

Automated Dog Walker
Rajesh Verma
IMDL
Summer 2005

Table of Contents

| | |
|---------------------------------------|----|
| Table of Contents | 1 |
| Abstract | 2 |
| Executive Summary | 3 |
| Introduction..... | 3 |
| Integrated System..... | 4 |
| Block Diagram | 4 |
| | 5 |
| Flow Chart | 5 |
| Power | 6 |
| Mobile Platform | 7 |
| Actuation..... | 7 |
| Sensors | 7 |
| CMUcam..... | 7 |
| Ultrasonic Range Finder | 8 |
| Potentiometer | 8 |
| Bump Switch..... | 9 |
| Behaviors | 9 |
| Following | 9 |
| Avoidance | 9 |
| Experimental Layout and Results | 9 |
| Conclusion | 10 |
| Documentation..... | 10 |
| Appendices..... | 10 |

Abstract

The Automated Dog Walker is a prototype robot that is meant to take a dog outside for a walk around the block and return it home without any assistance by humans. It allows a dog to walk only on sidewalk and avoids people, other dogs, fire hydrants, poles, or anything that the dog will be interested with. The dog will be taken out of the driveway, onto the sidewalk, around the block, and back onto the driveway.

Executive Summary

The Automated Dog Walker is a three wheeled autonomous robot. It is based on the Atmel Atmega128 processor. It uses a CMUcam to track the color of the surface it is over. If it is over a white surface, it is considered to be a sidewalk and should passively follow the dog. If any other color is detected, such as green for grass, the dog is pulled back and is prevented from getting on such surfaces.

A potentiometer and a bump switch are used to track where the dog is going and tells which direction the robot should go. Once the bump switch is triggered, the potentiometer detects which direction to turn and the robot goes in that direction until the bump switch is deactivated.

An ultrasonic range finder is used to detect other dogs, people, and any other obstacles that the dog might approach. If any obstacles are found, the dog is pulled back and prevented from getting any closer.

Introduction

This robot is meant to do the boring and often neglected task of taking your dog out for exercise. The benefits of the robot are healthier and longer lives of man's best friend.

The robot can also be used when a dog is being hyperactive and nobody has time to play with it.

For safety of the dog, the robot will not cross roads and will be limited to walking around the block on a sidewalk. It will use a camera mounted on the leash to detect if the dog is walking on sidewalk, asphalt, brick, grass, or dirt. If the dog tries to walk anywhere except sidewalk, the robot will pull back on the dog and possibly give a shock if the dog does not cooperate.

For safety of people, cats, other dogs, and anything else that the dog could try to approach, an ultrasonic sensor will be used to prevent the dog from getting too close to these obstacles.

Integrated System

Block Diagram

The Figure 1 shows a block diagram of the system. The sensors are shown on the left and the outputs are shown on the right. The CMUcam locates the sidewalk and make sure that the dog is over it. The Range Finder looks for obstacles and helps the dog avoid them. The bump switch and the potentiometer are used as steering inputs.

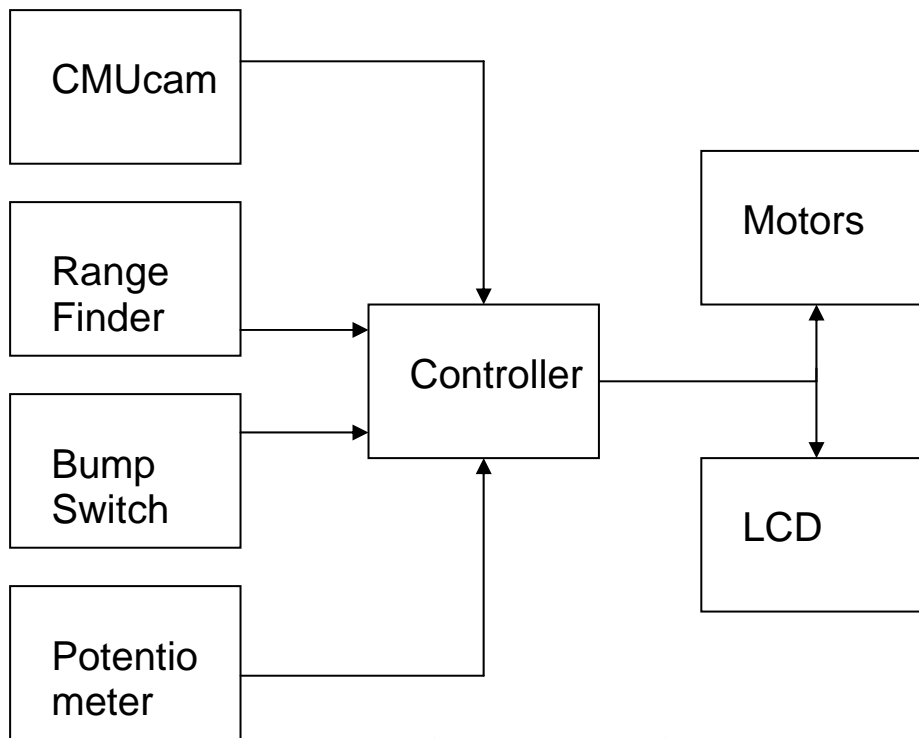


Figure 1 - Block Diagram

Flow Chart

Figure 2 shows a flow chart for the robot. The robot starts by waiting on the dog. It makes sure it is not over any surface other than sidewalk. It also makes sure that it is not too close to any other objects. It then checks which way the dog is trying to move and follows it in that direction.

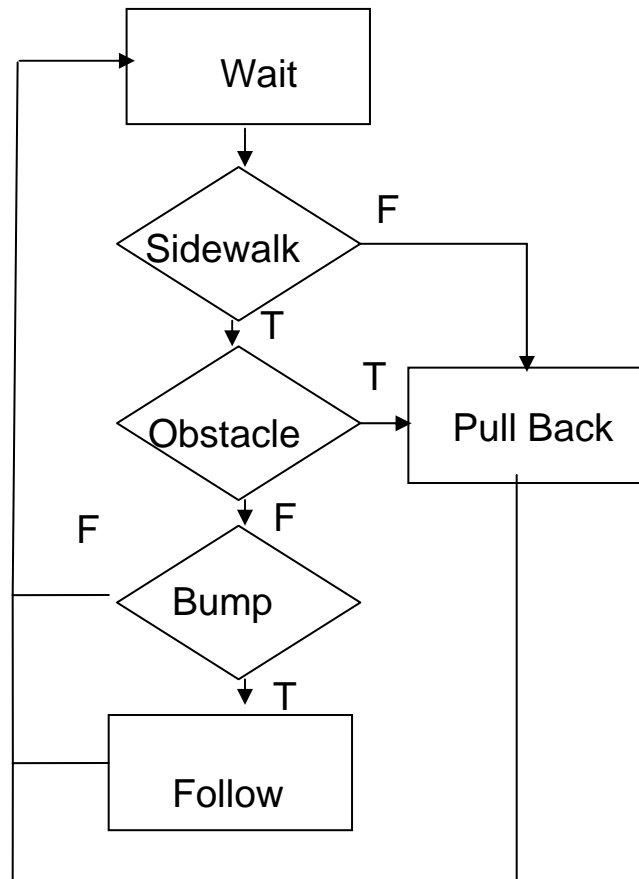


Figure 2 – Flow chart

Power

Since high-torque DC motors are used for this project, the robot requires eight Ni-CD batteries. These are preferred because they can supply the current required for the motors and are rechargeable.

Mobile Platform

Since the Automated Dog Walker is only a prototype and walks robotic dogs, a simple and small platform is made out of balsa wood. There are no requirements for anything special, so AutoCAD is not needed for the design of the platform.

Actuation

High-torque DC motors were used to move the robot. Motors were chosen over servos because they provide more torque and were cheaper. The motors chosen had the highest torque available to be on the safe side. The only downside of choosing high torque motors is you are trading off from speed.

Sensors

The sensors for the robot include a CMUcam, Ultrasonic Range Finder, Bump Sensor, and Potentiometer.

CMUcam

The CMUcam is the special sensor for the Automated Dog Walker. The CMUcam is a camera that is able to take a picture and monitor which colors are present and their relative strengths. This will be used to distinguish between sidewalk, asphalt, grass, dirt, and brick. If the dog walks over anything other than a sidewalk, the dog will be pulled back.

The camera is operated in polled mode, which returns only one frame operation per request, allowing the microprocessor to be free to do other things instead of handling

serial data. Auto White Balance was turned on and Auto gain was turned off. The image is lighted with white LED's to minimize shadows and keep consistent readings in various environments .

Ultrasonic Range Finder

An Ultrasonic Range finder will help detection of obstacles. It works by sending a pulse and waits to hear the echo and determines the distance to an object. To use the range finder, the hello world program was modified from dbmicro. The hello world program blinked an LED on the MAVRIC IIB using timers. After the trigger pulse is sent out, the echo is polled every 48us. Once the echo is received, the number of 48us periods that passed determines the distance. In order to calibrate time waited into inches, a sample measurement was taken at 12 inches. Since the operation of range finder is linear, the single calibration gave fairly accurate readings for distances.

Potentiometer

The potentiometer was used as steering input for the robot. The input of the potentiometer was placed to five volts and the output was hooked to the Analog to Digital Converter. Sample code for the IR range at bdmicro was modified to allow measurements of the potentiometer.

Once the robot is started up, an analog reading is taken from the potentiometer and is considered to be straight forward for the robot. Once the steering is moved, it changes from center and determines which way to steer the robot.

Bump Switch

A bump switch is used to make the robot move forward. The bump switch is hooked up to a pull up resistor and hooked up to an available pin on the controller. If the switch is closed, then the reading will be a logic one.

Behaviors

Following

The Automated Dog Walker will wait on the dog, if the dog moves forward, it will go forward also. If the dog turns, it will turn and follow it

Avoidance

The Automated Dog Walker will avoid any obstacles, such as trees, telephone poles, and other dogs. It will also avoid surfaces other than sidewalk. If the dog approaches any of these obstacles, it will be gently pulled back from them.

Experimental Layout and Results

The Automated Dog Walker was successful in gathering data from all of its sensors and performing actions based on them. If the dog tried to walk toward grass, water, or asphalt, it was prevented from getting there. When the dog tried to move forward, the

dog walker responded by moving forward too. If the dog went around a corner, it turned the correct direction and followed the dog.

Conclusion

The Automated Dog Walker is completely different from what was originally envisioned. Instead of walking a real dog, a robotic dog was used. Doing this was very difficult and time consuming. I found that interfacing with sensors was harder than I thought it would be. Even though other people have used the exact same sensor you have used, Their code might not work for you. Once all the sensors were being read, it was still difficult to perform the behaviors that you intended to do.

Documentation

www.seattlerobotics.com

www.hobbyengineering.com

Appendices

```
/******  
*Analog to Digital Convert Routine  
*modified from bdmicro's sample for gp2d12 IR Sensor  
*****/  
#include <avr/io.h>  
#include <stdio.h>  
  
/*  
* adc_init() - initialize A/D converter  
*  
* Initialize A/D converter to free running, start conversion, use  
* internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming  
* 16 MHz MCU clock)  
*/  
void adc_init(void)  
{  
/* configure ADC port (PORTF) as input */  
DDRF = 0x00;  
PORTF = 0x00;  
  
ADMUX = _BV(REFS0);
```

```
ADCSR = _BV(ADEN)|_BV(ADSC)|_BV(ADFR) | _BV(ADPS2)|_BV(ADPS1)|_BV(ADPS0);
}
```

```
/*
 * adc_chsel() - A/D Channel Select
 *
 * Select the specified A/D channel for the next conversion
 */
void adc_chsel(uint8_t channel)
{
    /* select channel */
    ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}

```

```
/*
 * adc_wait() - A/D Wait for conversion
 *
 * Wait for conversion complete.
 */
void adc_wait(void)
{
    /* wait for last conversion to complete */
    while ((ADCSR & _BV(ADIF)) == 0)
        ;
}

```

```
/*
 * adc_start() - A/D start conversion
 *
 * Start an A/D conversion on the selected channel
 */
void adc_start(void)
{
    /* clear conversion, start another conversion */
    ADCSR |= _BV(ADIF);
}

```

```
/*
 * adc_read() - A/D Converter - read channel
 *
 * Read the currently selected A/D Converter channel.
 */
uint16_t adc_read(void)
{
    return ADC;
}

```

```
/*
 * adc_readn() - A/D Converter, read multiple times and average
 *
 * Read the specified A/D channel 'n' times and return the average of

```

```

* the samples
*/
uint16_t adc_readn(uint8_t channel, uint8_t n)
{
    uint16_t t;
    uint8_t i;

    adc_chsel(channel);
    adc_start();
    adc_wait();

    adc_start();

    /* sample selected channel n times, take the average */
    t = 0;
    for (i=0; i<n; i++) {
        adc_wait();
        t += adc_read();
        adc_start();
    }

    /* return the average of n samples */
    return t / n;
}

/*
 * $Id: adc.h,v 1.1 2003/12/11 01:35:00 bsd Exp $
 */

#ifdef __adc_h__
#define __adc_h__

void  adc_init(void);

void  adc_chsel(uint8_t channel);

void  adc_wait(void);

void  adc_start(void);

uint16_t adc_read(void);

uint16_t adc_readn(uint8_t channel, uint8_t n);

#endif
////////////////////////////////////
CMUcam functions
/
=====
// CMUCam Functions
=====
//
=====
// originally Julio's code, modified and for individual needs by Fernando and myself

```

```

volatile int MAX_MSG_SIZE = 30;

#include <avr/io.h>

// initialize UART1 to 38.4k baud rate
void UART0_init(void)
{
    UBRROH = 0x00;
    UBRROL = 0x08;//UBRR0L = 0x33;
    UCSR0A |= 0x02;
    UCSR0B = 0x18;
    UCSR0C = 0x06;
}

// transmit a message (char array) over UART1
void USART0_Transmit(char data[MAX_MSG_SIZE])
{
    int t = 0;
    while ((t < (MAX_MSG_SIZE + 1)) & (data[t] != 0x00))
    {
        // wait for empty transmit buffer
        while ((UCSR0A & _BV(UDRE0)) == 0);
        UDR0 = data[t];
        t++;
    }
}

// wait for data to be received and then returns it
// NOTE: this is a BLOCKING receive!
unsigned char USART0_Receive( void )
{
    while ( !(UCSR0A & (1<<RXC0)) );
    return UDR0;
}

void CMU_init(void)
{
    USART0_Transmit("RS\r"); // reset
    ms_sleep(20);
    USART0_Transmit("PM 1\r"); // poll mode
    ms_sleep(20);
    USART0_Transmit("CR 18 44 17 2 19 32\r"); // RGB Auto WB on,17fps,auto gain of
    ms_sleep(20);
    USART0_Transmit("RM 3\r"); // raw output
    ms_sleep(20);
    //USART0_Transmit("SW 1 75 80 143 \r"); // use bottom half for tracking
    ms_sleep(20);
}

////////////////////////////////////
LCD functons
/*****

```

Title : HD44780U LCD library
 Author: Peter Fleury <pfleury@gmx.ch> <http://jump.to/fleury>
 File: \$Id: lcd.c,v 1.13.2.5 2005/02/16 19:15:13 Peter Exp \$\br/>
 Software: AVR-GCC 3.3
 Target: any AVR device, memory mapped mode only for AT90S4414/8515/Mega

DESCRIPTION

Basic routines for interfacing a HD44780U-based text lcd display

Originally based on Volker Oth's lcd library,
 changed lcd_init(), added additional constants for lcd_command(),
 added 4-bit I/O mode, improved and optimized code.

Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
 4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.

Memory mapped mode compatible with Kanda STK200, but supports also
 generation of R/W signal through A8 address line.

USAGE

See the C include lcd.h file for a description of each function

```

*****/
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"

/*
** constants/macros
*/
#define DDR(x) (*(&x - 1)) /* address of data direction register of port x */
#if defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
/* on ATmega64/128 PINF is on port 0x00 and not 0x60 */
#define PIN(x) ( &PORTF==&(x) ? _SFR_IO8(0x00) : (*(&x - 2)) )
#else
#define PIN(x) (*(&x - 2)) /* address of input register of port x */
#endif

#if LCD_IO_MODE
#define lcd_e_delay() __asm__ __volatile__( "rjmp 1f\n 1:" );
#define lcd_e_high() LCD_E_PORT |= _BV(LCD_E_PIN);
#define lcd_e_low() LCD_E_PORT &= ~_BV(LCD_E_PIN);
#define lcd_e_toggle() toggle_e()
#define lcd_rw_high() LCD_RW_PORT |= _BV(LCD_RW_PIN)
#define lcd_rw_low() LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
#define lcd_rs_high() LCD_RS_PORT |= _BV(LCD_RS_PIN)
#define lcd_rs_low() LCD_RS_PORT &= ~_BV(LCD_RS_PIN)
#endif

#if LCD_IO_MODE
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT LCD_FUNCTION_4BIT_1LINE

```

```

#else
#define LCD_FUNCTION_DEFAULT LCD_FUNCTION_4BIT_2LINES
#endif
#else
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT LCD_FUNCTION_8BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT LCD_FUNCTION_8BIT_2LINES
#endif
#endif

/*
** function prototypes
*/
#if LCD_IO_MODE
static void toggle_e(void);
#endif

/*
** local functions
*/

/*****
delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
*****/
static inline void _delayFourCycles(unsigned int __count)
{
    if ( __count == 0 )
        __asm__ __volatile__( "rjmp lf\n 1:" ); // 2 cycles
    else
        __asm__ __volatile__(
            "1: sbiw %0,1" "\n\t"
            "brne 1b" // 4 cycles/loop
            : "=w" (__count)
            : "0" (__count)
            );
}

/*****
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock frequency
*****/
#define delay(us) _delayFourCycles( ( ( 1*(XTAL/4000) ) *us) / 1000 )

#if LCD_IO_MODE
/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
    lcd_e_high();
    lcd_e_delay();
    lcd_e_low();
}

```

```

}
#endif

/*****
Low-level function to write byte to LCD controller
Input:  data  byte to write to LCD
        rs    1: write data
        0: write instruction
Returns: none
*****/
#ifdef LCD_IO_MODE
static void lcd_write(uint8_t data,uint8_t rs)
{
    unsigned char dataBits ;

    if (rs) { /* write data    (RS=1, RW=0) */
        lcd_rs_high();
    } else { /* write instruction (RS=0, RW=0) */
        lcd_rs_low();
    }
    lcd_rw_low();

    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
        && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN == 2) &&
(LCD_DATA3_PIN == 3) )
    {
        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= 0x0F;

        /* output high nibble first */
        dataBits = LCD_DATA0_PORT & 0xF0;
        LCD_DATA0_PORT = dataBits | ((data>>4)&0x0F);
        lcd_e_toggle();

        /* output low nibble */
        LCD_DATA0_PORT = dataBits | (data&0x0F);
        lcd_e_toggle();

        /* all data pins high (inactive) */
        LCD_DATA0_PORT = dataBits | 0x0F;
    }
}
else
{
    /* configure data pins as output */
    DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
    DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
    DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
    DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

    /* output high nibble first */
    LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
    LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
    LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
}
}

```

```

LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
  if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
  if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
  if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
  if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
  lcd_e_toggle();

  /* output low nibble */
  LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
  LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
  LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
  LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
  if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
  if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
  if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
  if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
  lcd_e_toggle();

  /* all data pins high (inactive) */
  LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
  LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
  LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
  LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
}
}
#else
#define lcd_write(d,rs) if (rs) *(volatile uint8_t*)(LCD_IO_DATA) = d; else *(volatile
uint8_t*)(LCD_IO_FUNCTION) = d;
/* rs==0 -> write instruction to LCD_IO_FUNCTION */
/* rs==1 -> write data to LCD_IO_DATA */
#endif

/*****
Low-level function to read byte from LCD controller
Input:  rs   1: read data
        0: read busy flag / address counter
Returns: byte read from LCD controller
*****/
#if LCD_IO_MODE
static uint8_t lcd_read(uint8_t rs)
{
  uint8_t data;

  if (rs)
    lcd_rs_high();          /* RS=1: read data   */
  else
    lcd_rs_low();           /* RS=0: read busy flag */
  lcd_rw_high();           /* RW=1 read mode     */

  if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
    && ( LCD_DATA0_PIN == 0 ) && ( LCD_DATA1_PIN == 1 ) && ( LCD_DATA2_PIN == 2 ) &&
( LCD_DATA3_PIN == 3 ) )
  {

```

```

    DDR(LCD_DATA0_PORT) &= 0xF0;    /* configure data pins as input */

    lcd_e_high();
    lcd_e_delay();
    data = PIN(LCD_DATA0_PORT) << 4; /* read high nibble first */
    lcd_e_low();

    lcd_e_delay();          /* Enable 500ns low    */

    lcd_e_high();
    lcd_e_delay();
    data |= PIN(LCD_DATA0_PORT)&0x0F; /* read low nibble    */
    lcd_e_low();
}
else
{
    /* configure data pins as input */
    DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
    DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
    DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
    DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

    /* read high nibble first */
    lcd_e_high();
    lcd_e_delay();
    data = 0;
    if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
    if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
    if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
    if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
    lcd_e_low();

    lcd_e_delay();          /* Enable 500ns low    */

    /* read low nibble */
    lcd_e_high();
    lcd_e_delay();
    if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
    if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
    if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
    if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
    lcd_e_low();
}
return data;
}
#else
#define lcd_read(rs) (rs) ? *(volatile uint8_t*)(LCD_IO_DATA+LCD_IO_READ) : *(volatile
uint8_t*)(LCD_IO_FUNCTION+LCD_IO_READ)
/* rs==0 -> read instruction from LCD_IO_FUNCTION */
/* rs==1 -> read data from LCD_IO_DATA */
#endif

/*****
loops while lcd is busy, returns address counter
*****/

```

```

static uint8_t lcd_waitbusy(void)

{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY) ) {}

    /* the address counter is updated 4us after the busy flag is cleared */
    delay(2);

    /* now read the address counter */
    return (lcd_read(0)); // return address counter

}/* lcd_waitbusy */

/*****
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
*****/
static inline void lcd_newline(uint8_t pos)
{
    register uint8_t addressCounter;

#if LCD_LINES==1
    addressCounter = 0;
#endif
#if LCD_LINES==2
    if ( pos < (LCD_START_LINE2) )
        addressCounter = LCD_START_LINE2;
    else
        addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
    if ( pos < LCD_START_LINE3 )
        addressCounter = LCD_START_LINE2;
    else if ( ( pos >= LCD_START_LINE2 ) && ( pos < LCD_START_LINE4 ) )
        addressCounter = LCD_START_LINE3;
    else if ( ( pos >= LCD_START_LINE3 ) && ( pos < LCD_START_LINE2 ) )
        addressCounter = LCD_START_LINE4;
    else
        addressCounter = LCD_START_LINE1;
#endif
    lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */

/*
** PUBLIC FUNCTIONS
*/

/*****
Send LCD controller instruction command

```

Input: instruction to send to LCD controller, see HD44780 data sheet

Returns: none

*****/

```
void lcd_command(uint8_t cmd)
```

```
{
    lcd_waitbusy();
    lcd_write(cmd,0);
}
```

Send data byte to LCD controller

Input: data to send to LCD controller, see HD44780 data sheet

Returns: none

*****/

```
void lcd_data(uint8_t data)
```

```
{
    lcd_waitbusy();
    lcd_write(data,1);
}
```

Set cursor to specified position

Input: x horizontal position (0: left most position)

y vertical position (0: first line)

Returns: none

*****/

```
void lcd_gotoxy(uint8_t x, uint8_t y)
```

```
{
#ifdef LCD_LINES==1
    lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#ifdef LCD_LINES==2
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
#endif
#ifdef LCD_LINES==4
    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else if ( y==1 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
    else if ( y==2 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
    else /* y==3 */
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif
}
```

```
}/* lcd_gotoxy */
```

*****/

```

int lcd_getxy(void)
{
    return lcd_waitbusy();
}

/*****
Clear display and set cursor to home position
*****/
void lcd_clrscr(void)
{
    lcd_command(1<<LCD_CLR);
}

/*****
Set cursor to home position
*****/
void lcd_home(void)
{
    lcd_command(1<<LCD_HOME);
}

/*****
Display character at current cursor position
Input:  character to be displayed
Returns: none
*****/
void lcd_putc(char c)
{
    uint8_t pos;

    pos = lcd_waitbusy(); // read busy-flag and address counter
    if (c=='\n')
    {
        lcd_newline(pos);
    }
    else
    {
        #if LCD_WRAP_LINES==1
        #if LCD_LINES==1
            if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
                lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        #elif LCD_LINES==2
            if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
                lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
            else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH )
                lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
        #elif LCD_LINES==4
            if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
                lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
            else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH )
                lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
            else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH )

```

```

        lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
    else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH )
        lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
#endif
    lcd_waitbusy();
#endif
    lcd_write(c, 1);
}

}/* lcd_putc */

/*****
Display string without auto linefeed
Input:  string to be displayed
Returns: none
*****/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }
}

}/* lcd_puts */

/*****
Display string from program memory without auto linefeed
Input:  string from program memory be be displayed
Returns: none
*****/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) ) {
        lcd_putc(c);
    }
}

}/* lcd_puts_p */

/*****
Initialize display and select type of cursor
Input:  dispAttr LCD_DISP_OFF      display off
        LCD_DISP_ON      display on, cursor off
        LCD_DISP_ON_CURSOR  display on, cursor on
        LCD_DISP_CURSOR_BLINK  display on, cursor on flashing
Returns: none
*****/
void lcd_init(uint8_t dispAttr)
{

```

```

#if LCD_IO_MODE
/*
 * Initialize LCD to 4 bit I/O mode
 */

if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
    && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT ==
&LCD_DATA0_PORT) && ( &LCD_E_PORT == &LCD_DATA0_PORT)
    && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN == 2) &&
(LCD_DATA3_PIN == 3)
    && (LCD_RS_PIN == 4) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6) )
{
    /* configure all port bits as output (all LCD lines on same port) */
    DDR(LCD_DATA0_PORT) |= 0x7F;
}
else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT ==
&LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
    && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN == 2) &&
(LCD_DATA3_PIN == 3) )
{
    /* configure all port bits as output (all LCD data lines on same port, but control lines on different ports)
*/
    DDR(LCD_DATA0_PORT) |= 0x0F;
    DDR(LCD_RS_PORT)  |= _BV(LCD_RS_PIN);
    DDR(LCD_RW_PORT)  |= _BV(LCD_RW_PIN);
    DDR(LCD_E_PORT)   |= _BV(LCD_E_PIN);
}
else
{
    /* configure all port bits as output (LCD data and control lines on different ports */
    DDR(LCD_RS_PORT)  |= _BV(LCD_RS_PIN);
    DDR(LCD_RW_PORT)  |= _BV(LCD_RW_PIN);
    DDR(LCD_E_PORT)   |= _BV(LCD_E_PIN);
    DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
    DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
    DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
    DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
}
delay(16000); /* wait 16ms or more after power-on */

/* initial write to lcd is 8bit */
LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN); // _BV(LCD_FUNCTION)>>4;
LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN); // _BV(LCD_FUNCTION_8BIT)>>4;
lcd_e_toggle();
delay(4992); /* delay, busy flag can't be checked here */

/* repeat last command */
lcd_e_toggle();
delay(64); /* delay, busy flag can't be checked here */

/* repeat last command a third time */
lcd_e_toggle();
delay(64); /* delay, busy flag can't be checked here */

/* now configure for 4bit mode */

```

```

LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN); // LCD_FUNCTION_4BIT_1LINE>>4
lcd_e_toggle();
delay(64);      /* some displays need this additional delay */

/* from now the LCD only accepts 4 bit I/O, we can use lcd_command() */
#else
/*
 * Initialize LCD to 8 bit memory mapped mode
 */

/* enable external SRAM (memory mapped lcd) and one wait state */
MCUCR = _BV(SRE) | _BV(SRW);

/* reset LCD */
delay(16000);      /* wait 16ms after power-on */
lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit interface */
delay(4992);      /* wait 5ms */
lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit interface */
delay(64);        /* wait 64us */
lcd_write(LCD_FUNCTION_8BIT_1LINE,0); /* function set: 8bit interface */
delay(64);        /* wait 64us */
#endif
lcd_command(LCD_FUNCTION_DEFAULT); /* function set: display lines */
lcd_command(LCD_DISP_OFF);        /* display off */
lcd_clrscr();                      /* display clear */
lcd_command(LCD_MODE_DEFAULT);    /* set entry mode */
lcd_command(dispattr);            /* display/cursor control */

}/* lcd_init */
#endif LCD_H
#define LCD_H
/*****
Title   : C include file for the HD44780U LCD library (lcd.c)
Author:  Peter Fleury <pfleury@gmx.ch> http://jump.to/fleury
File:    $Id: lcd.h,v 1.12.2.4 2005/02/28 22:54:41 Peter Exp $
Software: AVR-GCC 3.3
Hardware: any AVR device, memory mapped mode only for AT90S4414/8515/Mega
*****/

/**
 @defgroup pfleury_lcd LCD library
 @code #include <lcd.h> @endcode

 @brief Basic routines for interfacing a HD44780U-based text LCD display

Originally based on Volker Oth's LCD library,
changed lcd_init(), added additional constants for lcd_command(),
added 4-bit I/O mode, improved and optimized code.

Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.

Memory mapped mode compatible with Kanda STK200, but supports also
generation of R/W signal through A8 address line.

@author Peter Fleury pfleury@gmx.ch http://jump.to/fleury

```

@see The chapter Interfacing a HD44780 Based LCD to an AVR
on my home page.

```

*/

/*@{*/

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
#error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC compiler !"
#endif

#include <inttypes.h>
#include <avr/pgmspace.h>

/**
 * @name Definitions for MCU Clock Frequency
 * Adapt the MCU clock frequency in Hz to your target.
 */
#define XTAL 4000000      /**< clock frequency in Hz, used to calculate delay timer */

/**
 * @name Definitions for Display Size
 * Change these definitions to adapt setting to your display
 */
#define LCD_LINES      4      /**< number of visible lines of the display */
#define LCD_DISP_LENGTH 20    /**< visibles characters per line of the display */
#define LCD_LINE_LENGTH 0x40  /**< internal line length of the display */
#define LCD_START_LINE1 0x00  /**< DDRAM address of first char of line 1 */
#define LCD_START_LINE2 0x40  /**< DDRAM address of first char of line 2 */
#define LCD_START_LINE3 0x14  /**< DDRAM address of first char of line 3 */
#define LCD_START_LINE4 0x54  /**< DDRAM address of first char of line 4 */
#define LCD_WRAP_LINES 0      /**< 0: no wrap, 1: wrap at end of visibile line */

#define LCD_IO_MODE    1      /**< 0: memory mapped mode, 1: IO port mode */
#if LCD_IO_MODE
/**
 * @name Definitions for 4-bit IO mode
 * Change LCD_PORT if you want to use a different port for the LCD pins.
 *
 * The four LCD data lines and the three control lines RS, RW, E can be on the
 * same port or on different ports.
 * Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control lines on
 * different ports.
 *
 * Normally the four data lines should be mapped to bit 0..3 on one port, but it
 * is possible to connect these data lines in different order or even on different
 * ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.
 */
#define LCD_PORT      PORTA      /**< port for the LCD lines */
#define LCD_DATA0_PORT LCD_PORT  /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT LCD_PORT  /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT LCD_PORT  /**< port for 4bit data bit 2 */

```

```

#define LCD_DATA3_PORT LCD_PORT /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN 4 /**< pin for 4bit data bit 0 */
#define LCD_DATA1_PIN 5 /**< pin for 4bit data bit 1 */
#define LCD_DATA2_PIN 6 /**< pin for 4bit data bit 2 */
#define LCD_DATA3_PIN 7 /**< pin for 4bit data bit 3 */
#define LCD_RS_PORT LCD_PORT /**< port for RS line */
#define LCD_RS_PIN 0 /**< pin for RS line */
#define LCD_RW_PORT LCD_PORT /**< port for RW line */
#define LCD_RW_PIN 1 /**< pin for RW line */
#define LCD_E_PORT LCD_PORT /**< port for Enable line */
#define LCD_E_PIN 2 /**< pin for Enable line */

#elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) || defined(__AVR_ATmega64__)
|| \
defined(__AVR_ATmega8515__) || defined(__AVR_ATmega103__) || defined(__AVR_ATmega128__)
|| \
defined(__AVR_ATmega161__) || defined(__AVR_ATmega162__)
/*
* memory mapped mode is only supported when the device has an external data memory interface
*/
#define LCD_IO_DATA 0xC000 /* A15=E=1, A14=RS=1 */
#define LCD_IO_FUNCTION 0x8000 /* A15=E=1, A14=RS=0 */
#define LCD_IO_READ 0x0100 /* A8 =R/W=1 (R/W: 1=Read, 0=Write */
#else
#error "external data memory interface not available for this device, use 4-bit IO port mode"

#endif

/**
* @name Definitions for LCD command instructions
* The constants define the various LCD controller instructions which can be passed to the
* function lcd_command(), see HD44780 data sheet for a complete description.
*/

/* instruction register bit positions, see HD44780U data sheet */
#define LCD_CLR 0 /* DB0: clear display */
#define LCD_HOME 1 /* DB1: return to home position */
#define LCD_ENTRY_MODE 2 /* DB2: set entry mode */
#define LCD_ENTRY_INC 1 /* DB1: 1=increment, 0=decrement */
#define LCD_ENTRY_SHIFT 0 /* DB2: 1=display shift on */
#define LCD_ON 3 /* DB3: turn lcd/cursor on */
#define LCD_ON_DISPLAY 2 /* DB2: turn display on */
#define LCD_ON_CURSOR 1 /* DB1: turn cursor on */
#define LCD_ON_BLINK 0 /* DB0: blinking cursor ? */
#define LCD_MOVE 4 /* DB4: move cursor/display */
#define LCD_MOVE_DISP 3 /* DB3: move display (0-> cursor) ? */
#define LCD_MOVE_RIGHT 2 /* DB2: move right (0-> left) ? */
#define LCD_FUNCTION 5 /* DB5: function set */
#define LCD_FUNCTION_8BIT 4 /* DB4: set 8BIT mode (0->4BIT mode) */
#define LCD_FUNCTION_2LINES 3 /* DB3: two lines (0->one line) */
#define LCD_FUNCTION_10DOTS 2 /* DB2: 5x10 font (0->5x7 font) */
#define LCD_CGRAM 6 /* DB6: set CG RAM address */
#define LCD_DDRAM 7 /* DB7: set DD RAM address */
#define LCD_BUSY 7 /* DB7: LCD is busy */

```

```

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#define LCD_ENTRY_DEC      0x04 /* display shift off, dec cursor move dir */
#define LCD_ENTRY_DEC_SHIFT 0x05 /* display shift on, dec cursor move dir */
#define LCD_ENTRY_INC_    0x06 /* display shift off, inc cursor move dir */
#define LCD_ENTRY_INC_SHIFT 0x07 /* display shift on, inc cursor move dir */

/* display on/off, cursor on/off, blinking char at cursor position */
#define LCD_DISP_OFF      0x08 /* display off */
#define LCD_DISP_ON      0x0C /* display on, cursor off */
#define LCD_DISP_ON_BLINK 0x0D /* display on, cursor off, blink char */
#define LCD_DISP_ON_CURSOR 0x0E /* display on, cursor on */
#define LCD_DISP_ON_CURSOR_BLINK 0x0F /* display on, cursor on, blink char */

/* move cursor/shift display */
#define LCD_MOVE_CURSOR_LEFT 0x10 /* move cursor left (decrement) */
#define LCD_MOVE_CURSOR_RIGHT 0x14 /* move cursor right (increment) */
#define LCD_MOVE_DISP_LEFT 0x18 /* shift display left */
#define LCD_MOVE_DISP_RIGHT 0x1C /* shift display right */

/* function set: set interface data length and number of display lines */
#define LCD_FUNCTION_4BIT_1LINE 0x20 /* 4-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_4BIT_2LINES 0x28 /* 4-bit interface, dual line, 5x7 dots */
#define LCD_FUNCTION_8BIT_1LINE 0x30 /* 8-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_8BIT_2LINES 0x38 /* 8-bit interface, dual line, 5x7 dots */

#define LCD_MODE_DEFAULT ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC))

/**
 * @name Functions
 */

/**
 @brief Initialize display and select type of cursor
 @param dispAttr \b LCD_DISP_OFF display off\n
           \b LCD_DISP_ON display on, cursor off\n
           \b LCD_DISP_ON_CURSOR display on, cursor on\n
           \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashing
 @return none
 */
extern void lcd_init(uint8_t dispAttr);

/**
 @brief Clear display and set cursor to home position
 @param void
 @return none
 */
extern void lcd_clrscr(void);

/**
 @brief Set cursor to home position

```

```
@param void
@return none
*/
extern void lcd_home(void);

/**
@brief Set cursor to specified position

@param x horizontal position\n (0: left most position)
@param y vertical position\n (0: first line)
@return none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);

/**
@brief Display character at current cursor position
@param c character to be displayed
@return none
*/
extern void lcd_putc(char c);

/**
@brief Display string without auto linefeed
@param s string to be displayed
@return none
*/
extern void lcd_puts(const char *s);

/**
@brief Display string from program memory without auto linefeed
@param s string from program memory be be displayed
@return none
@see lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);

/**
@brief Send LCD controller instruction command
@param cmd instruction to send to LCD controller, see HD44780 data sheet
@return none
*/
extern void lcd_command(uint8_t cmd);

/**
@brief Send data byte to LCD controller

Similar to lcd_putc(), but without interpreting LF
@param data byte to send to LCD controller, see HD44780 data sheet
@return none
*/
```

```

extern void lcd_data(uint8_t data);

/**
 * @brief macros for automatically storing string constant in program memory
 */
#define lcd_puts_P(__s)    lcd_puts_p(PSTR(__s))

/* @} */
#endif //LCD_H
/////////////////////////////////////////////////////////////////
Sonar functions
/*****
Sonic range finder function obtained from Fernando Hernandez
A modification of hello world sample code from bdmicro

*****/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>

// Global vars
volatile uint16_t ms_count;
volatile uint16_t us_count;

// SRF04 Ultrasonic =====
// Port Pin           0       1       2       3       4       5       6       7
// LCD Pin            OUT TRG
//      --#1--
#define SRF_PORT      PORTD
#define SRF_DDRX      DDRD
#define SRF_PINX      PIND

// Delay for a specified number of milliseconds
// Note: will not work for more than about 3100ms because of interger overflow!
// 65536/21 = 3120ms max
void ms_sleep(uint16_t ms)
{
    TCNT0 = 0;
    ms_count = 0;
    while (ms_count != ms*21);
}

// Delay for specified number of microseconds * 48
void us_sleep(uint16_t us)
{
    TCNT0 = 0;
    us_count = 0;
    while (us_count != us);
}

```

```

}

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

// initialize timer 0 to generate an interrupt every 48usec
void init_timer(void)
{
    TIFR |= _BV(OCIE0);
    TCCR0 = _BV(WGM01)|_BV(CS02)|_BV(CS00); // CTC, prescale = 128
    TCNT0 = 0;
    TIMSK |= _BV(OCIE0); // enable output compare interrupt
    OCR0 = 3; // match in 48usec
}

//
=====
=====
// SRF04 Functions
=====
//
=====
=====

// Function to get distance data from an SRF04, first
// argument specifies which one. ( 1 = left, 2 = right )
int dist(int times)
{
    int i,counter = 0;

    for (i=0; i<times ;i++)
    {
        ms_sleep(50); // wait 50msec between pulses for echo to settle

        // Trigger a ping on coresponding SRF04 and wait a few usec
        SRF_DDRX = 0x02; // set output pin
        SRF_PORT = 0x02; // trigger
        us_sleep(8); // wait a few usec
        SRF_PORT = 0x00; // end trigger

        // wait a few usec for echo circuit initialization
        us_sleep(8);

        while(1)
        {
            if(SRF_PINX & 0x01){ // check if echo has been received
                counter++; // increment counter
                us_sleep(4);} // wait a few usec
            else break; // break if echo was received
        }
    }
}

```

```

    }
}
return ((counter/times)*1.8);           // return average (calibrated for inches)
}

```

```

/////////////////////////////////////////////////////////////////

```

Main function

```

/*****

```

Title: testing output to a HD44780 based LCD display.

Author: Peter Fleury <pfleury@gmx.ch> <http://jump.to/fleury>

File: \$Id: test_lcd.c,v 1.6 2004/12/10 13:53:59 peter Exp \$

Software: AVR-GCC 3.3

Hardware: HD44780 compatible LCD text display

ATS90S8515/ATmega if memory-mapped LCD interface is used

any AVR with 7 free I/O pins if 4-bit IO port mode is used

```

*****/

```

```

#include <stdlib.h>

```

```

#include <avr/io.h>

```

```

#include <avr/interrupt.h>

```

```

#include <avr/pgmspace.h>

```

```

#include "lcd.h"

```

```

#define L_MOTOR          OCR1B

```

```

#define L_DIR            0x02

```

```

#define R_DIR            0x01

```

```

#define R_MOTOR          OCR1A

```

```

int left_direction;

```

```

int right_direction;

```

```

volatile unsigned char CMUResponseBuffer[15];

```

```

int minR=60;

```

```

int minG=60;

```

```

int minB=20;

```

```

void stop_motors()

```

```

{

```

```

while (L_MOTOR !=0 && R_MOTOR !=0)

```

```

{

```

```

    /* put string to display (line 1) with linefeed */

```

```

    //lcd_puts("Stoping\n");

```

```

    if (L_MOTOR>0)

```

```

        L_MOTOR=L_MOTOR-100;

```

```

    if (R_MOTOR>0)

```

```

        R_MOTOR=R_MOTOR-100;

```

```

    if (L_MOTOR<0)

```

```
        L_MOTOR=0;
    if (R_MOTOR<0)
        R_MOTOR=0;

    ms_sleep(3);
}

}

void start_motors()
{
while (L_MOTOR !=900 && R_MOTOR !=900)
{
    if (L_MOTOR<900)
        L_MOTOR=L_MOTOR+100;

    if (R_MOTOR<900)
        R_MOTOR=R_MOTOR+100;

    if (L_MOTOR>900)
        L_MOTOR=900;
    if (R_MOTOR>900)
        R_MOTOR=900;

    ms_sleep(3);
}
}

void turn_left()
{
PORTB = L_DIR | R_DIR;// right forward left back
lcd_puts(" L ");
start_motors();
}

void turn_right()
{
PORTB = 0;// left forward right back
lcd_puts(" R ");
start_motors();
}

void go_forward()
{
PORTB = R_DIR;//both forward
lcd_puts(" F ");
start_motors();
}
```

```
void go_back()
{
PORTB = L_DIR;//both back
lcd_puts(" B ");
start_motors();
}

// queries the CMU cam to get the mean values of R, G, B
// stores them in the global "CMUResponseBuffer"
void CMU_GetMean(void)
{
    int i = 0;
    char tempChar='a';

    USART0_Transmit("GM\r");

    // initial 255 framing byte is read in, discarded
    //tempChar = USART0_Receive();
    while(tempChar != 255)
    {
tempChar = USART0_Receive();
    }

tempChar = USART0_Receive();//packet type

    // this command returns a "type S" packet, 7 bytes long
    // we read in those 7 bytes
    for(i=0;i<6;i++)
    {
        CMUResponseBuffer[i] = USART0_Receive();
    }

    /*// last 255 framing byte is read in, discarded
    while(tempChar != ':')
    {
        tempChar = USART0_Receive();
    }*/

    CMUResponseBuffer[i] = '\0';
}

int main()
{
    char buffer[7];
    int num=134;
    unsigned char i;

    int distance;
    char dist_str[33];
    char ad_str[33];
```

```

int left_motor;
int right_motor;
int left_direction;
int right_direction;

uint16_t ir;
int bump;
uint16_t midAD;

double volts;

TCCR1A = 0xA0;//set down count clear up count
TCCR1B = 0x12;// mode 8 clock/8

/*****
*PortB  7      6      5      4      3      2
1      0
*      oc1b  oc1a
L_Dir  R_Dir
*****/

DDRB = 0x63;//
ICR1 = 1000;

// initialize A/D Converter */
adc_init();

// initialize UART */
UART0_init();

USART0_Transmit("Hello World!\r");

// initialize timer
init_timer();
sei(); // enable interrupts

/* initialize display, cursor off */
lcd_init(LCD_DISP_ON);

lcd_clrscr();
/* put string to display (line 1) with linefeed */
lcd_puts("Initializing CMUcam\n");
ms_sleep(1000);
CMU_init();

```

```
lcd_puts("bump to start");

while (!PINC & 0x01)//wait for bump switch
{

}

//ms_sleep(3000);

/*turn_right();//back
ms_sleep(1000);
stop_motors();

turn_left();
ms_sleep(1000);
stop_motors();

/*go_forward();
ms_sleep(1000);
stop_motors();

go_back();
ms_sleep(1000);
stop_motors;*/

lcd_clrscr();
ms_sleep(1000);
midAD = adc_readn(2, 5);

for (;;)
{

//lcd_clrscr();

distance=dist(5);/* loop forever */
(void)itoa(distance, dist_str, 10);

ir = adc_readn(2, 5); /* sample channel 0 5 times, take average */
//ir = ir - midAD;
(void)itoa(ir, ad_str, 10);

CMU_GetMean();

lcd_clrscr();
```

```
    for (int rgb=0;rgb<3;rgb++)
    {
        (void)itoa(CMUResponseBuffer[rgb], dist_str, 10);
        lcd_putc(' ');
        lcd_puts((dist_str));
    }

    if(PINC & 0x01)
    {
        bump=1;
        lcd_puts("\nB=1");
    }
    else
    {
        bump=0;
        lcd_puts("\nB=0");
    }

    lcd_puts(" DIST:");
    lcd_puts(dist_str);

    lcd_puts(" A/D:");
    lcd_puts(ad_str);

    if (CMUResponseBuffer[0] > minR && CMUResponseBuffer[1] > minG &&
CMUResponseBuffer[2] > minB && dist>1)
    {
        lcd_puts("\n Good");

        if (bump==1 & ir< midAD+25 & ir >midAD-25)
        {
            go_forward();
            ms_sleep(1000);
            stop_motors();
        }

        else if (bump==1 & ir >midAD+25) //positive chane
        {
            turn_left();
            ms_sleep(1000);
            stop_motors();
        }

        else if (bump==1 & ir <midAD-25) //positive chane
        {
            turn_right();
            ms_sleep(1000);
            stop_motors();
        }
    }
}
```

```
    }  
    else  
    {  
    lcd_puts("\nBad!");  
    go_back();  
    ms_sleep(1000);  
    stop_motors();  
    }  
  
    ms_sleep(100);  
    }  
}
```