

TDK-dolgozat

Tézsla Tibor, Keszthelyi Milán

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertechnológia Intézet



GESCAP – GESZTUSFELISMERŐ RENDSZER

Szerzők: **Keszthelyi Milán**
mérnök informatikus szak, IV. évf.
Tézsla Tibor
mérnök informatikus szak, III. évf.

Konzulensek: **Dr. Vámosy Zoltán**
egyetemi docens
Dr. Sergyán Szabolcs
Egyetemi docens

Budapest, 2012.

Tartalom

1	Bevezetés, célmegfogalmazás	2
2	Irodalomkutatás	4
2.1	Feladatok	4
2.1.1	Kézetektálás és jellemzők meghatározása.....	4
2.1.2	Kéz követése.....	8
2.1.3	Kéz letárolása	9
2.1.4	Osztályozás.....	11
2.2	Hasonló rendszerek	13
2.2.1	Recognizing Hand Gesture Using Motion Trajectories	13
2.2.2	Recognizing Hand Gestures with Microsoft's Kinect.....	13
2.2.3	Combining RGB and ToF Cameras for Real-time 3D Hand Gesture Interaction.....	14
2.3	Szükséges algoritmusok	15
2.3.1	Kapcsolódó komponens analízis	15
2.3.2	Medián szűrő	15
3	Saját rendszer bemutatása	16
3.1	Hardver.....	16
3.2	Célkonkretizálás	18
3.3	Rendszervázlat	18
3.4	Részletes rendszerelemzés	20
3.4.1	Általánosságban a rendszerünkről.....	20
3.4.2	Kézszegegmentálás	21
3.4.3	Gesztusfelismerés.....	23
4	Tesztek és eredmények.....	26
4.1	Kézszegegmentálás	26
4.2	Gesztusfelismerés.....	29
4.3	Kurzorvezérlés	30
5	Fejlesztési lehetőségek	30
6	Összegzés	31
7	Irodalomjegyzék.....	33

1 Bevezetés, célmegfogalmazás

Az első személyi számítógépek feltalálása óta napjainkig a beviteli perifériák területén nem történt olyan változás, mely alapjaiban változtatta volna meg az információ bevitelének módját. A felhasználó az egér-billentyűzet kombináció, vagy ezek alapvető funkciójával azonos, működésében eltérő eszközzel eljuttatta a számítógép feldolgozó egységébe az általa végrehajtani kívánt utasítást. A billentyűzet esetében ez lehet valamilyen parancs, vezérlési információ vagy akár egy szövegbevitelre szánt karakter leütése. Az egér esetében lehetséges a kurzor a képernyő egy másik pontjára irányítása, esetleg valamilyen esemény kiváltása a gombok lenyomásával, illetve a görgő görgetésével. A touchpad és a trackpad megvalósításában vannak különbségek, de alapvető funkciójuk azonos.

A projekt célja, melyet sikeresen meg is valósítottunk egy olyan szoftver létrehozása volt, mely egy képalkotó eszköz segítségével képes bármilyen fizikai kontaktus nélkül kiértékelni a felhasználó mozdulatait, összehasonlítani a korábban letárolt mintákkal, és kiválasztani a megfelelő interakciót. Ezek a mozdulatok természetesen lehetnek sokfélék. Mindennapjaink során is akaratosan-akaratlanul használjuk ezeket a gesztusokat, amikor másokkal kommunikálunk, vagy amikor érzelmeinket szeretnénk kifejezni. A legkifejezőbb mozgásokat a kezünkkel hajtjuk végre, ezen testrészünk mozdulatai a világ nagy részén ugyanazt jelentik. Ha ujjunkkal mutatunk valamire, ha legyintünk, ha eltolunk magunktól, vagy éppen magunkhoz húzunk valamit, ezek mind olyan gesztusok, melyeket könnyen fel lehet ismerni és értelmezni. Ezért választottuk mi is a kézzel leírt mozdulatok felismerését, így a felhasználónak nem kell megtanulnia használni egy teljesen új beviteli perifériát, a mindennapokban használt mozdulataival képes lehet kommunikálni a számítógéppel, ez látható az 1. ábrán.



1. ábra - Számítógép irányítása gesztusokkal

Ezt a fajta számítógép vezérlést az élet több területén fel lehet használni. Természetesen alkalmas az otthoni PC-k irányítására, legyen akár az laptop, esetleg asztali számítógép. A felhasználónak érdemes betanítania a saját mozdulataival az általunk létrehozott rendszert, a különböző gesztusokhoz parancsokat hozzárendelnie (ez bármilyen billentyű-, illetve egérkombináció lehet), ezután máris tudja használni a szoftvert egy Kinect segítségével. Ezen kívül alkalmazható előadások prezentálásához is, lehetséges diák között előre-, hátralépni, vagy bármilyen más parancsot be lehet tanítani.

A következő lépés az egérmutató irányítása volt. Ennek a megvalósítása nélkülözhetetlen ahhoz, hogy teljes mértékben irányítani tudjuk számítógépünket.

A rendszer létrehozásához szükségünk volt képalkotó eszközre. Erre a feladatra a Microsoft által fejlesztett Kinectet használtuk, hiszen az Xbox konzolhoz tartozó Kinect segítségével több olyan játék is létezik, mely hasonló képességekkel rendelkezik, a felhasználó különböző mozdulatokkal képes irányítani a játékot, mindezt controller vagy bármilyen fizikai érintkezés nélkül. Nagy előnye a mélységi kamera, mely infravörös projektor és érzékelő segítségével képes egy mélységi képet létrehozni. Projektünk teljes mértékben erre épül, aminek nagy előnye abban nyilvánul meg, hogy a rendszer szinte teljesen függetlenül működik a fényviszonyoktól.

A gesztusfelismerés sikeres megvalósítása mellett foglalkoztunk a kézszegmentálással is, mely a későbbi fejlesztésekhez nélkülözhetetlen lesz. Létrehoztunk egy olyan szegmentálást, mely változó fényviszonyok mellett is képes majdnem tökéletes minőségben a kéz pixeleit a mélységi képen meghatározni.

2 Irodalomkutatás

2.1 Feladatok

2.1.1 Kézetektálás és jellemzők meghatározása

Detektálás

Színes kép alapján történő detektálás

A kéz detektálását mind a színes, mind a mélységi kép alapján elvégezhetjük. A színes kép alapján történő kézfelismerési eljárásoknak többféle módszere ismert. Az egyik, egyben magától adódó eljárás, a kéz színjellemzőit használja a szegmentáláshoz [Ste10]. Ennek is több módja van, az egyik megközelítés az, amikor egy RGB képen a kéz színeihez hasonló intenzitásblokkokat keresünk [Fang07], ez a 2. ábrán látható. Ennek alapja, hogyha kezünkön nem viselünk semmit, akkor színei vagy előre meghatározhatóak, vagy közvetlenül a szoftver indítása után kezünk megmutatásával meghatározható hozzá egy intenzitástartomány, mely alapján később felismerhető, majd ezt használva később megállapíthatjuk az egyes pixelekről, hogy kézpixelet-e. Ezekre a megközelítésekre fokozottan igaz, hogy a megvilágítás sok problémát vet fel, hiszen más fényviszonyok között (az esetleges normalizálás ellenére) teljesen más színeket fogunk kapni. Ráadásul felhasználás közben is változhatnak a fényviszonyok, tehát ezeket is folyamatosan figyelniük kell, hogyha olyan rendszert fejlesztettünk, amelyik valós körülmények között is működik. Ez plusz erőforrásokat emésztene fel, ráadásul a megfelelő eredmény még így sem garantált. A legnagyobb probléma akkor jelentkezik, amikor a háttér színe közel azonos a kéz színével, ilyenkor nagyon nehezen tud ez a megközelítés megfelelő eredményt produkálni, erre láthatunk példát a 3. ábrán.

Több olyan megoldás is létezik, mely nem az RGB színteret veszi alapul, hanem például a HSV vagy az YCbCr színteret. Ezeknek a megvalósításoknak az a nagy előnye, hogy a színt a három dimenzió közül csak az egyik tárolja, a másik kettő megvalósítástól függően például a fényességet, vagy esetleg az RGB komponensek előre meghatározott különbségét tárolja. [Con08] Ezekkel a bőrszín-pixelet detektálása jóval hatásosabbá és egyszerűbbé válik, de problémák még ilyenkor is jelentkeznek, ha a megvilágítás erősebb/gyengébb az előre meghatározottnál. Ha ezt még sikerül is kiküszöbölnünk, ha a kéz nem egyenletesen van megvilágítva, a detektálás még elvégezhető, de az esetleges szegmentálás már nem megfelelően.



2. ábra - Szín alapján történő kézetektálás [Fang07]

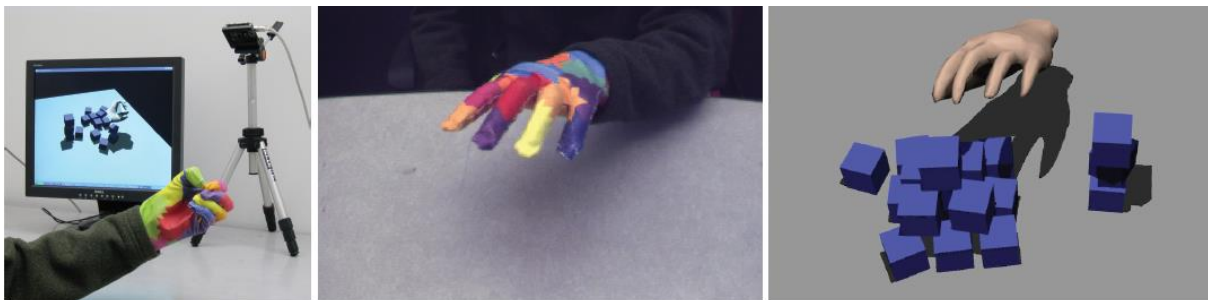


3. ábra - Problémák, ha a szín alapján történő kézdetektálással [Ber11]

Léteznek olyan megoldások is, melyek a felhasználó számára előírnak néhány kötelező kelléket, ilyen lehet például egy színes kesztyű, esetleg az ujjakon viselt színes megjelölő gyűrű [Wang09]. Ezek használatával a rendszernek jóval egyszerűbb dolga van, a kéz helyzetének meghatározásához, mint az a 4. ábrán is látható, pont ezért a mozdulatok felismeréséhez jelentősen gyorsabb eljárások használhatóak. Mindezek ellenére, mi nem szeretnénk, hogyha rendszerünk használatakor a felhasználónak olyan eszközöket, kiegészítőket kellene hordania, amit a mindennapi használat során nem venne fel.

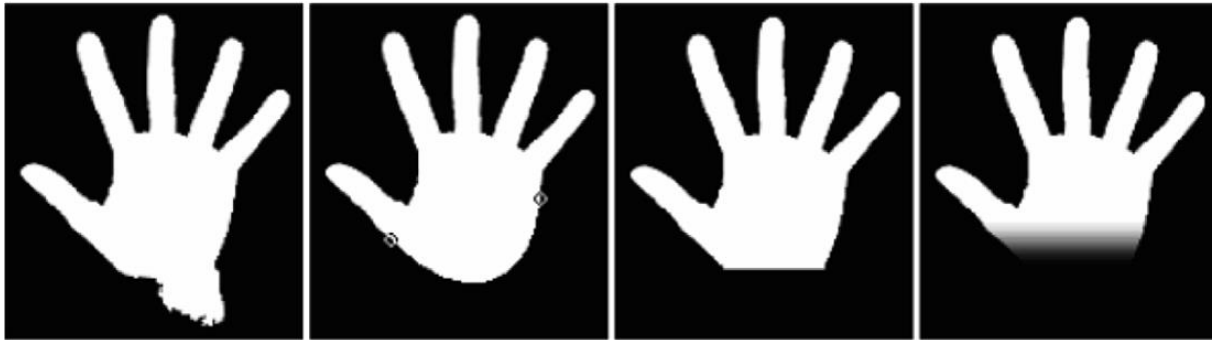
Olyan megközelítésekkel is találkoztunk, melyek a kéz jellegzetes alakja alapján detektálják azt, különböző maszkokkal végigiterálva a képen, az alakra jellemző erős intenzitáskülönbségeket keresnek [Fang07] [Yör06]. Ez a megközelítés bizonyos helyzetekben igen jól használható, amikor a kéz megfelelő pozíciót vesz fel (általában a kamerának a tenyeret látnia kell, az ujjaknak széttárva kell lenniük), és a háttér eltérő színű. Tehát olyan helyzetben, ahol a környezet sötét, de a kezét megvilágítja valami ez az eljárás nagyon jól használható, de ha a háttér és a kéz színei hasonlóak, ez sem ad megfelelő eredményt.

Ennek a megoldásnak egyfajta továbbgondolásai azok, melyek csak a gesztus felismerésére koncentrálnak, és megteremtik az ehhez szükséges körülményeket. Ez általában azt jelenti, hogy a kéz szinte az egyedüli objektum a képen. Természetesen ez is megoldható, a felhasználó a kezét mutathatja a kamerának, és még az is kivitelezhető, hogy a kéz mögé valami sötét háttér kerüljön, de ez a megoldás mindenképpen körülményes. Ilyen munka például a Manchester Egyetem projektje [Ahm95], vagy Erdem Yörük és társai 2006-ban publikált kutatása [Yör06]. Az ehhez hasonló rendszerekkel nagyon jó minőségű kézdetektálást lehet elérni, mint ez az 5. ábrán is látható. A körülményeket otthoni környezetben is képesek lehetünk produkálni, de semmiképpen nem nevezhető kényelmesnek ez a megoldás, tehát a mi szempontunkból nem érdemes használni. Ezek a megoldások általában a kéz alakja alapján határozzák meg a statikus gesztust, mivel a háttér itt eltérő



színű, így természetesen a színekkel nem kell foglalkozniuk.

4. ábra - Színes kesztyűvel történő kézdetektálás [Wang09]



5. ábra - Megfelelő körülmények mellett kiváló minőségű kézmodellt lehet előállítani [Yör06]

Összegezve ezeket a módszereket, az kijelenthető, hogy bár képesek megfelelő körülmények között eredményt produkálni, az otthoni, változó körülmények közötti felhasználásra (amik főleg a megvilágításból, illetve a háttér kézhez hasonló intenzitásából adódhatnak) nem megfelelőek.

Mélységi kép alapján történő detektálás

A másik lehetőség az, hogy a Kinect mélységi képét használjuk fel a célra. Tegyük egy kis kitekintést szoftverünk felhasználási területére: rendszerünk működésének szempontjából két, jelentősen eltérő felhasználási lehetőséget tudunk megkülönböztetni. Az egyik, amikor prezentációt tartanak a szoftver segítségével, vagy a felhasználó olyan helyzetben van, ahol a teljes teste látható, ilyenkor a kéz helyzete könnyedén meghatározható, az SDK Skeleton Tracking funkciója lehetőséget nyújt arra, hogy pontosan megkapjuk a kézfej közepének pozícióját. Ezentúl, a legtöbb képfeldolgozó osztálykönyvtárral, vagy annak kibővítésével lehetőség van emberi csontváz felismerni, ami alapján ugyanúgy megkaphatjuk a kéz pozícióját, mint a beépített Skeleton Trackinggel. Ez a megvalósítás gyakorlati vizsgálatokat igényel, hiszen a Kinect kameráinak felbontása annyira alacsony, hogyha a felhasználó három méternél messzebb áll a kamerától, a kéz egy megközelítőleg $64 * 64$ pixeles, vagy annál kisebb felbontással látható [Tang11]. A 6. ábrán látható egy példa a Kinect mélységi kamerájának képére.

A másik lehetőség az, amikor a felhasználó a számítógép előtt ül, ekkor a legközelebbi összefüggő ponthalmazt tekinthetjük a kéznek.

Bár a megvilágítás ennél a megoldásnál is probléma lehet, az infrakamera csak túl erős fénynél ad vissza hibás, vagy egyáltalán nem értékelhető képet. Beltérben ezzel a problémával nem fogunk találkozni. Egy másik hibaforrás, hogy a Kinect helyenként pontatlan, kiugró értékeket ad vissza (ez akár megfelelő, ellenőrzött körülmények között is lehetséges).



Ábra 6 - Kinect mélységi képe

Ez jelentkezhet olyan formában, hogy a kéz közepén egy, a környezetétől erősen eltérő távolságadat jelenik meg, vagy a kézfejen kívül, de ahhoz közel jelenik meg egy olyan távolsági adat, ami hasonló értékkel bír, mint maga a kéz (valójában mégse tartozik a kézhez). A problémának lehetséges megoldása az, hogy ezekkel a kiugró értékekkel nem foglalkozunk, ha megfelelő leírókat hozunk létre a kézfej tárolásához, akkor vagy egyáltalán nem, vagy csak olyan kis mértékben fogják az eredményt befolyásolni, amivel az még mindig tűréshatáron belül marad.

Ha az eredmény szempontjából szükséges lenne a legjobb minőségű adat továbbítása, mediánszűréssel¹ némileg orvosolható a probléma [Ste10], így a hibás, kiugró értékek eltűnnek. Lehetséges még a kapcsolódó komponens analízist felhasználni a kéz felismerésére, ezzel megtalálni azt a legnagyobb összefüggő szegmenst², ami maga a kézfej.

Bár a mélységi kép alkalmazásakor első körben mennyiségben kevesebb adat fog rendelkezésünkre állni a kézről (elvesztjük az intenzitásokat, illetve a mélységi kamera helyenként pontatlan képet ad vissza), képesek lehetünk a megfelelő eljárásokkal a két képet “szinkronba” hozni, tehát az egyik videó folyamán minden egyes pontnak megtalálni a megfelelőjét a másik videó folyamán. Így pontosan meg tudjuk mondani, hogy az adott (a, b) pixelnek a mélységi kamerán melyik (c, d) pixel felel meg a színes képen.

Továbbá az a lehetőség is adott, miszerint mind a színes, mind a mélységi képet felhasználjuk a detektálás és a gesztusfelismerés során [Tang11] [Ber11], így rendszerünk elég robosztus lesz ahhoz, hogy bármilyen körülmények között megfelelő eredményt szolgáltatson. Ez természetesen nagyobb számítási teljesítményt igényel, de a két eljárás akár egymásnak is szolgáltatathat adatot, ezáltal csökkentve futási idejüket. Az is az előnyök közé

^{1,2} a szükséges algoritmusok fejezet alatt tárgyaljuk

sorolható, hogy így a gesztus-felismerési eljárásnak is több adattal szolgálhatunk, így az gyorsabban és nagyobb biztonsággal végezheti munkáját.

2.1.2 Kéz követése

[Lah10] [Zab09] A kéz felismerése minden egyes képkockán eléggé időigényes feladat, és akár ki is váltható. A kéz követése például egy alternatíva. Különböző követő algoritmusok segítségével a mozgásvektorokat is meg tudjuk határozni, a most következő részben ilyeneket mutatunk be.

Az aktuális képkockából megjósolható, hogy a következő képkockán hol lesz nagy valószínűséggel a kéz. A **Kálmán szűrő** alkalmas erre a feladatra. Népszerűségének oka a valós idejű teljesítmény és a bizonytalanságok kezelése (8. Ábra). A Kálmán-szűrő olyan matematikai módszer, mely képes meghatározni becsléssel változó rendszerek időállapotait. Az adott időpontban végzett mérés és a korábbi mérések alapján képes az előremutató becslésre a rendszer jövőbeni állapotára.

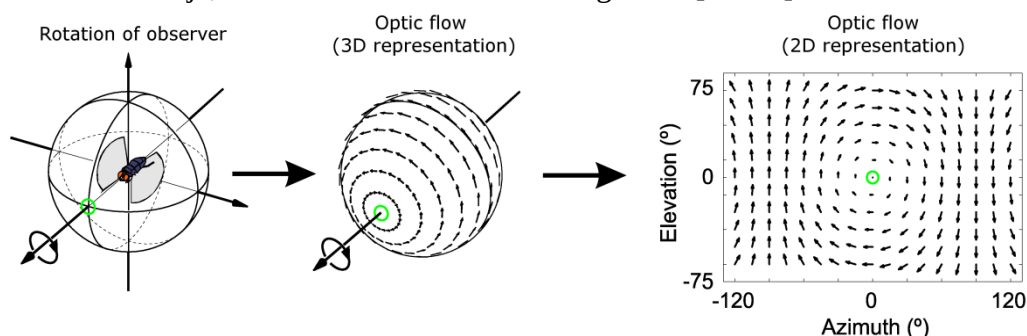
Kicsi deformáció esetén alkalmas követésre az **Optical flow**³ algoritmus is. Az algoritmus megbecsüli a képintenzitásokból a kétdimenziós mozgásmezőt, tehát 3 dimenziós mozgásokat tudunk ábrázolni kétdimenziós képeken vektorok segítségével (7. Ábra).

[Fang07] Ebben az irodalomban kombinálják az Optical flow-t és szín alapján való detektálással kapja meg a feldolgozandó pixeleket. Két kényszerfeltétel van, két jellemző nem lehet közelebb egymáshoz egy bizonyos küszöbérték távolságnál és a jellemző nem lehet messzebb a mediánjától, mint egy másik küszöbérték távolság. Célja, hogy egy beadott képsorozat két szomszédos képkockáján, adott képrészleteket megkeressük egy másik képen, és sebességvektort számoljunk ebből.

Optical flow fajtái:

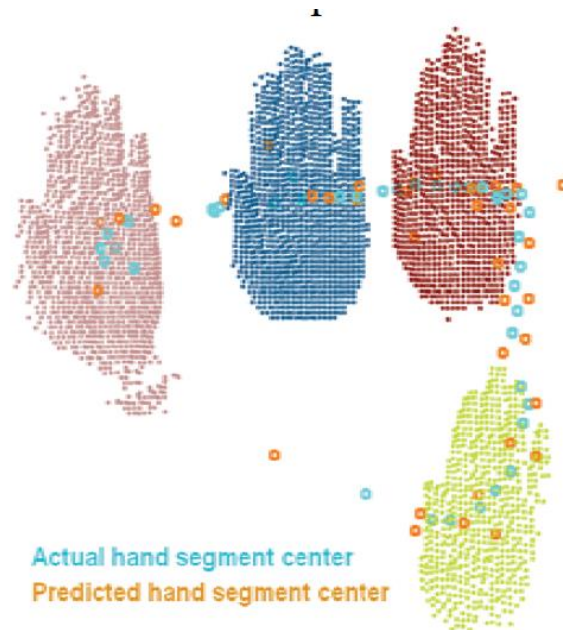
Differenciáló technikák számítják az irányvektort a tér-időbeli deriváltjából, vagy a kép szűrt verziójából (alul vagy sáváteresztő szűrők).

Precíz numerikus differenciálás sok esetben nem praktikus vagy a zajok miatt, vagy azért, mert kevés képkockánk van (például akkor okozhat hibát, ha gyors mozgást kell lekövetnünk, viszont keveset mintavételezünk), vagy azért mert kicsi a felbontás és így a vizsgált objektumot tévesen, nem látjuk összefüggőnek a képfeldolgozás során. Ezekben az esetekben alkalmatlanok a differenciáló technikák és ilyenkor fordulunk a **Régió-alapú illesztéshez**. Az irányvektort a képrégiók elmozdulásából, meggyörbüléséből és megnyúlásából számítja, ami invariáns eltolásra és forgatásra [Bar94].



7. ábra - Optical flow field³

³ http://en.wikipedia.org/wiki/Optical_flow



8. ábra - Kézkövetésre példa Kálmán szűrővel, világoskék: a kéz aktuális mediánja, narancs: előre megjósolt medián [Lah10]

Kanade–Lucas–Tomasi jellemző követés

Első változatban kifejlesztették a helyi keresés ötletét (csak lokális jellemzőket használ, az adott pixel kicsi környezetét nézi), ami olyan gradienseket alkalmaz, amik a kép második deriváltjának megközelítése által vannak súlyozva [Luc81]. Második ötlet ugyanannak az alapmetódusnak a továbbfejlesztése. Az adott jellemzőket akkor használják, ha a gradiensekből meghatározott kovariancia mátrixnak sajátértéke nagyobb, mint a megadott küszöbérték [Tom91].

Az Optical flow algoritmus hátránya, hogy gyors mozgások esetén nem működik [Vám8], mert az intenzitásértékek gyorsan változnak, valamint a homogén részeknél is nehéz a gradiens számolása, azonban megtámogathatjuk a Kálmán szűrővel, valamint rendelkezésünkre áll a mélységi kamera, ezért reményeink szerint kiküszöbölhetőek ezek a problémák.

2.1.3 Kéz letárolása

Nézzük meg, mik azok a tulajdonságok, amikre érzéketlenné kell tennünk rendszerünk (ezeket hívjuk invariáns jellemzőknek), hogy a megfelelő mintához párosítsa az új bemenetet.

Elforgatás:

Fontos, hogy csak a kisebb változásokra nézve legyen elfogadó a rendszer, tehát ha a kéz (az adatbázisban tárolt mintához képest) 5-10 fokban fordul el, jelezze a hasonlóságot, de nagyobb, akár 30 fok feletti elforgatásra már ne adjon egyezést. A pontos értékeket a szoftver fejlesztése során be tudjuk állítani.

Méretváltozás:

A kéz méretére is hasonlóan kell reagálnia a rendszernek. Ez egy többváltozós probléma, hiszen a szoftvert bár egy időben csak egy felhasználó használhatja, nem kizárt, hogy a számítógép elé a következő nap más valaki fog leülni, és ugyanúgy használni szeretné

rendszerünk szolgáltatásait. A probléma kiküszöbölésére több megoldás is használható. Az egyik, hogy olyan leírókat alkalmazunk, amik nem érzékenyek az egyébként is kismértékben eltérő kezekre, esetleg az eredmények vizsgálatánál ezt a hibalehetőséget is figyelembe vesszük. De nem szabad megfeledkeznünk arról sem, hogy a kéz nem fix távolságra lesz a kamerától, így ezt is számításba kell vennünk. Mindkét problémára megoldás lehet, ha olyan leírókat használunk, melyeket normalizálunk. Egy alternatíva, mivel ismerjük az objektumok szenzortól való távolságát, ezt használjuk fel a normalizáláshoz. A másik lehetőség az lehet, hogy a szoftverrel történő érintkezés kezdetén minden felhasználónak előre meghatározott nézetekből meg kell mutatnia a kezét, a rendszer pedig letárolja a fontosabb jellemzőket, és működése során végig ezekkel számol.

Alakváltozás:

A méretváltozással ezt is képesek lehetünk kezelni.

A másik fontos, már említett szempont, hogy minél kisebb kimeneti adathalmazzal kell rendelkezni. A rendszer oldaláról ez kritikus funkció, a legtöbb időt a különböző minták összehasonlítása, a keresés veheti el, tehát célszerű a lehető legtömörebb adatszerkezetet továbbadni. Bár a kézfej körülvágása és továbbküldése is megvalósítható, ennél jóval hatékonyabb eljárások is léteznek, melyekkel bár a feldolgozási idő megnő, a keresést és felismerést jelentősen gyorsíthatjuk.

Ha a kézhez tartozó pixeleket meghatároztuk, létre kell hoznunk olyan adatszerkezetet, ami minél kevesebb adatból áll, de lényegi információvesztés nélkül reprezentálja a tárolni kívánt kéz alakját. Mivel a meghatározáshoz használhatjuk a mélységi képet is, ezért jelentősen többletinformációval rendelkezünk, mintha a színes kép alapján próbálnánk meghatározni az alakot.

Az egyszerű RGB kép alapján történő kéz meghatározások (ilyen például [Yör06], [Ren11]), csak a kéz körvonalát tudják felhasználni a felismeréshez, nekünk viszont a teljes kéz minden egyes pixeléről rendelkezésünkre áll annak a kamerától (pontosabban az infraszenzortól) mért távolsága.

[Ren11] **Time-Series Curve** osztályozásra és csoportosításra is jól használható. A TSC relatív távolságvektort (vertex⁴) tárol a körvonallról a középponthoz képest. A középpontként az objektum súlypontját veszik, kezdőpont meghatározásához pedig a felhasználónak viselnie kell egy fekete karkötőt, és a képen beazonosított karkötő legbaloldalibb pontját veszik, és így forgásra invariánssá teszik. Reprezentáláskor a súlyponttól mért eukleidészi távolságot ábrázolja (függőleges tengelyen az eukleidészi relatív távolság van, vízszintesen a körvonal és a kezdőpont közötti szög relatívan a középponthoz képest).

A mi rendszerünkben is alkalmazhatunk szignatúrákat a kéz reprezentálására, az előző projekt alapján invariáns eltolásra, forgatásra, ha találunk valami fix pontot, ami minden kéz képén azonos, és az infraszenzornak hála a skálázásra is invariánssá tehetjük.

⁴ Vertex – olyan adatstruktúra, ami egy vektor koordinátái mellett egyéb információt is tartalmazhat. (például: pozíció, textúrakoordináták, normálvektor)

2.1.4 Osztályozás

Rejtett Markov-modellek (Hidden Markov Models)

Kutatásaink során felkeltette a figyelmünket, hogy milyen jó eredményeket értek el Rejtett Markov-modellekkel, időben végbemenő véletlen folyamatok kezelésében [Ves01], [Law89], [Elm08].

A Rejtett Markov-modellekkel (továbbiakban: HMM) jól leírhatóak időben végbemenő véletlen folyamatok. Ilyen a mozgás is, mert egy olyan rendszer generálja, melynek környezeti paraméterei időben változnak. "HMM alaptulajdonsága: (markovi) annak a valószínűsége, hogy a rendszer mit csinál egy adott állapotában (az ún. kibocsátási és átmeneti valószínűség), csak az aktuális állapottól függ, az előzményeknek nincs szerepe." [Ban1]

Egy szemléletes példát találhatunk az egyik irodalomban is [Sta12], mely abból indul ki, hogy a fák évgyűrűinek nagyságából következtetni lehet arra, hogy az adott évben milyen volt az átlagos hőmérséklet. Tehát nem tudjuk megvizsgálni, hogy milyen volt a hőmérséklet, de az évgyűrűből következtethetünk rá, hogy valószínűleg milyen volt. Mivel az állapotok (a hőmérsékletek) rejtettek, ezt a fajta rendszer HMM rendszer.

Projektünkben a Hidden Markov Model megvalósításának alapjai César de Souza projektjére⁵ épül, melyet [Law89] és [Elm08] tanulmány alapján készített.

Hagyományosan a következőképpen szokás jelölni a HMM-et:

$$\lambda = (N, M, A, B, \pi)$$

Paraméterek:

- Az állapotok összessége $S = \{s_1, s_2, \dots, s_N\}$, ahol N az állapotok száma a modellben.
- A kezdőállapot valószínűségi vektor $\pi_i, i=1, 2, \dots, N$.
- Az $N \times N$ -es állapotátmenetek valószínűségének mátrixa: $A = \{a_{ij}\}$ ahol a_{ij} az átmenet valószínűsége S_i -ből S_j -be; $1 \leq i, j \leq N$ és az értékek összegének az A mátrix minden egyes sorában 1-nek kell lennie, mert ez a valószínűsége az átmenetnek, egy adott állapottól a másikba.
- A lehetséges megfigyelések $O = \{o_1, o_2, \dots, o_T\}$ ahol T a megfigyelt szekvencia (gesztus) hossza.
- A diszkrét szimbólumok összessége $V = \{v_1, v_2, \dots, v_M\}$ ahol M a diszkrét szimbólumok száma (ahány fajta megfigyelés lehetséges, esetünkben a kódszavak).
- Az $N \times M$ megfigyelési mátrix: $B = \{b_{im}\}$ ahol b_{im} adja meg v_m szimbólum valószínűségét S_i állapotban, és az értékek összegének az B mátrix minden sorában 1-nek kell lennie, az előző indokból kifolyólag.

Három alapvető problémát fogalmazhatunk meg a Rejtett Markov Modellekkel kapcsolatban, amiket meg kell oldanunk, amennyiben működő alkalmazásokat akarunk létrehozni [Law89]:

1. **Becslés (Evaluation):** Adott a modell „ λ ”, számítsuk ki valószínűséget egy egyéni állapot-sorozatra „ O ”, tehát mi a valószínűsége annak, hogy az adott modell generálta az adott állapotsorozatot. Ezen probléma megoldására a Forward algoritmust használtuk, ami egy formája a dinamikus programozásnak.
2. **Dekódolás (Decoding):** Adott a modell „ λ ”, számítsuk ki valószínűséget egy egyéni állapot-sorozatra „ O ”, találjuk meg azt az állapot sorozatot, ami legvalószínűbben generálta ezt a megfigyelés sorozatot. Viterbi algoritmus segítségével oldottuk meg. (Lényege, hogy „ t ” időpontbeli legvalószínűbb

⁵ <http://www.codeproject.com/Articles/69647/Hidden-Markov-Models-in-C>

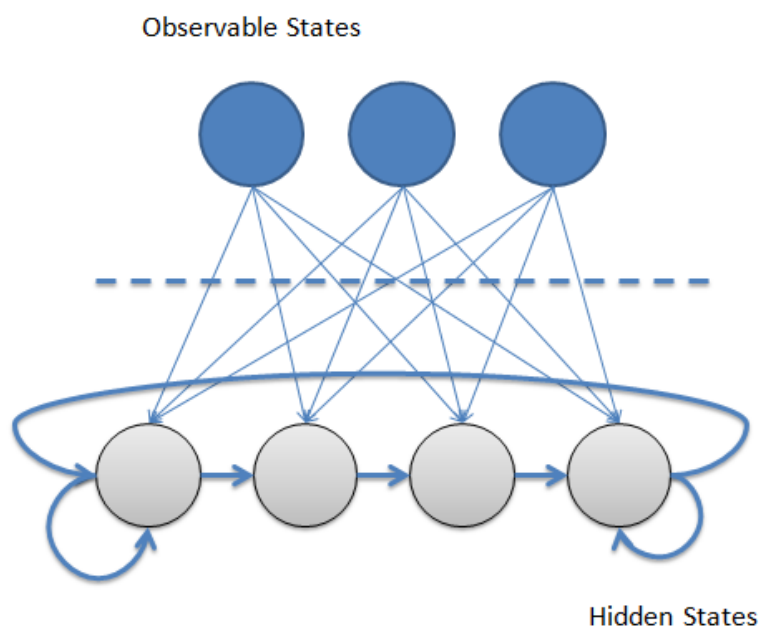
állapotot $t-1$ -edik legvalószínűbb állapotból származtatjuk, azaz minden állapotban megnézzük, hogy melyik állapot felől következett a maximális valószínűség.)

3. **Tanítás (Learning):** Keressük meg az optimális paramétereket a modellhez, annak érdekében, hogy adott megfigyelés sorozatra a legjobb eredményt produkálja. Nem ismert pontos tanuló algoritmus erre a problémára, de lokálisan maximális valószínűségek számolhatóak Baum-Welch algoritmust használva. (Ki tudja számolni a maximum valószínűségek becslését valamint meg tudja becsülni az átmeneti és kibocsátási valószínűségeket, csak a tanító adatok segítségével.)

A saját rendszerünkben a kódszavakat tekinthetjük a rendszer állapotainak. A rendszer indításakor betanítjuk a modelleket (felismerendő gesztusokat), az előzőleg megfigyelt szekvenciákkal (rendszerünk által előállított kódszavak).

Felismeréskor végigiteráljuk az aktuálisan megfigyelt szekvenciánkat az összes eddigi modellen, megbecsüljük annak a valószínűségét, hogy az adott modell generálta az adott megfigyelés sorozatot és a legnagyobb valószínűségűt választjuk.

A 9. ábrán láthatunk egy példát a Rejtett Markov Modellek egy ábrázolására. Fent láthatóak a megfigyelhető állapotok, alul pedig a rejtett állapotok, a nyilak az ezek közötti kapcsolatokat jelölik.



9. ábra Példa Rejtett Markov Modell vizuális ábrázolására⁶

⁶ <http://www.codeproject.com/Articles/69647/Hidden-Markov-Models-in-C>

2.2 Hasonló rendszerek

2.2.1 Recognizing Hand Gesture Using Motion Trajectories

[Yang99] A módszer kifejlesztésének célja az amerikai jelnyelvet felismerő rendszer készítése volt. Időben késleltetett, többrétegű előrecsatolt neurális hálót (TDNN) használnak. Ennek az az előnye, hogy az időbeli változásokat le lehet követni vele. Egy hisztogram alapján küszöböl, ellipszissel reprezentálja a kezet, majd kiszámolja az ellipszisek matematikai jellemzőit. Tanításra sztenderd hiba visszaterjesztéses algoritmust használnak. A vizuális bemenet osztályozására Rejtett Markov Modelleket alkalmaznak (HMMs). Videófolyammal tanították, és minden egyes vizuális esemény egy jellemző vektor. Tanítás után az új megfigyeléseket azzal a modellel osztályozták, ami a legnagyobb valószínűséget jelöli meg. 93% pontosságot értek el ismeretlen videót feldolgozva.

2.2.2 Recognizing Hand Gestures with Microsoft's Kinect

[Tang11] A rendszert Matthew Tang, a Stanford Egyetem Villamosmérnök tanszékének egy hallgatója fejlesztette a 2011/12-es tavaszi félévben. A megvalósítás célja az, hogy a teljes testre vonatkozó adatok segítségével előre definiált statikus gesztusokat (tehát nem mozgásokat, csak statikus kézpozíciókat) ismerjen fel.

A minta felismerése három lépésben lett meghatározva: először is azon pixelek azonosítása, amelyeket kézpixelnek tekinthetünk, ezután jellemzők kinyerése, majd ezek alapján a gesztus felismerése. A kéz pozíciójára becslést a test elemzéséből, a body tracking algoritmus segítségével kapott (ennek egy megvalósítása a Kinect SDK-ban is implementálva van). A pixelek pontos azonosítása első körben az RGB adatok alapján történt meg normalizálás után. A normalizáláshoz egy olyan képletet használtak, melyben minden egyes pixel intenzitásait leosztják a képen megjelenő legerősebb intenzitással (legyen az akármelyik színösszetevő). Így megkapták azokat a pontokat a képen, amelyek színük alapján lehetséges, hogy a kéz pixelei. Az a feltevés egyértelmű, miszerint ha egy lehetséges kéz pixelt olyan pixelek vesznek körbe, amelyek úgyszint a kézhez tartoznak, akkor feltehetően a kérdéses pixel is oda fog tartozni. Ezért a fejlesztő egy három képpont átmérőjű elemmel végighaladt a képen, így meghatározva egy közel végső eredményt. Ha az így létrejött adatokat összehasonlítjuk a Kinect által kapott mélységi képpel, pontos eredményt kaphatunk.

Mivel a Kinect által visszaadott mélységi kép elég pontatlan, ezért a szokásos elforgatásra invariáns leírók nem használhatóak a kézre (például a fő gradiens irány), de a body tracker (magyarul test követés) segítségével képesek lehetünk megállapítani a kézfej helyzetét is, így a leírókat forgatás invariánssá lehet tenni.

Az első leíró, amit használtak a radial histogram (magyarul sugárirányú hisztogram). Ehhez szükség van a kéz középpontjára, melyet a következő képlettel számolt ki:

$$x_c = \frac{1}{N} \sum_{x,y} xp(x,y); y_c = \frac{1}{N} \sum_{x,y} yp(x,y)$$

Ahol x_c a középpont x irányú összetevője, y_c az y irányú összetevő, N az elemszám, $p(x,y)$ pedig annak a valószínűsége, hogy (x,y) helyen levő pixel kézpixel.

$$\alpha(x, y) = \tan^{-1} \left[\frac{(x - x_c)}{(y - y_c)} \right]$$

Ahol $\alpha(x, y)$ az (x, y) helyen levő pixel középponttal bezárt szöge. Ezt minden pixelre kiszámolták, majd az így kapott eredményt normalizálták. Sajnos ez a megoldás a zajokra túlságosan érzékeny, így a Kinecttel való használata korlátozott.

A legeredményesebb leírónak a SURF leíró bizonyult [Bay08]. A gesztus felismerő alrendszer számunkra nem lényeges, mert módszerük összesen két állapot közül képes kiválasztani a megfelelőt, a nyílt és a zárt kézfej közül.

2.2.3 Combining RGB and ToF Cameras for Real-time 3D Hand Gesture Interaction

[Ber11] A szerzők megoldásából leginkább az az algoritmus érdekes, amelyet a kéz felismerésére használtak. Rendszerükhöz ők is egy mélységi, illetve egy RGB kamerát használtak. A mélységi kamera adatait hozzárendelték az RGB képéhez, így minden egyes pixelről meg tudták állapítani, pontosan milyen színű, és hol helyezkedik el a térben. A kéz felismerésénél az általunk is vázolt problémákkal szembesültek, a kéz színeihez hasonló színű objektumokat kellett kiszűrniük, különben a rendszer azokat is a kéz pixelei közé sorolta. A megoldás érdekében egy teljesen újszerű megközelítést alkalmaztak. Egy előre beállított bőrszín modellt és egy adaptív modellt is létrehoztak. Az utóbbi valós időben mintákat vett az arc színeiből (amelyet egy arc detektorral találtak meg), így folyamatosan a megfelelő bőrszínhez tudták igazítani rendszerüket. Megvalósításuk azon az egyszerű logikán alapult, ha a kamera látja a felhasználó kezét, akkor az arcát is, és a két testrészen a bőr pixeleinek intenzitása közel azonos, így folyamatosan elő tudnak állítani egy referencia intenzitást. Bár ezzel egy problémát megoldottak, újakat is generáltak, és több régít sem sikerült kiküszöbölni. Ha a kamera az arcot a kéz takarásában látja, nem tudja megfelelően szétválasztani, melyik pixel melyik testrészhez tartozik. Illetve ha egy másik személy bőrfelülete a kézhez közel kerül a kamera képén (ha fedésbe kerül, a probléma még nagyobb), a rendszer azt is kézpixeleként határozza meg. Ennek a kiküszöbölésére alkalmaztak mélységi kamerát (a ToF kamera egy betűszó, a Time of Flight rövidítése; a működés elve az, hogy mivel tudjuk, mekkora a fény sebessége, és ismerjük, hogy a fény mikor hagyta el a kamerát, ki tudjuk számolni az egyes visszaverődések távolságát, így előállíthatunk egy mélységi képet a kamera előtt elterülő térről), így már egyértelműen tudták a kezét szegmentálni.

Amikor a kéz mögött nem jelent meg semmilyen hozzá hasonló intenzitású objektum, a rendszer hatékonysága a szín alapú detektálásnál 92%-os volt, míg a mélységi kép alapjánál 99,2%-os. Ha az arc előtt helyezkedett el a kéz, akkor 71,8% és 100% volt a mélységi mérés javára. A legutolsó mérésakor egy másik személy keze előtt helyezkedett el a kéz, ilyenkor a színes kép 19,8%-os pontossággal találta meg a kezét, a mélységi kép 98,8%-os pontossággal, mindezt 362 felvett képkockán mérve.

Az eredményekből egyértelműen látszódik, hogy bár ők ToF kamerát használtak, ami jóval drágább és pontosabb, mint a Kinect, a mélységi kép alapján jelentősen jobb eredményt is értek el, így számunkra is megfontolandó a mélységi kamera elsődleges alkalmazása.

2.3 Szükséges algoritmusok

2.3.1 Kapcsolódó komponens analízis

[Vám04] Gépi látás területén használt technika, mellyel képesek lehetünk detektálni egy bináris képen az egymással összekapcsolt régiókat. Megkülönböztetünk 4-es és 8-as szomszédságot.

A kétdimenziós képtömbön kétszer kell végighaladnunk, először amikor a különböző komponenseket megcímkézzük, közben eltároljuk a címkék közötti ekvivalenciát (hogya két címke egymással a meghatározott szomszédság alapján szomszédos, mégis különböző a címkéjük), majd amikor ezeket az eltárolt egyezéseket azonos címkével látjuk el. Az első körben történő címkézési szabály a következő:

ha egy elem nem magas (tehát "1" az értéke, vagy 255), akkor nem vizsgáljuk, egyébként a következőt tesszük (minden esetben, ha a feltétel nem teljesül, a következő feltételre ugrunk):

1. Ha a felette levőnek nem 0 az értéke, akkor a vizsgált elem megkapja a felette levő elem címkéjét.

- 1.1 Ha a vizsgált elemtől balra levő elemnek is van címkéje, akkor eltároljuk a hasonlóságot.

2. Ha a balra levő pixelnek van címkéje, akkor azt megkapja.

3. Ha az előzőek közül egyik sem teljesül, akkor új címkét kap.

2.3.2 Medián szűrő

[Ser10] Olyan ablakos művelet, melyet a kiugró zajok eltüntetésére használunk. Az ablak elemeit sorrendbe állítjuk, és a középső pixelének értéke a sorrendbe állított elemek közül a középső lesz.

3 Saját rendszer bemutatása

3.1 Hardver

Mint azt már a bevezetőben említettük, projektünkhöz a Microsoft által fejlesztett Kinectet használjuk. Úgy gondoljuk, megérné pár szót ejteni erről a hardverről és a tulajdonságairól.

A Kinect for Xbox (10. ábra) 2010 novemberében jelent meg. A hardver alapvetően több szenzorból álló érzékelő rendszer, melyet a Microsoft Xbox konzoljához készülő játékokhoz terveztek. A cél hasonló volt, mint a mi projektünk esetén: a konzolhoz készült különböző játékok gesztusokkal, hanggal történő irányítása.

Balról jobbra haladva ezek a szenzorok találhatók meg a Kinectben: infravörös projektor, RGB kamera, infrakamera, illetve a kamerák alatt négy mikrofon. Kezdjük a talán legegyszerűbb érzékelővel, az RGB szenzorral. Ez egy teljesen átlagos RGB kamera, mely 640*480 pixel felbontású képet szolgáltat 30 képkocka/másodperc sebességgel. Minden egyes pixelt 32 biten, tehát 4 bájtban ábrázol, ebből 1-1-1 bájt a vörös, a zöld és a kék csatorna, a negyedik bájt pedig kihasználatlanul marad.

A mélységi kamera működési elve a következő: az infravörös projektor kibocsát egy előre meghatározott mintát, mely infravörös pontokból áll, és ezt képes az infrakamera érzékelni. Ezek a szenzorok a szabad szem által nem látható tartományban sugároznak, illetve vesznek, így az RGB kamerát, illetve a szemet nem zavarják. Természetesen mint a legtöbb technológiának, ennek a megoldásnak is van hátránya. Ha a képbe üveg, esetleg tükör kerül, a szenzor nem képes megfelelően mérni a távolságot, így ezekről nem is képes adatot visszaadni, ilyenkor ezen pixelek helyén a „nem ismert” adatot, tehát 0-t ad vissza. Hasonló a helyzet akkor is, amikor túl fényes a környezet. Épületen belül ezt csak nagy teljesítményű izzókkal lehet elérni, átlagos felhasználási körülmények között nem jelentkezik a hiba, viszont ha a Kinectet napfényben, szabad ég alatt szeretnénk használni, az érzékelt adatok teljesen használhatatlanok bármilyen számításához.

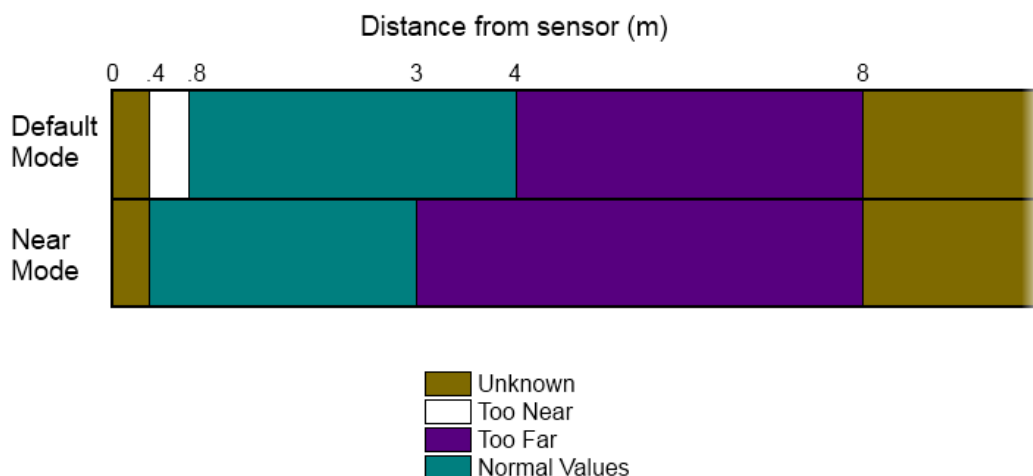


10. ábra - Kinect for Xbox

A hardver által visszaadott adatokról többféle forrás eltérő formátumot említ. Saját programunkkal tudjuk igazolni, hogy a mélységi szenzor 16biten, tehát 2bájton adja vissza a mélységi képet, melyből 15 az értékes bit, egy bit pedig őrzőbit. Minden pixelhez tartozik egy érték, mely megadja, hogy az adott pixel melyik felismert skeletonhoz tartozik. Ezt 3 biten kapjuk meg, tehát a maradék 12 bit számít „értékes bitnek” a távolsági adatok szempontjából. Méréseink alapján is a legnagyobb visszaadott távolság 4095, amit pont 12 biten vagyunk képesek ábrázolni. A hardver ezekből a képkockákból is 30-at ad vissza másodpercenként.

Microsoft 2012. február elsején a világ több országában elérhetővé tette a Kinect for Windows-t, mely az elődhöz képest (Kinect for Xbox) több újítást, fejlesztést is hozott (és ezzel egy időpontban mutatták be a végleges SDK-t (Szoftver-fejlesztő csomag), mely az 1.5-ös verziószámot kapta meg). Ezen változások nagy része a jobb használhatóságot tűzi ki célul, támogatják az általános kereskedelmi felhasználást, és minden platformot, illetve az RGB kamera képe már nem csak 640*480-as felbontásban, hanem 1280*960-as felbontásban is elérhető a fejlesztők számára, viszont így le kell mondanunk a 30 képkocka/másodpercről, a magasabb felbontás mellett csak 12 képkocka másodpercenkénti beolvasására van lehetőségünk. Bevezetésre került az úgynevezett „near mode”, melynek segítségével a szenzor képes mélységi képet alkotni a 40 és 80 centiméter közötti objektumokról is (11. ábra). Sajnos hátránya is van ennek az újításnak. Bár a Kinect a közelebbi objektumokat is látja, amennyiben ezt a módot használjuk, a távoli eredmények torzulnak, megnövekszik a zaj, tehát a mérések pontatlanabbak lesznek. A másik nagyobb újítás is ezzel a fejlesztéssel kapcsolatos, az új SDK képes felismerni az emberi skeletont ülő helyzetben is, ezt nevezik „Seated” módnak.

A két új módnak az együttes alkalmazásával tehát képesek lehetünk a Kinect előtt ülő embert is felismerni. Ezt a módszert mi is kipróbáltuk, de sajnos nagyon sok erőforrást használ, és messze nem lehet vele olyan eredményeket elérni, mint a normál skeleton felismeréssel. Jóval nehezebben találja meg a felhasználót, és gyakran „elveszti”, tehát több másodpercig nem ismeri fel, így számunkra, folyamatos használatra alkalmatlan.



11. ábra - Kinect near mode⁷

⁷ <http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/20/near-mode-what-it-is-and-isn-t.aspx>

A hardver és a bemenő jelek formátuma tehát adott, így hát rátérhetünk arra, hogyan szeretnénk elérni szoftverünkkel a kívánt célt, a számítógéppel történő kommunikációt.

3.2 Célkonkretizálás

Célunk a projekt során folyamatosan módosult, alakult. Alapvetően a gesztusok tulajdonságával kapcsolatban több dolgot is elképzelhetők tartottunk. A cél mindenképpen az volt, hogy a felhasználó mozdulatait összehasonlítsuk az előre letároltakkal, és így kiválasszuk, melyik volt az a gesztus, amelyet végrehajtott.

Első elképzeléseink között is többféle megoldás szerepelt, felvetődött, hogy a kézfej különböző mozdulatait (a csukló fix helyen tartásával), esetleg a kézfej középpontja által leírt pályát tekintjük gesztusnak.

Végül úgy döntöttünk, egyik fő célunk egy megfelelő kézszegegmentálás előállítása, és a és fizikai kontaktus nélküli számítógép vezérlés elérése lesz, a lehető legrobosztusabb gesztusfelismerés elkészítésével, illetve a kéz egéreként használatának lehetővé tételével.

3.3 Rendszervázlat

A megvalósítás a Microsoft által fejlesztett C# programozási nyelven készült el, .NET 4.0-ás keretrendszer segítségével. A felhasználói felületet WPF segítségével valósítjuk meg, mivel jelenleg ez a leginkább támogatott platform, az 1.5ös Kinect SDK is ezzel képes a legjobban együttműködni. A program futása közben a UI (User Interface) nem látható folyamatosan a felhasználó számára, a program a háttérben való futtatásra lett tervezve, de természetesen bármikor előhozható a grafikus interfész.

A rendszer egyszerű vázlatát a 12. ábra mutatja, a felhasználó végrehajtja az általa kigondolt gesztust, ezt a Kinect érzékeli és a videofolyamokat átadja a programnak, ami létrehozza a kódszavakat a felhasználó által végrehajtott mozdulatból, majd a HMM segítségével felismeri azt, és végrehajtja a megfelelő interakciót.

Rendszerünk másik fontos képessége a kéz megfelelő szegegmentálása. Ez egy teljesen eltérő feladat, habár mindkettőt a Kinect SDK által szolgáltatott Skeleton Tracking (tehát emberi szkeleton felismerés) alapján valósítjuk meg.



12. ábra - A gesztustól az interakcióig

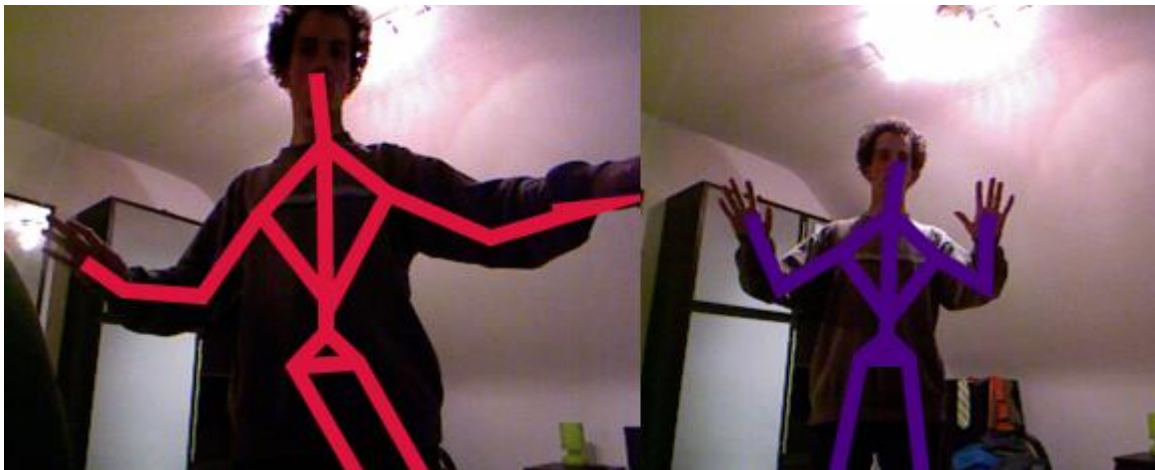
Szerettük volna ezt a két részt külön kezelni, mivel a gesztusfelismerést képesek voltunk megoldani a kéz szegmentálása nélkül is, viszont úgy gondoljuk, lényeges tapasztalatokat, információkat szereztünk a szegmentálással kapcsolatos próbálkozások, tesztek során.

3.4 Részletes rendszerelemzés

3.4.1 Általánosságban a rendszerünkről

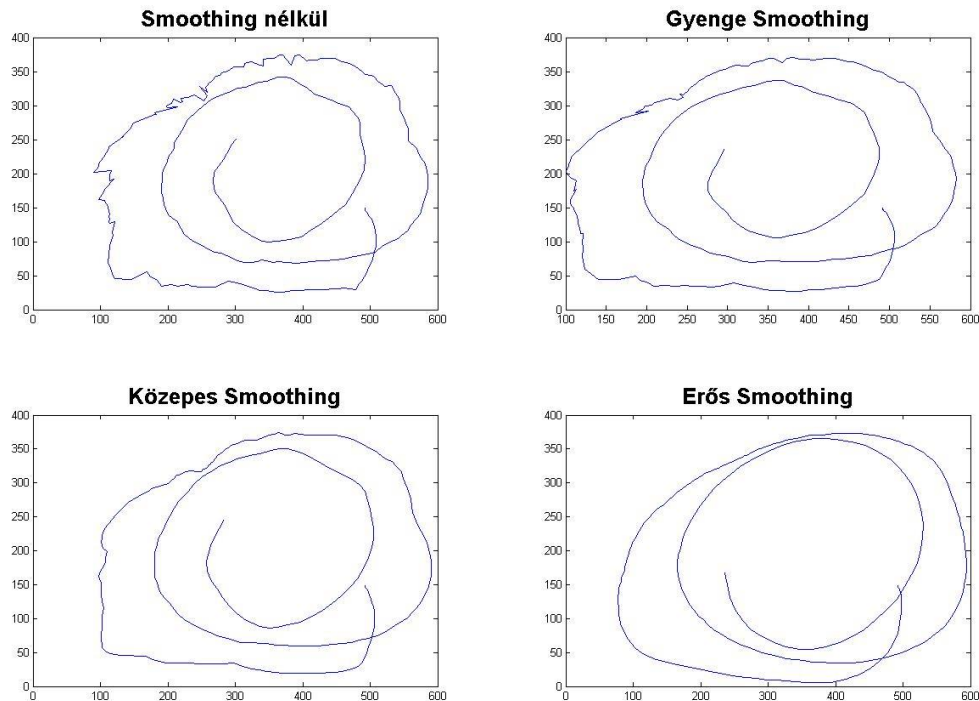
Mint azt már korábban is említettük a Kinect SDK képes az emberi skeletont felismerésére, ezt a megoldást mi is használtuk a programunkban. Miután a mélységi, illetve az RGB folyamat megkaptuk, megkapjuk a skeletont is, amennyiben azt felismerte a mélységi videofolyamon a Kinect. Ez akkor lehetséges, ha a fejet, a kéz kezét és a test nagy részét látja. Egy felismert példát láthatunk a 13. ábrán.

Fontos megemlítenünk egy lehetőséget, melyet a Kinect SDK-ja biztosít számunkra, és bizonyos esetekben rendkívül hasznos tud lenni: a skeleton létrehozásához használhatunk smoothingot⁸. A Kinect infrakamerájának nincs meg az elegendő felbontása ahhoz, hogy következetesen pontos legyen, sok a téves Skeleton pont adat, nagy ugrások is előfordulnak két egymást követő framen. Ezen problémákat hivatott kiküszöbölni a Kinect SDK-jában megtalálható Smoothing. Szűri és simítja a szenzorból beérkező adatokat, eltünteti vagy csillapítja a zajokat, miközben feljavítja a valós adatokat, ezáltal lekicsinyíti az elmozdulásokat az egymást követő frame-eken. Megbecsüli az átlagos elmozdulást a következő frame-re, ezáltal is lecsökkenti a nagy ugrásokat. Kettős exponenciális simítást (Holt's Double Exponential Smoothing) használ. Ezen módszer lényege, hogy vesszük visszamenőlegesen a mért értékeket, majd azokat különböző súlyértékekkel vesszük figyelembe, minél régebbi az adat annál kisebb súllyal esik latba, majd az így előállított simítási értékeket még egyszer simítjuk. Ezáltal hatékonyan csillapítja a véletlenek hatását. Általában gazdasági folyamatok statisztikai elemzésére használatos ez a módszer. A 14. ábrán láthatunk egy példát különböző smoothing beállítások alkalmazására. A képeken a jobb kéz középpontjának koordinátáinak időbeli változásait láthatjuk. Az ábrázolás a Matlab 2009a program segítségével történt.



13. ábra - Kinect által sikeresen felismer skeleton

⁸ <http://msdn.microsoft.com/en-us/library/jj131024.aspx>



14. ábra - Kéz követése különböző Smoothing beállításokkal

A Kinect által elkészített képkockák kinyerésére két lehetőségünk is van: Polling modell vagy Event modell⁹. Az Event modell értesít, ha egy képkocka előállt azáltal, hogy kivált egy eseményt. Ebben az eseményben írhatjuk bele azokat a parancsokat, melyeket szeretnénk a képkocka elkészültekor lefuttatni. Ezzel a módszerrel egyszerűen lehetőségünk van a háromféle event (RGB kép, mélységi kép, skeleton elkészülte) külön kezelésére, így megadva hogy valamelyik elkészültekor mit szeretnénk végrehajtani.

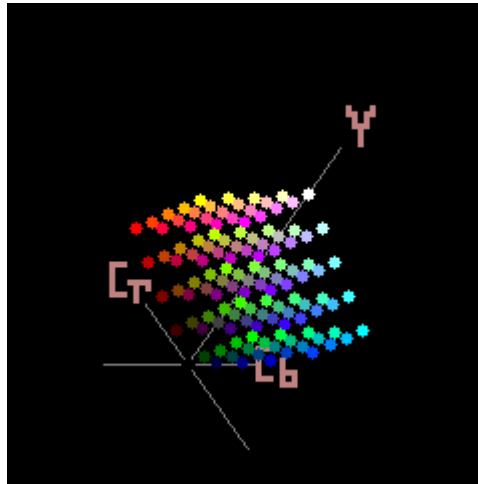
Polling modell esetén mi kérhetjük a következő képkockát, megadva, hogy meddig várunk az elkészültére. Ha a képkocka elkészült, vagy az idő lejárt a metódus visszatér a megfelelő értékkel. Amennyiben nem állt elő képkocka akkor „null” értéket ad vissza GetNextFrame függvény.

3.4.2 Kézssegmentálás

Ahogy az az irodalomkutatásunkban is látszódik, sokféle kézssegmentálási lehetőséggel találkoztunk. Azóta többet kipróbáltunk, megismertünk. Az egyik ilyen az **YCbCr**¹⁰ színtérre alapul, egy reprezentálását a 1. ábrán láthatjuk. Ezt többféleképpen is hívják, többek között YCC, Y’CbCr vagy YPbPrnak, leggyakrabban fotó és videó kódolásra használják, a YPbPr pedig ennek az analóg verziója. A leggyakrabban használt és számunkra is a legkifejezőbb színtér az RGB, ez ugye három alapszín, a vörös, a zöld és a kék színek keverésével adja meg a többi színt. De ezen kívül léteznek más színterek is, a másik legelterjedtebb az YCbCr. Ez is ugyanúgy három komponensből áll, az Y a fényesség, a másik két összetevő a színhelyességért felel, ezeket az RGB kék és vörös összetevőiből

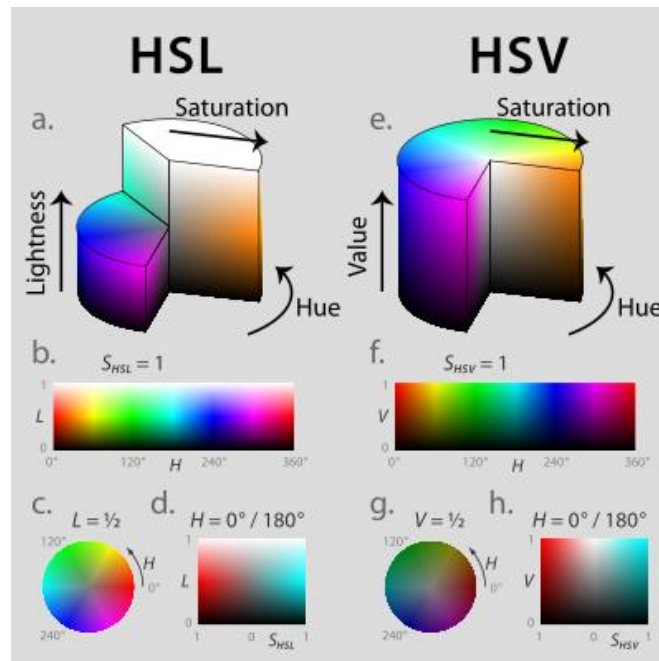
⁹ <http://msdn.microsoft.com/en-us/library/hh973076.aspx>

¹⁰ <http://en.wikipedia.org/wiki/YCbCr>

15. ábra - YCbCr színtér⁸

képezzük a fényesség segítségével. Az alapötlet az volt, hogy az emberi érzékenyebb a fényességre, mint a többi összetevőre, így ha ezt több biten kódoljuk, a másik két összetevőt lehet tömöríteni. Viszont az emberi bőrszínre is jobban lehet ezáltal konstans értékeket meghatározni, mint az RGB színtérben, hiszen a Cb, illetve a Cr színösszetevők a fényességtől függetlenebbek. Természetese pontos értéket nem lehet itt előre definiálni, de intervallum deklarálására lehetőségünk van.

A másik lehetőség a **HSV**¹¹ színtér használata. Ez a rövidítés a színárnyalat, telítettség, érték szóhármast takarja. Talán a legjobban úgy lehet elképzelni a három összetevőt, hogy a Hue a szín, a Saturation a szín „mennyisége”, a Value pedig a fényerő, világosság, ezt ábrázolja a 16. ábra. Azért nagyon közkedvelt ez a megvalósítás, mert csak egy csatorna kódolja a színt, ez pedig a H, így elég erre intervallumot felállítanunk, nem szükséges mind a három színre, mint az RGB színtérben.



16. ábra - HSV színtér

¹¹ http://en.wikipedia.org/wiki/HSL_and_HSV

A harmadik megoldás az, amit mi gondoltunk ki, mintegy **saját megoldásként** szeretnénk az előző kettővel összevetni. Mint ahogy a fejezet elején is említettük, a Kinect SDK képes az emberi skeleton meghatározására, ezzel együtt a kézfej közepének is a meghatározására. Mivel adott ehhez a mélységi képünk, és meg tudjuk határozni hogy egy emberi kéz hozzávetőlegesen mekkora lehet x, y és z irányokba, a középpontból kiindulva meg tudjuk határozni azokat a pixeleket, melyek a kézhez tartoznak. Ha ezt a módszert használjuk, a háttér semmiképpen nem zavarhat meg minket, hiszen egyértelmű a nagy távolságbeli különbség (a Kinecttől mérve). Természetesen problémák itt is felmerülhetnek, hiszen láttunk példát arra [Tang11], hogy ez a megoldás sem volt megfelelő, hiszen a kéz egy 64*64 pixeles négyzetben helyezkedett el, és a Kinect pontatlansága mellett ezt nem lehet megfelelően felismerni.

3.4.3 Gesztusfelismerés

Gesztus betanítása

Gesztusonként 20-20 tanítóminta lett beadva a korábban említett Rejtett Markov Modell tanítás részének, a megvizsgált tanulmányokból kiindulva ennyi adat már elég megfelelő tanítómintának. Ezen adatok segítségével gesztusonként külön modellt alkotott, hogy a későbbiekben felismerhesse.

Gesztus határainak felismerése

Eddig nem esett szó róla, de mindenképpen fontos megjegyeznünk, hogy az olvasott irodalmakban kétféle, egymástól nagyon különböző gesztust is gesztusnak tekintenek. Az egyik általunk statikus gesztusnak elnevezett verzió az álló, mozdulatlan kéz felismerését tűzi ki célul, ilyen például [Fang07]. A másik fele az kutatásoknak a mozdulatokat hívja gesztusnak, és ezeket kívánja felismerni, ilyen például [Ber11]. Nekünk a célunk kezdetektől a mozdulatok felismerése volt, hiszen ezek jóval kifejezőbbek, mint egy egyszerű kézjel. Ha mégis szeretnénk később ezeket is felismerni, ami például jól jöhet a jelbeszédben használt gesztusok felismeréséhez.

Tehát dinamikus gesztusok felismerését tűztük ki célul. A Kinect kamerája segítségével kapunk egy élő videó folyamatot. Felvetődött azon kérdés, hogy mik legyenek a gesztus határok, honnan lehet tudni, hogy mikor van vége egy gesztusnak?

Első ötletünk az volt, hogy a két kéz összeérintésekor induljon, majd eltávolodás után, ismételt összeérintéskor álljon is meg az adott gesztus felismerése. Ez a módszer csak olyan esetekben kényelmes, ha a gesztusnak a kezdete és a vége is a másik kéznél van, más esetben körülményes koordinálni a mozgást. Nem akartunk ilyen kompromisszumot kötni.

Következő ötlet, hogy fix ideig nem mozog a kéz, akkor legyen a felismerés vége. Tapasztalatunk szerint 2-3 másodpercbe is beletelik mire a felismert és a valódi kézközéppont közötti távolság 10 pixelnél kisebb lesz és csak további 3-4 másodperc múlva áll be konstansra, feltéve persze, ha eddig hajlandóak vagyunk pontosan egy helyben tartani a kezünket. Ez köszönhető egyrészt a Kinect pontatlanságának, valamint a Smoothing algoritmus működési elvének.

Egy másik gondolatunk az volt, hogy egy FIFO elvű felismerés volt, ami minden új frame előállásakor megvizsgálná, hogy az azt megelőző fix időben történt-e olyan gesztus, amit fel tud ismerni. Több probléma is felvetődik: gesztusok be vannak határolva időtartamba, nem lehetnek hosszabbak, mint a meghatározott fix idő, számításigényes, minden képkocka előállásakor vizsgáldni.

A következő módszert alkalmaztuk végül: amennyiben a kéz egy adott referenciátávolságnál közelebb van a Kinecthez, akkor elkezdjük a gesztus rögzítését, és ha ezután távolabb kerül a referenciátávolságnál, akkor abbahagyjuk a rögzítést és megkezdjük a letárolást. Ez a felhasználónak semmilyen problémát nem okoz, több alannyal tesztelve mindenki érezte, hogy ha egy picit közelebb teszi a kezét a Kinecthez, a felvétel elkezdődik, ha visszahúzza, a felvétel abbamarad.

Gesztus letárolása

[Elm08] Tanulmányban található ötletet továbbgondolva alkottuk meg a gesztustárolási algoritmusunkat. A Skeletonból kinyert jobb kéz középpontjának koordinátáit felhasználva megvizsgáljuk az időben egymás után következő elmozdulásokat az alábbi képlet segítségével:

$$\alpha_t = \arctan \left[\frac{\left(\sum_{i=1}^4 y_{t+i} - \underset{i=1}{\overset{4}{\text{MIN}}} y_{t+i} - \underset{i=1}{\overset{4}{\text{MAX}}} y_{t+i} \right) - \left(\sum_{i=0}^3 y_{t+i} - \underset{i=0}{\overset{3}{\text{MIN}}} y_{t+i} - \underset{i=0}{\overset{3}{\text{MAX}}} y_{t+i} \right)}{\left(\sum_{i=1}^4 x_{t+i} - \underset{i=1}{\overset{4}{\text{MIN}}} x_{t+i} - \underset{i=1}{\overset{4}{\text{MAX}}} x_{t+i} \right) - \left(\sum_{i=0}^3 x_{t+i} - \underset{i=0}{\overset{3}{\text{MIN}}} x_{t+i} - \underset{i=0}{\overset{3}{\text{MAX}}} x_{t+i} \right)} \right]; t = 1, 2, \dots, T - 5$$

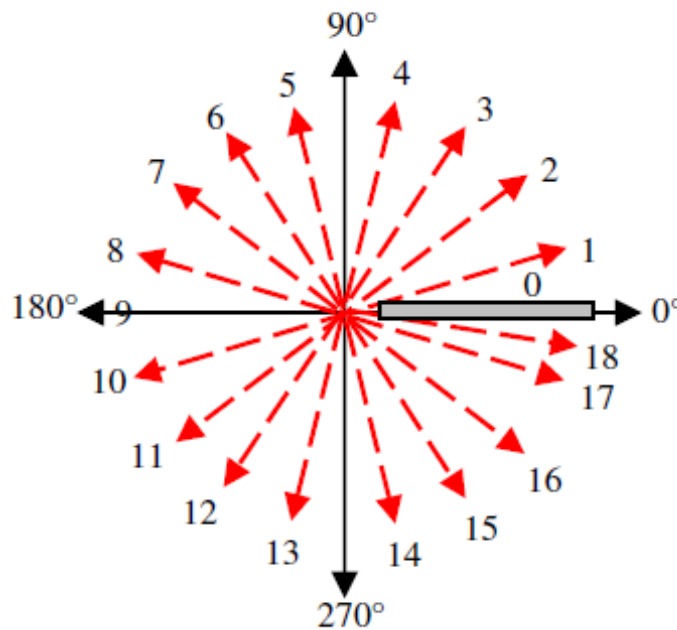
17. ábra - Egymást követő képkockákon felismert kézközéppontok irányszögének kiszámítása

Ahol T a gesztus hossza, x_t, y_t a jobb kéz koordinátái adott „t” időpillanatban. Annak érdekében, hogy a zajokat eltüntessük, a négy adatból a legkisebbet és a legnagyobbat elhagyjuk, majd a maradék két adat számtani átlagát vesszük.

Miután megkaptuk az elmozdulás orientációját, leosztjuk 20°-kal így kódszavakat generálunk 1-től 18-ig (18. ábra).

Gesztus felismerés

A korábban említett HMM felismerés segítségével a letárolt gesztus felismerését elvégeztetjük. Kiszámolja minden modellre, hogy mekkora valószínűséggel generálta a beadott állapotsorozatot, majd kiválasztja legnagyobb valószínűségűt.



18. ábra - Egymást követő képkockán felismert kézközéppont elmozdulási kódszavai [Elm08]

Grafikus interfész

Miután sikeresen felismertük a kívánt gesztust, megindítjuk a megfelelő interakciót, ezzel a Kinectet a programunk segítségével parancsbeviteli perifériaként használván, képesek voltunk a felismert gesztusokhoz társítani olyan interakciókat, mint kicsinyítés (Ctrl+-), nagyítás (Ctrl++), lapozás (PageUp, PageDown), aktív ablak váltása (Alt+Tab) stb. Bizonyos billentyűkombinációk univerzálisak, azaz a programok nagy hányadában ugyanazt az interakciót váltják ki, ezért nem kell program-specifikusan deklarálni őket (ilyenek a fent említett interakciók is). Programunk így akkor is működött, ha nem aktív ablakként működött, tehát a háttérben futott. Kipróbáltuk Firefox böngészőben, Microsoft Word szövegszerkesztőben, illetve Microsoft Access programban is.

4 Tesztek és eredmények

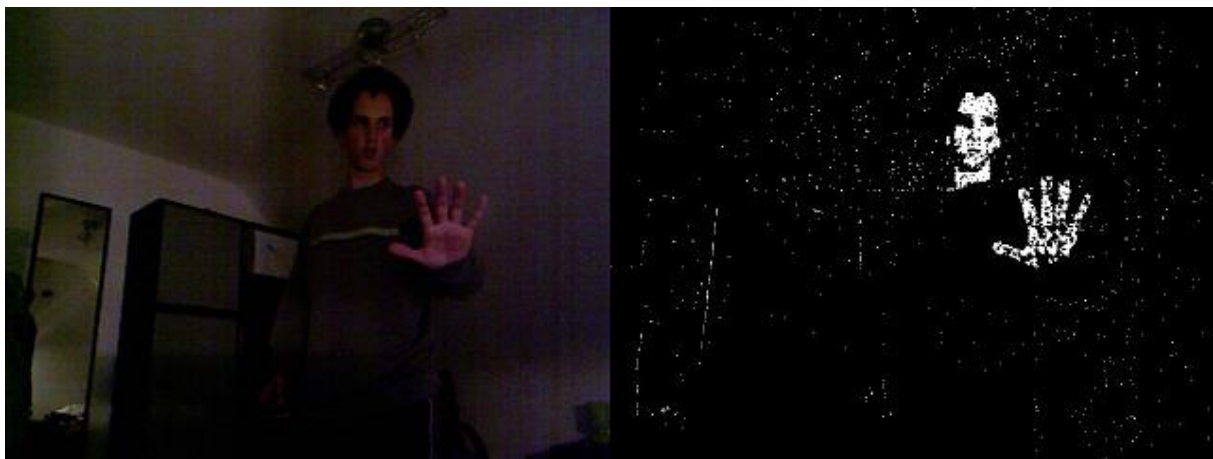
4.1 Kézssegmentálás

Mint azt már korábban is írtuk, három módszert hasonlítottunk össze, a kérdés az, hogy melyik adja a legjobb minőségű kéz modellt. A tesztkörülmenyeket is változtattuk, így lehetőséget adva a kétféle színtérnek, hogy különböző fényviszonyok között megmutathassák, mire képesek összehasonlítva a mélységi képpel. Az első változó a megvilágítás volt. Négy lámpa a plafonon folyamatosan égett, ehhez a fényerőhöz jött hozzá még másik kettő-négy lámpa, melyek bizonyos esetben az arcot is megvilágították. A felhasználó mögötti háttér végig állandó volt, de a ruha színét változtattuk egy közel bőr színű és egy kék színű felső között. Utóbbi elég egyértelműen elkülöníthető a bőrpixeletől.

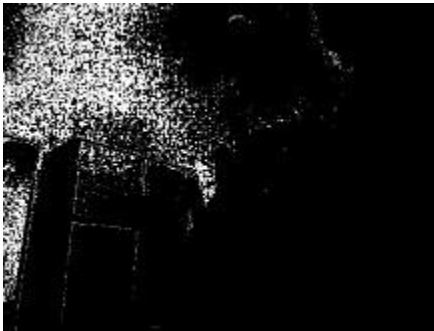
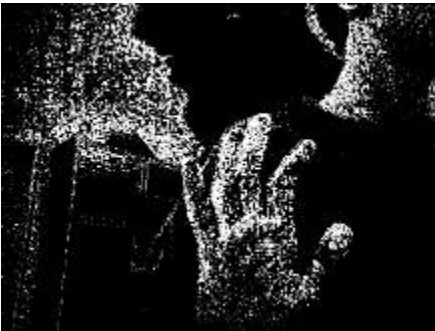
Természetesen mind a YCbCr, mind a HSV színtérnél nagyon fontosak az intervallumok, melyek között bőrpixelnek tekintjük a pixelt, így ezt többszöri próbálkozás után sikerült beállítani a (egy bizonyos megvilágításhoz tartozó) megfelelő értékekre. Alapvetően az interneten található különböző kész projektek értékeiből indultunk ki, és ezeket módosítottuk aszerint, hogy a lehető legjobb minőségű detektálást elérhessük.

A HSV színtérnél a megfelelő intervallum keresését képekkel dokumentáltuk. A fényképek alatt az szerepel, hogy az adott képen éppen mit állítottunk be, ez látható az 1. táblázatban. A fehér pixelek felelnek meg a bőrpixelként felismert pixelmaszknak. A minimális érték a HSV-hez (140, 100, 50) lett, a maximális (160, 255, 180). Az YCbCr-hez pedig (0, 142, 88) és (255, 180, 127) lett a két paraméter. A megfelelő megvilágításhoz tartozó megfelelő paramétereket láthatjuk a 19. ábrán.

Miután sikerült beállítani a megfelelő intervallumokat, és a skeleon alapján a kézfej közepe is megvan, a kézről készült tesztképeket vettük fel, ezek láthatóak a 2. táblázatban. A két teszt között a megvilágítás változott. Ekkor lettünk arra figyelmesek, hogy mindkét színtér bár elméletileg jelentősen független a fényességtől, az mégis erősen befolyásolja az eredményt. Az egy bizonyos fényerősségnél beállított paraméterek egyértelműen rossznak bizonyultak akkor, amikor a felhasználó a szobának egy másik részén állt, vagy eggyel több-kevesebb lámpa világította meg. Ha megvizsgáljuk az 1. táblázat utolsó elemét, illetve a 2. táblázat HSV oszlopában levő bármely elemet, akkor láthatjuk a problémát.



19. ábra - Megfelelő paraméterekkel és megvilágítással a YCbCr színtér képes jó eredményeket elérni

		
Saturation maximum	Saturation minimum	Value minimum
		
Value maximum	Megfelelő saturation-value	Megfelelő saturation-value
		
Hue minimum	Hue maximum	Megfelelő beállítások

1. táblázat - Megfelelő HSV beállítások keresése

A 2. táblázatban látható tesztképek alapján kijelenthető, hogy sem a YCbCr, sem a HSV nem képes változó körülmények között önmagában megfelelő bőrszindekeltálást nyújtani, míg a mélységi kép alapján történő detektálás legjobb eredményei kielégítőek. A zajtól, pontatlanságoktól eltekintve használhatóak további számításokhoz, ha medián szűrést végrehajtottunk rajtuk, az eredmények még pontosabbak lehetnek, mint azt a 20. ábrán is láthatjuk. Jelen esetben egy medián szűréssel az eredmény nagyban javítható, tapasztalataink alapján a többszörös szűrés felesleges, az eredmény nem lesz jelentősen jobb minőségű.

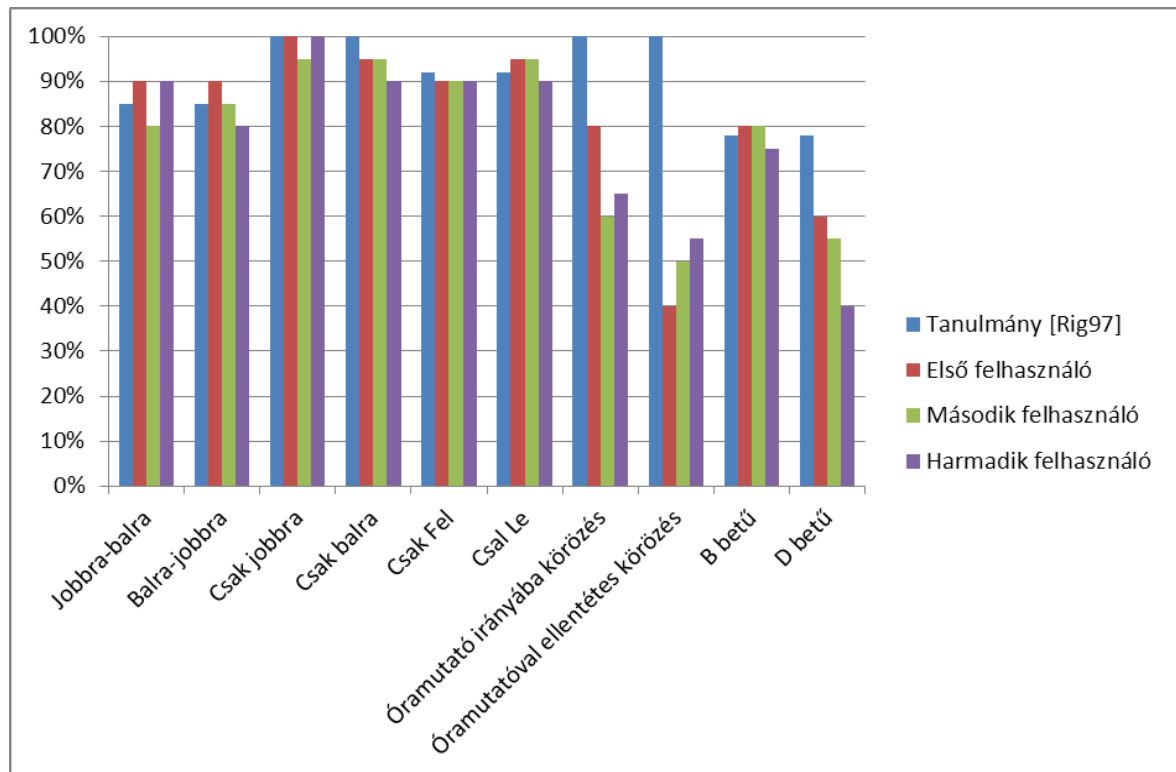


20. ábra - Mélységi kép alapján detektált kéz medián szűrés nélkül, egyszeres, illetve hatszoros medián szűréssel

		YCbCr	HSV	Mélységi
Bőrszínű háttér	Első teszt			
	Második teszt			
Kék háttér	Első teszt			
	Második teszt			

2. táblázat - Legjobb minőségű eredményképek

4.2 Gesztusfelismerés



1. Grafikon - Teszteredmények összevetése

Tesztelésünk során 20-20 tanítómintát rendeltünk minden egyes gesztushoz egyazon felhasználótól (Első felhasználó), majd összevetettük ugyanaz a felhasználó és másik kettő felhasználó 20-20 tesztmintájával, valamint egy tanulmányban található gesztuscsoporthoz (21. ábra). Az eredmények a 1. grafikonon láthatóak.



21. ábra - Rendszerünkben is betanított gesztusok ábrázolása[Rig97]

Mint ahogy minden egyes embernél a kézírás, úgy a gesztusok végrehajtása is egyéni lesz, ezért lehetséges, hogy ha ugyanazon ember tanítja be, és teszteli is a mozdulatot, akkor nagyobb százalékban ismeri fel, mintha egy másik ember akarja ugyanazt a gesztus produkálni. Amint az valószínűsíthető volt a rövid, és az általunk használt jelölésrendszerünkben jól szeparálható mozdulatok, mint a „Csak jobbra” gesztust nagy százalékban felismeri a rendszerünk és kevés hibát vét. Azonban ha hasonló mozdulatokat tanítunk és tesztelünk, akkor hajlamos tévesen felismerni azokat, ilyenek például a „B betű”, és a „D betű”. Tapasztalataink azt mutatják, hogyha ugyanott kezdődött a körözés gesztus akkor nagyobb valószínűséggel ismerte fel azt. A mérési eredményeken torzít, hogy 24 tanítómintával dolgoztak a konkurens tanulmányban.

Eredmények értékelése

Több módon is lehetne javítani rendszerünkön.

- A felhasználók jobban begyakorolhatnák a mozdulatok végrehajtását, hogy minél kisebb legyen abban a véletlen tényező
- A felhasználók maguk taníthatnák be a gesztusokat, ebben az esetben saját magukra pontosabb lenne.
- Több felhasználó taníthatná be, így valószínűsíthetően, általános esetben gyakrabban ismerné fel azokat, de könnyebben össze is tévesztené a rendszer, mert annyira eltérően tanítanak be a mozdulatokat.

4.3 Kurzorvezérlés

Megvalósítottunk egy kurzorvezérlést is, hogy a számítógép irányítása teljes legyen. A program figyelembe veszi az aktuális képernyő méretét, és a jobb kéznek a mozgását ennek megfelelően skálázva a felhasználó képes lehet a képernyőn a kurzort mozgatni. Természetesen betanítható a szoftver az egér jobb, illetve bal gombjának kattintására is, így teljes mértékben képesek lehetünk kiváltani az egeret. Itt is a gesztusfelismeréshez hasonlóan működő elmosást használjuk, a legutolsó két szélsőértéket elhagyjuk, a középső megmaradó értékeket pedig átlagoljuk, hogy a legkevésbé remegjen a kurzor az esetleges pontatlanságok miatt.

5 Fejlesztési lehetőségek

A siketek által használt Jelnyelv ugyanolyan nehezen tanulható, mint a beszélt nyelvek nagy része, ezért ebben is nagy potenciált látunk, hogy felismertessük vagy akár betanításra is használjuk programunkat. A kézmozgások felismerésére, és betaníttatására is használható jelenleg korlátozottan a programunk, amennyiben az adott gesztusban csak a kéz útja számít és az alakja nem. Amennyiben a kéz alakját és fel tudjuk ismertetni a programunkat, abban az esetben teljes körűen használható az előbb felvázolt célra. [Che03] Ebben a kutatásban van példa a kéz alakjának felismertetésére.

A kézszegegmentálásnál egyedül a mélységi kép hozott olyan eredményeket, amelyekkel tovább lehet dolgozni, de ez nem azt jelenti, hogy egyéb módszerekkel nem lehet elérni megfelelő eredményeket. Az arcdetektálás egy elég jól körbejárt téma, az egyik leginkább releváns kutatás a Viola-Jones féle arcdetektálás [Vio04], ha ezt megvalósítjuk, majd az arc színéből mintát veszünk, és azt a színt keressük az RGB csatornán, jobb

hatásfokot érhetünk el. Természetesen itt is van létjogosultsága a YCbCr vagy a HSV színtér használatának.

6 Összegzés

Jelen dokumentumban megpróbáltunk mindenre kitérni, ami a dolgozat szempontjából fontos lehet. Viszont azt már látjuk, hogy bármennyit is foglalkozunk a témával, mindig fellelhetőek új megoldások, fejlesztési irányok, új kutatások, eredmények, esetleg olyanok is, amelyekre csak később leltünk rá.

Hasonló témában elég sok publikáció olvasható a szakirodalomban, de a mi általunk tervezett megvalósításhoz egyik sem áll annyira közel, mint amennyire mi azt az elején gondoltuk. A legtöbb rendszer színes kép alapján dolgozik, és elég kevés mozdulatot ismer csak fel, vagy szinte csak statikus gesztusokkal foglalkozik, ezek közül valós körülmények között kevés működik megfelelően. A mélységi képre alapozó rendszerek csak napjainkban kezdtek el terjedni, továbbra is nagy potenciált látunk ebben a területben. Ezek sokkal jobb hatásfokkal ismerik fel a különböző kézjeleket, de gesztusokat (tehát kézmozdulatokat) ugyanúgy nagyon kevesen próbálnak értelmezni.

A Microsoft a Kinect for Windows-t még csak fejlesztői verziónak szánja, ez mind az árából, mind a kiforrottságából látható, de remélhetőleg nemsokára szélesebb körben elterjed ez a szenzor is. Projektünkhöz viszont a Kinect for Xbox is elegendőnek bizonyult, tudásában nem különbözik jelentősen a Kinect for Windows-tól.

Sikeresen létrehoztunk egy olyan rendszert, mely relatíve nagy gyakorisággal képes felismerni a felhasználó mozdulatait, megvizsgálva a tesztünkben összehasonlított tanulmányt, kimondhatjuk, hogy sikerült az egyszerűbb gesztusok terén elérni közel egyforma felismerési rátát. Amennyiben egyénre szabottan tanítottuk be a szükséges gesztusokat egy adott feladathoz, még túl is szárnyaltuk az adott tanulmányt. Mindemellett képesek vagyunk képesek vagyunk levegőben leírt kézmozdulatokkal irányítani a kurzort. Rendszerünknek ezen része számtalan módon felhasználható, akár otthoni számítógép vezérlésre, akár tanórákon prezentálásra, így az előadónak nem szükséges a kezében egy eszközt hordoznia, a diák közötti váltáshoz, nem szükséges odamenni a számítógépéhez, egyszerűen a Kinecttől megfelelő távolságra a látóterébe kell állnia, és itt végrehajtani a megfelelő mozdulatot. Ha olyan egyedi, bonyolult mozdulatot tanítunk be a rendszernek, amit nehéz leutánozni, akár „aláírásként”, hitelesítési célokat is szolgálhat a gesztusfelismerés.

Sikeresen szegmentáltuk a kezet a mélységi kamera segítségével, a YCbCr és a HSV színtereket implementáltuk, ezekbe RGB képeket átkonvertáltunk, így összehasonlítva a saját és az irodalmakban leírt módszereket. Megismerkedtünk a Kinect programozásával, számtalan lehetőséggel, melyet a programozóknak kínál. A gesztusok felismerésében, helyzettől függően képesek lettünk olyan szintű felismerésre, mely jobb az irodalmakban taglaltaknál, mindezt valós, változó körülményekkel (például fényesség), amiről nem sok más rendszer dicsekedhet.

Látjuk a lehetőséget arra, hogy a kézleíró is fejlesszük, ezáltal rendszerünket egy picit más irányba is elindíthassuk. Ezen a vonalon a további fejlesztésekkel tervezzük azt elérni, hogy a jelbeszédet képesek legyünk felismerni, viszont pontosan a mélységi kép miatt ez egy olyan terület (jellemző pont detektorok alkalmazása a mélységi képen), amellyel még nem sokan próbálkoztak.

Összegezve a tapasztaltakat úgy gondoljuk, hogy ez egy nagyon érdekes kutatási terület, mely nagy jövő előtt áll, így mindenképpen érdemes vele foglalkozni!

7 Irodalomjegyzék

[Ahm95] T. Ahmad, C. J. Taylor, A. Lanitis és T. F. Cootes, *Tracking and Recognising Hand Gestures Using Statistical Shape Models* Dept. of Medical Biophysics, University of Manchester
BMVC '95 Proceedings of the 6th British conference on Machine vision (Vol. 2)
Pages 403 – 412, ISBN:0-9521898-2-8

[Ban1] Szegedi Tudományegyetem, Bánhalmi András, Adatbányászat, Naive Bayes, HMM
http://www.inf.u-szeged.hu/~banhalmi/adatb/9_eloadas.ppt Látogatva: 2012.11.10.

[Bar94] J.L. Barron, D.J. Fleet and S.S Beauchemin *Performance of Optical Flow Techniques*, Dept of Computer Science, University of Western Ontario, Dept of Computing Science, Queens University.
International Journal of Computer Vision, Volume 12 Issue 1, Feb. 1994, Pages 43 – 77,
ISSN:0920-5691

[Bay08] Herbert Bay, Tinne Tuytelaar, Luc Van Gool, *SURF: Speeded Up Robust Feature*, ETH Zurich és Katholieke Universiteit Leuven
Computer Vision and Image Understanding Volume 110 Issue 3, June, 2008
Pages 346-359, ISSN:1077-3142

[Ber11] Michael Van den Bergh, Luc Van Gool, *Combining RGB and ToF Cameras for Real-time 3D Hand Gesture Interaction*, ETH Zurich, Switzerland és KU Leuven, Belgium
WACV '11 Proceedings of the 2011 IEEE Workshop on Applications of Computer Vision (WACV), Pages 66-72 , ISBN: 978-1-4244-9496-5

[Che03] Feng-Sheng Chen, Chih-Ming Fu, Chung-Lin Huang, *Hand gesture recognition using a real-time tracking method and hidden Markov models*, Institute of Electrical Engineering, National Tsing Hua University, Hsin Chu 300, Taiwan, ROC, Image and Vision Computing 21 (2003) 745–758

[Con08] A. Conci, E. Nunes, Juan José Pantrigo, Ángel Sánchez, *Comparing Color and Texture-Based Algorithms for Human Skin Detection*, ICEIS 2008, Volume HCI, Barcelona, Spain, June 12-16

[Elm08] Mahmoud Elmezain, Ayoub Al-Hamadi, Jörg Appenrodt, and Bernd Michaelis, *A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition*, Institute for Electronics, Signal Processing and Communications (IESK), Otto-von-Guericke-University Magdeburg, Germany, ISSN 2070-3740

[Fang07] Yikai Fang, Kongqiao Wang, Jian Cheng and Hanqing Lu, *A real-time hand gesture recognition method*, National Lab of Pattern Recognition, Nokia Research Center, Multimedia and Expo, 2007 IEEE International Conference on, 2-5 July 2007, page 995- 998, Conference Publication

[Lah10] Hervé Lahamy and Derek Litchi, *Real-time Hand Gesture Recognition Using Range Cameras*,

Department of Geomatics Engineering, University of Calgary, Canada, 2010

http://www.isprs.org/proceedings/XXXVIII/part1/03/03_03_Paper_54.pdf Látogatva: 2012.11.10.

[Law89] R. R. Lawrence, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceeding of the IEEE, Vol. 77(2), pp. 257-286, 1989

[Li09] Zhi Li, Ray Jarvis, *Real time Hand Gesture Recognition using a Range Camera*, Monash University, Wellington Road Clayton, Victoria AUSTRALIA

Australasian Conference on Robotics and Automation (ACRA), December 2-4, 2009, Sydney, Australia, Conference Publication

[Luc81] Bruce D. Lucas and Takeo Kanade, *An Iterative Image Registration Technique with an Application to Stereo Vision*. IJCAI'81 Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2, Pages 674-679, Conference Publication

[Pat04] Kaustubh S. Patwardhan, Sumantra Dutta Roy, *Dynamic Hand Gesture Recognition using Predictive EigenTracker*, Department of Electrical Engineering, IIT Bombay, Powai, Mumbai - 400 076, INDIA, 2004

http://www.cse.iitd.ac.in/~sumantra/publications/icvgip04_gesture.pdf Látogatva: 2012.05.15

[Ren11] Zhou Ren, Junsong Yuan, Zhengyou Zhang, *Robust Hand Gesture Recognition Based on Finger Earth Mover's Distance with a Commodity Depth Camera*, Nanyang

Technological University, Singapore, Microsoft Research Redmond Washington, USA, 2010
MM '11 Proceedings of the 19th ACM international conference on Multimedia
Pages 1093-1096, ISBN: 978-1-4503-0616-4

[Rig97] Gerhard Rigoll, Andreas Kosmala, Stefan Eickeler, *High Performance Real-Time Gesture Recognition Using Hidden Markov Model*, Gerhard Mercator University Duisburg, Germany, 1997, Proceedings of the International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction, Pages 69 – 80, ISBN:3-540-64424-5

[Ser10] Sergyán Szabolcs, Szűrész, Óbudai Egyetem, 2010, <http://users.nik.uni-obuda.hu/vamosy/Kepfeldolgozas2011/02Filtering/Filtering.pdf>

Látogatva: 2012.11.10.

[Smith95] S. M. Smith, J. M. Brady, *SUSAN - A New Approach to Low Level Image*

Processing, Department of Engineering Science, Oxford University, Oxford, UK, 1995

International Journal of Computer Vision, Volume 23 Issue 1, Pages 45 – 78, ISSN:0920-5691, May 1997

[Sta12] Mark Stamp, *A Revealing Introduction to Hidden Markov Models*, Department of Computer Science, San Jose State University,

<http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf> Látogatva: 2012.12.29.

[Ste10] Ilan Steinberg, Tomer M. London, Dotan Di Castro, *Hand Gesture Recognition in Images and Video*, Department of Electrical Engineering, Technion- Israel Institute of

Technology, CCIT Report #763, EE Pub No. 1720, March 2010.

http://webee.technion.ac.il/people/dot/Steinberg_London_DiCastro_Tech_Report.pdf

Látogatva: 2012.11.10.

[Tang11] Matthew Tang, *Recognizing Hand Gestures with Microsoft's Kinect*, Department of Electrical Engineering Stanford University, 2011

http://www.stanford.edu/class/ee368/Project_11/Reports/Tang_Hand_Gesture_Recognition.pdf Látogatva: 2012.11.10.

[Tom91] Carlo Tomasi and Takeo Kanade. *Detection and Tracking of Point Features*. Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

<http://www.lira.dist.unige.it/teaching/SINA/slides-current/tomasi-kanade-techreport-1991.pdf>

Látogatva: 2012.11.10.

[Vám04] Vámosy Zoltán, Szegmentálás II., Budapesti Műszaki Főiskola 2004-2006,

<http://users.nik.uni-obuda.hu/vamosy/GepiLatas2007/8Szegegmentalas/Szegegmentalas.ppt> ,

Látogatva: 2012.11.10.

[Vám08] Budapesti Műszaki Főiskola, Vámosy Zoltán, Optical Flow, 2008

<http://users.nik.uni-obuda.hu/vamosy/GepiLatas2007/8MotionOpticalFlow/OF2008.ppt>

Látogatva: 2012.11.10.

[Ves01] Ves-Matti Mantyla, *Discrete hidden Markov models with application to isolated user-dependent hand gesture recognition*, Technival Research Centre of Finland, Espoo, 2001, ISBN: 9513858758, 9789513858759

[Vio04] Paul Viola, Michael J. Jones, *Robust Real-Time Face Detection*, Microsoft Research, Mitsubishi Electric Research Laboratory

International Journal of Computer Vision, Volume 57 Issue 2, Pages 137-154, ISSN:0920-5691, May 2004

[Wang09] Robert Y. Wang, Jovan Popović, *Real-Time Hand-Tracking with a Color Glove*, Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology

SIGGRAPH '09 ACM SIGGRAPH 2009 papers

Article No. 63, ISBN: 978-1-60558-726-4

[Yang99] Ming-Hsuan Yang és Narendra Ahuja, *Recognizing Hand Gesture Using Motion Trajectories*, Department of Computer Science and Beckman Institute University of Illinois at Urbana-Champaign, Urbana, IL 61801

1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99) - Volume 1, Fort Collins, Colorado, June 23-June 25

ISBN: 0-7695-0149-4

[Yör06] Erdem Yörük, Ender Konukoglu, Bülent Sankur, Senior Member, IEEE, and Jérôme Darbon, *Shape-Based Hand Recognition*,

IEEE Transaction on Image Processing, vol. 15, no. 7, July 2006, pages 1803-1815,

ISSN:1057-7149

[Zab09] X. Zabulis, H. Baltzakis, A. Argyroszy, *Vision-based Hand Gesture Recognition for Human-Computer Interaction*, Institute of Computer Science, Heraklion, Crete, Greece, Computer Science Department, University of Crete, Heraklion, Crete, Grece, 2009
Vision-based Hand Gesture Recognition For Human Computer Interaction: A Review,
Editor: Katherine Blashki, Pages 125-132 ISBN: 978-972-8924-59-1

[Ziv02] Ziv Yaniv, *Edge Detection*, School of Engineering and Computer Science
The Hebrew University, Jerusalem, Israel, 2002
http://isiswiki.georgetown.edu/zivy/writtenMaterial/edge_detection.pdf Látogatva:
2012.11.10.