Felhasználói felület kialakítása Microsoft Kinect 3D érzékelő segítségével

Pál Gábor

Konzulens: Dr. Vajda Ferenc

Irányítástechnika és Informatika Tanszék

Villamosmérnöki és Informatikai Kar

Budapesti Műszaki és Gazdaságtudományi Egyetem

2013. 05. 20.

Tartalomjegyzék

1. Eddig elért eredmények[1]

Az előző félév során, Önálló laboratórium 1. tárgy keretében kezdtem el tanulmányozni a Natural User Interfaces (NUI) elveket, amelyek a számítógép bármi nemű fizikai beavatkozás nélküli irányítását teszik lehetővé. Tanulmányoztam az ezen elveket megvalósító, szenzorvezérelt szoftver- és hardverkörnyezeteket, külön kitérve a számítógéphez, illetve a különböző konzolokhoz készített változatokra. A kutatások alapján részletesebben kezdtem el foglalkozni a Kinecttel, valamint a programozását lehetővé tevő egyik legelterjedtebb keretrendszerrel, az OpenNI-jal, valamint a rá épülő modulokkal. Részletesen ismertettem a Kinect összes szenzorjának lehetőségeit, valamint ezek méréstartományát és korlátait. Foglalkoztam az OpenNI keretrendszerrel, az abban rejlő lehetőségekkel és korlátokkal, részletes leírást adtam a keretrendszer telepítésére Linux rendszer alá (a teljes telepítés részletes leírása - a dependenciákkal együtt - egészen eddig nem volt megtalálható az interneten, még az OpenNI weboldalán sem). Áttekintettem az összes jelentősebb keretrendszert, amely lehetővé teszi a Kinect programozását, ezek előnyeit és hátrányait összevetettem az OpenNI tulajdonságaival. Részletesen bemutattam az OpenNI felépítését, a programok készítésének menetét a keretrendszer segítségével.

A megszerzett ismeretek segítségével elkészítettem két példaprogramot, amely szemlélteti az OpenNI környezet használatának alapjait. Az első a Kinect mélység érzékelő szenzorja segítségével követi a felhasználó kezének bal-jobb irányú mozgását az eszköz előtt, valamint automatikusan érzékeli a kéz érzékelési tartományból való kimozdulását, valamint a visszatérését a tartományba. Ez a program HandSlider néven megtalálható a git repoban (a git reporól részletesebben a 4. bekezdés szól).

A második példaprogram egy mélységi mátrix (ennek mérete a mélység érzékelő szenzor aktuális konfigurációjától függ, alapértelmezésben 640*480 képpont) középső elemének (azaz az érzékelt tartomány közepének) távolságát vizsgálja a szenzortól, valamint közli a felhasználóval a szenzortól való távolságát. valamint a másodpercenként rögzített képkockák számát. Ez a program a git repo MiddlePoint mappájában található meg.

Az említett eredmények részletes leírása az Önálló laboratórium 1. című tárgyhoz elkészített beszámolómban olvasható, amely megtalálható a git repo docs nevű mappájában.

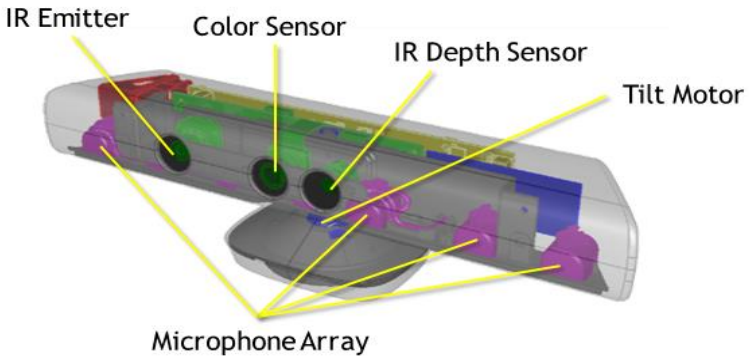
Jól látható, hogy az Önálló laboratórium 1. tárgy során végzett munkám inkább szólt a kutatásról és a lehetőségek megismeréséről, míg az Önálló laboratórium 2. alatt kevesebb kutatási munka mellett több programozási feladatot végeztem el.

2. Kinect[1]

Az alábbiakban rövid áttekintést adok a Kinect felépítéséről azok számára, akik először találkoznak az eszközzel. Részletesebb leírás az Önálló laboratórium 1. tárgyhoz készített beszámolómban található.

1. ábra: Kinect for Xbox

Az eredetileg Project Natal néven futó technológiát 2010. harmadik felében mutatta be a Microsoft. Az első változat Xbox-hoz készült, majd a különböző PC-s „törések" terjedésének hatására a Microsoft előbb kiadta a hivatalos SDK-t (Software Development Kit) PC-re való alkalmazások fejlesztéséhez, majd 2012. elején megjelent a Kinect for Windows változat is. Ez pontosabb szenzorokat és jobb SDK-t tartalmaz, viszont a Microsoft teljesen megváltoztatta az irányítás és adatgyűjtés módját, ezért a nyílt fejlesztőeszközök nagy részével nem kompatibilis.



2. ábra: A Kinect felépítése

A 2. ábrán látható az eszköz felépítése és a benne található szenzorok. Az IR Emitter infravörös tartományú fényt bocsájt ki, amely visszaverődve a különböző tárgyakról, és az emberi testről az IR Depth Sensor-ban (ez egy monokróm CMOS infravörös szenzor) kerül feldolgozásra. Ebből a szenzorból lehet kinyerni a mélységi adatokat, ami talán a leggyakrabban használt része a Kinectnek. Az IR Depth Sensor a képpontokat távolsági adatokká konvertálja át, így ezek az adatok később mélységi mátrixként (vagy mélységi

térképként) is felhasználhatóak. A szenzor megbízható érzékelési tartománya 50 cm-től 3 méterig terjed (a valós, ámde néha bizonytalan érzékelési tartomány 50 cm-től kb. 5 méterig terjed). A szenzor érzékelési pontossága mélységi adatokra (z tengely) 1 cm, míg szélességi és magassági adatokra (x és y tengely) milliméterekben mérhető. Az érzékelési pontosság a távolság változásával nem lineáris arányban változik, erről egy bővebb cikk itt olvasható: http://mathnathan.com/2011/02/depthvsdistance/. Az IR Depth Sensor által továbbított adatfolyam 640x480 pixel felbontású, és 11 bites távolsági adatokat tartalmaz, ezáltal 2048 db különböző távolsági adat ábrázolható. Az adatfolyam maximális frekvenciája 30 Hertz, de ez szoftveresen konfigurálható. Az IR Depth Sensor látószöge vertikálisan (x tengely) 43°, míg horizontálisan (y tengely) 57°. Ez utóbbi a beépített motor segítségével tovább javítható.

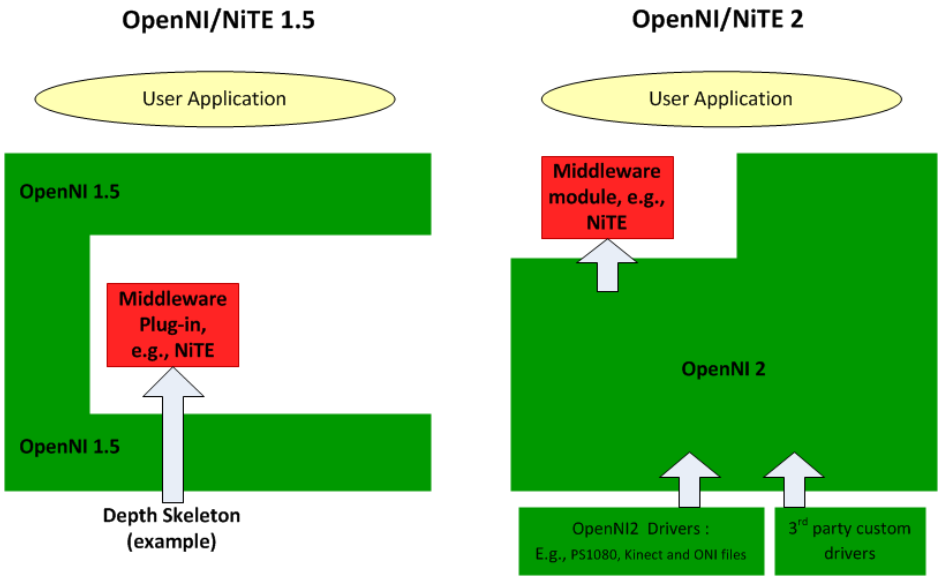
Az eszközben található Color sensor egy átlagos webkamerához hasonló teljesítményt nyújt a maga 640*480 képpontos felbontásával.

A 4 db mikrofon feladata a hangok rögzítése és továbbítása a konzol vagy a számítógép felé. A mikrofonok 16 kHz-es hangmintákat rögzítenek PCM (Pulse Code Modulation) kódolással.

Található még az eszközben egy, az ábrán nem látható status LED, amely villogással jelzi, ha az eszköz áramellátás alatt van, és konstans módon világít, ha a szenzor használat alatt is van (ez azt jelenti, hogy a Kinect valamely szenzora éppen adatokat továbbít a konzol vagy a PC felé). Érdekes tulajdonság, hogy a LED színe és villogási frekvenciája szoftveresen állítható, és a Kinect egészen addig megőrzi az adott állapotot, amíg új utasítást nem kap a LED állapotának változtatására (azaz ha egy program pl. piros színűre változtatja a LED-et, akkor a program befejezése után is piros színű marad a LED, amíg valaki vagy valami meg nem változtatja azt).

Ahogy az 1. ábrán is látható, a Kinectet nem csak a számítógép vagy az Xbox USB portjához kell csatlakoztatni, de külső áramellátásra is szüksége van. A Kinect maximális fogyasztása 12 watt, és ez jóval több az USB szabványban rögzített 2.5 wattos maximális áramfelvételnél, amely egy USB portra kötött eszköztől elvárható lenne.

3. OpenNI2[2][3][4][5][6]



3. ábra: Az OpenNI 1.5 és a 2.0 közti felépítésbeli különbség

2012 decemberében az OpenNI közösség kiadta az OpenNI SDK 2.0 változatát. Ez nem sok újdonságot tartogatott a felhasználók számára:

- Refaktorálták az API-t, azaz az algoritmusok gyorsabban futnak az 1.x változatokhoz képest
- A nem közvetlenül a bejövő adatok feldolgozásával foglalkozó osztályok átkerültek a Middleware rétegbe. Ilyenek többek között a magasabb szintű kódok írását elősegítő osztályok (pl. XnList, XnHash, XnBitSet, XnThreadSafeQueue)
- A 2.0-s változatban már közvetlenül is elérhetőek a beépülő Middleware komponensek, nem csak közvetett módon az OpenNI-on keresztül

Ezeken felül még néhány apróbb módosítás is belekerült a kódba többet között az alkalmazások fordításával kapcsolatban, de ezek nem befolyásolják jelentősen az API használatát.

A Kinect felhasználók számára viszont jelentős változás történt. A fórumok tanúsága alapján igazából emiatt volt szükség az új változat kiadására, és nem a fenti néhány módosítás miatt. A Microsoft rosszallóan tekintett az OpenNI közösségre, ugyanis hivatalosan terjesztette a Kinect olyan driverjét, amihez a Microsoft nem járult hozzá. Jelenleg a Microsoft a Kinect for Windowst támogatja hivatalosan, a Kinect for Xbox eszköz használatát pedig „eltűri” PC-s környezetben, viszont mindkettőt csak a saját SDK-jával és Windows operációs rendszer felett. Mivel az OpenNI 1.x saját drivert használt a Kinect minden szenzorához, így nem teljesítette a fenti feltételt. Az OpenNI 2.0-ban kidobták ezt a drivert (SensorKinect néven található róla bővebb leírás az Önálló laboratórium 1-hez készített beszámolómban), és a Microsoft hivatalos SDK-ja szükséges a Kinect használatához OpenNI felett. Ez azt jelenti,

hogy OpenNI 2.0-val megszűnt a Kinect nem Windows platformon történő támogatása. Mivel az eddigi munkám és a további tervek is a Linux platformhoz kötnek, ezért számomra vállalhatatlan volt ez a váltás, így maradtam az OpenNI 1.5 használatánál.

Mellékesen jegyzem meg, hogy az ismert hibák sokasága és a néhány fórumon leírtak alapján ez nem feltétlenül csak ezért volt jó döntés.

Mégsem teljesen reménytelen a helyzet azok számára, akik nem Windows felett szeretnék használni a Kinectet OpenNI 2.0-val. Elkészült ugyanis egy OpenNI2-FreenectDriver nevezetű szoftver az OpenKinect közösség jóvoltából, amely lehetővé teszi a Kinect használatát Linux platformon OpenNI2-vel. Ez hivatalosan nem az OpenNI 2.0 része, és hivatalosan nem is támogatott (inkább amolyan hobbiprojektként fut), viszont teljesen jól használható OpenNI 2.0-val Linux platform felett.

4. Finger detection vizsgálata[7][8][9]

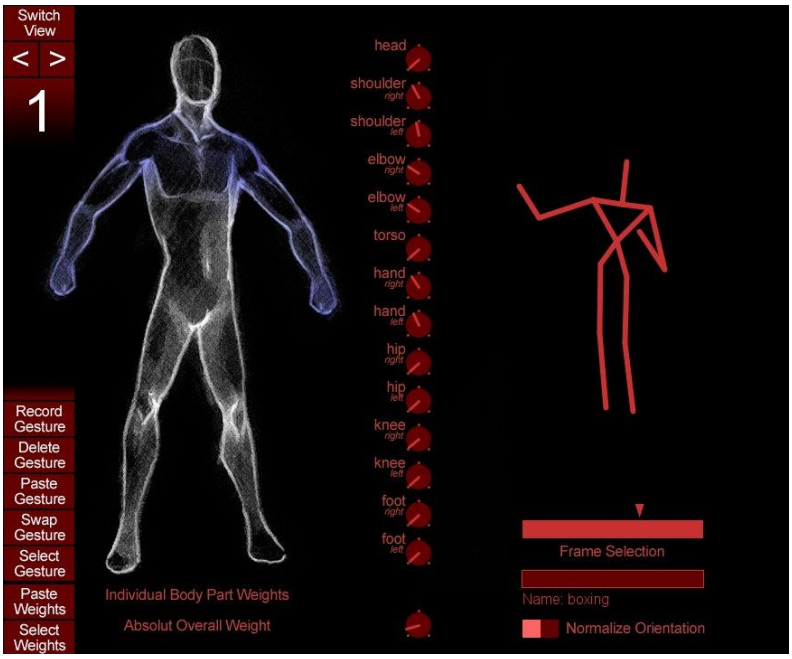
A félév elején-közepén a szenzor előtt mozgó kéz ujjainak felismerésével kezdtem el foglalkozni. Ehhez több függvénykönyvtárat is megvizsgáltam:

- Finger tracking MW (PrimeSense)
- FingerTracker (Java)
- Finger-precise tracking (OpenNI market-ready solution)
- Candescentport

A fentiek közül egyikkel sem sikerült jó eredményt elérnem. A PrimeSense megoldása látszólag jól használható lenne, viszont csak fizetős változatban érhető el. A FingerTracker egy Java alapú megoldás, előnye az egyszerűsége, viszont C++ kódhoz illeszteni elég nehézkes lett volna. Az OpenNI megoldása a PrimeSense-hez hasonlóan szintén csak fizetős alkalmazásként érhető el. A Candescentport egy diplomamunka keretében készült függvénykönyvtár, amely C++-ban OpenNI fölé íródott. Áttekintve azonban a dokumentációt, a dependenciák nagy része csak Windows felett érhető el, így sajnos ez a megoldás sem használható Linux felett.

Számos egyéb lehetőséget is áttekintettem, és arra jutottam, hogy a jelenleg adott feltételek mellett (OpenNI 1.5, Linux) túl sok munkát igényelne részletesen feldolgozni egy kész függvénykönyvtárat, esetleg használható programokat is írni rá, így más irányok mentén folytattam a munkát.

5. Kineticspace[10]



4. ábra: Kineticspace

A Kineticspace az OpenNI 1.x-re és NITE-ra épülő szoftver, ami lehetővé teszi a Kinect (továbbá az Asus Xtion) különböző szenzorainak tesztelését, valamint a NITE-ban megvalósított mozgás formációk (pl. integetés, ütés, bólogatás, stb.) követését. A szoftver lehetővé teszi különböző mozgássorok rögzítését, ezáltal a program tanítható is. Egy komplexebb mozdulatsor rögzítése után megpróbálhatjuk újra végrehajtani ugyanazt a mozdulatsort, és a program kiértékeli, hogy sikerült-e ezt pontosan megtennünk.

A Kineticspace felhasználó független, azaz az egyik felhasználó által rögzített mozdulatsorokat ugyanúgy fel tudja ismerni bármely másik felhasználó által végrehajtva is. További fontos tulajdonság, hogy ezek a mozdulatsorok függetlenek az időbeliségtől, azaz mindegy milyen gyorsan hajtjuk végre őket.

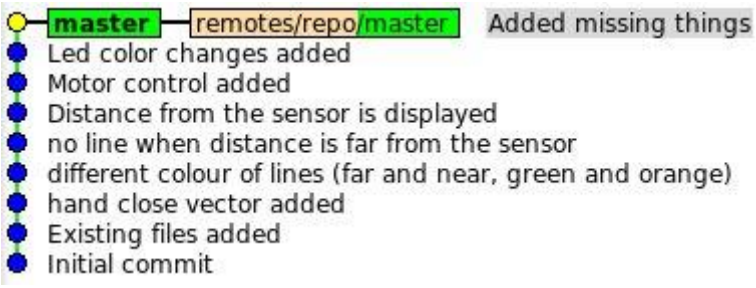
Habár kiválóan alkalmas a Kinect diagnosztikai vizsgálatára (én is nagyrészt erre használtam a munkám során), jelenleg nagyrészt előadóművészek és tánctanárok használják mozdulatsorok gyakorlására és oktatásra.

6. Git

Az általam írt kódok és dokumentációk sokasága és széttagoltsága szükségessé tette egy verziókezelő rendszer használatát. Ehhez a Git rendszert használtam fel, amely a nyílt forráskódú szoftverek világában manapság a legelterjedtebb és a legkönnyebben használható verziókezelők egyike. Az általam készített munkákat a jól ismert Github-ra töltöttem fel, ahonnan akár egy web böngésző segítségével is olvasható a kód és a hozzá tartozó dokumentáció.

A Git repository az alábbi címen érhető el: https://github.com/palgabor/Onlab--Kinect- . Jelenleg az alábbi mappákat tartalmazza:

- docs: dokumentációs fájlok, félév végi beszámolók, prezentációk
- DrawGame: a 9. fejezetben részletezett rajzolós játék
- HandSlider: az előző félév során írt, 1. fejezetben részletezett program
- MiddlePont: az előző félév során írt, 1. fejezetben részletezett program
- MotorMoving: a 7. fejezetben részletezett program
- NITELib: a NITE middleware komponens által biztosított header fájlok, amelyek szükségesek az alkalmazások fordításához
- NiHandTracker: OpenNI példaprogram másolata, a DrawGame és a ReactionGame programok alapja
- OpenNILib: az OpenNI keretrendszer által biztosított header fájlok, amelyek szükségesek az alkalmazások fordításához
- ReactionGame: a 8. fejezetben részletezett program



5. ábra: A repository egy részlete

7. Motor és LED vezérlése OpenNI alól[11]

A 2. fejezetben leírtak alapján a Kinectben található egy státusz RGB LED, amely visszajelzést ad a szenzor aktuális állapotáról, valamint egy motor, amely a szenzor vertikális mozgását teszi lehetővé, így bővítve az y tengely menti látószöget. Ezek használatát a fejlesztői környezetek egy jelentős része nem támogatja, többek között az OpenNI sem (és a Windows hivatalos SDK-ja sem). Mégis van lehetőség az OpenNI alóli használatukra, viszont ehhez kicsit hozzá kell nyúlni az OpenNI forráskódjához.

A már feltelepített OpenNI forráskódja megtalálható a `~/kinect/OpenNI/Source/OpenNI/Linux/` könyvtár alatt. Itt az `XnUSBLinux.cpp` fájlban kell módosítani a 761. és a 819. sort, a `return (XN_STATUS_USB_WRONG_CONTROL_TYPE);` helyett a `bmRequestType = nType;` értékadást bemásolni. Ezután az alábbi parancsok végrehajtásával újra kell fordítani az OpenNI libraryt:

```
cd ~/kinect/OpenNI/Platform/Linux/CreateRedist/
chmod +x RedistMaker
./RedistMaker
```

```
cd ../Redist/OpenNI-Bin-Dev-Linux-x86-v1.5.2.23/
sudo ./install.sh
```

Az OpenNI fölé írt alkalmazások újrafordítására nincs szükség. Ezután a „hackelés” után már az OpenNI-t használó alkalmazások is tudják vezérelni a Kinect motorját és LED-jét. Az OpenNI újrafordítása után szükség lehet az operációs rendszer újraindítására (Linux esetén is), ez egy ismert, ám eddig nem kijavított probléma.

A git repositoryban a MotorMoving könyvtár alatt található egy példaprogram a motor és a led használatára, valamint a 9. fejezetben részletezett Draw game is használja ezeket a szolgáltatásokat.

8. Reaction game

A félév közepén-végén két darab nagyobb volumenű programot (játékot) készítettem el, törekedve a Kinect minél sokoldalúbb használatára, valamint a téma által leírt felhasználói felület megvalósítására.

8.1. Felhasználói útmutató

A Reaction game egy észlelési reakciókat tesztelő játék, ahol a képernyőn véletlenszerűen felvillanó kék négyzeteket kell minél hamarabb „elkapni”, míg a véletlenszerűen felbukkanó piros négyzeteket el kell kerülni. A program számolja a kék (jó) és a piros (rossz) négyzetek „elkapásának” számát, valamint az átlagos reakcióidőt.

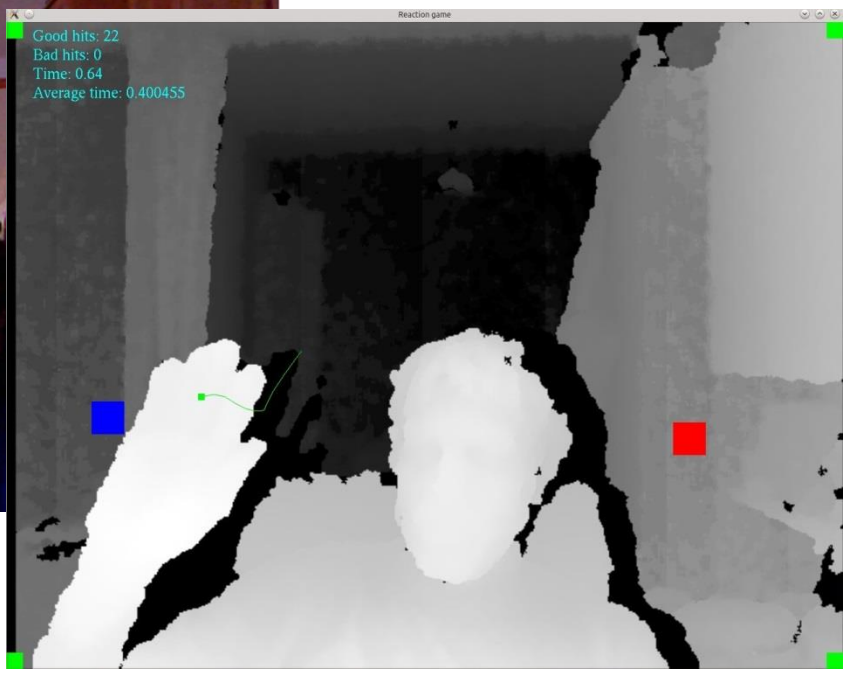
A program fordítása a ReactionGame főkönyvtárában a `make` parancs kiadásával lehetséges, amely a `ReactionGame/Makefile`-ban leírtak szerint lefordítja a programot (Linux platformra), és elhelyezi a binárist a bin könyvtárban. Ezután a futtatás (a Config.xml miatt fontos, hogy szintén a főkönyvtárból történjen) az alábbi paranccsal történik: `./bin/ReactionGame`.

A játék indítása után a VGA kamera képét látjuk a képernyőn, ezt az 1-2-3 gombokkal tudjuk váltani a mélységi képre, valamint a VGA-mélységi kép egymásra vetítésére. Az m billentyű megnyomásával pedig a kép tükrözését tudjuk ki-be kapcsolni. A képernyő négy sarkában 1-1 piros négyzetet látunk, ezek azt jelölik, hogy jelenleg nem aktív a játék. A játék indításához integetni kell a szenzornak, hogy azonosítani tudja a kezet. A játék jelenlegi változatában csak egy kézzel lehet játszani (erre a megszorításra több okból is szükség volt). Az integetés után a program egy pontot rajzol az azonosított kézre, valamint 30 db apró pontot is húz a kéz vonala után, hogy a felhasználó számára könnyebben követhető legyen a kéz által megtett út. Amint a program érzékelte a kéz jelenlétét, máris megjelenít 1-1 piros és kék négyzetet, valamint a bal felső sarokban az eredményjelzőt, továbbá a négy sarokban a státuszjelző négyzetek zölddé válnak. A játékban a kék négyzetek egészen addig a helyükön maradnak, amíg el nem kapják őket, míg a piros négyzetek 1 és 10 másodperc közötti véletlen idő után eltűnnek, és új helyre kerülnek. Ha véletlen egy piros négyzet kerül elkapásra, az akkor is eltűnik. A program számolja a legutolsó kék négyzet elkapása óta eltelt időt, valamint az átlagos reakcióidőt is (ez

a kék négyzetek elkapása közt eltelt idő). Egy piros négyzet elkapása nem változtatja meg az óra állapotát.

A játék befejezéséhez elég a kezet a szenzor hatáskörén kívülre vinni, ilyenkor eltűnik a kézfejről a pont, valamint a státuszjelző négyzetek a sarkokban ismét pirosra váltanak. Ekkor egy újbóli integetéssel újrakezdhető a játék (minden érték nullázódik), vagy kiléphetünk a játékból az ablak bezárásával.

6. ábra: A Reaction game VGA képe

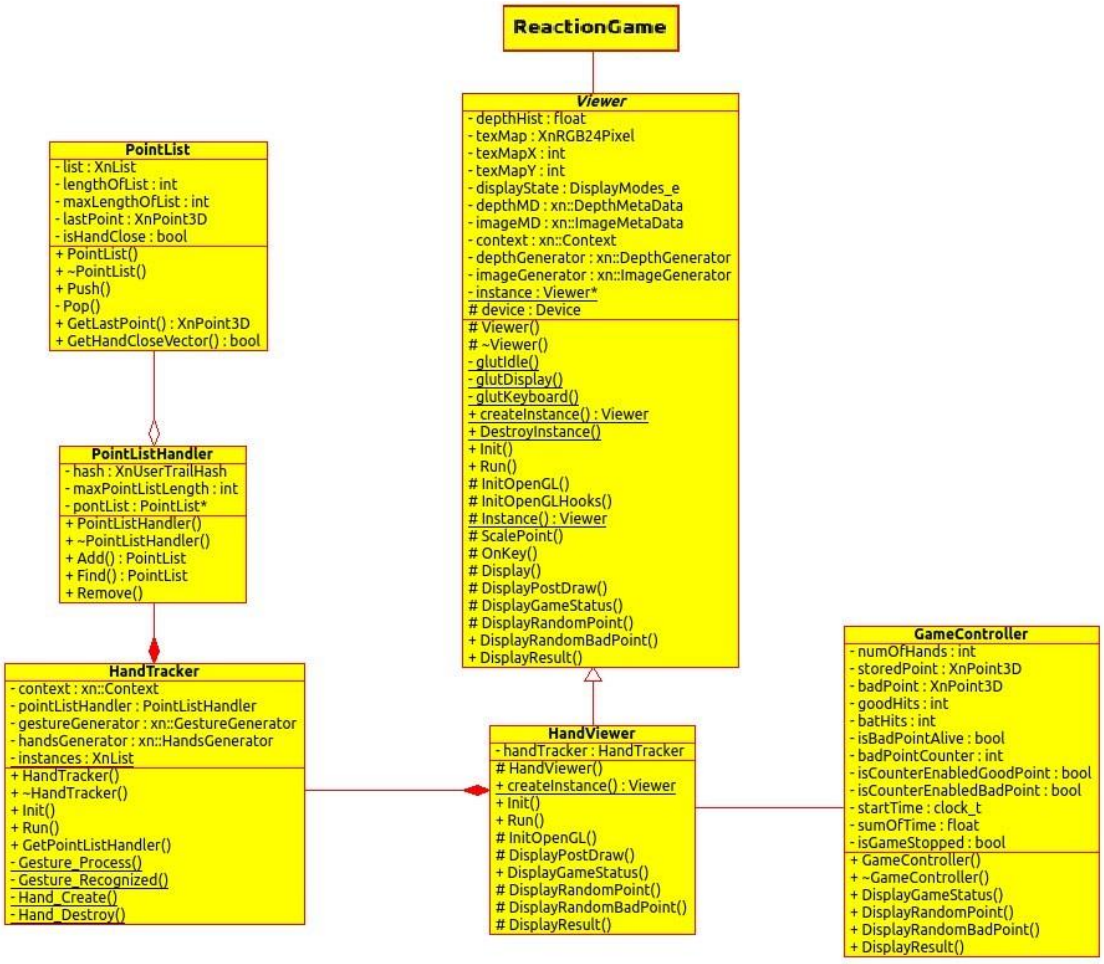


7. ábra A Reaction game mélységi szenzor képe

8.2. Fejlesztői útmutató

A Reaction game az OpenNI NiHandTracker nevű példaprogramjára épül, innen származik a megjelenítés, továbbá a kéz követésének algoritmusa. A program C++ nyelven íródott, és a fordításhoz standard gcc fordító szükséges. A megjelenítés és az alakzatok kirajzolása az OpenGL függvénykönyvtár segítségével történik, amely ingyenesen elérhető bármely elterjedt platform alá.

A program osztálydiagramja az alábbi ábrán látható:



8. ábra: A Reaction Game osztálydiagramja

Az ábrán látható osztályok leírása:

- **ReactionGame**: a program fő osztálya, ez tartalmazza a main függvényt, inicializálja a HandViewer és a Viewer osztályokat.
- **Viewer**: absztrakt osztály, amely biztosítja a megjelenítést, itt vannak implementálva az OpenGL egyes szükséges függvényei, továbbá tartalmaz felüldefiniálandó függvényeket (Display kezdetűek), amelyek különböző objektumok megjelenítését teszik lehetővé az ablakban. Ezen függvények szerepei:
    - DisplayPostDraw: A kézen a pontot, valamint a kéz mozgásának történetét mutatja.
    - DisplayGameStatus: A játék állapotát jelző státusnégyzeteket mutatatja a képernyő négy sarkában.
    - DisplayRandomPoint: A véletlenszerűen megjelenő „jó” pont megjelenítéséért felelős.
    - DisplayRandomBadPoint: Hasonló az előző függvényhez, viszont ez a rossz pontokat jeleníti meg.
    - DisplayResult: A képernyő jobb felső sarkában lévő eredményt jeleníti meg.

- **HandViewer**: A Viewer absztrakt osztály függvényeit valósítja meg, valamint tartalmazza a HandTracker osztályt.
- **HandTracker**: A kéz követéséért felelős osztály, amely callback függvényekkel rendelkezik, amiket az OpenNI meghív a kéz pozíciójának változásakor. Ezek a callback függvények hívják meg a HandViewer megfelelő függvényeit. Tartalmazza továbbá a PointListHandler osztályt.
- **PointListHandler**: Egy map struktúrát valósít meg, ahol az egyes kezekhez tárolja a megtett út koordinátáit. A megtett út legutolsó koordinátája a kéz aktuális helyzete. Ennek az értéke kerül összevetésre a négyzetek koordinátáival az „elkapás" ellenőrzésekor.
- **PointList**: A megtett út koordinátái. Ehhez a programhoz nem lenne feltétlenül szükséges tárolni a megtett utat, viszont a program készítése során megpróbáltam generálisan megközelíteni a problémát, és egy olyan megoldást készíteni, amit a későbbi programok során is változtatás nélkül lehet használni.
- **GameController**: A megfelelő számításokat elvégző osztály, amely a számítások elvégzése után az adatokat továbbadja a HandViewer osztálynak az alakzatok megfelelő megjelenítéséhez.

9. Draw game

A második, félév során részben elkészült program a Draw game nevű játék.

9.1. Felhasználói útmutató

A Draw game-ben kezünk segítségével rajzolhatunk a képernyőre különböző alakzatokat anélkül, hogy ehhez bármilyen beviteli eszközt igénybe kellene vennünk (kivéve természetesen a kezünket figyelő szenzort). Egyszerre akár több kézzel is rajzolhatunk (egy időben maximum 8), de tapasztalataim alapján 3-nál több kéznél már áttekinthetetlen a képernyő. A szenzor viszonylag pontosan érzékeli a kezek megtett útját, viszont a kéz anatómiai tulajdonságait figyelembe (például a kéz remegését, vagy a levegőben egyhelyben tartás nehézségét) nem várható el teljesen pontos adatbevitel.

A program fordítása a DrawGame főkönyvtárában a `make` parancs kiadásával lehetséges, amely a `DrawGame/Makefile`-ban leírtak szerint lefordítja a programot (Linux platformra), és elhelyezi a binárist a bin könyvtárban. Ezután a futtatás (a Config.xml miatt fontos, hogy szintén a főkönyvtárból történjen) az alábbi paranccsal történik: `./bin/DrawGame`.

A játék indítása után a VGA kamera képét látjuk a képernyőn, ezt az 1-2-3 gombokkal tudjuk váltani a mélységi képre, valamint a VGA-mélységi kép egymásra vetítésére. Az m billentyű megnyomásával pedig a kép tükrözését tudjuk ki-be kapcsolni (a tükrözés kikapcsolásakor a kéz mozgásával párhuzamos rajzolás iránya is ellentétére változik). A képernyő négy sarkában 1-1 piros négyzetet látunk, ezek azt jelölik, hogy jelenleg nem aktív a játék. A játék

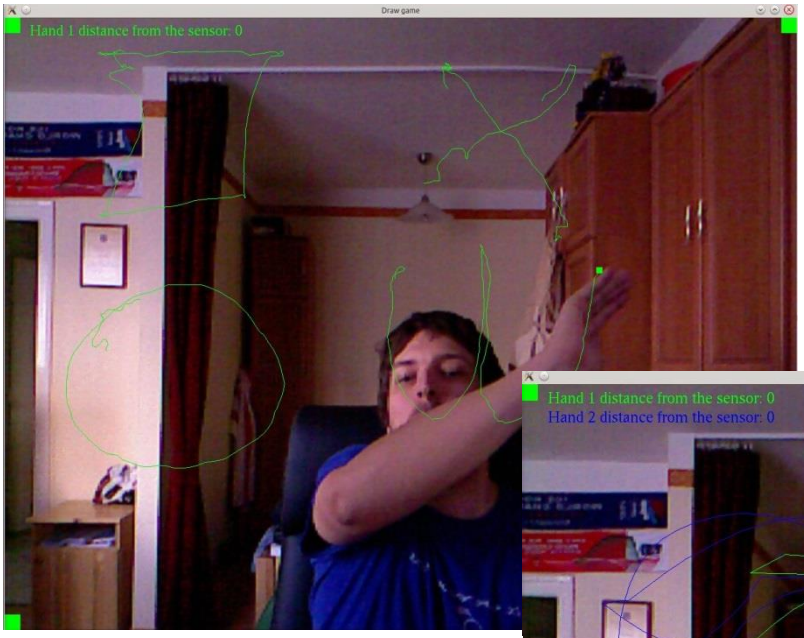
indításához integetni kell a szenzornak, hogy azonosítani tudja a kezet. Az integetés után a program egy pontot rajzol az azonosított kézre, amennyiben az 1 méternél közelebb található a szenzorhoz. Ez egyben a rajzolás megkezdését is jelenti. Amennyiben a szenzortól 1 méternél nagyobb távolságban integetünk, úgy a játék kezdetéről képernyő négy sarkában elhelyezett státuszjelző négyzet pirosról zöldre váltásával tájékozódhatunk. Célszerű ez utóbbi megoldást választani, hogy az integetés során rajzolt alakzat ne legyen része a végleges alakzatnak. A játék elindulását a Kinect szenzoron lévő LED-en is láthatjuk. Amíg a játék nem aktív, addig a LED pirosan világít, ha a kéz az 1 méteres tartományon belül van, akkor zölden, ha pedig a kéz 1 méteres tartományon kívül van, akkor narancssárgán. Több kéz esetén ha valamelyik az 1 méteres tartományon kívülre kerül, a LED narancssárgára vált. Az 1 méteres tartományon kívülre kerüléskor a program nem rajzolja ki az adott pontokat (habár tárolja azokat, de megjelöli őket, hogy nincs szükség a kirajzolásukra), így lehetőség van nem összefüggő alakzatokat is rajzolni.

A program a képernyő bal felső sarkában közli a felhasználókkal, hogy az egyes kezek milyen messze találhatóak a szenzortól. Ha 1 méteres távolságon belül találhatóak, akkor ez az érték 0, ha 1 méteres távolságon kívül, akkor pedig annyi centiméter, ahány centiméterrel kívül esik az 1 méteres tartományon. Ennek segítségével finomabb mozgásokat lehet végezni a szenzortól távolodva vagy közeledve. Minden játékban lévő kéz külön színt kap, ilyen színű pont található az azonosított kézen, ilyen színnel jelenik meg az általa rajzolt alakzat, valamint ezzel a színnel kerül kiírásra a szenzortól való távolsága, így több kéz esetén is jól figyelemmel követhető, hogy melyik kéz milyen messze van a szenzortól.

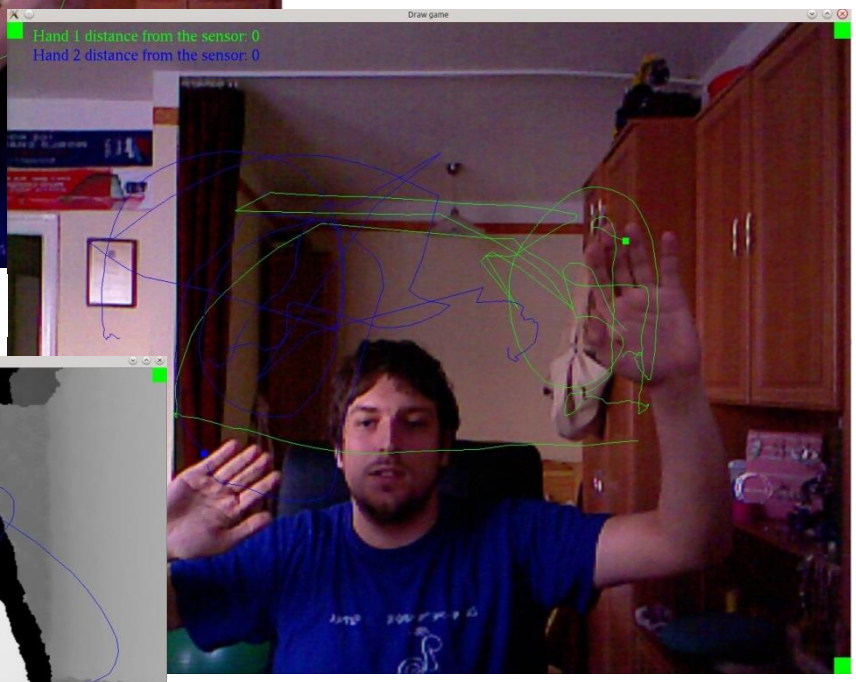
Azon felhasználók számára, akiknek túl közeli vagy távoli az 1 méteres határ, lehetőségük van ezen érték megváltoztatására. Ehhez a `DrawGame/source/Constants.h` fájl 30. sorában kell a `static const unsigned HAND_CLOSE = 1000;` értéket 1000-ről a kívánt mennyiségre átírni. Ez a mennyiség centiméterben értendő. A kívánt mennyiség megadása után a `DrawGame` könyvtárba visszalépve kell kiadni a `make` parancsot, majd a `./bin/DrawGame` parancs kiadása után a játék már az új értékkel indul el.

A szenzor állása a program futása során bármikor változtatható. Ezeket a q és az a billentyűk lenyomásával lehet megtenni. A q billentyű 1°-kal feljebb, az a billentyű 1°-kal lejjebb mozgatja a szenzort. A program az indulása során a szenzort mindig a kezdeti állapotba (0°) mozgatja vissza. A mozgatás -40° és +40° között történhet, amely nem befolyásolja a már megrajzolt alakzatokat (sőt, egy kis trükkel vertikális irányban akár a szenzor mozgatásával is lehet rajzolni, ugyanis a megrajzolt alakzat a szenzorral együtt mozog).

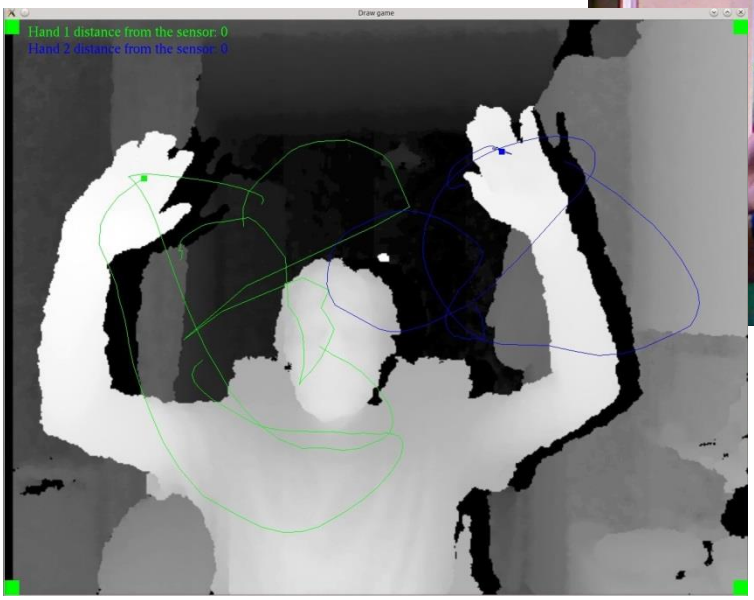
A játék befejezéséhez elég a kezet a szenzor hatáskörén kívülre vinni, ilyenkor eltűnik a kézfejről a pont, valamint a státuszjelző négyzetek a sarkokban és a Kinecten található LED ismét pirosra váltanak. Ekkor egy újbóli integetéssel újrakezdhető a játék, vagy kiléphetünk a játékból az ablak bezárásával. A játék újrakezdésével az összes addig rajzolt alakzat eltűnik. Ha csak egy kéz fejezi be a rajzolást, akkor természetesen csak a saját alakzatai tűnnek el, míg a többi játékban lévő kéz összes alakzata megmarad.



10. ábra: A Draw game egy kezes rajzolása



9. ábra: A Draw game két kezes rajzolása

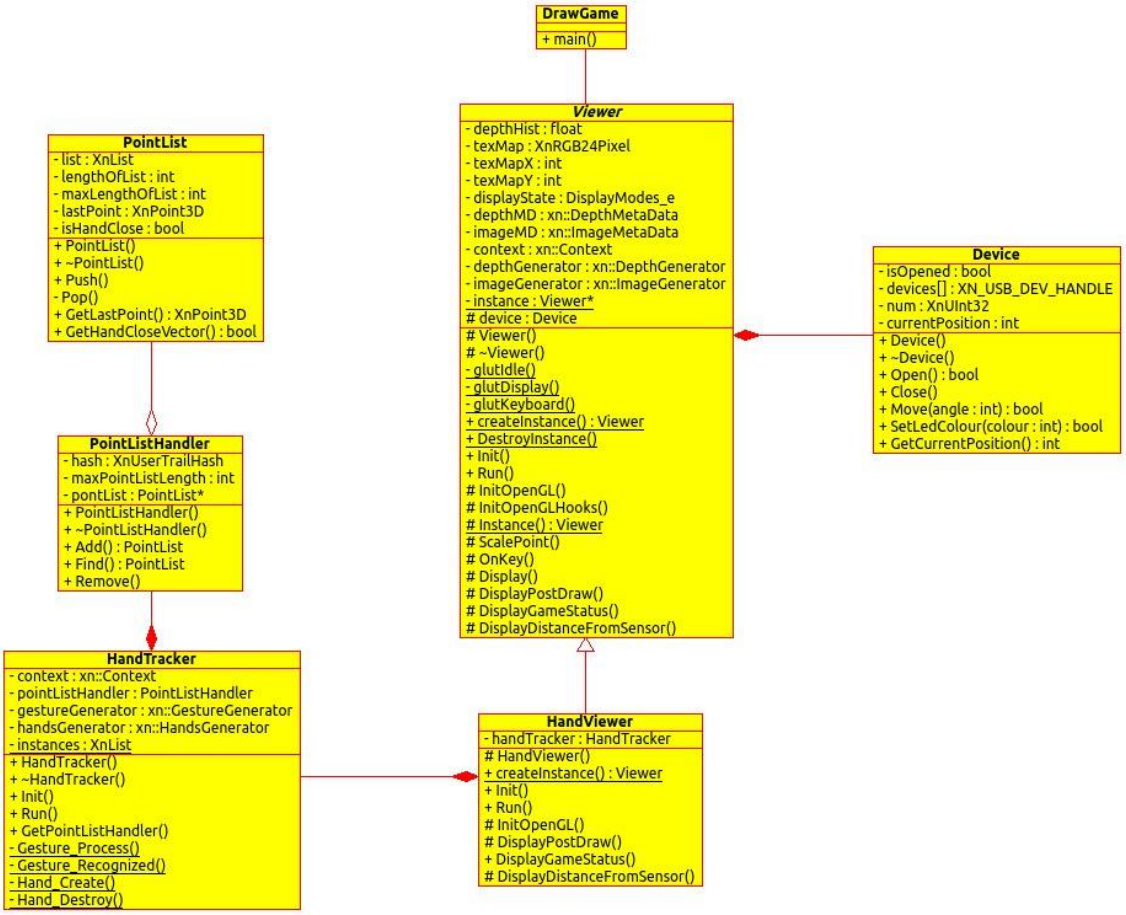


11. ábra: A Draw game két kezes rajzolása mélységi képpel

9.2. Fejlesztői útmutató

A Draw game hasonló felépítésű, mint az előző pontban tárgyalt Reaction game, az alap adatszerkezetei ugyanazok. Ezt a nagyfokú újrafelhasználhatóságot a Reaction game adatszerkezeteinek gondos tervezése tette lehetővé. Természetesen a Draw game is az OpenNI NiHandTracker nevű példaprogramját használja alapul, innen származik a megjelenítés, továbbá a kéz követésének algoritmusa. A program C++ nyelven íródott, és a fordításhoz standard gcc fordító szükséges. A megjelenítés és az alakzatok kirajzolása az OpenGL függvénykönyvtár segítségével történik, amely ingyenesen elérhető bármely elterjedt platform alá.

A program osztálydiagramja az alábbi ábrán látható:



12. ábra: A Draw game osztálydiagramja

Az ábrán látható osztályok felépítése nagyvonalakban megegyezik a Reaction Game-ben leírtakkal, így ezt itt nem írnám le újra. Az egyetlen különbség a **Device** osztály, amely a Kinect szenzor motorjának mozgatását, továbbá a benne található LED állapotának változtatását teszi lehetővé. Ezen két funkció külön osztályba szervezését az tette indokolttá, hogy ezen funkciók nem támogatottak az OpenNI által, így (a 7. fejezeben leírt „hackelés mellett") további inicializáló műveletekre van szükség a használatukhoz. Ezek a függvények biztosítják az OpenNI XnUSB függvényéhez való hozzáférést, ezzel pedig alacsonyszintű USB utasítások küldését a szenzornak.

A **PointList**, a **PointListHandler** és a **HandTracker** osztályok teljes mértékben megegyeznek a Reaction game-ben használt osztályokkal, míg a **Viewer** és a **HandViewer** osztályok tartalmaznak néhány különbséget, mivel ezek az osztályok valósítják meg az egyes alakzatok képernyőre való rajzolását.

A program lehetőséget ad a továbbfejlesztésre, ehhez a **HandViewer** osztályból kinyerhetőek a rajzolt objektumok tárolt pontjai, valamint új információk is kirajzolhatóak a képernyőre, ehhez a **HandViewer** jól használható interface-ként.

10. Alakzat felismerési lehetőségek[12]

A 9. fejezetben tárgyalt Draw game program ugyan jelen körülmények között is teljesen működőképes, azonban teljes értékű akkor lenne, ha a rajzolt alakzatokat a program megpróbálná azonosítani, és az azonosítás eredményét közölné a felhasználóval. Ezzel lehetővé válna a felhasználó számára, hogy különböző alfanumerikus karaktereket vagy síkidomokat rajzoljon, majd ezek értelmezését követően a program a bevitt információt továbbítsa egy másik program vagy weboldal számára. Ennek több felhasználási felülete is elképzelhető:

- matematikai alakzatok rajzolása
- gyerekeknek új alakzatok tanítása
- billentyűzettel nem leírható karakterek bevitele
- billentyűzet sniffelés megkerülése jelszavak megadásakor
- új jelszóbeviteli metódusok kidolgozása: nem alfanumerikus karaktereket tárolnánk jelszavakhoz, hanem bizonyos alakzatokat vagy alakzat halmazokat, és később a belépni kívánó felhasználónak ezeket az alakzatokat (vagy legalábbis hozzájuk nagyon hasonlókat) kellene újra lerajzolnia a képernyőre. Ezáltal elkerülhetnénk a jelszó „véletlen” elmondásából következő biztonsági réseket, ugyanis a felhasználók nem tudnák megmondani, hogy mi a jelszavuk, ugyanis nagyon nehéz lenne azt körülírni, viszont könnyű őket lerajzolni, ami egy telefonos/online támadás esetén megvédheti a felhasználót. Ehhez természetesen a tárolt pontok megfelelő titkosítására van szükség.

A fenti tulajdonságok teljesítéséhez egy jó lehetőségnek tűnik a PCL (Point Cloud Library) használata, amely C++ felett nyújt kimagasló lehetőségeket létező pontokból 2 dimenziós alakzatok rekonstruálására. Ebben a félévben sajnos idő hiányában már nem tudtam foglalkozni ezekkel a lehetőségekkel, de a PCL, valamint egyéb hasonló függvénykönyvtárak vizsgálata és kipróbálása további izgalmas problémák megoldásához vezethet el.

11. További lehetőségek[13]

A Kinect 2010-es megjelenése után világrekordot döntött eladásokban: 2 hónap alatt 8 millió darabot adtak el belőle. Ez a siker meghozta más hardvergyártók kedvét is a hasonló típusú szenzorok gyártásához, így azóta jobbnál jobb, hasonló képességekkel rendelkező hardverek születnek. A Kinectet eredetileg konzoljátékokhoz tervezték úgy, hogy a felhasználó a szenzor előtt áll, és a végtagjaival irányítja a játékot. Jelenleg a PC-s felhasználók szinte kivétel nélkül csak a kezüket használják Kinecttel vezérelt programok irányítására, ami messze áll a Kinect eredeti tervezési céljaitól, ezért kicsit pontatlanabb, mint a piacon található eszközök nagy része. Cserébe a nagyszámú felhasználói közösség rengeteg felhasználói programot és függvénykönyvtárat írt hozzá, így viszonylag kevés keresgélés után használható programokat lehet hozzá találni.

A fent leírtak alapján felül kellene vizsgálni azt, hogy továbbra is a Kinect-e a legmegfelelőbb hardver az ilyen típusú programok készítéséhez. Ehhez az OpenNI keretrendszerrel szerencsénk van, ugyanis ez támogat bármely olyan hardvert, amely biztosítja a NUI-ban leírt követelményeket, így akár a programok minimális változtatása nélkül is van lehetőség új szenzor bevezetésére a Kinect helyett.

Az Önálló laboratórium 1. tárgy keretében elvégzett kutatásaim eredményeképpen szóba jöhet a PrimeSense Sensor, vagy az Asus Xtion PRO nevezetű szenzora, amelyek már kifejezetten PC-hez lettek tervezve.

Többszöri módosítások után véglegesnek tűnik a Leap motion nevű szenzor 2013. júliusi kiadási dátuma, amely forradalmi újításokat ígér a NUI technológiákat illetően. Ugyan ez már csak a kéz felismerését és követését teszi lehetővé a Kinect egész testet azonosítani képes tulajdonságával szemben, viszont a kezet és annak részeit sokkal pontosabban, és sokkal megbízhatóbban követi. Jelenleg (2013. május 20.) az eszköz béta tesztelése zajlik, valamint néhány kiválasztott alkalmazás fejlesztő gőzerővel dolgozik új alkalmazásokon, hogy a júliusi megjelenéskor már szép számú alkalmazás álljon készen a szintén elindítani kívánt alkalmazásboltban (ez a tervek szerint csak a nevében lesz bolt, ugyanis az alkalmazások nyílt forrásúak és ingyenesek lesznek).

Az eszközről az alábbi linken tekinthető meg egy bemutatkozó videó: https://www.youtube.com/watch?v=_d6KuiuteIA

13. ábra: Lep motion

12. Irodalomjegyzék

[1]https://github.com/palgabor/Onlab--Kinect-/blob/master/docs/onlab1/beszamolo.pdf

[2]http://www.openni.org/openni-sdk/openni-sdk-release-notes/#.UZoJ4JyfKkp

[3]http://www.openni.org/openni-sdk/#.UZoI7JyfKko

[4]http://www.openni.org/openni-migration-guide/#.UZoOxJyfKko

[5]https://github.com/piedar/OpenNI2-FreenectDriver

[6]http://www.primesense.com/casestudies/3gear-systems-2/

[7]http://makematics.com/code/FingerTracker/

[8]http://www.openni.org/solutions/finger-precise-hand-tracking/#.UZoUiJyfKko

[9]https://github.com/06casswp/Kinect-finger-tracking-library

[10]https://code.google.com/p/kineticspace/

[11]http://csmartonline.com/blog/2012/03/29/compile-openni-for-kinect-motor-control-on-linux/

[12]http://pointclouds.org/

[13]https://www.leapmotion.com/

13. Ábrajegyzék