Ministry of Education, Culture and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automatic Engineering

REPORT

Laboratory Project nr.2
at Embeded Systems

Done by:
Gr.172-FAF Ion Dodon

Checked by: Andrei Bragarenco

Chișinău 2020

Laboratory Project Nr.2

Topic: Sensors

Objective: To gain skills on implementing on gathering information from the sensors and to filter the information based on the following two filters: median filter and weighted average filter..

Domain: In the broadest definition, a **sensor** is a device, module, machine, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor. A sensor is always used with other electronics.

Sensors are used in everyday objects such as touch-sensitive elevator buttons (tactile sensor) and lamps which dim or brighten by touching the base, besides innumerable applications of which most people are never aware. With advances in micromachinery and easy-to-use microcontroller platforms, the uses of sensors have expanded beyond the traditional fields of temperature, pressure or flow measurement,[1] for example into MARG sensors. Moreover, analog sensors such as potentiometers and force-sensing resistors are still widely used. Applications include manufacturing and machinery, airplanes and aerospace, cars, medicine, robotics and many other aspects of our day-to-day life. There are a wide range of other sensors, measuring chemical & physical properties of materials. A few examples include optical sensors for Refractive index measurement, vibrational sensors for fluid viscosity measurement and electro-chemical sensor for monitoring pH of fluids.

Some types of sensors:

Temperature Sensors

Temperature sensors measure the amount of heat energy in a source, allowing them to detect temperature changes and convert these changes to data. Machinery used in manufacturing often requires environmental and device temperatures to be at specific levels. Similarly, within agriculture, soil temperature is a key factor for crop growth.

Humidity Sensors

These types of sensors measure the amount of water vapor in the atmosphere of air or other gases. Humidity sensors are commonly found in heating, vents and air conditioning (HVAC) systems in both industrial and residential domains. They can be found in many other areas including hospitals, and meteorology stations to report and predict weather.

Ultrasonic sensor

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e. the sound that humans can hear). Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has travelled to and from the target).

Component description:

Arduino UNO Rev3 - The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable.

VSM Virtual Terminal - The VSM Virtual Terminal enables you to use the keyboard and screen of your PC to send and receive RS232 asynchronous serial data to and from a simulated microprocessor system.
It is especially useful in debugging where you can use it to display debug/trace messages generated by the software which you are developing.
The Virtual Terminal is specified as follows:
Fully bi-directional - received serial data is displayed as ASCII characters whilst key presses are transmitted as serial ASCII data.
Simple two wire serial data interface: RXD for received data and TXD for transmitted data.
Simple two wire hardware handshake interface: RTS for ready-to-send and CTS for clear-to-send.
Baud rate from 300 to 57,600 baud.
7 or 8 data bits.
Odd, even or no parity.
0, 1 or 2 stop bits.
XON/XOFF software handshaking in addition to hardware handshaking.
Normal or inverted polarity for both RX/TX and RTS/CTS signals.

Ultrasonic Distance Sensor - HC-SR04 - Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work: (1) Using IO trigger for at least 10us high level signal, (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back. (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning. Test distance = (high level time×velocity of sound (340M/S) / 2.

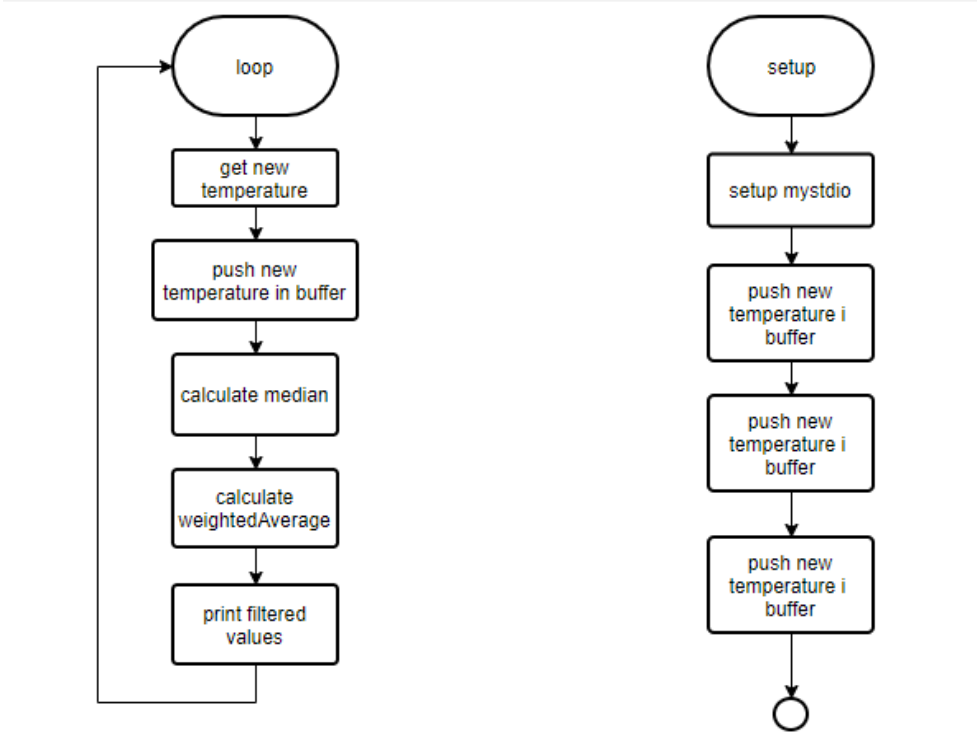
LM35 - The LM35 series are precision integrated-circuit temperature devices with an output voltage linearlyproportional to the Centigrade temperature. The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range. Lower cost is assured by trimming and calibration at the wafer level. The low-output impedance, linear output, and precise inherent calibration of the LM35 device makes interfacing to readout or control circuitry especially easy. The device is used with single power supplies, or with plus and minus supplies. As the LM35 device draws only 60 μA from the supply, it has very low self-heating of less than 0.1°C in still air. The LM35 device is rated to operate over a −55°C to 150°C temperature range, while the LM35C device is rated for a −40°C to 110°C range (−10° with improved accuracy). The LM35-series devices are available packaged in hermetic TO transistor packages, while the LM35C, LM35CA, and LM35D devices are available in the plastic TO-92 transistor package. The LM35D device is available in an 8-lead surface-mount small-outline package and a plastic TO-220 package.

Implementation:

a) Temperature sensor LM35
https://drive.google.com/open?id=14c0-66ZRgrdMs2tVyE7SZQwV4bIWQIln

b) Distance sensor HC-SR04
https://drive.google.com/open?id=1kG8T4UJxriyq4Mk4ftbKrPZAuL1U5oNr

The following flowcharts are available for both codes. For the LM34 sensor and for the Ultrasonic HC-SR04 Distance Sensor



The algorithms to perform filtering:

```
void pushTemp(int *arr, int newTemp) {
    arr[0] = arr[1];
    arr[1] = arr[2];
    arr[2] = newTemp;
}

void swap(int &a, int &b) {
    int c = a;
    a = b;
    b = c;
}

void sortArr(int *arr) {
    if(arr[0] > arr[1]) swap(arr[0], arr[1]);
    if(arr[1] > arr[2]) swap(arr[1], arr[2]);
    if(arr[0] > arr[1]) swap(arr[0], arr[1]);
}

void calculateMedian(int *rawBuff, int *medianBuff) {
    sortArr(rawBuff);
    pushTemp(medianBuff, rawBuff[1]);
}

int getWeighterAverage(int *medianBuff) {
    int weight0 = 50, weight1 = 25, weight2 = 10;
    return (medianBuff[0] * weight0 + medianBuff[1] * weight1 + medianBuff[2] * weight2)
            /
           (weight1 + weight1 + weight2);
}
```

Conclusions: Working on this laboratory work I have managed to read data from two sensors that give analog data and to filter this data independently through two filters: median filter for resolving the problem with salt and piper noise and the other one – the weighted average filter. Both filters are used to make the data stream smoother.

Annex:

1) main.cpp for the ultrasonic sensor

```
#define ULTRASONIC_TRIG 3
#define ULTRASONIC_ECHO 2
#define DELAY_TIME 200

int *rawBuff = new int[3];
int *medianBuff = new int[3];

Mystdio mystdio;
Ultrasonic ultrasonic(ULTRASONIC_TRIG, ULTRASONIC_ECHO);

void setup()
{
  mystdio.open(StreamIO::SERIALIO);
```

```
15)  printf("6\r");
16)
17)  medianBuff[0] = rawBuff[0] = ultrasonic.read();
18)  medianBuff[1] = rawBuff[1] = ultrasonic.read();
19)  medianBuff[2] = rawBuff[2] = ultrasonic.read();
20)
21)  printf("Clean    Median      WAVG\r");
22)}
23)
24)void loop()
25){
26)  int newDist = ultrasonic.read();
27)
28)  pushTemp(rawBuff, newDist);
29)
30)  calculateMedian(rawBuff, medianBuff);
31)
32)  printf("%d    ", rawBuff[2]);
33)  printf("%d        ", medianBuff[2]);
34)  printf("%d\r", getWeighterAverage(medianBuff));
35)
36)  delay(DELAY_TIME);
37)}
```

ultrasonic.cpp

```
Ultrasonic::Ultrasonic(uint8_t trigPin, uint8_t echoPin, unsigned long timeOut) {
  trig = trigPin;
  echo = echoPin;
  threePins = trig == echo ? true : false;
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
  timeout = timeOut;
}

unsigned int Ultrasonic::timing() {
  if (threePins)
    pinMode(trig, OUTPUT);

  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
```

```
  digitalWrite(trig, LOW);

  if (threePins)
    pinMode(trig, INPUT);

  previousMicros = micros();
  while(!digitalRead(echo) && (micros() - previousMicros) <= timeout);
  previousMicros = micros();
  while(digitalRead(echo)  && (micros() - previousMicros) <= timeout);
  return micros() - previousMicros;
}

unsigned int Ultrasonic::read(uint8_t und) {
  return timing() / und / 2;
}

unsigned int Ultrasonic::distanceRead(uint8_t und) {
  return read(und);
}
```

2) main.cpp for LM35 sensor

```
#define pinoSensor 1
#define DELAY_TIME 200

Mystdio mystdio;
LM35Sensor lm35;
double *rawBuff = new double[3];
double *medianBuff = new double[3];

void setup() {
    mystdio.open(StreamIO::SERIALIO);
    printf("%d\r", 3);

    medianBuff[0] = rawBuff[0] = lm35.getCelsius();
    medianBuff[1] = rawBuff[1] = lm35.getCelsius();
    medianBuff[2] = rawBuff[2] = lm35.getCelsius();

    printf("Clean    Median      WAVG\r");
}
```

```
void loop() {
    lm35.read(pinoSensor);
    double newTemp = lm35.getCelsius();

    pushTemp(rawBuff, newTemp);

    calculateMedian(rawBuff, medianBuff);

    mystdio.printDouble(rawBuff[2]);
    printf("    ");
    mystdio.printDouble(medianBuff[2]);
    printf("        ");
    mystdio.printDouble(getWeighterAverage(medianBuff));
    printf("\r");

    delay(DELAY_TIME);
}
```

LM35.cpp

```
const double DIVIDER_HIGH_RES = 9.309;
const double DIVIDER_LOW_RES = 2.048;

LM35Sensor::LM35Sensor(void) {
  setSamples(500);
  setHighRes(false);
}

LM35Sensor::LM35Sensor(int pSamples) {
  setSamples(pSamples);
  setHighRes(false);
}

LM35Sensor::LM35Sensor(int pSamples, bool pHighRes) {
  setSamples(pSamples);
  setHighRes(pHighRes);
}

void LM35Sensor::setSamples(int pSamples) {
```

```
  samples = pSamples;
}

void LM35Sensor::setHighRes(bool pHighRes) {
  highRes = pHighRes;
  if (highRes) {
    divider = DIVIDER_HIGH_RES;
    analogReference(HIGH_RES);
  } else {
    divider = DIVIDER_LOW_RES;
    analogReference(LOW_RES);
  }
}

void LM35Sensor::read(int port) {
  double tempSum = 0.0;
  for (int i = 0; i < samples; i++) {
    tempSum = tempSum + (analogRead(port) / divider);
  }
  celsius = tempSum / samples;
  fahrenheit = 1.8 * celsius + 32;
  kelvin = celsius + 273.15;
}

double LM35Sensor::getCelsius(void) {
  return celsius;
}

double LM35Sensor::getFahrenheit(void) {
  return fahrenheit;
}

double LM35Sensor::getKelvin(void) {
  return kelvin;
}
```

mystdio.cpp

```
String Mystdio::readStr() {
  char c;
  String str;
  do {
    scanf("%c", &c);
    if(c != 0 && c != '\r'){
      str.concat(c);
    }
  }while(c != '\r');
```

```
  return str;
}

void Mystdio::writeStr(String str) {
  for(int i = 0; i < str.length(); i++) {
    printf("%c", str[i]);
  }
}

void Mystdio::printDouble(double nmb) {
  int left = (int) nmb;
  int right = ((int)(nmb * 10)) % 10;

  printf("%d.", left);
  printf("%d", right);
}

static int Mystdio::putCharSerial(char c, FILE *stream)
{
    Serial.write(c) ;
    return 0 ;
}

static char Mystdio::getCharSerial(FILE *stream)
{
    char c;
    while(!Serial.available()) {}
    c = Serial.read();
    printf("%c", c);
    return c;
}

void Mystdio::open(StreamIO streamIO) {
  if(streamIO == SERIALIO) {
    Serial.begin(9600);
    f = fdevopen(Mystdio::putCharSerial, Mystdio::getCharSerial);
  }

  stdout = f;
  stdin = f;
}

Mystdio::Mystdio() {

}
```