
    }

    return success;
}


void uart_clr(void)
/* Send a 'clear screen' to a VT100 terminal */
{
    uart_putstr(UART_CLR);
}


void uart_nl(void)
/* Send a 'new line' */
{
    uart_putstr(UART_NL);
}



int sprintf(u08 *buf, const u08 *format, ...)
/* simplified sprintf */
{
  u08 scratch[SCRATCH];
  u08 format_flag;
  u16 u_val=0, base;
  u08 *ptr;
  va_list ap;
  va_start (ap, format);
  for (;;){
    while ((format_flag = *format++) != '%'){      /* Until '%' or '\0' */
      if (!format_flag){va_end (ap); return (0);}
      *buf = format_flag; buf++; *buf=0;
    }

    switch (format_flag = *format++){

    case 'c':
      format_flag = va_arg(ap,int);
    default:
      *buf = format_flag; buf++; *buf=0;
      continue;
    case 'S':
    case 's':
      ptr = va_arg(ap,char *);
      strcat(buf, ptr);
      continue;
    case 'o':
      base = 8;
      *buf = '0'; buf++; *buf=0;
      goto CONVERSION_LOOP;
    case 'i':
      if (((int)u_val) < 0){
        u_val = - u_val;
        *buf = '-'; buf++; *buf=0;
      }
      /* no break -> run into next case */
    case 'u':
      base = 10;
      goto CONVERSION_LOOP;
    case 'x':
      base = 16;

    CONVERSION_LOOP:
      u_val = va_arg(ap,int);
      ptr = scratch + SCRATCH;
      *--ptr = 0;
      do {
        char ch = u_val % base + '0';
        if (ch > '9')
          ch += 'a' - '9' - 1;
        *--ptr = ch;
        u_val /= base;
      } while (u_val);
      strcat(buf, ptr);
      buf += strlen(ptr);
    }
  }
}
```