EÖTVÖS LORÁND TUDOMÁNYEGYETEM
Informatikai Kar
Információs Rendszerek Tanszék

Környezeti intelligencia adatgyűjtés és adatelemzés

Diplomamunka

Szendrő Balázs
programtervező matematikus hallgató
nappali tagozat

témavezető:
dr. habil. Lőrincz András
tudományos főmunkatárs

Budapest, 2006

Tartalomjegyzék

1. fejezet

Bevezetés

Környezeti Intelligenciának (Ambient Intelligence, AmI) nevezik összefoglaló néven azokat az egymással kommunikáló hardver- és szoftvereszközöket, amelyek képesek a felhasználó szokásaihoz alkalmazkodni, ezáltal kényelmesebb, hatékonyabb ember-számítógép interakció jöhet létre. Adaptivitás a gépi tanulás módszereivel valósulhat meg. A fejlesztett (eleinte nem adaptív) beviteli eszközök használatakor adatokat kell gyűjtenünk majd elemeznünk, hogy megállapíthassuk hogyan lehetne később a gépi tanulás módszereit alkalmazni. A felhasználó segítéséhez tehát képesnek kell lennünk jellemezni, modellezni az ő interakcióját.

Az ELTE NIPG csoport egy része alternatív ember-számítógép interakciós eszközök fejlesztésével foglalkozik testi fogyatékos beszédértő, de nem beszélő gyermekek számítógéppel történő kommunikációjának segítésére. Az eszközök az SKMK[1] központban kerülnek tesztelésre. E kutatási program részeként felmerült az igény testre rögzíthető gesztikuláció mérő rendszer fejlesztésére. A rendszer elsődleges célja testi fogyatékkal rendelkező gyermekek számára a számítógéppel történő kommunikációban az egérfunkciók kiváltása és így a sikeresebb kommunikáció lehetőségének megteremtése. Az RF-egér - amely e rendszer első prototípusa - azonban jóval szélesebb körben is használható: Tetszőleges 2 szabadságfokú vezérlés megvalósítására alkalmas lehet.

A dolgozatban bemutatjuk az RF-egeret mint szoftvert, és elemezzük a használatakor kapott adatokat. Megvizsgáljuk milyen jellemzés adható a felhasználó viselkedéséről, illetve azt, hogy hatékonyabb lett-e az interakció. Az elemzésekhez Rejtett Markov Modelleket használunk.

1.1. A dolgozat felépítése

A dolgozatban először bemutatjuk az SKMK-ban jelenleg használt szoftvereket (2. fejezet). Ezek azok a szoftverek amelyek használatát egyáltalán vizsgálhatjuk. Ezután részletesen ismertetjük a diplomamunka egyik fő részét képező RF-egér programot (3. fejezet), majd rátérünk az elemzésekhez alkalmazott matematikai eszköztárra (4. fejezet) és részletezzük az elvégzett elemzéseket (5. fejezet).

[1]Segítő Kommunikáció-módszertani Központ, H-1112 Budapest XI., Neszmélyi út 36., http://www.bliss.org.hu/Segitkomm.htm

1.2. Köszönetnyilvánítás

A dolgozatot az ELTE Információs Rendszerek Tanszéken működő NIPG csoport segítségével írtam. Köszönet nekik: dr. habil. Lőrincz András, Hévízi György, Palotai Zsolt, Kiszlinger Melinda, Póczos Barnabás, Szabó Zoltán, Réthey-Prikkel Brigitta, Gerőfi Balázs. Köszönöm még dr. Loványi Istvánnak és a BME Irányítástechnika és Informatika Tanszékének amiért rendelkezésünkre bocsátották az RF-MEMS eszközöket, ezáltal lehetővé tették az RF-egér kifejlesztését.

2. fejezet

Az SKMK-ban használt interakciós eszközök

A hagyományos alternatív kommunikációs eszközök többnyire speciális billentyűzetek, kommunikációs táblák és elektronikus kommunikátorok amelyekre képeket, grafikus ábrákat, jelképeket, betűket, szavakat tartalmazó lapokat helyeznek.[1] Hátrányuk, hogy többnyire segítő személy jelenléte szükséges használatukhoz - ha nem elektronikus az eszköz, gyakran csak a jeleket ismerő tolmácson keresztül lehet társalogni; nem eléggé rugalmasak - cserélni kell a lapokat, illetve új szöveget kell mondani a kommunikátorba, ha más fogalmak kifejezését is lehetővé akarjuk tenni; és gyakran drágák.

Célunk olyan beviteli eszközök fejlesztése, amelyek önállóan használhatóak, kellően sokoldalúak, és nem igényelnek drága berendezéseket. Külön foglalkozunk tisztán beviteli (2.1. alfejezet), és kommunikációs (2.2. alfejezet) szoftverek készítésével.

2.1. Speciális beviteli eszközök

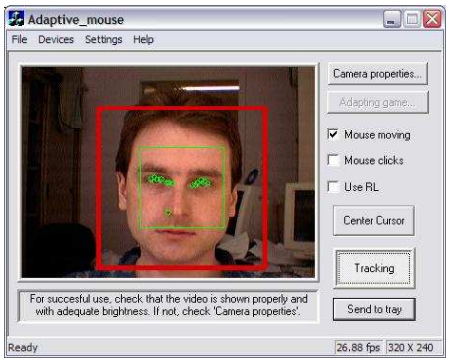
Itt néhány - a NIPG csoport tagjai által fejlesztett - olyan beviteli eszközt említünk meg, amelyek használatához nem szükséges speciális hardvert beszerezni. E szoftverek használatakor gyűjtött adatok alapján folynak a jelenlegi fejlesztések.

Fejegér.[2] A Fejegér a felhasználó fejének mozgása alapján egérkurzort vezérel. A fej követéséhez webkamerát használ. Tapasztalatok szerint gyakorlás után milliméteres pontossággal lehet vele irányítani az egérkurzort és gyakori használatával javul a fejmozgás koordinációja is.

Hangegér.[12] A Hangegér hang által vezérelt beviteli eszköz. Működési elve roppant egyszerű: A kurzort az egyik tengely mentén a hang erősségének, a másik tengely mentén a hang magasságának változtatásával mozgathatjuk. A program használható a hangképzés fejlesztésének eszközéül is siket felhasználók esetén. A Hangegér elemzését szintén Rejtett Markov Modellek alkalmazásával végezték [13].

[1] http://www.bliss.org.hu/Segitkomm.htm

[2] http://nipg.inf.elte.hu/headmouse/headmouse.html



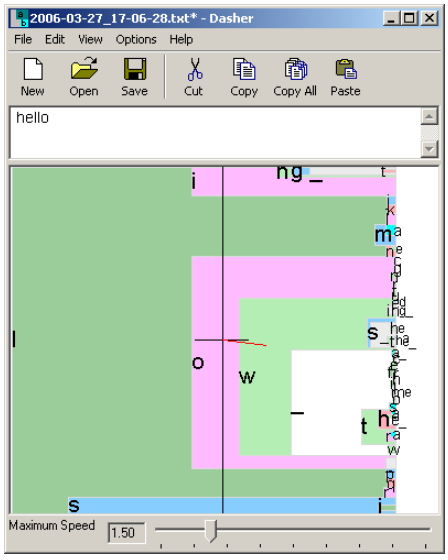
2.1. ábra. A Fejegér kezelőfelülete.

2.2. Kommunikációs alkalmazások

2.2.1. Dasher

A Dasher[3] egérrel működtethető szövegbeviteli program. Segítségével tudunk gépelni akkor is, ha nem áll rendelkezésünkre billentyűzet (tenyérgépet használunk), vagy ha nem tudjuk használni a billentyűzetet (mozgássérültek vagyunk).

A Dasher indításakor különböző színű téglalapokat látunk a képernyő jobb oldalán. Ezek mindegyike egy-egy betűt tartalmaz. Ha az egérrel rámutatunk, a Dasher nagyítja a téglalapot, és amint a téglalap bal széle áthalad a középső fonálkereszten, a betűt legépeltük. A téglalap jobb oldalán közben ismét megjelenik a teljes ábécé, amelyből kiválaszthatjuk a következő betűt, s. í. t. Ahogy haladunk a téglalapok jobbról-balra haladnak. Ha hibázunk, a kurzort a középső fonálkereszt bal oldalára visszük, így a Dasher balról-jobbra mozgatva kicsinyíteni fogja a téglalapokat, és visszatörli azokat a betűket, amelyeknek téglalapjaiból ily módon kijövünk.



2.2. ábra. A Dasher kezelőfelülete.

[3]http://www.inference.phy.cam.ac.uk/is/

A Dasher rendelkezik nyelvi modellel is. Ez azt jelenti, hogy azokat a téglalapokat rajzolja nagyobbra, amelyek nagyobb valószínűséggel vezetnek értelmes szavakhoz az adott nyelven. A többi téglalap kisebb lesz, ezáltal a felhasználó kevesebbet hibázik. A nyelvi modell hátterében magasabb rendű Markov modellek állnak, amelyeket a Dasher offline épít a nyelv szavaiból. A modellt használat közben is javítja a program, így a gyakrabban használt szavakat gyorsabban tudjuk gépelni.

A dolgozatban bemutatásra kerülő kísérletekhez 3.0.1-es verziószámú Dasher módosított változatát használtuk. Ebben a változatban kikapcsolható a nyelvi modell, így a program adaptivitása nélkül figyelhetjük a kísérleti alany gépelési sebességének fejlődését. Beépítettünk még egy visszalépési funkciót is későbbi elemzésekhez. A *visszalépés* azt jelenti, hogy visszatörlünk néhány karaktert, majd szoftveresen középre helyezzük az egérkurzort, hogy a Dasher ne nagyítson, amíg a felhasználó ismét el nem indul jobbra. Ezáltal gyorsabban lehet korrigálni, mint a hagyományos visszatörléssel.

Az olyan kurzoros eszközöknél ahol nehéz a hirtelen nagy és a lassú finom mozgások között váltani gyakori a *túlkorrigálás*. Ez a következőt jelenti: Ha gyorsan nagyítunk gépelés közben de rossz irányba megyünk, a Dasher sok rossz betűt fog gépelni. Ezért gyorsan próbálunk ellenkező irányba korrigálni, miáltal túl sok betűt törlünk vissza. Ilyenkor sokáig eltarthat, amíg újra a megfelelő ponton tudjuk folytatni a gépelést. Az RF-egér használatakor - főleg a gyakorlás kezdeti szakaszában - jellemző a túlkorrigálás. A kísérleti alany ennek megelőzésére használhatta a visszalépési funkciót.

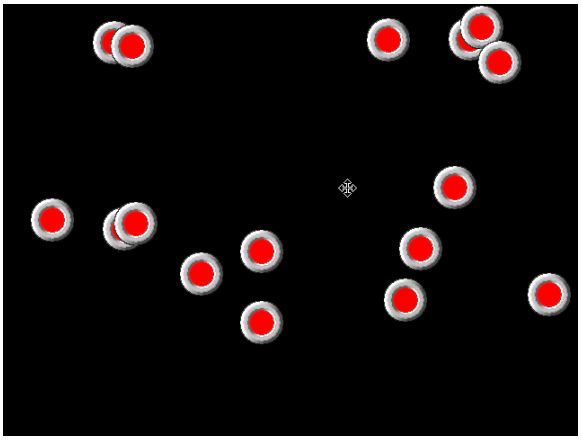
Ezenfelül a későbbiekben szeretnénk a megerősítéses tanulás technikáit alkalmazni adaptív beviteli eszközök kialakításában. Ilyen eszköz tanításakor a visszalépés fogja az elkerülendő büntetést jelenteni a megerősítéses tanulás számára.

2.2.2. Tracker

A Tracker a Fejegér használatának gyakorlását segítő játék. Játék közben a feladat a képernyőn látható tányérok eltüntetése kurzorral való rámutatással (2.3. ábra). A tányérok száma és mozgási sebessége állítható. A Tracker-el gyűjtött adatokat felhasználhatjuk többek között annak vizsgálatára, hogy mennyit fejlődött a felhasználó a Fejegér használatában.

2.2.3. VisTab

A VisTab speciálisan olyan felhasználók számára készült, akik nem képesek tartósan egy ponton tartani az egérkurzort és klikkelni. A VisTab-ban nagy méretű ikonok felett kell tartani a kurzort meghatározott ideig egy előre beállított funkció aktiválásához (2.4. ábra). A funkció leggyakrabban adott szöveg gép általi kimondását jelenti, de jelentheti új ikonokat tartalmazó táblára lapozást, vagy egy másik program elindítását is. A VisTab tulajdonképpen a hagyományos kommunikációs tábla számítógépes megfelelője. A táblák előre megszerkeszthetőek, így bővíthető a felhasználó által használható szavak, mondatok, kifejezések repertoárja.



2.3. ábra. A Tracker játék felülete.

2.4. ábra. A VisTab jellegzetes ikonjai.
A bal oldali ikon aktiválásával a felhasználó közli, hogy zenét szeretne hallgatni. Hogy ezután mi történik, az a beállítástól függ: A számítógép továbbíthatja a gondozónak a kérést élőszóban vagy távolra, vagy elindíthat egy zenelejátszó programot.

3. fejezet

Az RF-egér

A diplomamunka jelentős részét szoftverfejlesztés tette ki. Ennek során szenzoros gesztikuláció-mérő beviteli eszközt fejlesztettünk RF-MEMS alapú hardverre építkezve. A most következő fejezet célja az RF-egér bemutatása. Az egyes szakaszok a gesztikuláció-mérés módszereit (3.1), az RF-egér fejlesztéséhez használt hardver (3.2) és szoftver (3.3) eszközöket, végül a működési elvet (3.4), az implementációt (3.5), és a továbbfejlesztési lehetőségeket (3.6) tárgyalják.

3.1. A gesztikuláció mérésének módjai

A gesztikuláció felismerésének, mérésének felhasználási területei igen sokrétűek: A cél Virtuális Valóság rendszerekben a felhasználó mozgásának behelyezése a virtuális térbe, animációs filmek készítésekor a színészek felvett mozgása alapján mozgási modellek készítése, hadiiparban pedig hadieszközök hatékony vezérlésének megvalósítása. Utóbbira példa UAV[1] földi irányítása kézjelek segítségével [21][27].

A gesztikuláció mérésének többféle módja lehetséges az alkalmazott hardvertől függően. *Aktív* rendszerekben testre rögzített szenzorokat, *passzív* rendszerekben kamerákat, infra- és radar eszközöket, *hibrid* rendszerekben a kettő keverékét használják a mozgás mérésére. Az aktív rendszerek előnye, hogy használhatóságukat nem befolyásolja az időjárás és a fényviszonyok, ugyanakkor a testre rögzített szenzorok nehezek lehetnek, kábelek lóghatnak ki belőlük, ezért csak helyhez kötötten használhatók. Passzív rendszerekről épp ennek ellenkezőjét mondhatjuk el, míg a hibrid rendszerek célja a másik két típus gyengéinek kiküszöbölése.

A vezeték nélküli szenzorokat alkalmazó rendszerek igen hatékonyak, de számos új - elsősorban a vezeték nélküli technológiából származó - problémát vetnek fel. A különböző frekvenciákon, sávszélességen működő, és ezzel egy időben különböző teljesítményű adatfeldolgozást elosztott módon végző eszközök működését időben és térben össze kell hangolni.

[1]Unmanned Aerial Vehicle - pilóta nélküli robotrepülőgép

3.2. Hardver: RF-MEMS

A mikro-elektro-mechanikus-rendszerek (MEMS) [1][2] mikrorendszer technológiák[2] alkalmazásával készült hardverkomponensek, amelyekben szenzoros vagy beavatkozó mikromechanikus elemeket, CPU-t, memóriát, és rádiós vagy optikai kommunikációs elemeket egyetlen lapkára integrálnak. [29] Ennek eredményeképpen mikroméretű, pontos, energiatakarékos, megbízható eszközök gyárthatók nagy tömegben, ezáltal olcsón. A technológia alkalmazásának egyik legnagyobb problémája az eszközök működtetéséhez szükséges energia előállítása. Fény, hő, mikrohullámok, és kémiai, biológiai vegyületek használhatók energiaforrásként. Például az emberi testen belülre juttatott orvosi (RF-)Bio-MEMS [3] eszközök vércukor lebontásából nyerhetnek energiát.

3.1. ábra. MEMS gyorsulásmérő szenzorok egy pénzérmén.

Mikrohullámokkal kommunikáló MEMS-ek az RF-MEMS-ek. Ad hoc vezeték nélküli protokollokat alkalmazó RF-MEMS hálózatokat komplex környezetfigyelési, adatgyűjtési feladatokra használnak épületeken belüli helymeghatározó- és biztonsági rendszerekben [4], ipari környezetben üzemi viszonyok monitorozására, épületek, hidak megfigyelésére, robotikai alkalmazásokban, repülőgép- és autóiparban stb. [6] Az ilyen szenzorhálózatok előnye a hagyományos megfigyelő rendszerekkel szemben az elosztott feldolgozásból adódó nagyfokú hibatűrés és az egyszerű, gyors telepítés lehetősége.

Az RF-egér fejlesztésekor a Crossbow [5] MOTE-KIT5040 készletet használtuk. A készlet vezeték nélküli processzoros egységeket és rájuk rögzíthető szenzorlapokat tartalmaz. A szenzorlapokon MEMS gyorsulásmérők, magnetométerek, fény- és hőérzékelők, mikrofonok találhatók.

3.3. Szoftver: TinyOS

A TinyOS [8][9] beágyazott szenzorhálózatok nyílt forrású operációs rendszere. Komponens alapú architektúrája gyors fejlesztést tesz lehetővé, miközben a kód méretét - a szenzorhálózatok nódusaira jellemző kicsi memóriakapacitás miatt - minimalizálja. A TinyOS komponenskönyvtára hálózati protokollokat, elosztott szolgáltatásokat, szenzorok meghajtó programjait, és adatgyűjtő programokat tartalmaz - ezek mindegyike módosítható, finomítható a konkrét alkalmazás követelményeinek meg-

[2]Ilyen technológiák pl.: surface technology, microelectronics, micromechanics, microoptics, fibre optics, microfluidics

felelően. Esemény-vezérelt végrehajtási modellje eredményeképpen a rendszer energiatakarékos, ütemezője pedig kellően rugalmas a vezeték nélküli hálózati kommunikációra és a szenzoros mérésekre oly jellemző bizonytalanság hatékony kezelésében. A TinyOS programozási modelljét nesC nyelven [15][16] implementálták - e nyelven íródott az operációs rendszer nagy része. A továbbiakban röviden vázoljuk a modell általános jellemzőit (3.3.1) majd részletezzük a TinyOS komponens-architektúráját (3.3.2).

3.3.1. Általános jellemzők

Komponens alapú architektúra. A TinyOS újrafelhasználható rendszer-komponensekből épül fel. Egy alkalmazás a komponenseinek egymással való kapcsolatát leíró *kötési specifikáció* megadásával jön létre. Ez a specifikáció független a komponensek implementációjától. Az operációs rendszer egyes szolgáltatásainak külön komponensekbe helyezése lehetővé teszi, hogy a nem használt szolgáltatásokat kihagyjuk az alkalmazásból.

Taszkok és esemény-alapú konkurencia. A *taszkok* késleltetett lefutású műveletek; mindig befejeződésig futnak és nem preemptálják egymást. A komponensek indíthatnak taszkokat a *post* művelet segítségével. Hívásakor a post azonnal visszatér, az indított taszk pedig akkor kerül végrehajtásra, amikor az ütemező a sorban előtte levő összes taszkot végrehajtotta már. Minden nem időkritikus tevékenységet megvalósíthatunk taszkok használatával. A gyors válaszidő érdekében az egyes taszkoknak rövideknek kell lenniük, a hosszabb műveleteket több kisebb taszkba kell darabolni. A reaktivitás fenntartása érdekében a nagyon hosszú számításokat nem célszerű a szenzorhálózattal végeztetni.

Az *események* jelezhetik kétfázisú művelet (lásd alább) befejeződését vagy környezetből jövő eseményt - üzenet érkezését vagy egy időzítő lejártát. A TinyOS programot végső szinten hardver megszakítások hajtják. Ezek a programozási modellben *aszinkron események* formájában jelennek meg, amelyek mindig megszakítják az aktuális taszk vagy esemény futását.

Kétfázisú (split-phase) műveletek. Mivel a taszkok nem párhuzamosan futnak, nincsenek blokkoló műveletek sem. A várakozást igénylő műveletek végrehajtásakor szétválik a hívás és a válaszadás fázisa. Hívást *parancssal* kezdeményezhetünk. Kétfázisú műveletek esetén a parancs azonnal visszatér és a művelet eredményét később egy esemény fogja jelezni; nem kétfázisú műveleteknek ilyen eseménye nincsen. Tipikus példa kétfázisú műveletre az üzenetküldés: A hívó komponens a `send` parancssal indítja a küldést, amelynek befejeztét a kommunikációs komponens a `sendDone` esemény hívásával jelzi - ezt az eseményt a hívó implementálja. Tehát mindkét fél a kétfázisú művelet egyik fázisát implementálja és a másik komponensben hívja a másik fázist.

Energiatakarékosság. A reaktivitás hosszú távú fenntartása érdekében a végrehajtási sorban levő taszkok lefuttatása után a TinyOS ütemezője alvó üzemmódba kapcsolja a rendszert. Felébredés a környezetből jövő stimulus - vala-

mely mért fizikai jel megváltozása vagy üzenet érkezése - hatására következik be, majd a megfelelő reakció végrehajtása után a rendszer ismét elalszik. Ezért tapasztalhatjuk reaktív, energiatakarékos hálózatokban az aktivitás *hullám-szerű* terjedését.

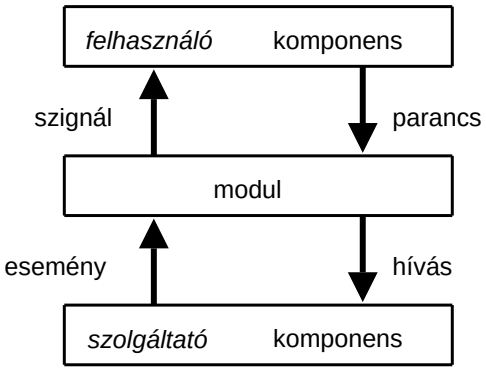
3.3.2. A TinyOS architektúrája

Kétféle komponens létezik: *modul* és *konfiguráció*. A modulok metódusok implementációit tartalmazzák. A kötési specifikációt is tartalmazó konfigurációk a komponensek egymással való kapcsolatát írják le. Minden TinyOS programot egy *top-level konfiguráció* határoz meg.

A komponensek kétirányú *interfészeken* keresztül kommunikálnak egymással. Az interfész *parancsokat* és *eseményeket* specifikál. A parancsokat az interfészt *szolgáltató*, míg az eseményeket az interfészt *felhasználó* komponensnek kell implementálnia.

A TinyOS modulokban *eseménykezelőket*, *parancsokat*, és *taszkokat* definiálhatunk. Az eseménykezelők és a parancsok a megfelelő interfész implementációját képezik, a taszkok pedig a modul nem időkritikus tevékenységeit tartalmazzák.

A konfigurációk komponenseket *kötnek* egymáshoz, és maguk is szolgáltathatnak, használhatnak interfészeket. Utóbbi esetekben a kötési specifikációban a konfiguráció leírja, hogy ténylegesen melyik komponense szolgáltatja, használja az adott interfészt.



3.2. ábra. TinyOS modul kapcsolata a környezetével.
Szolgáltató komponens *parancsának* futtatását *hívásnak*, felhasználó komponens *eseménykezelőjének* futtatását *szignálnak* nevezi a nesC terminológia.

3.4. Az RF-egér működési elve

Az RF-egér használatakor a testre rögzített szenzort különböző tengelyek mentén dönthetjük. Vízszintes nyugalmi helyzetben a szenzorlapon található gyorsulásmérő tengelye éppen merőleges a nehézségi erő irányára. Ilyenkor a mért gyorsulás közel nulla. A szenzor dőlésekor ez az érték szinuszosan növekszik egészen a függőleges helyzet eléréséig. Ezt a tényt kihasználva mérhetjük a dőlési szöget, ez pedig alkalmas vezérlés megvalósítására. A gyakorlatban a mérés lineáris leképezése majd küszöbölése elegendő az egérkurzor irányításához.

3.5. Az alkalmazás

Mivel az RF-egér kísérleti hardverre készített szoftver, nehéz élesen elkülöníteni a felhasználói és a fejlesztői dokumentációt, hiszen a hardvert birtokló felhasználó nagy valószínűség szerint ismeri a TinyOS fejlesztői környezetét is. A program használatakor éppen ezért a hardvereszközök beprogramozását és beállítását a programozó végezte, a felhasználóknak már csak a PC-s kezelőfelület megtanulása maradt. Így a most következő felhasználói dokumentációban is a kezelőfelület használatára fektetjük a hangsúlyt, majd az azt követő fejlesztői leírásban tárgyaljuk a hardvereszközök összeszerelésének, beprogramozásának módját.

3.5.1. Felhasználói dokumentáció

Az RF-egér szoftver segítségével egyetlen testre rögzíthető RF-MEMS szenzort használhatunk beviteli eszközként. A program kétféle vezérlési lehetőséget biztosít jelenleg: irányíthatjuk az egérkurzort, vagy küldhetünk gesztus-üzeneteket a VisTab-ba. A vezérlés sebességét és érzékenységét személyre szabottan beállíthatjuk.

Hardver és szoftver követelmények. A hardver összeszerelését és beállítását a fejlesztői dokumentációban tárgyaljuk. A program futtatásához *Java Runtime Environment 1.5*[3] szükséges. A Java telepítése után győződjünk meg róla, hogy futtatható-e parancssorból a JVM. Ehhez nyissunk egy parancssori ablakot és írjuk be, hogy `java -version`. Ha hibaüzenetet kapunk, akkor adjuk hozzá a `PATH`-hoz a Java `bin` könyvtárát. Részletes leírást a futtató környezet beállításáról a Java weboldalán[4] találhatunk.

A Java `jre1.5.0_05/bin` és a `jre1.5.0_05/bin/client` könyvtárait szintén adjuk hozzá a `PATH`-hoz. Ez a beállítás nem kötelező, szerepe a következő részből kiderül.

Telepítés és futtatás. Az RF-egér programjának telepítése a `legmouse_v16.zip` lemásolásával és kitömörítésével történik. A kitömörítés után kapott könyvtárban található `lm.cmd` parancsfájllal vagy a `legmouse.exe`-vel indíthatjuk el a programot. Ha a korábban említett `PATH` beállítást nem tettük meg, akkor csak az `lm.cmd` fog működni, ezt indíthatjuk rákattintással vagy parancssorból is. A `legmouse.exe`

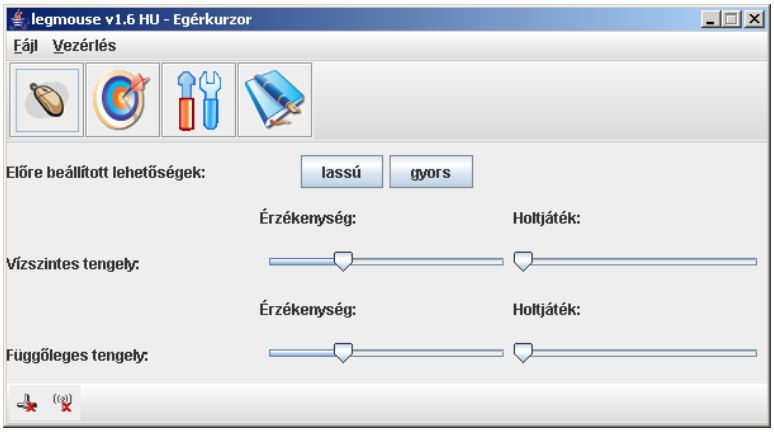
[3] http://java.sun.com/j2se/1.5.0/download.jsp
[4] http://java.sun.com/j2se/1.5.0/install.html

azért került bele a programba, mert futtatásakor nem jelenik meg parancssori ablak és ikonja is van, így valamivel kultúráltabb a megjelenése. A program későbbi változataihoz készülő telepítőprogramok már csak a `legmouse.exe`-t fogják használni a szükséges környezeti beállítások automatikus elvégzése után.

A PROGRAM KEZELŐFELÜLETE. Sikeres elindítás után egy dialógusablak jelenik meg, amelybe beírhatjuk a felhasználó nevét. Ennek az azonosításnak kettős szerepe van. Egyrészt minden felhasználó személyre szabott beállításait megjegyzi (és később automatikusan betölti) a program, így csak egyszer kell testreszabni az RF-egeret, másrészt a mérési adatok naplózásánál szükséges a felhasználó azonosítása. Ha már korábban használtuk a programot, a felhasználónevet nem szükséges minden indításkor újra beírni, kiválaszthatjuk a legördülő menüből is.

Az azonosítás megtörténte után megjelenik a 3.3. ábrán látható főablak, és még néhány panel amelyekkel később foglalkozunk. A főablak négy fő részből áll, ezek fentről lefele a következők.

1. Menüsor.
2. Eszköztár. Négy nagy színes gomb található rajta.
3. Középső panel. Ezen találhatjuk az aktuális egérmeghajtó beállítását lehetővé tevő vezérlőket.
4. Státuszsor. Két ikont tartalmaz, amelyek a hardver működési állapotáról adnak információt.



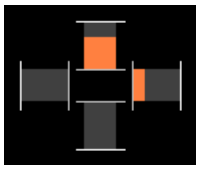
3.3. ábra. Az RF-egér programjának kezelőfelülete.

A részletek ismertetése előtt vázoljuk mit is csinál a program. Az RF-egér szoftvere az RF-MEMS hardver által mért gyorsulási adatok alapján vezérel valamit. A gyorsulási adatokat vezérléssé átalakító program-egységeket nevezzük *meghajtó*knak. Jelenleg kétféle meghajtót ismer a program. Az egyik az egérkurzort mozgatja, a másik pedig a VisTab program számára küld gesztus-üzeneteket. A kezelőfelületen tehát egyrészt a gyorsulási adatok forrásának paramétereit, másrészt a kiválasztott meghajtó paramétereit állíthatjuk be.

Amikor a programot egy új felhasználónévvel először indítjuk el, automatikusan megkeresi a csatlakoztatott hardvert, és ha megtalálta, el is indítja az egeret. Ezt a két műveletet követhetjük nyomon a *státuszsor ikonjai*val. Ha a bal oldali ikonon nincsen piros kereszt, akkor a program megtalálta a bázist. A jobb oldali ikon pedig az előzőhöz hasonló módon azt jelzi, hogy érkeznek-e adatok az egértől.

Ha a forrás automatikus keresése sikertelen, úgy lehetőségünk van a paraméterek kézi beállítására is a *Beállítások* dialógusban, amelyet a *Fájl* menü *Beállítások...* pontjára kattintva érhetünk el, vagy az eszköztár csavarhúzót ábrázoló ikonjára kattintva. Ezekről a paraméterekről később még lesz szó.

HANGOLÁS ÉS VEZÉRLÉS. Ha az RF-egér sikeresen elindult - ezt láthatjuk a testre szerelt eszköz villogó piros ledjéből és a státuszsor jobb oldali ikonjáról - elkezdődhet a meghajtó *hangolása*. A *Vezérlés* menü utolsó menücsoportjában találhatók a használható meghajtók. Miután ezek közül kiválasztottuk a megfelelőt, a középső panelen láthatjuk az adott meghajtó paramétereinek hangolását szolgáló vezérlőket. A hangolást segíti a képernyő bal felső sarkában található *ellenőrző ablak*, ill. az egyes meghajtókhoz tartozó *hangoló* ablakok is. Az ellenőrző ablakon azt láthatjuk, hogy az eszköz milyen irányokban mekkora gyorsulást mér, ill. hogy ehhez képest mekkora küszöböket használ az adott meghajtó (3.4. ábra). A *küszöbök* szerepét az egyes meghajtók tárgyalásánál részletezzük.



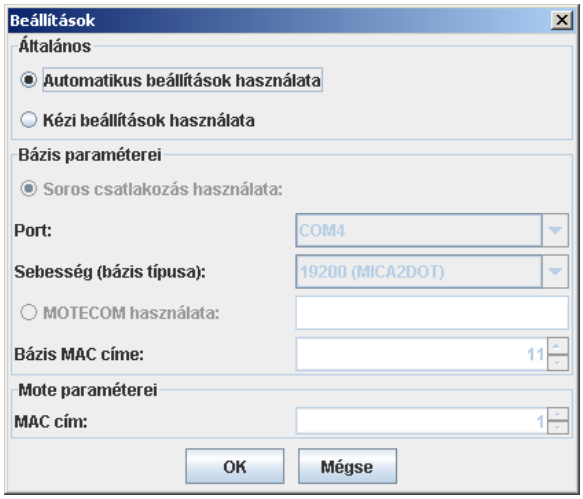
3.4. ábra. Az ellenőrző ablak.
A narancssárga csíkok hossza mutatja az adott irányban mért gyorsulás mértékét.

Három meghajtó-független hangolási paraméter létezik. Az első az ún. *referenciapont*, amely azt mutatja, hogy mely mérési értéket tekintünk 0-nak, vagy középpontnak. Beállításához hozzuk a szenzort a kívánt helyzetbe - például tegyük le egy vízszintes felületre - majd kattintsunk az eszköztár céltáblát ábrázoló ikonjára, vagy válasszuk ki a *Vezérlés/Középpont rögzítése* pontot. Ezt a kalibrálást minden szenzorra el kell végezni egyszer, ui. a rajtuk levő gyorsulásmérők sosem tökéletesen vízszintesek, azaz az egér nyugalmi helyzetében is valamilyen irányú mérést fog jelezni az ellenőrző ablak.

A másik két paraméterrel a vízszintes és függőleges irányú mérési adatok előjelét fordíthatjuk meg, ezáltal felcserélhetjük a vezérlési irányokat. *Vezérlés/Vízszintes irányok megfordítása* ill. *Vezérlés/Függőleges irányok megfordítása* menüpontokban találhatjuk ezeket.

Miután elvégeztük a hangolást, bekapcsolhatjuk az egeret a *Vezérlés/Egér bekapcsolása* menüpont kiválasztásával, vagy a Ctrl+E billentyűkombináció használatával. Ilyenkor eltűnnek a hangolást segítő ablakok. A nem egérkurzort vezérlő meghajtók használata esetén működés közben is hangolhatjuk a paramétereket, míg az egérkurzort vezérlőknél a hangolás előtt célszerű kikapcsolni az egeret.

Az adatforrás paraméterei. A *Fájl* menü *Beállítások...* pontjára vagy az eszköztár csavarhúzót ábrázoló ikonjára kattintva állíthatjuk be a gyorsulási adatok forrását. Általános esetben az automatikus beállítások megfelelőek, és nem kell használnunk ezt a dialógust (3.5. ábra).



3.5. ábra. A forrás paramétereinek beállítása.

Kézi beállítások esetén a bázis sebességét, fizikai címét, és a kommunikációs port számát, ill. a testre rögzített eszköz fizikai címét adhatjuk meg. A sebesség hardvertípustól függő érték, a fizikai címeket a beprogramozásnál adja meg a programozó, a kommunikációs port pedig attól függ, hogy a PC melyik portjára csatlakoztattuk a bázist. A MOTECOM segítségével speciális forrásokat állíthatunk be, ezt csak fejlesztési célra használjuk [9].

Az Egérkurzor-meghajtó. Az RF-egér alapértelmezett meghajtója az Egérkurzor-meghajtó. Hangoló ablakában kék színű kurzort láthatunk, amely azt mutatja, hogyan fog mozogni a kurzor az egér bekapcsolása után a jelenlegi paraméterekkel. Mindkét irányban állíthatjuk az érzékenységet és a holtjátékot. Az érzékenység a kurzor sebességét határozza meg, míg a holtjáték annak mértéke, hogy a középponthoz mennyire közeli méréseket tekintünk nullának. Például ha remeg a felhasználó keze, érdemes nagyobb holtjátékot beállítani, így könnyebb lesz a középpontban tartani a szenzort. A meghajtónak van egy *lassú* (kis érzékenységű, nagy holtjátékú) és egy *gyors* (nagy érzékenységű, kis holtjátékú) beállítási lehetősége is. Az ellenőrző ablakban láthatjuk, hogy a holtjáték az alsó küszöbnek felel meg, az adott irányban tehát csak efölötti mérésre van mozgás.

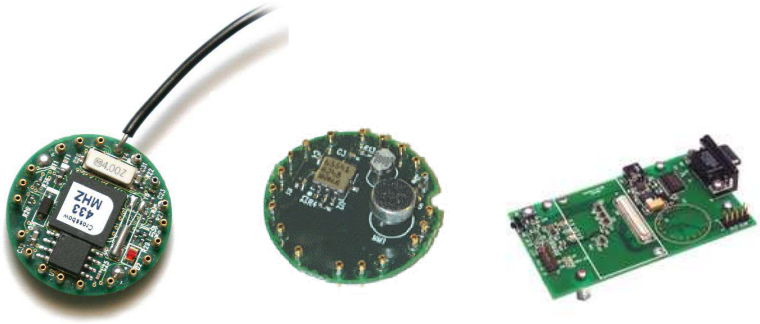
A Gesztusok-meghajtó. Hangoló ablaka hasonló az ellenőrző ablakhoz, benne az adott irányú gesztus kiváltódását piros téglalap felvillanása jelzi. Ennél a meghajtónál az érzékenység a felső küszöböt, míg a holtjáték a felső és az alsó küszöbök különbségét jelenti. Ha a gesztus a *szenzor bedöntését* jelenti a beállítás szerint, akkor a felső küszöböt meghaladó mérés fogja kiváltani a gesztust, és újabb gesztust csak alsó küszöb alatti mérés után lehet kiváltani. Vagyis az adott irányú döntés után vissza kell hozni középponti helyzetbe a szenzort. Ha pedig *bedöntést és visz-*

szaállítást jelent a gesztus, ugyanezek a szabályok azzal a különbséggel, hogy az alsó küszöb alá visszatérve váltódik ki a gesztus. Vagyis a bedöntés ÉS a visszahozás jelenti az aktiválást. Itt is van két gyorsállítási lehetőség, a *durva* nagyobb, míg az *érzékeny* kisebb kimozdításokat igényel a gesztusok aktiválásához. Ha az egeret bekapcsoljuk, a VisTab-ba küldi a gesztus-üzeneteket.

AZ ADATOK NAPLÓZÁSA. Az RF-egér szoftvere minden egyes egér-bekapcsoláskor nyit egy új naplót, amelybe a gyorsulási értékeket és a hozzájuk tartozó időbélyegeket menti. Ez a napló a `log` alkönyvtárba kerül tömörített szöveges fájl formájában. Neve tartalmazza az RF-egér meghajtó nevét, a felhasználó azonosítóját, és a dátumot.

3.5.2. Fejlesztői dokumentáció

A HARDVER ÖSSZESZERELÉSE. Az RF-egér használatához egy szenzoros egységet és egy bázist kell összeraknunk. A teszteléshez használt konfiguráció építőkövei a 3.6. ábrán láthatók.



3.6. ábra. Az RF-egér hardver elemei.
(A) Crossbow MICA2DOT processzor egység (*mote*); (B) Crossbow MTS510CA szenzorlap; (C) Crossbow MIB510CA interfész-lap.

A testre rögzíthető szenzor megépítéséhez illesszünk össze egy - előzőleg a `SenseAccel` programmal feltöltött - processzoros egységet (*mote*-ot) egy szenzorlappal és helyezzük bele az elemet. A bázist egy - a `TOSBase` programot futtató - processzoros egység és az interfész-lap összeillesztésével kapjuk. Ezután csatlakoztassuk az interfész-lapot a PC-hez az RS-232 csatlakozó segítségével és kössük rá az adapterét is.

A HARDVER-EGYSÉGEK BEPROGRAMOZÁSA. A mote-okat az első használatkor be kell programozni. A terjedelemre való tekintettel itt csak röviden összefoglaljuk a tennivalókat. Részletesebben lásd [9]. Először a beprogramozni kívánt mote-ot helyezzük az interfész-lapra, az interfész lapot pedig csatlakoztassuk a PC-hez és kössük rá az adapterét. Ezután bármely program feltöltéséhez lépjünk bele a program könyvtárába, majd TinyOS környezetben adjuk ki a következő parancsot:

```
make mica2dot install,11 mib510,com5
```

A megadott példában a `COM5` soros portra volt csatlakoztatva az MIB510CA típusú interfész-lap és a beprogramozott mote fizikai címe 11 lesz. A parancs először lefordítja az adott könyvtárbeli programot, majd feltölti azt a mote memóriájába. Az RF-egérhez a szenzor `SenseAccel` programját és a bázis `TOSBase` programját kell a fenti módon (különböző fizikai címekkel) két mote-ra telepíteni.

Az RF-egér szenzor programja. A `SenseAccel` program a szenzor processzorán fut. A gyorsulásmérőket mintavételezi, és simítja, majd elküldi az adatokat rádión a bázisállomás felé. A szoftver TinyOS-nesC környezetben készült és a következő modulokból áll:

radio.h Paraméter-konstansok és a hálózati protokoll adatszerkezeteinek definícióit tartalmazza.

SenseAccel.nc A program top-level konfigurációja. A komponensek egymással való kapcsolatát írja le.

SenseAccelM.nc A program egyetlen saját modulja. Tulajdonképpen ez a komponens tartalmazza a tényleges implementációt.

A korlátozott erőforrások miatt a program energiatakarékos: Csak akkor mér, ha erre rádión kérik. Éppen ezért induláskor a mote *alvó* állapotban van, és szabályos időközönként belehallgat a rádiós csatornába. Ha ilyenkor kap mérést kérő üzenetet, akkor bekapcsolja a szenzorokat és megkezdi a *mérést*, különben pedig ismét elalszik. Az utolsó mérésre vonatkozó kérés után meghatározott idővel visszatér *figyelő* állapotba, és ha nem kap kérést, elalszik. A folyamatos működéshez tehát szabályos időközönként kéréseket kell küldenünk a szenzor felé. A szenzor által küldött adatok a méréseken kívül időbélyeget is tartalmaznak.

A programnak két állítható paramétere van. A `FILTER_WIDTH` 1-nél nagyobb egész szám és a küldendő értékekben szereplő minták számát adja meg. Nagyobb érték esetén tehát jobban simítunk. A `HALF_BUF_SIZE` értéke 1 és 5 közötti egész lehet és az egyes csomagokban (tengelyenként) átvitt értékek számát határozza meg. A fejlesztés során mindkét paraméter úgy lett meghatározva, hogy az átlagos átvitt mintaszám - a gyorsulásmérő rögzített, legpontosabb mérést adó mintavételi frekvenciája mellett - maximális legyen.

Az RF-egér programja. A szoftver TinyOS-Java környezetben készült NetBeans IDE[5] segítségével. A következő táblázatban megadjuk az egyes modulok funkcióját, szerepkörét. Ennél részletesebb információt a forráskódból nyerhet az olvasó.

Funkció, szerepkör	Modulok (`nipg.gesture.*`)
A TinyOS hálózati szolgáltatásait használó modulok. Csatlakozást biztosítanak az adatforráshoz manuálisan vagy automatikusan beállított paraméterekkel.	`AutoSource`, `DataSource`, `DataSourceWrapper`, `MoteIFThread`, `MessageListenerItem`, `NotifyListener`
Az egérmeghajtók absztrakt ősosztálya és a két meghajtó: A Gesztusok- és az Egérkurzor-meghajtó.	`MouseDriver`, `GestureImpl`, `VelocityImpl`
A beállításokat író és olvasó modulok. A `java.utils.prefs` csomagot használják.	`Preferences`, `PreferencesCallback`
Az adatok naplózását végző modul.	`Log`
A platformfüggő kódrészek interfész modulja.	`Native`
A főprogram.	`Main`
A bejelentkező-, Beállítások..., és Napló dialógusok.	`LoginDlg`, `SettingsDlg`, `LogDlg`
A meghajtók hangoló ablakait tartalmazó modulok.	`CursorTestDlg`, `RemoteDlg`, `RemotePanel`
A bázis-kereső protokollt tesztelő és a beállításokat törlő segédprogramok.	`TestBaseProtocol`, `ClearPreferences`

Az RF-egér programjához a Java projekten túl két C++ nyelvű projekt is tartozik. A *gesture_native* tartalmazza az egérkurzor vezérlést és a gesztus-üzenetek küldését megvalósító Windows specifikus kódrészeket. Az *Invoker* az RF-egér `legmouse.exe`-vel történő - parancssori ablak nélküli - indítását teszi lehetővé. A C++ és a Java közötti kapcsolatot megvalósító JNI technológiáról lásd [7].

3.6. További lehetőségek

Az RF-egeret eredetileg az SKMK-ban élő mozgássérült gyermekek számára készítettük. A program egyetlen szenzort használva egérkurzort vezérel. Ezen túlmenően bármilyen 2 szabadságfokú vezérlés megvalósítására alkalmas.

A technológia fejlődésének eredményeképpen a szenzorok a későbbiekben egyre kisebbek lesznek, így lehetőség nyílik arra, hogy több szenzort rögzítsünk a felhasználóra anélkül, hogy jelentősen megterhelnénk őt. Több szenzorral komplexebb elemzések lehetségesek és összetettebb, finomabb vezérlés valósítható meg. Ilyen beviteli eszközökhöz bonyolultabb kezelőfelületek tervezése is szükséges, hiszen az

[5] http://www.netbeans.org/

irányításra használható szabadsági fokok száma lényegesen nagyobb lehet.

Az RF-egeret számos rendezvényen bemutattuk már. Ezek közül a legjelentősebbek voltak: (i) Body Sensor Networks Workshop 2005 [24] alkalmával AIBO robotkutya[6] vezérlése; (ii) BME Villamosmérnöki és Informatikai Karon tartott bemutatón Puma robot[7] irányítása; (iii) V. Kelet-, Közép-Európai Regionális Augmentatív és Alternatív Kommunikációs Konferencia "Kommunikáció a Közös Világban"[8] alkalmával.

Egy öszetettebb gesztikuláció mérő, felismerő rendszer megalkotásához a következő problémákat kell még megoldanunk:

- Több eszköz egyidejű használatakor csökkenni fog az egyes szenzorok sávszélessége, ezért bonyolultabb kommunikációs és adatfeldolgozási protokollra lesz szükségünk.

- A jelenlegi RF-egér nem képes a függőleges tengely körüli elfordulásokat mérni. Ezért várhatóan a gyorsulásmérőn kívül a magnetométer adatait is használni kell majd.

- A nagy pontosságú orientációs szenzorok általában 3-tengelyes gyorsulásmérőt, magnetométert, giroszkópot tartalmaznak. A mi eszközeinkben csak 2-tengelyes gyorsulásmérő és magnetométer van. Vajon ezekkel mekkora pontosságot lehet elérni? Ha egyenként nem elég pontosak, hogyan használhatnánk kis csoportokban őket? Ilyenkor ismét probléma a hatékony hálózati kommunikáció megoldása.

- Különösen érdekelnek bennünket a felhasználók készségeit fejleszteni képes interakciós eszközök. A Fejegér használói jelentős mértékben fejlesztették fejmozgásuk koordinációját. Szeretnénk, ha az RF-MEMS alapú gesztikulációmérő, felismerő rendszer is ilyen módon javítaná a felhasználók mozgáskoordinációját.

- A konfiguráció mérést nem csak az egyes testrészek orientációjának mérésével valósíthatjuk meg. A szenzorok egymástól való távolságának meghatározásával is megállapíthatjuk a test konfigurációját. A távolságbecslés történhet pl. rádió-jelerősség (RSSI) alapján, vagy rádió- és ultrahang jelek érkezése közötti időbeli késés mérésével. A rádió-jelerősség mérésével végzett kísérleteink alapján elmondhatjuk, hogy az a gyakorlatban nem használható az elnyelődés és visszaverődés által okozott nagy szórás miatt. Ultrahangos eszközökkel már lehetne néhány cm-es pontosságú méréseket végezni, azonban a rendelkezésünkre álló ultrahangos szenzorok túl nagyok ahhoz, hogy testre rögzítsük őket.

[6] http://www.eu.aibo.com/
[7] http://www.rpautomation.com/
[8] http://www.bliss.org.hu/

4. fejezet

Alkalmazott matematikai eszköztár

4.1. Rejtett Markov Modellek

A Rejtett Markov Modellek legismertebb felhasználási területe a beszédfelismerés. Ugyanakkor - mint a kísérleteknél is látni fogjuk - hasznosnak HCI[1] eszközök használatának elemzésekor is. A (4.1.1) pontban definiáljuk a Markov folyamatot, majd bevezetjük a Rejtett Markov Modell fogalmát (4.1.2), bemutatjuk az alkalmazáshoz szükséges legfontosabb feladatokat (4.1.3) - köztük a tanítási feladatot - és azok megoldásait (4.1.4). Felírjuk a tanítási egyenleteket folytonos esetre is (4.1.5), végül bemutatunk egy explicit állapot-időtartam sűrűséggel rendelkező modellt (4.1.6). Ez precízebb modellezésre képes, de számításigénye jóval nagyobb.

A fejezet elsődleges forrása Lawrence R. Rabiner munkája [25].

4.1.1. Diszkrét Markov folyamat

Legyen egy folyamat minden $t = 1, 2, \ldots$ diszkrét időpillanatban az

$$S = \{S_1, S_2, \ldots, S_N\}$$

állapotok valamelyikében. Ezt az állapotot jelölje q_t. A folyamat minden lépésben új állapotba kerül valószínűségi értékek alapján. Általános esetben $q_t = S_j$ valószínűsége függhet attól, hogy a t-t megelőző pillanatokban milyen állapotban voltunk. Elsőrendű diszkrét Markov folyamat esetében ez a valószínűség csak az aktuális és az azt megelőző állapottól függ, azaz

$$P[q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots] = P[q_t = S_j | q_{t-1} = S_i] \tag{4.1}$$

A továbbiakban csak olyan folyamatokkal foglalkozunk amelyekben a (4.1) valószínűségek időben nem változnak. Ez a *stacionaritási* feltevés. Az állapotátmenetek valószínűségi mátrixára vezessük be a következő jelölést:

$$a_{ij} = P[q_t = S_j | q_{t-1} = S_i] \qquad 1 \leq i, j \leq N$$

[1]Human Computer Interaction - Ember-számítógép interakció

ahol $a_{ij} \geq 0$ és $\sum_{j=1}^{N} a_{ij} = 1$. Az ilyen sztochasztikus folyamatot nevezhetnénk megfigyelhető Markov folyamatnak is, mert itt a megfigyelés éppen a folyamat aktuális állapota.

4.1.2. A Rejtett Markov Modell

Az előbb olyan Markov modellt vizsgáltunk, amelynek állapotai megfigyelhetőek. A következőkben olyan modellről lesz szó, amelyben a megfigyelés az állapotok valamilyen sztochasztikus függvénye, így a belső állapotok rejtve maradnak. Az ilyen modellt Rejtett Markov Modellnek (RMM, Hidden Markov Model - HMM) nevezzük. A következő paraméterekkel írunk le egy RMM-t:

1) N, a rejtett állapotok száma. A gyakorlatban ezt a számot gyakran a modellezni kívánt folyamatra vonatkozó előzetes ismeretünk alapján adjuk meg.

2) M, a különböző megfigyelhető szimbólumok száma, vagyis a modellezni kívánt folyamat kimenetének megfelelően választott diszkrét ábécé elemeinek száma. Jelölje $V = \{v_1, v_2, \ldots, v_M\}$ az ábécét.

3) $A = \{a_{ij}\}$, az állapotátmeneti eloszlásmátrix, ahol

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i] \qquad 1 \leq i, j \leq N$$

Ha $\forall i, j : a_{ij} > 0$, akkor minden állapotból minden állapotot elérhetünk az RMM-ben. Léteznek olyan modellek is, amelyekben ezt nem engedjük meg, például balról-jobbra modellek, amelyeket időben változó adatsorok (pl. beszédhangok) reprezentálására használnak.

4) $B = \{b_j(k)\}$, a megfigyelt (kimeneti) szimbólumok eloszlása a j állapotban, azaz

$$b_j(k) = P[v_k \text{ a megfigyelés} | q_t = S_j] \qquad \begin{matrix} 1 \leq j \leq N \\ 1 \leq k \leq M \end{matrix}$$

5) $\pi = \{\pi_i\}$ a kezdeti-állapot valószínűségek, azaz

$$\pi_i = P[q_1 = S_i] \qquad 1 \leq i \leq N$$

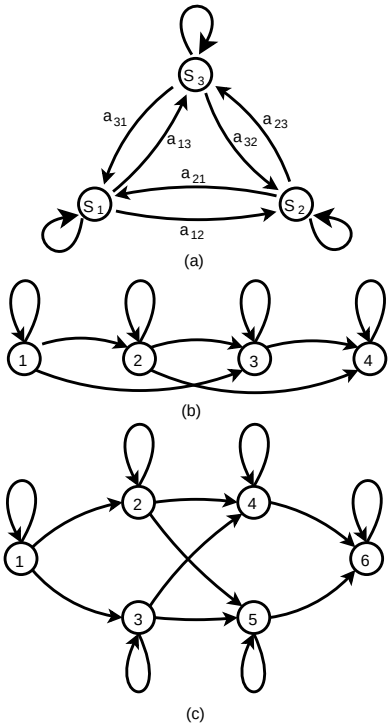
A tömörség érdekében vezessük be a $\lambda = (A, B, \pi)$ jelölést a modell paramétereinek leírására.

4.1.3. Az RMM három alapfeladata

A Rejtett Markov Modell használatához három alapfeladatot kell megoldanunk:

1. Adott $O = O_1 O_2 \ldots O_T$ megfigyeléssorozat és $\lambda = (A, B, \pi)$ modell esetén hogyan tudjuk hatékonyan kiszámítani a $P(O|\lambda)$ valószínűséget?

2. Az $O = O_1 O_2 \ldots O_T$ megfigyeléssorozat és $\lambda = (A, B, \pi)$ modell ismeretében melyik az a $Q = q_1 q_2 \ldots q_T$ belső állapotsorozat, amelyik valamilyen értelemben a legjobban leírja a megfigyelt sorozatot?



4.1. ábra. Különböző típusú RMM-ek.
(a) 3-állapotú ergodikus modell. (b) 4-állapotú balról-jobbra modell. (c) 6-állapotú párhuzamos balról-jobbra modell.

3. Hogyan maximalizáljuk a $P(O|\lambda)$ valószínűséget a modell $\lambda = (A, B, \pi)$ paramétereinek függvényében?

Az 1. kérdés a *kiértékelési* feladat. Hogyan számíthatjuk ki a $P(O|\lambda)$ valószínűséget, ha ismerjük O-t és λ-t? Másként fogalmazva, a modell mennyire jól írja le a megfigyeléssorozatot, vagyis mennyire illeszkedik jól? Ez utóbbi különösen fontos, hiszen e probléma megoldása lehetővé teszi, hogy több modell esetén kiválasszuk a megfigyeléseinkhez legjobban illeszkedőt.

A 2. kérdés a *dekódolási* feladat. A modell belső állapotainak felfedésére vonatkozik, arra, hogy megtaláljuk azt az állapotsorozatot, amely a megfigyelést generálta. Látnunk kell azonban, hogy a pontos állapotsorozatot - a degenerált modellek kivételével - nem tudjuk megadni. Ezért a gyakorlatban a konkrét alkalmazástól függő optimalitási kritériumokat vezetnek be az állapotsorozatokra. Ennek a kérdésnek a megválaszolásával következtethetünk például a modell belső szerkezetére, egy adott megfigyeléssorozathoz tartozó optimális állapotsorozatra, stb.

A 3. kérdés a *tanítási* vagy modellillesztési feladat (identifikáció). Legtöbbször rögzített O megfigyeléssorozat birtokában keressük a modell $\lambda = (A, B, \pi)$ paramétereinek azon értékét, amelyre a $P(O|\lambda)$ valószínűség maximális. Ez tulajdonképpen az RMM tanítását jelenti az O sorozatra.

4.1.4. Az RMM három alapfeladatának megoldásai

A következő három alfejezetben áttekintjük a három alapfeladat - gyakorlatban leginkább használt - megoldásait. A leírt algoritmusok programozásához még számos numerikus problémát kell megoldani, amelyeket itt nem tárgyalunk.

A kiértékelési feladat megoldása: Forward algoritmus

Ismerjük tehát az $O = O_1 O_2 \dots O_T$ megfigyeléssorozatot, és a $\lambda = (A, B, \pi)$ modell paramétereket. Feladatunk a $P(O|\lambda)$ valószínűség kiszámítása. Ehhez felhasználjuk, hogy

$$P(O|\lambda) = \sum_Q P(O, Q|\lambda) = \sum_Q P(O|Q, \lambda) P(Q|\lambda)$$

ahol

$$Q = q_1 q_2 \dots q_T \tag{4.2}$$

tetszőleges állapotsorozat. Az O megfigyeléssorozat Q állapotsorozathoz tartozó valószínűsége

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda)$$

alakú, ahol feltettük, hogy a megfigyeléseink függetlenek. Jelölésünk alapján

$$P(O|Q, \lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T)$$

A (4.2) állapotsorozat valószínűsége

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T}$$

Az eddigieket összegezve:

$$\begin{aligned} P(O|\lambda) &= \sum_Q P(O|Q, \lambda) P(Q|\lambda) \\ &= \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T) \end{aligned} \tag{4.3}$$

A $P(O, Q|\lambda)$ kiszámítását a fentiek alapján a következőképpen interpretálhatjuk. Kezdetben ($t = 1$) a q_1 állapotban vagyunk π_{q_1} valószínűséggel, és az O_1 szimbólumot generáljuk $b_{q_1}(O_1)$ valószínűséggel. A következő pillanatban ($t = t + 1$) átlépünk q_1-ből a q_2 állapotba $a_{q_1 q_2}$ valószínűséggel, majd generáljuk O_2-t $b_{q_2}(O_2)$ valószínűséggel, s. í. t., egészen az utolsó T-edik szimbólum generálásáig.

Bár az eddig leírt kiszámítási mód helyes, a gyakorlatban használhatatlan, hiszen a $P(O|\lambda)$ kiszámítása (4.3) alapján nagyságrendileg $2T \cdot N^T$ műveletet igényel: A lehetséges T hosszú állapotsorozatok száma N^T, és minden ilyen sorozatra kb. $2T$ műveletet kell elvégezni. (Egészen pontosan $(2T - 1)N^T$ szorzásra, és $N^T - 1$ összeadásra van szükség.)

A következőkben ismertetünk egy hatékony eljárást $P(O|\lambda)$ kiszámítására, amelyet *Forward-Backward* módszernek hívnak. Vezessük be az

$$\alpha_t(j) = P(O_1 O_2 \cdots O_t, q_t = S_j|\lambda) \tag{4.4}$$

Forward változót, azaz $\alpha_t(j)$ jelentse annak valószínűségét, hogy az $O_1 O_2 \ldots O_t$ sorozatot figyeltük meg, és a t pillanatban az S_j állapotban vagyunk. Könnyen belátható, hogy az $\alpha_t(j)$ kiszámítható az

$$\alpha_1(j) = \pi_j b_j(O_1) \qquad 1 \le j \le N \tag{4.5}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad 1 \le t \le T-1, \quad 1 \le j \le N \tag{4.6}$$

egyenletekkel megadott rekurzióval, és így

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j) \tag{4.7}$$

$P(O|\lambda)$ kiszámítása (4.7) alapján már csak N^2T lineáris műveletet igényel, szemben a $2T \cdot N^T$ exponenciális komplexitással.

A dekódolási és tanítási feladatok megoldásához szükségünk lesz még a *Backward* változó bevezetésére, amely a maradék $O_{t+1} O_{t+2} \ldots O_T$ megfigyelés sorozat $q_t = S_i$ feltételre vonatkozó valószínűségét adja meg. Formálisan

$$\beta_t(i) = P(O_{t+1} O_{t+2} \cdots O_T | q_t = S_i, \lambda) \tag{4.8}$$

A $\beta_t(i)$ szintén kiszámítható rekurzióval:

$$\beta_T(i) = 1 \qquad 1 \le i \le N \tag{4.9}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \qquad t = T-1, T-2, \ldots, 1, \quad 1 \le i \le N \tag{4.10}$$

Megjegyezzük, hogy $\alpha_t(i)\beta_t(i) = P(O, q_t = S_i|\lambda)$ és így

$$P(O|\lambda) = \sum_{i=1}^{N} P(O, q_t = S_i|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) \tag{4.11}$$

tetszőleges t-re, azaz újabb módszert nyertünk $P(O|\lambda)$ kiszámítására.

A dekódolási feladat megoldása: Viterbi algoritmus

A dekódolási feladatra - adott megfigyeléssorozathoz optimális állapotsorozat megtalálására - számos algoritmus adható attól függően, hogy mit nevezünk optimális állapotsorozatnak.

Egy lehetséges megközelítés, ha az O sorozat minden O_t tagjához egyenként választjuk ki azt az S_j állapotot, amelyre $P(O_t|q_t = S_j)$ maximális. Az ilyen S_j állapotokat *optimális* állapotoknak nevezzük. Vagyis az optimális állapotsorozat az, amelyben az optimális állapotok számának várható értéke maximális. A megoldáshoz vezessük be a

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{4.12}$$

változót, annak valószínűségét, hogy a t pillanatban S_i állapotban vagyunk, feltéve, hogy ismerjük az O sorozatot. A (4.4) és a (4.8) definíciókból adódik, hogy

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

A feladat tehát azon q_t állapotok meghatározása, amelyekre

$$q_t = \arg\max_{1\leq i\leq N}[\gamma_t(i)] \qquad 1 \leq t \leq T \tag{4.13}$$

Habár (4.13) maximalizálja a helyes állapotok számát (minden O_t tagra a legvalószínűbb S_j kiválasztásával), nem vettük figyelembe az a_{ij} állapotátmenet valószínűségeket, így előfordulhat, hogy a kapott állapotsorozat illegális, mert tartalmaz 0 valószínűségű átmeneteket. (Előfordul benne S_iS_j, miközben $a_{ij} = 0$.)

Egy lehetséges megoldás, ha helyes állapotok helyett helyes állapotpárok (q_tq_{t+1}), állapothármasok ($q_tq_{t+1}q_{t+2}$), stb. várható értékét maximalizáljuk. Leggyakrabban a teljes Q sorozatra maximalizálják $P(Q|O, \lambda)$ értékét. Mivel O rögzített, ez megegyezik $P(Q, O|\lambda)$ maximalizálásának feladatával.

A dinamikus programozás technikáját alkalmazó megoldás a Viterbi-algoritmus [11][10][18]: Legyen $O = O_1O_2 \cdots O_T$ rögzített megfigyeléssorozat. A hozzátartozó legvalószínűbb $Q = q_1q_2 \cdots q_T$ állapotsorozat megtalálásához definiáljuk a következő mennyiséget:

$$\delta_t(i) = \max_{q_1q_2\cdots q_{t-1}} P[q_1q_2 \cdots q_t = S_i, O_1O_2 \cdots O_t|\lambda]$$

azaz $\delta_t(i)$ jelöli az első t megfigyeléshez tartozó, S_i-ben végződő legvalószínűbb állapotsorozat valószínűségét. Nyilván igaz a következő rekurzió:

$$\delta_1(j) = \pi_j b_j(O_1) \tag{4.14}$$

$$\delta_{t+1}(j) = \max_i[\delta_t(i)a_{ij}] \cdot b_j(O_{t+1}) \tag{4.15}$$

Ahhoz, hogy $\max_{1\leq i\leq N}[\delta_T(i)]$ kiszámítása után visszafelé lépkedve megkapjuk az optimális állapotsorozatot, fel kell jegyeznünk (4.15)-ben a max argumentumát minden t-re és j-re. Ehhez bevezetjük a $\psi_t(j)$ változót. Az algoritmus tehát:

1. Inicializálás:

$$\delta_1(j) = \pi_j b_j(O_1) \qquad 1 \leq j \leq N$$
$$\psi_1(j) = 0$$

2. Iteráció:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \qquad 2 \leq t \leq T$$
$$1 \leq j \leq N$$

$$\psi_t(j) = \arg\max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \qquad 2 \leq t \leq T$$
$$1 \leq j \leq N$$

3. Terminálás:

$$p^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$
$$q_T^* = \arg\max_{1 \leq i \leq N} [\delta_T(i)]$$

4. Az optimális állapotsorozat kinyerése:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, \ldots, 1$$

Az algoritmus végrehajtása után $p^* = \max_Q P(Q, O|\lambda)$, és $Q^* = (q_t^*)_{1 \leq t \leq T}$ a keresett állapotsorozat, amelyre $P(Q^*, O|\lambda) = p^*$.

A tanítási feladat megoldása: Baum-Welch algoritmus

A tanítási feladat megoldására többféle megközelítés létezik. Ezek közül a legfontosabbak a Maximum Likelihood (ML) és a Maximum Mutual Information (MMI) kritériumokon alapuló módszerek. Az ML kritérium használata esetén célunk egyetlen RMM paramétereinek hangolása rögzített megfigyeléssorozat alapján. A gyakorlatban azonban ez nem mindig elegendő. Az RMM-ek legfontosabb felhasználási területe a beszédfelismerés, ahol a beszéd egy O részletét kell osztályozni. Minden osztályt egy λ_k RMM-el reprezentálnak, és az O sorozat abba a k osztályba sorolódik, amelyre $P(O|\lambda_k)$ maximális. Az ilyen diszkriminatív modellek használata esetén olyan kritérium alapján végezzük a tanítást, amely biztosítja, hogy az O megfigyeléssorozatnak megfelelő λ_O osztályra $P(O|\lambda_O)$ növekedjen, miközben az összes többi λ_k osztályra $P(O|\lambda_k)$ csökkenjen a tanítás során. Ezt a kritériumot nevezik MMI-nek, mert a sorozatok és a nekik megfelelő osztályok közötti információ mennyiségét maximalizálja.

A következőkben ML kritériumot alkalmazunk, azaz a $P(O|\lambda)$ likelihood függvény maximumát keressük. Nem ismert olyan analitikus módszer, amellyel tetszőleges rögzített megfigyeléssorozatra kiszámíthatjuk az optimális modell paramétereit. Léteznek azonban lokálisan konvergens eljárások, mint a Baum-Welch algoritmus,

vagy a gradiens módszerek [28]. Itt a Baum-Welch algoritmust mutatjuk be részletesebben. (A statisztikából ismert Expectation Modification (EM) módszerrel ugyanezeket az egyenleteket kaphatjuk meg.)

Vezessük be a következő változót:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{4.16}$$

A korábban használt $\alpha_t(i)$ és $\beta_t(j)$ változókkal kifejezhetjük $\xi_t(i,j)$-t:

$$\begin{aligned}\xi_t(i,j) &= \frac{P(q_t = S_i, q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}\end{aligned}$$

A (4.12) pontban definiált $\gamma_t(i)$ is kifejezhető $\xi_t(i,j)$-vel:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{4.17}$$

Ha a $\gamma_t(i)$ értékeket t szerint összegezzük, akkor az O generálásakor bejárt állapotsorozatban az S_i előfordulásainak várható értékét kapjuk, vagy ami ezzel ekvivalens, az S_i állapotból induló állapotátmenetek előfordulásainak várható értékét. (Utóbbi esetben csak $(T-1)$-ig összegzünk.) Hasonlóan a $\xi_t(i,j)$-k $t = 1, \ldots, T-1$ szerinti összege az $S_i \to S_j$ állapotátmenetek számának várható értékével egyezik meg. Összefoglalva:

$$\sum_{t=1}^{T-1} \gamma_t(i) = S_i\text{-ből induló állapotátmenetek várható értéke}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = S_i \to S_j \text{ állapotátmenetek várható értéke}$$

Az RMM paramétereinek becslésére a következő iterációs eljárást használjuk:

$$\begin{aligned}
\bar{\pi}_i &= \text{az } S_i \text{ állapot előfordulásának várható értéke a } (t=1) \text{ pillanatban} \\
&= \gamma_1(i) \qquad (4.18) \\
\bar{a}_{ij} &= \frac{S_i \to S_j \text{ állapotátmenetek várható értéke}}{S_i\text{-ből induló állapotátmenetek várható értéke}} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad (4.19) \\
\bar{b}_j(k) &= \frac{\text{„}S_j \text{ állapotban } v_k\text{-t figyeltük meg” esetek előfordulásának várható értéke}}{\text{az } S_j \text{ állapot előfordulásának várható értéke}} \\
&= \frac{\sum_{\substack{1\le t\le T \\ O_t = v_k}} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \qquad (4.20)
\end{aligned}$$

Jelölje a modell paramétereit $\lambda = (A, B, \pi)$, a fenti iterációs egyenletekkel kapott új modell paramétereit $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. Ekkor bizonyítható [14], hogy az iterációk során λ a likelihood függvény lokális maximumához konvergál, azaz $P(O|\bar{\lambda}) > P(O|\lambda)$ vagy $\bar{\lambda} = \lambda$. Rögzített O megfigyeléssorozatból és véletlen λ paraméterű modellből kiindulva tehát a fenti egyenletek alkalmazásával az RMM tanítható az O megfigyeléssorozatra.

4.1.5. Kiterjesztés folytonos esetre

Eddig olyan RMM-eket vizsgáltunk, amelyekben a megfigyelések egy véges ábécé szimbólumai voltak. Most a megfigyeléseink folytonos értékűek lesznek.

A legáltalánosabb sűrűségfüggvény, amelyhez már adtak RMM tanító algoritmust [23][19][20] a következő alakú:

$$b_j(\mathbf{O}) = \sum_{m=1}^{M} c_{jm} \Re[\mathbf{O}; \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] \qquad 1 \le j \le N \qquad (4.21)$$

ahol $\mathbf{O}$ a modellezni kívánt vektor, c_{jm} a komponens eloszlásokat összekeverő együtthatók, a $\Re$ pedig valamilyen log-konkáv[2] [23] (pl. Gauss) sűrűségfüggvény $\boldsymbol{\mu}_{jm}$ várható értékkel és $\mathbf{U}_{jm}$ kovarianciamátrixszal. Természetesen igaz még, hogy

$$\begin{aligned}
\sum_{m=1}^{M} c_{jm} &= 1 && 1 \le j \le N \\
c_{jm} &\ge 0 && 1 \le j \le N \\
& && 1 \le m \le M
\end{aligned}$$

[2] Az $f(x)$ log-konkáv, ha $\ln(f(x))$ konkáv függvény.

A (4.21) alak segítségével tehát tetszőleges korlátos folytonos sűrűségfüggvényt M darab log-konkáv komponens konvex kombinációjával közelíthetjük. A gyakorlatban $\Re$ Gauss eloszlás sűrűségfüggvénye szokott lenni.

Bizonyítható [23][19][20], hogy a $c_{jk}, \boldsymbol{\mu}_{jk}, \mathbf{U}_{jk}$ paraméterek megfelelő iterációs formulái a következők:

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)} \tag{4.22}$$

$$\bar{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot \mathbf{O}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{4.23}$$

$$\bar{\mathbf{U}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (\mathbf{O}_t - \boldsymbol{\mu}_{jk})(\mathbf{O}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{4.24}$$

ahol a vessző transzponálást jelöl, a $\gamma_t(j,k)$ pedig annak valószínűségét, hogy a t pillanatban S_j állapotban vagyunk feltéve, hogy ismerjük az O megfigyelést és beleszámítva az O_t k-adik komponens szerinti valószínűségét, azaz

$$\begin{aligned}\gamma_t(j,k) &= \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}\Re(\mathbf{O}_t; \boldsymbol{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{m=1}^{M} c_{jm}\Re(\mathbf{O}_t; \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm})}\right] \\ &= \gamma_t(j)\left[\frac{c_{jk}\Re(\mathbf{O}_t; \boldsymbol{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{m=1}^{M} c_{jm}\Re(\mathbf{O}_t; \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm})}\right]\end{aligned}$$

A $\gamma_t(j,k)$ tehát a $\gamma_t(j)$ általánosabb, az összetett sűrűségfüggvény egy komponensére vonatkozó változata. Értékük $M = 1$ esetben egyezik meg, vagyis ha a keverék egyetlen komponensből áll.

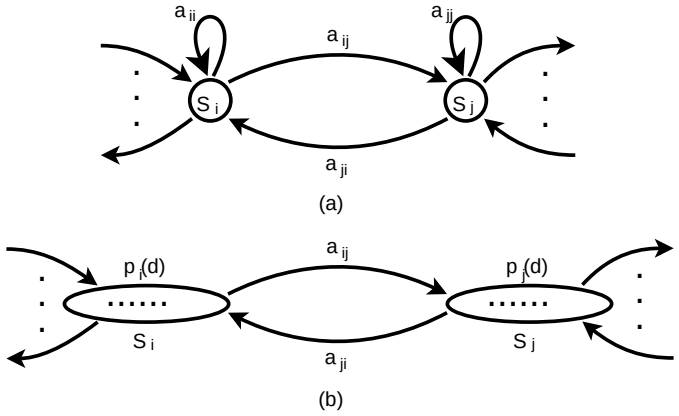
4.1.6. Explicit állapot-időtartam sűrűség megadása

A hagyományos RMM implicite határozza meg annak valószínűségét, hogy rögzített S_i állapotban maradunk pontosan d egymás utáni lépésen keresztül. Az állapot-időtartam sűrűség tehát

$$p_i(d) = (a_{ii})^{d-1}(1 - a_{ii}) \tag{4.25}$$

amely d függvényében exponenciálisan csökken. Ez a tulajdonság a legtöbb fizikai jel modellezésekor nem megfelelő, ezért explicit módon adjuk meg a $p_i(d)$ állapot-időtartam sűrűségeket [26][22]. Ilyen modellben az $S_i \to S_i$ átmeneteket nem engedjük meg, azaz $\forall i : a_{ii} = 0$. Csak akkor lépünk valamely következő állapotba,

ha már d számú megfigyelés történt a jelenlegi állapotban. A modellből tehát ugyanolyan módon gyárthatunk megfigyeléseket mint hagyományos esetben kivéve, hogy minden megfigyelés generálása előtt d időtartamot kell választanunk a $p_i(d)$ sűrűségnek megfelelően, és ennyiszer fogunk kimeneti szimbólumokat gyártani azok $b_i(O_1O_2\cdots O_d)$ együttes sűrűsége alapján. A gyakorlatban rögzített D-re $d \geq D$ esetén a D időtartamot használjuk. Általában a megfigyeléseink függetlenek, azaz $b_i(O_1O_2\cdots O_d) = \prod_{t=1}^{d} b_i(O_t)$. Látható, hogy az így megadott RMM általánosítása a hagyományosnak és a (4.25) teljesülése esetén a kétféle modell ekvivalens.



4.2. ábra. A hagyományos RMM és az explicit állapot-időtartamú RMM közötti különbség szemléltetése.
A hagyományos (a) esetben megengedjük az $S_i \to S_i$ típusú átmeneteket, míg az explicit állapot-időtartamú (b) esetben nem.

A következő szakaszban megadjuk az explicit állapot-időtartam sűrűséggel rendelkező RMM kiértékelési és tanítási iterációs formuláit. Tegyük fel, hogy az első állapot $t = 1$ pillanatban kezdődik és az utolsó állapot $t = T$ pillanatban végződik. Az $\alpha_t(j)$ Forward változót most a következőképpen definiáljuk:

$$\alpha_t(j) = P(O_1O_2\cdots O_t, S_j \text{ állapot } t \text{ pillanatban végződik}|\lambda) \tag{4.26}$$

Legyenek $q_1, q_2, \ldots, q_r = S_j$ az első t pillanatban bejárt állapotok $d_1, d_2, \ldots, d_r$ időtartamokkal, vagyis

$$\sum_{s=1}^{r} d_s = t$$

Ezek alapján a (4.26) felírható

$$\begin{aligned}\alpha_t(j) = &\sum_{q}\sum_{1\leq d\leq D} \pi_{q_1}\cdot p_{q_1}(d_1)\cdot P(O_1O_2\cdots O_{d_1}|q_1)\\ &\cdot a_{q_1q_2}p_{q_2}(d_2)\cdot P(O_{d_1+1}\cdots O_{d_1+d_2}|q_2)\cdots\\ &\cdot a_{q_{r-1}q_r}p_{q_r}(d_r)\cdot P(O_{d_1+d_2+\cdots+d_{r-1}+1}\cdots O_t|q_r)\end{aligned} \tag{4.27}$$

alakban, ahol az összes lehetséges állapot és időtartam szerint összegeztünk. Ismét megadhatunk egy rekurziót $\alpha_t(j)$ kiszámítására.

$$\alpha_t(j) = \sum_{i=1}^{N} \sum_{d=1}^{D} \alpha_{t-d}(i) a_{ij} p_j(d) \prod_{s=t-d+1}^{t} b_j(O_s) \tag{4.28}$$

ahol D jelöli a legnagyobb állapot-időtartamot az összes állapotra vonatkozóan. A rekurzió inicializálása:

$$\alpha_1(j) = \pi_j p_j(1) \cdot b_j(O_1) \tag{4.29}$$

$$\alpha_2(j) = \pi_j p_j(2) \prod_{s=1}^{2} b_j(O_s) + \sum_{\substack{i=1 \\ i \neq j}}^{N} \alpha_1(i) a_{ij} p_j(1) b_j(O_2) \tag{4.30}$$

$$\alpha_3(j) = \pi_j p_j(3) \prod_{s=1}^{3} b_j(O_s) + \sum_{d=1}^{2} \sum_{\substack{i=1 \\ i \neq j}}^{N} \alpha_{3-d}(i) a_{ij} p_j(d) \prod_{s=4-d}^{3} b_j(O_s) \tag{4.31}$$

s. í. t., amíg $\alpha_D(j)$ értékét megkapjuk; ezután már használhatjuk a (4.28) formulát. A kiértékelési feladat megoldása tehát

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j) \tag{4.32}$$

A tanítási formulákhoz még három változót kell definiálnunk:

$$\alpha_t^*(j) = P(O_1 O_2 \cdots O_t, S_j \text{ állapot } t+1 \text{ pillanatban kezdődik}|\lambda) \tag{4.33}$$

$$\beta_t(j) = P(O_{t+1} \cdots O_T | S_j \text{ állapot } t \text{ pillanatban végződik}, \lambda) \tag{4.34}$$

$$\beta_t^*(j) = P(O_{t+1} \cdots O_T | S_j \text{ állapot } t+1 \text{ pillanatban kezdődik}, \lambda) \tag{4.35}$$

Az α, α^*, β, β^* változók közötti összefüggések:

$$\alpha_t^*(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} \tag{4.36}$$

$$\alpha_t(j) = \sum_{d=1}^{D} \alpha_{t-d}^*(j) p_j(d) \prod_{s=t-d+1}^{t} b_j(O_s) \tag{4.37}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \beta_t^*(j) \tag{4.38}$$

$$\beta_t^*(i) = \sum_{d=1}^{D} \beta_{t+d}(i) p_i(d) \prod_{s=t+1}^{t+d} b_i(O_s) \tag{4.39}$$

A tanítási formulák a változókat és a korábbi definíciókat felhasználva:

$$\bar{\pi}_i = \frac{\pi_i \beta_0^*(i)}{P(O|\lambda)} \tag{4.40}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T} \alpha_t(i) a_{ij} \beta_t^*(j)}{\sum_{j=1}^{N} \sum_{t=1}^{T} \alpha_t(i) a_{ij} \beta_t^*(j)} \tag{4.41}$$

$$\bar{b}_i(k) = \frac{\sum\limits_{\substack{1 \le t \le T \\ O_t = v_k}} \Big[\sum_{\tau<t} \alpha_\tau^*(i)\beta_\tau^*(i) - \sum_{\tau<t} \alpha_\tau(i)\beta_\tau(i) \Big]}{\sum_{l=1}^{M} \sum\limits_{\substack{1 \le t \le T \\ O_t = v_l}} \Big[\sum_{\tau<t} \alpha_\tau^*(i)\beta_\tau^*(i) - \sum_{\tau<t} \alpha_\tau(i)\beta_\tau(i) \Big]} \tag{4.42}$$

$$\bar{p}_i(d) = \frac{\sum_{t=1}^{T} \alpha_t^*(i) p_i(d) \beta_{t+d}(i) \prod_{s=t+1}^{t+d} b_i(O_s)}{\sum_{d=1}^{D} \sum_{t=1}^{T} \alpha_t^*(i) p_i(d) \beta_{t+d}(i) \prod_{s=t+1}^{t+d} b_i(O_s)} \tag{4.43}$$

A (4.40) formulában $\bar{\pi}_i$ értéke rögzített O mellett annak valószínűsége, hogy S_i volt az első állapot. A (4.41) formula hasonló mint a hagyományos esetben - ott a (4.19) formulát kell kifejteni α-ra és β-ra. A (4.42) formula jobb oldala annak várható értéke, hogy az S_i állapotban $O_t = v_k$ megfigyelés történt (diszkrét esetet feltételezve) az S_i állapotban bekövetkező bármely megfigyelések várható értékével normalizálva. Végül a (4.43) annak várható értéke hogy az S_i állapot d lépésig tart normalizálva ugyanezen várható értékek tetszőleges $d = 1, \ldots, D$ állapot-időtartam szerinti összegével.

A tapasztalat azt mutatja, hogy az explicit állapot-időtartam sűrűségfüggvénnyel rendelkező RMM sokkal kifinomultabb modellezésre képes. Azonban az ilyen RMM tanítása - az $\alpha_t(j)$ (4.29)-(4.31) formulái alapján - körülbelül D-szeres tárigényű és $D^2/2$-szeres számításigényű a hagyományos RMM tanításhoz képest. Ezenfelül több paramétert kell hangolni és - tekintve, hogy átlagosan kevesebb az állapot-átmenetek száma, mint a megfigyelések száma - kevesebb adat áll rendelkezésre a $p_i(d)$ paraméterek közelítéséhez. Ezért a tanítási feladat a gyakorlatban sokkal nehezebb mint a hagyományos RMM esetében.

5. fejezet

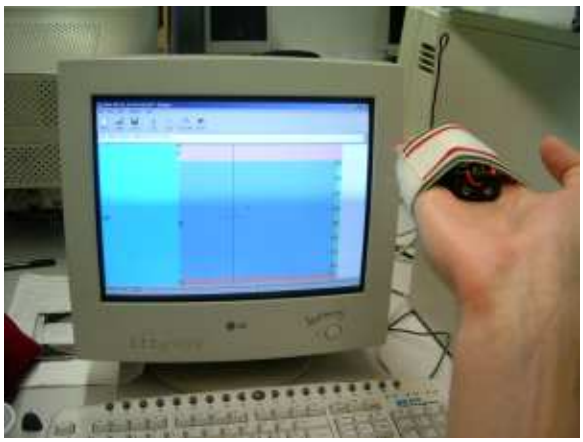
Elemzések

Ebben a fejezetben a következő kérdésekre keressük a választ:

- Rejtett Markov Modellek segítségével milyen információkat nyerhetünk a felhasználó viselkedéséről a Dasher használatakor?
- Meg lehet-e tanulni az RF-egér használatát, vagyis beviteli eszközként alkalmazható-e?

Először ismertetjük a kísérletet (5.1. alfejezet), majd bemutatunk egy lehetséges intuitív interpretációt a kísérlet méréseire illesztett RMM belső állapotaira (5.1.1. alfejezet), és megvizsgáljuk hogyan alkalmazható ez az interpretáció az RF-egér esetében (5.1.2. és 5.1.3. alfejezet).

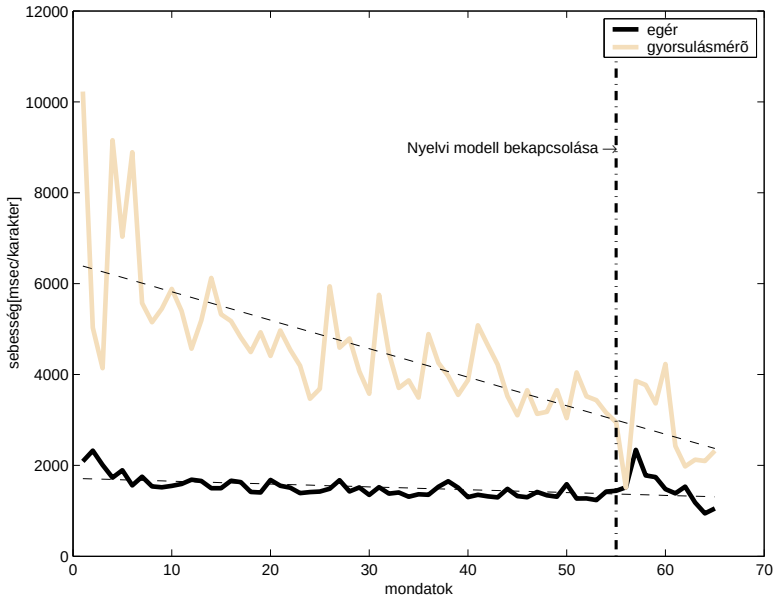
5.1. Szövegbevitel RF-egérrel



5.1. ábra. Szövegbevitel RF-egérrel.

E kísérlet során egyetlen - a Dasher és az RF-egér használatában gyakorlatlan - felhasználónak kellett véletlenszerűen választott angol nyelvű mondatokat gépelnie a Dasher-rel. A mondatok dalszövegek sorai voltak pl. „children need travelling shoes". A felhasználó a kísérlet során 13 alkalommal gépelt, alkalmanként 5-5 mondatot

hagyományos és RF-egérrel (5.1. ábra). A kísérlet során a gépelés sebessége RF-egérrel fokozatosan nőtt. Az utolsó két alkalommal bekapcsolt nyelvi modell - kisebb visszaesés után - még tovább javított ezen a sebességen (5.2. ábra).



5.2. ábra. Gépelési sebesség fejlődése Dasherben.
Az ábrán a karakterenkénti átlagos gépelési sebesség látható a kísérletek alatt gépelt mondatok függvényében. Hagyományos egér esetén nem változott jelentősen a sebesség, míg az RF-egérnél látványos a javulás.

A gépelés során lementettük az egérkurzor trajektóriáját 50 Hz-es mintavételi frekvenciával. Az 1. és 10. alkalom sebességvektoraival RMM-eket tanítottunk. Az RMM-ek állapotaiban normális eloszlású kimeneteket feltételeztünk, a tanítás Baum-Welch algoritmussal történt (lásd a 4.1.4. fejezetet). Mivel az RF-egér használatára jellemző a függőleges ill. vízszintes irányú mozgások éles szétválása, ezért az RMM-ekben is feltettük, hogy a sebességvektorok x-y koordinátái függetlenek egymástól. Így a kimeneti eloszlások kovarianciamátrixai diagonálisak, az (5.4)-(5.5) ábrákon látható ellipszisek tengelyei a főtengelyekkel párhuzamosak lesznek. A belső állapotok számára az $N = 3, 5, 8$ értékeket próbáltunk, az 5 állapotú RMM-eket találtuk a legkönnyebben értelmezhetőnek.

5.1.1. A belső állapotok interpretációja

A Fejegérrel végzett kísérletek alapján [17] tudjuk, hogy a Dasher használata közben az 5 RMM állapot jellegzetesen helyezkedik el: Mindig találunk egy központi, kis szórású állapotot (1), ettől felfele (2), lefele (3), jobbra (4) elhelyezkedő állapotokat, illetve egy olyan állapotot, amelynek szórása az összes többinél nagyobb (5). A Dasher működése alapján ezeket az állapotokat intuitívan a következőképpen interpretálhatjuk:

1. „jó irányba haladok”

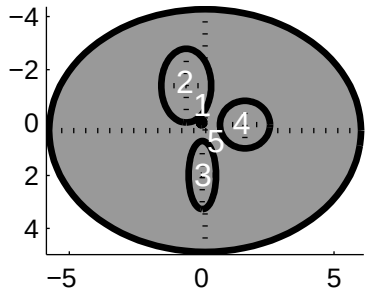
2. „ábécében előbbre levő betű kell”

3. „ábécében hátrébb levő betű kell”

4. „gyorsítok”

5. „hibáztam, korrigálnom kell”

Az 1. vagy *ok* állapotban nem szükséges beavatkozni mert a Dasher a megfelelő betű felé halad. A 2., 3., 4. állapotokban a megfelelő betű felé kell vezetnünk a gépelést - ezek a *kontroll* állapotok. Végül az 5. vagy *hiba* állapotba tartozik az összes többi mozgás, így a visszafele haladás és a túlkorrigálásból adódó hirtelen mozgások is (5.3. ábra). A következő diszkusszióból kiderül, hogy ez az egyszerű interpretáció - kisebb módosításokkal - átvihető az RF-egeres elemzésekre is.



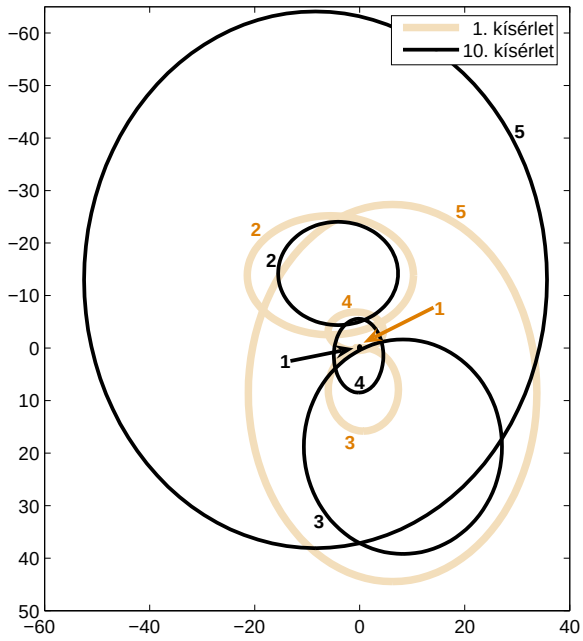
5.3. ábra. Az RMM állapotainak intuitív azonosítása.
(1) „jó irányba haladok”, (2) „ábécében előbbre levő betű kell”, (3) „ábécében hátrébb levő betű kell”, (4) „gyorsítok”, (5) „hibáztam, korrigálnom kell”

5.1.2. Diszkusszió

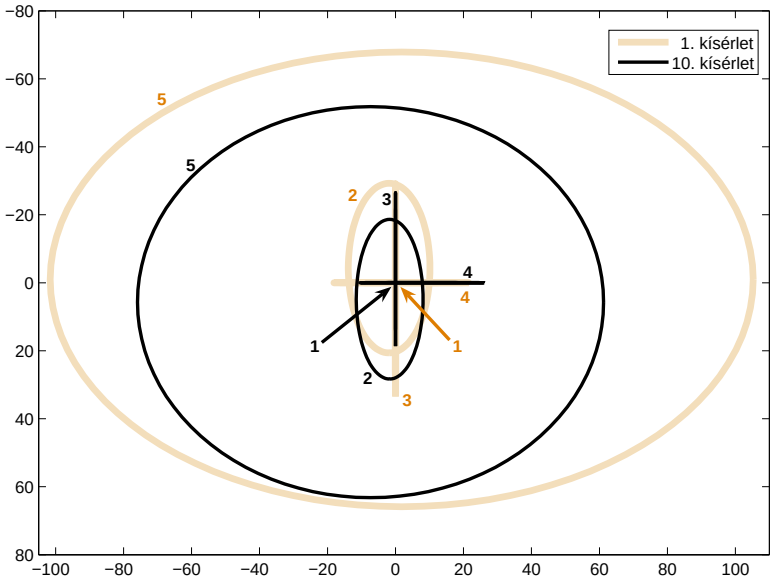
Az (5.4) ábrán a hagyományos egérrel végzett 1. és 10. kísérlet RMM-jei láthatók. Az ok (1), a hiba (5), és a függőleges kontroll állapotok (2,3) könnyen felismerhetőek. A gyorsítás állapotát a 4. állapot helyett a 2.,3.,5. állapotok rejtve tartalmazzák. A 10. kísérletben látható jóval nagyobb hibatartomány oka, hogy a felhasználó gyorsabb mozgásokat végzett hibázáskor. Ezzel együtt kevesebbet hibázott, mint az az állapotátmenet valószínűségek elemzésénél kiderül majd.

Az RF-egérrel kapott állapotok a következőképpen interpretálhatók (5.5. ábra): ok állapot (1), ferde kontroll (2), függőleges (3), vízszintes (4) kontroll, hibaállapot (5). A 10. kísérlet hibaállapota itt láthatóan kisebb, mint az 1. kísérletnél. Mindkét RMM-en feltűnőek az RF-egér használatára jellemző, tisztán függőleges v. vízszintes mozgások (elnyúlt ellipszisek), a ferde irányú elmozdulások minimálisak. (Ezt bizonyítja az 5.8. ábra is.) Elmondható még, hogy a 10. kísérletkor ügyesebb volt a felhasználó: A vízszintes ellipszis közepe jobbra tolódott, azaz több volt a gyorsítás, mint a lassítás. (Az 1. kísérletben a kettő nagyjából egyformán gyakori.)

Mivel a sebességtartományok egyik egértípusnál sem diszjunktak, ezért az RMM-ekkel nem tudjuk egyértelműen megmondani, hogy adott pillanatban a felhasználó mely állapotban tartózkodik.



5.4. ábra. A sebességvektorok Rejtett Markov Modelljei hagyományos egérnél. A „gyorsítok” állapot a 2,3,5 RMM állapotokban rejtve található.



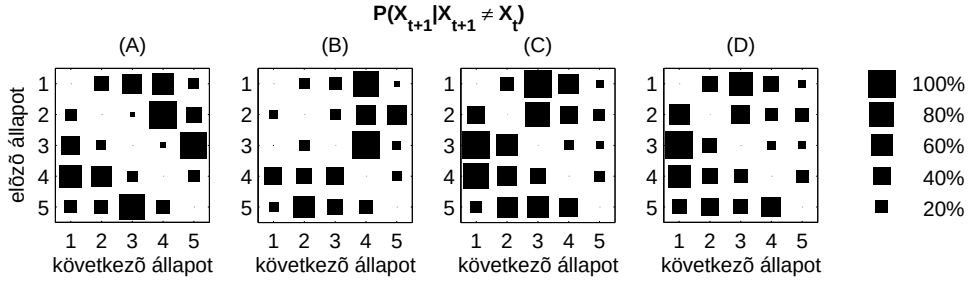
5.5. ábra. A sebességvektorok Rejtett Markov Modelljei RF-egérnél. Megfigyelhetők az RF-egér használatára jellemző csak függőleges (3) / vízszintes (4) mozgások.

5.1.3. Állapotátmenetek az RMM-ekben

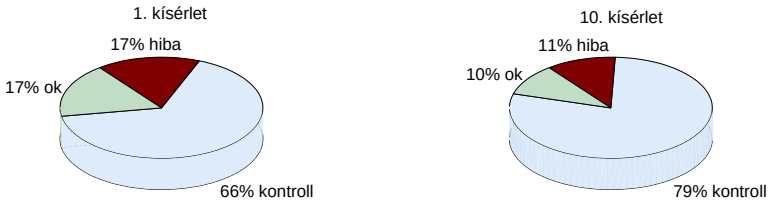
Az intuitívan kijelölt állapotok közötti átmenetek valószínűségét mutatják az 5.6. ábra diagramjai. Az 1. oszlopokon látszik, hogy az RF-egérrel sokkal kevesebbet mozgott a felhasználó mint a hagyományos egérrel, nagyobb valószínűséggel jutunk ok (1) állapotba (ilyenkor nincs mozgás), mint kontroll (2,3,4) állapotokba.

A módosított Dasher leírásánál már említettük, hogy az RF-egér használatakor jellemző a túlkorrigálás, nehéz az eszközt stabilan vezetni. A 10. kísérlet alkalmával a kontroll állapotba kerülés valószínűsége csökkent, míg a hibázásé nem változott. A felhasználó ezek szerint kevesebbszer avatkozott közbe az utolsó kísérletekkor.

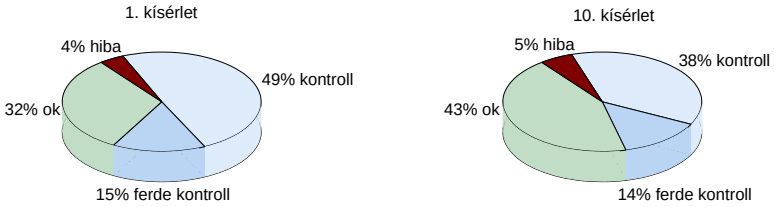
A hagyományos egér esetében épp az ellenkező mondható el: A kontroll állapotok valószínűsége nőtt, miközben az ok és a hiba állapotoké csökkent. A gyakorlás ez esetben tehát több, pontosabb mozgást eredményezett.



5.6. ábra. Állapotátmenet valószínűségek az RMM-ekben.
Az intuitíven megjelölt RMM állapotok (1 - ok; 2,3,4 - kontroll; 5 - hiba) közötti átmenetek valószínűsége látható a diagramokon. Az (A)-(B) diagramok a hagyományos egérrel végzett 1. és 10. kísérlet eredményei alapján készültek, a (C)-(D) diagramok pedig az RF-egérrel végzett 1. és 10. kísérletek alapján.



5.7. ábra. Az egyes állapotokban töltött idő eloszlása hagyományos egérnél.
A kísérleti alany a 10. alkalommal már láthatóan kevesebbet hibázott, és több időt töltött a kontroll állapotokban.



5.8. ábra. Az egyes állapotokban töltött idő eloszlása RF-egérnél.
Látható, hogy óvatosabbá vált a kísérleti alany: Kevesebbet mozgott a 10. alkalommal. A tisztán vízszintes, függőleges irányú mozgások végig jellemzőek maradtak, a ferde irányú mozdulatok ritkábbak. A hibaállapotban töltött idő nem változott jelentősen.

6. fejezet

Összefoglalás

A dolgozatban bemutattuk a mozgássérült gyermekek által használt beviteli eszközöket és alkalmazásokat. Az alkalmazások használatakor mért adatok birtokában elemezhetjük az interakciót. Ezután a dolgozat két fő irányra bomlott: Az egyikben részletesen bemutattuk az RF-egér szoftvert, amely egyetlen testre rögzített szenzor segítségével beviteli eszközként használható. A másikban pedig egy RF-egérrel végzett kísérlet adatait elemeztük Rejtett Markov Modellek segítségével.

Az elemzések során megmutattuk, hogy az RF-egér használható interakciós eszköz. Ezt a kísérleti alany gépelési sebességének javulásával igazoltuk. RMM-ek segítségével információt nyertünk az RF-egér használatakor megfigyelhető jellemző mozgásmintákról: Ezek szinte kizárólag függőleges/vízszintes irányú mozgások voltak. Megállapítottuk azt is, hogy az RF-egér kezelésénél fellépő túlkorrigálás kiküszöbölhető gyakorlással.

Kitekintésként megjegyezzük, hogy speciális, ember-számítógép interakcióra használható hardver- és szoftvereszközök fejlesztése jelenleg is intenzíven folyik, egyrészt mert a számítógépek egyre nagyobb kapacitása mellett az interakcióban már a felhasználó sebessége a szűk keresztmetszet, másrészt mert a felhasználóknak létezik egy köre, akik csak ilyen eszközökkel képesek boldogulni. Ezen túlmenően a környezeti intelligencia adaptív eszközeinek kutatása a szórakoztatóipar számára is fontos területet jelent.

Ábrák jegyzéke

Irodalomjegyzék

[1] *Allaboutmems*, http://www.allaboutmems.com/index.html.

[2] *The beginner's guide to mems processing*, http://www.memsnet.org/mems/processes/.

[3] *Biomems research group @ ucirvine*, http://mmadou.eng.uci.edu/.

[4] *The cricket indoor location system: An nms project @ mit csail*, http://cricket.csail.mit.edu/.

[5] *Crossbow technology - wireless sensor networks*, http://xbow.com/.

[6] *Industrial solutions - crossbow, smarter sensors in silicon*, http://www.xbow.com/Industry_solutions/industry.htm.

[7] *Java native interface*, http://java.sun.com/j2se/1.5.0/docs/guide/jni/index.html.

[8] *Tinyos home*, http://www.tinyos.net/.

[9] *Tinyos tutorial*, http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html.

[10] *The viterbi algorithm*, http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/viterbi_algorithm/s1_pg1.html.

[11] *The viterbi algorithm for isolated word recognition*, http://labrosa.ee.columbia.edu/doc/HTKBook21/node8.html.

[12] Gerőfi Balázs, *Hangegér - hang és beszédfelismerés*, Diplomamunka, ELTE IK Információs Rendszerek Tanszék, 2005.

[13] ———, *Intelligens gépi asszisztens fejlesztés. viselkedési minták kialakulása és felismerése hang-interakciós alkalmazásban.*, TDK dolgozat, ELTE IK Információs Rendszerek Tanszék, 2004.

[14] L. E. Baum and G. R. Sell, *Growth functions for transformations on manifolds*, Pac. J. Math., vol. 27, 1968, pp. 211–227.

[15] Rob von Behren Matt Welsh Eric Brewer David Gay, Phil Levis and David Culler, *The nesc language: A holistic approach to networked embedded systems*, Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003, http://www.tinyos.net/papers/nesc.pdf.

[16] David Gay, Philip Levis, David Culler, and Eric Brewer, *nesc 1.1 language reference manual*, http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf.

[17] Gy. Hévízi, M. Biczó, B. Póczos, Z. Szabó, B. Takács, and A. Lőrincz, *Hidden markov model finds behavioral patterns of users working with a headmouse driven writing tool.*

[18] Peer Itsik, *Viterbi algorithm*, http://www.math.tau.ac.il/~rshamir/algmb/00/scribe00/html/lec06/node4.html.

[19] B. H. Juang, *Maximum likelihood estimation for mixture multivariate stochastic observations of markov chains*, AT&T Tech. J., vol. 64, July-Aug. 1985, pp. 1235–1249.

[20] B. H. Juang, S. E. Levinson, and M. M. Sondhi, *Maximum likelihood estimation for multivariate mixture observations of markov chains*, IEEE Trans. Informat. Theory, vol. IT-32, Mar. 1986, pp. 307–309.

[21] Jean-Christophe Lementec and Peter Bajcsy, *Recognition of arm gestures using multiple orientation sensors: Gesture classification*, http://algdocs.ncsa.uiuc.edu/PB-20041003-1.pdf.

[22] S. E. Levinson, *Continuously variable duration hidden markov models for automatic speech recognition*, Computer, Speech and Language, vol. 1, March 1986, pp. 29–45.

[23] L. A. Liporace, *Maximum likelihood estimation for multivariate observations of markov sources*, IEEE Trans. Informat. Theory, vol. IT-28, 1982, pp. 729–734.

[24] A. Lőrincz, *Embedded body sensor network for persons with special communication needs to control and to interact with the world*, International workshop on wearable and implantable body sensor networks, 2005. http://ubimon.doc.ic.ac.uk/bsn/index.php?m=197.

[25] L. R. Rabiner, *A tutorial on Hidden Markov Models and selected applications in speech recognition. Proceedings of the IEEE, 77(2):257–286, 1989*, http://www.cs.ubc.ca/~murphyk/Software/HMM/rabiner.pdf.

[26] M. J. Russel and R. K. Moore, *Explicit modeling of state occupancy in hidden markov models for automatic speech recognition*, ICASSP'85 (Tampa, FL), March 1985, pp. 5–8.

[27] Martin Urban, Peter Bajcsy, Rob Kooper, and Jean-Christophe Lementec, *Recognition of arm gestures using multiple orientation sensors: Repeatability assessment*, http://algdocs.ncsa.uiuc.edu/PB-20041003-2.pdf.

[28] Narada Warakagoda, *Ml training*, http://jedlik.phy.bme.hu/~gerjanos/HMM/node24.html.

[29] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S. J. Pister, *Smart dust: Communicating with a cubic-millimeter computer*, Computer **34** (2001), no. 1, 44–51.