

TDK dolgozat

Sütő József

Debreceni Egyetem
Informatikai Kar
Informatikai Rendszerek és Hálózatok Tanszék

Életjel monitorozó és tevékenység felismerő rendszer

Sütő József
MSc. 2. éves programtervező
informatikus hallgató

Dr. Oniga István
egyetemi docens

Debrecen
2013

Tartalomjegyzék

1. A rendszer létrehozásának célja	1
2. A rendszer jellegzetessége és működésének alapjai	1
3. A rendszer hardveres elemei	2
4. A kliens oldal	3
5. Hálózati kommunikáció	4
6. A szerver oldal	5
7. Tevékenység felismerés	7
7.1. Párhuzamos működés	9
7.2. A mozgásfelismerő algoritmus leírása	10
7.2.1. A felhasznált gyors Fourier transzformáció algoritmus	13
7.3. Frekvenciatérbeli kategorizálás és időtérbeli mintafelismerés	16
7.4. Esésetektálás	19
7.5. Kézfesztusok felismerése	20
8. Összegzés	24

1. A rendszer létrehozásának célja

A következő rövid leírásban szeretnék bemutatni egy egyszerű és kis költségű rendszert, amely az emberek életjeleinek (pulzus, vérnyomás, stb.) és cselekvéseinek (ülés, alvás, esés, stb.) megfigyelésében nyújt segítséget. A rendszer lehetőséget biztosít arra, hogy ütemezetten gyűjtsünk be adatokat a vizsgált személyektől és a begyűjtött információt egy központi szerverre tároljuk, ahonnan az adatok bármikor elérhetőek elemzési célokra. Az összegyűjtött információ segít analízálni a vizsgált személy egészségi állapotát és lehetőséget nyújt az esetleges tünetek detektálására. Az egyik lehetséges megoldás a tünetek kiszűrésére, az adathalmazok közötti mintafelismerés. Feltehetőleg a különböző tünetek jellegzetesen befolyásolják a páciens mért életjeleit, ami kiindulási alapként szolgál a felismerésben. Hogyha ki tudjuk szűrni a begyűjtött adathalmazokból a tünetekre utaló mintákat, akkor nagyobb eséllyel lehet védekezni a betegségek ellen, illetve elkerülhető a jelentkező betegség kialakulása. Ezen felül, a rendszeres adatbegyűjtés segítségével hosszútávon vizsgálható a megfigyelt személy életjeleinek változása.

2. A rendszer jellegzetessége és működésének alapjai

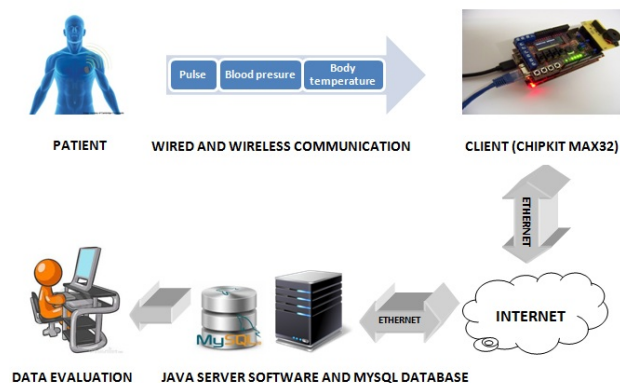
Napjainkban az életjelek vizsgálata egy meghatározó kutatási témakör, amely meglehetősen szétágazó és sok tudományterületet kapcsol össze. Világszerte rengeteg egyéni kutató és kutatóintézet foglalkozik az életjelekből származó plusz információk kinyerésével, amelyekből előreutató megállapításokat lehet tenni a vizsgált személy egészségi állapotáról [1-10].

Jelenleg az életjelek monitorozására rengeteg eszköz és különböző alkalmazás használható. Ilyenek például, az intelligens karóra vagy az egyre jobban terjedő mobil alkalmazások. Sajnos a már meglévő eszközök és alkalmazások közül a megbízhatóbbak meglehetősen költségesek, míg a kis költségűek megbízhatatlan és pontatlan mérési eredményeket adnak. Mindezekon felül, ami általánosan elmondható a legtöbb hasonló célú eszközről, hogy nem nyújtanak általános megoldást az életjelek monitorozására és nehezen alkalmazhatóak egy komplex rendszer részeként. Ezek hatására merült fel bennünk az ötlet, hogy hozzunk létre egy olyan általánosan alkalmazható rendszert, amely sokoldalúan alkalmazható és mindenki számára elérhető. Továbbá, a rendszer megalkotása megalapozza egy számunkra új kutatási témakör kezdetét.

Az eddig elkészített rendszer néhány mikrovezérlő és mini számítógép (Raspberry Pi) alapú kliens egységből és egy Java-ban megírt szerverprogramból áll. Minden kliens több hasznos tulajdonsággal rendelkezik, mint például: egyszerűen kezelhető, kisméretű, kis költségű és hordozható. A kliens egységek programjának készítésekor kitüntetett figyelmet fordítottunk arra, hogy a klienseket sokoldalúan lehessen felhasználni. Ez azt jelenti, hogy egyrészt a vizsgált személy otthonról tudja a begyűjtött adatokat továbbítani egy távoli szerver felé, vagy egy intézményen belül több vizsgált személy adatait lehessen feltölteni az intézményi szerverre.

Két típusú klienst különböztethetünk meg, az egyik típust az asztali kliensek alkotják, amelyek a nagyobb méretű szenzorok vezérlését oldják meg. A másik típusba a hordozható kliensek tartoznak, amit a vizsgált személy a testére erősítve visel. A kliensek alapvető feladata a szenzorok vezérlése és a begyűjtött adatok vagy detektált események továbbítása a szerver felé. Mindezek mellett, az asztali kliensek rendelkeznek egy szoftveresen létrehozott utasítás értelmezővel. Ennek segítségével a kliensek vezérelhetőek a távoli szerverről is, a beépített utasítások elküldésével. Mindegyik asztali kliens egyedien vezérelhető és tesztelhető. A szerver szabályos időközönként kezdeményez adatbegyűjtést a kliensektől. Ezen túlmenően az utasítások segítségével a felhasználó tetszőleges pillanatban is kérheti a szenzorok értékeinek begyűjtését. A két típusú kliens képes a vezetékes és a vezeték nélküli szenzorokat is kezelni.

A szerver oldalon egy Java-ban elkészített szerverprogram fogadja a beérkező információt. A szerver a hozzá beérkező csomagokból kinyeri a hasznos adatokat és letárolja egy MySQL adatbázisban. A szerverről a felhasználók elérhetik a bejelentkezett klienseket és kommunikálhatnak is velük. Továbbá, a szerver oldalon történik az adatok feldolgozása is, ami jelenleg statisztikai és neurális hálózati algoritmusok alapján történik. Az 1. ábrán az általunk készített rendszer egyszerűsített struktúrája látható. (A dolgozatban a képek angol feliratosak)



1. ábra. A rendszer egyszerűsített struktúrája

3. A rendszer hardveres elemei

Jelenleg az asztali kliensek egy Chipkit Max32 mikrovezérlő kártyára épülnek. A kártya jellegzetessége, hogy egy Pic32MX795F512 mikrokontrollert tartalmaz és az alap órajele 80 MHz. A kártya még rendelkezik egy 512 K flash memóriával és egy 128 K RAM - al. 83 input-output vonalat biztosít és a soros kommunikációt is támogatja.

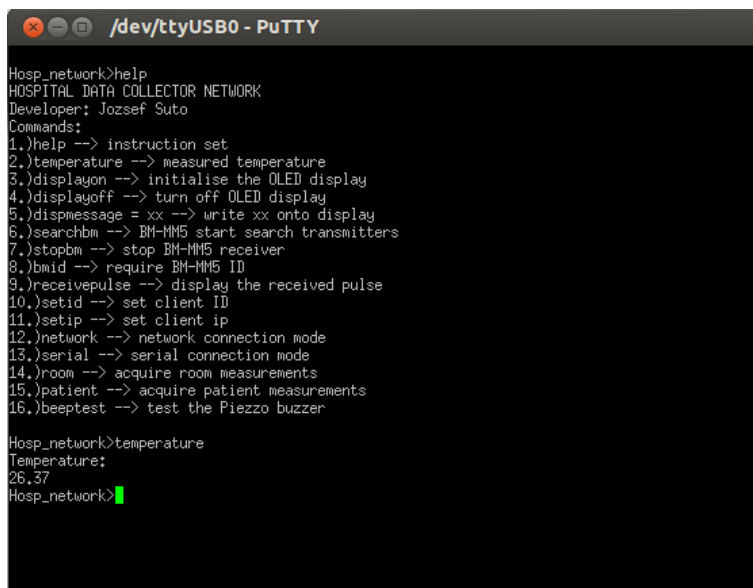
A hordozható kliensek Raspberry Pi (RPi) miniszámítógépek. Az RPi központi egysége egy ARM11 SoC (system on chip), amelynek az órajele 700 MHz. Az RPi nagy előnye, hogy több különböző Linux disztribúció futtatására képes, ami sok esetben rengeteg előnnyel jár a programozás szempontjából. A kliens típusának megfelelően különböző kiegészítőkkel bővülnek. Az asztali kliens estében az alap kártyához csatlakozik egy Ethernet modul, amely a hálózati kommunikációt biztosítja. Az Ethernet modul legfontosabb része az SMSCLAN8720 10/100 PHY (physical layer), ami lényegében az OSI modell fizikai rétegét fedi le. Ezen felül, egy IO modullal is ki lett bővítve, ami lényegében a felhasználói felületet adja. Az IO modul magába foglal egy OLED kijelzőt, amelyre a szerverről üzeneteket lehet küldeni. Ez sok esetben lehet hasznos, például, ha a páciens valamilyen eseményről szeretnének értesíteni. Amint egy üzenet beérkezik az asztali klienshez, a kliensen lévő Piezzo hangszóró egy hangos sípszóval hívja fel a páciens figyelmét az üzenet beérkezésére. A monitorozó rendszerben jelenleg egy úgynevezett „e-Health Platform V1.0” szenzorjait alkalmazzuk [35]. A platformban lévő szenzorok lehetővé teszik egy vizsgált személy életjeleinek megfigyelését, amire további kutatások építhetők. A 2. ábra szemlélteti az „e-Health Platform”-ot (forrás: <http://www.cooking-hacks.com/ehealth-sensors-complete-kit-biometric-medical-arduino-raspberry-pi>), amely a következő egészségügyi adatok mérését teszi lehetővé: vérnyomás, test hőmérséklete, vér oxigénszintje, véráramlás, elektrokardiogramm, pulzus, elektromiográfia.



2. ábra. E-health platform

4. A kliens oldal

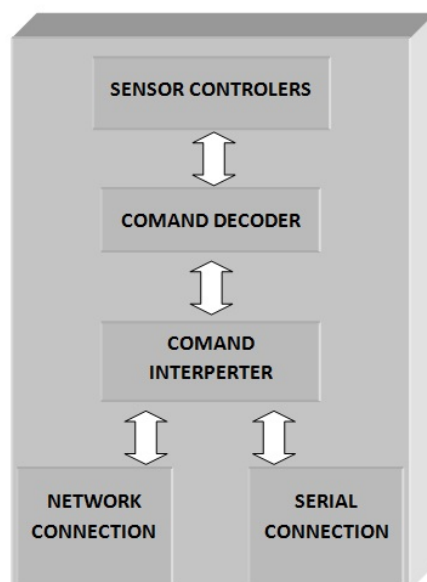
A kliensen futó programkód C++-ban készült. A Microchip cég a fejlesztők számára biztosít C/C++ fordítót és egy egyszerűen kezelhető fejlesztői környezetet (MPIDE) a Chipkit eszközökhöz. A Chipkit eszközökhöz tartozó eljárásokat és függvényeket osztályokba szervezik és egy eszközt a hozzá tartozó osztály egy példányaként érhetjük el. A kliensek elsődleges feladata a szenzorok kezelése és a mért értékek továbbítása a szerver felé. Az asztali kliensek tipikusan a vezetékes szenzorok működtetését végzik. A 2. ábrán látható érzékelők többsége ebbe a kategóriába sorolható. Ekkor a vizsgált személynek nem szabad mozognia a mérés során. Ezzel szemben a hordozható kliensek elsődlegesen a páciens tevékenységeinek megfigyelésében nyújtanak ki-magasló szerepet. Az alkalmazott szenzortól függően, a szenzort vezérlő protokollok többnyire eltérőek. Célszerű az egyes szenzorokhoz készített eljárásokat egy fejláblományba vagy osztályba foglalni, így a nagyméretű kód hordozhatóak és átláthatóbbak lesznek. Az klienst vezérlő programkód alapvetően a felhasznált hardverek összehangolását végzi, ahol a hardverek valamely osztály egyedeként jelennek meg. A kliensek eltérő kommunikációs lehetőségeket biztosítanak. Az asztali és a hordozható kliensek is támogatják a soros kommunikációt. Ennek segítségével a felhasználó beállíthatja a kliens alapvető paramétereit, mint például az azonosítója vagy a statikus IP címe. Ezek a paraméterek kötelezően egyediek minden kliens esetén és csak a soros interfészen keresztül állíthatók be. Ezen felül, hasznos lehet a kliens tesztelésekor és hibakeresési célokra is. A soros kapcsolat egy egyszerű terminálablak segítségével hozható létre, a megfelelő sebesség és portszám kiválasztását követően. A 3. ábra a soros kapcsolatot mutatja be a *putty* program termináljában. A szerver és a kliensek közötti kapcsolat IP alapú kommunikációra épül. Az asztali kliensek vezetékes (Ethernet), míg a hordozhatóak vezeték nélküli (WIFI) protokollokra támaszkodnak. A szenzoroktól begyűjtött információt IPv4 csomagokba ágyazva juttatják el a szervernek. Az egyik jellegzetessége az asztali klienseknek, hogy egy beépített parancsértelmezővel és dekódolóval rendelkeznek. Aktuálisan 16 utasítás van implementálva, amelyek a kliens paraméterezését, tesztelését és vezérlését végzik. Az utasításoknak köszönhetően a klienseket a távoli szerverről is tudjuk vezérelni. Amint egy utasítás érkezik a mikrovezérlőhöz az utasítás értelmező összehasonlítja a beépített utasításokkal, és ha talál egyezést, akkor az utasítás kódját (minden utasítás rendelkezik egy egyedi azonosítóval) átadja az utasítás dekódolónak. Az utasítás kódja alapján a dekódoló meghívja az utasítást kiszolgáló rutint, amely elvégzi a kért feladatot és valamilyen nyugtázó üzenettel tér vissza. A 4. ábrán látható a kliens programjának főbb blokkjai.



```
/dev/ttyUSB0 - PuTTY
Hosp_network>help
HOSPITAL DATA COLLECTOR NETWORK
Developer: Jozsef Suto
Commands:
1.)help --> instruction set
2.)temperature --> measured temperature
3.)displayon --> initialise the OLED display
4.)displayoff --> turn off OLED display
5.)dispmesssage = xx --> write xx onto display
6.)searchbm --> BM-MM5 start search transmitters
7.)stopbm --> stop BM-MM5 receiver
8.)bmwid --> require BM-MM5 ID
9.)receivepulse --> display the received pulse
10.)setid --> set client ID
11.)setip --> set client ip
12.)network --> network connection mode
13.)serial --> serial connection mode
14.)room --> acquire room measurements
15.)patient --> acquire patient measurements
16.)beepptest --> test the Piezzo buzzer

Hosp_network>temperature
Temperature:
28.37
Hosp_network>
```

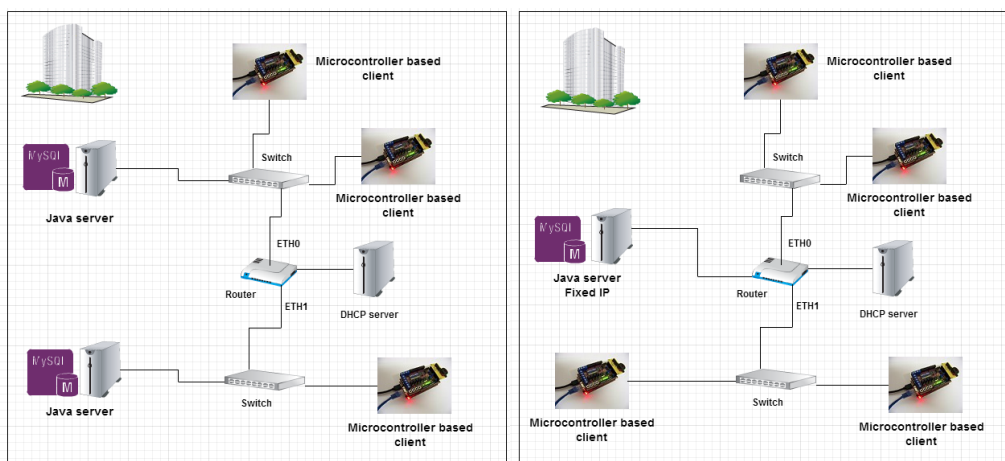
3. ábra. Soros kapcsolat



4. ábra. A kliens programjának komponensei

5. Hálózati kommunikáció

A kliensek a bekapcsolást követően megpróbálnak IP címet igényelni a DHCP szervertől. Abban az esetben, ha a DHCP szolgáltatás nem érhető el a hálózaton, akkor a kliensek a statikus IP címeken keresztül is képesek kommunikálni. Jelenleg a szerver és a kliensek közötti kommunikáció IPv4 protokollra támaszkodik. Az egyszerűség érdekében minden kliens ugyan azt a portot használja a kommunikációhoz (2013-as). A hálózati rétegeket tekintve, az adatkapcsolati rétegben minden csomópontnak egyedi MAC címmel kell rendelkeznie. A mikrovezérlő kártya hátoldalán a gyártó felajánl egyet a PHY-hoz, és ezt érdemes használni a programban. A kliens tervezésekor kitüntetett figyelmet fordítottunk arra, hogy a klienst sokoldalúan lehessen alkalmazni. Ennek alapján a kliens hálózati kommunikációért felelős kódrészének is alkalmazkodnia kell az felhasználási területhez. Először azt az esetet kell vizsgálni, amikor a felhasználó otthon szeretné használni a klienst. Ekkor a kliensnek tudnia kell a szerver IP címét, amely egy fix cím lesz. Ekkor a szerver és a kliens közötti kommunikáció a globális hálózaton keresztül történik és az IP címek is globálisak lesznek [13]. A klienseket használhatjuk egy intézményen belül is, ahol több alhálózat van kialakítva és minden alhálózaton belül van egy szerver. Napjainkban még a legtöbb intézet NAT (Network Address Translation) dobozon keresztül csatlakozik a globális hálózathoz. Ez azt jelenti, hogy az intézmény csak egy-két globális IP címmel rendelkezik és az intézetben belül lokális címeket osztanak ki a csomópontoknak speciálisan erre a célra kialakított tartományokból [12]. A NAT doboz az intézményi és a globális hálózat határán helyezkedik el és ez végzi a duplex címfordítást. Ebben az esetben a klienseknek és a szervereknek is lokális címük lesz. A szervernek nem szükséges fix IP-t használni, mivel az alhálózaton belül broadcast üzenetben eljuttatja az alapvető paramétereit a klienseknek. A szerver a broadcast üzeneteket természetesen szabályozható időközönként küldi. A kliens az üzenetből kinyeri és letárolja a szerver paramétereit, majd automatikusan válaszol egy „bejelentkező” üzenettel. Ez azt jelenti, hogy a kliensek automatikusan jelentkeznek be a szerverre, amint érzékelik a szerver jelenlétét az alhálózaton. Ekkor a kliensektől érkező csomagok nem hagyják el az alhálózatot. Végül, azt az esetet is figyelembe kell venni, amikor az intézetben belül csak egyetlen szerver van. Ekkor az előző két lehetőségre támaszkodhatunk. A kommunikáció a szerver és a kliensek között a helyi hálózaton belül zajlik, de ekkor a szervernek kötelezően fix címet kell használnia. Ebben az esetben a broadcast alapú értesítés nem működik, hiszen a routerek nem továbbítják a broadcast üzeneteket. Az alkalmazási területtől függően a kommunikációs folyamat kissé eltérő. A klienst működtető program támogatja az előbbieken felsorolt lehetőségeket, amelyet a felhasználónak kell beállítania a

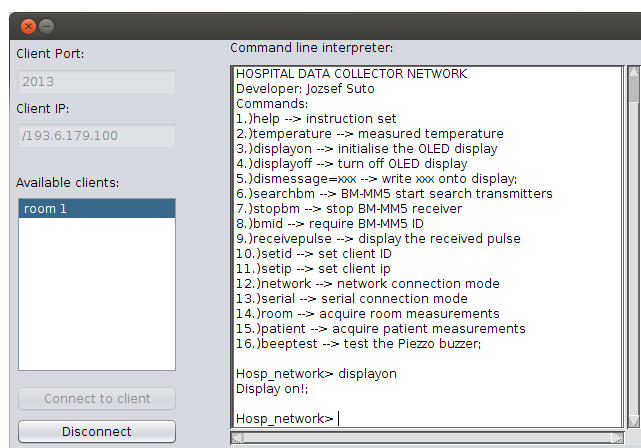


5. ábra. A rendszer alkalmazási lehetőségei

felhasználói felületen keresztül. Egy intézetben belül a kliensek elrendezésének az imént említett lehetőségeit mutatja a 5. ábra.

6. A szerver oldal

A szerver program Java programozási nyelven lett megírva. A Java platform független programozási nyelv, amely lehetővé teszi a fejlesztőknek a számítógép specifikus tulajdonságainak elérését a standard könyvtárak segítségével. A szerver egyszerűen és rugalmasan kezelhető. Elsődleges feladatai az adatbázis kezelés és az adatkiértékelés biztosítása. A program minden egyes részéhez külön grafikus felhasználói felületek (GUI) tartoznak, amelyek leegyszerűsítik a program kezelését. A szerver felületen a felhasználók megtekinthetik a bejelentkezett klienseket és kommunikálhatnak is velük. A klienseken futó utasítás értelmezőnek köszönhetően, a kliens vezérelhető a szerverről az utasítások felhasználásával. A kliensek beregisztrálását monitorozó felületen a felhasználó tetszőlegesen választhatja ki, hogy melyik klienssel szeretne kommunikálni. A kliens kiválasztását követően a kliens alapvető paraméterei kilistázódik. Miután kapcsolatot kezdeményezünk valamelyik klienssel, a grafikus felület jobb oldalán lévő szövegblokkban megjelenik a „Hosp network>” prompt. Ezt követően lehet begépelni a parancsokat, amelyet el szeretnénk küldeni a kliensnek. A parancs végét egy enter leütés jelzi. A 6. ábrán tekinthetjük meg a klienseket nyilvántartó felületet. Miután egy kliens bejelentkezett a szerverre, a Java kódban egy új szál jön létre és a kliens hozzárendelődik az új szálhoz. Ez teszi lehetővé, hogy a felhasználó egyenként kommunikálhasson a kliensekkel, hiszen a kommunikáció az elkülönített szálon keresztül fog zajlani.

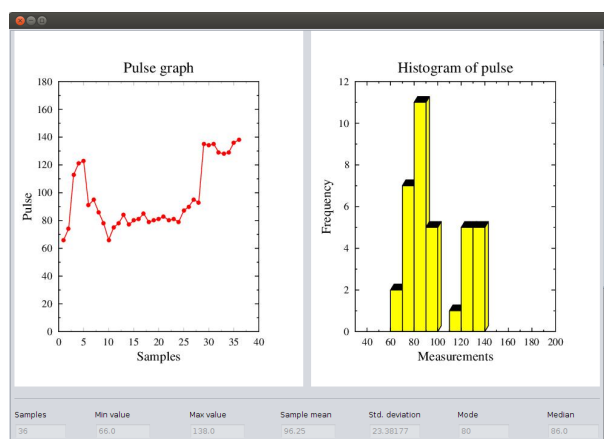


6. ábra. A kliensek nyilvántartása

Az adatbázis kezelés szintén a Java kódból történik. Mivel az adatbázis karbantartása (beszúrás, törlés, frissítés, stb.) meglehetősen sok műveletet generál, így a szerver programjában az összes adatbázissal kapcsolatos függvényt és eljárást egyetlen osztályba foglaltuk. A Java kódból a MySQL adatbázis elérését a JDBC (Java Database Connectivity) konnektor teszi lehetővé. A programban a MySQL utasítások be vannak ágyazva a függvények törzsébe és dedikáltan az adatbázis valamelyik táblájához kapcsolódó műveletet hajtanak végre (például lekérdezést).

Az adatbázis tervezésekor kimagasló szerepet játszott az elsődleges kulcsok helyes meghatározása. Az adatbázisban a mért értékek külön táblákban helyezkednek el, ahol összetett elsődleges kulcsokat használunk. Ezekben a táblákban az összetett kulcs egyik attribútuma a páciens azonosítója, míg a másik egy időbélyeg. Amint egy adatsomag beérkezik a szerverhez, a szerver kiolvassa a csomagból a hasznos információt, majd beszúrja az adatbázisba, ahol az új bejegyzéshez hozzárendelődik az aktuális időbélyeg. Az adatbázis kezelésnél az egyik kritikus kérdés, annak meghatározása, hogy a beérkező adat pontosan kihez tartozik. Mivel mindegyik kliensnek van egy saját azonosítója, így a felhasználót a klienshez lehetne társítani. Ez azt a problémát eredményezi, hogy minden személynek egy saját klienst kellene megvásárolnia, ami nem gazdaságos. Jelenleg mi a kliensek azonosítójára támaszkodunk, de a későbbiekben ezt a hibát szeretnénk orvosolni. Egy olyan megoldáson dolgozunk, amely a páciens az új lenyomata alapján azonosítja és addig nem kezdi el a mérést, amíg nem határozta meg, hogy a mért értékek kihez tartoznak.

A szerver biztosít egy kezdetleges adatelemzési lehetőséget is. Aktuálisan, adatelemzéskor az adathalmazok alapvető statisztikai mutatóit számítjuk: várható érték, szórás, módusz, medián, minimum, maximum. Az adatok kiértékelését segítő grafikus felületen kilistázódnak az adatbázisban lévő személyek, amelyek közül kiválaszthatjuk, hogy melyik páciens mért értékeit szeretnénk megtekinteni. Valamely személy kiválasztását követően a hozzá tartozó adathalmazokhoz kiszámolódnak a statisztikai indikátorok és kirajzolódik az adathalmazok grafikonja, illetve az adathalmaz hisztogramja. Linux operációs rendszer alatt a magas szintű programozási nyelvekből egyszerűen hívhatóak szkriptek. A programban a grafikonok megrajzoltatásánál ezt a lehetőséget használtam ki. Egyszerű bash szkriptekbe ágyaztam a GLE (Graphics Layout Engine) és gnuplot grafikus szkript nyelvek utasításait, valamint megadtam, hogy mely fájlban találhatóak a kirajzoltatáshoz szükséges értékek. A szkript nyelv lényegében kap egy bemeneti fájlt, amelyben a grafikon értékei vannak és néhány alapvető paramétert, ami alapján elkészíti az ábrát. Amikor a felhasználó rákattint az adatkiértékelés gombra, a háttérben a Java főszála egy rövid időre blokkolódik. Ez alatt az idő alatt a GLE elkészíti a grafikonokat, amit a Java program grafikus felületére fogunk kihelyezni. A felfüggesztett főszál egy rövid várakozás után folytatja a futását és felhasználja a GLE által elkészített ábrákat. Feltehetőleg nem ez a legpraktikusabb módja a különböző grafikonok kirajzoltatásának, viszont ez egy meglehetősen egyszerű módszer, ami egyben hatékony is. Miután minden művelet befejeződött, az adatok kiértékelését segítő grafikus felületen megjelennek a grafikonok és a szövegdobozokban az eredmények. A 7. ábrán látható az adatkiértékelést támogató felület.



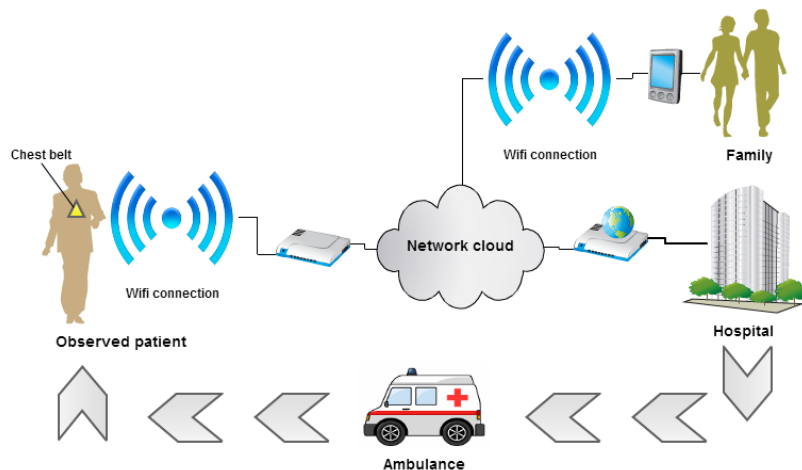
7. ábra. Az adatkiértékelést támogató felület

7. Tevékenység felismerés

Napjainkban az egyik leggyorsabb ütemben növekvő kutatási terület az emberi viselkedés és tevékenységek megfigyelése. Ennek a kutatási területnek a népszerűsége az előző években kezdett el kibontakozni és fokozatosan egyre több egyéni kutató és kutatóintézet kapcsolódik be.

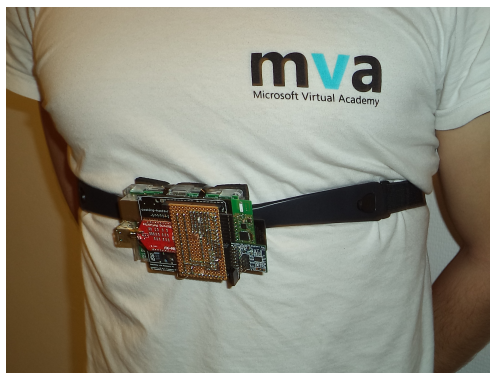
Az eddig kidolgozott mozgásfelismerő technikák többsége egy vagy több gyorsulásszenzortól begyűjtött jelek feldolgozásán alapszik. Az emberi test különböző pontjaira csatolnak kisméretű adatgyűjtő eszközöket, amelyek az adott testrész mozgásváltozásáról szolgáltatnak információt. A begyűjtött információk alapján lehet következtetéseket levonni a megfigyelt személy mozgásáról, viselkedéséről és mindezekből adódóan, az egészségi állapotáról is. Ebben a témában számos cikk született a gyorsulásszenzorok felhasználási lehetőségeiről [1-5]. Ezek a tanulmányok azt bizonyítják, hogy gyorsulásszenzorok segítségével kategorizálható az emberi mozgás. Néhány tanulmányban a szenzorokat a test fix pontjaira rögzítették, mint például, a mellkas vagy csípő. Ezekben az esetekben a felsőtest mozgásának elemzése a cél, emiatt elegendő egyetlen adatgyűjtő készülék [1, 2]. Más esetekben, amikor a testtartás elemezzük, akkor több szenzort helyeznek a test különböző pontjaira, mint például, fej, mellkas, sípcsont, lábfej [3, 4, 5, 6]. A mozgás analízis meglehetősen széleskörű. Magába foglalja a nemvárt események figyelését, mint például az esések, a testtartásbeli problémák figyelését: hátfájás, túlsúly, fiziológiai remegések, a mentális rendellenességek meghatározását és néhány betegség (Parkinson kór, csontízületi gyulladás, stb.) vizsgálatát [2, 4, 6]. Ezen felül, ide tartozik a mozgási minták megfigyelése is. Ez lényegében egy átlagos ember mindennapi mozgássorozatának elemzését foglalja magába. Úgy mint az ülés, állás, sétálás, kocogás, futás, lépcsőzés és még sok más hétköznapi tevékenységet. Hasonlóan a mozgásfelismeréshez, a nem várt események detektálása is nagyon fontos szerepet tölt be, főként az idősebb korosztálynál [7, 8, 9]. A mozgás analízis nagy segítséget nyújt az idősebb és mozgáskorlátozott páciensek távoli megfigyelésében is. Ez azért is fontos, mert az európai társadalom egyre jobban elöregszik, így az életvitel javítására szolgáló eszközök és rendszerek kereslete fokozatosan növekszik [1]. Az említett tanulmányok hatására, a monitorozó rendszerhez kapcsolódóan, mi is elkezdtük elemezni az emberi tevékenységeket. Az volt a kítűzött célunk, hogy létrehozzunk egy általánosan alkalmazható mozgásfelismerő rendszert, amely valós időben ismeri fel a mozgási mintákat és alkalmazható a különböző típusú esések detektálására is. Az általunk létrehozott rendszer felépítését és elvi működését illusztrálja a 8. ábra.

Az ábrán láthatóan, a megfigyelt személy a mellkasán viseli az adatgyűjtő eszközt, amely egyben az adatkiértékelést is végzi. Amennyiben egy váratlan vagy kitüntetett esemény bekövetkezik, akkor az eszköz erről értesítést küld a megfigyelt személy hozzátartozóinak és a központi szervernek, amit célszerű valamilyen egészségügyi központban elhelyezni. Ez lényegében egy távfelügyeleti rendszerként is elképzelhető. Amennyiben a kitüntetett esemény bekövetkezik, akkor erre gyorsan tudnak az illetékes személyek reagálni és segítséget tudnak nyújtani.

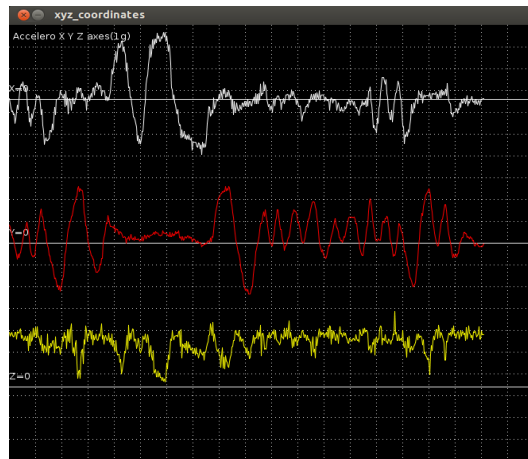


8. ábra. A mozgásmegfigyelő rendszer felépítése és elvi működése

Mivel egy folytonos megfigyelőrendszerrel van szó, így az adatgyűjtő eszközt a megfigyelt személynek napközben folyamatosan hordania kell. Ebből adódóan az eszköznek hordozhatónak és kis méretűnek kell lennie. Az egyik legfontosabb kritérium az eszközzel szemben, a költsége. Mivel a mi megoldásunkban csak egyetlen adatgyűjtő eszközre van szükség, ezért ez a megoldás költséghatékonyabb azokkal szemben, amelyek több adatgyűjtőt igényelnek. Végül, az általunk adott megoldás nem igényel kórházi környezetet [1, 2, 3]. Ahogy már említve volt, az általunk összeállított adatgyűjtő eszköz egy Raspberry Pi (RPi) miniszámítógépen alapszik. Az RPi egy ADXL345 3 tengelyű gyorsulásszenzort és egy Roving RN-171 WIFI modult vezérel. Az eszköz egy vékony lemezre van felfogatva, ami stabil pozíciót biztosít a vizsgált személy mellkasán. A 9. ábra illusztrálja a összeállított eszközt. A már meglévő cselekvésfelismerő technikák legnagyobb hátránya az adatbegyűjtés és az elemzés szétválasztása [1-6]. Ez azt jelenti, hogy először adatokat gyűjtenek be az elemzéshez, majd később a számítógépen elemzik általában Matlab segítségével. Ezzel szemben a mi megoldásunk szinte párhuzamosan végzi az adatbegyűjtést és a kiértékelést. Ez az egyik legfontosabb különbség a már meglévő módszerekkel szemben. Továbbá, a kidolgozott mozgáselemző algoritmusunk egy mintafelismerő algoritmuson alapszik. Az alkalmazott mintafelismerő megbízhatóbb eredményt ad más, a szakirodalomban javasolt módszereknél (pl.: konvolúció). Ez az egyszerű módszer egy "ideális" mintát léptet végig az időben változó jelen és minden pontban kiszámolja az összegzett négyzetes hibát. Mivel a legtöbb mozgáshoz jellemezhető egy sajátos időtartománybeli mintasorozattal, így, ha egy algoritmus képes felismerni a cselekvéshez tartozó mintákat, akkor képes a cselekvés azonosítására is. Első lépésként, a tevékenység felismeréshez szükséges algoritmus kifejlesztését természetesen mi is számítógépen próbáltuk meg kidolgozni. Miután a vizsgálatainkból kiderült, hogy az elképzeléseink működőképesek, ezt követően átültettük a Raspberry Pi-ra is. Az első fázisban létrehoztunk egy egyszerű grafikus szoftvert, amely fogadja a gyorsulásszenzor adatait, amit grafikusan meg is jelenít és a kapott adatokat fájlba menti. Mivel a gyorsulásszenzor az adatokat kettes komplementben tárolja, így az adattovábbítás előtt az RPi elvégzi az átalakítást. A program „PROCESSING”-ben készült, ami egy programozási nyelv és egy hozzá tartozó fejlesztői környezet. A PROCESSING teljes mértékben Java alapú, így az ebben elkészített program és a szerver program között egyszerűen megoldható a kapcsolat, ha szükség van rá. A programban négy opció van beépítve. Az első három a gyorsulásszenzor x, y, z értékeit ábrázolja folyamatosan a képernyőn. A negyedik opcióban, a három értéket együttesen ábrázolja. A 10. ábrán tekinthetjük meg a PROCESSING-ben létrehozott programot, amely a gyorsulásszenzortól beérkező nyers értékeket ábrázolja. A gyorsulásszenzorból kinyert nyers értékek meglehetősen zajosak, ami az ábrán is megfigyelhető. Mivel a mi mozgásfelismerő algoritmusunk mintafelismerésen alapul, ezért egy időtartománybeli szűrést fogunk elvégezni a gyorsulásszenzor paramétereiből származtatott jelen. A begyűjtött adatokból kinyert információ alapján következtetni lehet a vizsgált személy szokásaira és viselkedésére is. A különböző cselekvések felismerése mellett a begyűjtött adatokból más hasznos információkat is ki tudunk nyerni, amennyiben nemcsak idő, hanem frekvenciatérben is végzünk vizsgálatokat. Ennek alapján, ha a sétálás alatt összegyűjtött adatahalmazunkat a frekvenciatérben vizsgáljuk, akkor az adatahalmazból kinyerhető a vizsgált személy lépéseinek gyakorisága.



9. ábra. Az összeállított adatgyűjtő eszköz



10. ábra. A gyorsulásszenzor paramétereit megjelenítő program

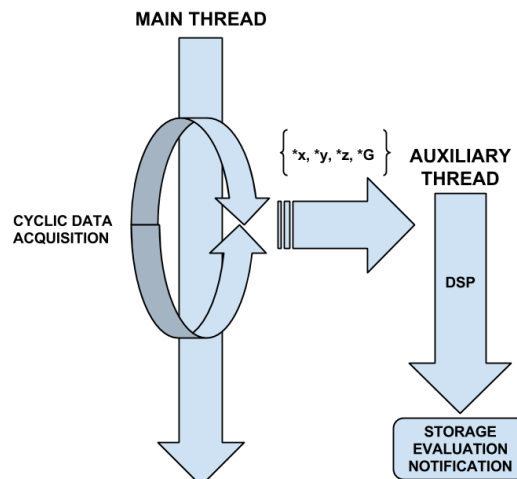
7.1. Párhuzamos működés

Egy RPi-n futó C++ program látja el az adatbegyűjtést és a adatkiértékelést. A programban a POSIX vagy Pthread könyvtár segítségével van megoldva a párhuzamosítás [14]. A Pthread lehetőséget biztosít többszálú programok készítésére C/C++ nyelven. Az adatbegyűjtés és az adatkiértékelés két különálló feladat, ezért párhuzamosan is végrehajthatóak.

Az analízáló program két szálát használ. A főszál felelős az ciklikus adatbegyűjtésért, míg a segéd szál végzi a feldolgozást. Amíg az eszköz aktív állapotban van, addig folyamatosan gyűjti az adatokat a gyorsulásszenzortól, amit vektorokba tárol. Jelenleg az mintavételezési frekvencia 150 Hz és a vektorokba 2^9 elemet lehet letárolni. Lényegében, a folyamatos jelsorozatot rövid darabokra tördeljük (megközelítőleg 3.5 másodperces) és ezekre végezzük el az elemzést.

Amint a vektorok feltöltődtek adatokkal, a segéd szál elindul és paraméterül kap egy *void** struktúrát, ami a vektorokat tartalmazza. A Pthread csak egyetlen paraméter átadását engedélyezi, tehát ha több paramétert szeretnénk átadni, akkor azt egy struktúrába kell beágyazni.

A segéd szál elindulása után meghívja a digitális jelfeldolgozó algoritmusokat (DSP) és a min-tafelismerő függvényeket. Továbbá, a segéd szál végzi az adattárolást és az értesítést. Amint az adatelemzés befejeződött, az eredmény letárolódik az RPi memóriakártyájára és ha egy vizsgált esemény bekövetkezik, akkor erről a szervernek értesítést küld. Egy fontos kikötés, hogy a segéd szálnak hamarabb be kell fejeződnie, mint ahogy a következő elemzési folyamat indulna, így elkerülhető a szálütközés. A 11. ábra szemlélteti a két szál kapcsolatát és feladataikat. Az ábrán a **x*, **y*, **z* a gyorsulásszenzor x, y, z komponenseit tartalmazó vektorokra utal, valamint a **G* tároló tartalmazza a normalizált gyorsulásvektor nagyságát.



11. ábra. A párhuzamos működés folyamata

7.2. A mozgásfelismerő algoritmus leírása

A mozgásanalízis idő és frekvenciatérbeli jelfeldolgozáson alapul. Első lépésben, a program frekvenciatérben kategorizálja a gyorsulásvektor nagyságának változását. Ezt követően, a kiválasztott frekvencia kategóriának megfelelően, a kategóriához tartozó mintafelismerő függvények pásztázzák át a jelet. Az algoritmus működésének folyamatábráját mutatja a 12. ábra.

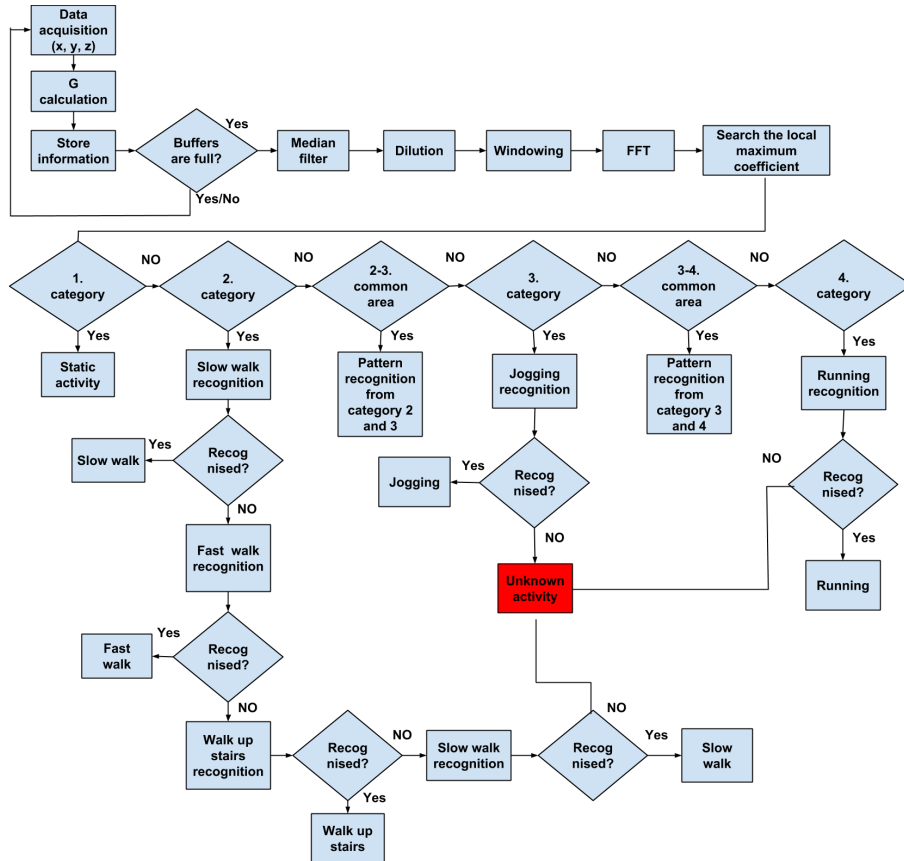
Amint azt a 11. ábra is szemlélteti, a program az adatbegyűjtés közben számolja ki a normalizált gyorsulásváltozást ($G[i]$) a szenzor x, y, z paramétereiből (a dolgozat hátralévő részében erre fogok hivatkozni jelkét).

$$G[i] = \frac{\sqrt{x[i]^2 + y[i]^2 + z[i]^2}}{lg} \quad (1)$$

ahol az lg a gyorsulásszenzor felbontásától függ (a beállított érzékenységi tartományban lg mekkora értékű). A kiszámított $G[i]$ értékek jól definiálják a mozgás dinamikáját és ritmusát [1, 7]. A G vektort frekvenciatérben vizsgálva meghatározható a mozgás frekvenciája. Amint tudjuk, a Fourier transzformáció segítségével lehet meghatározni a jelben előforduló frekvenciakomponenseket. A transzformáció előtt érdemes kibővíteni ("higítani") az időtartománybeli jelet nullákkal a végén. Ez az egyszerű művelet növeli a frekvenciatérbeli felbontást, ami az egymáshoz közeli frekvenciakomponensek elhatárolódását eredményezi [21, 22]. Ennek következtében, a jel mérete 2^9 -ről 2^{10} -re bővül. Mivel a frekvenciatérbeli felbontás növekedett így a kategorizálás is egyszerűsödik. A kibővítés után, egy ablak függvénnyel szorozzuk össze a jelet, aminek köszönhetően, a spektrumban lévő meghatározó komponensek jobban kiszűrhetők [21, 22]. Ablakozásra a Blackman-Nuttall függvényt használtuk, amit a következőképpen definiálunk:

$$\omega[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cos\left(\frac{4\pi n}{N-1}\right) - a_3 \cos\left(\frac{6\pi n}{N-1}\right) \quad (2)$$

ahol $a_0 = 0.3635819$, $a_1 = 0.4891775$, $a_2 = 0.1365995$, $a_3 = 0.0106411$ és N a jel mérete.



12. ábra. A mozgásfelismerő algoritmus folyamatábrája

Annak érdekében, hogy a segédszál minél gyorsabban be tudja fejezni az adatkiértékelést, egy optimalizált gyors Fourier transzformációt (FFT) használtunk [15]. Mivel a program ciklikusan futtatja az FFT-t, így már egy kis gyorsítás is nagy hasznot hoz. Az általunk létrehozott FFT algoritmus jól alkalmazható bármilyen platformon és egyszerűen megvalósítható az alacsony szintű programozási nyelveken is.

Az FFT lényegében egy gyorsított számítási módszer a diszkrét Fourier transzformációhoz (DFT). A DFT-t a (3) képlettel írhatjuk le.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i \frac{2\pi}{N} kn} \quad (3)$$

amit felírhatunk egy kompaktabb formában is,

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad (4)$$

ahol az $X(k)$ a k . frekvencia komponens és,

$$W_N = e^{-i \frac{2\pi}{N}}$$

W_N^k , $k=0, \dots, N-1$ az N . gyöke 1-nek [16]. Minden komplex számnak N darab N . gyöke van. Jelen esetben a gyökök az egységkörön helyezkednek el a komplex számsíkon. Az Euler formula felhasználásával a gyökök a következő alakban is megadhatóak:

$$W_N^k = \cos(k \frac{2\pi}{N}) - i \sin(k \frac{2\pi}{N}) \quad (5)$$

A képlet felhasználásával szétválasztható a komplex szám valós és képzetes részekre, ahol a valós rész a $\cos(k \frac{2\pi}{N})$, a képzetes rész pedig a $-i \sin(k \frac{2\pi}{N})$. Az FFT lényege, hogy az időben változó jelet $x(n)$ szétbontja a páros és páratlan összetevőire [24].

$$\left. \begin{aligned} e(n) &= x(2n) \\ o(n) &= x(2n+1) \end{aligned} \right\} \quad n = 0, 1, \dots, \frac{N}{2}-1 \quad (6)$$

Így a DFT (3) felírható,

$$\begin{aligned} \sum_{n=0}^{N/2-1} e(n) W_N^{2nk} + \sum_{n=0}^{N/2-1} o(n) W_N^{(2n+1)k} = \\ \sum_{n=0}^{N/2-1} e(n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} o(n) W_{N/2}^{nk} \end{aligned} \quad (7)$$

A fenti forma a következőképpen is megadható,

$$X(k) = E(k) + W_N^k O(k) \quad k=0, 1, \dots, \frac{N}{2}-1 \quad (8)$$

ahol,

$$\begin{aligned} E(k) &= \sum_{n=0}^{N/2-1} e(n) W_{N/2}^{nk} \quad k=0, 1, \dots, \frac{N}{2}-1 \\ O(k) &= \sum_{n=0}^{N/2-1} o(n) W_{N/2}^{nk} \quad k=0, 1, \dots, \frac{N}{2}-1 \end{aligned}$$

A DFT periodikusságát kihasználva,

$$X(k) = E(k - \frac{N}{2}) - W_N^k O(k - \frac{N}{2}) \quad k=\frac{N}{2}, \dots, N-1 \quad (9)$$

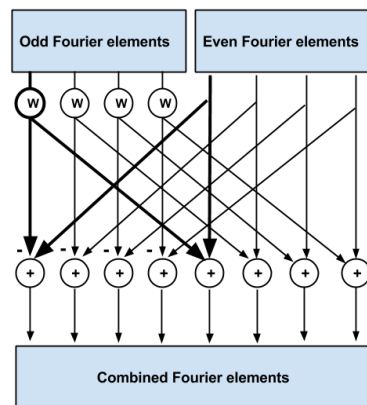
A fenti képletekkel írható le a *radix-2* FFT. Az FFT-ben, a szétbontást (8) (9) addig kell ismételgetni, amíg kételemű vektorokra bontjuk szét a jelet.

Nyilvánvalóan, a DFT (3) N^2 komplex szorzást és $N^2 - N$ komplex összeadást igényel, abban az esetben, ha W_N^{kn} adott. Mivel W_N^{kn} független a begyűjtött jeltől ($x(n)$), ezért W_N^{kn} elkülönítetten is kiszámolható. Az algoritmusban W_N^{kn} két vektorként is tekinthető (valós és képzetes) ami előre meghatározott értékeket tartalmaz. Ez természetesen akkor alkalmazható, ha előre rögzített méretű jelet kell transzformálni.

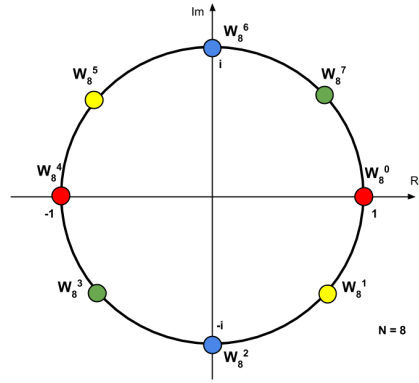
Az N^2 komplex szorzás egyenlő $4N^2$ valós szorzással és $2N^2$ összeadással, míg $N^2 - N$ komplex összeadás $2N^2 - 2N$ valós összeadásnak felel meg. Ennek alapján a DFT $4N^2$ valós szorzást és $4N^2 - 2N$ valós összeadást számol ki. A szétbontásnak köszönhetően az FFT $\frac{N}{2}$ komplex szorzást, $\frac{N}{2}$ komplex összeadást és $\frac{N}{2}$ komplex kivonást igényel minden szinten. Tehát a szorzások, osztások és kivonások száma $\log_2 N \frac{N}{2}$. Azonban, az utolsó két szinten a W_N^k $\{1, -1\}$ és $\{1, -1, i, -i\}$. Ezért, ezeken a szinteken nem kell szorzást végezni, ami előnyt jelent. Napjainkban már létezik több összetett *valós idejű* FFT algoritmus, amik gyorsabbak a *hagyományos* FFT-nél. A hagyományos FFT-k csak akkor kezdik el a transzformációt ha a teljes jel beérkezett, ezzel szemben a valós idejű FFT algoritmusok az adatbegyűjtéssel párhuzamosan számolják a *pillangó* (13. ábra) struktúrákat. Mivel a valós idejű FFT hamarabb kezdi el a számításokat, ezért hamarabb ad eredményt, viszont a műveletek száma itt is megegyezik a hagyományoséval. Továbbá, az algoritmus megvalósítása jóval bonyolultabb [17, 18, 19]. Ezzel szemben az általam készített algoritmus egyszerűen implementálható minden programozási nyelven, mivel az algoritmus struktúrája főként logikai műveleteken alapul. Ezért, az algoritmus gyorsabb mint más általános célú implementációk [15]. Az FFT algoritmus úgy is tekinthető, mint pillangó minták hálózata [23]. Az 13. ábrán látható a pillangó műveletek általánosított alakja. Az ábrán W -vel van jelölve a megfelelő W_N^k szorzó, úgy mint a (8) (9) képletekben. Ahogyan már említve volt, W_N^k $k=0, \dots, N-1$ az egységkörön elhelyezkedő gyökök, ahol $(W_N^k)^N = 1$ minden k -ra. A gyökök úgy is elképzelhetők, mint egy szabályos poligon csúcsai. Az algoritmusban kihasználtuk a gyökök közötti összefüggéseket, így az előre meghatározott gyököknek csak a negyedét kell kiszámolni.

$$\begin{aligned} W_{N_{Im}}^{k+\frac{N}{4}} &= -W_{N_{Re}}^k \\ W_{N_{Re}}^{k+\frac{N}{4}} &= W_{N_{Im}}^k \\ W_N^{k+\frac{N}{2}} &= -W_N^k \\ W_N^{k+\frac{3N}{4}} &= -W_N^{k+\frac{N}{4}} \end{aligned} \quad (10)$$

A 14. ábra egy példát mutat $N = 8$ gyökre, ahol azonos színnel van jelölve a kapcsolatban álló gyökpárok. Az előre kiszámított gyököket, a pillangó számításban fogjuk felhasználni. A 13. ábrán a megvastagított vonalak illusztrálnak egy pillangó műveletet.



13. ábra. A pillangó műveletek általánosított formában



14. ábra. Az egységkörön elhelyezkedő gyökök $N = 8$ esetén

Matematikailag a pillangó műveleteket a radix-2 Cooley-Tukey FFT algoritmushoz a következőképp adjuk meg [25]:

$$\begin{aligned} E(k) &= e(k) + W_N^k o(k) \\ O(k) &= e(k) + W_N^{k+\frac{N}{2}} o(k) = e(k) - W_N^k o(k) \end{aligned} \quad (11)$$

A fenti képlet két komplex műveletet tartalmaz. A komplex műveletek felbontásával kapjuk a kiterjesztett képletet, amit az algoritmusba is implementáltunk.

$$\begin{aligned} E(k) &= (e(k)_{Re}, e(k)_{Im}) + (W_{N_{Re}}^k, W_{N_{Im}}^k)(o(k)_{Re}, o(k)_{Im}) \\ &= (e(k)_{Re}, e(k)_{Im}) + (W_{N_{Re}}^k o(k)_{Re} - W_{N_{Im}}^k o(k)_{Im}, \\ &\quad W_{N_{Im}}^k o(k)_{Re} + W_{N_{Re}}^k o(k)_{Im}) \\ O(k) &= (e(k)_{Re}, e(k)_{Im}) - (W_{N_{Re}}^k o(k)_{Re} - W_{N_{Im}}^k o(k)_{Im}, \\ &\quad W_{N_{Im}}^k o(k)_{Re} + W_{N_{Re}}^k o(k)_{Im}) \end{aligned} \quad (12)$$

7.2.1. A felhasznált gyors Fourier transzformáció algoritmusa

A mozgásfelismerő algoritmus kezdetekor lefut egy függvény amely kiszámolja és letárolja az FFT-hez szükséges gyök értékeket. Az inicializáló függvényt az 1. pszeudokód mutatja be.

Algorithm 1 W_N^k pre-calculation

Require: $N, *W_r, *W_i$

Ensure: W_N^k

$N1 \leftarrow N \gg 1$

$N2 \leftarrow N \gg 2$

$W_r \leftarrow \text{allocateMemory}(N1)$

$W_i \leftarrow \text{allocateMemory}(N1)$

for $k : 0$ **to** $N2 - 1$ **do**

$W_r[k] \leftarrow \cos(k\pi/N1)$

$W_i[k] \leftarrow -\sin(k\pi/N1)$

$W_r[k + N2] \leftarrow W_i[k]$

$W_i[k + N2] \leftarrow -W_r[k]$

end for

Algorithm 2 FFT algorithm

Require: $N, xr, xi, *Wr, *Wi$ **Ensure:** Xr, Xi $N1 \leftarrow N - 1$ $N2 \leftarrow N \gg 1$ $stages \leftarrow \log_2 N$ **for** $i : 0$ **to** $N - 1$ **do** $n \leftarrow i$ $nrev \leftarrow n$ **for** $j : 1$ **to** $STAGES - 1$ **do** $n \leftarrow n \gg 1$ $nrev \leftarrow nrev \ll 1$ $nrev \leftarrow nrev \text{ or } (n \text{ and } 1)$ **end for** $nrev \leftarrow nrev \text{ and } N1$ $R[i] \leftarrow xr[nrev]$ $I[i] \leftarrow xi[nrev]$ **end for****for** $s : 0$ **to** $stages - 1$ **do** $blocks \leftarrow (N \gg (s + 1))$ $butterf \leftarrow 1 \ll s$ $butterf1 \leftarrow butterf - 1$ $tp \leftarrow 0$ $index \leftarrow 0$ **for** $i : 0$ **to** $N2 - 1$ **do****if** $((i \text{ and } butterf1) = 0) \text{ and } (i \neq 1)$ **then** $index \leftarrow index + butterf$ $tp \leftarrow tp \ll 1$ **end if** $p \leftarrow tp \text{ and } N1$ **if** $s = 0$ **or** $s = 1$ **then****if** $p = 0$ **then** $Wbr \leftarrow R[index + butterf]$ $Wbi \leftarrow I[index + butterf]$ **else** $Wbr \leftarrow I[index + butterf]$ $Wbi \leftarrow -R[index + butterf]$ **end if****else** $Wbr \leftarrow Wr[p] * R[index + butterf] - Wi[p] * I[index + butterf]$ $Wbi \leftarrow Wi[p] * R[index + butterf] + Wr[p] * I[index + butterf]$ **end if** $R[index + butterf] \leftarrow R[index] - Wbr$ $I[index + butterf] \leftarrow I[index] - Wbi$ $R[index] \leftarrow R[index] + Wbr$ $I[index] \leftarrow I[index] - Wbi$ $index \leftarrow index + 1$ $tp \leftarrow tp + blocks$ **end for****end for**

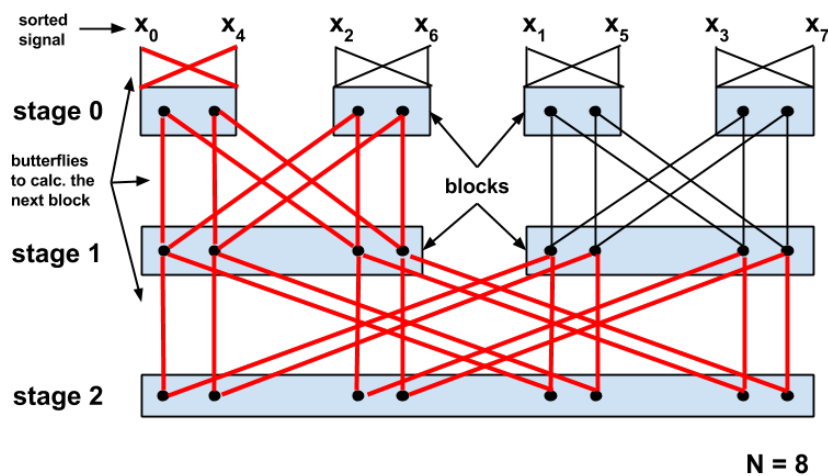
A függvény az Euler formulát felhasználva (5) kiszámítja a W_N^k gyökök valós és képzetes részeit. Mivel a gyökök egymásból származtathatóak (10), a gyököknek csak egy részét kell letárolni. Az inicializáló függvény, amely a gyököket határozza meg, csak egyszer fut le a program elején.

Nyilvánvalóan az előre meghatározott szorzók gyorsítani fogják az FFT-t. Mivel a program futásának teljes időtartamára le kell tárolni az értékeket, ezért a felhasznált adattípustól függően ez több bájt memóriát igényel.

Az általam készített FFT algoritmust a 2. pszeudokód mutatja be. Az FFT egy bit fordított rendezéssel kezdődik [21]. Az időtartománybeli szétszedés (8), (9) nagymértékben egyszerűsíthető, ha a jel vektorának az indexeit bináris alakban írjuk fel és úgy vizsgáljuk meg. Ha megfigyeljük a teljesen szétszort jel indexeinek sorrendjét, akkor azt vehetjük észre, hogy az indexek bitfordított sorrendben helyezkednek el a kiindulási sorrendjükhöz képest. Tehát, a szétválogatást a leggyakrabban bitfordított rendezés felhasználásával oldják meg. Az algoritmusban használt rendezés kizárólag logikai műveletek felhasználásával lett kialakítva. A rendezés után, az algoritmus külső ciklusa végiglépked a $\log_2 N$ szinten, míg a belső ciklus a szinten minden elemen végighalad. A pillangó műveletek a belső ciklusba vannak beágyazva. A külső ciklusban a *blocks* változó jelöli a blokkok számát az aktuális szinten. Ha vizualizáljuk a pillangó mintákat, akkor láthatóvá válik, hogy a pillangó számításban résztvevő két elem két szomszédos blokkból származik. Hasonlóan, a *butterf* változó jelöli a pillangó műveleteknek a számát, amely a következő (vagy újabb) blokk kiszámításához szükséges. A kód megértéséhez a 15. ábra mutat egy példát, $N = 8$ elemre. Az ábrán a piros pillangók utalnak az algoritmusban használt *butterf* változóra, míg a világos kék blokkok és a szintek az algoritmusban használt azonos nevű változókat jelölik.

A belső ciklusban az első feltétel szabályozza az *index* és a *tp* változókat. Az *index* változó segítségével tartjuk nyilván, hogy aktuálisan hány pillangó művelet van kiszámítva, míg a *tp* egy segédváltozó az előre kiszámolt gyökök indexének meghatározásához. A belső ciklusban, amikor $s = 0$ vagy $s = 1$, azt jelenti, hogy az alsó két szinten vagyunk, ahol nem kell szorzást végezni az új elemek meghatározásához.

Az algoritmus futási sebességét összehasonlítottuk két általános célú FFT algoritmussal [21, 26] több különböző platformon. Az eredmények azt mutatták, hogy a mi algoritmusunk sebessége (az előszámítással) a többszöröse, a másik kettőhöz képest. Abban az esetben, ha a mintavételezés frekvenciája nagy és a feldolgozandó jel kicsi (legfeljebb néhány ezer elemű), akkor véleményem szerint felesleges egy valós idejű FFT használata, hiszen egy általános célú is viszonylag gyorsan lefut. Végül, az algoritmus sajátossága, hogy a belső ciklusban lévő pillangó műveletek egyszerűen párhuzamosíthatók, ami lehetővé teszi az algoritmus párhuzamosított változatának implementációját hardverleíró (HDL) nyelveken is.



15. ábra. Egy egyszerű példa az algoritmushoz

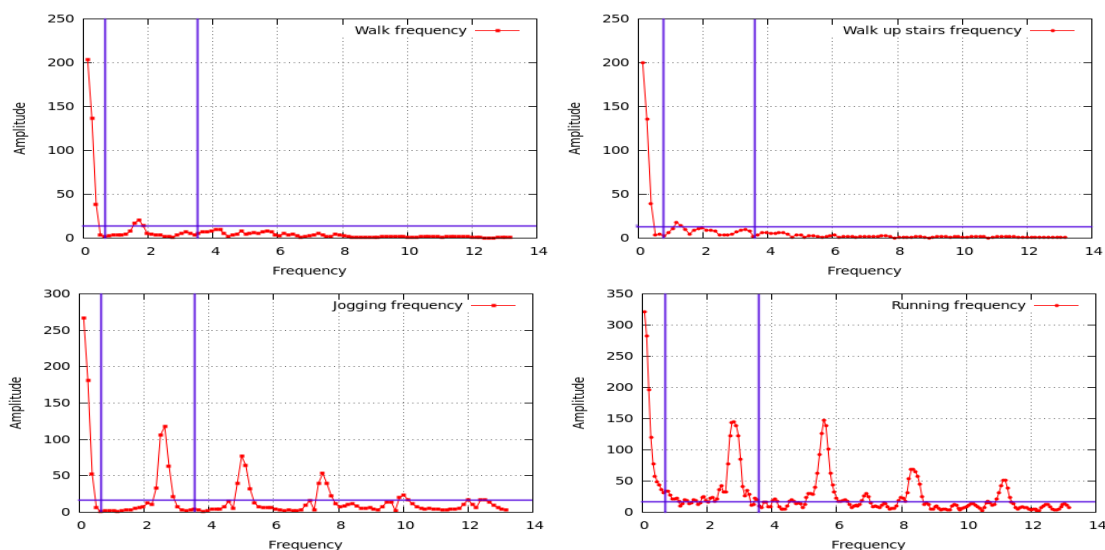
7.3. Frekvenciatérbeli kategorizálás és időtérbeli mintafelismerés

Miután a frekvencia komponensek előálltak, a mozgásdetektáló algoritmus megkeresi a legnagyobb komponens azon az intervallumon belül ahová a vizsgált mozgások frekvenciája eshet. A legnagyobb frekvencia komponens jelöli ki, hogy melyik frekvencia kategóriába tartozik a begyűjtött jel. A kategóriának megfelelő mintafelismerő algoritmusok fognak végigfutni a jelen, hogy az algoritmus eldönthesse, hogy a kategórián belüli mozgások közül melyikhez tartozik a vizsgált jel. A kategorizálásra valóban azért van szükség, hogy csökkentsük a műveletek számát. Hiszen, ha ezt a lépést kihagynánk, akkor az összes cselekvéshez tartozó mintát le kellene futtatni a jelen. Sajnos, tapasztalataink szerint néhány kategóri között kismértékben átfedés van. Abban az esetben, ha a legnagyobb frekvencia komponens két tartomány közötti területre esik, akkor a két kategóriában előforduló összes mintát keresni kell benne. Az 1. táblázatban található az általunk behatárolt intervallumok és azok frekvenciatartományai.

1. táblázat. Frekvencia kategóriák

Categories	Min frequency	Max frequency
1.	0.0 Hz	0.8 Hz
2.	0.81 Hz	2.25 Hz
2. - 3.	2.26 Hz	2.5 Hz
3.	2.51 Hz	2.65 Hz
3. - 4.	2.66 Hz	2.8 Hz
4.	2.81 Hz	3.8 Hz

Valójában, mivel a számítógépen az adatok diszkrét formában tárolódnak, így az intervallumok a frekvenciakomponenseket tartalmazó vektor indexein alapulnak. A 12. ábrán a frekvencia kategorizálás komplikát feladatnak tűnik, azonban ez lényegében egyszerű összehasonlításokból áll. A frekvencia kategóriák egy jól definiált intervallumot jelölnek ki 0.8 Hz - 3.8 Hz között. Ebben az intervallumban keresi az algoritmus a legnagyobb komponens. Az algoritmusban egy amplitúdó limit is be van állítva, amitől a legnagyobb komponensnek nagyobbnak kell lennie. A tapasztalatom szerint az amplitúdó korlátnak 14.5 és 16.5 között kell lennie. Amennyiben a legnagyobb komponens kisebb mint az amplitúdó korlát, akkor a megfigyelt személy valamilyen statikus mozgást végez. A 16. ábrán látható néhány fontosabb cselekvés spektruma. A képen a kék vonalak jelölik a kategóriák által meghatározott frekvencia intervallumot és az amplitúdó korlátot.



16. ábra. A spektruma néhány fontosabb mozgásnak

A mintafelismerés előtt a jelet megsűrjűk egy 7 mintás medián szűrő segítségével, hogy az

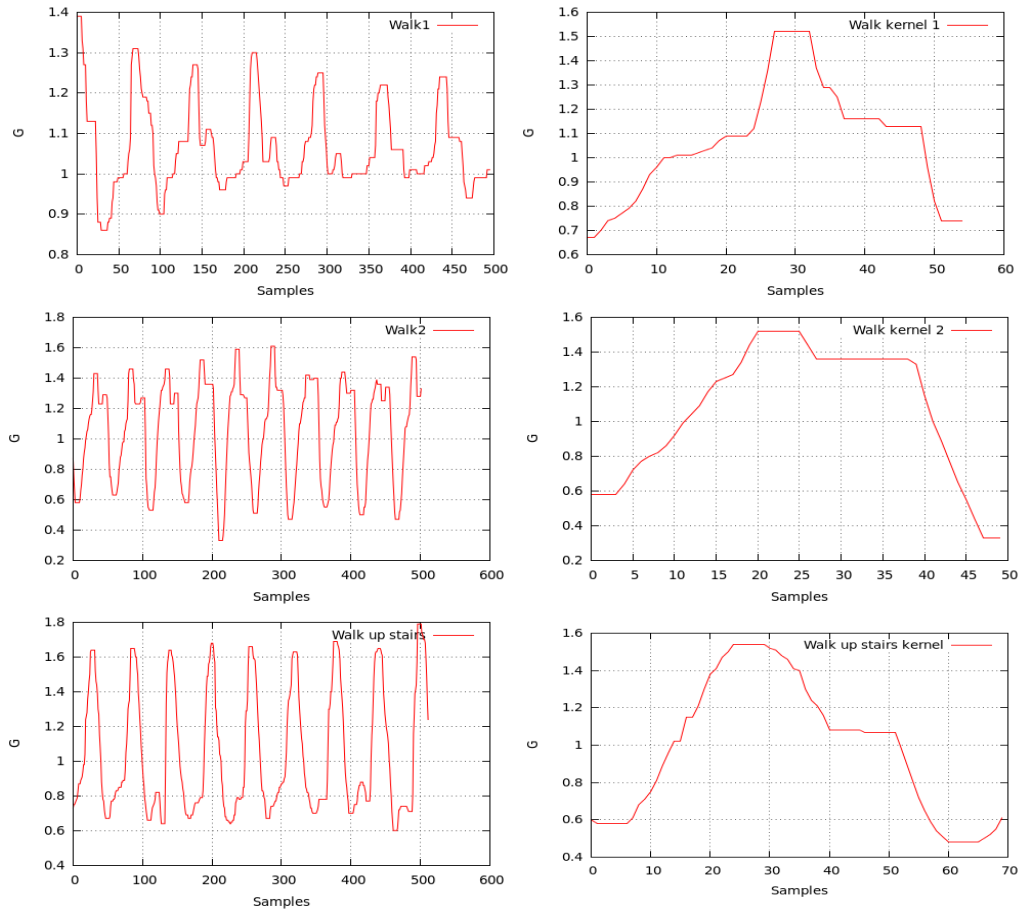
időbeli jelben lévő zajok mértékét csökkentjük [6, 7]. A szakirodalmakban, az egyik legelterjedtebb mintafelismerő technika a korreláció [23].

$$(f \star g)[n] := \sum_{m=0}^{N-1} f[m]g[m+n] \quad (13)$$

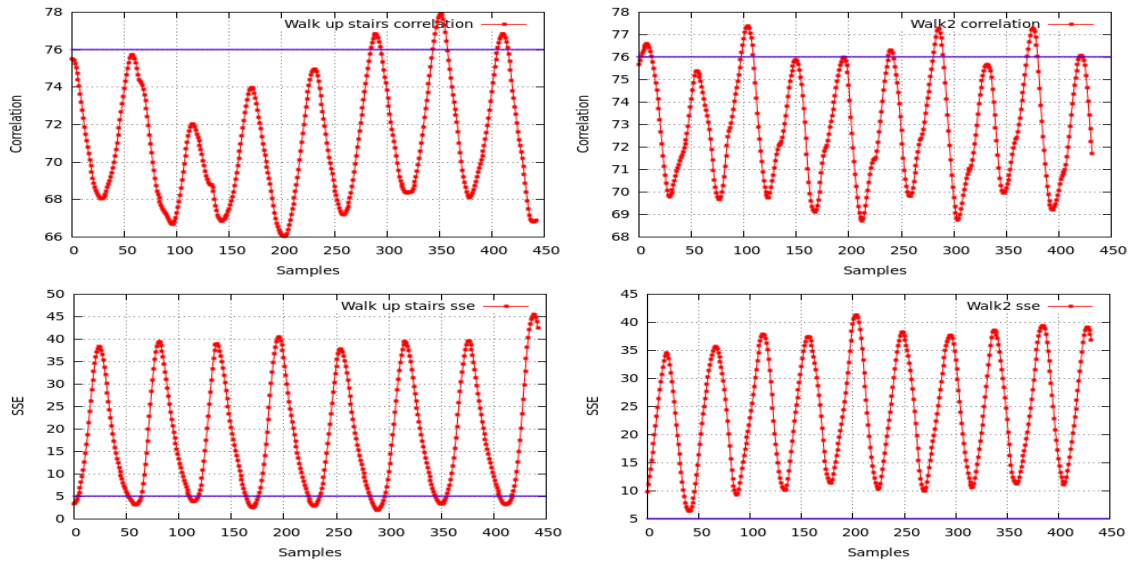
ahol f és g valós vektorok. Sajnos, néhány esetben a korreláció nem tudja élesen elhatárolni a mintát a jeltől, tehát a korrelációra alapozott mintafelismerő algoritmusok eredménytelenek lesznek. Ebből adódóan, megpróbáltam keresni más lehetőséget a mintafelismerésre. Az általam létrehozott mintafelismerő algoritmus a korreláción és a négyzetes hibán alapul. Hasonlóan a korrelációhoz, egy "ideális" kernelt csúsztatunk végig az időbeli jelen és vesszük a jel és a kernel közötti eltérések összegét. Ezt a módszert SSE-vel jelöltem.

$$SSE(f, g)[n] := \sum_{m=0}^{N-1} (f[m] - g[m+n])^2 \quad (14)$$

A korrelált jelben a csúcsok minél magasabbak annál nagyobb a hasonlóság a vizsgált jel és a kernel között. Ezzel szemben, az SSE-nél az kis értékek jelzik a hasonlóságot, mert ekkor az eltérés a jel és a kernel között kicsi. A 17. ábrán látható néhány mozgásmintához a felhasznált kernel. A képen két különböző sétából származó jelet láthatunk. Szemmel láthatóan a jelformák nagymértékben eltérnek a két mintában. Ez azzal magyarázható, hogy a két sétálás között nagymértékű dinamikai eltérés van. A sétálás rendelkezik a legszélesebb frekvenciatartománnyal (első kategória a 1. táblázatban), ezért a sétáláshoz legalább két kernelt kell használni. Ezzel szemben más cselekvések esetén elegendő egyetlen kernel használata. A legegyszerűbben egy példán keresztül lehet szemléltetni az eltérést a korreláció és az SSE nyújtotta eredmények között. A példában két hasonló cselekvést hasonlítottunk össze: a sétálást és a lécsőn felmenést (a 17. ábrán a második és a harmadik mintát).



17. ábra. Néhány főbb mozgást leíró mintasorozat és a felhasznált kernelek



18. ábra. A két mintafelismerő technika összehasonlítása

Mintkét mintához a lépcsőn felmenés kernelét használtuk és összehasonlítottuk az eredményeket. A 18. ábra mutatja az összehasonlítás eredményét, ahol a kék vonalak jelölik a képzeletbeli határokat a mintafelismeréshez. Ideális esetben, ha a keresett minta nem jelenik meg a vizsgált jelben, akkor minden egyes pontnak kisebbnek vagy nagyobbak kell lennie egy jól definiált határértéknél (kék vonal), az alkalmazott technikának megfelelően. Ahogy a példa is mutatja, a korreláció nem képes elkülöníteni a két cselekvést egymástól a felhasznált kernellel. Ha a 76-ot (kék vonal) választanánk határértékként, akkor hasonló számú pont lenne a korlát felett mindkét cselekvés esetén. Ezzel szemben az SSE szemmel láthatóan elkülönítette az cselekvéseket. A 18. képen meglehetősen sok pont esik 5 alá a lépcsőn felmenésnél, míg a sétálásnál mindegyik pont jóval 5 felett marad.

Az SSE-re alapozva fel tudunk írni egy egyszerű, mégis hatékony mintafelismerő algoritmust. Az algoritmusnak paraméterként az időben változó jelet, a kernelt, a határértéket és a határérték alatt lévő pontok számát kell átadni. A 3. pseudokód ismerteti az algoritmust.

Algorithm 3 pattern recognition algorithm

Require: $g[N], f[M], LIMIT, POINTS$

Ensure: *decision*

$size \leftarrow N - M + 1$

$correct\ points \leftarrow 0$

$SSE\ point \leftarrow 0$

for $i : 0$ **to** $size - 1$ **do**

for $j : 0$ **to** $M - 1$ **do**

$SSE\ point \leftarrow SSE\ point + (f[j] - g[i + j])^2$

end for

if $SSE\ point \leq LIMIT$ **then**

$correct\ points \leftarrow correct\ points + 1$

end if

$SSE\ point \leftarrow 0$

end for

if $correct\ points > POINTS$ **then**

$decision \leftarrow \text{true}$

else

$decision \leftarrow \text{false}$

end if

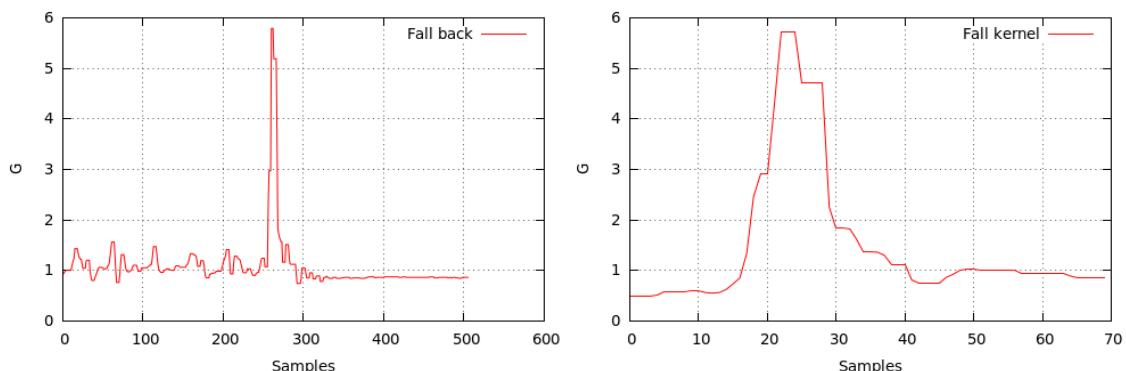
Az algoritmus az SSE alapján $N - M + 1$ új pontot fog legenerálni. Amennyiben, a határ alatt lévő (vagy a határértékkal megegyező) pontok száma meghaladja a paraméterben megadott limitet, akkor az algoritmus *igaz* értékkel tér vissza, ellenkező esetben pedig *hamissal*. Az algoritmusban figyelemmel kell lenni az indexelésekre. A (14). képletben az indexelés $N - 1$ -ig terjed, viszont az algoritmusban a kernel méretére kell alapozni ($f[M]$ jelöli a kernelt), azaz $M - 1$ -re mivel tudjuk, hogy a legnagyobb index felett már nincsenek elemek, tehát ezek nullának tekinthetők. Annak érdekében, hogy az algoritmus megbízhatóbb döntést adjon, érdemes az aktuális mintasorozatot megelőző sorozat utolsó M elemét letárolni és az aktuális mintasorozat elejéhez hozzáfűzni. Így, a mintafelismerő algoritmusnak már a kibővített jelsorozatot kapja meg és ezáltal, egy megbízhatóan eredményt fog adni.

A 12. ábrán látható egy *unknown activity* állapot. Ez azt jelenti, hogy amennyiben egy értelmezhetetlen mintasorozat lesz a bemenete a mozgásfelismerő algoritmusnak, az algoritmus nem fog döntést adni a cselekvésről. Ezt az kitélt azért implementáltuk az algoritmusba, mivel véleményünk szerint, sokkal biztonságosabb az, ha az algoritmus nem mond semmit a vizsgált cselekvésről, ahelyett hogy téves következtetést adjon.

7.4. Esésdetektálás

Az esésdetektálás az előző fejezetben bemutatott mozgásfelismerő technikán alapul. Az esésdetektálásra is már létezik több megoldás, amelyekben a kutatók kissé eltérő módszerekkel próbálják meg jelezni a különböző típusú esések bekövetkeztét. Az ide tartozó cikkek többségében itt is a test több pontján elhelyezett gyorsulásszenzorok adatai alapján történnek a vizsgálatok [7, 8, 27]. Egyes cikkekben az esésdetektálásnál a gyorsulásszenzor adatai alapján a függőleges pozíciótól számított szögeltérést vizsgálják. Ebben az esetben fontos szerepet játszik, hogy a gyorsulásszenzor hová van rögzítve a páciens testén. Mivel az esés az egész testre hatással van, így célszerű a gyorsulásszenzort a test egy viszonylag fix pontjára rögzíteni. Ezért, általában a szenzort a páciens csípőjéhez rögzítik.

Mi a már bemutatott mintafelismerő technikát alkalmattuk most is. Az esésdetektálás során több különböző esést is megvizsgáltunk és azt tapasztaltuk, hogy az eséseknek is egy jellegzetes mintájú gyorsulásváltozást írnak le. Alap állapotban a gyorsulás vektor nagysága megközelítőleg $1g$. Az esések során a gyorsulás változásában egy hirtelen felugró, majd egy gyorsan lecsengő minta jelenik meg. Eséskor a gyorsulás csökken a földet érésig, majd ezt követően hirtelen megnő és végül beáll egy „nyugalmi” pozícióba. A nyugalmi pozíció itt megint $1g$ -t jelent, ez addig tart, amíg a megfigyelt személy a földön fekszik. A 19. ábra egy mintát mutat a hátraesésről és az esésekhez használt kernelt.



19. ábra. Hátraeséskor rögzített jel és az esésekhez használt kernel

7.5. Kézzesztusok felismerése

Az életjel monitorozó rendszer részeként szeretnénk kidolgozni egy kézzesztusokat felismerő alrendszert is. A mozgásdetektálás mellett, a gyorsulásszenzorral felszerelt hordozható kliensek a kézzesztusok felismerésére is kiváló lehetőséget biztosítanak. A klienst a vizsgált személy csuklójára erősítve, a kar mozgását vizsgálhatjuk. Abban az esetben, ha fel tudjuk ismerni a különböző kézzesztusokat, akkor következtetni tudunk a vizsgált személy jellegzetes szokásaira és esetlegesen rejtett információk is feltárhatóak. Erre jó példa lehet a kéz remegésének megfigyelése, amely a begyűjtött adat frekvenciakomponensei alapján meghatározható. A kézzesztusok felismerése nemcsak megfigyelési célokra alkalmazható, hanem esetlegesen elemi vezérlésre is. Ha egy intelligens környezetet feltételezünk, akkor az abban lévő elektronikai eszközök is vezérelhetők kézmozdulatok segítségével.

Mivel a kézmozdulatok már összetettebb események a mozgásoknál és nem azonosíthatók egyértelműen minták segítségével, ezért a kézzesztusok felismeréséhez mesterséges neurális hálózatot alkalmaztunk. Az első tesztekben a neurális háló bemenő paraméterei a gyorsulásszenzor x , y és z komponense volt [11, 28]. Tekintettel arra, hogy a neurális hálózatot egy mini számítógépen futtatjuk, ezért arra törekedtünk, hogy viszonylag egyszerű legyen a hálózat szerkezete. Első lépésként magasszintű programozási nyelven (Qt-C++), osztályokba szervezve létrehoztunk egy tesztkörnyezetet, ahol tetszőleges architektúrájú előrecsatolt neurális hálózat alakítható ki [29].

Először négy kézmozdulathoz tartozó adatokat gyűjtöttünk be a gyorsulásszenzortól. A nagymennyiségű mintahalmaz elengedhetetlen a neurális hálózat helyes betanításához. A teszthez egy kétrétegű és egy háromrétegű neurális hálózatot hoztunk létre, amelyek hasonlóan 3 neuront tartalmaznak a bemeneti rétegen, 12 neuront a köztes rétegen és 4 neuront a kimeneti rétegen. A 4 kimenő neuron értéke alapján lehet eldönteni, hogy a bemenetre érkező adatok melyik kézmozdulathoz tartoznak.

A többi rétegű neurális hálózatoknál egy réteg kimenete, a rákövetkező réteg bemenő paramétere lesz. Ezt a következő képlettel írhatjuk le,

$$\mathbf{o}^{k+1} = f^{k+1}(\mathbf{W}^{k+1} \mathbf{o}^k + \mathbf{b}^{k+1}) \quad k = 0, 1, \dots, M-1 \quad (15)$$

ahol M jelöli a szintek számát, k az aktuális szintet, f az aktivációs függvényt, $\mathbf{W}$ a súlymátrixot, $\mathbf{b}$ a korrekciót (bias) és $\mathbf{o}$ a k . szint kimeneti értékeinek a vektorát. A hálózat első rétegének kimeneti értékei megegyeznek az összegyűjtött mintahármasokat tartalmazó vektorral.

$$\mathbf{o}^0 = \mathbf{p} \quad (16)$$

A hálózat utolsó rétegének kimenete fogja adni a végleges eredményt. A háló hatékonyságának a mérése a *várható négyzetes hibának* a közelítésén alapul.

$$F(\mathbf{x}) = (\mathbf{t}(m) - \mathbf{o}(m))^T (\mathbf{t}(m) - \mathbf{o}(m)) \quad (17)$$

ahol az $\mathbf{x}$ a szintekhez tartozó súlyokat és korrekciókat jelöli és $\mathbf{t}$ az ideális kimenetet. A neurális hálózat tanításához a momentummal bővített *backpropagation* tanító algoritmust használtuk. Ennek alapján a súly és korrekció paraméterek meghatározása az $m+1$. iterációban a következőképpen adható meg,

$$w_{i,j}^k(m+1) = \gamma w_{i,j}^k(m) - (1-\gamma) \alpha \frac{\partial F}{\partial w_{i,j}^k} \quad (18)$$

$$b_i^k(m+1) = \gamma b_i^k(m) - (1-\gamma) \alpha \frac{\partial F}{\partial b_i^k} \quad (19)$$

ahol α a tanulási arány és γ a momentum. Továbbá,

$$\frac{\partial F}{\partial w_{i,j}^k} = \frac{\partial F}{\partial n_i^k} \frac{\partial n_i^k}{\partial w_{i,j}^k} \quad (20)$$

$$\frac{\partial F}{\partial b_i^k} = \frac{\partial F}{\partial n_i^k} \frac{\partial n_i^k}{\partial b_i^k} \quad (21)$$

az iménti két egyenletben az n_i^k egy neuron összegzett bemenetét szimbolizálja.

$$n_i^k = \sum_{j=1}^{S^{k-1}} w_{i,j}^k o_j^{k-1} + b_i^k \quad (22)$$

Itt S egy adott szint neuronjainak a számát jelöli. A parciális deriváltakat egyszerűsítve,

$$\frac{\partial n_i^k}{\partial w_{i,j}^k} = o_j^{k-1} \quad (23)$$

$$\frac{\partial n_i^k}{\partial b_i^k} = 1 \quad (24)$$

$$s_i^k \equiv \frac{\partial F}{\partial n_i^k} \quad (25)$$

az egyszerűsített képleteket felhasználva,

$$\frac{\partial F}{\partial w_{i,j}^k} = s_i^k o_j^{k-1} \quad (26)$$

$$\frac{\partial F}{\partial b_i^k} = s_i^k \quad (27)$$

A fenti képletben s az F függvény érzékenységeire utal. Az iménti képleteket felhasználva megadható az egymásra épülő súly és korrekció paraméterek kiszámítása a $k+1$. lépésben.

$$w_{i,j}^k(m+1) = \gamma w_{i,j}^k(m) - (1-\gamma) \alpha s_i^k a_j^{k-1} \quad (28)$$

$$b_i^k(m+1) = \gamma b_i^k(m) - (1-\gamma) \alpha s_i^k \quad (29)$$

ami felírható mátrix alakban is,

$$\Delta \mathbf{W}^k(m+1) = \gamma \Delta \mathbf{W}^k(m) - (1-\gamma) \alpha \mathbf{s}^k(m) (\mathbf{o}^{k-1})^T \quad (30)$$

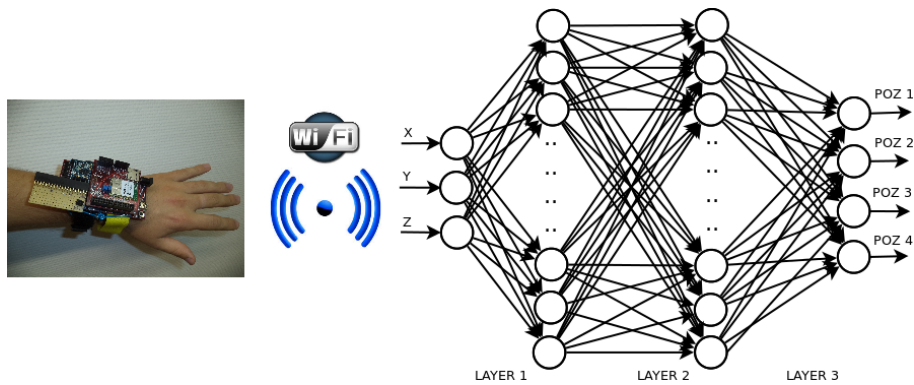
$$\Delta \mathbf{b}^k(m+1) = \gamma \Delta \mathbf{b}^k(m) - (1-\gamma) \alpha \mathbf{s}^k(m) \quad (31)$$

ahol,

$$\mathbf{s}^k \equiv \frac{\partial F}{\partial \mathbf{n}^k} = \begin{bmatrix} \frac{\partial F}{\partial n_1^k} \\ \frac{\partial F}{\partial n_2^k} \\ \vdots \\ \frac{\partial F}{\partial n_{S^k}^k} \end{bmatrix} \quad (32)$$

A fenti képletekben a félkövérrel jelölt szimbólumok vektorra és mátrixra utalnak. A neurális háló minden neuronjához egy paraméterezhető szigmoid aktivációs függvényt használtunk.

$$f(n_i) = \frac{1}{1 + e^{-\lambda n_i}} \quad (33)$$



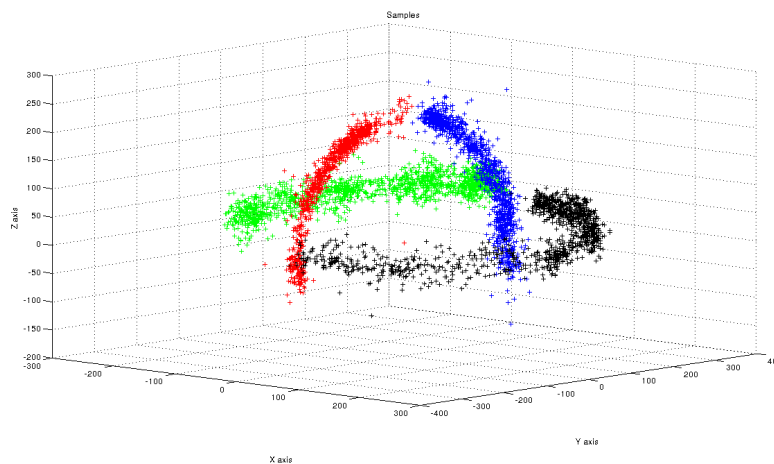
20. ábra. A tesztkörnyezet struktúrája

ahol n_i egy neuron összegzett bemenetét jelöli és a λ felhasználásával lehet az aktivációs függvény alakját szabályozni. A neurális háló tanítása előtt, a mintahalmaz elemeit össze kell keverni, annak érdekében, hogy a hálózat ne csak az utoljára megtanult gesztust ismerje fel. A 20. ábra az általunk elkészített tesztkörnyezetet illusztrálja.

Az összegyűjtött adatok a négy pozíciónak megfelelően 4 halmazra bontható. Mindegyik halmaz azokat az értékhármasokat tartalmazza, amelyek egy adott kézmozdulathoz tartoznak. Az x , y , z értékek -256 és $+256$ között vannak és a gyorsulásváltozást írják le. A minták eloszlását a 3D-s koordináta rendszerben a 21. ábra szemlélteti. Az ábrán a különböző színek a 4 halmazra utalnak, melyek megközelítőleg egyenként 1000 mintát foglalnak magukba.

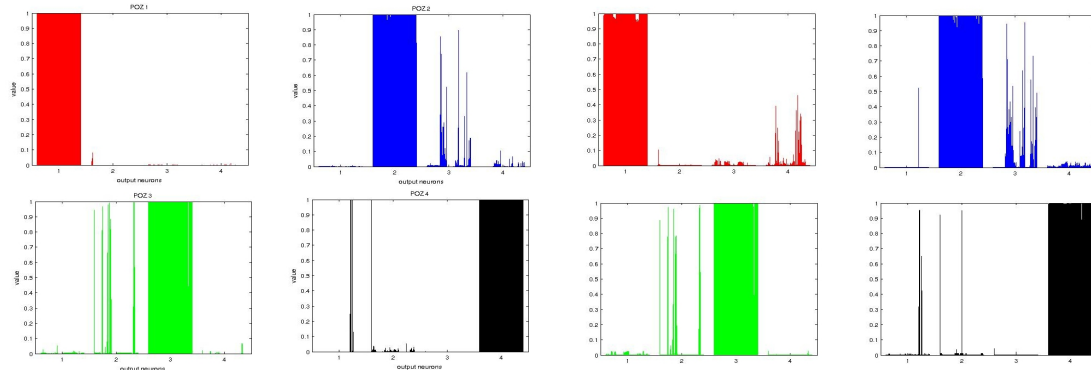
A neurális hálózat tanítása alatt nagyszámú mátrix műveletet kell elvégezni. A tanítás során figyelmet kell fordítanunk arra, hogy a nem használt memóriaterületeket (mátrixokat) felszabadítsuk, nehogy kifusson a program a rendelkezésre álló memóriából. A számításokban résztvevő mátrixok a számítógépen egy adott memóriaterületre hivatkoznak. Mivel a tanítási folyamat több százezer iterációt is tartalmazhat, így mindig tekintettel kell lenni a memóriakezelésre.

Miután elkészült a tesztkörnyezet, megvizsgáltuk, hogy a neurális hálózat milyen hatékonysággal képes felismerni a különböző kézmozdulatokat. A vizsgálatok alapján megállapítható, hogy a kétrétegű neurális hálózat erre a problémára hasonló eredményt szolgáltat, mint a háromrétegű, így az egyszerűsége törekedve a kétrétegű hálózatot alkalmaztuk a továbbiakban. Az általunk készített program helyességének vizsgálatához a Matlab Neural Network Toolbox-ot használtuk, amely napjaink egyik leghatékonyabb neurális hálózat készítő szoftvere. A programunkban létrehozott hálózat mintájára elkészítettünk egyet a Matlab-ban is, amely szintén log-sigmoid aktivációs függvényt használ a köztes rétegeken. A tanítás során a hálózat minden x , y , z bemenő paraméteréhez megadtuk a célkimenet.



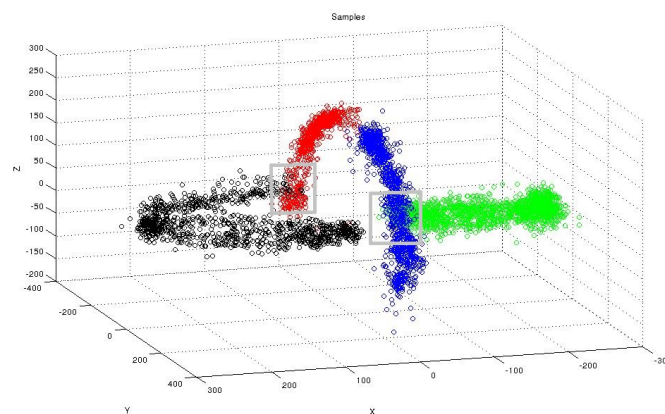
21. ábra. A minták eloszlása 3D-s koordináta rendszerben

Ideális esetben, ha az x , y , z értékhármass valamely vizsgált kézmozdulathoz tartozik, akkor a hálózat kimenetén a kézmozdulathoz tartozó neuronnak egyes értékűnek kell lennie, míg a többi nulla. Ennek alapján, minél nagyobb a kimeneti neuron értéke, annál nagyobb a valószínűsége, hogy a bemenetre érkező értékhármass az általa jelölt kézmozdulathoz tartozik. Az általunk készített program eredménye a kézgesztusfelismerési problémára a 22. ábra baloldali részén, míg a Matlab eredménye az ábra jobboldali részén látható.



22. ábra. Az általunk készített program és a Matlab eredménye a kézgesztus felismerés problémára

A színek a 21. ábrának megfelelően azonosítanak egy kézmozdulatot. A grafikonok x tengelye a neurális hálózat 4 kimeneti neuronját jelöli, míg az y tengely a neuronokhoz tartozó értéket a teszt alatt. Mind a négy tesztet összeváltottunk minden kézmozdulathoz párszáz mintát, amelyet a tesztelés során a hálózat bemenetére helyeztünk. Amint az ábrák is megfigyelhető, a kimeneti értékek mindig 0 és 1 közé esnek. A két ábra alapján az első következtetés, hogy a programunk képes legalább olyan jó eredményt nyújtani, mint a Matlab. Ideális esetben egy kézmozdulat felismerésének tesztelésekor csak egyetlen kimeneti neuronnak kellene 1-es értéket adni, viszont a teszteredményeken egyértelműen látható, hogy a cél neuron mellett más neuronok is szolgáltatnak kisebb-nagyobb értékeket. Az eredmény kismértékben javítható a köztes rétegek log-sigmoid aktivációs függvény meredekségének növelésével, amit a λ paraméter beállításával adhatunk meg, úgy hogy λ nagyobb legyen, mint 1 (33). Ezt a módszert a programunkban már alkalmaztuk, viszont nem oldja meg a problémát. Az eredmények alapos áttekintése után arra a következtetésre jutottunk, hogy a teszteléskor jelentkező hibákat a mintahalmazok közötti átfedés okozza. A 23. ábrán a szürke téglalapok jelölik a mintahalmazok átfedő részeit. Ha összehasonlítjuk a 22. és 23. ábrákat, akkor meggyőződhetünk arról, hogy a neurális hálózat valóban azokat a kézmozdulatokat téveszti el, amiknek az értékei kismértékben átfednek. A mintahalmazok közötti átfedésekből eredő hibák egy lehetséges megoldása a mintahalmazokon belül néhány minta összevonásával egy újabb mintahalmaz kialakítása.



23. ábra. A mintahalmazok átfedő részei

8. Összegzés

Az eddig elért eredményeink alapján azt mondhatjuk, hogy lefektettük az alapjait egy általános célú páciens megfigyelő rendszernek. Létrehoztunk egy Java-n alapuló szerverprogramot, amely a hozzá beérkező csomagok feldolgozásáért, a csomagokban lévő hasznos információ letárolásáért és az adatelemzés segítéséért felelős. Összeállítottunk több mikrokontroller és mini számítógép alapú asztali és vezeték nélküli klienst, amelyek a szenzorok vezérlését, a begyűjtött adatok továbbítását és bizonyos esetekben az adatkiértékelést végzik. Működik a szerver és a kliensek közötti kapcsolat és a kliensek távoli vezérlése. Az adatbázis sémája is elkészült és a szerverről tetszőlegesen lehet frissíteni és módosítani az táblázatok bejegyzéseit. A fejlesztés előrehaladtával, ha új szenzorokkal bővül a rendszer, akkor az adatbázis is bővülni fog.

A tevékenységdetektálás és a kézgesztus felismerés témakörét is hozzákapcsoltuk a monitorozó rendszerünkhöz, ami növelte a rendszer robusztusságát. Az eddig elért eredményeinkből és a szakirodalmak alapján azt a következtetést vontuk le, hogy a megfigyelt személy cselekvései és a váratlan helyzetek észlelése megoldható a gyorsulásszenzor adatainak az elemzése által.

A dolgozatban bemutatott mozgásfelismerő technika (12. ábra) jól alkalmazható a tevékenységek és az esések beazonosítására. Abban az esetben, ha egy cselekvés egy sajátos és periodikus gyorsulásváltozást ír le, akkor a bemutatott módszer képes beazonosítani a cselekvést. Amennyiben tudjuk egy cselekvés frekvencia intervallumát és ismerjük az időtérbeli mintáját, akkor könnyedén bővíthetjük az algoritmust az új cselekvéssel. Annak a lehetősége, hogy képesek vagyunk beazonosítani a különböző mozgási mintákat, sok információt szolgáltat az orvosoknak a megfigyelt személyről. A mozgás analízis segítséget nyújt a testtartási és mozgási rendellenességek okainak feltárásában [20]. A dolgozat elején már említve volt, hogy az emberi mozgás kutatása szoros kapcsolatban áll más betegségek vizsgálatával, úgy mint: mozgáskorlátozást kiváltó betegségek (agyvérzés, elhízás, stb.), szklerózis és Parkinson kór [2, 4, 20]. Összességében azt állapíthatjuk meg, hogy a mozgás analízis rengeteg hasznos információt szolgáltat a megfigyelt személy egészségi állapotáról.

Napjainkban a jövő internetének (Future Internet) kidolgozása egy rohamosan fejlődő kutatási terület. Ennek az egyik legfontosabb része a tárgyak internetének megvalósítása (Internet of Things - IoT). Az IoT egy hálózat, amelyben különböző objektumok (emberek, szenzorok, eszközök, stb.) állnak kapcsolatban egymással az Interneten keresztül [31, 33, 34]. A dolgozatban bemutatott megfigyelő rendszer egy remek példa az IoT céljainak tükrözésére. Véleményem szerint az IoT és ehhez kapcsolódóan az életvitelt segítő rendszerek fogják a legkiemelkedőbb szerepet betölteni a kutatási életben a következő tíz évben.

Mivel az általunk elképzelt rendszer meglehetősen szerteágazó, ezért a rendszer fejlesztése egyetlen ember számára hatalmas feladat. Viszont, ha más hallgatók is bekapcsolódnak tudományos munka vagy szakdolgozatként, akkor a komplex feladat részekre darabolható és párhuzamosan végezhető. Ha mindenki megoldja a saját feladatát, akkor a rendszer fejlesztése gyors ütemben tud előrehaladni. Ez egy komplex rendszer, melynek fejlesztése során több tudományterületet is érintünk és ebből adódóan sok esetben kell új ismereteket elsajátítanunk. A rendszer kidolgozásának még csak az első felében járunk és annak érdekében, hogy az általunk elképzelt ötlet megvalósuljon, rengeteg időt és energiát kell még befektetnünk.

Köszönetnyilvánítás

A kutatás a TÁMOP 4.2.4.A/2-11-1-2012-0001 Nemzeti Kiválóság Program című kiemelt projekt keretében zajlott. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Hivatkozások

- [1] Smith, T.F., Waterman, M.S.: Elderly activities recognition and classification for applications in assisted living, *Expert Systems with Applications*, vol. 40, pp. 1662–1674 (2013)
- [2] Godfrey, A., Bourke, K.A., Ólaighin, M.G., Ven de van, P., Nelson, J.: Activity classification using a single chest mounted tri-axial accelerometer, *Medical Engineering & Physics*, vol. 33, pp. 1127–1135 (2011)
- [3] Lyons, M.G., Culhane, M.K., Hilton, D., Grace, A.P., Lyons, D.: A description of an accelerometer-based mobility monitoring technique, *Medical Engineering & Physics*, vol. 27, pp. 497–504 (2005)
- [4] Kavanagh, J.J., Menz, B.H.: Accelerometry: A technique for quantifying movement patterns during walking, *Gait & Posture*, vol. 28, pp. 1–15 (2008)
- [5] Forester, F., Smeja, M., Fahrenberg, J.: Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring, *Computers in Human Behavior*, vol. 15, pp. 571–583 (1999)
- [6] Lugade, V., Fortune, E., Morrow, M., Kaufman, K.: Validity of using tri-axial accelerometers to measure human movement—Part I: Posture and movement detection, *Medical Engineering & Physics*, vol. 36, pp. 169–176 (2014)
- [7] Kangas, M., Konttila, A., Lindgren, P., Winblad, I., Jamsa, T.: Comparison of low-complexity fall detection algorithms for body attached accelerometers, *Gait & Posture*, vol. 28, pp. 285–291 (2008)
- [8] Chern-Sheng Lin, Hung Chun Hsu, Yun-Long Lay, Chuang-Chien Chiu, Chi-Shih Chao: Wearable device for real-time monitoring of human falls, *Measurement*, vol. 40, pp. 831–840 (2007)
- [9] Bourke, K.A., O’Brien, V.J., Lyons, M.G.: Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm, *Gait & Posture*, vol. 26, pp. 194–199 (2007)
- [10] Ge Wu: Distinguishing fall activities from normal activities by velocity characteristics, *Journal of Biomechanics*, vol. 33, pp. 1497–1500 (2000)
- [11] Suto, J., Oniga, S.: Testing artificial neural network for hand gesture recognition, *Creative Mathematics and Informatics*, vol. 22, pp. 223–228 (2013)
- [12] Petrényi, J.: Windows szerver TCP/IP alapok. Microsoft Hungary, Magyarország (2010).
- [13] Tanenbaum, A.S.: Computer networks. Fourth Edition. Prentice Hall PTR Upper Saddle River, New Jersey (2003)
- [14] Buttlar, D., Farrell, J., Nichols, B.: PThreads Programming. O’Reilly Media, USA (1996)
- [15] Suto, J., Oniga, S., Hegyesi Gy.: A simple fast Fourier transformation to microcontrollers and mini computers, In: 18th International Conference on Intelligent Engineering Systems, folyamatban.

- [16] Thomas, G.B, Weir, D.M., Hass, J., Giordano, R.F.: Thomas's Calculus, 11th Edition, Addison-Wesley, USA: (2005)
- [17] Lo, C.P., Lee, Y.Y.: Real-time implementation of the split-radix FFT — An algorithm to efficiently construct local butterfly modules, *Signal processing*, vol. 71, pp. 291–299 (1998)
- [18] Lo, C.P., Lee, Y.Y.: Real-time implementation of the moving FFT algorithm, *Signal processing*, vol. 79, pp. 251–259 (1999)
- [19] Yen, F.W, You, D.S., Chang, C.Y.: Real-time FFT with pre-calculation, *Computers and Electrical Engineering*, vol. 35, pp. 435–440 (2009)
- [20] Godfrey, A., Conway, R., Meagher, D., Ólaighin, G.: Direct measurement of human movement by accelerometry, *Medical Engineering & Physics*, vol. 30, pp. 1364–1386 (2008)
- [21] Smith, W.S.: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publisher, USA (1999)
- [22] Lyons, G.L.: *Understanding Digital Signal Processing*. Prentice Hall PTR, USA (2001)
- [23] Kammler, D.W.: *A First Course in Fourier Analysis*. Cambridge University Press, UK (2008)
- [24] Mertins, A.: *Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications*. John Wiley & Sons, England (1999)
- [25] Bracewell, N.R.: *The Fourier Transform and Its Applications*, 3rd ed., McGraw-Hill Companies, USA (2000)
- [26] Loan, V.C.: *Computational Frameworks for the Fast Fourier Transform*, Siam, USA (1992)
- [27] Jung, J.Y., Lee, J.W.: ZigBee device access control and reliable data transmission in ZigBee based health monitoring system. In: *Advanced Communication Technology (ICACT)*, pp. 795–797. IEEE Press, South Korea (2008)
- [28] Oniga, S., Orha, I.: Hardware implemented neural networks used for hand gesture recognition. *Carpathian Journal of Electronic and Computer Engineering*. vol. 1, pp. 93–96 (2011)
- [29] Suto, J., Oniga, S.: A new C++ implemented feed forward neural network simulator. *Carpathian Journal of Electronic and Computer Engineering*. vol. 6, pp. 3–6 (2013)
- [30] Hagan, T.M., Demuth, B.H., Beale, M.: *Neural Network Design*, PWS Publishing Company, USA (1996)
- [31] Lung, C., Oniga, S., Buchman, A., Tisan, A.: Wireless data acquisition system for IoT applications, *Carpathian Journal of Electronic and Computer Engineering*, vol. 6, pp. 64–67 (2013)
- [32] Suto, J., Oniga, S., Orha, I.: Microcontroller based health monitoring system. In: *IEEE 19th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pp. 227–230. IEEE Press, Romania (2013)
- [33] Terdik, Gy., Gal, Z.: Advances and practice in Internet of Things: A case study. In: *IEEE 4th International Conference on Cognitive Infocommunications*, pp. 435–440. IEEE Press, Hungary (2013)
- [34] Berczes, T., Sztrik, J.: Tool supported analysis of queueing systems with Future Internet applications. In: *Abstract & Pre-proceedings of 9th International Conference on Applied Mathematics*, pp. 104–107. Romania (2013)
- [35] Cooking hacks. E-Health Sensor Platform V2.0 for Arduino and Raspberry Pi. <http://www.cooking-hacks.com>