SISTEM FITNESS INTELIGENT

Student: Gheorghe Andrei-Alexandru

Grupa: 331AA

Coordonator proiect: Daniel-Constantin ȚICU

Cuprins

Prezentarea problematicii

Condiția fizică este starea de sănătate și bunăstarea și, mai specific, capacitatea de a efectua aspecte ale sportului, ocupațiilor și activităților zilnice. Condiția fizică este, în general, realizată printr-o nutriție adecvată, exercițiu fizic moderat și viguros și odihnă suficientă.

Înainte de revoluția industrială, fitness-ul a fost definit ca aptitudinea de a desfășura activitățile zilei fără oboseală excesivă.

Cu toate acestea, prin automatizarea și schimbările în stilul de viață, aptitudinea fizică este acum considerată o măsură a capacității organismului de a funcționa eficient inclusiv în activitățile de muncă și de agrement, de a fi sănătos, de a rezista bolilor hipokinetice și de a face față situațiilor de urgență.

Totuși, odată cu evoluția tehnologiei , din ce în ce mai mulți oameni, de la tineri la vârstnici , au început să aibă o viață din ce în ce mai sedentară. Pentru combaterea sedentarismului din ce în ce mai multe mijloace de realizare a exercițiilor fizice au fost implementate. S-a început cu realizarea de spații specializate dotate cu dispozitive fitness , pe care în ziua de astăzi le cunoaștem sub denumirea de săli fitness. Deși aceasta, la momentul respectiv, a părut o soluție perfectă , această impresie era cu mult încă departe de realitate. Majoritatea sălilor de fitness lucrează cu utilizatorii pe baza unui abonament cu program fix sau flexibil , iar accesul la ele este limitat de mai mulți factori, după cum urmează:

a) Cele mai multe persoane cu vârste de peste 30 ani abia își pot face timp suficient pentru întreținerea unui job și a propriilor copii (mai ales dacă sunt încă mici)
b) Există multe situații în care poziționarea locației sălii de fitness nu este una avantajoasă în raport cu locația domiciliului persoanei.
c) Există multe persoane care nici măcar nu își permit luxul plății unui abonament lunar pentru accesul la o sală de fitness cu program flexibil deoarece apar extrem de multe cheltuieli necesare simplului proces de supraviețuire (benzină, întreținere, rate, etc) .

În contextul descris, a apărut o cerere a utilizatorilor pentru dispozitive portabile miniaturizate, care să aibă diverse funcționalități, iar printre cele mai simple dintre ele se numără :

a) Monitorizarea pulsului

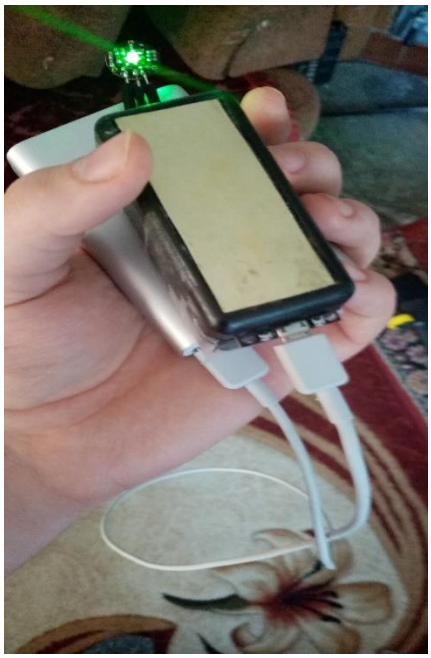
b) Monitorizarea numărului de pași

c) Monitorizarea realizării corecte a exercițiilor simple de tipul flotare sau tracțiune.

Aceste dispozitive sunt ceea ce cunoaștem în ziua de astăzi sub forma de ”fitness trackers” și au devenit accesibile aproximativ tuturor categoriilor de utilizatori.

Soluția propusă (Descriere completă).

Aplicația propusă de mine este una realizată cu scop didactic , însă care implementează o soluție pentru problemele simple enunțate în cadrul paragrafului anterior.



Pentru monitorizarea activității fitness se folosește un modul MPU6500 ce integrează un giroscop cu 3 axe, un accelerometru cu 3 axe dar și un procesor DMP (Digital Motion Processor), toate într-un singur cip. Comunicarea între placa de dezvoltare și modulul utilizat se face prin intermediul protocolului de comunicație I2C. De asemenea, modulul utilizat poate primi informații de la alte dispozitive cu I2C extern.

Protocolul de comunicație I2C

Protocolul Inter Integrated Circuit (I2C) este un protocol creat pentru a permite mai multor circuite integrate “slave” să comunice cu unul sau mai mute cipuri “master”. Acest tip de comunicare poate fi folosit doar pe distanțe mici de comunicare și ,asemenea protocolului UART, are nevoie doar de 2 fire de semnal pentru a trimite/primi informații.

De ce I2C și nu alte protocoale de comunicație precum SPI sau comunicație prin porturi seriale?

Deoarece porturile seriale sunt asincrone, dispozitivele ce utilizează porturile seriale trebuie sa aibă ceasuri cu o rată apropiată de transmisie a datelor si au comunicarea UART la bază, un tip de comunicare complex si dificil de implementat.

Un alt minus prezent în abordarea unei comunicări bazate pe porturi seriale (Fig 1.) este rata de transmisie a datelor. Teoretic nu este nicio limita in comunicarea seriala asincrona, dar practic in comunicarea UART, dispozitivele suportă o rată de transfer fixată și cea mai mare rată este de aproximativ 230400 biti/secundă.

Fig 1. Comunicație prin intermediul porturilor seriale

Cel mai mare minus pe care îl prezintă comunicația SPI (Fig 2) este numărul mare de pini de care are nevoie. Pentru conectarea unui singur ”master” la un singur ”slave” este nevoie de 4 linii . Pentru fiecare slave conectat, este ulterior necesară conectarea la un cip adițional pentru selecția unui pin I/O de pe dispozitivul folosit ca ”master”. Multitudinea de conexiuni, face ca această abordare să rezulte într-o transmisie înceată a semnalului, iar eu , în implementarea proiectului aveam nevoie de o transmisie cât mai rapidă a semnalelor.

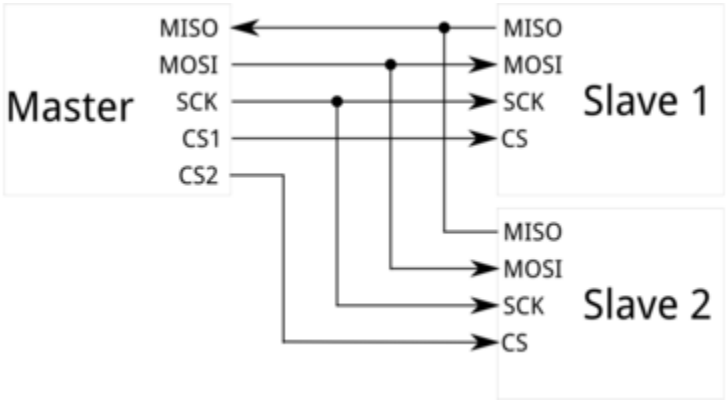


Fig 2.

I2C (Fig 3.) necesita două fire, ca și la comunicarea seriala asincrona, cu diferența că acest tip de comunicare poate suporta pana la 1008 dispozitive de tip "slave". Mai mult decat atat, in comunicarea de tip I2C pot exista mai multe dispozitive de tip "master" la un bus, lucru nepermis in comunicarea SPI.

Rata datelor nu este foarte bună, fiind asemănătoare cu cea de la portul serial, cele mai multe dintre I2C-uri comunicând cu o rată de transmisie cuprinsă între 100kHz si 400kHz, însă este suficient pentru scopul dorit (transmisie la 115.2Khz).

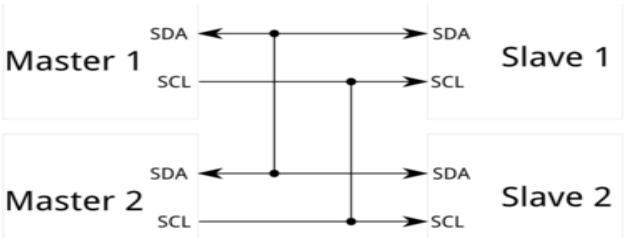


Fig 3. Protocolul de comunicație I2C

În ceea ce priveste implementarea (Fig 4.), aceasta este mai complexă decât în cazul implementării SPI, dar mult mai ușoară decât in cazul comunicării asincrone, cu UART.

Fiecare bus I2C este compus din doua semnale: SCL (semnalul de ceas) si SDA (semnalul de date). Semnalul de ceas este întotdeauna generat de bus-ul masterul curent.

Spre deosebire de alte metode de comunicare, precum UART, magistrala I2C este de tip “open drain”, eliminând astfel problema de "bus contention”, unde un dispozitiv încearcă să tragă una dintre linii în starea "high” în timp ce altul o aduce in "low”, eliminând posibilitatea de a distruge componente.

```
//metodele de implementare ale protocolului I2C
void I2C_Write(uint8_t deviceAddress, uint8_t regAddress, uint8_t data){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.write(data);
  Wire.endTransmission();
}

// read all 14 register
void Read_RawValue(uint8_t deviceAddress, uint8_t regAddress){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.endTransmission();
  Wire.requestFrom(deviceAddress, (uint8_t)14);
  AccelX = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelY = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelZ = (((int16_t)Wire.read()<<8) | Wire.read());
  Temperature = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroX = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroY = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroZ = (((int16_t)Wire.read()<<8) | Wire.read());
}

//configurare MPU6050
void MPU6050_Init(){
  delay(150);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SMPLRT_DIV, 0x07);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_1, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_2, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_CONFIG, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_GYRO_CONFIG, 0x00);//set +/-250 degree/second full scale
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_ACCEL_CONFIG, 0x00);// set +/- 2g full scale
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_FIFO_EN, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_INT_ENABLE, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SIGNAL_PATH_RESET, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_USER_CTRL, 0x00);
}
```

Fig 4. Exemplu de implementare a comunicației I2C

Pentru implementarea conexiunii I2C am utilizat biblioteca Wire.h disponibilă în cadrul Arduino IDE ce prezintă metode-suport pentru transmisia și citirea de date folosind protocolul de comunicație I2C.

Momentan , însă am obținut doar coordonatele de poziție de la accelerometru (AccelX,AccelY,AccelZ) , respectiv unghiurile de rotație (GyroX,GyroY,GyroZ). Aceste informații trebuie utilizate pentru atingerea următoarelor obiective:

1. Monitorizare numărului de pași (lucru care se traduce prin schimbarea de coordonate în spațiul tridimensional) .Distanța absolută obținută este filtrată cu o anumită constantă de filtrare acordată prin intermediul metodei ”Trial and Error”.

2. Recunoașterea mișcărilor corecte de tip tracțiune(pe verticală sau pe orizontală). Acest aspect se traduce prin faptul că mișcarea completă nu presupune atât efort precum mișcarea completă a corpului , ca în cazul monitorizării numărului de pași, însă presupune menținerea constantă a mișcării în cadrul unui plan, fie vertical fie orizontal.

Prin urmare, pentru mișcarea de tip tracțiune vom folosi o constantă de filtrare a datelor mai mică decât cea folosită pentru rezolvarea primei probleme, însă suficient de mare încât să filtrăm comportamente în care utilizatorul stă degeaba și spre exemplu schimbă poziția brățării pe mână.

Vom impune , de asemenea, constrângerea ca mișcarea pe un anumit plan să fie liniară – adică unghiul format de tangenta la traiectorie să fie minim , chiar 0, dacă este posibil.

Pentru cerința 1 constanta de filtrare folosită este 0.7, iar pentru cerința 2 , intervalul de filtrare a componentelor este [0.3,0.5]. Această abordare , însă nu garantează o acuratețe mare (80%). Pentru o acuratețe mai mare , putem folosi fie mai mulți senzori PIR de mișcare , fie un senzor EMG care măsoară efortul depus de mușchi, iar clasificarea tipului de mișcare poate fi făcută cu o precizie mult mai fină.

Pentru realizarea monitorizării pulsului în timp ce utilizatorul realizează diverse exerciții fitness am ales să folosesc un senzor de puls XD-58C . Senzorul de puls XD-58C este un senzor de tip ”plug-and-play” ce este proiectat să funcționeze împreună cu o placă de dezvoltare echipată cu cel puțin un pin analog. Acest produs este util pentru a colecta date despre bătăile inimii în diferite situații, cum ar fi în timpul exercițiilor fizice. Pentru implementare am utilizat biblioteca PulseSensorPlayground.h care prezintă atât o soluție de identificare a valorii BPM folosind un sistem de înreruperi intern, dar și o soluție ce nu folosește un sistem de întreruperi , pe care am utilizat-o în cadrul proiectului meu(Fig 5.).

```
if (pulseSensor.sawNewSample()) {
  if (--samplesUntilReport == (byte) 0) {
    samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE;

    pulseSensor.outputSample();
    if (pulseSensor.sawStartOfBeat()) {
      pulseSensor.outputBeat();
    }
    c = pulseSensor.getBeatsPerMinute();
  }
}
```

Fig 5. Implementare soluție preluare puls.

Pulsul normal pentru o persoană ce nu desfășoară sport este între 60 și 90 bpm. Totuși, în urma depunerii de efort, acesta poate ajunge la o valoare maximă Hrmax = 208 – (0.7 x vârstă). Pentru proiectul curent, fiind unul cu scop didactic , care încă poate fi dezvoltat însă, nu am generalizat ppentru toate categoriile de vârstă și am presupus că eu voi fi cel care va testa dispozitivul propus(vârstă = 21 ani , deci BPM se va afla în intervalul [60,193]).

Această restricție este utilă pentru identificarea momentului în care utilizatorul este notificat că ar trebui să ia o pauză de la activitatea sa sportivă. Modul în care notificarea este transmisă va fi detaliat când vorbim despre plăcuța de dezvoltare folosită și suportul pe care îl are pentru transmiterea datelor prin protocol WI-FI.

În continuare, voi detalia algoritmul folosit pentru identificarea schimbării de coordonate în spațiul tridimensional:

Etapa 1. În primul rând, programul începe calibrarea datelor în ”void setup” primite de la modulul MPU6500 imediat ce este pornit și obține valorile medii de acelerație – xAvg, yAvg, zAvg.

Etapa 2. În ”void loop” se obțin 100 date de la axa X, Y și Z , atât timp cât brățara este utilizată. Această identificare se face prin intermediul monitorizării pulsului , în măsura în care este necesar ca senzorul să atingă pielea persoanei pentru a prelua date.

> ***Etapa 2.1*.** Calculez vacc = $\sqrt{(\mathrm{x} - \mathrm{xAvg})\ \hat{}\ 2\ + (\ \mathrm{y} - \mathrm{yAvg})\ \hat{}\ 2\ + (\mathrm{z} - \mathrm{zAvg})\ \hat{}\ 2}$ de fiecare dată când obțin date.
>
> ***Etapa 2.2***. Apoi, se compară vacc cu valoarea parametrului de filtrare, în cazul nostru fiind 0.7. Dacă vectorul de accelerație trece valoarea pragului, atunci crește numărul de pași, iar în caz contrar, se ignoră vibrațiile nevalide, numărul de pași rămânând același.

Și totuși cum pot utilizatorii să vizualizeze datele obținute sau pot fi notificați în legătură cu progresul lor?

Răspunsul se obține prin detalierea sumară a funcționalităților plăcii de dezvoltare alese - Placă de Dezvoltare WiFi cu ESP8266 și CH340G . Aceasta este o placă de dezvoltare compatibilă cu NodeMCU V1.0 (ESP12-E) , care prezintă suport pentru transmisia datelor prin internet(ESP8266WiFi.h). Această funcționalitate ne permite , prin urmare, să realizăm următoarele două funcționalități:

a) Notificarea utilizatorului în legătură cu numărul de pași realizat pe minut.

b) Notificarea utilizatorului în legătură cu numărul de tracțiuni realizate.

c) Notificarea utilizatorului în momentul în care pulsul său a depășit limita maximă permisă.

Pentru transmisia notificărilor se folosește un serviciu de tip PushBullet. De asemenea, țin evidența datelor prin intermediul serviciului ThingSpeak. Astfel, pentru ulterioare dezvoltări se poate realiza o aplicație Web/Mobile care să folosească datele obținute de la ThingSpeak API pentru monitorizarea progresului fitness al diverșilor utilizatori logați. În continuare vom detalia algoritmul de transmisie a

datelor către Thingspeak (ALGORITM 1), respectiv cel de transmisie a datelor către serviciul PushBullet (ALGORITM 2). Un exemplu de implementare se poate observa în (Fig 7.).

ALGORITM 1 – TRANSMISIE DATE CĂTRE THINGSPEAK

Etapa 1: Setare SSID și PAROLA necesare pentru realizarea unei conexiuni către un punct de acces WI-FI (Fig 6).

Etapa 2: Conectare la rețea + tratare cazuri excepție

Etapa 3: Transmisie date prin folosirea unui request de tip POST specific protocolui HTTP folosind cheia de conectare la API cu permisiune de tip ”Write” furnizată în momentul creării canalului în care se face vizualizarea datelor.

Etapa 4: Asigurarea update-ului (minim 15 secunde) + închiderea conexiunii.

ALGORITM 2 – TRANSMISIE DATE UTILIZATE DE CĂTRE SERVICIUL PUSHBULLET

Etapa 1: Setare SSID și PAROLA necesare pentru realizarea unei conexiuni către un punct de acces WI-FI.

Etapa 2: Conectare la rețea + tratare cazuri excepție

Etapa 3: Transmisia datelor prin folosirea unui request de tip POST. Astfel, mesajul de notificare este completat cu parametrii transmiși. Ulterior folosim un request de tip GET pentru transmiterea notificării prin Email , respectiv prin aplicația PsuhBullet instalată pe telefon(Fig 8.).

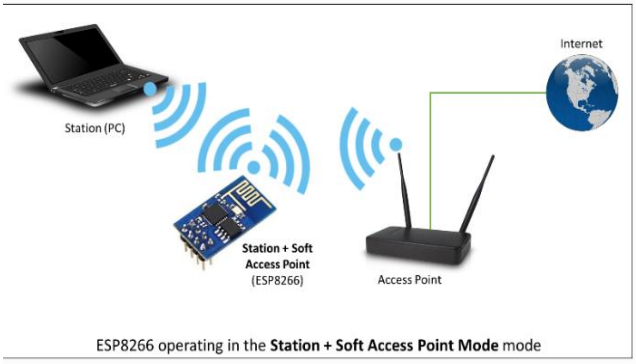


Fig 6. Folosirea plăcii de dezvoltare

Fig 8. Exemplu de notificare primită de utilizator pe telefon

```
String deviceId_Email= "vBA817A59D0E33C0";
String deviceId = "v4A962E06F959B36";
const char* logServer = "api.pushingbox.com";

//variabile pentru transmisia de date catre platforma ThingSpeak
String apiKey = "PCHRV9E9M11ZLIBN";     // API KEY FROM THINGSPEAK

const char *ssid =  "[redacted]";     // replace with your wifi ssid and wpa2 key
const char *pass = "[redacted]";
const char* server = "api.thingspeak.com";

WiFiClient client;
```

Setarea parametrilor necesari conectării la internet respectiv transmisiei de date

```
WiFi.begin(ssid, pass);

while (WiFi.status() != WL_CONNECTED)
{
   delay(500);
   Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
```

Realizare conexiune la internet

```
void sendNotification(int noSteps){

  Serial.println("- connecting to pushing server: " + String(logServer));
  if (client.connect(logServer, 80)) {
    Serial.println("- succesfully connected");

    String postStr = "devid=";
    postStr += String(deviceId);
    postStr += "&numberOfSteps=";
    postStr += String(noSteps);
    postStr += "\r\n\r\n";

    Serial.println("- sending data...");

    client.print("POST /pushingbox HTTP/1.1\n");
    client.print("Host: api.pushingbox.com\n");
    client.print("Connection: close\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);
  }
  client.stop();
  Serial.println("- stopping the client");
  if (client.connect(logServer, 80)) {
    client.print("GET /pushingbox?devid=");
    client.print(deviceId_Email);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(logServer);
    client.println("User-Agent: Arduino");
    client.println();
  }
}
```

Funcție de transmitere a notificărilor pe Email respectiv telefon

```
void sendTractiuniToThingspeak(int tractiuniCorecte){
      if (client.connect(server,80))   //   "184.106.153.149" or api.thingspeak.com
            {

                  String postStr = apiKey;
                  postStr +="&field1=";
                  postStr += String(tractiuniCorecte);
                  postStr += "\r\n\r\n";

                  client.print("POST /update HTTP/1.1\n");
                  client.print("Host: api.thingspeak.com\n");
                  client.print("Connection: close\n");
                  client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
                  client.print("Content-Type: application/x-www-form-urlencoded\n");
                  client.print("Content-Length: ");
                  client.print(postStr.length());
                  client.print("\n\n");
                  client.print(postStr);

                  Serial.print("Numar tractiuni corecte: ");
                  Serial.print(tractiuniCorecte);
            }
      client.stop();

      Serial.println("Waiting...");

  // thingspeak needs minimum 15 sec delay between updates, i've set it to 30 seconds
  delay(10000);
}
```

Funcție de transmisie a datelor către ThingSpeak

Bibliografie:

https://circuitdigest.com/microcontroller-projects/diy-arduino-pedometer-counting-steps-using-arduino-and-accelerometer

https://medicalxpress.com/news/2019-04-maximum-heart.html

https://nodemcu.readthedocs.io/en/master/modules/wifi/

https://people.cs.umass.edu/~silee/pub/C9.pdf

https://pulsesensor.com/pages/installing-our-playground-for-pulsesensor-arduino