Scott Stewart, Tim Galvin
T. Folsom
CSS 427
13 March 2015

Range Sensor

Problem Addressed

The problem addressed by our team was to develop and test the functionality of a sonar board for use in the Elcano project. The sonar board uses multiple sonar sensors simultaneously to gather range data of its surrounding environment. Each sonar board uses eight sonar sensors: one at each hour of the top half of the clock, and one behind. Harmonious operation of sonar sensors, their scope, and communication were problems to address.

Work to be Done

Existing datasheets and empirical data were to be used to determine a correct manner to poll sensors in a way that is efficient. The datasheets were also to be used to analyze any overlap of the sonar sensors. Code needed to be written to handle the coordination of the timing of the sonar sensors and the communication between them and the microcontroller using SPI. The code was to be robust and usable by others that work on the Elcano project.

Solution

Software Design

Our solution uses an Arduino microcontroller to communicate with the sonar board using SPI bus to coordinate the sonar sensors. The sonar sensors were put into groups of two or three sensors per group. A round of sensor data is gathered by simultaneously getting and reading range data from all sonar sensors in a group. The groups are pinged for data once per round in a set order. As a result, data is received in an orderly manner without much interference between sensors.

The defined rounds of polling using position on an imaginary clock as position

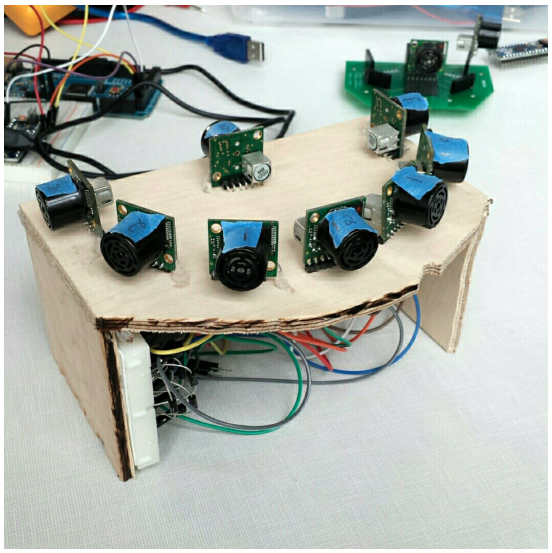
	Sensor 1	Sensor 2	Sensor 3
Round 1	12 o'clock	3 o'clock	9 o'clock
Round 2	1 o'clock	6 o'clock	10 o'clock
Round 3	2 o'clock	11 o'clock	

Rounds are handled by driving pins corresponding to a given group HIGH or LOW depending on which group's turn it is to get sonar data. This is handled in our setRound() function that takes two integers representing high and low order bits for each group.

The sensors themselves send a PWM signal that is read via an interrupt that records the time length of the signal. This time is then processed and converted into a range measurement in centimeters. This data is then printed.

Test Methods

We were faced with the problem of testing whether our code worked or not without having the new sonar board on hand. As a solution, we set about building a wooden jig that would hold the sensors in their correct orientation, and allow us to get meaningful readings from them.



Our make-shift "sonar-board"

Once we had the sensors in place, we were able to take the sonar board outside and test our ability to get raw data. In the beginning, this simply meant looking at a series of numbers in the serial monitor. This was useful data, but it was still difficult to convince ourselves that the data was legitimate without some sort of visual.

Enter processing.org. This open-source project allows for the rapid development of visualizations, and works very well with serial data. Using built in libraries, we were able adapt an example program written by Christan Marc Schmidt [1] (which shows random radii pie slices) to read serial data and give us a visual of the data from the sonars.

The program works by reading in a series of values from the given serial port, and then mapping those values to the radius of a given “slice” of the pie.

Output showing obstacles at 11 and 1 o’clock.

We discovered that noise was rampant anytime we tested indoors, but the readings seemed very reasonable once the board was outdoors.

Physical/Mechanical Considerations

We were aware throughout our testing that our wooden board was likely not ideal for using the sonar sensors. The sensors are angled correctly in terms of yaw, but they are able to wiggle their pitch somewhat.

Also, our knowledge of circuitry isn’t as robust as our coding experience, and we admit there may be some additional circuitry needed for reliable measurements. For example, we spent many hours wondering why we couldn’t get readings from multiple groups of sensors, only to find out that we needed diodes in series with the PWM output from the sensors in order to isolate them from each other. This of course will be a moot point once the new fabricated sonar boards arrive.

SPI Communication

We weren’t able to achieve reliable SPI communication in the time that we had. Currently the code will successfully send data over Serial from the sonar arduino.

The interaction of sending a command from the master, having the master wait for a reply, and then reliably sending the 26 bytes needed proved to be very difficult. Simple single byte communication was trivial, and we were able to get data sent from master to slave and back

again without any issue. However, when we moved to more complicated scenarios, things became more difficult. Here is a series of steps explaining what we were hoping to accomplish:

1. The master sends a command over SPI to the slave, indicating it wants sonar values.
2. The slave (sonar arduino) then goes into a mode of reading values from all sonar sensors. During this time, the master knows to stay idle.
3. Finally, the slave has data to send back to the master and begins transmission back over SPI.
4. The master is intelligent enough to know that the real data has begun being sent back, and stores the data accordingly.
5. The communication ends, the slave goes back into idle, and the master does what it needs to with the data.

References

1. Example Pie-chart with random radii. http://www.openprocessing.org/sketch/49035

Source Code

We will post the code here for reporting's sake, but we will also send the .ino files via Canvas message.

```
/* sonar board */
#define IRQ2_PIN 1  // pulse 10, 11 or 12
#define IRQ2_ID 3   // The interrupt ID relating to pin 1

#define IRQ3_PIN 0  // Pulse on 6 or 9
#define IRQ3_ID 2   // The interrupt ID relating to pin 0

#define SDA_PIN 2 //  I2C SDA
#define IRQ1_SCL_PIN 3  // Pulse on 1, 2, or 3; or I2C
#define IRQ1_ID 0       // The interrupt ID relating to pin 3

#define SNR_10_PIN 4  // Analog in 6 is Sonar 10
#define PWR_SEL_PIN  5  // Use 5V or 3.3V
#define SNR_11_PIN 6  // Analog in 7
#define RND0_PIN    11 // Low order bit for selecting the round
#define RND1_PIN     7  // Hi bit of round select
#define PWR_ON_PIN   8  // Apply power to sonars
#define FRONT_PIN    9  // Do not disturb if high
#define REAR_PIN    10  // Do not disturb if high

#define SNR_SEL_PIN  SS  // Chip select from host processor
#define SNR_SI_PIN   MOSI // This processor's serial input
```

```
#define SNR_SO_PIN   MISO // This processor's serial output
//#define SNR_12_PIN   A0

// In this version of the code, we use A1-A3 to
// Signal which sensors should be triggered in each
// round. Once the real board arrives, these can be
// reclaimed as the rounds will be controlled by
// ciruitry connected to the RND0_PIN and RND1_PIN pins
#define GROUP1_PIN    A1
#define GROUP2_PIN    A2
#define GROUP3_PIN    A3
//#define GROUP2_SEL   A4
//#define GROUP3_SEL   A5
//#define SNR_3_PIN    A3
//#define SNR_6_PIN    A4
//#define SNR_9_PIN    A5

#define TIMEOUT_PERIOD 100 // ms
#define ROUND_DELAY 100 // ms between rounds, seems to help stability

// When true, send data over Serial, false send data over SPI
#define DEBUG true

// When sedning over Serial, true gives verbose output
#define VERBOSE_DEBUG false

/* Two boards can hold 12 sonars on the points of a clock.
   The front board holds Sonar 12 (straight ahead)
   and 1, 2, and 3F to the right.
   It also holds 9F, 10, and 11 to the left.
   The rear facing board holds 3R through 9R, with 6R pointing back.

   The front sonar board uses the SPI interface to appear as a slave to a host
computer.
   When the host asks for sonar readings, the front board sends all data.

   The front board controls the rear board over an I2C interface. It will intruct the
rear
   board to take readings when the front board is taking readings.
   The hardware is identical for the front and rear boards.

   There may be no rear board present.  In this case, the front board may use the 9F
socket.

   Sonars are grouped into three rounds so that multiple sonars can fire simultaneously
and
   not intefere with each other. Rounds are selected by the (RND0, RND1) signal,
   which is used as a Gray code (only one bit changes at a time).
```

```
   Round      Sonars
   00          none
   01          12, 3, 9
   11          1, 6, 10
   10          2, 11
   A round may fire up to 3 sonars. The sonar range may be reported as either an analog
value
   or as a pulse width. A pulse width will cause an interrupt.
   There are three interrupt handlers, one for each sonar fired in the round.
*/
// Modify EXPECTED_SIGNALS to mtch which sonars are on the board.
#define EXPECTED_SIGNALS1 3
#define EXPECTED_SIGNALS2 3
#define EXPECTED_SIGNALS3 2

#define STATE_INITIAL 0
#define STATE_READY   1
#define STATE_ROUND1  2
#define STATE_ROUND2  3
#define STATE_ROUND3  4
#define COMMAND_GO 1
#define RANGE_DATA_SIZE 13

#include "pins_arduino.h"

byte BoardCount = 1;  // just the front board
byte State = STATE_INITIAL; // initializing
volatile byte SignalsReceived;
volatile unsigned long timeLeft;
volatile unsigned long timeRight;
volatile unsigned long timeBehind;
volatile unsigned long timeStart;
volatile boolean timeStartSet;
volatile byte valueIn;
volatile boolean processIt;


int range[RANGE_DATA_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int roundCount = 0, timeoutStart = 0, timeSinceLastRound = 0;
volatile int isr1Count, isr2Count, isr3Count;

void setup()
{
  // SPI Setup
  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE); // turn on SPI in slave mode
  SPCR |= _BV(SPIE); // turn on interrupts for SPI
  processIt = false; // false until we have a request to process
```

```
  timeLeft = timeRight = timeBehind = timeStart = 0;
  isr1Count = isr2Count = isr3Count = 0;
  timeStartSet = false;
  Serial.begin(115200);
  pinMode(A1, OUTPUT);
  pinMode(A2, OUTPUT);
  pinMode(A3, OUTPUT);
  pinMode(0, INPUT);
  pinMode(1, INPUT);
  pinMode(3, INPUT);
  State = STATE_INITIAL;
  BoardCount = 1;
//    digitalWrite(RND0_PIN, LOW);
//    digitalWrite(RND1_PIN, LOW);
  setRound(0, 0);
  digitalWrite(FRONT_PIN, HIGH);  // signal rear board that we are here; See if there
is a rear board
  // select 5V power
  digitalWrite(PWR_SEL_PIN, HIGH);
  // turn on sonars
  digitalWrite(PWR_ON_PIN, HIGH);

//    for (int i = 0; i < 1000; i++)
//    {  // See if there is a second board
//      byte Rear = digitalRead(REAR_PIN);
//      if (Rear == HIGH)
//      {
//           BoardCount = 2;
//           // need to dynamically attach/detacy IRQ1_SCL_PIN so that it is shared
between IntRight and I2C
//           break;
//      }
//    }

   delay(200); // Per sonar datasheet, needs 175ms for boot
   attachInterrupt(IRQ2_ID, IsrLeft, CHANGE);
   attachInterrupt(IRQ3_ID, IsrBehind, CHANGE);
   if (BoardCount == 1)
   { // don't need D3 for I2C
     attachInterrupt(IRQ1_ID, IsrRight, CHANGE);
   }
   digitalWrite(FRONT_PIN, LOW); // open for business
   State = STATE_READY;
}

// SPI interrupt routine
ISR (SPI_STC_vect) {
```

```
  valueIn = SPDR;  // grab byte from SPI Data Register
  processIt = true;
}

void IsrLeft()
{  // ISR for end of pulse on sonar 10, 11 or 12
  isr1Count++;
  noInterrupts();
  SignalsReceived++;
  if (digitalRead(IRQ2_PIN) == HIGH && timeStartSet == false)
  {
    timeStart = micros();
    timeStartSet = true;
  }
  else
  {
    timeLeft = micros();
  }
  interrupts();
}
void IsrBehind()
{  // ISR for end of pulse on sonar 6 or 9
   isr2Count++;
   noInterrupts();
   SignalsReceived++;
   if (digitalRead(IRQ3_PIN) == HIGH && timeStartSet == false)
  {
    timeStart = micros();
    timeStartSet = true;
  }
  else
  {
    timeBehind = micros();
  }
   interrupts();
 }
void IsrRight()
{// ISR for end of pulse on sonar 1, 2 or 3
   // Log time of the pulse end
   isr3Count++;
   noInterrupts();
   SignalsReceived++;
   if (digitalRead(IRQ1_SCL_PIN) == HIGH && timeStartSet == false)
  {
    timeStart = micros();
    timeStartSet = true;
  }
  else
```

```
  {
    timeRight = micros();
  }
   interrupts();
}

void loop ()
{
    State = STATE_READY;
//    digitalWrite(RND0_PIN, LOW);
//    digitalWrite(RND1_PIN, LOW);
    setRound(0, 0);


    //readInput();   // Sonar is slave waiting for command from master
    // TO DO: Send start signal to rear board.

//******Round 1*******
    State = STATE_ROUND1;
    SignalsReceived = 0;
    // send a pulse on sonars
    interrupts();
    setRound(0, 1);
//    digitalWrite(RND1_PIN, HIGH);

    timeoutStart = millis();
    while (SignalsReceived < EXPECTED_SIGNALS1)
    {
      if ((millis() - timeoutStart) > TIMEOUT_PERIOD) break;
    }
    delay(ROUND_DELAY);
    timeStartSet = false;
    range[12] = (timeLeft - timeStart) / 58;
    range[3]  = (timeRight - timeStart) / 58;
    range[9]  = (timeBehind - timeStart) / 58;


//******Round 2*******
    State = STATE_ROUND2;
    SignalsReceived = 0;
//    digitalWrite(RND0_PIN, HIGH);

//    send a pulse on sonars
    setRound(1, 1);

    timeoutStart = millis();
    while (SignalsReceived < EXPECTED_SIGNALS2)
    {
```

```
      if ((millis() - timeoutStart) > TIMEOUT_PERIOD) break;
    }
    delay(ROUND_DELAY);
    timeStartSet = false;
    range[10] = (timeLeft - timeStart) / 58;
    range[1]  = (timeRight - timeStart) / 58;
    range[6]  = (timeBehind - timeStart) / 58;



//******Round 3*******
    State = STATE_ROUND3;
//    digitalWrite(RND1_PIN, LOW);
    SignalsReceived = 0;

    // send a pulse on sonars
    setRound(0, 1);

    timeoutStart = millis();
    while (SignalsReceived < EXPECTED_SIGNALS3)
    {
      if ((millis() - timeoutStart) > TIMEOUT_PERIOD) break;
    }
    delay(ROUND_DELAY);
    timeStartSet = false;
    range[2] = (timeLeft - timeStart) / 58;
    range[11]  = (timeRight - timeStart) / 58;

//  // TO DO: Receive data from rear board
    writeOutput();
    State = STATE_READY;
    roundCount++;
    //delay(120);
//    return;
}

// This is how we are signaling the sensors to go on
// each round without having the proper sonar board
// on hand. With the proper board on hand, the process
// of signaling sensors is reduced to toggling the
// RND0_PIN and RND1_PIN pins
void setRound(int highOrderBit, int lowOrderBit)
{
  if (highOrderBit == 0 && lowOrderBit == 0)
  {
    // All off
    digitalWrite(A1, LOW);
    digitalWrite(A2, LOW);
```

```
    digitalWrite(A3, LOW);
  } else if (highOrderBit == 0 && lowOrderBit == 1)
  {
    // 12, 3, 9
    digitalWrite(A1, HIGH);
    digitalWrite(A2, LOW);
    digitalWrite(A3, LOW);
    delay(10);
    digitalWrite(A1, LOW);
  } else if (highOrderBit == 1 && lowOrderBit == 0)
  {
    // 2, 11
    digitalWrite(A1, LOW);
    digitalWrite(A2, HIGH);
    digitalWrite(A3, LOW);
    delay(10);
    digitalWrite(A2, LOW);
  } else if (highOrderBit == 1 && lowOrderBit == 1)
  {
    // 1, 6, 10
    digitalWrite(A1, LOW);
    digitalWrite(A2, LOW);
    digitalWrite(A3, HIGH);
    delay(10);
    digitalWrite(A3, LOW);
  }
}
void readInput()
{
  byte select;
  // wait for host to request information
  do
  {
    select = digitalRead(SNR_SEL_PIN);
  } while (select == HIGH);
  if (DEBUG) return;
  while (!processIt || valueIn != COMMAND_GO); // stay in routine until commanded to
start on MOSI
  processIt = false;
}

void writeOutput()
{
  if (DEBUG) {
    if (VERBOSE_DEBUG) {
      Serial.print("Round: ");
      Serial.print(roundCount);
      Serial.print(" millis: ");
```

```
      Serial.print(millis());
      Serial.print(", millis since last: ");
      Serial.print(millis() - timeSinceLastRound);
      timeSinceLastRound = millis();
      Serial.print(", SigReceived: ");
      Serial.print(SignalsReceived);
      Serial.print(", isr1: ");
      Serial.print(isr1Count);
      Serial.print(", isr2: ");
      Serial.print(isr2Count);
      Serial.print(", isr3: ");
      Serial.print(isr3Count);
      Serial.print(" Vals: ");
    }
    for (int i = 1; i < 13; ++i)
    {
      Serial.print(range[i]);
      Serial.print(" ");
    }
    Serial.println();
  }
  else
  {
    // Here is where data will be sent over SPI. Still needs work
    int dataPos = 0;
    processIt = false;
    while (dataPos < RANGE_DATA_SIZE) {
      int timeStart = millis();
      while(!processIt) {
        if (millis() - timeStart > TIMEOUT_PERIOD) break;
      }
      writeIntData(range[dataPos++]);
      delay(20);
    }
    dataPos = 0;
  }
}

// Send two bytes over SPI in order to represent an int
void writeIntData(int data)
{
  byte highOrderByte = (byte) (data >> 8);
  byte lowOrderByte = (byte) data;

  // Send first byte
  SPDR = highOrderByte;
  processIt = false;
```

```
  // Wait for SPI interrupt before sedning second byte
  while(!processIt)
  {
        if (millis() - timeStart > TIMEOUT_PERIOD) break;
  }
  // Send second byte
  SPDR = lowOrderByte;
  processIt = false;
}
```