ELEC-C9820 ED Workshop, Exercise 4, 28.-1.2.2019

(Based on the example in the Sähköpaja course by Tommi Pulli)

Current version 28th of January 2019 by Salu Ylirisku

Sensor Experiment

This experiment helps you to familiarise with the use of sensors with Arduino. We shall connect different types of sensors to and read their values with the microcontroller. We also study how to average signals as well as to calibrate sensors.

Equipment

- Arduino UNO (or similar
- USB cable
- Accelerometer (ADXL335, Analog Devices)
- Ultrasonic proximity sensor (LV-MaxSonar-EZ1, Maxbotix)
- Digital temperature sensor (DS18B20, Dallas Semiconductors)
- Resistor (4.7 kΩ)
- Breadboard + wires

Accelerometer

Accelerometers measure acceleration. They function typically in three different axes, namely X, Y, Z. They are based on tiny moving parts inside the sensor that cause small changes in capacitance. The sensor component outputs these changes as voltage differences, which can be measured by the microcontroller.

Connections

1. ADXL335 uses input of 1.8-3.6 V. Thus, we cannot use the 5V voltage source of Arduino UNO. Connect the 3.3 V source (POWER 3.3 V) to function as the supply voltage (V_{in} tai V_{CC}). Connect the GND-pin of the accelerometer to ground.
    a. NOTE: If the microcontroller does not have a 3.3V output, you can use e.g. 5V supply voltage and a 3.3V regulator. Instructions for connections you find from the regulator's data sheet
2. Connect the X, Y and Z pins to Arduino's ANALOG IN pins (A0, A1 and A2)
3. After checking the connections, connect Arduino to a computer with the USB cable

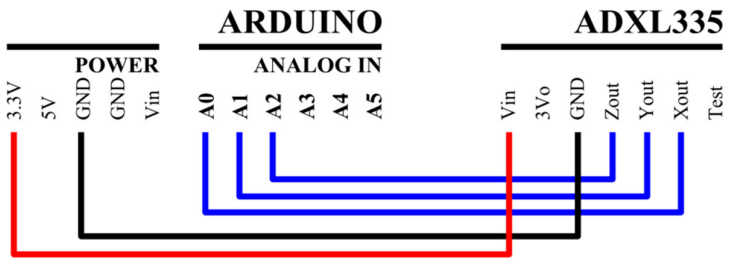

Figure 1. Connections between the accelerometer and Arduino

Using the sensor

The code can be copy-pasted from here: https://codepad.co/snippet/XGy6lTiw

```
01.  const int xPin = A0;
02.  const int yPin = A1;
03.  const int zPin = A2;

04.  int xVal, yVal, zVal;

05.  void setup() {
06.       Serial.begin(9600);
07.  }

08.  void loop() {
09.       xVal = analogRead(xPin);
10.       yVal = analogRead(yPin);
11.       zVal = analogRead(zPin);

12.       Serial.print("Values: ");
13.       Serial.print(xVal);
14.       Serial.print(",");
15.       Serial.print(yVal);
16.       Serial.print(",");
17.       Serial.println(zVal);

18.       delay(200);
19.  }
```

The first part defines the pins (xPin, yPin, and zPin). We define these as constants (const), as they do not change while the program is running.

We also use the names (A1, A2, A3) of the pins rather than numbers, because the exact port numbers are different on different Arduino boards. The pre-processors knows, which numbers are correct depending on the board you have chosen, and it will replace those with numbers during the compilation.

The sensor output is read through the analogRead() function, which takes pin as its parameter. The values vary between 0-1023 with the analog pins. This gives us a resolution between readings of: 5 volts / 1024 units, or 0.0049 volts (4.9 mV) per unit.

Once uploaded to Arduino, open the serial monitor and check the output. Are the values staying the same, or are they jumping back and forth? If they keep jumping, that is called *noise*. This can be smoothened by averaging across several subsequent values.

Signal smoothing

The code can be copy-pasted from here: https://codepad.co/snippet/qEWWWfvB

```
01.  const int xPin = A0;
02.  const int yPin = A1;
03.  const int zPin = A2;
04.  int xVal, yVal, zVal;
05.  const int nAve = 10;

06.  void setup() {
07.       Serial.begin(9600);
08.  }

09.  void loop() {
10.       xVal=0; yVal=0; zVal=0;
11.       for (int i=0; i<nAve; i++) {
12.            xVal += analogRead(xPin);
13.            yVal += analogRead(yPin);
14.            zVal += analogRead(zPin);
```

```
15.            delay(5);
16.       }

17.       xVal = xVal/nAve;
18.       yVal = yVal/nAve;
19.       zVal = zVal/nAve;

20.       Serial.print("Values: ");
21.       Serial.print(xVal);
22.       Serial.print(",");
23.       Serial.print(yVal);
24.       Serial.print(",");
25.       Serial.println(zVal);
26.       delay(200);
27.  }
```

This example uses a for-loop to collect multiple readings from the sensor pins (the amount is defined by the variable nAve, row 05). The values are summed up using the notation +=. It saves some writing.

The example code here is much shorter than if it would be made with the same algorithm as the official Arduino-example, which can be found in the folder *File>Examples>03. Analog>Smoothing*.

Can you point out the key differences in the algorithm as compared to the above?

(An example made with the official running average way can be found here: https://codepad.co/snippet/v6uH9dQ0)

Simple gesture recognition

Accelerometers can be used to detect particular motions and orientations of a device. Let us do an experiment to detect whether the device (or accelerometer) is set upright or laying flat on a table.

First thing that we need to do is to identify the 'gestures,' i.e. the desired value-ranges for the sensor values. This is called calibration.

Start the serial monitor while the previous code is still running. Let the device stay flat for some time. You can now pull off the USB cable and the serial monitor stops. Check the minimum and maximum values for all three axes across several lines of input from the serial monitor.

They can look like these:

```
Values: 336,326,408
Values: 336,327,408
Values: 336,326,407
```

These would result in the following conclusion: xVal is between 336-336, yVal between 326-327, and zVal between 407-408. These are the ranges of values that enable us to say that the device is laying flat.

Now do the same tracking of the values when the device is held upright. Then update the code to your needs. It can be copy-pasted from here: https://codepad.co/snippet/WRSTIqvr

We have added two functions to the code to detect the target 'gestures'. These are called "isDeviceFlat()" and "isDeviceUpright()". These functions use the calibrated min and max values for each accelerometer output axes.

Digital Thermometer

A digital thermometer converts temperature into digital format. The DS18B20 is based on internal oscillator that is very sensitive to temperature changes.

Reading the digital thermometer is very different from the reading the accelerometers. This stems from the thermometer's communication protocol, which must be implemented into the program code. Each DS18B20 has a unique address, and they can be specifically addressed in larger set of sensors with only one wire.

Connecting the DS18B20

1. The sensor uses 3.0-5.5V current, so you can use 3.3V or 5V connectors on the Arduino board. Connect to the V_{DD} pin of the sensor.
2. Connect the DQ pin to the DIGITAL 2 pin on the Arduino board.
3. DQ pin requires a 4.7kΩ pull-up resistor. As it is pull-up, you need to connect it to the supply voltage.
4. Check the connections! When wrongly connected, the sensor heats up a lot.

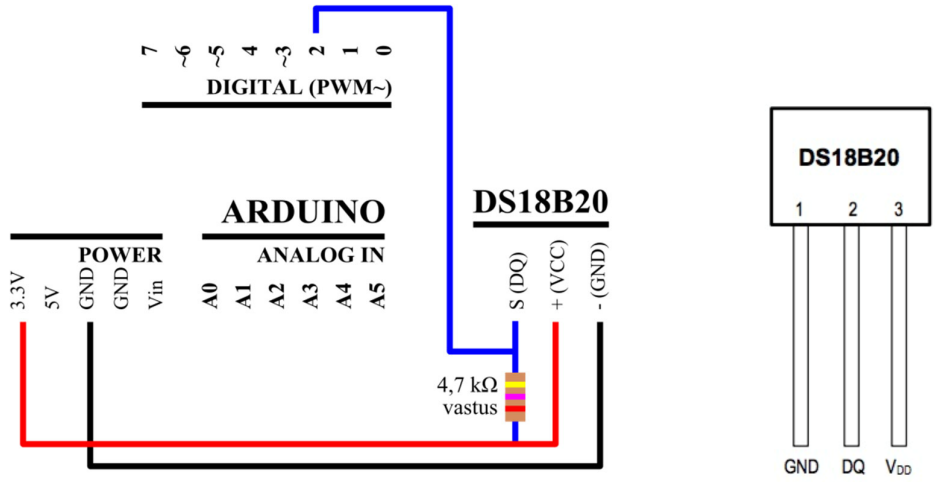

Figure 2. Connections between DS18B20 and Arduino. The pin configuration of the sensor is on the right.

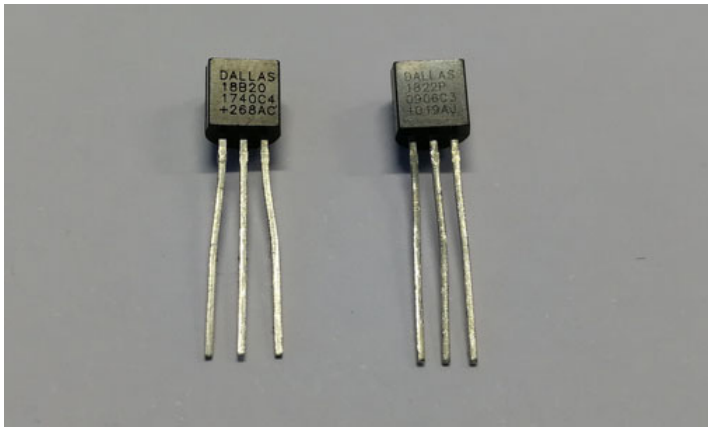

Figure 3. DS18B20 and DS1822-PAR look identical. but have differences in connectors. DS1822-PAR uses only parasite current and the third leg is just a decoration. It uses only the current coming through the pullup resistor, which also makes it slower and less accurate.

Communicating with the thermometer

The thermometer is a small-sized computer in itself! It has memory, some of which is dynamic, i.e. SRAM and EEPROM. It follows an algorithm and expects specific commands through the 1-Wire interface. This is what makes the initial steps of setting up the communications so laborious, as we need to learn the proper commands to use with this particular machine calle d DS18B20.

The commands are always found in the datasheets of each component. Thermometers are also widely used by hackers and builders, and it is easy to find proper libraries to help to get started in communications with a thermometer.

In order to read the thermometer, we need to install the OneWire-library to Arduino IDE. This can be done from *Tools>Manage Libraries...*

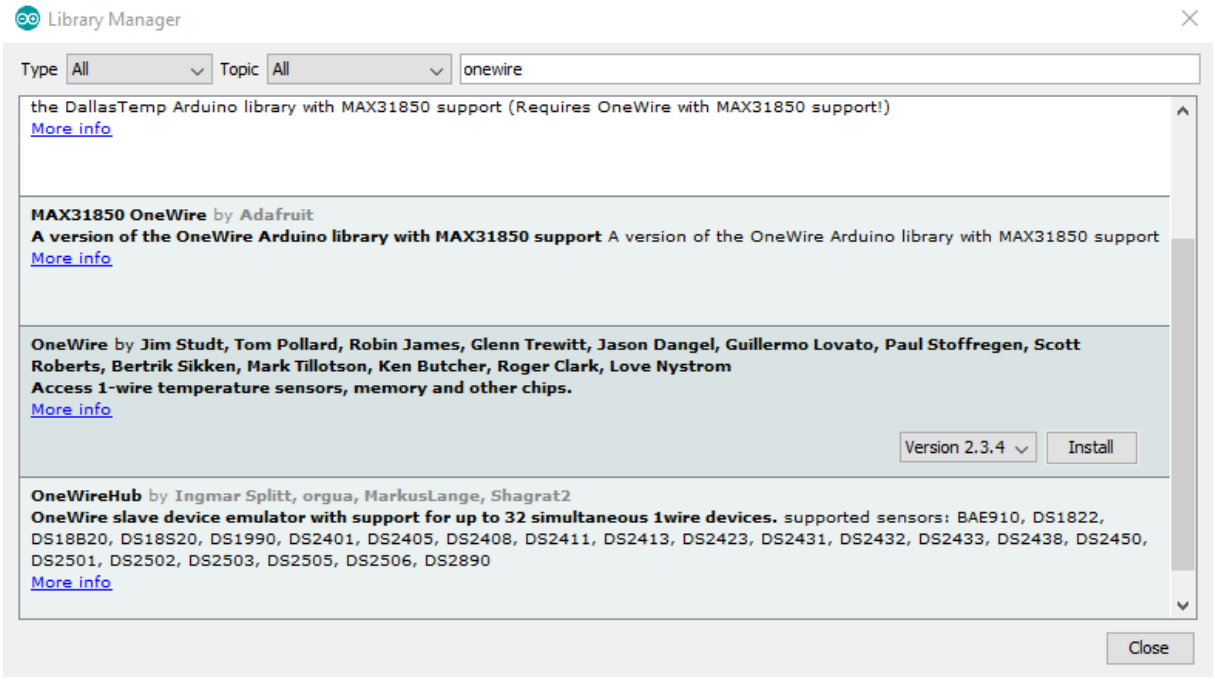

Figure 4. Installation window for libraries

And one this is done, you can find a new examples folder *File>Examples>OneWire*. Choose DS18x20. It works on multiple Dallas Semiconductor sensors.

In case the sensor is working and wires are set up correctly, you should see in the Serial Monitor something that looks like the following:

```
ROM = 28 F AB 9 A 0 0 80
  Chip = DS18B20
  Data = 1 50 1 4B 46 7F FF 10 10 49  CRC=49
  Temperature = 21.00 Celsius, 69.80 Fahrenheit
No more addresses.
```

Now you can be sure that the sensor is working. If you only get the text “No more addresses” then there are problems either with your wiring or with the sensor, which may be broken (probably due to earlier misuse).

You find a concrete example of the exercise also here:

https://create.arduino.cc/projecthub/everth-villamil-ruiz/temperature-sensor-ds18b20-3decfc

That example uses also the DallasTemperature.h library, which makes it easier to get data from different sensors, in case you want to build a larger network of temperature sensors with the OneWire bus. Alike the OneWire library, you need to install the library first from *Tools>Manage Libraries…*

Proximity Sensor

The Ultrasonic proximity sensor (LV-MaxSonar-EZ1, Maxbotix) uses ultra-sound to make measurements of the free distance in front of the device. It transmits a signal at 42kHz and measures the time for the signal to reflect back to the sensor.

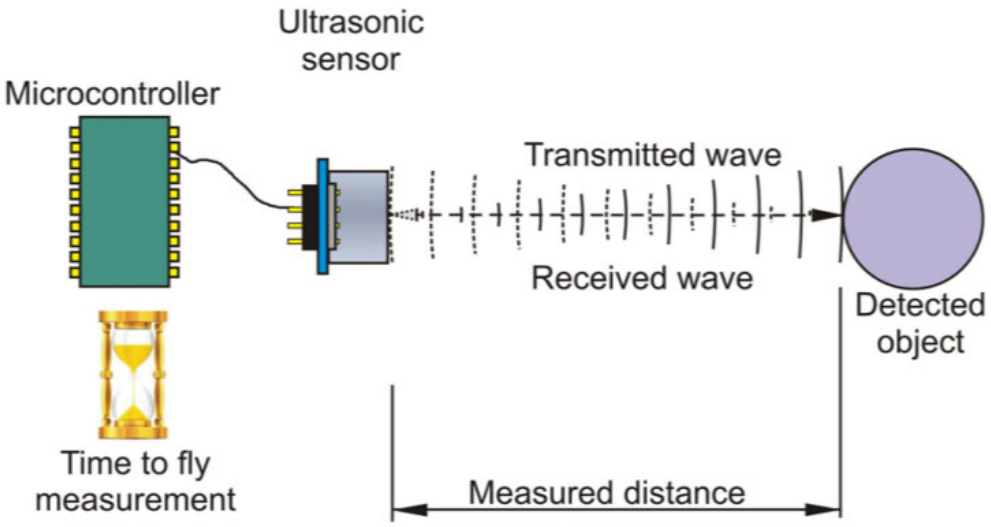

Figure 5. Ultrasonic sensor working principle. Image from [1].

The sensor has outputs for analog, digital and PWD signals. We shall use the analog output in this exercise.

Connections

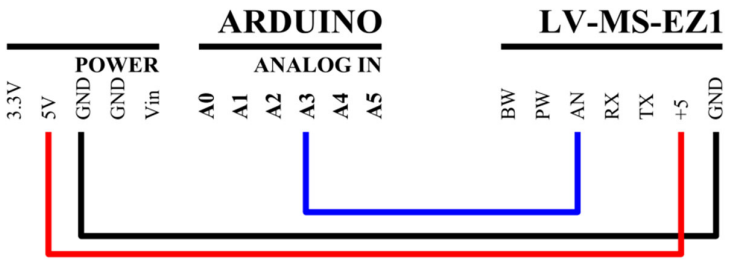

Figure 6. Connections between the ultrasonic sensor and Arduino.

The code for this exercise is simple for getting the values from the sensor. The challenge relies in translating the sensor values through Arduino's 10-bit AD to centimetres. In the datasheet (https://www.maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf) Maxbotix informs about the output in terms of inches:

> *"**Pin 3-AN**- Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data."*

An inch is 2.54 cm. Therefore, we need to do the translation from these inch-based calculations to cm. For a centimetre, the voltage difference/cm is calculated as follows for the 5V supply:

> 9.8mV/2.54cm ~ 3.86mV/cm

However, information on the manufacturer's site: https://www.maxbotix.com/ultrasonic-sensor-hrlv-maxsonar-ez-guide-158 states:

> "*The formula for the voltage scaling on an HRLV-MaxSonar-EZ is:*
>
> ```
> [(Vcc/1024) = Vi]
> Vcc = Supplied Voltage
> Vi = Volts per 5 mm (Scaling)
> ```
>
> *Example 1: Say you have an input voltage of +5.0V the formula would read:*
>
> ```
> [(5.0V/1024) = 0.004883V per 5 mm = 4.883mV per 5 mm]"
> ```

This would give us a ~9.8mV/cm. Apparently inches and centimetres have been confused even by the manufacturer's representatives! Let us trust the datasheet and 9.8mV/inch.

What is the voltage resolution of our Arduino's AD?

In the voltage range is 0-5V the analog-digital converter of Arduino can produce 1024 values, thus resulting in max. resolution of 5/1024V ~ 5mV. Based on this, the accuracy could be theoretically roughly 2cm, because 3.86mV of one cm is too little to be recognised with certainty. 2 cm gives us a double value of ~7.7mV voltage difference, which should be recognisable.

This, however, will not be the case but in only some parts of the range for the following reasons. Firstly, the minimum distance that the sensor reports is according to the datasheet is 6-inches (15.2 cm). Below that distance we will get the same result, which may be something else than 0, which is equivalent for the 6-inch reading. Between 6-20-inches there may occur acoustic phase cancellation, whereby, the resulting waveform may suffer from inaccuracies of up to 2-inches.

To complicate the calculation even further, we can see that the maximum output of the sensor is not 5V, but about half of it, if we rely on the scaling factor of (Vcc/512) per inch. The maximum range of the sensor is 254 inches. 254*5/512 ~2.48V. Therefore, we only can use half of the capacity of the resolution of Arduino's AD. This reduces the accuracy to some 4 cm – if we only trust the information from the manufacturer and our knowledge about Arduino.

Let us try to create a code that gives a translation of the input from the sensor as volts:

Code can be found here: https://codepad.co/snippet/wHxnsQYc

```
int inputPin = A3;
int inputValue = 0;
float volts = 0;

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

void loop() {
```

```
  inputValue = analogRead(inputPin);

  Serial.print("Input value ");
  Serial.println(inputValue);
  Serial.print("Input value in Volts: ");
  volts = (float)(inputValue*5)/1024.0;
  Serial.println(volts);

  delay(500);
}
```

Do we get anywhere near the maximum value 1023, even if we point the sensor towards an open space beyond the sensor's limit of 6.45m? Based on the reasoning above, if the maximum theoretical input from the sensor is ~2.48V, it is 49.6% of the range up to 5V (i.e. 2.48/5 ~ 0.496). If we take 49.6% out of 1023, it gives us ~507. Did you get max. values around 500? If so, then our reasoning has worked out! And thus, we can calculate a scaling factor for our measures. 507/645cm ~0.79 units per 1 cm. The scaling factor would thus be 1/0.79 ~1.27.

Let us confirm this with an empirical test.

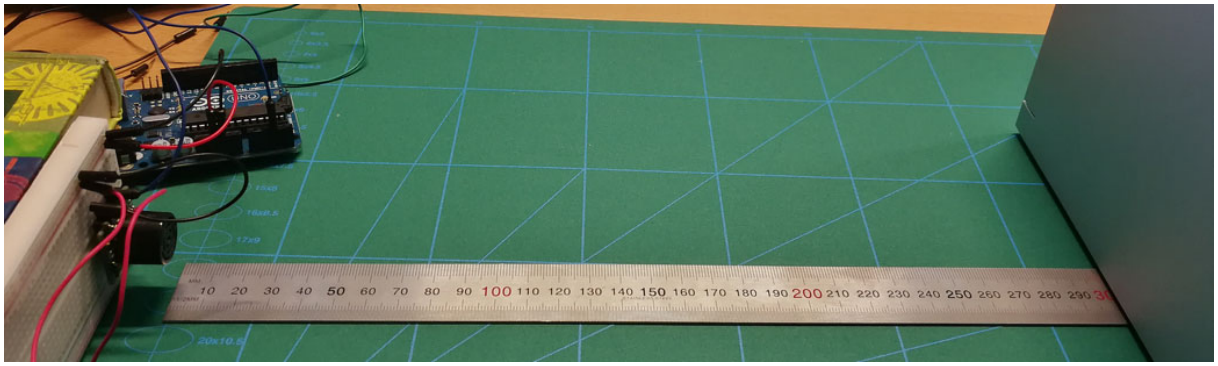

Figure 7. Empirical test of the proximity sensor with a liner.

Take a liner or measurement tape and put a block at a particular range, e.g. 30 cm from the sensor. What values do we get? I got the value 21. Now we know that 21 corresponds 30cm. This would let us assume that the scaling factor is 30/21 ~1.43. Let us do another measurement from 50 cm. What values? I got 35. And 50/35 ~1.43. Consistent! Nice.

Let us now try to create a code that calculates distance in centimetres. The code can be found here: https://codepad.co/snippet/Ie4mWPxi

We get the centimetres now simply by multiplying the input value with the scaling factor of 1.43.

NOTE: If this scaling factory actually works, then it is likely that the maximum input value is around 450, corresponding to voltage around (450/1024) 44% of 5V ~2.2V.

References

[1] M. Kelemen *et al.*, "Distance Measurement via Using of Ultrasonic Sensor," *J. Autom. Control*, vol. 3, no. 3, pp. 71–74, 2015.