ButlerBot

Special Sensor Report:
SRF08 Sonar

Erik Sjolander
EEL5666: IMDL
Fall 2002
Professor: Dr. Arroyo
TAs: Jason Plew, Uriel Rodriguez

Introduction

ButlerBot was intended to roam a tabletop, looking for glasses to fill when it was first conceived. Since IR might not work well to find clear glasses, I decided to use sonar to detect them. During the course of design, roaming the table became line-following, so the main use of the sonar disappeared. In my final design, it ended up being used only for obstacle-avoidance (which it turns out to be very good at).
I decided to use the SRF08 sonar, which interfaces via the I^2C (TWI) bus, rather than the cheaper SRF04, because I thought it would be easier to use.

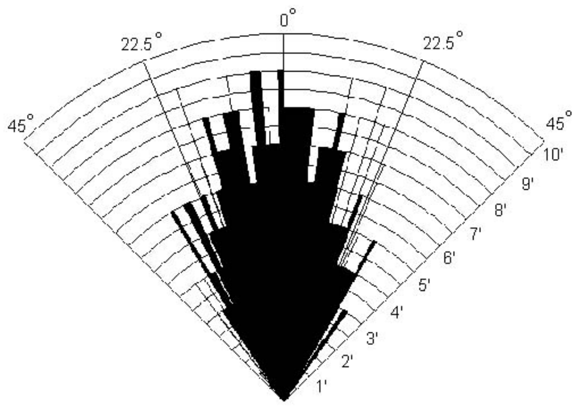
Basic Functions

The SRF08 is a sonar module with a nominal range of 2" to 18'. It connects to the host microcontroller via the I^2C bus, a common interface in embedded systems (the Atmel Mega323 has a hardware I^2C bus on-board; many other chips have software implementations of the protocol available). It has a resolution of about 1", and provides readings in inches, cm or microseconds. Basically, a sonar "ping" is started by writing 0x50 to register 0 (for results in inches). After 65ms the result is available in register locations 2-35. The results are presented as 2-byte values, with the most significant byte in the lower address. Readings are given for the distances to the 17 closest objects. It also has an on-board CdS cell, which provides a 0-255 reading of the ambient light. The sensor sees almost ±45° on either side of its centerline at short distances (<3').

Testing Results

The sonar seems to have a range somewhat longer than its stated 18'. In testing, I got readings for walls up to about 400" (33') away. When pointed at a nearby wall, it never gave a maximum reading longer than the distance to the wall (it didn't give spurious results). The sonar seems to have a jitter of about an inch, that is, a given object may bounce between two readings. The distance given for distant echoes depends partly on the positioning of nearby objects (which is reasonable, since a nearby object can force the sound to take a longer path to something farther away), but I never saw a change of more than 3-4" for objects less than 10' away (out at 33' the readings bounce around by as much as 30").

The maker of the sonar provides this test graphic of the maximum distance at which the sonar would detect a broom handle:



The sonar is not blocked or reflected from thin materials. I tested it both with a sheet of paper and a thin piece of cloth, and neither seemed to have any impact on the sonar readings given. This could allow the sonar module to be hidden, or be protected from the elements, by some suitably thin material. However, it does reliably detect people, walls, chairs, and all sorts of other obstacles.
Echoes are not received from surfaces that are almost parallel to the sound waves. This seemed to be both good and bad. The advantage is that the sonar can be mounted near the floor without getting junk readings. The disadvantage is that it can miss nearby walls, if it is traveling almost parallel to the wall. In ButlerBot I avoided this problem by having IR sensors in the cross-eyed configuration looking for obstacles on the sides of the robot.
In theory, the sonar returns the echoes of the 17 closest objects. In practice, I found that the top 2-3 locations were only used when the sonar was in an environment with lots of surfaces, and a long distance to the farthest object (e.g. pointed through my room and into the kitchen). When pointed at a nearby wall, it usually picks up about 4-8 echoes (off me, my computer, the desk, and so forth). It has to be pointed at a completely blank wall at fairly close range (about 2'-3') to return only one echo. I found the multiple echoes more useful than I expected – my robot platform caused an echo at 3" on every ping, and with this sensor I was able to ignore this reading and get only the interesting ones.

Uses

I used the sonar to do obstacle-avoidance on ButlerBot. It worked very well for this, mainly because the sonar sees everything within 45° of the front of the robot, and it will pick up even very small obstacles (e.g. the leg of a chair). The long range and linearity of the sensor, plus the fact that it gives range in inches, made it very easy to interface with the rest of my obstacle avoidance code once I got the I^{2}C bus working.
One interesting use of the sensor would be as a clear-path detector. Basically the robot would travel in the direction that returns an echo from the largest distance. This might allow the robot to navigate a clear route over large distances.
Using a sonar based on the I^{2}C bus obviously forces one to get this working correctly. Once the time has been spent to get it working, it should be very easy to add other peripherals to the bus. Among other things, A/D converters, digital port expanders,

EEPROMs and a variety of specialized parts are available from Texas Instruments (www.ti.com) and Maxim (www.maxim-ic.com).

Code

This is the C code needed to test the SRF08. It continuously pings with the sonar, and prints the results out over the serial port. Please see my main report for the UART code (uart.c/uart.h/global.h).

i2c_test.c

```
////////////////////////////////////////////////////
//(c)2002 Erik Sjolander
//This code may be used and modified for non-profit,
//educational purposes, as long as this copyright
//notice is preserved.  Any other use requires my
//permission.
//
//i2c_test.c
//Tests the Devantech SRF08 I2C sonar module.
////////////////////////////////////////////////////

#include <io.h>
#include <interrupt.h>
#include "global.h"
#include "uart.h"
#include "sonar.h"


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


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

    //u08 data[36]; //sonar data
    u16 ranges[17];
    u08 light_sensor=0;
    char out[64];

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

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

    u16 closest=0;
    s08 error1, error2;
    u08 i;

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

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

}
```

sonar.h

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

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

#endif
```

sonar.c

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

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

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

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



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

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

```
//start the sonar "ping"
//note that it takes at least 65ms for the data to
//be available (so it's best to do something else
//while the sonar is processing)
s08 sonar_start(void)
{
    loop_until_bit_is_clear(TWCR, TWSTO);  //check no transmission in progress

    //based on demo code on p. 112 in mega323 manual
    TWCR = _BV(TWSTA) | _BV(TWEN); //send start condition
    while(!(TWCR & _BV(TWINT))){;} //wait for interrupt flag to get set
    if(TWSR != TW_START) {
        TWCR = _BV(TWINT)|_BV(TWSTO)|_BV(TWEN);
        return -1;  //error occured - should have been start
    }

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

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

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

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

    return 0;

}

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

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

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

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

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

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

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

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

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

    return 0;
}


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

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

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

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

}


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

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

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

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

    s08 error1, error2;
    u08 i;
    for(;;) {
        error1 = sonar_start();
        error2 = sonar_data(ranges, &light_sensor);

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

}
*/
```