Debreceni Egyetem

Informatikai Kar

Wii Remote alkalmazása PC-s környezetben

Témavezető:

Dr. Herendi Tamás
egyetemi adjunktus

Készítette:

Bereczki Tamás
Programtervező informatikus (Msc) szak

Debrecen
2010

TARTALOMJEGYZÉK

BEVEZETÉS

2006 novemberét írjuk. A Nintendo piacra dobja a Revolution (forradalom) kódnéven fejlesztett új játékkonzolját, a Wii –t[1]. De miért is olyan „forradalmi" ez a rendszer akkor, amikor a konkurencia ezt - technikai paraméterekben - jócskán felülmúló fejlesztéseket reprezentál?

A válasz, az újszerű játékélmény. Itt most nem konkrétan a szoftverre gondolok, hanem egy eszközre, ami a Wii mint rendszer szoros része ez pedig a Wii Remote (vagy Wiimote).

A Wiimote egy egykezes vezérlőegység, ami egy 3D-s gyorsulásmérővel és infravörös érzékelővel lett felszerelve. Ez a funkció lehetővé teszi a játékos fizikai mozgásának figyelembevételét a játék közben.

2008 októberében egy blogban olvastam először egy saját készítésű infravörös tollal működő eszközről, mellyel bármilyen felület (monitor, vetített kép) interaktívvá tehető.

A bejegyzésben egy amerikai fiatalember (Johnny Chung Lee) aki ma, a Microsoft hardverfejlesztési részlegén dolgozik kutatóként, egy videón a Wii remote -ot használva, saját szoftverrel és IR tollal olcsón megvalósítható interaktív táblát mutatott be. Nem sokkal később oldalhoz tartozó fórumba a világ minden tájáról rengeteg elismerő írás érkezett, bemutatva az ötlet egyéni megvalósításait.

Rögtön megragadott az elképzelés, de mint mérnök informatikus engem a dolog működési elve érdekelt igazán, így arra gondoltam, hogy ez a dolog mindenképpen megér egy szakdolgozati témát.

A dolgozatban első fejezetében, kitérek a rendszer hardveres felépítésére, ill. ezen elemeknek a kontrolleren betöltött funkcióira.

Szó lesz a gyakorlati megvalósításhoz szükséges matematikai háttérről, azok helyéről az alkalmazáslogikában.

A diplomamunka részét képezi – demonstrációs célból – egy olyan tesztelő alkalmazás, amely az eszköz képességeit hivatott reprezentálni, és amelynek bemutatásán keresztül megismerhetjük a

1 A Wii, mint szó kettő darab kis „ i " betűvel írandó, amely két egymás mellett álló embert szimbolizál, akik éppen a konzollal játszanak.

kontroller programozási lehetőségeit.

Kidolgozásra kerül egy, az eszköz konkrét gyakorlati felhasználását megvalósító alkalmazás is. Ennek célja, hogy a Wii remote infravörös kamerájának segítségével nyomonkövesse egy adott fényforrás mozgását, a kapott adatokat feldolgozza, és azokból meghatározza a felületre vetített tényleges helyzet koordinátáit, hogy aztán a számítógép monitorán az egérkurzor egy infravörös leddel ellátott mutatóeszközzel közvetlenül vezérelhető legyen.

A program megalkotásának keretében megállapításra kerül, hogy a feldolgozott működési folyamat igen egyszerű illetve, hogy maga az eszköz milyen sok lehetőséget rejt magában.

A tárgyalási rész zárásaként ábrák, leírások segítségével mutatom be a program elvi működését, tanulmányozom működésének fontos részeit.

1. HARDVER

A Wii remote Bluetooth kapcsolaton keresztül kommunikál a hozzá társított berendezésekkel. A fejezet első részében szó lesz röviden a Bluetooth szabványról [1], a felhasznált Bluetooth profilokról (HID), ill. a Wiimote és az adapter közötti kommunikációs protokollokról (RFCOMM és L2PAC).

A második fejezetrész bemutatja a Wii távirányító felépítését kiemelve az adatkommunikációs, bementi, kimeneti, és memória interfészek működési mechanizmusát.

1.1. Bluetooth szabvány

A Bluetooth[2] nevet adták annak a technológiának, amelyet hordozható vagy fix irodai eszközök rövid távolságú rádiós összeköttetésére lehet használni.

Az eredetileg Ericsson fejlesztésű technológiát más gyártók is elfogadták, így a kicsi és olcsó beépített Bluetooth chipek segítségével számítógépek, nyomtatók, mobiltelefonok és más eszközök közötti vezeték nélküli kommunikációra nyílt lehetőség.

A chipet arra fejlesztették ki, hogy az eddigi vezetékes összeköttetések helyett, az információt egy speciális rádiós csatornán juttassa el a számítógépbe, vagy éppen a mobiltelefonba. Olyan rádiós kapcsolat megteremtése volt a cél, amely a bonyolult kábelezést egy univerzális, minden eszköz számára elérhető átviteli rendszerrel helyettesíti.

1.1.1. Rádiós átviteli jellemzők

A Bluetooth technológia egy széles spektrumú, teljes-duplex, frekvenciaugráltatásos átvitellel működik, 1600 ugrással másodpercenként. A rendszer a 2.402GHz és 2.480GHz közötti tartományban 79 vivőfrekvenciát használ, 1 MHz-es csatornaosztással. Néhány országban (például Franciaországban) ezt a frekvenciasávot szűkítették, és 23 vivős rendszert használnak [2].

2 A Bluetooth elnevezés eredetileg egy a X. század második felében élt dán Viking királytól származik, akinek a neve Kékfog volt. Harald király egyesítette és kormányozta Dánia és Norvégia országait, és ez ihlette a szabvány megalkotóit a technológia nevének kitalálásakor (eszközök együttműködése a Bluetooth-on keresztül).

Egy hálózatban, egy időben egy „mester” eszközhöz legfeljebb hét másik eszköz csatlakozhat. Az egymáshoz csatlakozott eszközök ún. Personal Area Network-öt (PAN), más szóval piconetet hoznak létre, ami például az egy szobában lévő eszközök által alkotott hálózatot jelenti. A Bluetooth készülékek teljesítmény-osztályuktól függően 1, 10, illetve 100 méteres távolságon belül képesek kommunikálni.

1.1.2. Profilok

A Bluetooth profil egy olyan része a szoftvernek, mely meghatározza, hogy adott kapcsolás kiépítése során az egyes eszközöknek milyen feladatokat látnak el, és leírja, hogy egy adott feladatot hogyan kell, illetve hogyan lehet megoldani. Például ha számítógép közli, a hozzá csatlakoztatni kívánt eszközzel, hogy kapcsolatba lépne vele, akkor a stabil kapcsolat felépítésének lépéseit egy meghatározott profil írja le. A Wiimote a HID profilt használja:

- ***HID – Human Interface Device Profile:***

  A Human Interface Device profil definiálja azokat a protokollokat, eljárásokat amik a Bluetooth Human Interface eszközök, úgymint billentyűzet, mutatóeszközök, játék kiegészítők, távoli monitorozó eszközök használatához kellenek. A specifikáció általában szorosan együttműködik az USB (Universal Serial Bus) protokollokkal.

1.1.3. Az eszközök párosítása

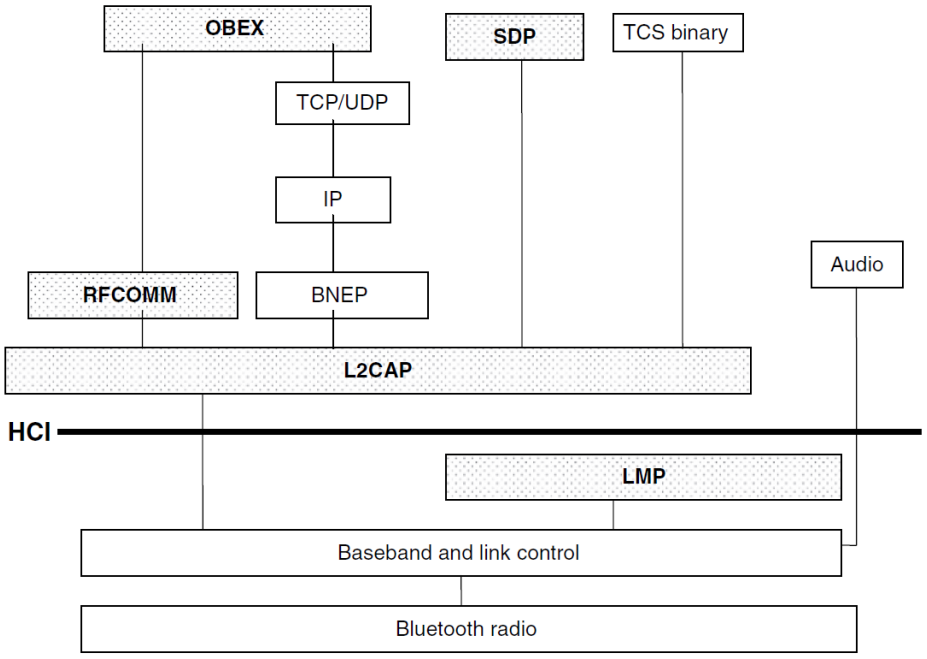
Alapértelmezés szerint a Bluetooth kommunikáció nem hitelesítődik és bármelyik eszköz képes bármelyik másikkal felvenni a kapcsolatot. Egy Bluetooth eszköz egy adott szolgáltatáshoz igényelhet hitelesítést (például betárcsázáshoz). A Bluetooth alapú hitelesítés többnyire PIN kódokkal történik.

A PIN kód egy legfeljebb 16 karakterből álló ASCII karakterlánc. A felhasználóknak mind a két eszközön ugyanazt a PIN kódot kell megadniuk. Miután megadtuk a PIN kódot, az eszközök létrehoznak hozzájuk egy összekötettésbeli kulcsot (link key). Ezt az eljárást nevezik párosításnak (pairing). Ha valamelyik eszköz elveszti az össszeköttetés kulcsát, akkor a párosítást meg kell ismételni.

A HID eszközök általában PIN nélkül párosíthatóak így a Wii remote is.

1.1.4. A Bluetooth protokollok

A protokoll stack tulajdonképpen a Bluetooth belső magját alkotja. Lehetővé teszi, hogy különböző eszközök megtalálják egymást, vagy kapcsolatot létesítsenek egymással. Ezen a kapcsolaton keresztül a kapcsolatot alkotó felek különböző típusú adatokat küldhetnek egymásnak. Az alábbi ábra a Bluetooth stack-en belüli támogatott szinteket mutatja:



1. ábra: Bluetooth Protokoll felépítése (Bluetooth Protocoll Stack)

Fontosabb szintek:

- A ***Bluetooth radio*** (réteg) a specifikáció legalsó rétege. Ez definiálja az adó-vevő működéséhez szükséges paramétereket a 2.4-GHz ISM frekvenciasávban.
- ***A Baseband*** (and link control) réteg alapvető eljárásokat definiál a Bluetooth eszközök egymás közötti kommunikációjának megvalósításához.
- A ***Link Manager Protocol (LMP)*** felelős a kapcsolat beállításáért és kialakításáért a Bluetooth eszközök között, kezeli és továbbítja a Baseband rétegből származó csomagokat. Továbbá ez a réteg felelős a biztonságért. Elvégzi az azonosítást és a titkosítást.

- A **Host Controller Interface (HCI)** egy parancsfelületet nyújt a működési sáv vezérlőjéhez (baseband controller) és az összeköttetések kezelőjéhez (link manager), valamint hozzáférést a hardver állapot és vezérlő regiszterekhez.
- A ***Logical Link Control and Adaptation Protocol (L2CAP)*** a Bluetooth Protocol Stack adatkapcsolati rétege. Összefogja az alsóbb rétegek által létrehozott különböző logikai kapcsolatokat és továbbítja a felsőbb rétegek felé. Az L2CAP az alapsávi protokoll (Baseband Protocol) felett helyezkedik el. A felsőbb rétegek számára nyújt kapcsolat-orientált, vagy kapcsolat nélküli összeköttetést.
- Az ***RFCOMM*** protokoll végzi a soros portok emulációját az L2CAP-n keresztül. Egy eszköznek lehet több konkurens kapcsolata, és több eszköz kapcsolódhat össze egyszerre.

1.1.5. A JAVA és a Bluetooth - JSR-82

A JSR-82 egy szabványos programozási felülete a Java programozási nyelvnek. A Java programok számára hozzáférhetővé teszi a Bluetooth-t, valamint az objektumok cseréjét elsősorban a kicsi, hordozható eszközökön. Mivel a JSR-82 Javat használ, a JSR-82 alkalmazások letölthetőek és futtathatóak bármely olyan platformon vagy eszközön, amely támogatja ezt a szabványt, de többnyire inkább kis teljesítményű mobil eszközökön.

1.2. Wii remote controller

Sajnos a Wii remote hivatalos specifikációja nem publikus. A technikai információk nagy része a GHC (Global Hacking Community) által lett összegyűjtve egy úgynevezett reverse-engineering[3] módszerrel. A gyűjtés eredménye olyan online wikipédiákban található meg mint a http://wiili.org és http://wiibrew.org. Nyilván az alacsonyabb szintű részletek a fent említett ok miatt még pontosításra szorulnak, de a tudástár folyamatosan bővül.

A következőkben a Wiimote komponensek hardverszintű részleteivel foglalkozom, azokon belül is csak a legszükségesebbekkel.

1.2.1. A Wii remote anatómiája

Összefoglalva, táblázatszerűen [3]:

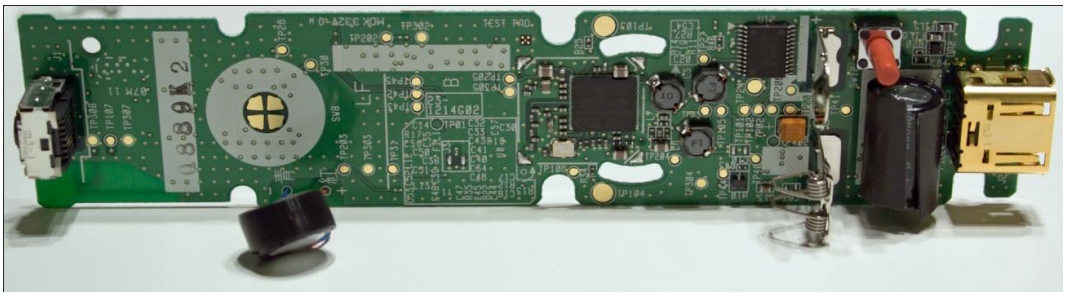
- Kommunikáció
  - Broadcom 2042 Bluetooth vezérlő chip
  - Működés Bluetooth HID eszközként
- Bementek (Inputs)
  - Gombok
  - Mozgásszenzor (gyorsulásmérő)
  - Infravörös szenzor
- Kimenetek (Outputs)
  - 4 db LED
  - Zümi (rezgőmotor)
  - Csipogó (speaker)
- Memória
  - Flash memória + Vezérlő regiszterek
- Egyebek
  - Bővítőport
  - Elemek

2. ábra: Wii remote controller

3 Egy adott architektúra, működési elv, vagy algoritmus elemzése és dokumentálása az eredeti tervek ismerete nélkül, pusztán a működés és a tárgykód analizálásán keresztül.

1.2.2. Kommunikáció

Amint azt korábban már leírtam a kommunikáció Bluetooth kapcsolaton keresztül történik. Ezt a feladatot egy Broadcom BCM2042 bluetooth SOC (System-on-a-chip) típusú mikrovezérlő látja el. Maga a kommunikáció illeszkedik a Bluetooth Human Interface eszközszabványra.

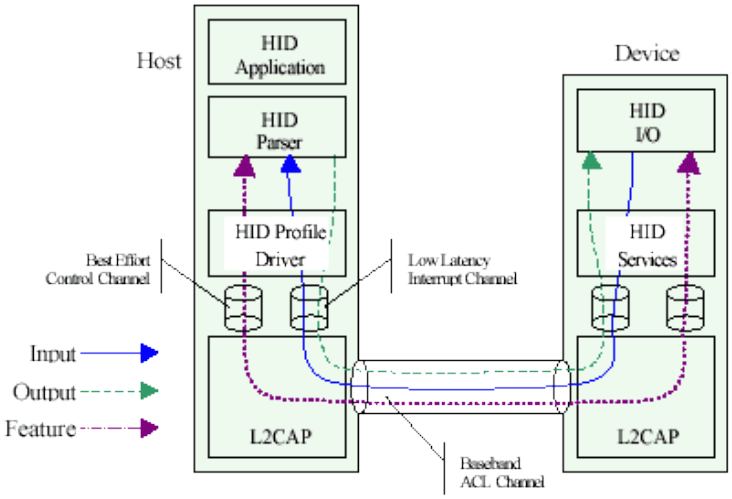


3. ábra: A "csupasz" Wii remote

A Wii Remote standard Bluetooth HID [4] protokollal kommunikál a hoszttal, amely közvetlenül a USB HID szabványon alapul. Mint ilyen, a hoszt oldalán, hagyományos beviteli eszközként jelenik meg. Ugyanakkor a Wii Remote nem használ a szabvány által leírt standard adattípusokat és HID leírókat. Meglehetősen összetett műveleteken keresztül továbbít HID kimenet jelentéseket, és különböző adatcsomagokat.

Nem használja továbbá a szabvány autentikációs és titkosítási funkcióit sem. Szinkronizáláshoz (a tulajdonképpeni kapcsolatteremtéshez) csak le kell nyomni az eszköz „1” és „2” gombjait. Ha ez rendben megtörténik és a párosítás lezajlott a Wiimote -ot a rendszerre telepített standard Bluetooth HID eszközmeghajtón keresztül tudjuk elérni.

A HID szabvány lehetővé teszi, hogy az eszköz egy saját HID leíró blokkon keresztül definiálja adatcsomagjainak felépítését. Ebben a leíróblokkban van egy felsorolás az adatcsomagok struktúrájáról (Ezt úgy kell elképzelni, mint például a hálózati portokhoz rendelt adott szolgáltatásokat.)



4. ábra: HID - L2CAP leképezés

Az adatlekérés a Bluetooth L2CAP protokolljának segítségével történik, amikor is a leíróblokk tartalmazza a jelentés azonosítóját, irányt, illetve a csomag „hasznos terhének” méretét.

I/O	ID	Méret	Funkció
...			
O (52)	0x19	1	Speaker Mute
I (a1)	0x30-0x3f	2-21	Data reports
...			

1. táblázat:HID leíró blokkok

Például a **(a1) 30 00 00** riport egy Input csomagot jelent a x30 csatornán 2 db 00 -ás adatbájttal. Vagy küldjünk egy 0x04 -et a 0x19 -es Output reportra ez elhallgattatja a beépített csipogót: **(52) 19 04**

Egy ilyen „DATA REPORTING” -ra másodpercenként 100 -szor kerülhet sor.

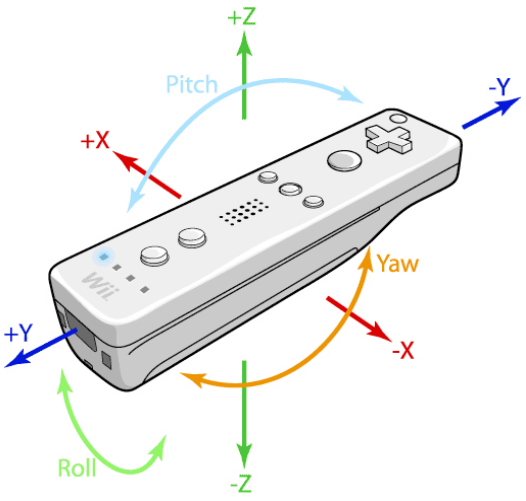
1.2.3. Bemenetek

1.2.3.1. Gombok

A Wii remote -nak 11 gombja van elől és egy az eszköz hátulján. Ha egy gombot megnyomunk, akkor az a gomb azonosítójának megfelelő bitet 1 -re állítja.

1.2.3.2. Mozgásszenzor (vagy gyorsulásmérő)

Erről a funkcióról az Analog Devices ADXL330 -as típusú 3 irányú, lineáris gyorsulásmérője gondoskodik. Ez paraméterekben +/- 3 g -s gyorsulást, irányonként 8 bitet, és 100Hz -es frissítési rátát jelent.



5. ábra: Érzékelt mozgásirányok

1.2.3.3. Infravörös kamera

Az IR kamera szenzorát a PixArt Imaging gyártja. Ennek különlegessége a MOT (Multi Objective Tracking) motor, amely akár 4 IR forrás nagy sebességű és magas felbontású követésére teszi képessé az eszközt. A kamera tényleges fizikális felbontása 128x96 –os nyolcszoros szubpixel analízissel, ami így a gyakorlatban 1024 x 768 -as felbontást, 100Hz -es képfrissítést és 45 fokos látószöget jelent.

A MOT teljes képfeldolgozást végez, minimalizálja az adatátvitelt a Bluetooth csatornán és nagymértékben leegyszerűsíti a kamera alapú nyomkövető alkalmazások végrehajtását.

A kamera használatakor be tudjuk állítani, hogy a feldolgozott adatok milyen formában jelenjenek meg a kimeneten (DATA REPORT MODE). Ezek a következők lehetnek

- **BASIC:** Ebben a módban az infravörös kamera 10 bájtot küld a kimenetre, a maximálisan 4 IR pont X és Y koordinátáival. Minden paraméter 10 biten van tárolva, az X–ek 0 - 1023 –ig, az Y–ok pedig 0-767 –ig. A pontok párosával kerülnek egy-egy 5 bájtos csomagba. A 2. táblázat a BASIC mód adatszerkezetét mutatja.

	Bit							
Bájt	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X1**<7:0>							
1	**Y1**<7:0>							
2	**Y1**<9:8>		**X1**<9:8>		**Y2**<9:8>		**X2**<9:8>	
3	**X2**<7:0>							
4	**Y2**<7:0>							

2. táblázat:a BASIC mód adatszerkezete

- **EXTENDED:** Ebben az üzemmódban a kamera ugyanolyan adatokat küld mint a BASIC módnál, kiegészítve a pontok „durva” méretével, ami tulajdonképpen azok távolságának felel meg. Az adat mérete 12 bájt, 3 bájt IR pontonként. Mértéke 0-15 – ig terjed
- **FULL**: Ekkor minden rendelkezésre álló adat elküldésre kerül, egy összesen 36 bájt méretű adatcsomagban.

1.2.4. Kimenetek

1.2.4.1. LED -ek (látás)

A vizuális visszajelzés az előlapon elhelyezett 4 db kék fénykibocsátó dióda feladata. Wii konzolnál a játékosok beazonosítására szolgál.

1.2.4.2. Vibrációs motor (tapintás)

A kis vibrációs motor a tapintási érzékszervekre hat. Hasonlókat használnak a mobiltelefonokban is. A motor állapota bináris, de változtatható az intenzitása, aktivitása (lüktető, elnyújtott).

1.2.4.3. Csipogó (hallás)

4 bites, 4 Khz -es hangminták megszólaltatására, általában figyelemfelhívásra hivatott.

1.2.4.4. Memória

Az eszköz 5,5 kbyte perzisztens (flash) memóriát tartalmaz. Ez felhasználható mind beállítások, mind output állapotok, mind pedig általános adatok tárolására.

1.2.4.5. Egyéb

Megemlíteném még a távirányító alján található bővítőportot amely, 3.3 V -al 400KHz -es I2C soros kommunikációt támogat. Ehhez a külső bővítmények az eszköz Bluetooth to I2C hídján keresztül csatlakoztathatók.

1.2.5. Reportmódok

A Wii remote több különböző adatszolgáltatási módokkal rendelkezik. Az eszköz külső és belső adatok kombinációit küldi el a hoszt felé azonosítóval ellátott csatornákon keresztül, amiket a reportmód határoz meg. A perifériák adatformátumát maga a priféria határozza meg, amit aztán a kontroller továbbít a vevőnek. Amikor beállítunk egy ilyen módot akkor meghatározzuk, hogy a Wii remote mely részegységétől, milyen adatszerkezetű csomagokat szeretnénk megkapni. A következő táblázat a beállítható módokat tartalmazza:

ID	Data reporting
0x30	Core Buttons
0x31	Core Buttons and Accelerometer
0x32	Core Buttons with 8 Extension bytes
0x33	Core Buttons and Accelerometer with 12 IR bytes
0x34	Core Buttons with 19 Extension bytes
0x35	Core Buttons and Accelerometer with 16 Extension Bytes
0x36	Core Buttons with 10 IR bytes and 9 Extension Bytes
0x37	Core Buttons and Accelerometer with 10 IR bytes and 6 Extension Bytes
0x3d	21 Extension Bytes
0x3e/0x3f	Interleaved Core Buttons and Accelerometer with 36 IR bytes

3. táblázat: DATA REPORT módok

2. AZ ALKALMAZOTT MATEMATIKAI HÁTTÉR

Ez a fejezet taglalja a következő fejezetben használt algoritmusok matematikai hátterét. A fő irányvonalat a kalibrációs eljárás határozza meg, ezen belül is a projektív transzformációval kapcsolatos eljárásokat, műveleteket kell alkalmaznunk. Szó lesz a lineáris egyenletrendszerek megoldhatóságáról, mátrixműveletekről és persze ezzel összefüggésben a Gauss eliminációról is.

2.1. Koordináta- és ponttranszformációk [5]

Az analitikus és a numerikus kezelés érdekében mind grafikai, mind geometriai problémák megoldásához az objektumokat valamilyen koordináta-rendszerben írjuk le. A derékszögű Descartes-féle koordináta-rendszerek a leggyakrabban használt, de nem kizárólagos vonatkoztatási rendszerek.

Általában az elemi alakzatok saját koordináta-rendszerükben vannak leírva (amiben könnyű őket leírni), és ezekhez az elemi alakzatokat közös térbe elhelyezve hozunk létre új, összetett alakzatokat. Ez azt jelenti, hogy koordináta-rendszerek közötti áttéréseket és az azokban megjelenő alakzatok transzformációját is le kell tudnunk írni.

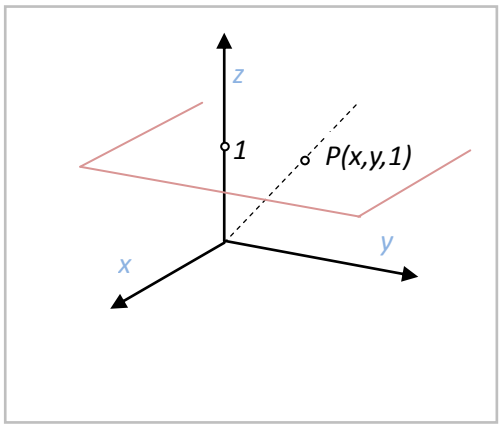
Vizsgálataink során, a Descartes-féle koordinátákon kívül gyakran használunk homogén koordinátákat is. A homogén koordináták használata projektív transzformációknál (pl. centrális vetítésnél) elengedhetetlen, más esetekben ugyan csak formális, de egységes tárgyalásmódot tesz lehetővé, ami nagymértékben megkönnyíti munkánkat.

2.1.1. Homogén koordináták

A projektivitás a projektív sík egy általános egyenestartó leképezése. Az ábrázoló geometriában a projektív síkot úgy tekintjük, mint az euklideszi sík végtelen távoli elemekkel való kibővítése.

2.1.2. Síkbeli homogén koordináták

Legyen adott a végtelen távoli térelemekkel kibővített euklideszi térben egy Descartes-féle derékszögű koordináta-rendszer és ebben a z = 1 sík. Ezen sík pontjai és a koordináta-rendszer kezdőpontján áthaladó egyenesek között egy kölcsönösen egyértelmű kapcsolat értelmezhető úgy, hogy az egyenesekhez a síkkal alkotott döféspontjukat rendeljük hozzá, a sík tetszőleges pontjához pedig az origóval összekötő egyenest. Az x – y koordinátasíkban fekvő egyeneseknek a z = 1 sík végtelen távoli pontjai felelnek meg (lásd 6. ábra).



6. ábra

Az origón áthaladó egyenesek egyértelműen megadhatók irányvektorukkal, melyek hossza tetszőleges, tehát a $v(x,\ y,z)$ és $\lambda v(\lambda x,\ \lambda y,\ \lambda z),\ 0 \neq \lambda \in IR$ ugyanazt az egyenest, azaz ugyanazt a z = 1 síkra illeszkedő pontot reprezentálja. Ilyen módon a z = 1 sík pontjait olyan rendezett számhármasokkal reprezentáljuk, amelyek rangja (nem lehet mindhárom egyszerre nulla) és arányosság erejéig vannak meghatározva. Ezeket, a koordinátákat nevezik homogén koordinátáknak.

A homogén koordinátákat a projektív geometria tételeinek bizonyításánál használják, de mint látni fogjuk, a számítógépi geometriai és grafikai problémák megoldásánál is igen hasznosak.

2.1.3. Ponttranszformációk

Ponttranszformáción egy olyan megfeleltetést értünk, amely során egy ponthalmaz minden pontjához hozzárendelünk egy-egy pontot.

Most csak olyan ponttranszformációkat vizsgálunk, amelyek kölcsönösen egyértelműek, ezen kívülilleszkedés és egyenes tartók. Az itt tárgyalt legbővebb ponttranszformáció a projektív transzformáció. A ponttranszformációk leírásához homogén koordinátákat használunk fel.

A transzformációk általános formája

$$\mathbf{p' = M \cdot p}$$

ahol p a transzformálandó a p' a transzformált pont helyvektora, M pedig a transzformációt reprezentáló mátrix – a transzformáció mátrixa. Síkbeli transzformáció esetén kiterjedése M 3 × 3 -as, és természetesen|M|≠ 0.

2.1.4. Projektív leképezés

A projektív leképezés [6], más néven a projektív vagy homogén transzformáció, egy vetítés egy síkból adott ponton keresztül egy másik síkra. A projektív transzformációt széles körben használják a 3D-s modellezésekben affin transzformációk és a perspektív transzformációk formájában.

Hasonlóan az affin transzformációhoz a projektív transzformáció is kifejezhető úgy, mint a homogén koordináták lineáris leképezése. Ez homogén mátrix jelölésekkel (1):

$$\begin{pmatrix} x' \\ y' \\ h' \end{pmatrix} = \begin{pmatrix} h'x' \\ h'y' \\ h' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Ekkor,

$$h' = a_{31}x + a_{32}y + 1$$

Az egyenlet átírható úgy, hogy

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \frac{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}{\begin{pmatrix} a_{31} & a_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}$$

A projektív leképezés általános alakja ebből kifejezve:

$$x' = \frac{1}{h'} \cdot (a_{11}x + a_{12}y + a_{13}) = \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + 1}$$

$$y' = \frac{1}{h'} \cdot (a_{21}x + a_{22}y + a_{23}) = \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + 1}$$

A projektív leképezés inhomogén formában is leírható [7]:

$$x'_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3$$

$$x'_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3$$

$$x'_3 = a_{31}x_1 + a_{32}x_2 + a_{33}x_3$$

A fenti összefüggést tovább alakítva, az első relációt elosztva a harmadikkal:

$$\frac{x'_1}{x'_3} = \frac{a_{11}x_1 + a_{12}x_2 + a_{13}x_3}{a_{31}x_1 + a_{32}x_2 + a_{33}x_3}$$

A jobb oldali törtet bővítve $1/x_3$-mal, valamint felhasználva a Descartes és homogén koordináták közötti kapcsolatot *(* $x = \frac{x_1}{x_3}$, $y = \frac{x_2}{x_3}$)a projektív leképezés általános alakja kifejezhető (2):

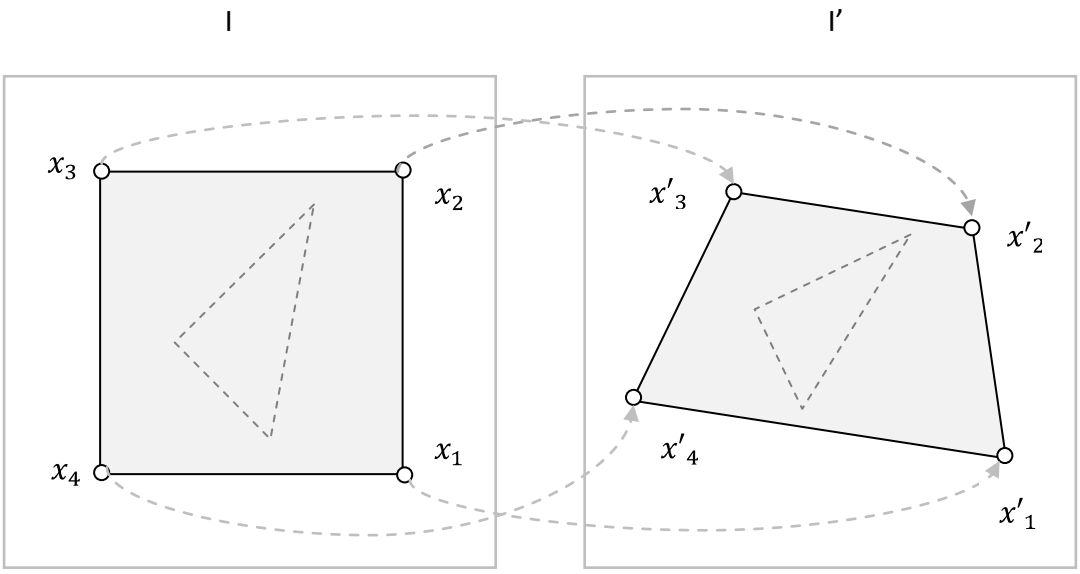
$$x' = \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + a_{33}}\ ,\ y' = \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + a_{33}}$$

A fenti eljárásnak természetesen nincs értelme, ha $x_3 = 0$ vagy a $a_{31}x + a_{32}y + a_{33} = 0$. Tehát minden projektív transzformáció az euklideszi síkon Descartes koordinátákkal megadható, mint lineáris tört transzformáció közös nevezővel.

Mint látható a leképezés eredménye nemlineáris. A nemlinearitás ellenére a projektív transzformációk során az egyenes képe egyenes marad, de a párhuzamosság nem marad változatlan. Ezért a négyzet képe tetszőleges négyszög lehet, a körből pedig lehet parabola vagy hiperbola is. A projektív transzformáció mátrixára csak az a kikötés, hogy determinánsa legyen nullától különböző, azaz létezzen inverze. Tehát bármely nemszinguláris 3 × 3-as mátrix síkbeli projektív transzformációt ír le.

Bár a fenti mátrixnak 9 együtthatója van, ezek a leképezések homogének, így bármely nem nulla skalárral való szorzata a mátrixnak ugyanolyan leképezést ad eredményül.

A kétdimenziós síkból vett négy tetszőleges $(x_1,\ x_2,\ x_3,\ x_4)$ pontjának $(x'_1,\ x'_2,\ x'_3,\ x'_4)$ pontokba történő leképezéséhez 8 szabadságfokra van szükség.



7. ábra: Projektív leképezés

2.1.5. Transzformációs paraméterek meghatározása

Adva van tehát négy pár síkbéli pont $(x_1,\ x'_1) \dots (x_4,\ x'_4)$, egy $x_i = (x_i,\ y_i)$ pont a forrás síkon, egy $x'_i = (x'_i,\ y'_i)$ pedig a cél síkon. Keressük a nyolc ismeretlen transzformációs paramétert $a_{11} \dots a_{32}$. Behelyettesítve az adott pont koordinátáit $(x'_i,\ y'_i)$ a fenti *(2)* egyenletbe, minden pontpárra kapunk egy lineáris egyenletpárt (3):

$$x'_i = a_{11}x_i + a_{12}y_i + a_{13} - a_{31}x_i x'_i - a_{32}y_i x'_i$$

$$y'_i = a_{21}x_i + a_{22}y_i + a_{23} - a_{31}x_i x'_i - a_{32}y_i x'_i$$

Mátrixba rendezve a nyolc egyenlet paramétereit a következőt kapjuk:

$$\begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{pmatrix} \cdot \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{pmatrix}$$

vagy

$$x' = M \cdot a$$

Az ismeretlen $a = (a_{11}, a_{12}, \dots, a_{32})$ paraméterek így kiszámíthatóak a lineáris egyenletrendszerek megoldásával. Erre használhatunk olyan, a lineáris algebrából ismert módszert, mint például a Gauss elimináció.

2.2. Lineáris egyenletrendszer

Számos probléma megoldása visszavezethető lineáris egyenletrendszer megoldására [8]. Ilyen például az előzőekben ismertetett projektív leképezés is. Direkt és iteratív módszereket egyaránt alkalmazhatunk, a megoldáskeresés során.

A lineáris egyenletrendszer általános alakja

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\dots$$

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

mátrix írásmódban

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

$$A \cdot x = b$$

A feladat az *x* vektor meghatározása. Mint látható ez az általános alak ugyanaz mint a 2.1.5 fejezetben kapott, csak más jelölésekkel.

Feltételezzük, hogy minden ismeretlennek legalább egy együtthatója nem nulla, és hogy az egyenlet együtthatóinak determinánsa sem nulla. Ilyenkor az egyenletrendszernek létezik megoldása:

$$x_1\ x_2 \dots x_n$$

Ha a determinánsa nulla, akkor vagy nincs, vagy végtelen sok megoldása létezik. Ebben az esetben egy vagy több ismeretlen értéke szabadon megválasztható, a többi pedig ezek lineáris függvénye.

A lineáris egyenletrendszerek megoldása különböző módszerekkel történhet

- kiküszöbölési eljárással (direkt módszer)
- fokozatos közelítési módszerrel (iteratív módszer).

Az *b* jobb oldali vektort elhelyezzük az *A* mátrix *n+1* –edik oszlopában. Ha az *A* mátrixot egy oszloppal kiterjesztjük, könnyebb az algoritmus kidolgozása:

$$a_{in+1} = b_i$$

Az egyenlet ekkor a következő alakot veszi fel:

$$\sum_{j=1}^{n} a_{ij} \cdot x_j = a_{in+1} \qquad (i = 1,2, \dots n)$$

2.2.1. Gauss- féle kiküszöbölési eljárás [9]

A lineáris egyenletrendszert sorozatos átalakításokkal először felső háromszög mátrixú egyenletrendszerré alakítjuk, melyből sorozatos visszahelyettesítéssel közvetlenül megkapjuk a megoldásvektor elemeit:

$$a_{11}x_1\ a_{12}x_2 \dots\ a_{1n}x_n = a_{1n+1}$$

$$a_{22}x_2 \dots\ a_{2n}x_n = a_{2n+1}$$

$$\dots$$

$$a_{nn}x_n = a_{nn+1}$$

Ha sikerül az *A* együttható mátrixot ilyen formára átalakítani, akkor az alsó egyismeretlenes egyenletből kiindulva, sorozatos visszahelyettesítéssel megoldható az egyenletrendszer. Az átalakításoknál ügyelni kell arra, hogy a jobboldali vektor is transzformálódik. Ha egy egyenletrendszernek több jobb oldala van, akkor mindent újra kell számolni.

Vegyük tehát az *A* n-ed rendű mátrixot amit n-1 iterációs lépésnek vetünk alá. Legyen *k* az iterációs index, *i* és *j* pedig a mátrix adott elemének indexe. Az iteráció képlete:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \cdot a_{kj}^{(k-1)}$$

ahol:

$$k = 1, 2, \ldots, n\text{-}1$$

$$i = k+1, \ldots, n$$

$$j = k+1, \ldots, n+1$$

Az n-1 lépés után előáll a háromszög mátrix

$$a_{11}^{(0)} x_1 + a_{12}^{(0)} x_2 + \cdots + a_{1n}^{(0)} x_n = a_{1n+1}^{(0)}$$

$$\ldots$$

$$a_{nn}^{(n-1)} x_n = a_{nn+1}^{(n-1)}$$

Az átrendezést követően a főátló alatti elemek nullák lesznek. Visszahelyettesítésnél az utolsó egyenletből kell kiindulni. Probléma akkor van, ha a főátlóbeli elem 0 vagy nagyon kicsi. Az ismeretlenek meghatározását az n-edik ismeretlen számításával kezdjük

$$x_n = \frac{a_{nn+1}^{(n-1)}}{a_{nn}^{(n-1)}}$$

Ezután az ismert gyökök visszahelyettesítésével kapjuk meg az egyenletrendszer további gyökeit:

$$x_{n-1} = \frac{1}{a_{n-1n-1}^{(n-2)}} \left[a_{n-1n+1}^{(n-2)} - a_{n-1n}^{(n-2)} \cdot x_n \right] \quad stb.$$

2.2.2. Gauss elimináció részleges főelemkiválasztással [10]

Ha az együtthatók különbsége nagy, és a főátlóban levő elem (az osztó) értéke kicsi, a megoldás során jelentős hiba keletkezhet. Jobb eredményt kapunk, ha az *i*-edik ismeretlent az egyenletrendszernek abból az egyenletéből küszöböljük ki, ahol az ismeretlen együtthatója abszolút értékben a legnagyobb. Ezt a módszert részleges főelemkiválasztásnak nevezzük.

Ha a főátlóbeli elem volt a legnagyobb, az egyenleteket felcseréljük. Ha az egyenletrendszer megoldható, akkor a részleges főelemkiválasztás nem biztos, hogy a legjobb eredményt adja a kerekítési hibák miatt.

3. A WIIMOTEST ALKALMAZÁS

A diplomadolgozat keretében kidolgozásra került alkalmazás két főbb részre tagolódik. Az első programrész tesztelési célokat lát el. Ennek érdekében készítettem egy grafikus felhasználói felületet mely az eszköz képességeit próbálja vizuális formába önteni.

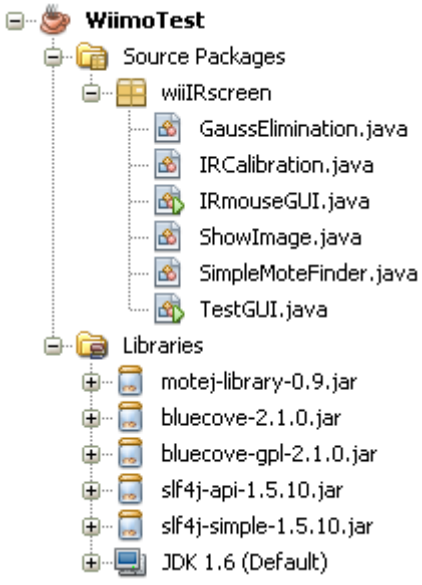
A második részben következik az eszköznek egy konkrét alkalmazása melynek megvalósításával a számítógép monitorának képernyője „interaktívvá" tehető.

Nem célom, hogy mélyrehatóan belemenjek a programozási technikák részletezésébe, inkább a funkcionalitás megismertetése törekszem.

3.1. Alkalmazás architektúra

A programot JAVA nyelven írtam, a NetBeans fejlesztői környezet segítségével. A nyelv kiválasztásában szerepet játszott a platformfüggetlenség és nem utolsó sorban a gyors implementálhatóság.

A projekt felépítését a következő ábra mutatja



8. ábra: A projektstruktúra

Amint az az ábrán jól látható a projek két jól elkülönülő részre bontható

1. Forrás csomag
2. Osztálykönyvtárak

3.1.1. Forrás csomag (Source Package)

Mindössze egy csomagot hoztam létre, aminek a **wiiIRScreen** nevet adtam. Itt jegyezném meg, hogy szándékosan használtam az implementálás során angol elnevezéseket. Úgy vélem , hogy a kódolás szintaktikája sokkal kifejezőbb, ha a programozás „anyanyelvét" használjuk az elnevezési konvencióknál.

A „package" öt programegységre tagolódik, ami tulajdonképpen egy-egy JAVA osztályt is jelent:

- ***GaussElimination.java:*** Az osztály egyetlen függvényt tartalmaz, aminek a feladata a matemetikai résznél tárgyal Gauss-elimináció eredményének kiszámítása lesz.
- ***IRCalibration.java:*** Ez a fájl tartalmazza azt az osztályt és metódusait, amelyek a Wii remote és a képernyő kalibrációjához kellenek.
- ***IRMouseGUI.java:*** Létrehoz egy grafikus felhasználói interfészt annak érdekében, hogy az kalibráció, az egérkurzor, ill. az eszköz státusza nyomon követhető legyen. A kalibáció után ez a programegység vezérli az egérmutatót a kiszámított adatoknak megfelelően.
- ***ShowImage:*** Képet jelenít meg egy panelben.
- ***SimpleMoteFinder.java:*** Egy lenttebb részletezett java könyvtárra épülő segédosztály. A feladata, hogy vezérelje az alkalmazás és a Wii remote egymás közötti egyszerű kapcsolódását.
- ***TestGUI.java:*** Ez tulajdonképpen egy grafikus panel amely az eszköz képességeit próbálja bemutatni. Tesztelési céllal hoztam létre, hogy az itt mért adatokból információt gyűjthessek az kontroller működéséről.

Az itt felsorolt programrészek részletesebb ismertetését egy későbbi bekezdésben fogom tárgyalni.

3.1.2. Osztály könyvtárak (Libraries)

Itt találhatóak azok a JAR állományok (archívumok) amelyeket beépítek a megírt alkalmazásba. Ezek tulajdonképpen olyan import algoritmus gyűjtemények, amelyeket a programozó felhasználhat a kódolás során, és nem kell újra megírnia azt, amit egyszer már megírt vagy mások megírtak. Mind egy-egy kölün funkciót lát el, és olyan függvényeket, metódusokat, osztályokat biztosítanak, amik meggyorsítják a programozási munkát.

Lássuk, melyek ezek tehát:

- **JAVA JDK 1.6:** A J2SE - Java 2 plattform fejlesztői keretrendszerét tartalmazza. A J2SE az általános programozás számára kifejlesztett technológiák és specifikációk gyűjteménye. Egy konfiguráció adja az architektúra alapját. Erre épül az adott eszköz működését meghatározó profil és egyéb kiegészítő API-k. Ezek képezik az alapot saját alkalmazásunk elkészítéséhez.

- **Bluecove-2.1.0 (gpl):** A Bluecove egy JSR-82 implementáció a J2SE plattformon. Egy alkalmazásprogramozási felület (API) aminek az a feladata, hogy megoldja a Bluetooth protokollok megvalósítását. Nyílt forráskódú és a főleg PC-s alkalmazásokra lett kifejlesztve. A GPL állomány a Linux rendszereken használt General Public License miatt szükséges.

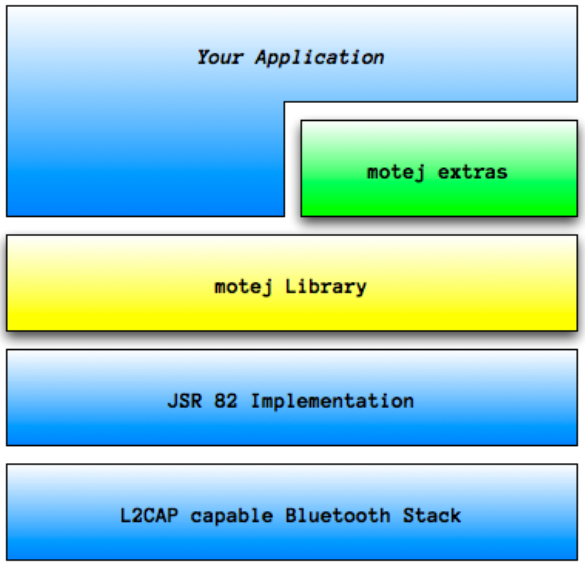
- **slf4j-1.5.10:** A slf4J (Simple Logging Facade for Java) széles körben elterjedt, könnyen használható Java nyelven implementált naplózó rendszer. A MOTEJ nevezetű library működéséhez szükséges.

- **motej-library-0.9:** Nem véletlenül hagytam a végére a legfontosabb könyvtárat. A világhálón több olyan API is fellelhető ami a Wiimote programozását hivatott támogatni. Hosszasan tanakodtam azon, hogy saját libraryt fejlesszek a Wii kontrollerhez, de ez a szakdolgozat kereteibe már nem fért volna bele. Ezért egy olyan API –t választottam, ami szabadon rendelkezésre áll, de még nem implementálódott egyetlen hasonló projektben sem[4].

4 Legalábbis én nem találtam ilyet.

A motej [11] egy olyan java könyvtár amely a Wii remote számítógépen való programozásához biztosít keretrendszert. Ingyenes és nyílt forrású (Apache Common License 2.0), bárki által felhasználható.

A motej két fő csomagja, a *motej könyvtár* és a *motej extrák*. Míg a motej extra a motej könyvtárra épül, és további funkciókat nyújt, a motej könyvtár hozzáférést biztosít az alapvető Wiimote funkciókhoz, és csak egy JSR 82 (Bluetooth) implementációt - esetünkben a Bluecove - illetve az slf4j naplózórendszert meglétét igényeli.



9. ábra: A motej rétegződése

A könyvtár szolgáltatásai:

- ✓ Wiimote megkeres és kapcsolódás
- ✓ Wiimote IR kamera basic, extended és interleaved módok
- ✓ Wiimote gyorsulásmérő
- ✓ Wiimote output (rezgőmotor, led, speaker) vezérlés
- ✓ Wiimote gombok
- ✓ Olvasás a Wiimote flash memóriából
- ✓ A Wiimote regiszterek írása
- ✓ Wiimote állapot információk

3.2. Eszközprogramozás és tesztelés - TestGUI

Ez az osztály egy wii remote tesztelésére használható grafikus felhasználói felületet implementál, melyhez a JAVA JDK könyvtár AWT és Swing komponenseit, és természetesen a motej könyvtárat fogjuk felhasználni.

A TestGUI osztályt egy keretosztályként definiáljuk, mely a JFrame osztályból van származtatva. A keret összerakása belülről történik, azaz a beállításokat belülről végezzük el, nem pedig kivülről küldött üzenetekkel.

10. ábra: a TesztGUI frame Windows XP rendszeren

3.2.1. A TestGUI frame felépítése

A keret kisebb JPanel típusú elemekből építkezik. Ezek között vannak olyanok, amelyek JPanel oszályból származtatott újabb osztályok. A TestGui frame paneljei:

- **StatuszPanel (class extends JPanel):** Négy kisebb JPanelt fog össze GridLayout elrendezés menedzserrel[5].

5 Az elrendezés menedzser automatikusan rendezi a konténer elemeit

- *Eszközstátusz (JLabel):* A Wii remote kapcsolat állapotát jelzi. (Kapcsolódva/Nincs kapcsolat)
- *MAC cím (JLabel):* az eszköz MAC (Media Access Control) azonosítója hexadecimális fromában.
- Elemtöltöttség (JProgressbar): Információt ad az kontroller tápellátásának kondíciójáról.
- Kapcsológombok (JButton): Az egyik gomb kapcsolódást, a másik pedig kapcsolat bontást kezdeményez.

- **MotionPanel (class extends JPanel):** A gyorsulásmérő regiszterállapotait jeleníti meg. A kézben tartott vezérlő valós idejű mozgását követve, megjelennek az X, Y és Z tengelyen való elmozdulás mértékei. A panel alján, a mért legkisebb, ill. legnagyobb elmozdulás látszik.
- **IRPanel (class extends JPanel):** A Wii remot iR szenzorának adatai fogadja és megjelenít egy fehér pontot a kapott helyzet koordinátáknak megfelelően. Az IR szenzor ilyenkor a infravörös fénypont relatív helyzetét érzékeli, és azt feldolgozva átküldi a programnak az X és Y koordinátákat.
- **ButtonsPanel (JPanel):** A kontroller képe mellé felsorolja a rajta található gombokat. Amikor megnyomunk egy vagy akár több gombot is a Wiimote –on a gomb aktívvá válik, azaz a háttere kizöldül.
- **ACCPanel (class extends JPanel):** A gyorsulásmérő elmozdulástengelyeinek értékeit ábrázolja egy grafikonon. Az X -et a piros, az Y -t a zöld a Z -t pedig a kék vonal szimbolizálja.

3.2.2. A TestGUI frame működése

Az alkalmazás indításakor a TestGUI osztály *main()* metódusa hívódik meg. Mivel a komponensek megjelenítése nem platformfüggő, az UIManager osztály segítségével befolyásolhatjuk a felszínünk grafikus megjelenítését, különböző „look and feel” osztályokat rendelhetünk hozzá. Első lépésként tehát meghatározzuk a panel kinézetét, méghozzá a

rendszer natív stílusának lekérdezésével.

Ezután közvetlenül létrehozzuk a TestGUI osztály egy példányát (konstrukció), amely megjeleníti a keretet a képernyő közepén.

```
public static void main(String[] args) throws AWTException {

    try
    {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception ex)
    {
        System.err.println("Nem sikerült UI menedzsert meghatározni" + ex);
    }

    new TestGUI();

}
```

A TestGUI osztály konstruktora egymás után hozzáadja a panelobjektumokat a kerethez, inicializálja, majd a *setVisible()* metódussal láthatóvá teszi a keretet.

Most nézzük mi történik, amikor csatlakoztatjuk a Wii remote –ot, azaz megnyomjuk a „Csatlakoztat” gombot?

```
protected Action findAction = new AbstractAction("Csatlakoztat",
                                new ImageIcon("images/connect.png")) {

public void actionPerformed(ActionEvent arg0) {

    SimpleMoteFinder simpleMoteFinder = new SimpleMoteFinder();
    mote = simpleMoteFinder.findMote();

    if (mote != null) {
        while (mote.getStatusInformationReport() == null) {
            try {
                Thread.sleep(101);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

A program meghívja a gombhoz társított eseménykezelő eljárást, amellyel létrejön a *SimpleMotefinder* osztály egy példánya. Ez az osztály végzi a Wiimote eszköz csatlakoztatását. Ennek az osztálynak meghívjuk a *findMote()* metódusát ezzel elindul az eszköz keresése, majd ezt követően a kapcsolódás. A *findMote()* metódus visszatérési értéke egy objektum lesz mely a *mote* mezőben tárolódik.

Ezután ha a *mote* ≠ *null*, azaz a *findMote()* talált elérhető Wii remote kontrollert megpróbáljuk lekérni annak státuszát a létrejött mote objektum *getStatusInformationReport ()* eljárásával. Ez adatokat szolgáltat a kapcsolódott eszköz állapotáról úgymint elemállapot, MAC address, bővítények, stb. Ezekből az adatokból ezután alaphelyzetbe tudjuk hozni a TestGUI megfelelő objektumait.

Szeretnénk nyomon követni a mozgásszenzor adatainak változásait is. Ennek érdekében a *mote* objektumhoz hozzáadunk egy *AccelerometerListener()* figyelőt, aminek a belsejében lévő *accelerometerChanged()* metódus végrehajtódik, ha a mozgásparaméterekben változás történt.

```
mote.addAccelerometerListener(new AccelerometerListener<Mote>() {
    public void accelerometerChanged(AccelerometerEvent<Mote> evt) {

      x = (int)evt.getX();
      y = (int)evt.getY();
      z = (int)evt.getZ();

      MotParam[0].setVal(x);
      MotParam[1].setVal(y);
      MotParam[2].setVal(z);

      if (x<MotParam[3].getVal()) MotParam[3].setVal(x);
      if (y<MotParam[5].getVal()) MotParam[5].setVal(y);
      if (z<MotParam[7].getVal()) MotParam[7].setVal(z);
      if (x>MotParam[4].getVal()) MotParam[4].setVal(x);
      if (y>MotParam[6].getVal()) MotParam[6].setVal(y);
      if (z>MotParam[8].getVal()) MotParam[8].setVal(z);

      ACCgraph.repaint();

    }
});
```

A *MotParam[]* tömb tárolja a valós idejű mozgásadatokat, melyeknek változásakor ebből a tömbből frissülnek a képernyőn megjelenő adatok is. Az eljárás *ACCgraph.repaint()* parancsa frissíti az új adatok alapján a TestGUI panel *ACCgraph* nevű mozgásgrafikonját.

Az infravörös kamera figyelésénél is ugyanígy járunk el. Ráteszünk az addIrCameraListener() – el a *mote* objektumrera egy irImageChanged() eseményfeldolgozót, amiben a *p0* objektum az infravörös fénypont adatait rögzíti. Ezt követően a keret IRPanel objektumát a kapott adatokkal újrarajzoltatjuk.

```
mote.addIrCameraListener(new IrCameraListener() {
    public void irImageChanged(IrCameraEvent evt) {
        p0 = evt.getIrPoint(0);
        IRcomp.repaint();
    }
});

mote.enableIrCamera(IrCameraMode.BASIC, IrCameraSensitivity.WII_LEVEL_3);
mote.setReportMode(ReportModeRequest.DATA_REPORT_0x37);
mote.setPlayerLeds(new boolean[] {true, false, false, true});
```

Most már van eseménykezelője így engedélyezhetjük az IR kamerát. A három működési, és öt érzékenységi mód közül, itt a *BASIC* módot, ill. a hármas alapszintet állítjuk be.

Meg kell azt is határoznunk, hogy a riportoknak milyen legyen a struktúrája. Nekünk elég most a mozgás, az IR szenzor és a gombok adatait lekérdezni így a *DATA_REPORT_0x37* –es mód erre éppen megfelelő.

Már tudunk adatot fogadni az eszköztől, a mozgás és az infravörös kamerától. Már csak a gombok maradtak hátra. A gombok a wiiButton tömbben vannak JLabel objektumként. Két metódusa a *setOn()* amely bekapcsolt állapotúra állítja a gombot és a *setOff()* ami pedig kikapcsolja azt.

```
mote.addCoreButtonListener(new CoreButtonListener() {
    public void buttonPressed(final CoreButtonEvent evt) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                if (evt.isButtonAPressed())
                    wiiButton[4].setOn();
                else
                    wiiButton[4].setOff();

                if (evt.isButtonBPressed())
                    wiiButton[5].setOn();
                else
                    wiiButton[5].setOff();

                ...
```

Itt a *buttonPressed()* eljárással egy kicsit „trükköznünk kell”, ugyanis a Swing egy speciális AWT eseménykibocsájtó szálon hívja meg az eseménykezelőket. Ezt a szálat egyrészt nem szabad sokáig lefoglalni, mert arra az időre lefagy a GUI, másrészt az esemény meghívásakor további belső hívások is sorban állnak, hogy ugyanezen a szálon végrehajtódjanak, és ha egyéb műveleteket futtatunk a kezelőben, akkor a további hívások még módosíthatnak az

eredményen. Ezért van egy állandóan használatos metódus, ami a végrehajtási sor végére teszi a teendőket: *SwingUtilities.invokeLater()*. Az eseménykezelő ezt meghívja, és a belső *run()* metódus, a megfelelő időben fog végrehajtódni.

Van még egy figyelőnk ez pedig a *Szétkapcsol* gomb funkcióját határozza meg. Nincs más dolga, minthogy bontsa a kapcsolatot az eszközzel és a megfelelő állapotba állítsa a keretben a panelobjektumokat. Kapcsolatbontáskor a *mote.disconect()* eljárás kerül meghívásra.

```
protected Action disconnecting = new AbstractAction("Szétkapcsol",
                                 new ImageIcon("images/disconnect.png")) {
    public void actionPerformed(ActionEvent arg0) {
        mote.disconnect();
        status.setText("Nincs kapcsolat.");
        MAC.setText("null");
        disconnect.setEnabled(false);
        connect.setEnabled(true);
    }
};
```

Tulajdonképpen ennyi kell ahhoz, hogy a Wii remote input részeit működésbe hozzuk. Az alacsonyabb szintű vezérléseket a már ismertetett osztálykönyvtárak végzik.

3.2.3. Mérések, tapasztalatok

Mivel a dolgozat témája egy tulajdonképpeni hardverprogramozás szót kell, hogy említsek az eszköz programozása közben szerzett tapasztalatokról.

Nos, kezdjük akkor a szoftveres résszel. A *motej* mint vezérlőosztály szépen ellátja a feladatát, vannak viszont hiányosságai. Ebből megemlítenék néhányat felsorolásszerűen.

- Amikor az eszköz elemei teljes töltöttségen vannak a *motej* egy negatív számmal tér vissza. (Bár ezt lehet, hogy az eszköz állítja elő.)
- Másik gyenge pont a kapcsolat bontása. A *disconnect()* metódus nem minden esetben zárja le jól a *Bluecove* kapcsolatokat. Ez Windows –os *WIDCOMM* rendszereken a bluetooth vezérlés lefagyásához vezet.
- Nagyon ritkán, véletlenszerűen a rendszer bontja a kapcsolatot bluetooth hibára hivatkozva.
- Ha a *mote* objektum nem semmisül meg valamilyen ok folytán, következő indításnál a

nem kapcsolt eszközt a *motej-library* mégis online -nak látja.

Az eszközzel kapcsolatban is megemlítenék néhány tapasztalatot.

- Az infravörös kamera esetében, ha túl közel kerül a az IR pont a kontrollerhez – az IR pass (áteresztési) szűrő ellenére - a tükröződések miatt a szenzor a jelet nem képes értelmezni
- Ugyanígy viselkedik túl világos helyiségben, nappal szemben[6], vagy ha irányított fény kerül a kamera látószögébe.
- Sajnos sokszor kevés az IR szenzor 45^{o} –os látószöge, illetve a néhány méteres látótávolság
- Szabadesés közben a gyorsulásmérő nem mér gyorsulást, viszont pontos, megfelelően érzékeny és nagyon gyorsan követi a helyzetváltozásokat.
- Általános felhasználás mellett bőven elegendőek az 1 bájtos (0-255) paraméterek

Mindent összevetve az eszköz nagyon korrektül működött, és jól felhasználható szemléltetésre vagy akár kutatási célokra is.

6 Mivel a természetes fénynek is van infravörös tartománya

3.3. Interaktív felület megvalósítása – IRmouseGUI

Ebben a fejezetben kerül tárgyalásra az eszköz használatának egy gyakorlati megvalósítása. Egy olyan alkalmazát készítek, amelyjel a számítógép monitorán az egérkurzor egy infravörös leddel ellátott mutatóeszközzel vezérelhető. Kvázi érintőképernyőssé alakítok egy hétköznapi kijelzőt.

3.3.1. „Rendszerkövetelmények”

Az elképzelés megvalósításához szükségünk lesz

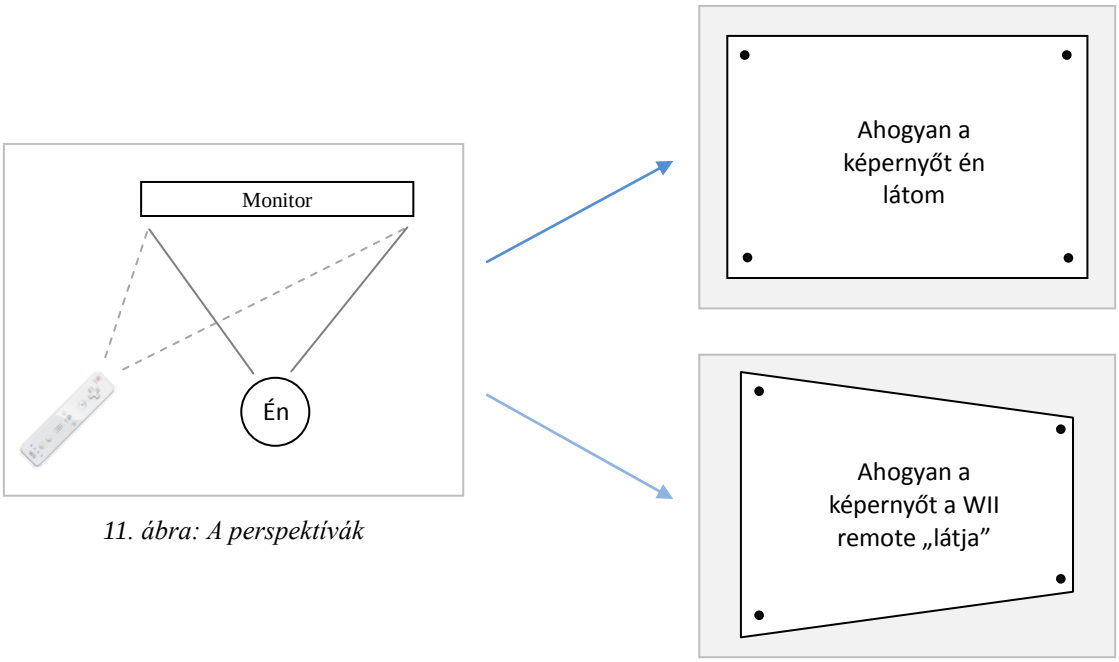
- ✓ egy számítógépre,
- ✓ egy minimum 1024 x 768 –as felbontású monitora,
- ✓ egy bluetooth adapterre,
- ✓ egy Wii remote kontrollerre,
- ✓ egy infravörös tollra[7]
- ✓ és persze az itt tárgyalt alkalmazásra.

3.3.2. Hogyan működik?

Amikor az infravörös fény mozog a képernyő előtt a Wii remote ezt a mozgást érzékeli. A fényforrás jelenlegi helyzetéből kiszámítja az X és az Y koordinátákat, amiket aztán továbbít a számítógép felé. A szoftver a kapott paramétereknek megfelelően, az IR led pozíciójába viszi az egérmutatót.

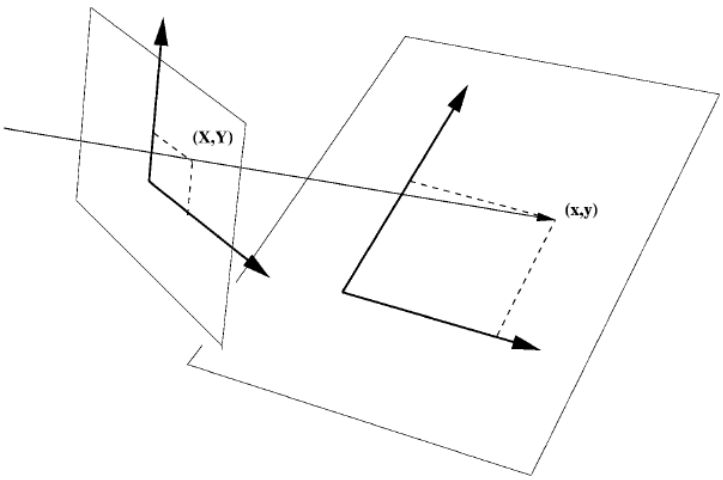
Igen ám, de amint azt a 12. ábrán láthatjuk a kontroller másként látja a képernyő felületét, mint mi magunk. Ő ugyanis más szemszögből, más perspektívából kapja meg ugyanazt, mivel a betekintési szögnek megfelelően a perspektivikus kép „torzul”. Ilyenkor ugyanaz a távolság a kontroller szemszögéből rövidebbnek, vagy éppen hosszabbnak látszik. De akkor honnan tudja a program, hogy a kapott koordináták hova esnek a képernyőn valójában?

[7] A tollat magam készítettem, egy elem, egy mikrokapcsoló, és egy TV távirányítójából kiszerelt IR LED segítségével



11. ábra: A perspektívák

Itt kerül előtérbe a 2. fejezetben tárgyalt projektív transzformáció. Ha ezt magyarra fordítjuk – vetített leképezés – már láthatjuk, hogy ez éppen arra való, hogy egy síkbeli pont egy másik síkba vetített képét, vetületét meghatározza.



12. ábra: Pont vetülete

De van még egy probléma, ami megoldásra vár. Valahogyan közölnünk kell a programunkkal, hogy a Wiimote milyen perspektívát lát. Ez a folyamat lesz a kalibráció első fázisa, amikor is a képernyő 4 sarkánál elhelyezünk 4 konstans helyzetű pontot. Ezeknek a pontoknak a koordinátáit a program pontosan tudja.

Amikor a pontokra kattintunk az IR tollal, akkor a Wii remote által érzékelt koordináták szintén eltárolódnak a gép memóriájában.

Ezután, már csak annyi a történik, hogy a kalibrációs algoritmus megfelelteti egymásnak a képernyőn lévő és az eszköz által kapott pontok helyzetét, és kiszámítja, hogy a kontroller által látott pont az hová esik (vetül) a képernyőn.

Amennyiben a kalibrációs folyamat rendben lezajlott a program átveszi az egérmutató irányítását, és oda viszi a kurzort ahol az IR tollat érzékeli. Ha megszűnik a fénypont, a program is elengedi az egérmutatót.

3.3.3. Az IRmouseGUI felépítése

Az IRmouseGUI egy JFrame osztályból származtatott keret, mely négy panelt és két vezérlőgombot foglal magában.

13. ábra: Az IRmouseGUI keret

Az első panel a Wii remote MAC címét, a második annak elemtöltöttségét mutatja. A *„Kalibrálva”* panel arról ad tájékoztatást, hogy a kalibrálás megtörtént-e már, míg a negyedik a monitor aktuális felbontását tartalmazza.

A *„C* gomb megnyomásával indíthatjuk el a kalibrációs folyamatot. Amint ez rendben lefutott a program átveszi az irányítást az egérkurzor felett.

Ha be szeretnénk fejezni a program működését a *Szétkapcsol* gombra kell kattintanunk, ami bontja az eszközzel a kapcsolatot, majd kilép.

3.3.4. Az IRmouseGUI működése

Mielőtt a keret megjelenne, párosítsuk a Wii remote –ot a számítógéppel. Erre egy üzenet hívja fel a figyelmünket: *„A kapcsolódáshoz nyomd meg az 1 & 2 gombokat a Wii Remote - on!”.*

A kapcsolódás lépései ugyanazok mint a TestGUI keretnél, a program meghívja a *wiiconnect()* metódust és létrehoz egy *mote* objektumot. Ha ez hiba nélkül lezajlott, a keret megjelenik.

```
public void wiiConnect() {

        SimpleMoteFinder simpleMoteFinder = new SimpleMoteFinder();
        mote = simpleMoteFinder.findMote();

        if (mote != null) {
            while (mote.getStatusInformationReport() == null) {
                try {
                    Thread.sleep(101);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            System.out.println(mote.getStatusInformationReport());
            mote.setPlayerLeds(new boolean[] {true, false, false, true});

        }
}
```

A IRmouseGUI frame is belülről építkezik. Az IRmouseGUI() konstruktor egymás után *Gridlayout* elrendezésben kirajzolja a paneleket a keretre, és inicializálja azokat.

Ahhoz, hogy használni lehessen a program „Irmouse” funkcióját az előző részben említett kalibrálásra van szükség. Nyomjuk meg a *„Kalibrálás”* gombot.

3.3.4.1. A „Kalibrálás” gomb

A gombhoz egy eseményfigyelő van rendelve, ami kattintáskor az IRListener() metódust hajtja végre. Az eljárás definiál egy *IrCameraListener()* figyelőt az infravörös kamerához, amely a kameraváltozásokat kezeli *irImageChanged()* metódusán keresztül.

A következő metódusrészek már bonyolultabbak. Az algoritmus egyszerre vezérli a kalibrációs folyamatot, majd annak végeztével az egérkurzor mozgását.

```
mote.addIrCameraListener(new IrCameraListener() {
    public void irImageChanged(IrCameraEvent evt) {
    p0 = evt.getIrPoint(0);
    if (newIRdata == false && p0.x == 1023) newIRdata = true;
    if (mouseIsPressed && p0.x == 1023) {
                mouseIsPressed = false;
                mouseRobot.mouseRelease(InputEvent.BUTTON1_MASK);
    }
    if (p0.x != 1023 && newIRdata == true)
        if (calibrator.calibrated == false)
        {
            calibrator.Calimg(p0.x,p0.y);
            newIRdata = false;
            if (calibrator.calibrated) {
                calibrator.setVisible(false);
                calibrate.setEnabled(false);
                CalStatus.setText("Kalibrálva.");
            }
        }
        else
        {
            calibrator.translateCoords(p0.x, p0.y);
            mouseRobot.mouseMove(calibrator.correctX, calibrator.correctY);
            if (!mouseIsPressed)
            {
                mouseRobot.mousePress(InputEvent.BUTTON1_MASK);
                mouseIsPressed = true;
            }
        }
    }
});
```

A *p0* objektum lesz az a pont, amit a Wiimote fénypontként érzékel. A *motej-library IrCameraListener()* metódusa akkor is küld adatot, ha infravörös fénypont nem érzékelhető, ekkor az X pont értéke 1023 lesz. Be kell vezetnünk egy kapcsolót, ami arról informálja az algoritmust, hogy az infratollat felengedtük majd újra kattintottunk, azaz új adatfolyam érkezik. Ennek kezelésére definiáljuk a *newIRdata* logikai (*boolean*) típusú változót.

```
        if (newIRdata == false && p0.x == 1023) newIRdata = true;
```

Ez akkor billen igaz (*true*) állásba ha, volt előtte adatfolyam (*newIRdata == false*) de az

infrapont már megszűnt (*p0.x == 1023*).

A következő szelekció is hasonlót vizsgál csak itt a *mouseIsPressed* változó az egérgomb állapotát tárolja.

```
if (mouseIsPressed && p0.x == 1023) {
            mouseIsPressed = false;
            mouseRobot.mouseRelease(InputEvent.BUTTON1_MASK);
}
```

Azaz, ha az egérgomb állapota lenyomott, de már az X –en 1023 –as értékek jönnek akkor az infrakapcsolót felengedték, vagyis az egérkapcsolót is fel kell engedni.

Ezután egy feltétele elágazás megnézi, hogy jönnek e fénypontadatok az eszközről, vagyis az IR toll be van-e kapcsolva.

```
if (p0.x != 1023 && newIRdata == true)
```

Ez két okból történhet. Vagy a kalibrálás folyamata zajlik, vagy az már megtörtént és az egérkurzort vezéreljük a tollal.

```
if (calibrator.calibrated == false)
{
    calibrator.getCalPoint(p0.x,p0.y);
    newIRdata = false;
    if (calibrator.calibrated) {
        calibrator.setVisible(false);
        calibrate.setEnabled(false);
        CalStatus.setText("Kalibrálva.");
    }
}
```

A *calibrator* az *IRCalibration* osztály egy példánya. Ennek van egy *calibrated* tulajdonsága, ami akkor lesz *igaz*, ha a kalibrációs eredmény már rendelkezésre áll. Ha hamis, akkor még nem történt kalibráció így azt végre kell hajtanunk. A folyamat 4 képpont kijelölését kéri, ezért minden kattintás új adatfolyamot kell hogy eredményezzen, vagyis a *newIRdata* állapotát *false* -ba kell billenteni. Mivel a kalibrációs folyamat az elágazáson belül ér véget ezért újabb elágazással kell ezt ellenőriznünk, és amint teljesül kikapcsolni a kalibrációs felületet, letiltani a kalibrálás gombot, valamint a *CalStatus* –t átírni *„Kalibrálva.”* értékűre.

Ha a kalibráció már lezajlott és fénypontadat érkezik, akkor az elágazás másik része hajtódik végre.

```
else
{
    calibrator.translateCoords(p0.x, p0.y);
    mouseRobot.mouseMove(calibrator.correctX, calibrator.correctY);
    if (!mouseIsPressed)
    {
        mouseRobot.mousePress(InputEvent.BUTTON1_MASK);
        mouseIsPressed = true;
    }
}
```

Ez a beérkező X és Y koordinátákat a *translateCoords()* metódussal átalakítja képernyő koordinákká. A *mouseRobot* egy java AWT Robot osztálypéldány ami az egér vezérlését végzi a grafikus felületen. Ha az IR toll nem volt aktív, de koordinátaadat érkezett akkor ez kattintásnak minősül és a program „megnyomja” a bal egérgombot, és a mouseIsPressed kapcsolót „nyomva tartott” állapotba állítja. A program így le tudja kezelni a duplaklikk eseményt is.

Ezzel le is kezeltük az infrakamera eseményeit, most már aktiválhatjuk azt az eszközön.

```
mote.enableIrCamera(IrCameraMode.BASIC, IrCameraSensitivity.WII_LEVEL_3);
mote.setReportMode(ReportModeRequest.DATA_REPORT_0x37);
try {mouseRobot = new Robot();} catch (Exception e) { e.printStackTrace();}
calibrator.setVisible(true);
calibrator.DrawImg();
```

A kameramód itt is hármas szintű *BASIC* lesz, *0x37* –es adatcsomagokkal. A *Robot()* objektum példányosítását követően elindíthatjuk a kalibrálási folyamatot a kalibrációs képernyő megjelenítésével.

3.3.4.2. Az IRCalibration osztály – Kalibráció

Ennek az osztálynak a feladata, hogy megjelenítse a kalibrációs felületet, kirajzolja a kalibrációs pontokat, majd végrehajtsa a kapott koordinátákon a projektív transzformációt. Az osztály konstruktora előkészíti a felületet, majd értéket ad az *scrDots[]* tömbnek. Ebbe kerülnek a kalibrációs képernyőpontok koordinátái.

Az *IRmouseGUI* a kalibráláshoz a *getCalPoint()* metódust futtatja. Ennek bemenő paraméterei az érzékel képpont X és Y koordinátái. Az algoritmus a kalibrációs pontok meghatározásakor ellenőrzi, hogy a képernyőn megközelítőleg oda kattintottunk-e az IR tollal ahol a kalibrációs

pont kirajzolódott. Egy case szekvenciával ellenőrizzük ezt a kitételt, és az eredményt a pointOK változóban tároljuk.

```
public void getCalPoint(int wiiX, int wiiY) {
    boolean pointOK = false;
    if (calibrated == false) {
        System.out.println(wiiX + ", " + wiiY);
        switch (calPoints){
            case 1:
                if ((calDots[0][0]*1.2) < wiiX && (calDots[0][1]/1.2) < wiiY)
                    pointOK = true;
                break;
            case 2:
                if ((calDots[1][0]/1.2) > wiiX && (calDots[1][1]/1.2) > wiiY)
                    pointOK = true;
                break;
            case 3:
                if ((calDots[2][0]*1.2) < wiiX && (calDots[2][1]*1.2) > wiiY)
                    pointOK = true;
                break;
            default: pointOK = true;
        }
        if (pointOK) {
            setCalPoint(wiiX, wiiY);
            if (calibrated == false){
                calimg.img = Toolkit.getDefaultToolkit().createImage("images/ok.png");
                calimg.repaint();
                DrawImg();
            }
        }
    }
}
```

Ezt az ellenőrzést persze csak abban az esetben kell megtenni, ha a kalibráció még nem futott le azaz a *calibrated* kapcsoló állapota *false.*

Ha *pointOK igaz* vagyis a jó helyre kattintottunk akkor a pont képét kipipáljuk, meghívjuk a *setCalPoint()* eljárást, majd a következő kalibrációs pont képét megjelenítjük.

A *setCalPoint()* feladata, hogy a Wii remote által küldött helyes koordinátákat eltárolja a *calDots[]* tömbben. Inkrementálja a *calPoints* változó értékét, ami azt mutatja, hogy melyik kalibrációs pont következik a sorban. Ezután ellenőrzésre kerül ez az érték, ami ha mind a 4 pont kijelölésre került, akkor az a kalibrált állapot bekövetkezését jelenti. Ekkor a lefut a *calibration()* metódus, ami kiszámítja a ponttranszformációkat és sikeres végrehajtás esetén nullával tér vissza.

```
void setCalPoint(int wiiX, int wiiY)
{

    System.out.println(calPoints);

    calDots[calPoints][0] = wiiX;
    calDots[calPoints][1] = wiiY;

    calPoints++;

    try { Thread.sleep(501); }
    catch (Exception e) {}

    if( calPoints == 4 ){
        if( calibration() == 0 ) this.calibrated = true;
        else calPoints = 0;
    }
}
```

Az *calibration()* egyszerűen működik. A 2. fejezetben leírtakat alkalmazva (*A·x=b*) feltölti az *A* együttható mátrixot és a *b* jobboldali vektort, és ezeken végrehajtja a Gauss-eliminációs eljárást.

```
int calibration()
{
    double [][] A =
    {
        { calDots[0][0],calDots[0][1],1,0,0,0,-calDots[0][0]*scrDots[0][0],-calDots[0][1]*scrDots[0][0] },
        { calDots[1][0],calDots[1][1],1,0,0,0,-calDots[1][0]*scrDots[1][0],-calDots[1][1]*scrDots[1][0] },
        { calDots[2][0],calDots[2][1],1,0,0,0,-calDots[2][0]*scrDots[2][0],-calDots[2][1]*scrDots[2][0] },
        { calDots[3][0],calDots[3][1],1,0,0,0,-calDots[3][0]*scrDots[3][0],-calDots[3][1]*scrDots[3][0] },
        { 0,0,0,calDots[0][0],calDots[0][1],1,-calDots[0][0]*scrDots[0][1],-calDots[0][1]*scrDots[0][1] },
        { 0,0,0,calDots[1][0],calDots[1][1],1,-calDots[1][0]*scrDots[1][1],-calDots[1][1]*scrDots[1][1] },
        { 0,0,0,calDots[2][0],calDots[2][1],1,-calDots[2][0]*scrDots[2][1],-calDots[2][1]*scrDots[2][1] },
        { 0,0,0,calDots[3][0],calDots[3][1],1,-calDots[3][0]*scrDots[3][1],-calDots[3][1]*scrDots[3][1] }
    };

    double [] b = {scrDots[0][0], scrDots[1][0], scrDots[2][0], scrDots[3][0],
                   scrDots[0][1], scrDots[1][1], scrDots[2][1], scrDots[3][1]};

    double[] x = GaussElimination.gauss(A, b);

    a1=x[0]; b1=x[1]; c1=x[2];
    a2=x[3]; b2=x[4]; c2=x[5];
    a3=x[6]; b3=x[7];

    if (Double.isNaN( a1 ) || Double.isNaN( a2 ) || Double.isNaN( a3 ) ||
        Double.isNaN( b1 ) || Double.isNaN( b2 ) || Double.isNaN( b3 ) ||
        Double.isNaN( c1 ) || Double.isNaN( c2 )) return 1;

    System.out.println("Kalibrálás rendben!");
    return 0;

}
```

Erre egy külön osztályt hoztam létre *GaussElimination* névvel. Ez tulajdonképpen egy függvény, aminek a bemenő paraméterei *A* mátrix és *b* vektor, visszatérési értéke pedig az ismeretleneket tartalmazó *x* vektor.

Végezetül ellenőrizzük a kapott vektorértékeket az *isNAN()* függvénnyel, hogy az ismeretlenek nem lettek e hibásak, és amennyiben nem átadjuk a 0 visszatérési értéket.

Ezzel befejeződik a kalibráció és elindulhat az egérmutató infratollal való vezérlése. Az IrmouseGUI keret addig működik a háttérben, amíg a *Szétkapcsol* gombra nem kattintunk.

ÖSSZEFOGLALÁS

A szakdolgozatot alapjául szolgáló program megalkotását sok kutatómunka előzte meg. Tanulmányoztam a már meglévő hasonló projektek megvalósítását és azok eredményességét. Ezekből az derült ki számomra, hogy a szerzők komoly elméleti háttérrel rendelkeznek, ami a néha bizony a megvalósítás rovására ment. Sok felesleges „tudományos", matematikai háttér kihagyható lett volna az implementációkból egyszerűbb gondolkodásmódot követve.

Nos, én éppen erre törekedtem a megvalósítás során. A programlogikát úgy próbáltam felépíteni, hogy az később is érthető, bővíthető legyen.

A tulajdonképpeni képfeldolgozáson alapuló mozgásvizsgálat megvalósításához szükséges program elkészítéséhez a JAVA nyelv használata is a legjobb választásnak bizonyult. Igazán hasznos lehetősége a nyelvnek, hogy szerteágazó eszközkészlettel rendelkezik többek között matematikai, megjelenítési illetve hibakeresési területeken. A dokumentációja bőséges, szintaktikája könnyen elsajátítható. Magának a programozásnak legérdekesebb részét a felület-kalibrálással járó akadályok leküzdése jelentette, melyeket úgy vélem legvégül sikerrel tudtam megoldani.

A dolgozatírás során képet kaphattam az eszköz lehetőségeiről és hiányosságairól is. Sajnos az már a projekt elején kiderült, hogy Wii remote –ot kézbe vételhez tervezték tehát arra, hogy az eszköz a saját mozgását elemezze, ne pedig egy másik objektumét. Ez aztán le is szűkítette a lehetőségeimet. Megítélésem szerint ennek ellenére a Wii remote sok lehetőséget rejt magában csak ötletre és kreativitásra van szükség.

Mindent összevetve úgy gondolom sikerül megvalósítanom a kitűzött célokat amellett, hogy konzekvenciaként levonható, hogy a témának még bőven vannak feltárásra váró területei.

IRODALOMJEGYZÉK

[1] **Mi is az a Bluetooth?**
www.quantumr.hu/plantronics/MiisazaBluetooth.doc

[2] **Bluetooth**
http://hu.wikipedia.org/wiki/Bluetooth

[3] **WIIMOTE**
http://wiibrew.org/wiki/Wiimote

[4] **HUMAN INTERFACE DEVICE (HID) PROFILE**
www.bluetooth.com/SiteCollectionDocuments/HID_SPEC_V10.pdf

[5] **Juhász Imre, Lajos Sándor: Számítógépi grafika**
Miskolci Egyetem Gépészmérnöki és Informatikai Kar
Miskolc, 2007

[6] **Wilhelm Burger, Mark James Burge: Digital Image Processing**
Springer Science+Business Media, LLC 2008

[7] **Kovács Zoltán: A projektív geometria alapjai**
Előadásvázlat, 2003

[8] **Jim Hefferon: Linear Algebra**
ftp://joshua.smcvt.edu/pub/hefferon/book/book.pdf

[9] **I.N. Bronshtein · K.A. Semendyayev · G.Musiol · H.Muehlig: Handbook of Mathematics**
Springer-Verlag Berlin Heidelberg, 2007

[10] **Tóth Bertalan, Benkő László, Benkő Tiborné: Programozzunk C nyelven**
Computerbooks, 1998

[11] **MOTEJ v0.9**
http://motej.sourceforge.net

[12] **Angster Erzsébet:**
Objektumorientált tervezés és programozás, JAVA, 1-2. kötet
4Kör Bt., 2004

KÖSZÖNETNYILVÁNÍTÁS

Köszönettel tartozom témavezetőmnek, Dr. Herendi Tamásnak segítőkészségéért, szakmai hozzáértéséért és tanácsaiért.