Student Name: Marcum Greeson
TA : Louis Brandy
William Dubel
Max Keossick
Instructor. A. A Arroyo

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Final Report

Nolte

High Speed Line Follower

Table of Contents

Introduction

This paper describes a robot that has a small compact design with a minimum number of sensors and components in order to track a complex line at high speed with as much efficiency as possible. It uses a single sonar that helps the robot detect obstacles in its path. This sensor is highly sensitive and can detect more objects in its path than IR sensors. The only other type of sensor this robot has is photoreflectors, which are used to detect a black line on a contrasting surface.

Purpose

This robot was designed to be a basic platform that could be modified in the future, and provide versatility in multiple situations. Nolte is designed to be compact, with a modular design that could be easily taken apart and put back together.

Its only tasks are to avoid obstacles, and track lines that lead the robot down a given path. This platform tackles the action of line following and incorporates high speed capabilities. This behavior is a staple in most beginning robots platforms that perform certain tasks. Since navigation is a high level ability for robots, line following allows the robot to know where to go without sophisticated components and programming.

Environment

Nolte functions on a white surface using black electric tape for the actual path that it will follow. It is only allowed to travel on the white surface following the given path. With this setup, it is easy to adapt Nolte to many different environments. However, since the line following is a function of the photoreflectors that use infrared light to detect reflective and non-reflective surfaces, Nolte is limited to indoor use. This is because outdoor light has high concentrations of infrared light that saturates the line following head unit

Design

The platform for this robot was designed using AutoCad drawing software, and cut out off 1/8" hobby board using a T-tech machine. The main chassis was glued together with regular wood glue, and components were mounted with simple nuts and bolts. The initial platform was highly contemplated making sure that all the components would fit properly and have accuracy in positioning to give the most efficient placement as possible. Designing using AutoCad and the computer aided cutting machine allowed for this type of precision which ended with a platform that was functional and unmodified. All electronics were soldered on boards and mounted with hardware that would be reliable and easy to remove and replace.

Electronics and Hardware

This robot consists of a minimum number of electronic parts that consist of the microcontroller, power board, motor controller, dc motors, sonar, line follower board, LCD, and minimum number of wire. There is no wire raping or temporary circuits on the robot, which allow the design to have a reliable long lasting structure. All cables are soldered and hot glued to provide sturdy connects and rugged long lasting functionality.

Microcontroller

The microcontroller is a Mavric II, an atmel atmega 128 based processor, purchased from BDmicro. This board has many nice features such as 53 I/O pins, 10 bit A/D port, I2C bus, two uarts, onboard power regulator, PWM outputs for motor control signals, plenty of nonvolitle memory, and fast 16Mhz clock speed for the processor. This processor has a lot to offer and was chosen for its small size and availability of supported software, including the programmer and the compiler. Gcc compiler was chosen to set up programs for the board.

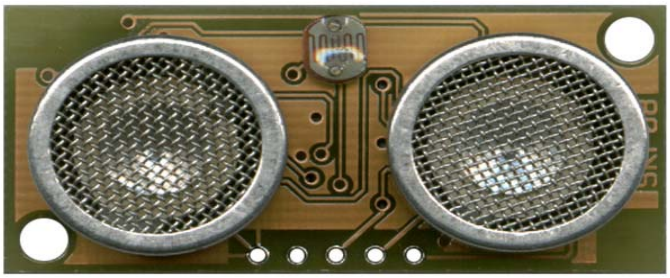
Motorcontroller

The motor controller that was used it the Sozbot HB-03. It operates in a voltage range from 6V-18V. It has proven to be a well built component that can handle a peak current of 5 amps. The controller is dual channel, bidirectional, and operates on servo style PWM's (or RC style). It was an expensive board, but the reliability and great characteristic made it a worthwhile purchase.

Sonar

This is a great piece of hardware, which allows Nolte to have a great perception of obstacles in its environment. The Sonar is the ultrasonic range finder, a Devantech SRF08, which is shown below. It is also an expensive piece of hardware, but all the nice features allow the user to integrate it easily with any microcontroller that has I2C capabilities. The range of this sensor is quite impressive. It can read a distance of 3 cm to 6 meter, with a cone shaped line of sight of about 60 degrees wide extending from the front of the sensor. One of these sensors can provide great vision for a robot, and it a pair of

these sensors is used, it is possible to get a viewing pattern of 180 degrees. Nolte only need a limited amount of sight, so only onle was chosen and proved to be very receptive.



Photoreflectors

Hamamatsu P5587 photoreflectors were chosen because of there compact and reliable design. Another plus is that they are really cheap. These photoreflectors are perfect for mounting on a circuit board, because the diode for IR and the detector are both mounted on one chip. The devices are almost ready for direct integration into the microcontroller, connecting to any I/O pins. Only other components that are need are a load resistance and a capacitor across the power and ground pin. These sensors are very reliable, however they do have a limited functioning distance from the surface where they are used to detect a reflecting surface. The range is within about 0 to 2 cm.

DC Motors

The motors that were chosen are the Hsiang Neng dc metal gear head motors. These are 12 V motors that run at 290 rpms. These motors have a 43:1 gear ratio and paired with enough voltage can provide accurate a very high speed propulsion and torque to really get the platform moving. However, there have been some trouble with the gears stripping for platforms that require a high degree of torque. Since Noltle doesn't have a need for extremely high torque, these motors were suitable for the performance expected by this platform.

Batteries

The batteries that were chosen for Nolte are the Rayovac IC3 NiMH rechargeable batteries. These batteries have been great. Even though they are quite expensive, they recharge in about 15 minutes. A set of eight AA batteries provide the 11 volts that Nolte uses to power itself. The batteries are rated a 2000 mAh, and have proven to be a great addition to the parts that are integrated on the platform. These batteries prove to

output many hours of use by the platform, provided that the circuitry that is used doesn't have a lot of power loss due to poor circuit design.

Tail-wheel

Another nice feature of this platform is the tail wheel that was used. It is a model airplane tail-wheel that was modified to be a rear wheel for the platform. This wheel supports the rear section of the platform and provides friction free turning ability. This wheel is exceptionally sturdy and adds nice turning characteristics to the robot, unlike using a caster.

Behaviors

Nolte is designed to follow a number off complex paths at high speed. This robot has a lot of functionality with a minimum amount of complexity. Since this was a first time experience in building an entire robot, these characteristics were very desirable. Even though Nolte's concepts were quite simple, line following is not an extremely simple task, especially when one plans on going extremely fast. I used similar logic for the line following as set forth by previous students that had success in designing efficient line followers. I took a lot of working ideas from students such as William Dubel, who provided an assortment of working solutions to the line following problem. Nolte has the ability to follow many different types of paths. For instance, Nolte can follow strait lines, curved lines, right angle turns, and some acute angle turns.

Conclusion

Nolte is a great robot. It has been somewhat of a trouble free platform. The potential of this robot is very great with its extreme agility and speed it is really fun to watch. It can run at speed that most robots can't. However, this proves to be quite detrimental to the functionality in line following. It complicates the problems involved in line following. With a few more additions to the logic that was available from previous student, Nolte would be able to blaze around a path with great precision.

Besides programming, a few other changes to components would increase the accuracy to this robot. Namely, the board that was designed by Matt Yoder, that was used on this robot is a little limited based on the dimensions of the platform and the distance that the robot moves. Also, the wheels that are being used are very large with a 5.5" diameter. The wheels that I had special ordered never came in, so I had to improvise with a set of training wheels used for small bikes. They work decently, but the circumference of the wheels is not very true causing the robot to veer off the path quite often. This constant veering causes the robot to consistently overcorrect its direction. Finally, a more complex sensor system could be incorporated to allow the robot to track the line at a faster speed.

Reference

Gear Head Motor Datasheet. LynxMotion, Copyright 2000
http://www.lynxmotion.com/ghm02.htm

Devantech SRF08 UltraSonic Range Finder. Junun.org
http://www.junun.org/MarkIII/Info.jsp?item=32

Hamamatsu P5587 Photoreflector Datasheet, Junun.org
http://www.junun.org/MarkIII/Info.jsp?item=48

Sozbots Dual RC H-Bridge Motor Controller Users Guide, Lynxmotion, Copyright 2003
http://www.lynxmotion.com/Product.aspx?productID=90&CategoryID=10

Source Code

```
/*********************************************
* Title:          Servo.c
* Programmer:     Anne Harmeson
* Author:         Cyrus Harrison (cdh@cise.ufl.edu)
* Date:                 4/16/2004
* Version:        3
* Adapted By:  Marcum Greeson
* other reference code: William Dubels line following example
*                                  Bdmicro example code for SRfo8
sonar and I2C
* Description:
*                 Code to use Servos.

*********************************************/
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/twi.h>

#include <stdio.h>

#include "i2c.h"
#include "srf08.h"
  //added from sonar3

#include <stdio.h>
#include "LCD.h"

#include "Servos.h"
// added
#define CPU_FREQ 16000000L  /* set to clock frequency in Hz */

#define BAUD_RR ((CPU_FREQ/(16L*9600L) - 1))

volatile uint16_t ms_count;


//#define   LT_PORT     PINC & 0xF0

int main(void)
{
      //INITIALIZATIONS

        sei();
        lcd_set_ddr();
        lcd_init();
            servos_init();
            // added sonar3 shit
            uint16_t distance;
        uint16_t temp;

        init_timer();
```

```
        /* enable UART0 */
              UBRR0H = (BAUD_RR >> 8) & 0xff;
              UBRR0L = BAUD_RR & 0xff;
              UCSR0B = BV(TXEN); /* enable transmitter only */

        /* enable interrupts */
            //  sei();

        /* initialize stdio */
              fdevopen(lcd_send_byte, NULL, 0);

        /* set the I2C bit rate generator to 100 kb/s */
              TWSR &= ~0x03;
              TWBR  = 28;
              TWCR |= BV(TWEN);

  while (1) {

                  ms_sleep(1);                  /* sleep for .5 second
*/
                  srf08_ping(0x70, RANGE_CM);      /* initiate a ping,
distance in cm */
                  ms_sleep(70);                    /* wait for 70 ms */
                  srf08_range(0x70, 0, &distance);
                  temp = distance;/* read first echo */
                  lcd_send_command(00);
                  lcd_send_command(01);
                  printf("range = %3u cm", temp);


                while (temp>=30)
                {

                  ms_sleep(1);                  /* sleep for .5second
*/
                  srf08_ping(0x70, RANGE_CM);      /* initiate a ping,
distance in cm */
                  ms_sleep(70);                    /* wait for 70 ms */
                  srf08_range(0x70, 0, &distance);
                  temp = distance;/* read first echo */
                  lcd_send_command(00);
                  lcd_send_command(01);
                  printf("range = %3u cm", temp);


                   trackToLine();



            }

                  stop_servos();
```

```
      }
}


      void servos_init(void)
            {
/*    Setup PWM on OC1A, OC1B, OC1A, LOOK AT WWW.AVRFREAKS.COM AND
DOWNLOAD AVRcalc, JUST SEARCH FOR AVRcalc
* YOU NEED A 20ms total period, USE THE INPUT CAPTURE FOR 10MS COUNT,
DON'T BE LAZY LOOK IN M. CONTROLLER MANUAL
*I AM USING THE MAVRIC II with a atmega 128 atmel processor running at
16mHz
*THIS MEANS THAT INPUT CAPTURE COUNTS UP TO 10ms, THEN COUNTS DOWN 10MS
*THEN KIDS YOU HAVE A NICE 20ms PERIOD THAT DOESN'T COME EVERY 28 DAYS
*                     -
*                   _   -
*                 _     -
*               _       -
*             _         -
*           _           -
*         _             -
*       _               -
*_______-----------20ms-----------|_____________________ 20ms PERIOD
*
*
*     ___________  TOP (BTWN 1ms and 2ms)
*    |        |
*    |        |_____________________________________________________
*    0                                              20ms
*
*
*/
      // Data Direction for PWMs
      outp(0xF0,DDRB);
      //Setup the Prescaler

        outp(0x13,TCCR1B);

        outp(0x09,ICR1H);
        outp(0xC4,ICR1L);

        outp(0x09,OCR1AH);
        outp(0x08,OCR1AL);

        outp(0x09,OCR1BH);
        outp(0x08,OCR1BL);

        outp(0x09,OCR1CH);
        outp(0x08,OCR1CL);
      }



// START TIMERS ON TIMER 1, 16 BIT TIMER- FOR PINS P7,P6,P5-- MY PIN6
IS f@(% !!!
// because I didn't have ddrb set right!
      void start_servos(void)
```

```
            {
            outp(0xFC, TCCR1A);
            }


// DISCONNECTS THE TIMERS THAT ARE INIATED FOR OCR1A, OCR1B, OCR1C
      void stop_servos(void)
            {
            set_Lwheel_speed(0x09,0x11);
            set_Rwheel_speed(0x09,0x11);

            }


// FULL SPEED AHEAD BABY, NO LOOK'IN BACK UNLESS SHE'S CUTE
      void drive_forward(void)
            {
            set_Lwheel_speed(0x09,0x18);
            set_Rwheel_speed(0x09,0x18);

            }
// TO MANY BEERS, YOU GOT TO BACK UP OR YOU'LL HIT THE CAR
      void drive_backward(void)
            {
            set_Lwheel_speed(0x08,0xCA);
            set_Rwheel_speed(0x08,0xCA);
            }

      void drive_right(void)
            {
            set_Lwheel_speed(0x09,0x18);
            set_Rwheel_speed(0x09,0x13);
            }

      void drive_left(void)
            {
            set_Lwheel_speed(0x09,0x13);
            set_Rwheel_speed(0x09,0x18);
            }

            void small_right(void)
            {
            set_Lwheel_speed(0x09,0x18);
            set_Rwheel_speed(0x09,0x17);
            }

            void small_left(void)
            {
            set_Lwheel_speed(0x09,0x17);
            set_Rwheel_speed(0x09,0x18);
            }

//*****************************************************************
*******************************


      void set_Rwheel_speed(unsigned int val1, unsigned int val2)
```

```
      {
      outp(val1,OCR1AH);
      outp(val2,OCR1AL);
      }


void set_Lwheel_speed(unsigned int val1, unsigned int val2)
      {
      outp(val1,OCR1CH);
      outp(val2,OCR1CL);
      }

unsigned char trackToLine()
      {

            static unsigned char prevDir = 0;
            unsigned char LT_PORT;
            LT_PORT = PINC & 0xF0;

      if (LT_PORT!=prevDir)
      {

            switch (LT_PORT)  // using a two-pair line-tracker
            {
               case 0x00:     // centered
                  {
           lcd_send_command(00);
                  lcd_send_command(01);
                  printf("on the line");
                  start_servos();
           drive_left();
           break;

                  }

                case 0x10:     //
                  {
           lcd_send_command(00);
                  lcd_send_command(01);
                  printf("right on");
                  start_servos();
                  drive_left();
                  break;

                  }

              case 0x20:     // centered on the line
                  {

           lcd_send_command(00);
                  lcd_send_command(01);
                  printf("left");
                  start_servos();

           drive_left();
                  break;
                  }
```

```
      case 0x30:     // centered on the line
         {
 lcd_send_command(00);
         lcd_send_command(01);
         printf("left");
         start_servos();

 drive_left();
         break;
         }

      case 0x40:     // centered on the line
         {
         lcd_send_command(00);
         lcd_send_command(01);
         printf("right");
         start_servos();

 drive_right();
         break;
         }

case 0x50:     // centered on the line
         {
 lcd_send_command(00);
         lcd_send_command(01);
         printf("right");
         start_servos();

 small_right();
         break;
         }

case 0x60:     // centered on the line
       {
 lcd_send_command(00);
         lcd_send_command(01);
         printf("left");
         start_servos();

 drive_forward();
         break;
         }

case 0x70:     // centered on the line
         {
 lcd_send_command(00);
         lcd_send_command(01);
         printf("left");
         start_servos();

 drive_left();
         break;
         }

case 0x80:     // centered on the line
```

```
        {
 lcd_send_command(00);
        lcd_send_command(01);
        printf("off the line");
        start_servos();

 drive_right();
        break;
        }

case 0x90:     // centered on the line
        {
 lcd_send_command(00);
        lcd_send_command(01);
        printf("going left");
        start_servos();

 drive_forward();
        break;
        }

case 0xA0:     // centered on the line
        {
 lcd_send_command(00);
        lcd_send_command(01);
        printf("left");
        start_servos();

 small_left();
        break;
        }

case 0xB0:     // centered on the line
        {
 lcd_send_command(00);
        lcd_send_command(01);
        printf("off the line");
        start_servos();

 small_left();
        break;
        }

case 0xC0:     // centered on the line
        {
 lcd_send_command(00);
        lcd_send_command(01);
        printf("off the line");
        start_servos();

 drive_right();
        break;
        }

case 0xD0:     // centered on the line
        {
 lcd_send_command(00);
```

```
                        lcd_send_command(01);
                        printf("going left");
                        start_servos();

                 small_right();
                        break;
                        }

                case 0xE0:     // centered on the line
                        {
                 lcd_send_command(00);
                        lcd_send_command(01);
                        printf("oh no right");
                        start_servos();

                 drive_right();
                        break;
                        }

                case 0xF0:     // centered on the line
                        {
                        lcd_send_command(00);
                        lcd_send_command(01);
                        printf("off the line");
                        drive_right();



                        break;
                        }


            }

                  if (LT_PORT==0xF0 && prevDir!=0xF0) // lost the line,
                        prevDir = LT_PORT;            // continue
prevDir


                  }
                        return LT_PORT;
      }
// THIS IS A DELAY FOR THE SERVOS FOR JUST CHILL'IN

      void delay_servos(void)
            {
            uint16_t i;
            uint16_t k;

            uint16_t var1 = 0;

            for (i = 0; i < 20; i++)
            {
                  for (k = 0; k < 20; k++)
                  {
                        var1 = 0;
                  }
```

```
            }
      }


// added sonar3
int def_putc(char ch)
{
  /* output character to UART0 */
  while ((UCSR0A & BV(UDRE)) == 0)
    ;
  UDR0 = ch;
  return ch;
}


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


/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
  ms_count++;
}


/*
 * initialize timer 0 to use the real time clock crystal connected to
 * TOSC1 and TOSC2 to generate a near 1 ms interrupt source
 */
void init_timer(void)
{
  /*
   * Initialize timer0 to use the 32.768 kHz real-time clock crystal
   * attached to TOSC1 & 2.  Enable output compare interrupt and set
   * the output compare register to 32 which will cause an interrupt
   * to be generated every 0.9765625 milliseconds - close enough to a
   * millisecond.
   */
  TIFR  |= BV(OCIE0)|BV(TOIE0);
  TIMSK |= BV(OCIE0);    /* enable output compare interrupt */
  TIMSK &= ~BV(TOIE0);   /* disable overflow interrupt */
  ASSR  |= BV(AS0);      /* use asynchronous clock source */
  TCNT0  = 0;
  OCR0   = 32;           /* match in 0.9765625 ms */
  TCCR0  = BV(WGM01) | BV(CS00); /* CTC, no prescale */
  while (ASSR & 0x07)
```

```
    ;
  TIFR  |= BV(OCIE0)|BV(TOIE0);}
```

I2c Code

```
#include <avr/twi.h>
#include <avr/pgmspace.h>

#include <stdio.h>
#include <inttypes.h>

#include "i2c.h"

const char s_i2c_start_error[]   PROGMEM = "I2C START CONDITION ERROR";
const char s_i2c_sla_w_error[]   PROGMEM = "I2C SLAVE ADDRESS ERROR";
const char s_i2c_data_tx_error[] PROGMEM = "I2C DATA TX ERROR";
const char s_i2c_data_rx_error[] PROGMEM = "I2C DATA RX ERROR";
const char s_i2c_timeout[]       PROGMEM = "I2C TIMEOUT";
const char s_i2c_error[]         PROGMEM = "I2C ERROR\n";
const char s_fmt_i2c_error[]     PROGMEM = " TWCR=%02x STATUS=%02x\n";

extern volatile uint16_t ms_count;


/*
 * signal the end of an I2C bus transfer
 */
int8_t i2c_stop(void)
{
  TWCR = BV(TWINT)|BV(TWEN)|BV(TWSTO);
  while (TWCR & BV(TWSTO))
    ;
  return 0;
}


/*
 * display the I2C status and error message and release the I2C bus
 */
void i2c_error(const char * message, uint8_t cr, uint8_t status)
{
  i2c_stop();
  printf_P(message);
  printf_P(s_fmt_i2c_error, cr, status);
}


/*
 * signal an I2C start condition in preparation for an I2C bus
 * transfer sequence (polled)
 */
int8_t i2c_start(uint8_t expected_status, uint8_t verbose)
{
  uint8_t status;

  ms_count = 0;

  /* send start condition to take control of the bus */
```

```
  TWCR = I2C_START;
  while (!(TWCR & BV(TWINT)) && (ms_count < I2C_TIMEOUT))
    ;

  if (ms_count >= I2C_TIMEOUT) {
    if (verbose) {
      i2c_error(s_i2c_start_error, TWCR, TWSR);
      i2c_error(s_i2c_timeout, TWCR, TWSR);
    }
    return -1;
  }

  /* verify start condition */
  status = TWSR;
  if (status != expected_status) {
    if (verbose) {
      i2c_error(s_i2c_start_error, TWCR, status);
    }
    return -1;
  }

  return 0;
}


/*
 * initiate a slave read or write I2C operation (polled)
 */
int8_t i2c_sla_rw(uint8_t device, uint8_t op, uint8_t expected_status,
                  uint8_t verbose)
{
  uint8_t sla_w;
  uint8_t status;

  ms_count = 0;

  /* slave address + read/write operation */
  sla_w = (device << 1) | op;
  TWDR = sla_w;
  TWCR = I2C_MASTER_TX;
  while (!(TWCR & BV(TWINT)) && (ms_count < I2C_TIMEOUT))
    ;

  if (ms_count >= I2C_TIMEOUT) {
    if (verbose) {
      i2c_error(s_i2c_sla_w_error, TWCR, TWSR);
      i2c_error(s_i2c_timeout, TWCR, TWSR);
    }
    return -1;
  }

  status = TWSR;
  if ((status & 0xf8) != expected_status) {
    if (verbose) {
      i2c_error(s_i2c_sla_w_error, TWCR, status);
    }
    return -1;
```

```
  }

  return 0;
}


/*
 * transmit a data byte onto the I2C bus (polled)
 */
int8_t i2c_data_tx(uint8_t data, uint8_t expected_status, uint8_t
verbose)
{
  uint8_t status;

  ms_count = 0;

  /* send data byte */
  TWDR = data;
  TWCR = I2C_MASTER_TX;
  while (!(TWCR & BV(TWINT)) && (ms_count < I2C_TIMEOUT))
    ;

  if (ms_count >= I2C_TIMEOUT) {
    if (verbose) {
      i2c_error(s_i2c_data_tx_error, TWCR, TWSR);
      i2c_error(s_i2c_timeout, TWCR, TWSR);
    }
    return -1;
  }

  status = TWSR;
  if ((status & 0xf8) != expected_status) {
    if (verbose) {
      i2c_error(s_i2c_data_tx_error, TWCR, status);
    }
    return -1;
  }

  return 0;
}


/*
 * receive a data byte from the I2C bus (polled)
 */
int8_t i2c_data_rx(uint8_t * data, uint8_t ack, uint8_t
expected_status,
                   uint8_t verbose)
{
  uint8_t status;
  uint8_t b;

  ms_count = 0;

  if (ack) {
    TWCR = BV(TWINT)|BV(TWEN)|BV(TWEA);
  }
```

```
  else {
    TWCR = BV(TWINT)|BV(TWEN);
  }
  while (!(TWCR & BV(TWINT)) && (ms_count < I2C_TIMEOUT))
    ;

  if (ms_count >= I2C_TIMEOUT) {
    if (verbose) {
      i2c_error(s_i2c_data_rx_error, TWCR, TWSR);
      i2c_error(s_i2c_timeout, TWCR, TWSR);
    }
    return -1;
  }

  status = TWSR;
  if ((status & 0xf8) != expected_status) {
    if (verbose) {
      i2c_error(s_i2c_data_rx_error, TWCR, status);
    }
    return -1;
  }

  b = TWDR;

  *data = b;

  return 0;
}
```

LCD Code

```
/*
* lcd.c
*
* Author: Max Billingsley
* Adapted by: Greg Beckham

*/
/*
* LCD_PORT1  = RS
* LCD_PORT2  = R/W
* LCD_PORT3  = EN
* LCD_PORT4  = DB4
* LCD_PORT5  = DB5
* LCD_PORT6 = DB6
* LCD_PORT7  = DB7
*
* RS: Register Select:
*
* 0 - Command Register
* 1 - Data Register
*
*/
#include "lcd.h"
```

```
/* entry point */

void lcd_init(void)
{
lcd_send_command(0x83);
lcd_send_command(0x83);
lcd_send_command(0x83);
lcd_send_command(0x82);
lcd_send_command(0x82);
lcd_send_command(0x8c);
lcd_send_command(0x80);
lcd_send_command(0x0f);//0f
lcd_send_command(0x00);
lcd_send_command(0x01);
}

void lcd_set_ddr(void)
{
LCD_DDR = 0xff;
}

void lcd_delay(void)
{
uint16_t i;
for(i = 0;i<2000;i++){}
}

void lcd_send_str(char *s)
{
while (*s) lcd_send_byte(*s++);
}

void lcd_send_byte(uint8_t val)
{
uint8_t temp = val;
val &= 0xf0;
val |= 0x02;
LCD_PORT = val;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
temp <<= 4;
temp |= 0x02;
LCD_PORT = temp;
lcd_delay();

LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
}

void lcd_send_command(uint8_t val)
{
uint8_t temp = val;
val &= 0xf0;
LCD_PORT = val;
lcd_delay();
LCD_PORT |= ENABLE;
```

```
LCD_PORT &= ~ENABLE;
temp <<= 4;
LCD_PORT = temp;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
lcd_delay();
}
```

Srfo8 code

```
/*
 * $Id: srf08.c,v 1.1 2003/11/15 22:42:54 bsd Exp $
 *
 */

#include <avr/twi.h>

#include <inttypes.h>

#include "i2c.h"
#include "srf08.h"


/*
 * srf08_ping - initiate a ping to the SRF08 module.  Returns
 * immidiately, the caller must then wait the appropriate amount of
 * time before reading the echo results.  Use 'srf08_range()' to read
 * the echo results.  'device' is the 7 bit I2C address of the SRF08
 * module.  'cmd' is the command to execute.
 *
 * Returns 0 on success, or -1 if an I2C error occured.  */
int8_t srf08_ping(uint8_t device, uint8_t cmd)
{
  /* start condition */
  if (i2c_start(0x08, 1))
    return -1;

  /* address slave device, write */
  if (i2c_sla_rw(device, 0, TW_MT_SLA_ACK, 1))
    return -2;

  /* write address */
  if (i2c_data_tx(0x00, TW_MT_DATA_ACK, 1))
    return -3;

  /* write command */
  if (i2c_data_tx(cmd, TW_MT_DATA_ACK, 1))
    return -4;

  if (i2c_stop())
    return -5;

  return 0;
}
```

```
/*
 * srf08_range - Read echo results from the SRF08 module.
 *
 * 'device' should be the 7 bit I2C address of the unit.  'echo'
 * should contain which echo to read, 0=first echo, 1=second echo,
 * etc.  'range' should point to a 2 byte location to hold the 16 bit
 * echo result.
 *
 * Returns 0 on success, or -1 if an I2C error occured.
 */
int8_t srf08_range(uint8_t device, uint8_t echo, uint16_t * range)
{
  uint8_t vh, vl;
  uint8_t addr;

  /* start condition */
  if (i2c_start(0x08, 1))
    return -1;

  /* address slave device, write */
  if (i2c_sla_rw(device, 0, TW_MT_SLA_ACK, 1))
    return -2;

  addr = echo*2 + 2;

  /* write address */
  if (i2c_data_tx(addr, TW_MT_DATA_ACK, 1))
    return -3;

  /* repeated start condition */
  if (i2c_start(0x10, 1))
    return -4;

   /* address slave device, read */
  if (i2c_sla_rw(device, 1, TW_MR_SLA_ACK, 1))
    return -5;

  /* read high byte */
  if (i2c_data_rx(&vh, I2C_ACK, TW_MR_DATA_ACK, 1))
    return -6;

  /* read low byte */
  if (i2c_data_rx(&vl, I2C_NACK, TW_MR_DATA_NACK, 1))
    return -7;

  if (i2c_stop())
    return -8;

  *range = (vh << 8) | vl;

  return 0;
}
```