

+1
Alex Weinert

EEL 5666
2008 Jan 31
Dr. Schwartz
Dr. Arroyo
Adam Barnett
Mike Pridgen
Sara Keen

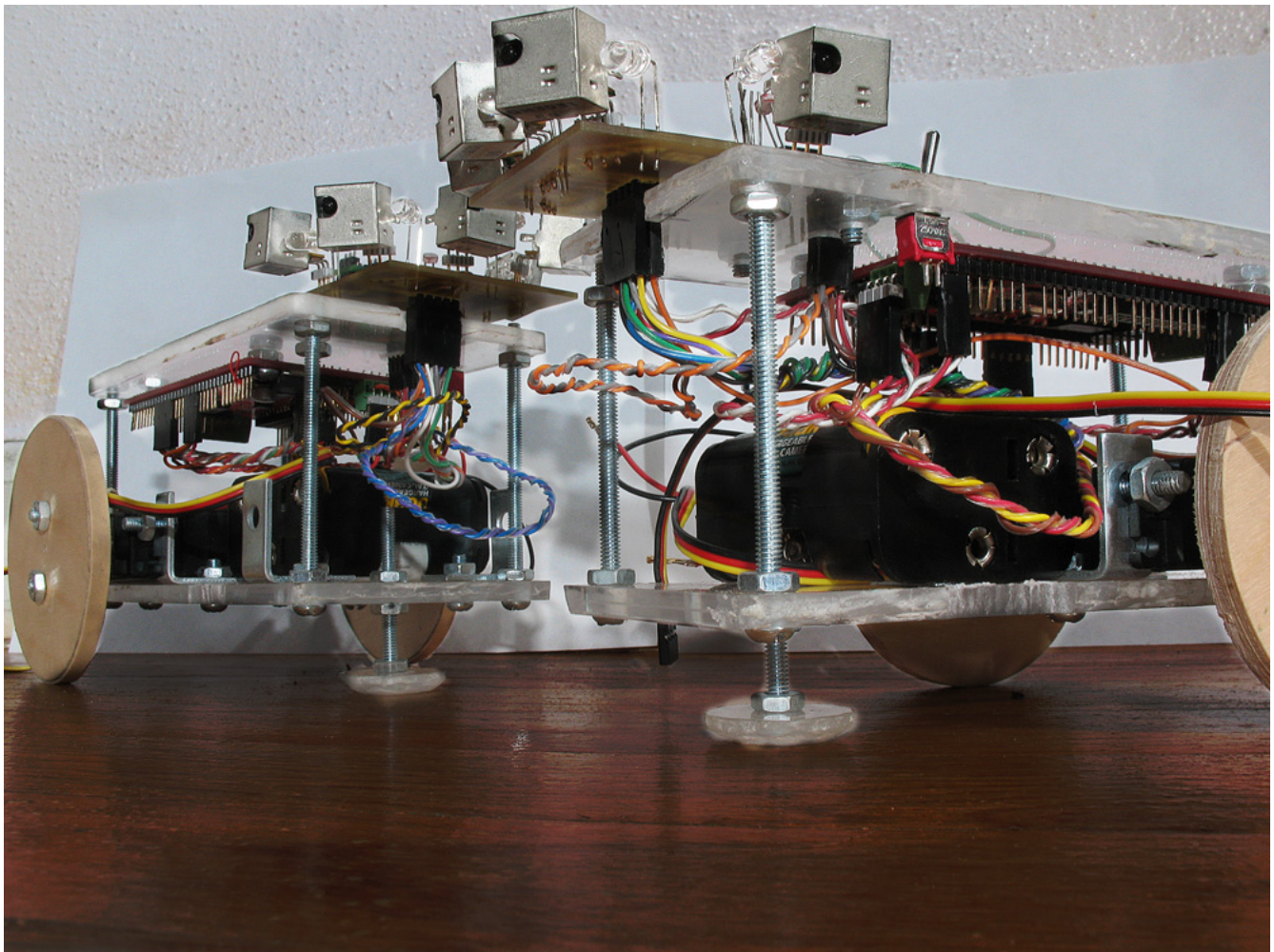
Table of Contents

Abstract.....	3
Integrated System.....	4
Overview.....	4
Sensor Board Design	5
Software Design.....	5
Mobile Platform.....	6
Actuation.....	7
Sensors.....	8
IR Rangefinder.....	8
Compass Module.....	8
CdS Cell.....	8
Wireless Serial.....	8
Behaviors.....	9
Obstacle Avoidance.....	9
Going some arbitrarycompass direction.....	9
Conclusion.....	10
Appendix A, The Code.....	11
Appendix B, Sensor Board Schematics.....	28

Abstract

+1 is intended to be an exercise in robot interaction. By knowing each other's orientation, and keeping withing a certain distance they are to travel until they reach some goal, say light to stave off the inevitable running down of battery power. It instead turned out to be a treatise on how often things should be a simple task suck all your time away in tedium and problems.

The robot is meant to be simple as possible. Mechanically, this is pretty much the case. Electrically, it still is simple, however all the problems I've had with my sensors suggests I am doing something wrong.



Integrated System

Overview

The robot is based around the BDMicro Mavric IIB development board. This board contains an Atmega128 micro-controller. The board contains pins to attach a servo directly to the development board. In addition to the Mavric board, there are two other boards: the sensor board and the wireless serial board. The sensor board contains 5 IR range finders, a CdS cell, and a digital compass (HM55B). The wireless board contains a TX433 and RX433.

The sensor board's inputs/outputs from the Mavric board are VCC, GND, the 38.8KHz signal for IR, clk, /en, and DINOUT, for the compass, and the 6 analog outputs for the IR and CdS. The wireless board's connection is simple, VCC, Rx, and Tx. The board also contains a regulator, to use with 1.5v batteries, as to not burn out the servos. An LCD can be attached for debugging purposes, but the to avoid extra clutter, is usually removed during normal operation.

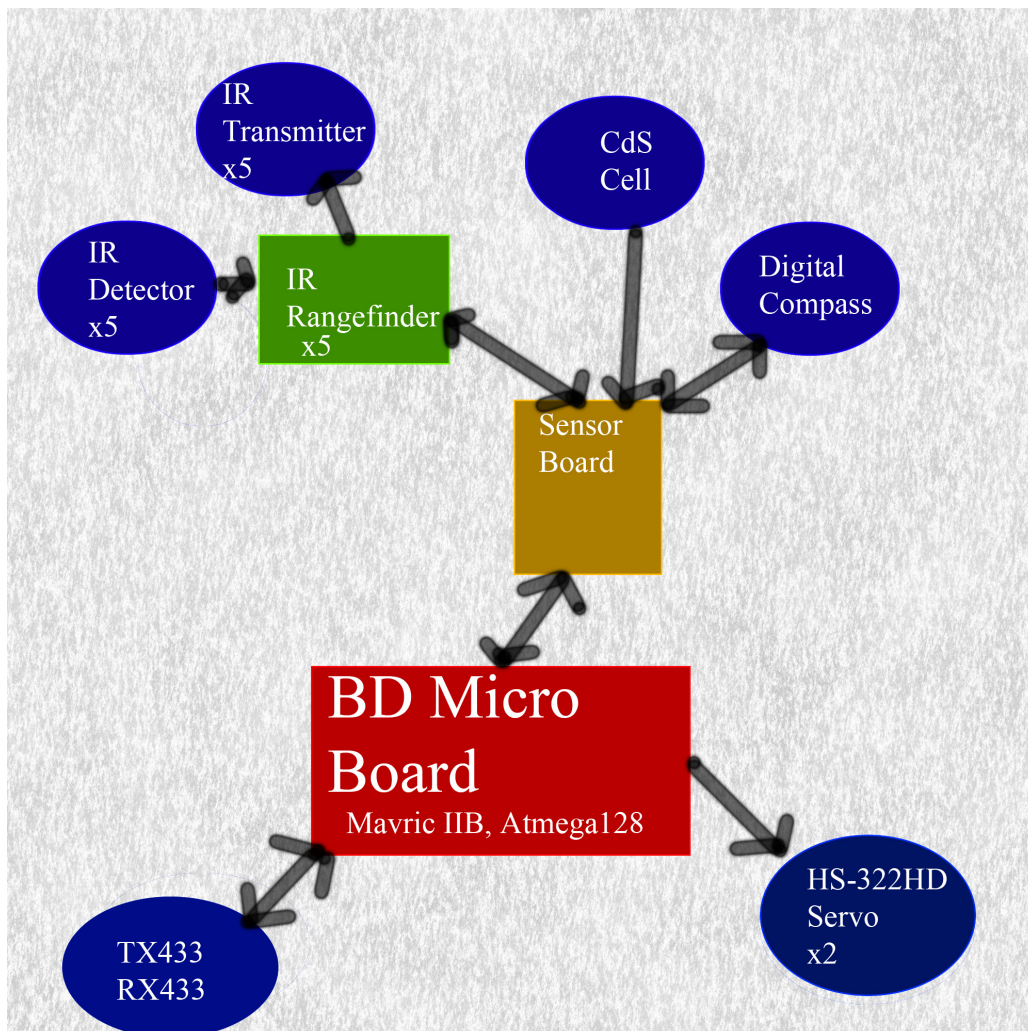


Figure 1: The integrated system of the robot

Sensor Board Design

I constructed the sensor board to make it much more convenient in dealing with the sensors. The board contains the 5 IR LED transmitter/receiver pairs, a transistor to power the LEDs, a CdS cell, the compass, and the needed resistors. [See appendix for schematic and layout] I designed the board in Altium Designer and had it cut on the T-Tech. Unfortunately, there is an inherent limitation in the T-Tech boards. All the IR range finders, as well as a couple other pins, kept dislodging from the board: the copper would un-bond with the fiberglass, causing a bad connection or the part to fall out. This can be avoided partially if designed correctly, and extra parts, such as the plastic part of a header were added as an extra layer of protection. A normal printed board has vias that form a more physical connection with the board, allowing much more strength.

The outputs of the CdS cell and 5 IR range finders is analog, and these are connected to Port F on the Mavric board, the A to D port. Four of the Port F pins are shared with the JTAG programmer, so when the robot is running, the programmer must be disconnected if the values are to be read properly. The compass uses three wires in its interface and is connected to port A. Because the board is single-sided, there are extra GND and VCC pins to allow all the connection to be routed.

Software Design

The behavioral code and reading of the compass is done in the main while loop. A interrupt every millisecond reads in the value of one pin on the A to D port, and updates the running average for that pin. In addition, a counter is updated, which keeps track of time for the mssleep function.

The LCD is controlled by bit twiddling. Data can be sent to the LCD as individual bytes, or by integers. The compass is also controlled by bit twiddling. A good portion of the code by number of lines is composed of functions for these two devices.

The servos are controlled by PWM. The 38.8KHz signal for the LEDs is also created using PWM. One initialized this systems do not need any further interaction. However, to change the speed of the servo, the duty cycle is adjusted.

In this stage of development, there is more processing power than needed. As such, a delay is added in the main loop so that the LCD can remain reasonable and the servos aren't changed abruptly to often.

Mobile Platform

The platform is very simple. It consists of two 3"x6" pieces of acrylic Plexiglas. Two servos are mounted at the rear, and a slider upfront. The wheels are wood cut with the T-Tech. The acrylic was machined using a Dremel. The Mavric board is mounted on the underside of the top piece of acrylic and the sensor board on top and to the front. All parts are attached with 8x32 machine screws. Thinner screws would work probably better on a robot this size, at least for mounting the electronics. I use a rubber band for traction on rougher surfaces.

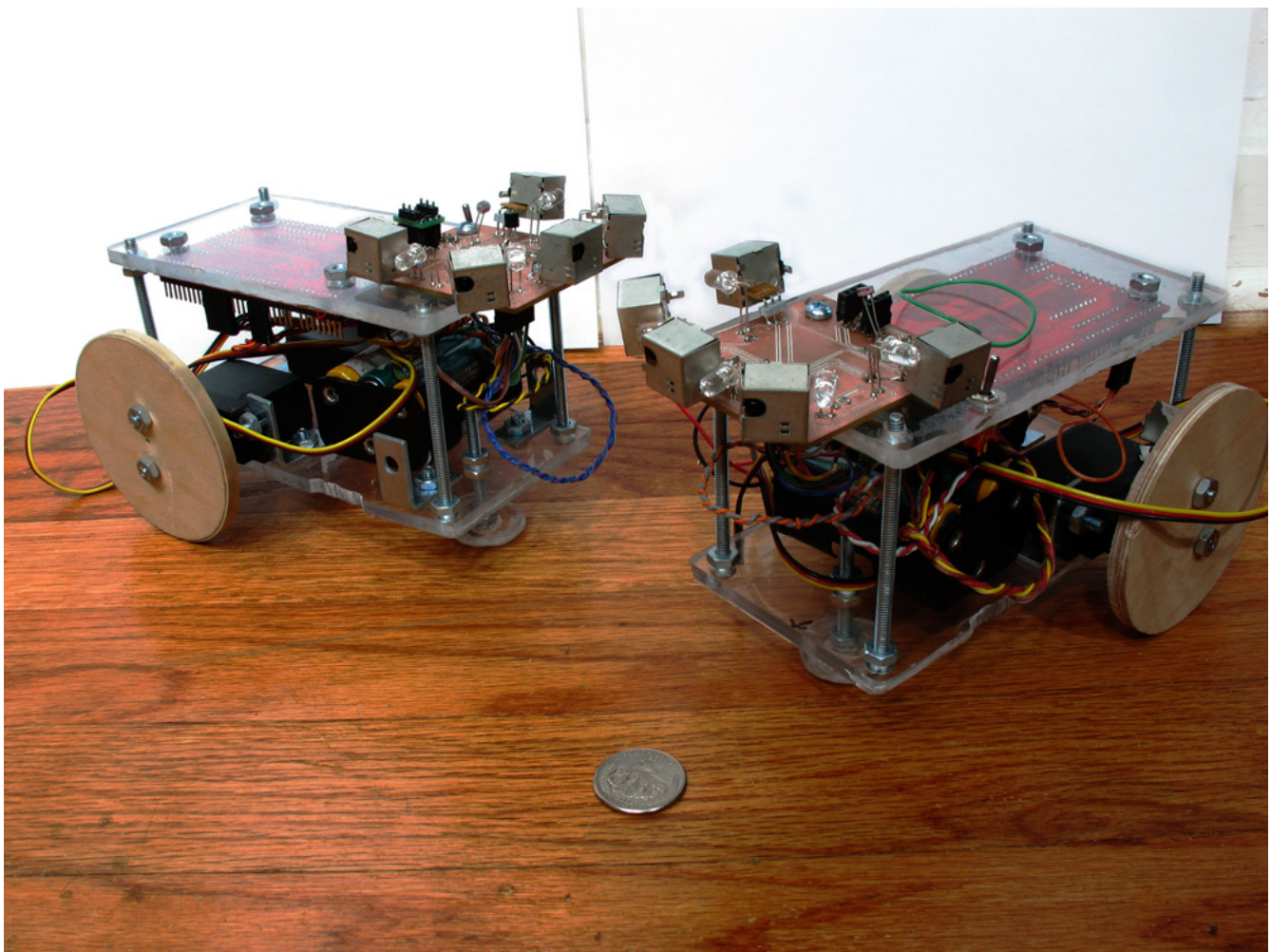


Figure 2: 90% Mechanically Complete

Actuation

The movement of the robot is powered by two hacked Hi-Tec HS-322HD servos. To control a servo, a PWM signal at 30-60 Hz is sent. In an unhacked servo, a duty cycle that's high for 1.5ms corresponds to the center position. When the servo is hacked, the mechanical stopper is removed. In addition, the potentiometer must be glued in place, in the center position. This way a 1.5 ms signal will keep the wheels motionless, and a greater or lesser signal (1-2 ms) will cause the servo to rotate. Because the potentiometer has been glued in place, the circuitry "thinks" the servo hasn't moved and continues rotation.

The speed of the robot is controlled by adjusting the duty cycle of PWM. I wrote a function that takes a value from -100 to 100 for each servo and converts that to a speed. Right now, these numbers do not directly correspond to velocity. A good working 100% speed is 50 on this scale.

There is some slippage of the wheels, but with a robot of this kind, this does not matter.

Sensors

IR Rangefinder

In an ill-fated attempt to keep costs down, I decided to create my own IR rangefinder. To do this, I used a QH3031 IR receiver. This receiver demodulates IR at 38.8KHz. Unlike the other kind of sharp can used in previous semesters, this receiver was harder to get a nice good analog signal out. By soldering a bridge from a capacitor to the digital out pin, and cutting the digital out trace, I was about to get a noisy signal out. I am using a 100 value running average to cut down on the noise [each range finder is sampled around 125Hz]. I am using PWM to create a 38.8KHz IR signal. Because the micro controller can only source around 20mA per pin, I used a transistor to drive the 5 LEDs.

Currently the range finder has a practical range of about 1-1½ feet. However, I haven't had a chance to tune the range finders or collimate them.

Compass Module

I am using a HM55B compass module in order to get absolute positioning. The interface can be three or four wires, depending on if one wants to multiplex. I implemented all communication with the compass using bit twiddling. The module uses synchronous serial communication, and had a DIN/DOUT line, /EN, and clk. Commands are 4 bits, the data come back as 22 bits, 11 bits for each component of the magnetic field – x and y.

The sensitivity is 1 uT. In my house I measured a maximum value for the earth's magnetic field of 34 uT when horizontal to the ground. The servos do not create enough magnetic fields to effect the compass's accuracy significantly.

CdS Cell

A CdS cell is an extremely simple sensor. When no light is shining, it has a high resistance, and when light is shining on it, a low resistance. In my original configuration, when it was as dark as I could get it, I read a value of 6, and as bright as I could get it I read around 180.

Wireless Serial

The wireless serial communication is used to communicate the orientation. I am using TX433/RX433 wireless transmitters/receivers. These are easy to implement, they attach straight to the TX/RX pins on the microcontroller. They have a line-of-site range of 200m if a 30-35 cm antenna is used.

Behaviors

Due to problems with my sensor board, I was not able to implement the behaviors I wanted. However, I have implemented two basic behaviors. The priority of a behavior at the moment is determined by how far down it is in a series of if statements. This is a pretty terribly system, but works for these simple behaviors.

Obstacle Avoidance

The obstacle avoidance in the robot is simple: if it sees something on the right, it goes left, an amount in proportion to the value that it receives [which is related to distance]. If something is in front of it, it will usually turn one way or the other. If it comes too close to an object straight ahead, it will back up a little and then turn some direction.

Going some arbitrary compass direction

When the robot doesn't detect any obstacles to avoid, it will travel due North or South [at least for the demonstration].

Conclusion

If I were to do this class again, I would do almost everything differently. I now know what tends to work and what definitely doesn't when it comes to robot design and time management. My biggest problem came early in the semester, when I didn't start working on my robot until late February, though this was partly due to computer problems.

Although T-Teched boards work in a lot of situations, they did not work well with my IR sensors. Well it is possible to make the board a bit sturdier now that I know what to do, it was impractical to get new boards made by the time I discovered the problems and figured out ways that would work better next time. Currently I have 3 out of 5 IR sensors working on 1 robot, and none on the other. It is a matter of too many bad connections on the board.

Before I got my robot to move in the 11th hour, I pretty much felt like I didn't accomplish anything. Even though at I have at least some behavior, I still let my self down this semester on this project.

Appendix A, The Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <limits.h>
#include <inttypes.h>

volatile uint16_t ms_count;
int ADR[8]; //The latest value read from ADC
int ADRA[8]; //The latest running average for each ADC port
int ADRAV[8][16]; //Stores 16 values for computing running average
int ADRA2[8];
int ADRA2V[8][4];
int ADCOUNT;
int ADRCOUNT;
int LCDROW;
int ARA2COUNT;
int ADRAM[4]; //Stores 4 max values out of last 16
float RAF[8];
volatile int dummy;
int POLLCOMP; //Flag to indicate compass status
int x;
int y;
int degrees;
int ObsLeft;
int ObsStraight;
int ObsRight;
int IRBL;
int IRFL;
int IRF;
int IRFR;
int IRBR;

//#define OCR0M 125 //for 16Mhz
```

```
#define OCR0M 115 //for 14.7 MHZ
#define OCR2P 189 //for 14.7 MHZ, IR LED modulator
#define SERVZERO 0xBC

int ServoSpeed;

void sendByteLCD(int data, int rsel);
void ms_sleep(uint16_t ms);
void initADC(void);
SIGNAL(SIG_OUTPUT_COMPARE0);
void initServo(void);
void sendByteLCD(int data, int rsel);
void sendIntLCD(int data, int newline);
int index_of(int Array[], int number);

/*
 * ms_sleep() - delay for specified number of milliseconds, From BDMICRO
 */
void ms_sleep(uint16_t ms)
{
    TCNT0 = 0;
    ms_count = 0;
    while (ms_count != ms)
        ;
}

/*
 * void shiftout(int data, int)
 * Shiftout an integer, msb first
 * Shiftout lowest n bits
 * Used to send data to compass
 * On port A[0:2]
 *Alex Weinert
```

```
*/
void shiftout(int data, int n)
{
    DDRA = DDRA | _BV(PA2);
    int i;
    int j;

    for (i = n - 1; i >= 0; i--)
    {
        PORTA = (PORTA & (0xFF ^ _BV(PA2))) | ((( data >> i ) & 0x01) << 2) ;
//Output bit n to port A2
        for (j = 0; j < 5; j++) { dummy++;} //Burn some time
        PORTA = PORTA | _BV(PA1); //clock it
        for (j = 0; j < 5; j++) { dummy++;} //Burn some time
        PORTA = PORTA & (0xFF ^ _BV(PA1)); //Falling Edge
        //Sensor should read on falling edge
    }

    for (j = 0; j < 5; j++) { dummy++;} //Burn time

    DDRA = DDRA & (_BV(PA2) ^ 0xFF); //set to input
}

/*
*int shiftin(int n)
*Shiftin an integer, msb first
*Gets data from compass
*On port A[0:2]
* Alex Weinert
*/
int shiftin(int n)
{
    DDRA = DDRA & (0xFF ^ _BV(PA2)); //Set port A2 to input
    int i;
    int j;
    int data = 0;
    for (i = 0; i < n; i++)
```

```
{

    PORTA = PORTA | _BV(PA1); //clock it
    //sensor should output on rising edge
    for (j = 0; j < 5; j++) { dummy++;} //Burn some time
    PORTA = PORTA & (0xFF ^ _BV(PA1)); //Falling Edge
    for (j = 0; j < 5; j++) { dummy++;} //Burn some time
    data = ((PINA >> PA2) & 0x01) + (data << 1); //Add the next bit
}

return data;
}

/*
 *void init_compass(void)
 *initialise the compass
 * Set up ports
 *Alex Weinert
 */

void init_compass(void)
{
    DDRA = DDRA | _BV(PA2) | _BV(PA1) | _BV(PA0); //Set PortA0-2 to output
    PORTA = PORTA | _BV(PA0);
    PORTA = PORTA & (0xFF ^ _BV(PA1));
    POLLCOMP = 0;
}

/*
 *int poll_compass(void)
 *poll the compass to see if it done with the measurement
 *Alex Weinert
 */
int poll_compass(void)
{
    int j;
```

```
    //Pulse Enable
    PORTA = PORTA | _BV(PA0); //Pull EN high
    for (j = 0; j < 2; j++) { dummy++;} //Burn some time
    PORTA = PORTA & (0xFF ^ _BV(PA0)); //Pull EN low
    shiftout(0xC, 4); //Send Report Status Command
    int status = shiftin(4); //Read status
    return status;
}

/*
 *void reset_compass(void)
 *Reset the compass
 * Send command to reset
 * Send command to start a measurement
 */
void reset_compass(void)
{
    int j;
    //Reset
    PORTA = PORTA & (0xFF ^ _BV(PA0)); //Pull EN low
    shiftout(0x00, 4); //Reset command
    PORTA = PORTA | _BV(PA0); //Pull EN high
    for (j = 0; j < 10; j++) { dummy++;} //Burn some time
    //Start Measurment Command
    PORTA = PORTA & (0xFF ^ _BV(PA0)); //Pull EN low
    shiftout(0x8, 4); //Start measurement command
    POLLCOMP = 1;
}

/*
 * initialize timer 0 to generate an interrupt every millisecond. From BDMICRO,
 * modified By Alex Weinert
 */
void init_timer(void)
{
    /*
     * Initialize timer0 to generate an output compare interrupt, and
```

```
* set the output compare register so that we get that interrupt
* every millisecond.
*/
TIFR  |= _BV(OCIE0);
TCCR0 = _BV(WGM01) | _BV(CS02) | _BV(CS00); /* CTC, prescale = 128 */
TCNT0 = 0;
TIMSK |= _BV(OCIE0); /* enable output compare interrupt */
OCR0  = OCR0M; /* match in 1 ms */
}

/*
*Initial the Analog to Digital Converter
*Set up for chanel 0, Free running, Left justified,
*with 128 Prescale, useing AREF
*/
//Note: remember to chrck on my board[port F]
void initADC(void)
{
    ADMUX = _BV(ADLAR);
    ADCSRA = _BV(ADEN) | _BV(ADSC) | _BV(ADFR) | _BV(ADPS2) | _BV(ADPS1) |
    _BV(ADPS0);
}

/*
* millisecond counter interrupt vector From BDMICRO, modified by Alex Weinert
* Reads the Value of the ADC, increments the port read
* Calculates a 16 value running average of each Port as well
*/
SIGNAL(SIG_OUTPUT_COMPARE0)
{
    //count for ms_sleep
    ms_count++;

    //Record Values
    ADR[ADCOUNT] = ADCH;
```



```
ADRAV[ADCOUNT][ADRCOUNT] = ADCH;

//Increment values
ADCOUNT++;
ADCOUNT = ADCOUNT & 0x7; //Make sure max is 7
ADMUX = _BV(ADLAR) | ADCOUNT;

//Compute Running Average
//This is a running average of the 4
// Highest values of the last 16
//[Approx 16ms]
ADCOUNT--;
float RA = RAF[ADCOUNT] * 99 + ADR[ADCOUNT];
RAF[ADCOUNT] = RA / 100;
//long int RA = ADRA[ADCOUNT] * 999 + ADR[ADCOUNT];
ADRA[ADCOUNT] = RAF[ADCOUNT];
ADCOUNT++;

}

void initServo(void)
{

    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM11); //Phase Corect, Frequency
Correct PWM, period from ICR
    //TCCR1B = _BV(WGM13) | _BV(CS12) | _BV(CS10); //Prescale = 1024
    //TCCR1B = _BV(WGM13) | _BV(CS12); //Prescale = 256
    TCCR1B = _BV(WGM13) | _BV(CS11) | _BV(CS10); //Prescae = 64
    ICR1 = 0x800; //Period F
    OCR1A = SERVZERO; //Duty Cycle 1 For 14.7 Mhz, BC is 0 rotation
    OCR1B = SERVZERO; //Duty Cycle 1
```

```
    //TCNT1 = 0x0000;
    DDRB = DDRB | 1 << 5 | 1 << 6; //Set PB5 and PB6 to output
    return;
}
```

```
void sendByteLCD(int data, int rsel)
{
    //int data - Byte to send to LCD
    //bool rsel - if 0: comm, if 1: data
    //LCD setup:
    //DB0:3 = PC0:3
    //E: PC4
    //R/NW: PC5
    //RS: PC6

    int HighNibble = (data >> 4) & 0xF;
    int LowNibble = data & 0xF;

    //Send High nibble
    PORTC = HighNibble | 1 << PC4 | rsel << PC6;
    //Send Low nibble
    PORTC = 0x00;
    PORTC = LowNibble | 1 <<PC4 | rsel << PC6;
    PORTC = 0x00;
    int z,i;
    for(i=0; i < 500; i++) //Delay
    {
        z++;
    }

    return;
}
```

```
void sendNibbleLCD(int data, int rsel)
{
```

```
    data = data & 0xF; //Bit mask, useonly lower 4 bits
    PORTC = data | 1 << PC4 | rsel << PC6; //send nibble
    ms_sleep(1); //wait
    PORTC = 0x00; //clear port
    return;
}

void newLineLCD(void)
{
    if (LCDROW == 0)
    {
        sendByteLCD(0xC0,0); //Send cursor to second row
        LCDROW = 1;
    }
    else
    {
        sendByteLCD(0x02,0); //Send cursor to home
        ms_sleep(2);
        LCDROW = 0;
    }
}

/*
 *Send an integer to the LCD. Only sends 3 chars
 *if newline = 1, go to next line
 *if newline = 0, print on current line
 */
void sendIntLCD(int data, int newline)
{
    int thousands = (data % 10000) / 1000;
    int hundreds = (data % 1000) / 100;
    int tens = (data % 100) / 10;
    int ones = data % 10;
    if (newline)
```

```
    {
        newLineLCD();
    }
    sendByteLCD(thousands+0x30,1);
    sendByteLCD(hundreds+0x30,1);
    sendByteLCD(tens+0x30,1);
    sendByteLCD(ones+0x30,1);
}
```

```
/*
 *int int11toint(int)
 *Convert the 11bit 2's complement number from the compass
 * to int
 */
int int11toint(int int11)
{
    int number = int11;
    if (int11 >> 11 & 0x01) //check if negative
    {
        number = number | (INT_MAX ^ 0x7FF); //if number is negative, fill in
rest of bits with 1's
    }
    return number;
}
```

```
void initLCD(void)
{
    DDRC = DDRC | 0x7F; //Set PC0:6 to output

    LCDROW = 0;
    //Initialise 4bit mode
    ms_sleep(15);
    sendNibbleLCD(0x3,0);
    ms_sleep(5);
}
```

```
    sendNibbleLCD(0x3,0);
    ms_sleep(1);
    sendNibbleLCD(0x3,0);
    ms_sleep(5);
    sendNibbleLCD(0x2,0);
    //2 Line Mode
    ms_sleep(1);
    sendByteLCD(0x28,0);
    //Display On
    sendByteLCD(0x0C,0);
    //Clear Screen, Cursor Home
    sendByteLCD(0x01,0);
    //Cursor Increment, Shift off
    ms_sleep(2);
    sendByteLCD(0x06,0);
    //Test
    sendByteLCD(0x41,1);
    return;
}

/*
 *
 *A simple function right now, just set
 *Duty cycle for servo
 * 0 = Right = A
 * 1 = Left = B
 */
void setServoSpeed(int speed, int servoL)
{
    if (servoL)
    {
        OCR1B = speed;
    }
    else
    {
        OCR1A = speed;
    }
}
```

```
}
```

```
/* Set the servo speed
```

```
* void servoSpeed(int,int)
```

```
* valid input range: -100:100
```

```
*
```

```
*/
```

```
void servoSpeed(int speedR, int speedL)
```

```
{
```

```
    if (speedR > 100 || speedR < -100)
```

```
    {
```

```
        return;
```

```
    }
```

```
    if (speedL > 100 || speedL < -100)
```

```
    {
```

```
        return;
```

```
    }
```

```
    setServoSpeed(SERVZERO - (speedL >> 2), 1);
```

```
    setServoSpeed(SERVZERO + (speedR >> 2), 0);
```

```
}
```

```
/*
```

```
*Initialise the 38.8Khz signal for the IR LED
```

```
* Using Port B7
```

```
* For the 14.7 Mhz clock, no prescale, OCR2 = 189 [Half Period]
```

```
*
```

```
*/
```

```
void initIRLED(void)
```

```
{
```

```
    TCCR2 = _BV(WGM21) | _BV(COM20) | _BV(CS21); //CTC mode, Toggle on match, no  
prescale
```

```
    OCR2 = OCR2P; //
```

```
    DDRB = DDRB | _BV(PB7); //Set PB7 to output
```

```
}
```

```
int main(void)
{
    /**
     *Initialise Everything
     */
    init_timer(); //millisecond timer
    DDRB = DDRB | 0x01;
    initServo();
    /* enable interrupts */
    sei();
    initLCD();
    initADC();
    initIRLED();
    init_compass();

    /**
     *Test / Debug Loop
     *
     */
    while(1) {
        ms_sleep(50);
        //ADRA2COUNT = ++ADRACOUNT & 0x3;
        sendIntLCD(ADRA[0],1);
        sendIntLCD(ADRA[1],0);
        sendIntLCD(ADRA[2],0);
        sendIntLCD(ADRA[3],0);
        sendIntLCD(ADRA[4],0);
        sendIntLCD(x,1);
        sendIntLCD(y,0);
        //setServoSpeed(y,0);
        //setServoSpeed(ADR[7],0);
        //sendIntLCD(ADR[7],1);
        //sendIntLCD(ADRAV[0][1],1);
        //sendByteLCD(0x51,1);
        PORTB ^= 0x01; //useful to see if things are runninh
```

```
//Behavioral Code
IRBL = 52; //ADRA[0];
IRFL = 52; //ADRA[1];
IRF = ADRA[3];
IRFR = ADRA[2];
IRBR = ADRA[0];
ObsStraight = IRF - 50;
ObsRight = IRFR - 50;
ObsLeft = IRFL - 50;

int diffRL = ObsLeft - ObsRight;

//Calculate servo speeds based on Right and left obs
if (diffRL < 0)
{
    diffRL = -diffRL;
}
int servoSpeedRIR = 50;
int servoSpeedLIR = 50;
if (ObsRight > ObsLeft)
{
    servoSpeedRIR = 50;
    servoSpeedLIR = 50 - diffRL;
    if (servoSpeedLIR < 0)
    {
        servoSpeedLIR = 0;
    }
}
if (ObsRight < ObsLeft)
{
    servoSpeedLIR = 50;
    servoSpeedRIR = 50 - diffRL;
    if (servoSpeedLIR < 0)
    {
        servoSpeedLIR = 0;
    }
}
```



```
    }  
}  
if (diffRL < 10)  
{  
    servoSpeedRIR = 50;  
    servoSpeedLIR = 50;  
}  
if (ObsStraight > 20)  
{  
    servoSpeedLIR >> 1;  
    servoSpeedRIR >> 1;  
}  
if (ObsStraight > 30)  
{  
    servoSpeedLIR >> 1;  
    servoSpeedRIR >> 1;  
}  
if (ObsStraight > 35)  
{  
    if ( diffRL < 10 )  
    {  
        servoSpeedLIR = -10;  
        servoSpeedRIR = -10;  
    }  
    if (ObsRight > ObsLeft)  
    {  
        servoSpeedRIR = 10;  
        servoSpeedLIR = -10;  
    }  
    if (ObsRight < ObsLeft)  
    {  
        servoSpeedLIR = 10;  
        servoSpeedRIR = -10;  
    }  
}
```

```
}

if (x > 1000)
{
    x = 2048 - x;
}
if (y > 1000)
{
    y = 2048 - y;
}

if (ObsStraight < 15 && ObsRight < 20 && ObsLeft < 20 && diffRL < 18)
{
    if (y > 5 && x > 0)
    {
        servoSpeedLIR = 50;
        servoSpeedRIR = 50 - y;
    }
    if (y < 5 && x > 0)
    {
        servoSpeedRIR = 50;
        servoSpeedLIR = 50 - y;
    }
    if ( x < 0)
    {
        servoSpeedRIR = 50;
        servoSpeedLIR = 10;
    }
}

servoSpeed(servoSpeedRIR, servoSpeedLIR);

if (POLLCOMP)
{
    int stat = poll_compass();
    if (stat == 0xC)
```

```
        {
            x = shiftin(11);
            y = shiftin(11);
            //float rad = -(float)y/(float)x;
            //degrees = (int)(rad * 57);
            PORTA |= _BV(PA0);
            POLLCOMP = 0;
        }
    }
else
{
    reset_compass();
}
}

return 0;
}
```

Appendix B, Sensor Board Schematics

