Documentation for 2017 ATHENS-Program course

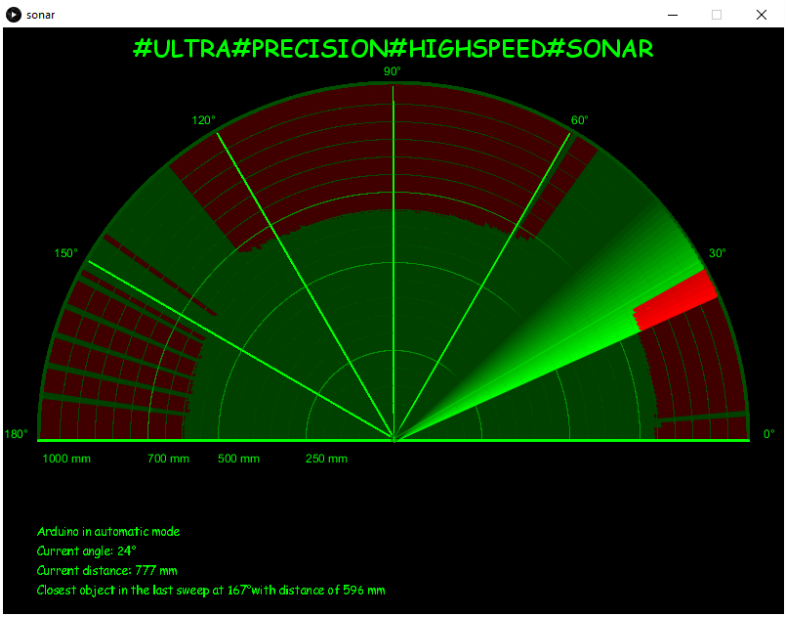
UPM-115 "Physical Computing based on Open Software and Hardware Platforms"

by:

Nathanael Bosch, Technische Universität München (TUM)
&
Stephan Puchegger, Technische Universität Wien (TUW)

Project:

„Ultra Precision High speed Sonar, UPHS"



Task description

The goal of this project is to build up a simple sonar device using Arduino and Processing. Some kind of communication between the Arduino device and the GUI (graphical user interface) on the laptop has to be established. The GUI displays the output of the sonar in range of up to 1m and 0…180 degree. The sweeping speed of the sonar can be controlled, via a potentiometer for example.

To keep the code and the hole project as simple as possible, error handling is not necessary.

Used materials and software

Material:

- 1x Arduino Nano, Rev3.0
- 1x Servo motor, TowerPro SG90
- 1x Ultrasonic Distance Sensor, HC-SR04
- 1x Potentiometer, 10k
- 1x Breadboard, standard size
- 1x Breadboard, mini size
- Wires

Software:

- Arduino IDE, v1.8.5
- Processing, v3.3.6

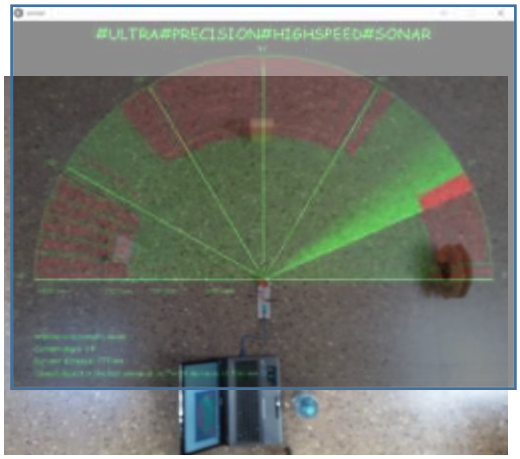
Realisation

To establish the connection between the Arduino and the GUI on the laptop a serial connection is used. The GUI works as the server while the Arduino serves as the client. The server sends out a call to the client, which then responds with the actual data (angle, distance, potentiometer value).

For sweeping, the sonar sensor is mounted on a mini breadboard which is then rotated by the servo motor. The servo motor has a rotation range of 0…180 degree. To control the speed of the sweeping, a potentiometer is connected to one of the analog input pins of the Arduino, as shown in the schematics.

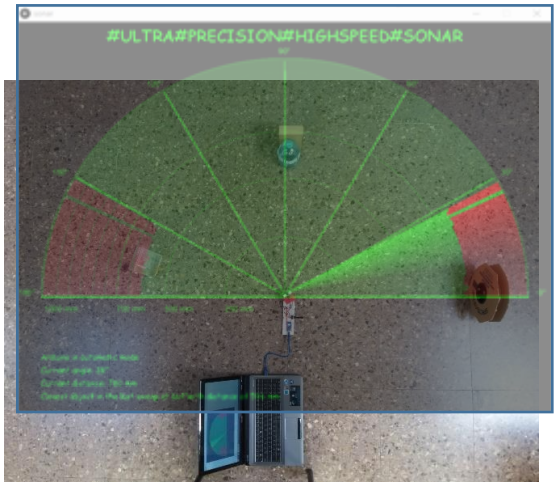
To reduce false measurements of the ultra-sonic sensor, multiple measurements of one angle are taken. Then the median value is taken. The parameters for the median calculation can be set by the code. The number of measurements taken can be increased, but this goes with a lack of speed.

Sample images



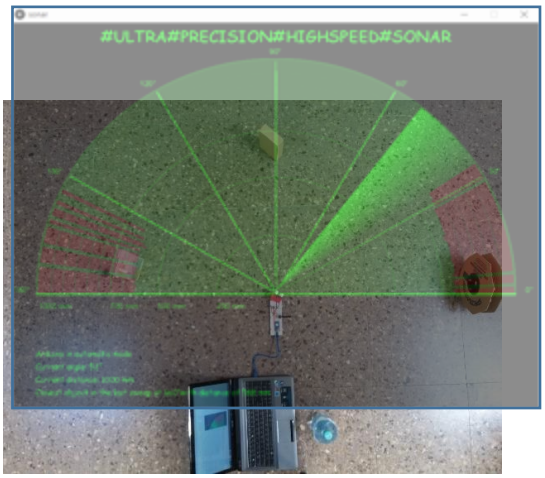
Normal setting:

Every object is recognizes by the sonar. Because the HC-SR04 sensor is not perfectly focused (measurement angle of 15…30 degree) small object appear to be bigger than in reality.



Stealth setting:

If we put a round object (a bottle for example) in-front of another object, this object disappears on the screen. So it gets invisible ("stealth"). This might be because of the way the sonic waves reflect on curved surfaces. So no signal is received by the sensor.



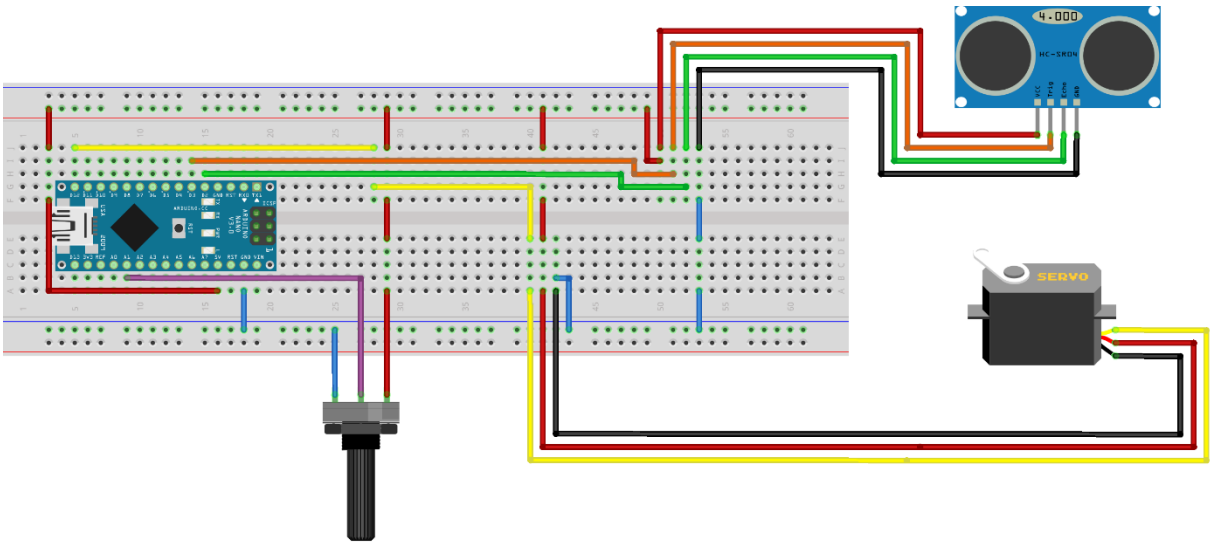
Angle setting:

In this setting the middle object is turned in a way, so that no outgoing sonic wave is reflected to the receiving part of the HC-SR04 sensor.

Test result:

Accuracy of measured distance is surprisingly good. But our tests also reveal the weakness of super-sonic sensors. They only work perfectly on good reflecting surfaces. Curved or angled surfaces can hardly be detected by super-sonic sensors.

Code and schematics



Processing code

```
import processing.serial.*;

Serial myPort;

// Define variables used to process the response
int responseLength = 4;
int[] serialInArray = new int[responseLength];
int serialCount = 0;
boolean firstContact = false;

int lastAngle = 0;
int angle = 0;

// Variables to store the state of the arduino in
int arduinoMode = 'A';
int arduinoDistance = 0;
int arduinoAngle = 0;
int arduinoSpeed = 0;
int[] allDistances = new int[181];

// Some constantsMINSPEED
int MINSPEED = 100;
int MAXSPEED = 0;
int MINDISTANCE = 20;
int MAXDISTANCE = 1000;

// Graphical variables
int centerX;
int centerY;
int maxLineLength;
boolean angleRadiant = false;

// Some variables used for debugging
int currentTime = 0;

void setup() {
  // Add Port
  printArray(Serial.list());
  String portName = Serial.list()[4];
  myPort = new Serial(this, portName, 9600);
```

```
  // Initialize display window
  size(800, 600);
  centerX = int(width/2);
  centerY = centerX+20;
  maxLineLength = int(width/2*0.9);

  background(0);
  delay(1000);
  drawRadar(true);

  PFont comic = loadFont("ComicSansMS-48.vlw");
  textFont(comic);
  textAlign(CENTER);
  textSize(24);
  text("#ULTRA#PRECISION#HIGHSPEED#SONAR", width/2, 30);
  textSize(12);
}

void draw() {
  /*
  This function loops all the time
  We only want to update the graphics, if there is
  new data from the arduino, which we check by comparing
  the current angle with the one from the last iteration.

  If we have new data:
  1. Delete old text
  2. Draw measurements
  3. Draw radar
  4. Write new text
  */
  angle = arduinoAngle;
  if(angle == lastAngle){
    delay(2);
    return;
  }

  // Clean text
  noStroke();
  fill(0);
  rect(0, centerY+25, width, height);

  // Draw radar and measurements
  drawMeasurements();
  drawRadar(false);

  // Write text
  writeText();

  // Save angle to compare it in the following iteration
  lastAngle = angle;
}

////////////////////////////////////////////////////////////////////////////
/////
// Everything concerning graphics
////////////////////////////////////////////////////////////////////////////
/////
void drawMeasurements(){
  /*
  Draw the current measurement
```

```
  This function draws the last measurement, and redraws the last 20 angles
in
  different shades improved aesthetics
  */
  angle = arduinoAngle;

  fill(0);
  noStroke();
  strokeWeight(7);
  for(int i=0; i<20; i++){
    // Iterate over the last 20 angles
    int drawAngle;
    if(angle > lastAngle){
      drawAngle = angle-i;
    }else{
      drawAngle = angle+i;
    }
    if(drawAngle < 0 || drawAngle > 180){
      // Dont process bad angles
      continue;
    }

    int lineLength = int(transformScale(
      allDistances[drawAngle]
      )/transformScale(MAXDISTANCE) * maxLineLength
      );

    // Green until `distance`, red afterwards
    stroke(0, 255-(10*i), 0);
    line(centerX,
         centerY,
         centerX + cos(radians(-drawAngle))*lineLength,
         centerY + sin(radians(-drawAngle))*lineLength);
    stroke(255-(10*i), 0, 0);
    if(lineLength < maxLineLength){
    line(centerX + cos(radians(-drawAngle))*lineLength,
         centerY + sin(radians(-drawAngle))*lineLength,
         centerX + cos(radians(-drawAngle))*maxLineLength,
         centerY + sin(radians(-drawAngle))*maxLineLength);
    }
  }
}

void drawRadar(boolean txt){
  /*
  1. Draw Distance Circles
  2. Draw Degrees
  3. Only write text when we call it the first time (handled by the bool
flag `txt`)
  */

  // 1. Draw circles every 50mm
  for(int d=50; d<=MAXDISTANCE; d+=50){
    float screenDist =
transformScale(d)/transformScale(MAXDISTANCE)*maxLineLength;

    fill(0, 255, 0);
    if(txt && (d==250 || d==500 || d==700 || d==1000)){
      String label = Integer.toString(d) + " mm";
      text(label, centerX-screenDist, centerY+25);}
    noStroke();

    noFill();strokeWeight(0);
```

```
    if(d==MAXDISTANCE){
      // Draw the last one thicker
      strokeWeight(3);
      d = int(MAXDISTANCE*1.01);
      screenDist =
transformScale(d)/transformScale(MAXDISTANCE)*maxLineLength;
      stroke(0, 255, 0);
    }
    if(d==250 || d==500 || d==700){
      // Draw these ones a bit thicker
      stroke(0, 150, 0, 127);
      strokeWeight(0);
    }else stroke(0, 80, 0, 80);
    arc(centerX, centerY, 2*screenDist, 2*screenDist, radians(180),
radians(360));
  }

  // 2. Draw degrees every 30 degrees
  String[] radNames = {"0", "π/6", "π/3", "π/2", "2π/3", "5π/6", "π"};
  for(int a=0; a<=180; a+=30){
    noStroke();
    strokeWeight(1);
    stroke(0, 255, 0, 127);
    if(a==0 || a==180){
      // Draw the bottom degrees thick and a bit lower
      strokeWeight(3);
      line(centerX, centerY+2,
           centerX + cos(radians(-a))*maxLineLength*1.01,
           centerY + sin(radians(-a))*maxLineLength*1.01+2);

    }
    else{
    line(centerX,
         centerY,
         centerX + cos(radians(-a))*maxLineLength,
         centerY + sin(radians(-a))*maxLineLength);
    }

    if(txt){
      // Write labels for the angles
      noFill();
      String label;
      if(angleRadiant){
        label = radNames[a/30];
      }else{
        label = Integer.toString(a)+"°";
      }
      textAlign(CENTER);
      text(label,
      centerX + cos(radians(-a))*maxLineLength*1.07,
      centerY + sin(radians(-a))*maxLineLength*1.03);
    }
  }
}

void writeText(){
  /*
  Print out additional info below

  Example:
  Arduino in automatic mode
  Current angle: 127°
```

```
  Current distance: 133 mm
  Closest object in the last sweep at 67° with distance of 133 mm
  */

  textAlign(LEFT);

  int leftDistance = centerX-maxLineLength-5;
  int upperDistance = 75;

  if(arduinoMode == 'A'){
    String modeText = "Arduino in automatic mode";
    text(modeText, leftDistance, centerY+upperDistance);
    upperDistance += 20;
  }

  String angleText = "Current angle: " + Integer.toString(angle) + "°";
  text(angleText, leftDistance, centerY+upperDistance);
  upperDistance += 20;

  String distanceText = "Current distance: " +
Integer.toString(allDistances[angle]) + " mm";
  text(distanceText, leftDistance, centerY+upperDistance);
  upperDistance += 20;

  String closestObjectText = (
    "Closest object in the last sweep at " +
    Integer.toString(argMin(allDistances)) + "°" +
    "with distance of " + Integer.toString(min(allDistances)) + " mm");
  text(closestObjectText, leftDistance, centerY+upperDistance);
  upperDistance += 20;
}

////////////////////////////////////////////////////////////////////////////
/////
// Client-Server-Communication
////////////////////////////////////////////////////////////////////////////
/////
// We decided to go for a client-server structure
// The computer with processing acts as the controlling instance, and the
// arduino processes the call and adjust its settings, before measuring
// and sending the data in the response
void serialEvent(Serial myPort) {
  /*
  1. Read a byte from the Serial port
  2. Wait for the first handshake
  3. Once we established contact
    3a. Save bytes in serialInArray
    3b. Once we filled it process it
  */
  int inByte = myPort.read();
  if (firstContact == false) {
    if (inByte == 'H') {
      println("First Contact made");
      myPort.clear();
      firstContact = true;
      callArduino();
    }
  }
  else {
    serialInArray[serialCount] = inByte;
    serialCount++;

    if (serialCount >= responseLength ) {
```

```
      /*
      This gets called if and only if we established contact with
      the initial handshake, and when we successfully got `responseLength`
bytes
      (three by default)

      Here we actually process the response from the arduino
      */
      // Some delay printing for debugging purposes
      print("Delay between calls: ");
      print(millis()-currentTime);
      println(" ms");
      currentTime = millis();

      // Process the new data and call for more
      processResponse(serialInArray);
      callArduino();

      // Reset serialCount:
      serialCount = 0;
    }
  }
}

void processResponse(int responseArray[]){processResponse(responseArray,
false);}
void processResponse(int responseArray[], boolean verbose){
  /*
  Process the response we get from the arduino
  This means decoding the values in each byte (we often mapped values to 0-
255
  so that it fits in a single byte) and store the new data in the global
variables
  */
  arduinoMode = responseArray[0];
  int arduinoDistanceCoded = responseArray[1];
  arduinoDistance = int(map(arduinoDistanceCoded, 0, 255, MINDISTANCE,
MAXDISTANCE));
  arduinoAngle = responseArray[2];
  int arduinoPotiValue = responseArray[3];
  arduinoSpeed = int(map(arduinoPotiValue, 0, 255, MINSPEED, MAXSPEED));

  allDistances[arduinoAngle] = arduinoDistance;

  // The differences in "speed" are implemented by delaying the calls
  delay(arduinoSpeed);
  // Print response for debugging
  if(verbose){
    print("Processing response: angle=");
    print(arduinoAngle);
    print(", distance=");
    print(arduinoDistance);
    print(", speed=");
    println(arduinoSpeed);
  }
}

void callArduino(){callArduino('A', 255, 255, false);}
void callArduino(char mode, int accuracy, int angle, boolean verbose){
  /*
  Send commands to the arduino
  This is not completely necessary as multiple modes are not implemented
yet,
```

```
  but this provides a framework in which the computer with processing is
  completely in charge of giving the command for the new measurements,
  and the arduino is a smart sensor with some functionality
  */
  myPort.clear();
  myPort.write(mode);
  if (accuracy>10 && accuracy!=255) accuracy = 10;  // More than 10 does
not make sense
  myPort.write(accuracy);
  myPort.write(angle);
  // Print out the call for debugging purposes
  if(verbose==true){
    print("Sent: mode=");
    print(mode);
    print(", accuracy=");
    print(accuracy);
    print(", angle=");
    print(angle);
    println();
  }
}


////////////////////////////////////////////////////////////////////////////
/////
// Helper Funtions
////////////////////////////////////////////////////////////////////////////
/////
float transformScale(int l){return transformScale(float(l));}
float transformScale(float l){
  /*
  We thought about transforming the employed scale to some
  different one, e.g. a logarithmic scale or a square root scale,
  to better see small differences in distance.
  The following function handles that in a modular way, but we
  decided not to use it for now.
  */
  return l;
}

int argMin(int arr[]){
  /* There is no argMin implemented, so we had to do our own */
  int lastMin = arr[0];
  int minInd = 0;
  for(int i=0; i<arr.length; i++){
    if(arr[i]<lastMin){
      print(i);
      print(" ");
      println(arr[i]);
      lastMin = arr[i];
      minInd = i;
    }
  }
  return minInd;
}
```

Arduino code

```
/*
 * Simple arduino sonar project
 *
 * 1. establishContact()
 * 2. loop():
 *    2a. Wait for incoming commands from processing
 *    2b. Update commands
 *    2c. Move servo and get distance measurement
 *    2d. Send back measurement and status
 */

#include <Servo.h>
#include "FastRunningMedian.h"
// See https://playground.arduino.cc/Main/RunningMedian for the
runningMedian
// or http://forum.arduino.cc/index.php?topic=53081.msg1160999#msg1160999
for the faster version we used

// Define pins
#define trigPin 3
#define echoPin 2
#define servoPin 12
#define potPin A1

// Define some constants
#define MAXSPEED 2
#define MINSPEED 50
#define MINDISTANCE 20
#define MAXDISTANCE 1000

Servo myServo;

int distance = 0;
int angle = 0;
int MINANGLE = 0;
int MAXANGLE = 180;
char mode = 'A';
int direction = 1;
int inByte = 0;

#define callLength 3
byte serialInArray[callLength] = {0};
int countIncoming = 0;

#define MEDIANWINDOW 3
int accuracy = 2;
FastRunningMedian<int, MEDIANWINDOW, 0> newMedian;

void setup()
{
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(servoPin, OUTPUT);
  pinMode(potPin, INPUT);

  myServo.attach(servoPin);

  // Send a byte to establish contact until Processing responds
  establishContact();
```

```
}

void loop()
{
  if (Serial.available() > 0) {
    /*
     * 1. Read incoming bytes and adjust controls accordingly
     * 2. Get sensor data
     * 3. Send back data
     */
    serialInArray[countIncoming] = Serial.read();
    countIncoming++;
    if(countIncoming >= callLength){
      /*
       * This gets called if we got the 3 bytes
       */
      countIncoming = 0;
      processIncoming(serialInArray);
      moveAndMeasure();
      sendBack();
    }
  }
}

void establishContact() {
  /*
   * Initial Handshake
   * We use a client-server structure
   */
  while (Serial.available() <= 0) {
    Serial.write('H');
    delay(300);
  }
}

void processIncoming(byte arr[]){
  /*
   * Update all settings
   * Angle and speed are not really used yet as we only
   * send the default values from processing
   */
  mode = arr[0];
  if(arr[1]<255){
    accuracy = arr[1];
  }
  if(arr[2]<255){
    angle = arr[2];
  }
}

void moveAndMeasure(){
  /*
   * Do the actual work here:
   * - Move servo in the right direction
   * - Measure the distance
   */
  if(mode == 'A'){
    /*
     * Automatic mode:
     * - Run from 0 to 180 degrees (or 180 to 0)
     * - Measure the distance at each degree
     */
    // Move Servo
```

```
    if (angle == MAXANGLE) direction = -1;
    if (angle == MINANGLE) direction = 1;
    angle += direction;
    myServo.write(angle);

    // Get distance
    for(int i=0; i<accuracy; i++){
      newMedian.addValue(getDistance());
    }
    distance = newMedian.getMedian();
  }
}

int getDistance(){
  /*
   * Measure the distance using the ultrasonic sensor
   */
  long duration;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH, 6000);
  if(duration == 0){
    duration = 6000;
  }
  return (duration/2) / 29.1 * 10;
}

void sendBack(){
  /*
   * Directly send the current status of the Arduino
   * Encode Distance and Poti Value so that we fit the maximum amount of
information into one byte
   */
  Serial.write(mode);
  int codedDistance = constrain(map(distance, MINDISTANCE, MAXDISTANCE, 0,
255), 0, 255);
  Serial.write(codedDistance);
  Serial.write(angle);
  int codedPotiVal = map(analogRead(potPin), 0, 1023, 0, 255);
  Serial.write(codedPotiVal);
}
```