DOI: 10.2478/ijasitels-2021-0002

Development of a smart clothing product using an Arduino platform

Ana Brad [1], Maria Brad [1]

[1] Politehnica University Timisoara, Romania, ana.brad99@gmail.com, maria.brad99@gmail.com

Abstract

This paper presents a "smart" clothing product implemented as a jacket that contains sensors, a processing unit for display and interaction. The system has the ability to remotely read the data provided by the sensors, ensuring the monitoring of several parameters of the wearer.

The following characteristics have been considered: body temperature and humidity, atmospheric temperature, pressure and altitude, the heart beat and number of steps converted into the number of calories consumed and traveled distance.

The data is acquired and processed by an Arduino AT Mega 2560, via the I2C bus, digital ports and analog to digital converters, depending on the type of sensors. The processed information is printed on a 128x64 pixel display. To be able to view more pages of information, one can interact with the 4-key keyboard that has been connected to the digital input ports or through a proximity sensor, which will function as a gesture sensor.

The processed information can also be accessed from a web server, built on the ESP8266 Wi-Fi module, connected to Arduino's TX/RX lines. A mobile phone or another device can connect to the Access Point and open a web page which displays the values of all sensors, as well as other information. The embedded system was inserted on a jacket and wired according to the sensors and modules usage.

Keywords: smart clothing, embedded system, Arduino, wearable electronics, e-textile

1. Introduction

The concept of e-textiles refers to textile fabrics and embedded devices with the purpose of detection, communication, decision making, sensing. As textiles are washable, light, flexible, the embedded devices have to be powered, breakable and cannot be washed [1]. For the textile industry, the potential offered by combining these two fields is very large. Beginning with smart medical products (which will be preferred to the classic ones due to their multi-functionality), passing through the field of protective clothing (another field in which smart materials are and will be widely used) and finishing with interior design objects, we are certain to express that smart textiles are the future of this industry, whatever will be their application [2].
In the future, conductive materials are expected to infiltrate all areas of application, as suggested by almost all experts, especially in the interior textile industry [3]. Electronic components, sensors and actuators will be used in all categories of textiles, except for construction textiles and especially in the interior textiles and the medical textiles sector [4]. Grafting materials as well as advanced polymers will primarily be used in the clothing sector [5]. Most experts ambiguously predict the development of

textile applications intended for construction, not including the textile products that generate energy and power supply [6].

2. The proposed hardware platform

To create a prototype of a smart jacket, we have considered the following functionalities, using an Arduino-based system:

- The system will achieve measuring the following data:
    1. Indoor or body temperature
    2. Body moisture
    3. Atmospheric temperature
    4. Atmospheric pressure
    5. The altitude
    6. The heart beat
    7. The number of steps (converted to calories consumed and distance traveled)
- The system will output the data measured and processed using:
    1. An OLED display integrated in the jacket
    2. A web page accessible through the integrated Wi-Fi module

In this respect, the following sensors were used:

- SHT-21 type temperature and humidity sensor
- BMP-180 type temperature, pressure and altitude sensor
- Pulse sensor from pusesensor.com
- 3-axis accelerometer ADXL345
- ADPS-9330 ambient light sensor

The display will be made on an OLED display resolution of 128x64 pixels. Due to the number of monitored parameters and the low resolution, the display will be made on several screens. The transition from one screen to another is done through a keyboard or a proximity sensor. If this viewing function is not used, it will be doubled by connecting a computing device to the Wi-Fi access point and accessing a web page via the browser. The interconnection of the components will be detailed in the next chapter.
To monitor the pulse, an infrared sensor mounted on the index finger will track the number of beats per minute. The pulse is read on an analog port connected to an analog-to-digital converter of Arduino. The 3-axis acceleration, temperature, humidity, pressure and light sensors are connected on the I2C bus.

2.1. Wiring diagram of the system

To assure all the functions mentioned in the previous subchapter, we have connected the sensor, display, keyboard and Wi-Fi module according to the diagram shown in figure 1. We used ribbon cable for wiring routes for the I2C bus that interconnect the SCL and SDA pins on the Arduino module with those of the BMP180 sensor, SHT-21, ADXL-345, ADPS-9930 and the OLED display, located as shown in figure 2.
The keyboard was connected to digital pins D2-D6, their description being defined in the source code part. Pin D6 is held in logical "0" and the others in "pull-up" mode, thus determining the logical value of "1" when they are pressed.

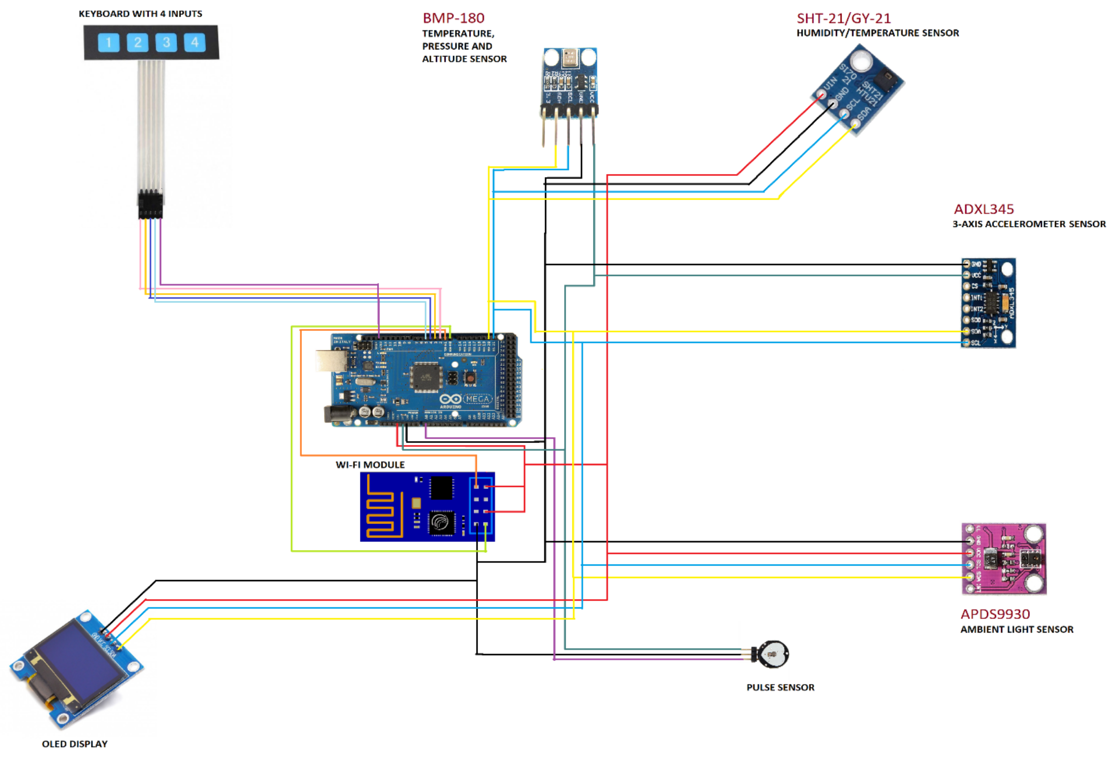


Figure 1. Electrical schematic design

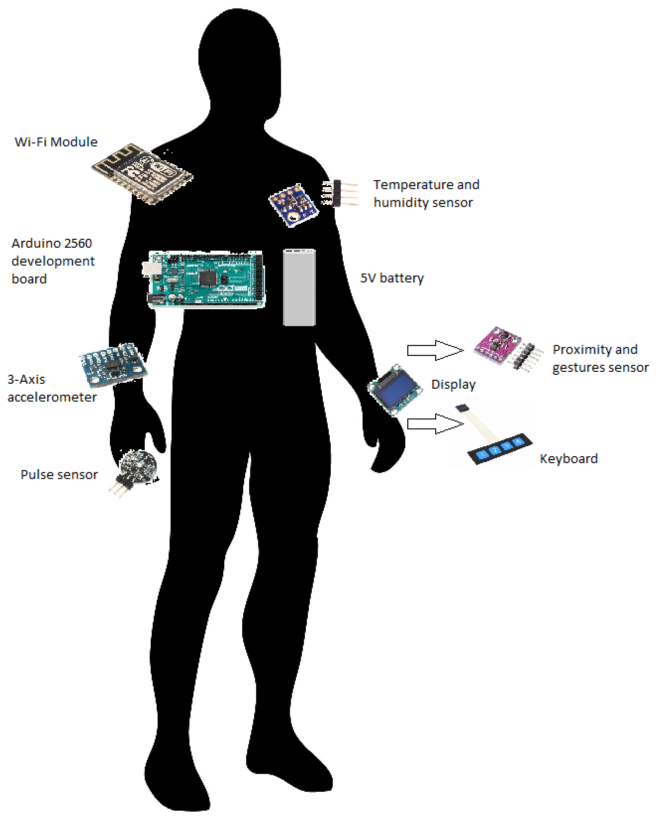

Wi-Fi Module
Temperature and humidity sensor
Arduino 2560 development board
5V battery
Proximity and gestures sensor
3-Axis accelerometer
Display
Keyboard
Pulse sensor

Figure 2. Location of the components

The Wi-Fi module communicates with Arduino through the TX and RX pins of the Serial Port 1. In this way, communication does not interfere with Serial Port 0 connected to the USB. The pulse sensor is powered at 5V and provides an analog output connected to pin A0, from which the value is converted numerically by an ADC.

2.2. Presentation of the employed components

2.2.1. Development board and display

The Arduino Mega 2560 development board [7] uses the ATmega2560 microcontroller, which has 54 digital I/O pins, 16 analog pins, 4 serial interfaces, a 16 MHz clock, a USB connector, a power connector and a push -reset button.
The OLED display module [8] can be used to display alphanumeric or graphical data with low power consumption. The resolution available for the device used is quite high, allowing the display of 128 x 64 pixels. It has a consumption of only 0.1W, a diagonal of 0.94 inches and a viewing angle greater than 160°.
Regardless of the size of the OLED display, the SSD1306 driver has a 1KB Graphic Display Data (GDDRAM) RAM that contains the bit image that will be displayed. This 1K memory area is organized into 8 pages, where a page comprises 128 columns/segments. Each column can memorize 1 byte of data (0 to 7).

2.2.2. Temperature and humidity precision sensor SHT21

The SHT21 sensor functions as a "slave" in I2C communication. At the request of the microcontroller, the sensor starts the process of measuring temperature and humidity. When the sensor has finished measuring, it sends the measured data to the microcontroller and enters idle mode to ensure low power consumption. The time spent measuring temperature and humidity varies between 5 and 30 seconds, depending on the thermal conductivity of the material with which the sensor is in contact. For example, the sensor will detect the temperature change earlier (if it is in contact with a good thermal conductor, such as metal) than when it is in contact with a thermal insulator [9].
The resolution can be set to 8/12 bits for temperature and 12/14 bits for humidity, respectively. The temperature is computed by entering the S_T output signal in the following formula 1 (resulting in °C), regardless of the resolution chosen:

$$T = -46.85 + 175.72\frac{S_T}{2^{16}} \tag{1}$$

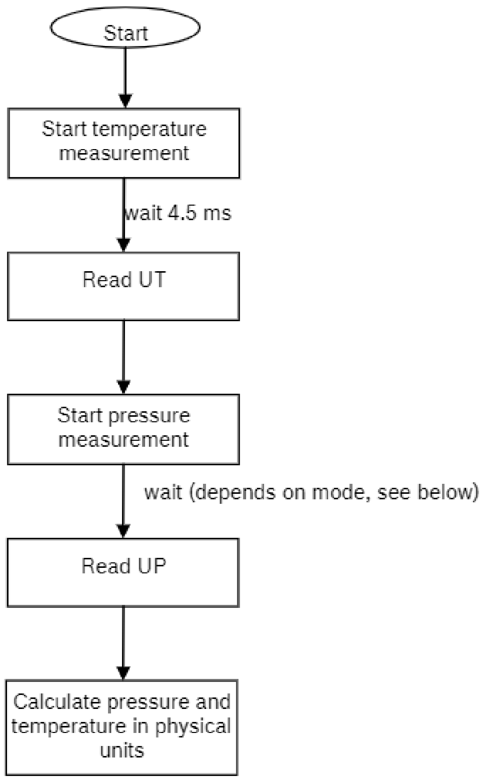
The output of the S_{RH} relative humidity signal is obtained by equation 2 (resulting in% RH), regardless of the chosen resolution:

$$RH = -6 + 125\frac{S_{RH}}{2^{16}} \tag{2}$$

Communication with the microcontroller is done via the I2C interface. The default resolution is set to 14 bits for relative humidity and 12 bits for temperature reading. The measured data is transferred in two-byte packets, i.e. in frames with a length of 8 bits where the most significant bit (MSB) is transferred first (from left to right).

2.2.3. Atmospheric pressure sensor BMP180

The BMP180 atmospheric pressure sensor module [10] is used to measure temperature, pressure and altitude, with extremely low consumption and very small dimensions. All information is transmitted using only 2 connections (SDA and SCL), to which is added the ground connection.
The microcontroller sends a start sequence to start a pressure or temperature measurement. After the conversion time, the value of the result (UP or UT respectively) can be read through the I2C interface. Calibration data must be used to compute the temperature in °C and the pressure in hPa. These constants can be read from the BMP180 E2PROM via the I2C interface when the software is initialized.


Start
Start temperature measurement
wait 4.5 ms
Read UT
Start pressure measurement
wait (depends on mode, see below)
Read UP
Calculate pressure and temperature in physical units
Figure 3. Operation diagram of the BMP180 sensor

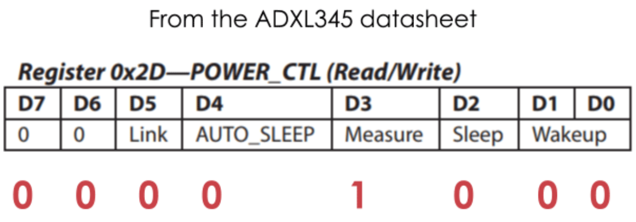
The computation of the actual temperature and pressure is done in steps of 1Pa (=0.01hPa = 0.01mbar) for the pressure and in steps of 0.1 °C for the temperature. The communication with the microcontroller is made through the I2C interface.
The barometric pressure changes depending on the altitude of the sensor. A change of 1 hPa in atmospheric pressure corresponds to a change in altitude of about 8 meters.
Altitude can be calculated using the international barometric formula in 3:

$$altitude = 44330\left(1-\left(\frac{P}{P_0}\right)^{\frac{1}{5.255}}\right) \qquad (3)$$

To calculate the altitude at the current location in relation to the sea level, the variable P_o is assigned the value of 1013.0 and P is the measured pressure at location.

2.2.4. 3-axis I2C ADXL345 accelerometer module

ADXL345 [11] is used for measurements of static gravity acceleration in applications that capture tilting, as well as dynamic acceleration resulting from motion or shocks. It measures the static acceleration of gravity in tilt detection applications, as well as the dynamic acceleration resulting from motion or shock. Its high resolution (4 mg / LSB) allows highlighting the inclination changes of 0.25°. The ADXL345 has a resolution (13 bits) of up to ± 16 g. Digitally, the output data is formatted as a two-bit by 16-bit complement. Communication with the microcontroller is done via the I2C interface.
Each device that uses I2C communication has a unique I2C address and this address can be found in the sensor data sheet. Once the address and variables for the three outputs are defined, in the configuration section, we must first initialize the working library with I2C (wire.h) and then set the accelerometer in the measurement mode. To do this, if we look at the data sheet again, we can see that we need to set bit D3 of the register POWER_CTL to HIGH.



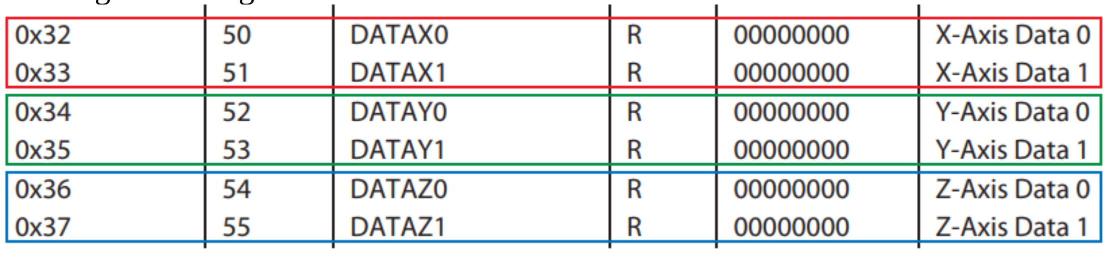
From the ADXL345 datasheet

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	
0	0	0	0	1	0	0	0

Figure 4. Setting the register flags for data measurement

The data for each axis is stored in two bytes or registers. One can see the addresses of these registers in figure 5.



0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1

Figure 5. Register addresses of the ADXL345

2.2.5. Pulse sensor

The pulse sensor [12] is a plug-and-play sensor, being designed to work together with a development board equipped with at least one analog pin. This product is useful for collecting heart rate data in various situations, such as during exercise.
It consists of an LED and a circuit. It works on the principle that when blood flows inside the veins, the light coming from the LED is reflected, detecting the heart rate. In order to obtain reliable measurements of the time between each rhythm, a regular sampling rate with a sufficiently high resolution is required. To do this, Timer2, an 8-bit hardware timer on ATMega, is configured to perform an interrupt every two milliseconds. It gives us a sampling rate of 500Hz and a beat-to-beat sync resolution of 2 ms.
The analog signal is sent by the sensor after a filtering and amplification is transformed into numerical value through the input pin A0 through the Arduino

analog-to-digital converter. The analog signal is directly proportional to the infrared light intensity reflected by the skin surface and is in the range 0.3 V - 5 V.

2.2.6. Ambient light and proximity sensor APDS-9930

The APDS-9930 [13] is used for ambient light detection and a complete single-chip proximity detection system. The proximity detection function is activated by strong sunlight up to low ambient light. The wide dynamic range also allows operation in a low power mode between ALS (Ambient Light Sense) and proximity measurement.
The APDS-9930 offers I2C communication in a single 8-pin package. The ambient light sensor uses a dual LED to approximate 0.01 lux. The proximity sensor can detect objects at up to 100mm. It works perfectly both in sunlight and in dark rooms. It has micro-optical lenses to provide efficient infrared transmission and reception.

2.2.7. Wi-Fi module ESP8266

ESP8266 [14] is a low-cost Wi-Fi module that belongs to the ESP family, which you can use to control your electronic projects anywhere in the world. It has a built-in microcontroller and a 1 MB flash memory that allows it to connect to a Wi-Fi. The TCP / IP protocol stack allows the module to communicate with Wi-Fi signals. The maximum working voltage of the module is 3.3 V.
We will configure ESP8266 through its serial interface, using AT commands. The Arduino program must generate AT commands to reset the Wi-Fi module ("AT + RST"). The next step is to configure this module as a Wi-Fi access point ("AT + CWMODE = 2"). After this step, read the IP address of the module, 192.168.4.1, using the command: "AT + CIFSR", which will also print the MAC address. To obtain the connection information (SSID), we will run the command ("AT + CWSAP?"): the network name and password will be returned (by default there is no password), and then we can configure the system to accept multiple connections ("AT + CIPMUX = 1“) and we will start the web server on port 80 ("AT + CIPSERVER = 1.80"). Each AT command must end with carriage return and newline ("\ r \ n").

3. Programming the Arduino and Wi-Fi module

The specific Arduino code generally contains two functions: `setup ()` and `loop ()`. The first of these will be executed when the system is turned on or reset. The second, as its name suggests, will be executed indefinitely. The simplicity of this way of working made it accessible to all categories of programmers, but also for building applications quickly.
For the most part, Arduino has a continuous and free development from the programmer community, offering libraries for most existing components and sensors on the market. In order to shorten the execution time in the main loop, we used for most of the sensors and the display system the libraries offered by the manufacturer, without the need to manage the I2C communication with the device. In the case of the 3-axis acceleration sensor, we tried the direct use of the command, status and data registers.

From the following subchapters, it will be seen that we used the default functions specified above, to which we had added others, necessary to read the data from the sensors or to communicate with web clients.

3.1. Macro definitions

First, we detailed the part of library statements, global variables, and library objects, and then described the setup function, the loop function, and the additionally added ones. Statements of global variables were required.
The inclusion in the compilation process of the libraries specific to the input-output devices was necessary as well, but also of the objects, where appropriate. The source code has been commented on accordingly to provide the necessary details.

```
#include <Adafruit_SSD1306.h> //THE LIBRARY FOR THE OLED DISPLAY
#include <Adafruit_GFX.h>     // THIS LIBRARY HAS A SET OF
//GRAPHICAL FUNCTIONS FOR THE OLED DISPLAY
#include <APDS9930.h>   // LIBRARY FOR THE ENVIRONMENTAL LIGHT SENSOR
#include <SHT21.h>     // TEMPERATURE AND HUMIDITY SENSOR LIBRARY
#include <Adafruit_BMP085.h> // LIBRARY FOR ATM.PRES./TEMP.SENSOR
#include <PulseSensorPlayground.h>     // PULSE SENSOR LIBRARY
#include <Fonts/FreeMonoBold12pt7b.h>  // ADDED A PREFERENTIAL FONT
#include <Fonts/FreeMono9pt7b.h>       // ADDED A PREFERENTIAL FONT
#define OLED_Address 0x3C      //0X3C IS THE ADDRESS FOR I2C OLED
#define USE_ARDUINO_INTERRUPTS true // SETTING OF THE SWITCHES FOR
BPM

Adafruit_SSD1306 display(128, 64);  // WE INITIALIZE A DISPLAY
APDS9930 apds = APDS9930();        // WE INITIALIZE A LIGHT SENSOR
SHT21 sht;                   //INIT.TEMPERATURE AND HUMIDITY SENSOR
Adafruit_BMP085 bmp;            // INIT. PRESS./TEMP. SENSOR
PulseSensorPlayground pulseSensor;     // INIT. A PULSE SENSOR
//  GLOBAL VARIABLES
int regAddress = 0x32;    // FIRST ADXL345 DATA REGISTER
const int PulseWire = 0;  // PULSEWIRE SIGNAL WAS CONNECTED TO PIN 0
int Threshold = 516;      // THE SIGNAL THRESHOLD AS A "BEAT"
int state=0;
float TempC, TempA;        //VALUES FOR TEMP., PRESS.,HUMID.
float Calorii, Distanta;
int Humidity, Pressure, Altitude, Pasi;
INT DEL=35;    //READING DELAY
INT myBPM;
BYTE BUFF[TO_READ] ;  //6 bytes BUFFER, READ FROM THE ACCELEROMETER
CHAR str[512];        //BUFFER FOR DATA SENT TO THE SERIAL PORT
INT x,y,z;        //VALUES FROM THE ACCELEROMETER
INT xavg, yavg,zavg, steps=0, flag=0;
INT xval[15]={0}, yval[15]={0}, zval[15]={0};
// KEYS
#define key1 2 //KEY 1 WAS CONNECTED TO THE DIGITAL PIN 2
#define key2 3 //KEY 2 WAS CONNECTED TO THE DIGITAL PIN 3
#define key3 4 //KEY 3 WAS CONNECTED TO THE DIGITAL PIN 4
#define key4 5 //KEY 4 WAS CONNECTED TO THE DIGITAL PIN 5
#define low6 6 //PIN 5 KEYB. WAS CONNECTED TO THE DIGITAL PIN 6
// PEDOMETER
#define DEVICE (0x53) // ADDRESS. FOR THE ADXL345 ACCELEROMETER
#define TO_READ (6)   //NUMBER OF BYTES READ, 2 BYTES PER AXIS
#define offsetX   -10.5 // OFFSET
#define offsetY   -2.5
```

```
#define offsetZ   -4.5
#define gainX     257.5 // GAIN
#define gainY     254.5
#define gainZ     248.5
```

3.2. Initialization loop

In this function we will set the communication on the I2C bus, we initialize the sensors with the help of the previously declared objects, we define the Arduino ports used, but also the communication on the serial bus to which the Wi-Fi module connects.

```
void setup()
{
  Wire.begin();          //INIT. OF THE COMMUNICATION ON I2C
  Wire.setClock(400000); //SETTING OF THE I2C BUS CLOCK AT 400 KHZ
  // INIT.THE AMBIENT LIGHT SENSOR AND THE ATM.PRESSURE/TEMP.SENSOR
  apds.init();
  bmp.begin();
  // DIGITAL PINS FOR THE KEYBOARD
  pinMode(key1, INPUT_PULLUP);// SETTING OF PIN 2 AS input
  pinMode(key2, INPUT_PULLUP);// SETTING OF PIN 3 aS input
  pinMode(key3, INPUT_PULLUP);// SETTING OF PIN 4 aS input
  pinMode(key4, INPUT_PULLUP);// SETTING OF PIN 5 aS input
  // INITIALIZATION OF ADXL345
  writeTo(DEVICE, 0x2D, 0); //MEASUREMENT START ORDER
  writeTo(DEVICE, 0x2D, 16);
  writeTo(DEVICE, 0x2D, 8);//BIT D3 HIGH FOR MEASUREMENT IS ENABLED
  // WE CONFIGURE THE PULSESENSOR OBJECT, WITH OUR OWN VARIABLES
  pulseSensor.analogInput(PulseWire);
  pulseSensor.setThreshold(Threshold);
  pulseSensor.begin();
  // TURNING ON THE APDS-9930 LIGHT SENSOR
  apds.enableLightSensor(false);
  // WE OPEN SERIAL COMMUNICATION FOR THE WIFI MODULE
  Serial1.begin(115200);
  wifi_init();
  delay(100);

  // INITIALIZATION OF THE DISPLAY
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();  // DELETE THE BUFFER
  display.setTextColor(WHITE);  // TEXT COLOR
  display.setRotation(0);  // SET THE ORIENTATION
  display.setTextWrap(false);  // LONG LINES OF TEXT WITHOUT wrap
  display.dim(0);  // BRIGHTNESS SETTING (0 IS HIGH, 1 CLOSER)
  display.setFont(&FreeMono9pt7b);  // SETTING of A CUSTOMIZED font
  display.setTextSize(0);  // SETTING TEXT SIZE (0 IS A CUSTOM FONT)
  // WAITING FOR THE INITIALIZATION AND CALIBRATION TO FINISH
  delay(500);
      }
```

3.3. The main loop

Within this function, which is executed infinitely, we will perform sequentially all the operations of reading, converting and displaying the data. Calls will be made to functions that will be detailed in the following subchapters, such as `Read_ambient_light ()`, with the help of which we detect a swipe gesture.

```
void loop()
{
  read_ambient_light();  // CALL THE GESTURE DETECTION FUNCT.
  // THESE FUNCTIONS ARE USED TO GET TEMPERATURE,
  // MOISTURE, ALT., PRESS. VALUES
  TempC =     sht.getTemperature();
  Humidity = sht.getHumidity();
  TempA = bmp.readTemperature();
  Pressure = ceil(bmp.readPressure()/100);
  Altitude = bmp.readAltitude();

  read_key();   // CHECK IF THE KEY IS PRESSED
  display.clearDisplay();  // clear THE DISPLAY TO REFRESH

  // THE READING OF THE ACCELEROMETER   DATA
  readFrom(DEVICE, regAddress, TO_READ, buff);

  // EACH AXIS READING HAS A RESOLUTION OF 10 BITS (2 BYTES)
  // (THE LSB IS THE FIRST), WE CONVERT BOTH BYTES TO INT
  x = (((int)buff[1]) << 8) | buff[0];
  y = (((int)buff[3])<< 8) | buff[2];
  z = (((int)buff[5]) << 8) | buff[4];
  Pasi = Pedometer(); // IT IS DETERMINED IF A STEP WAS TAKEN
  Calorii=0.045*Pasi; // NUMBER OF CALORIES CONSUMED
  Distanta=Pasi*0.6;  // CALCULATION OF THE DISTANCE

  // THIS FUNCTION OF THE PULSE SENSOR RETURNS THE NUMBER OF
  // BEATS PER MINUTE ON AN INT
  myBPM = pulseSensor.getBeatsPerMinute();
  ...
```

3.4. Displaying the sensors data

This part of the program can also be found in the main loop. The acquired data is displayed on different screens and for this reason it is presented separately. The global variable `STATE` is used to find on which of the 4 screens it will be displayed. Its value changes either inside the function that reads the input keys or inside the function that determines the change of light intensity detected by the light sensor.

```
SWITCH(STATE) // DEPENDING ON THE KEYS OR SWIPE GESTURE
  {
    CASE 0:  // THE FIRST SCREEN BODY TEMPERATURE AND HUMIDITY
      {
      DISPLAY.SETCURSOR(2, 15);  // POSITIONING THE CURSOR AT (X, Y)
      DISPLAY.PRINT("TEMP:");
      DISPLAY.PRINT(TEMPC);
      DISPLAY.SETCURSOR(2, 35);
      DISPLAY.PRINT("HYGR:");
      DISPLAY.PRINT(HUMIDITY);
      DISPLAY.DRAWRECT(0, 0, 127, 63, WHITE); // DRAWING A RECTANGLE
      BREAK;
      }
    CASE 1:   // THE SECOND SCREEN ENVIRONMENT TEMP., PRESS. AND ALT.
      {
      DISPLAY.SETCURSOR(2, 15);
      DISPLAY.PRINT("TEMP:");
      DISPLAY.PRINT(TEMPA);
      DISPLAY.SETCURSOR(2, 35);
      DISPLAY.PRINT("PRES:");
      DISPLAY.PRINT(PRESSURE);
      DISPLAY.SETCURSOR(2, 55);
      DISPLAY.PRINT("ALT: ");
      DISPLAY.PRINT(ALTITUDE);
      DISPLAY.DRAWRECT(0, 0, 127, 63, WHITE);
      BREAK;
      }
    CASE 2:   // THE THIRD SCREEN. PEDOMETER DATA
      {
      DISPLAY.SETCURSOR(2, 15);
      DISPLAY.PRINT("PASI:");
      DISPLAY.PRINT(PASI);
      DISPLAY.SETCURSOR(2, 35);
      DISPLAY.PRINT("CAL:");
      DISPLAY.PRINT(CALORII);
      DISPLAY.SETCURSOR(2, 55);
      DISPLAY.PRINT("DIST:");
      DISPLAY.PRINT(DISTANTA);
      DISPLAY.DRAWRECT(0, 0, 127, 63, WHITE);
      BREAK;
      }
    CASE 3:   // THE FOURTH SCREEN. PULSE DATA
      {
      DISPLAY.SETCURSOR(2, 15);
      DISPLAY.PRINT("BPM:");
      DISPLAY.PRINT(MYBPM);
      // TEST IF WE DETECT A HEARTBEAT
      IF (PULSESENSOR.SAWSTARTOFBEAT()) {
          DISPLAY.SETCURSOR(2, 35);
      // IF SAWSTARTOFBEAT FUNCTION RETURNS TRUE, DISPLAY A MESSAGE
          DISPLAY.PRINT("((H))♥");
          }
      DISPLAY.DRAWRECT(0, 0, 127, 63, WHITE);
      BREAK;
      }
  }
  DISPLAY.DISPLAY();  // DISPLAYING THE CURRENT DISPLAY
}
```

3.5. The Key reading function

With this function we change the state of the global variable STATE to determine what data will be displayed on the screen. Only three of the four keyboard keys are operational, as follows:

- Key 1 is used to decrease the status ("down")
- Key 2 is used to increase the status ("up")
- Key 3 resets the number of steps

The state variable is considered to have modulo 4 values and therefore its state is cyclical.

```
VOID READ_KEY()
{
  INT KEY1S = DIGITALREAD(KEY1);  // READING IF KEY1 IS PRESSED
  INT KEY2S = DIGITALREAD(KEY2); // READING IF KEY2 IS PRESSED
  INT KEY3S = DIGITALREAD(KEY3); // READING IF KEY3 IS PRESSED
  INT KEY4S = DIGITALREAD(KEY4); // READING IF KEY4 IS PRESSED
  IF(!KEY1S) // IF THE KEY1 HAS BEEN PRESSED, GO TO THE NEXT PAGE
    ++STATE;
  IF(!KEY2S) // IF THE KEY2 HAS BEEN PRESSED, GO TO THE PREVIOUS PAGE
    --STATE;
  IF (STATE<0)// FROM THE FIRST PAGE TO PAGE 3
    STATE=3;
  IF (STATE>3)// IF ON PAGE 3 THE KEY1 HAS BEEN PRESSED, GO TO PAGE 0
STATE=0;
  IF(!KEY4S) // IF THE KEY4 HAS BEEN PRESSED, STEPS COUNTER IS RESET
    STEPS=0;
}
```

3.6. The gesture detection function using the ambient light sensor

To simulate a gesture sensor using an ambient light sensor, mirroring the swipe function that phones or other mobile devices have, we have implemented the below function. It reads 5 consecutive values from the APDS 9930 light sensor and calculates their normalized derivative to determine if a variation has occurred. The delay between two successive readings (the `del` variable) defines the speed of the hand movement facing the sensor. Which each detection, the value of the state variable is incremented, which indicates the screen that will be displayed.

```
VOID READ_AMBIENT_LIGHT()
{
  INT I;
  FLOAT STD=0, SUM=0;
  FLOAT AMBIENT_LIGHT_T[5];

  // READ 5 CONSECUTIVE LIGHT LEVELS
  APDS.READAMBIENTLIGHTLUX(AMBIENT_LIGHT_T[0]);
  DELAY(DEL);
  APDS.READAMBIENTLIGHTLUX(AMBIENT_LIGHT_T[1]);
  DELAY(DEL);
  APDS.READAMBIENTLIGHTLUX(AMBIENT_LIGHT_T[2]);
  DELAY(DEL);
  APDS.READAMBIENTLIGHTLUX(AMBIENT_LIGHT_T[3]);
  DELAY(DEL);
  APDS.READAMBIENTLIGHTLUX(AMBIENT_LIGHT_T[4]);

  // SUM = THE SQUARE OF THE FIRST READ VALUE
  SUM=POW(AMBIENT_LIGHT_T[0],2);
  FOR(I=1; I<5; I++)
    {
   // STD=THE SUM OF THE SQUARES OF DIFFERENT CONSECUTIVE VALUES
   STD=STD+POW((AMBIENT_LIGHT_T[I]-AMBIENT_LIGHT_T[I-1]),2);
    // THE CALCULATING OF THE SUM OF THE SQUARES OF THE READ VALUES
    SUM=SUM+POW(AMBIENT_LIGHT_T[I],2);
    }
  // WE CALCULATE THE FIRST NORMALIZED DERIVATIVE FOR PEAK DETECTION
  STD=STD/SUM;

  // IF IT IS GREATER THAN 0.14, GO TO THE NEXT PAGE
  IF (STD>0.14) // EMPIRIC DETERMINED THRESHOLD
    STATE=++STATE;

  // THE INDEX OF THE LAST PAGE IS 3 AND THE STATE WILL BE RESET
  IF(STATE==4);
    STATE=0; // GO TO THE FIRST PAGE
}
```

The variations of the ambient light intensity are determined and, also, the presence of some peaks in these variations, as can be seen from figure 6.

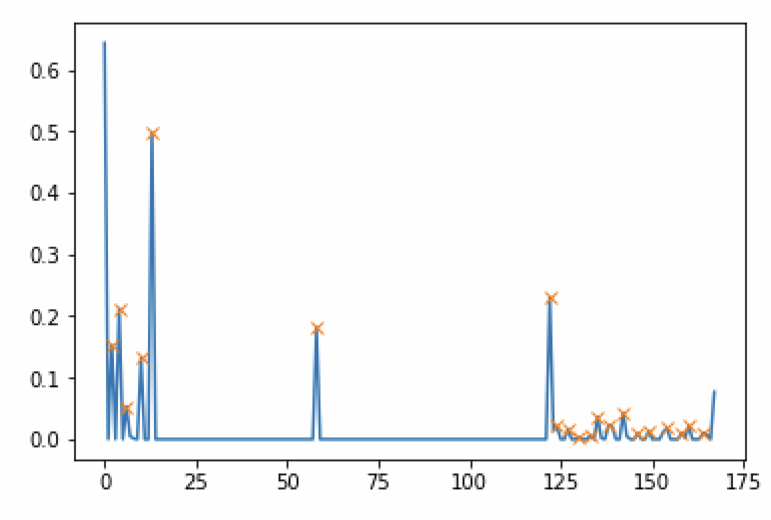


Figure 6. An example of light intensity variation and the peaks detected

3.7. The step detection function with the 3-axis acceleration sensor

To be able to determine an important parameter of any wearable device, the pedometer function has been added. To determine the steps taken, we needed the ADXL-345 3-axis acceleration sensor. We tried not to use an already developed library, but to give commands to the status registers of the device connected to the I2C bus. For this, it was first necessary to address the circuit, to configure it to read values and then to receive the values sent on the bus. However, we use the functions of the Wire library, working with the I2C bus, to send and receive data.

3.8. The pedometer function

The values read from the acceleration sensor, from the 3 axes, are stored in a vector. In order to be able to determine their variation and therefore recognize a walking step, we read 15 consecutive values. Then, we determine the mean value and standard deviation, which can provide us with information about the magnitude of the acceleration variation. If they exceed a certain threshold means that a step has been taken during the 15 samples. If not, it moves on to the next values, determining either a new step or a previously detected step.

3.9. Communication functions via the Wi-Fi module

The ESP8266 WI-FI module connected to the Arduino serial port will be used as a modem, a wireless connection interface. The firmware installed on the module allows us to give it "AT" commands (to configure it, send and receive data). The AT commands marked on the module's data sheet will allow us to configure it on its first power-up, using the wifi_init () function. Thus, after a reset, it will be configured as an access point (AP), allowing other computing devices to connect to it. Using the AT + CWDHCP = 0.1 command, a local DHCP server will be initialized, which will assign IP addresses to the connected devices after the SSID, connection password, and WEP authentication have been configured. Moreover, through AT commands, a transparent working mode of the modem from the Arduino's point of view will be configured. A server on port 80 HTTP protocol for web pages will be added as well.

```
// FUNCTION THAT INITIALIZES THE WIFI MODULE
VOID WIFI_INIT() // SEND COMMANDS TO THE MODULE
  {
  ESTABLISHCONNECTION("AT+RST\R\N",100); // RESETS THE MODULE
  ESTABLISHCONNECTION("AT+CWMODE=2\R\N",100); //SET AS ACCESS POINT
  ESTABLISHCONNECTION("AT+CWDHCP=0,1\R\N",100); //SET DHCP SERVER
  // SETTING AP PARAMETERS: SSID, PASSWORD, CHANNEL, SECURITY
  ESTABLISHCONNECTION("AT+CWSAP=\"E-
    JACKET\",\"12345678\",10,4\R\N",1000);
  ESTABLISHCONNECTION("AT+CIPMODE=0\R\N",100);// TRANSPARENT MODE
  ESTABLISHCONNECTION("AT+CIPMUX=1\R\N",100);//MULTIPLE CONNECTIONS
  // CREATING A TCP SERVER ON PORT 80
  ESTABLISHCONNECTION("AT+CIPSERVER=1,80\R\N",1000);
}
```

The ESP8266 module responds with the string "OK" to the correct completion of AT commands. Since these commands have different completion times, it was necessary to call the `establishconnection()` function. It receives as parameters the command (type string) and the waiting time for response. The command is sent through serial port 1 to the module and a time-out equal to the specified one is expected. Correspondingly, if no response has been received after the first command, at least twice attempts will be made to resend the command.

The WI-FI module uses a special command to send data through its modem, namely AT + CIPSEND = 0, N, where N represents the number of characters that will be sent immediately after receiving the command. The `senddata()` function receives the string, measures its length, terminates it with an enter (CRLF) and sends it via the modem.

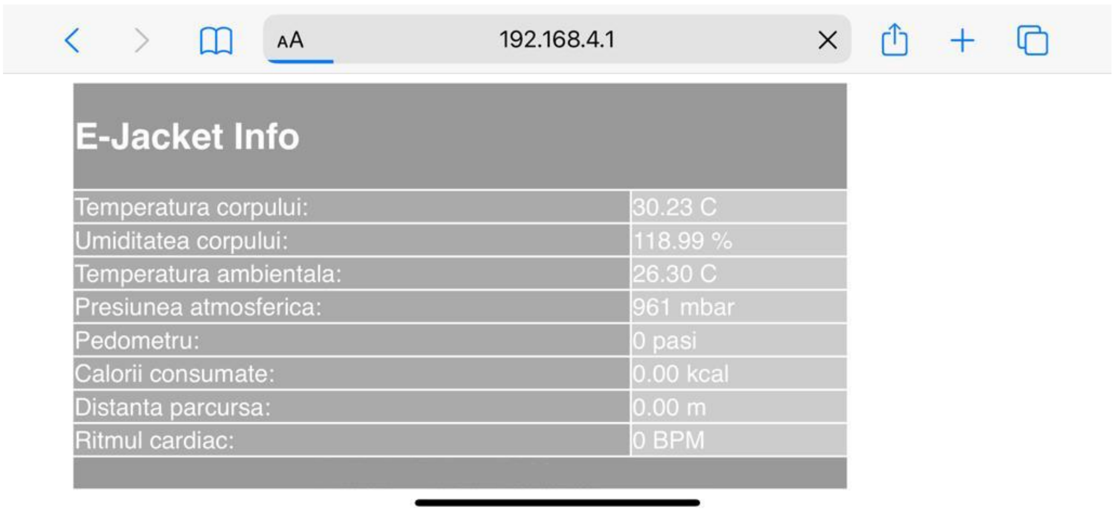
3.10. The HTML page build function

Following the receiving of a request on the server's port 80, the content the HTML code comprising all the measured values will be sent. In the HEADER's META information, we requested its renewal via the browser, every 10 seconds. In the string variable “server”, using the “append” command both the strings that define the table and the measured values will be added. Due to the variable’s length limitation, we form two such strings. In between sending the two from the WI-FI module to the client, the `senddata()` function is called.

```
// FORMING THE HTML PAGE TO BE SENT THROUGH THE WEB SERVER
VOID SENDTOSERVER(){

  STRING SERVER=""; // VARIABLE USED TO GENERATE THE HTML ELEMENTS

  SERVER ="<!DOCTYPE HTML> <HTML>";
  SERVER+="<HEAD><META HTTP-EQUIV=\"REFRESH\" CONTENT=\"10\"
 CONTENT=\"WIDTH=DEVICE-WIDTH\">"; //REFRESHING THE PAGE EVERY 10 SEC
  SERVER+="<TITLE>E-JACKET</TITLE></HEAD>"; // THE TITLE IS DEFINED
  SERVER+="<BODY><DIV ID=\"WEBPAGE\">"; // THE PAGE’S ID IS “WEBPAGE”
// WE DEFINED THE FONT, COLOR AND SIZE OF THE TEXT
SERVER+="<FONT  FACE  =  \"HELVETICA\"  SIZE  =  \"5\"  COLOR  =
\"#FFFFFF\">";
  // WE DEFINED THE TABLE SIZE
  SERVER+="<TABLE WIDTH = \"80%\" BORDER = \"0\">";
  SERVER+="<TR><TD COLSPAN = \"2\" BGCOLOR = \"#999999\"><H2>E-JACKET
INFO</H2></TD></TR>"; // WE DEFINED THE NUMBER OF COLUMNS, THE CELL
BACKGROUND COLOR, THE TITLE OF THE TABLE
  SERVER+="<TR><TD BGCOLOR = \"#AAAAAA\" >BODY TEMPERATURE: </TD><TD
BGCOLOR = \"#CCCCCC\" >"; // WE DEFINED THE BACKGROUND COLOR #AAAAAA
FOR THE FIRST CELL, THE OTHER COLOR FOR THE SECOND
  SERVER+=STRING(TEMPC); // WE ADDED THE TEMPERATURE TO OUR STRING
  SERVER+=" C</TD></TR>"; // THE SYMBOL FOR CELSIUS, THEN WE "CLOSED
THE ROW"
  SERVER+="<TR><TD  BGCOLOR  =  \"#AAAAAA\"  >BODY  HUMIDITY:  </TD><TD
BGCOLOR = \"#CCCCCC\" >";// WE DEFINED THE BACKGROUND COLOR #AAAAAA
FOR THE FIRST CELL, THE OTHER COLOR FOR THE SECOND
  SERVER+=STRING(HUMIDITY); // WE ADDED THE HUMIDITY
  SERVER+=" %</TD></TR>"; // ADDING THE PERCENTAGE SYMBOL THEN WE ARE
FINISHED WITH THIS ROW, WE TAKE IT FROM THE BEGINNING
  SERVER+="<TR><TD   BGCOLOR   =   \"#AAAAAA\"   >AMBIENT   TEMPERATURE:
</TD><TD BGCOLOR = \"#CCCCCC\" >";
  SERVER+=STRING(TEMPA);
  SERVER+=" C</TD></TR>";
  SERVER+="<TR><TD   BGCOLOR   =   \"#AAAAAA\"   >ATMOSPHERIC   PRESSURE:
</TD><TD BGCOLOR = \"#CCCCCC\" >";
  SERVER+=STRING(PRESSURE);
  SERVER+=" MBAR</TD></TR>\N";
  SENDDATA(SERVER); // WE SENT THE STRING TO THE CLIENT

  SERVER="<TR><TD BGCOLOR = \"#AAAAAA\" >PEDOMETER: </TD><TD BGCOLOR
= \"#CCCCCC\" >"; // WE DEFINED THE SERVER STRING AGAIN
  SERVER+=STRING(PASI);
  SERVER+=" PASI</TD></TR>\N";
  SERVER+="<TR><TD BGCOLOR = \"#AAAAAA\" >BURNED CALORIES: </TD><TD
BGCOLOR = \"#CCCCCC\" >";
  SERVER+=STRING(CALORII);
  SERVER+=" KCAL</TD></TR>";
  SERVER+="<TR><TD BGCOLOR = \"#AAAAAA\" >DISTANCE: </TD><TD BGCOLOR
= \"#CCCCCC\" >";
  SERVER+=STRING(DISTANTA);
  SERVER+=" M</TD></TR>";
  SERVER+="<TR><TD  BGCOLOR  =  \"#AAAAAA\"  >BPM:  </TD><TD  BGCOLOR  =
\"#CCCCCC\" >";
```

```
  SERVER+=STRING(MYBPM);
  SERVER+=" BPM</TD></TR>"; //THE TABLE IS NOW CLOSED
  SERVER+="<TR><TD COLSPAN = \"2\" BGCOLOR = \"#999999\"><CENTER>
AUTHORS - ANA & MARIA BRAD</CENTER></TD></TR>"; // WE MADE A NEW
TABLE WITH 2 COLUMNS USING A DIFFERENT COLOR
  SERVER+="</FONT></DIV></BODY></HTML>\N"; //THE TABLE IS NOW CLOSED
  SENDDATA(SERVER); // THE HTML STRING IS SENT TO THE CLIENT

  DELAY(100);
  SERIAL1.PRINT("AT+CIPCLOSE=0\R\N"); // THE TCP CONNECTION TO THE
CLIENT IS CLOSED
}
```

Figure 7 shows the web page via the Safari browser of an iPhone connected to the e-jacket's AP.



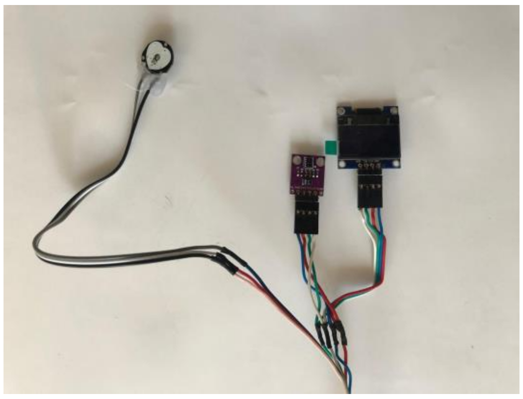
E-Jacket Info

Temperatura corpului:	30.23 C
Umiditatea corpului:	118.99 %
Temperatura ambientala:	26.30 C
Presiunea atmosferica:	961 mbar
Pedometru:	0 pasi
Calorii consumate:	0.00 kcal
Distanta parcursa:	0.00 m
Ritmul cardiac:	0 BPM

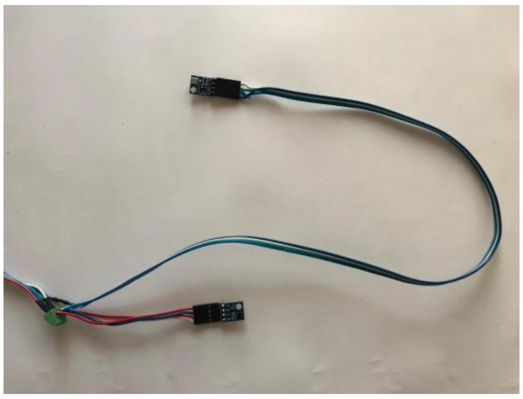
Figure 7. Web page of the e-jacket accessed from a mobile phone

4. Results

The first version of the system was used for testing purposes was tied to a cardboard panel. The interconnections were made with wire strips, dimensioned for their future placement on the jacket. As seen from figures 8 and 9, connectors were used for the connection between the module's wiring and the Arduino module.
After assembling the electrical connections and wiring, the software was implemented via the Arduino IDE. The source code was written for each module separately and an attempt was made to calibrate them. Optimal values were assigned to their corresponding variables in each code part. The OLED display can display, with a special set of characters, a maximum of 3 lines of text, as seen in figure 10. Therefore, we present the 4 screens displaying the data.

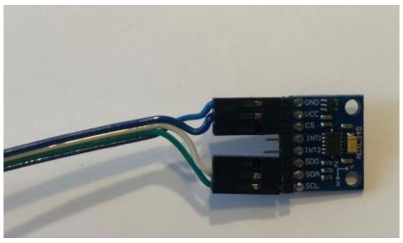


a)

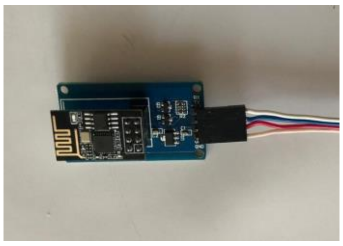


b)

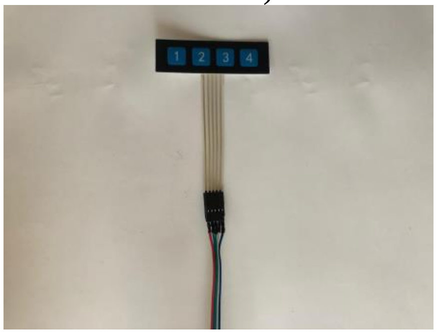
Figure 8. Connecting a) the pulse sensor, the ambient light sensor and the OLED display; and b) the two combined sensors of temperature, humidity, altitude, pressure on the I2C



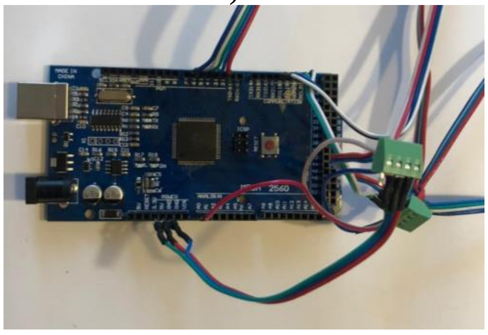
a)



b)



c)



d)

Figure 9. Connecting a) the acceleration sensor and b) the Wi-Fi module; c) the keyboard and d) the Arduino module to the other sensors

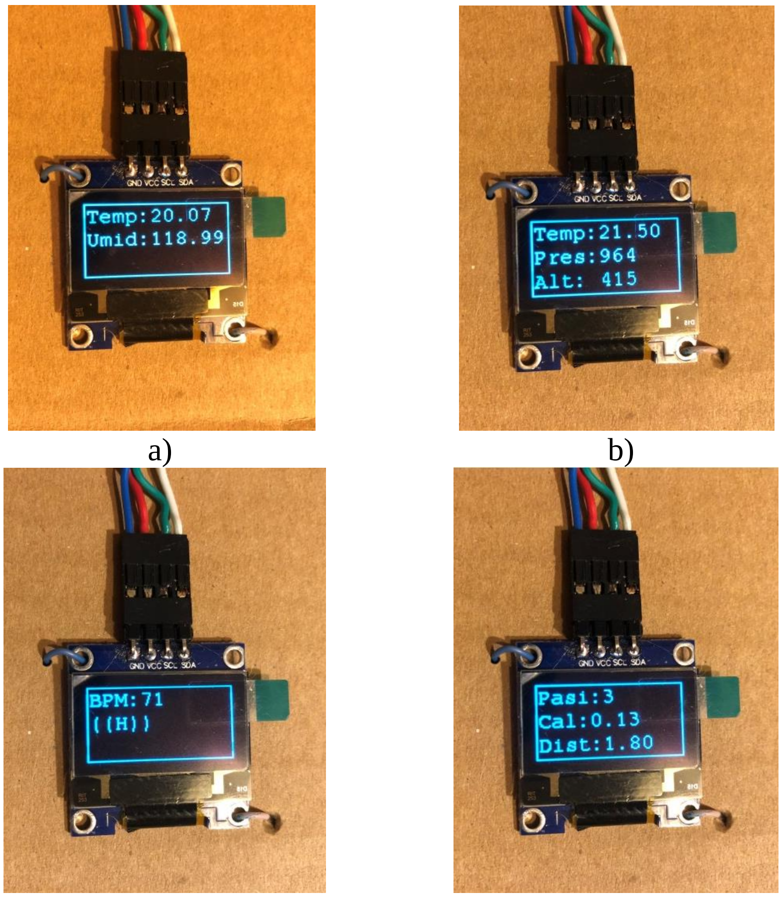
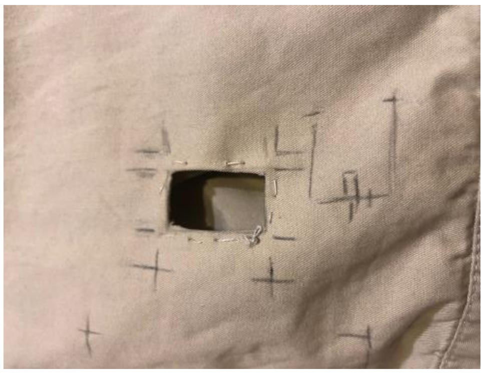
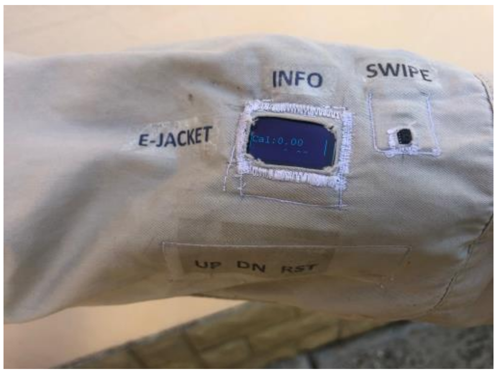


Figure 10. The display of the e-jacket data; a) the body temperature and humidity; b) the outdoor temperature, atmospheric pressure and altitude; c) the BPM detection; d) the number of steps, calories burned and the distance traveled

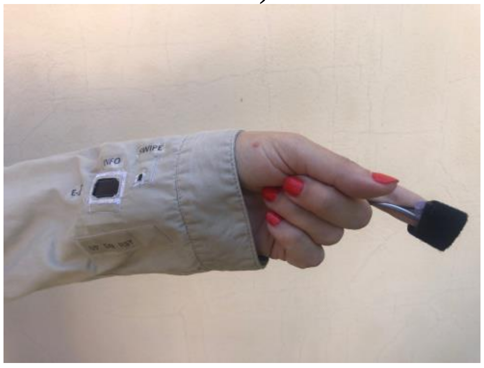
The jacket has two layers; the first one made of the base material of 100% cotton fabric, which is completed by an inner layer, the lining, with 100% cotton fabric. The devices will be placed in the clothing product so that they can perform their functions. The Arduino development board and the battery will be inserted in the inside pockets of the jacket and the temperature and humidity sensor will be fixed inside it, so that they can measure the two parameters as close as possible to the body. The Wi-Fi module can be fixed on the shoulder line, and the 3-axis accelerometer sensor, as well as the proximity and gesture sensor, the keyboard and the display can be fixed on the surface of the two sleeves. The pulse sensor can be fixed on a finger using a Velcro as shown in figure 11.



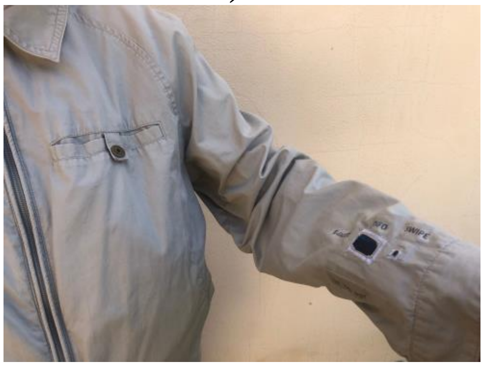
a)



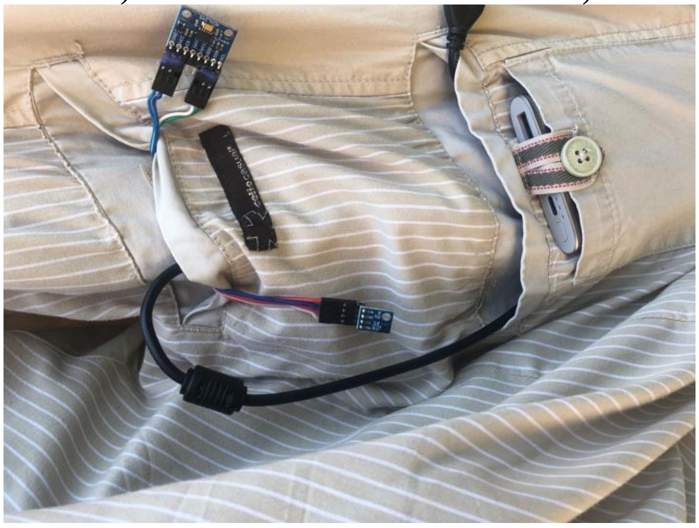
b)



c)

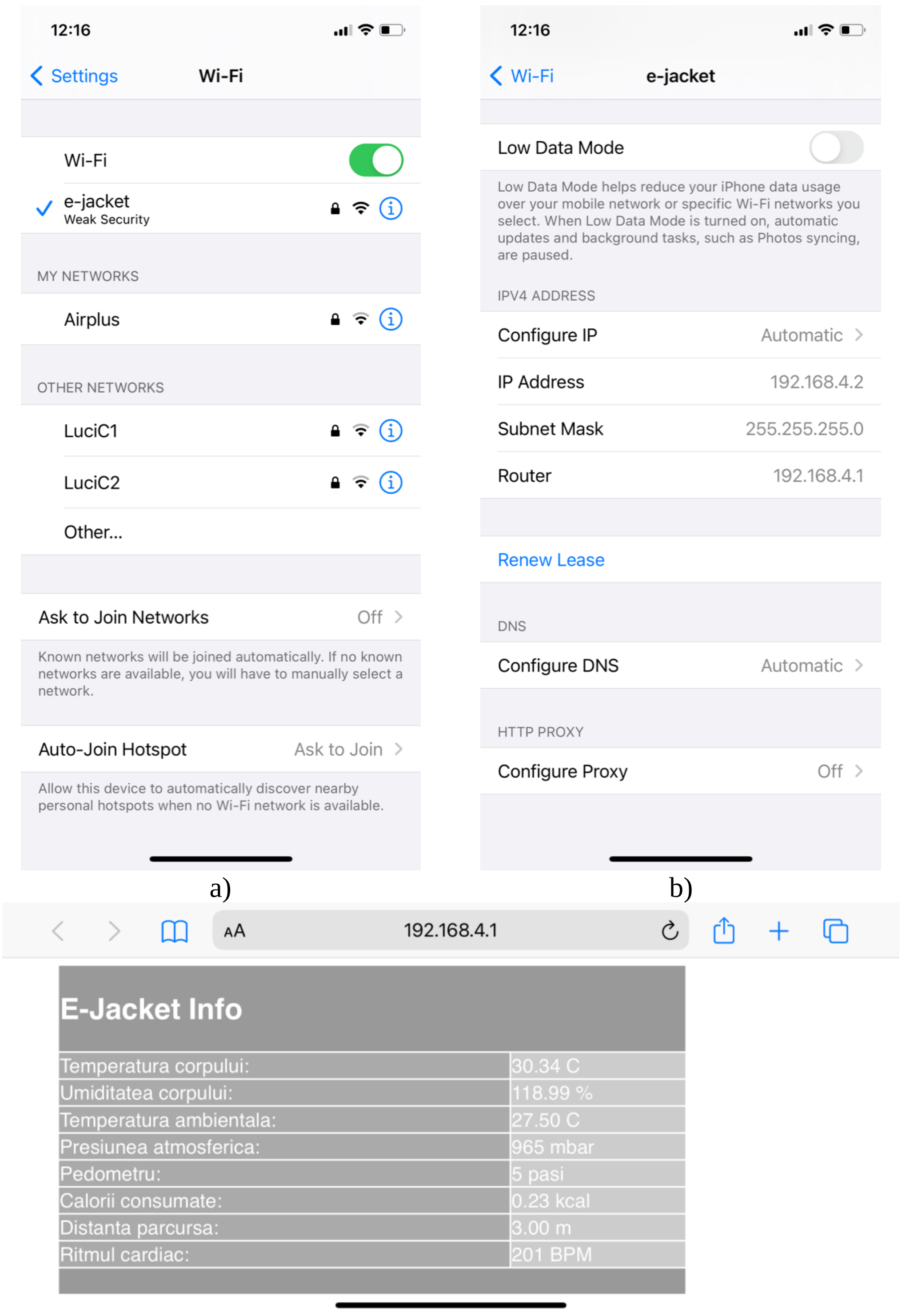


d)



e)

Figure 11. Positioning the display, keyboard and sensor; a) their position and the cut of the interior pockets and the display window; b) inscriptions by thermal transfer; c) position of the pulse sensor; d) the final clothing product; e) position of the battery, the Arduino module (in the pocket) and the sensors



12:16

Settings Wi-Fi

Wi-Fi

e-jacket
Weak Security

MY NETWORKS

Airplus

OTHER NETWORKS

LuciC1

LuciC2

Other...

Ask to Join Networks Off

Known networks will be joined automatically. If no known networks are available, you will have to manually select a network.

Auto-Join Hotspot Ask to Join

Allow this device to automatically discover nearby personal hotspots when no Wi-Fi network is available.

a)

12:16

Wi-Fi e-jacket

Low Data Mode

Low Data Mode helps reduce your iPhone data usage over your mobile network or specific Wi-Fi networks you select. When Low Data Mode is turned on, automatic updates and background tasks, such as Photos syncing, are paused.

IPV4 ADDRESS

Configure IP	Automatic
IP Address	192.168.4.2
Subnet Mask	255.255.255.0
Router	192.168.4.1

Renew Lease

DNS

Configure DNS Automatic

HTTP PROXY

Configure Proxy Off

b)

AA 192.168.4.1

E-Jacket Info

Temperatura corpului:	30.34 C
Umiditatea corpului:	118.99 %
Temperatura ambientala:	27.50 C
Presiunea atmosferica:	965 mbar
Pedometru:	5 pasi
Calorii consumate:	0.23 kcal
Distanta parcursa:	3.00 m
Ritmul cardiac:	201 BPM

c)

Figure 12. Connecting the e-jacket to the Wi-Fi module; a) retrieving the SSID of the e-jacket's access point; b) the TCP/IP configuration received from the e-jacket's DHCP server; c) the values displayed while accessing the e-jacket web server using a web browser

5. Conclusions

One of the challenges of e-textiles is the combination of suitable materials in order to provide the desired characteristics, aiming to create hybrid materials from both textile and electronic perspective. The miniaturization of the electronics has made them available on the market and, consequently, attracted considerable interest from researchers to incorporate these components into textiles.
Another important challenge of e-textiles is to deliver an adequate amount of reusability and effectiveness, surviving the washing process through their life cycle.
Nowadays, various research labs are looking for different approaches to integrate electronic components into textiles, using efficient, reliable and cost-effective methods. Although the field of smart textiles is at the beginning, the scientific researches, published papers and presented prototypes are increasing. However, it can also be seen that despite of that, there are relatively few products accessible to the consumers.
With the above mentioned limitations, we have managed to design, assembly, program and test a wearable smart cloth, presenting the most common usage functions. The platform used is a low cost, low power and easy to program. If a waterproof impregnation could be applied to the sensors, connectors and Arduino, a dry cleaning of the e-jacket should be made possible and therefor enlarging the life span of the smart product.

References

[1] Ismar, E., Kurşun, S., Kalaoglu, F., Koncar, V., *Futuristic Clothes: Electronic Textiles and Wearable Technologies*, Global Challenges, 4, 2020 https://doi.org/10.1002/gch2.201900092
[2] Raluca Brad, *E-textile: imbracamintea computerizata de la cercetare la productie*, Sesiunea de Comunicari stiintifice, Academia Fortelor Terestre "Nicolae Balcescu", 2003
[3] Koncar, Vladan, *Smart Textiles and Their Applications*, Woodhead Publishing, Print ISBN: 9780081005743, 2016
[4] Choudhry, N.A., Arnold, L., Rasheed, A., Khan, I.A. and Wang, L. *Textronics - A Review of Textile-Based Wearable Electronics,* Adv. Eng. Mater, 2100469, 2021 https://doi.org/10.1002/adem.202100469
[5] Wang, Shuguang and Wang, Zhongwu and Li, Jie and Li, Liqiang and Hu, Wenping, *Surface-grafting polymers: from chemistry to organic electronics*, Mater. Chem. Front., vol. 4, iss. 3, pp. 692-714, 2020 doi = 10.1039/C9QM00450E
[6] Bedon, C., Rajčić, V., *Textiles and Fabrics for Enhanced Structural Glass Facades: Potentials and Challenges*, Buildings, 9, 156, 2019 https://doi.org/10.3390/buildings9070156
[7] https://www.arduino.cc/en/Guide/ArduinoMega2560
[8] https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
[9] http://www.farnell.com/datasheets/1780639.pdf
[10] https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf
[11] https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf
[12]https://components101.com/asset/sites/default/files/component_datasheet/Pulse%20Sensor%20Datasheet.pdf
[13] https://docs.broadcom.com/doc/AV02-3190EN
[14] https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf