



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

3D MODELLEZÉS TÁMOGATÁSA GESZTUSFELISMERÉSSSEL

Szerző: **Kemenes Ádám**
mérnök informatikus szak, 4. évf.

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

Budapest, 2012.

Tartalomjegyzék

1 Bevezetés.....	3
1.1 A projekt célja.....	3
1.2 Gesztusfelismerés [1].....	3
1.3 Gesztusfelismeréshez használt eszközök.....	3
1.4 Alkalmazott algoritmusok.....	4
1.5 A rendszer előnyei.....	4
1.6 Technológia.....	4
2 Az irodalom bemutatása.....	6
2.1 Hardver.....	6
2.1.1 Kinect szenzor.....	6
2.1.2 ToF (Time of Flight) kamera.....	7
2.1.2.1 A ToF kamera részei.....	7
2.1.3 Sztereó kamera [6].....	8
2.1.3.1 Torzításmentesítés.....	8
2.1.3.2 Kalibráció.....	9
2.1.3.3 POSIT algoritmus.....	9
2.1.3.4 Sztereó képalkotás.....	10
2.2 Gesztusfelismerés, kezdetektálás.....	12
2.2.1 Arcfelismerés.....	12
2.2.2 Kézfelismerés.....	12
2.2.3 Statikus gesztusok felismerése.....	12
2.2.4 Dinamikus gesztusok felismerése.....	12
2.2.5 Boosted classifier.....	13
2.2.6 Meanshift algoritmus.....	13
2.2.7 k legközelebbi szomszéd (kNN) algoritmus [10].....	13
2.2.7.1 Egydimenziós kNN algoritmus.....	14
2.2.7.2 Kétdimenziós kNN algoritmus.....	14
2.2.8 Dynamic time warping [11].....	14
2.2.9 Spline interpoláció.....	16
2.3 Beépülők fejlesztése Blenderhez.....	17
3 A rendszer bemutatása.....	18
3.1 Cél.....	18
3.2 Rendszervázlat.....	18
3.3 A rendszer felépítése.....	19
3.3.1 Mélységi kép.....	19
3.3.2 Szkeletonizáció.....	19
3.3.3 Zajszűrés.....	19
3.3.4 Gesztusfelismerés.....	20
3.3.5 Nyers adatok megjelenítése.....	20
3.3.6 Beavatkozás.....	20
3.4 Az elkészült példaprogram.....	21
3.5 Továbbfejlesztési lehetőségek.....	22
3.5.1 Megvalósítás egyéb eszközzel.....	22
3.5.2 Integrálás a Blenderbe, vagy más modellező szoftverekbe.....	22
3.5.3 Zajszűrő paraméterek automatikus beállítása.....	22
3.5.4 Dinamikus gesztusok kezelése.....	22
3.5.5 Görbe bevitele dinamikus gesztussal.....	22
3.5.6 A gesztusfelismerő algoritmus robusztusságának növelése.....	22
4 Eredmények.....	23

1 Bevezetés

1.1 A projekt célja

3D modellezés során a modelltér egy adott nézőpontjának pontos beállítása gyakran nehézkes, az egér/billentyűzet nem ad megfelelő pontosságot. Kiváltható CAD alkalmazásokhoz tervezett speciális pozicionálóval, ez azonban költséges lehet. A projektem a modelltérben a nézőpont mozgását, forgatását, objektumok manipulálását teszi lehetővé kézmozdulatok segítségével.

1.2 Gesztusfelismerés [1]

A számítástudomány emberi gesztusok matematikai algoritmusok segítségével történő leírásával, értelmezésével foglalkozó része. Gesztusok alatt az emberi test, általában az arc, illetve a kéz meghatározott mozdulatait értjük. Gépi látás segítségével számos megközelítést tettek már arcról érzelmek leolvasására, kézmozdulatok értelmezésére, jelbeszéd szöveggé fordítására, testbeszéd értelmezésére. Gesztusfelismeréssel a jelenlegi szöveges, és grafikus felhasználói felületeknél sokkal természetesebb kapcsolat alakítható ki ember és gép között. A gesztusfelismerés lehetővé teszi, hogy az ember természetesebben tudjon kommunikálni a számítógéppel beviteli eszközök nélkül. Ez a technológia forradalmasítaná a számítógépek kezelését, szükségtelessé tenné az egerek, billentyűzetek, érintőképernyők használatát.

A gesztusfelismerés gépi látás, és képfeldolgozási technikákkal közelíthető meg.

A szakirodalomban néha ide sorolják az érintőképernyőkön értelmezett ujj-, vagy érintőceruza mozdulatok, valamint egérmozdulatok feldolgozását.

Gesztusok típusai:

- Offline: A kézmozdulat végrehajtása után kerül értelmezésre, és meghívódik a megfelelő funkció (például egy menü megnyitása)
- Online: A kézmozdulat közben folyamatosan értelmezésre kerül, és vele arányosan módosul az alkalmazás állapottára. Ilyen tipikusan a mozgatus, forgatus, méretezés.

1.3 Gesztusfelismeréshez használt eszközök

- Speciális kesztyű, amely mágneses tér érzékelésével képes pozíciójának meghatározására. Előnye, hogy haptikus visszajelzést tud adni a felhasználónak
- Mélységérzékelő kamera: Mélységi térképet tud kialakítani, így a látómezőjében elhelyezkedő objektumok kamerától mért távolsága megadható. (Microsoft Kinect)
- Sztereoszkopikus kamera: Két kamera képéből háromszögeléssel számítható ki a látómezőben levő objektumok távolsága, ha tudjuk a két kamera egymáshoz viszonyított távolságát.
- A felhasználó kezében tartott, gyorsulásérzékelővel felszerelt eszköz, amely mozdításkor a gyorsulás mértékét, és irányát képes meghatározni. (Wii Remote)
- Egy kamera: A fent felsoroltakhoz képest kevésbé hatékony, mert térbeli mozgást kisebb pontossággal tud leképezni, de szélesebb felhasználói körben alkalmazható.

1.4 Alkalmazott algoritmusok

Attól függően, milyen típusú kézmozdulatokat, mekkora pontossággal szeretnénk feldolgozni többféle megoldást választhatunk, habár legtöbbjük a mozgást egy háromdimenziós koordináta rendszerben képzi le. A test különböző mozdulatainak értelmezéséhez, osztályoznunk kell őket közös tulajdonságaik alapján.

- 3D modell alapú algoritmus
A mélységi térkép alapján egy háromdimenziós modellt épít fel a testről. A modell elég részletes lehet, és a rendszer a legfinomabb mozdulatokra is képes reagálni. Hátránya azonban, hogy ennek nagy az erőforrásigénye.
- Váz alapú algoritmus
A testen csak néhány pontot (tipikusan ízületeket) vizsgálunk. Az előzőhöz képest kevésbé pontos módszer, de jóval erőforrás-takarékosabb.
- Alakfelismerésen alapuló algoritmus
A kameraképen megpróbáljuk felismerni az embert és testtartását vizsgálni.

1.5 A rendszer előnyei

Egér használatával a háromdimenziós modell térben elhelyezkedő objektumok csak a monitor síkjára vetített irányokban manipulálhatók. Az én megoldásom három dimenzióban lenne képes objektumok manipulációjára, meggyorsítaná, pontosabbá tenné a modellezést. Ahogy a Kinect szenzor forradalmasította a videojátékok vezérlését, úgy technikai jellegű alkalmazásokban is megállja a helyét. Fejlesztés során megoldást keresek majd az egér és billentyűzet hosszú távú használatából eredő egészségügyi problémák kiküszöbölésére is.

1.6 Technológia

Fejlesztéshez a Blender nevű 3D-s modellező programot választottam, mert régóta ismerem, használok, és mert nyílt forráskódú, tehát a benne levő megoldásokat szabadon lehet tanulmányozni, valamint a megvalósítás korlátai miatt felmerült a Blender forráskódjába való integrálás is. A választott programozási nyelv C++ és Python mert a Blender API-ja ehhez ad támogatást, felhasználásra kerül az OpenCV, OpenNI gyűjtemény. a Kinect SDK-t. A gesztusfelismerés Kinect szenzorral történik, mivel a rendszernek háromdimenzióban kell tudnia lekövetni a felhasználó kézmozdulatait. A szoftver pedig a Blenderhez egy beépülő modul formájában kerül megvalósításra. Cross platform kompatibilitásra törekszem, Windows, Linux, és Apple OS X rendszereken, valamint tanulmányozom a megvalósítás lehetőségét más 3D modellező szoftvereken mint pl. a 3ds Max, vagy a Maya. a szélesebb körű felhasználhatóság érdekében.

A felhasználó kézmozdulatainak lekövetésére egy lehetséges megoldás az ún. time-of-flight kamera, amely infravörös fénnel világítja meg a vizsgált objektumot, majd a fény kibocsátása és visszaverődése között eltelt időből pontosan meghatározza a tárgy kamerától mért távolságát. A Kinectben ez adott, mint ahogy az emberi test detektálása háromdimenzióban, ezért a programozási feladat szűkíthető a karmozdulatok felismerésére, és arra reagálásra. (a modellező szoftver megfelelő funkciójának meghívásával)

Ha hagyományos technikával oldjuk meg a feladatot, a probléma megoldása bonyolultabb. A gépnek három dimenzióban kell látni, erre két kamera szükséges. Négy lépésben tudunk mélységet mérni két kamera segítségével:

1. A kamera lencséje által okozott torzítást eltávolítjuk. („ellentorzítás”) Nyomtatott ráccsal, vagy sakktábla mintával kalibrálható.
2. A két kamera által rögzített képet egymásra vetítjük
3. Kivonjuk egymásból a két kamera képét, megkapjuk az ún. diszparitástérképet.
4. Ha tudjuk a két kamera térbeli elhelyezkedését, az egyes pontok a kamerák síkjától vett távolságát megadhatjuk háromszögeléssel.

Az így kapott mélységi térképből felismerjük a felhasználó testét, karmozdulatai detektálhatók, majd ezek alapján meghívhatóak a modellező szoftver megfelelő funkciói.

2 Az irodalom bemutatása

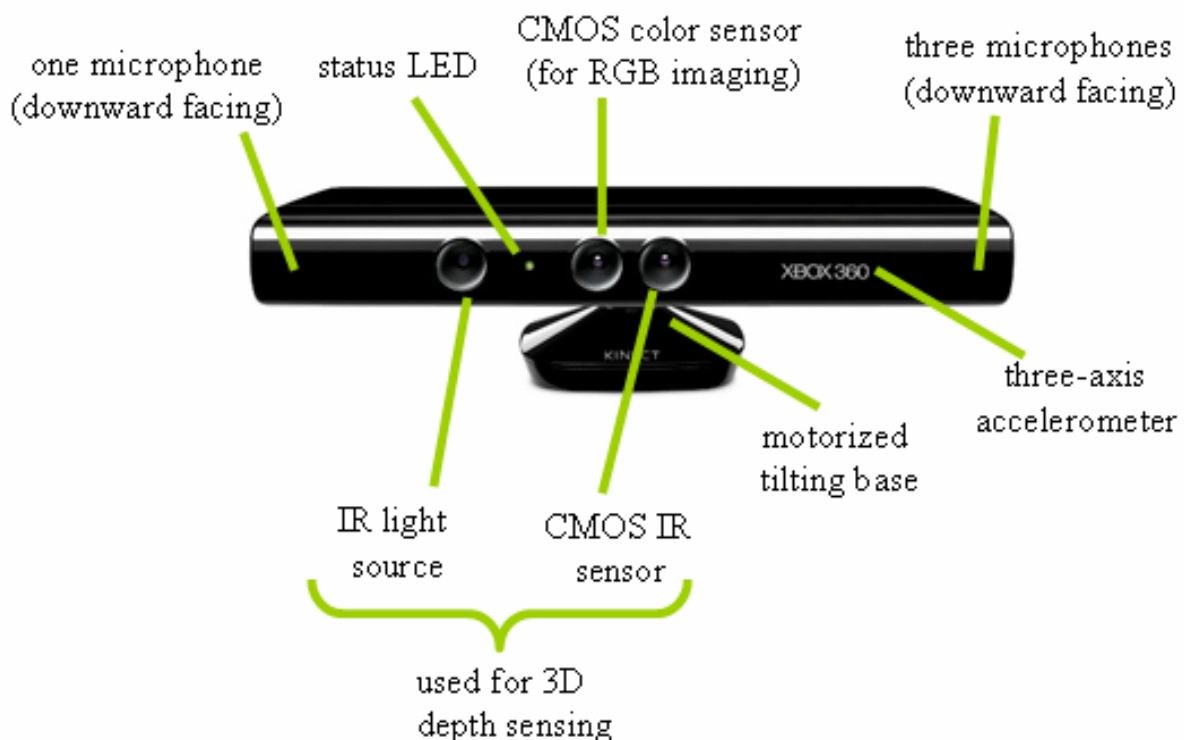
Tanulmányoztam a megvalósítás lehetőségét Kinect szenzorral, (ill. a későbbiekben lehetőségeimhez mérten az Asus Xtion Pro termékéhez) OpenNI függvénykönyvtárral, valamint két webkamerával OpenCV segítségével. A megvalósítás lényegében a felhasználó kezének detektálásából, azon jellemző pontok felvételéből, (ízületeknél) valamint ezen pontok térbeli elmozdulásának vizsgálatából áll. A kritikus funkcionalitás C++-ban kerül megvalósításra az említett programkönyvtárak felhasználásával.

2.1 Hardver

A probléma megközelítéséhez, ismernünk kell a képalkotáshoz használt hardverek működését.

2.1.1 Kinect szenzor

A Kinect szenzort (1. ábra) eredetileg az Xbox 360 játékkonzolhoz fejlesztették ki. Egy mozgásérzékelős bemeneti eszköz, amely játékok vezérlését teszi lehetővé kézmozdulatok, és hangparancsok segítségével. A legfontosabb részei az RGB kamera, a mélységérzékelő, illetve mikrofon, dőlésszöget szabályzó motor, illetve egy háromtengelyű dőlésérzékelő.[2]



1. ábra: A Kinect szenzor felépítése [2]

A mélységérzékelő egy infravörös fényforrást tartalmaz amely változó intenzitású pontokat vetít ki, (látszólag véletlenszerűen) majd figyeli a mintában keletkező torzulásokat, ebből következtet az objektumok tőle mért távolságára.

A mélységérzékelő által alkotott kép VGA felbontásban (640×480) 11 bites mélységi

értékekkel kerül tárolásra, 30 fps mintavételi sebességgel. Az RGB kamera szintén VGA felbontással, 30 fps mintavételi sebességgel dolgozik.

A hangrendszer 4 darab egyenként 16 bit/16 kHz-es mikrofonból áll.

A szenzor a felhasználót legalább 1,8 méteres távolságból képes detektálni (Xbox 360 verzió) [3][4].

A legjobb fejlesztőeszközök Kinecthez az OpenNI, illetve a Microsoft Kinect SDK.

2.1.2 ToF (Time of Flight) kamera

A ToF kamera az általa kibocsátott fény kibocsátásának pillanata, és a visszaverődés pillanata között eltelt időből a fénysebesség ismeretében határozza meg az előtte elhelyezkedő objektumok távolságát.

2.1.2.1 A ToF kamera részei

- Fénykibocsátó egység: Mivel nagy frekvenciával kell villognia, általában infravörös LED-et, vagy lézerdíódát alkalmaznak.
- Optika: A lencse összegyűjti környezetből visszaverődő fényt, egy sávszűrő biztosítja az infravörös tartományra való szűrést, illetve a háttérzajok kiszűrését.
- Képfeldolgozó: Ez a ToF kamera lelke. Minden pixel a fénykibocsátó egység által kibocsátott fény által megtett utat számítja ki, a fény kibocsátásának, ill. visszaverődésének időpillanata, és a fénysebesség ismeretében.
- Vezérlő elektronika: A fénykibocsátónak, és a képfeldolgozónak is nagyon gyorsan kell működnie (pl. ha a kibocsátás, és a visszaverődés között 10 pikoszekundum telik el, az objektum 1,5 mm-re van. Egy 3 GHz-es processzor óraciklusa 300 ps, ez csak 45 mm-es pontosságot enged meg.)
- Számítás/felület: A távolságszámítások a kamerában értékelődnek ki. A megfelelő teljesítmény elérése érdekében kalibrációt használnak. A kamera távolsági térképet alkot, és USB vagy ethernet felületen továbbítja.

A legegyszerűbb ToF kamera fényimpulzusokat használ. A világítás nagyon rövid időre bekapcsol, megvilágítja a környezetet, a fény visszaverődik a környezetben elhelyezkedő objektumokról. A kamera lencséje összegyűjti a visszaverődő fényt, és az érzékelő felületre vetíti. A fény annál később ér vissza a kamerába, minél messzebből verődött vissza. Mivel a fénysebesség kb. 300 millió m/s ez a késleltetés nagyon kicsi. Pl. egy 2,5 méterre elhelyezkedő objektumról a késleltetés így számítható ki:

$$t_d = 2 \cdot \frac{D}{c} = 2 \cdot \frac{2,5 \text{ m}}{300\,000\,000 \frac{\text{m}}{\text{s}}} = 16,66 \text{ ns}$$

A megvilágítás impulzusszélessége meghatározza, a kamera mekkora távolságot tud lefedni. Pl. egy 50 ns-os impulzus maximális hatótávolsága a következőképpen számítható ki:

$$D_{\max} = \frac{1}{2} \cdot c \cdot t_0 = \frac{1}{2} \cdot 300\,000\,000 \frac{\text{m}}{\text{s}} \cdot 50 \text{ ns} = 7,5 \text{ m}$$

Mivel ezek az idők nagyon rövidek, a megvilágítás a megvalósítás szempontjából nagyon kritikus. Csak nagyon gyors LED-ek, illetve lézerdíódák alkalmazhatók erre a célra.

Egy pixel egy fényérzékeny elemből, pl. egy fotódíótából áll, amely a beérkező fényt

elektromos árammá alakítja. A fény kibocsátásának pillanatában elindul egy digitális számláló (nagyon gyors, több gigahertzes) amelyet a beérkező fény által keltett impulzus állít le. Mivel az érzékelő felületet a kibocsátott fény csak kis része éri el, ezért 50 ns-on belül nem egy, hanem több ezer fényimpulzus kerül kiküldésre, ez növeli a jel-zaj arányt. Az „exponálás” után kiolvassuk a számlálók értékét, az eltelt időt megszorozzuk a fénysebességgel, és megkapjuk a mért távolságot [5].

2.1.3 Sztereó kamera [6]

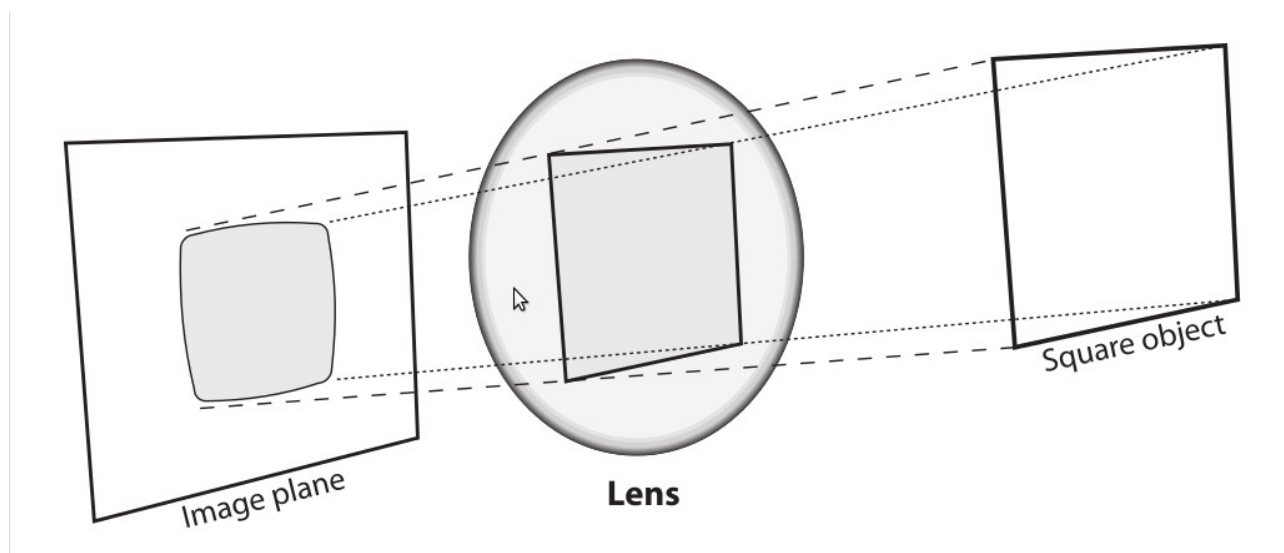
Ennek a megoldásnak előnye, hogy technológiailag egyszerű, olcsó eszközökkel megvalósítható, hátránya a kalibráció szükségessége, és a programozástechnikailag nehezebb megvalósítás.

Ha egyszerű, olcsó webkamerákat használunk, tudnunk kell ezen képalkotó eszközök tipikus hibáiról, ezek korrigálásáról.

2.1.3.1 Torzításmentesítés

Torzításokat maga a lencse, illetve az összeszerelés hibái idézhetnek elő. Ahhoz, hogy pontosan tudjunk mérni a képen ezen torzítások mértékét csökkenteni kell.

Radiális torzítás: Ezt a fajta torzítást az okozza, hogy az eszközökben az olcsóbb kivitelezhetőség miatt ideális parabolikus lencsék helyett gömbszerű lencsét alkalmaznak, valamint a képalkotó általában nem a lencse fókuszpontjában helyezkedik el. Ez a torzítás okozza az ún. hordó, vagy halszem effektust. A lencse közepétől távolabb eső fénysugarak jobban elhajlanak, mint a lencse közepén. Ez olcsó webkamerák esetén jól megfigyelhető, drága, csúcskategóriás eszközöknél igyekeznek a matematikailag ideálist jobban megközelítő lencsét előállítani, illetve a képalkotót a fókuszpontba helyezni. A radiális torzítás a kép közepén 0, és a kép szélei felé haladva növekszik.



2. ábra: Radiális torzítás [6]

A torzítás a következőképpen korrigálható:

$$x_{\text{korrigált}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{\text{korrigált}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

ahol x, y a képen adott pixel koordinátái, $x_{\text{korrigált}}, y_{\text{korrigált}}$ az adott pixel

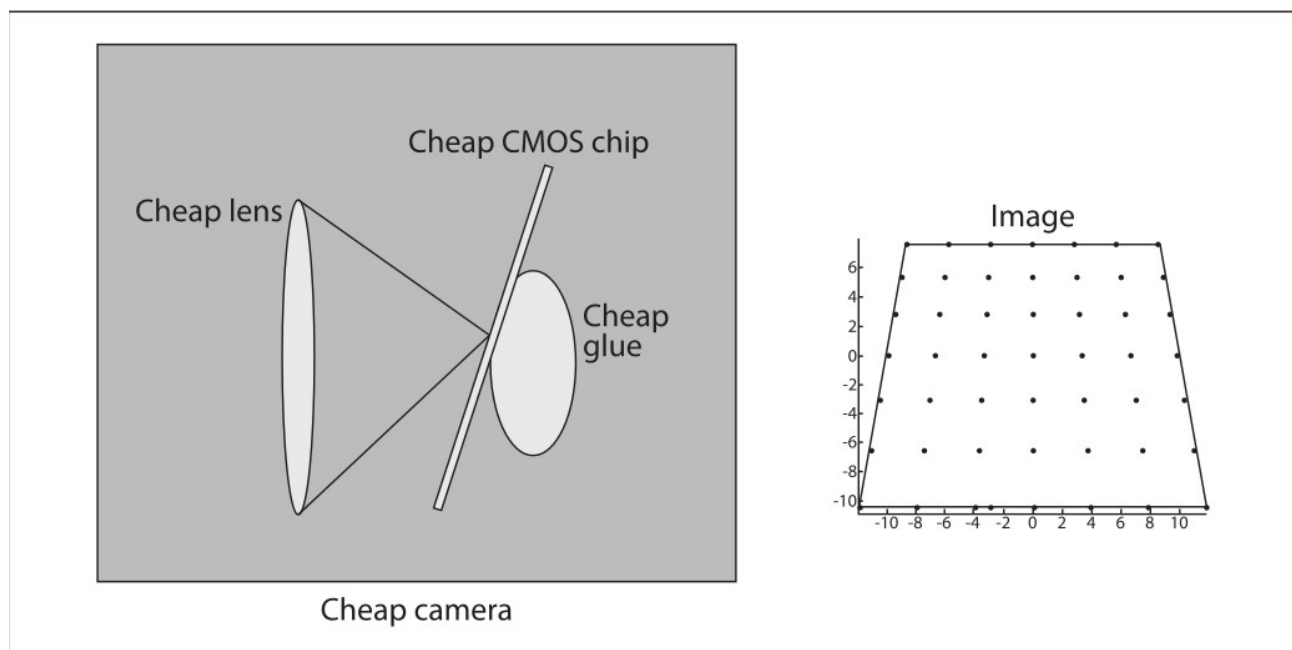
torzításmentesítés után kiszámított koordinátái, r a lencse sugara, k_1, k_2, k_3 a kompenzáció paraméterei.

Tangenciális torzítás: Ez a fajta torzítás a gyártás hibáiból adódik, a lencse nem párhuzamos a képalkotóval. Ez a torzítás két paraméterrel (p_1, p_2) adható meg, és a következőképpen korrigálható:

$$x_{\text{korrigált}} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$

$$y_{\text{korrigált}} = y + [p_1(r^2 + 2y^2) + 2p_2 x]$$

ahol x, y a képen adott pixel koordinátái, $x_{\text{korrigált}}, y_{\text{korrigált}}$ az adott pixel torzításmentesítéssel kiszámított új koordinátái, p_1, p_2 a torzítás paraméterei.



3. ábra: A tangenciális torzítást általában a ragasztás hibái okozzák [6]

2.1.3.2 Kalibráció

A kalibráláshoz általában sakktáblát használunk. Használhatnánk háromdimenziós objektumokat is, léteznek ilyen kalibrációs eljárások, de a sakktábla jól detektálható, és könnyebben kivitelezhető. A sakktábla egyenlő szélességű váltakozó fekete, és fehér négyzetekből áll. Mivel a kalibráció során egy síkbeli objektumot vizsgálunk, azt több látószögből kell rögzíteni. A kalibráció során vizsgáljuk a sakktábla képének torzulását, ezek alapján próbáljuk megbecsülni a torzítások paramétereit. A legjobb eredmény eléréséhez legalább 10 kép kell egy 7×8-as sakktábláról.

2.1.3.3 POSIT algoritmus

(Position from Orthography and Scaling with Iteration) 1992-ben publikálták olyan objektumok helyzetének, és orientációjának detektálására, amelyeknek méreteit pontosan ismerjük. Hogy ezt kiszámíthassuk az objektumon legalább négy nem egy síkba eső pontot kell találnunk. Az algoritmus feltételezi, hogy ezek a pontok azonos mélységben helyezkednek el, és az eredeti objektumtól való méreteltéréseket csak a kamerától való távolság miatti skálázódás okozza (torzításmentes a kép). Feltételezzük, hogy a tárgy elég

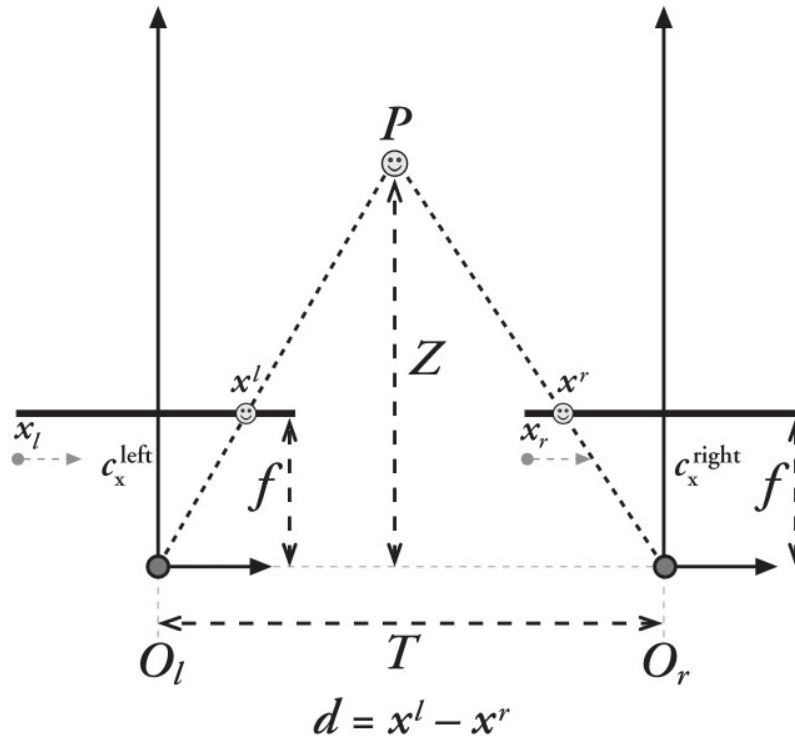
messze van a kamerától, hogy a belső mélységkülönbségeket elhanyagolhassuk. Ez az ún. gyenge perspektívaközelítés. Ha ismerjük a kamera belső paramétereit, megközelítőleg ki tudjuk számítani az objektum helyzetét. Ez a számítás nem túl pontos, de elég ahhoz, hogy a megtalált négy pont 3D helyzetét kiszámíthassuk az eredeti objektumon. Ezután keresünk négy új pontot, azok helyzetét is meghatározzuk, és így tovább. Az algoritmus három-négy iteráció után megfelelő pontosságú becslést tud adni az objektum helyzetére.

2.1.3.4 Sztereó képpalkotás

A számítógépek úgy képesek sztereó térlátásra, hogy a két kamera által alkotott képeken megpróbálják feltárni az összefüggéseket. Ha ismerjük az összefüggéseket, valamint a kamerák közti távolságot, kiszámíthatjuk az egyes pixelek távolságát a kamerák síkjától. Mivel az összefüggések keresése elég számításigényes művelet, hogy a keresési teret csökkenthessük a geometria eszközeit hívjuk segítségül. A gyakorlatban a sztereó képpalkotás négy lépésből áll.

1. Torzításmentesítés – mindkét kamera képen csökkentjük a torzítások hatását.
2. Rektifikáció – a kamerák helyzetét úgy igazítjuk, hogy a képpalkotók egy síkba essenek.
3. Megkeressük mindkét kamera képen a hasonló jellemzőket. Ennek a műveletnek az eredménye a diszparitástérkép, ahol a diszparitások a két kamera képen az x koordináták különbsége
4. Ha tudjuk a kamerák térbeli elhelyezkedését, a diszparitástérkép távolsági térképpé alakítható háromszögelés segítségével.

Háromszögelés: Feltételezzük, hogy mindkét kameránk torzításmentes, és a képpalkotók egy síkban vannak (a valóságban ez sosem igaz, ezért számolnunk kell a hibákkal is), a két kamera lencséjének fókusztávolsága egyenlő, valamint optikai középpontjaik mindkét képen ugyanazokra a koordinátákra esnek.



4. ábra: Háromszögelés [6]

Továbbá tételezzük fel, hogy a két kamera képe sorillesztett, azaz hogy az egyik kép adott sora tökéletesen illeszkedik a másik kép megfelelő sorához (ez a valóságban szintén sosem igaz, az elcsúszással is számolnunk kell). Keresünk egy pontot a térben, amelynek ismerjük mindkét kamerán vett koordinátáit (x^l, x^r) . Kiszámoljuk a diszparitást $(d = x^l - x^r)$, ez alapján pedig hasonló háromszögek keresésével meghatározható a pont kamerák síkjától vett távolsága.

Kalibráció: A kalibráció során azt a forgatási mátrixot, és eltolási vektort keressük, amellyel a két kamera képe pontosan egymásra vetíthető. Ha egy adott pontnak ismerjük mindkét kamerán a koordinátáit, az adott pontra vonatkozóan könnyen meg tudjuk határozni ezeket az adatokat:

$P_l = R_l P + T_l$ a bal kamerára, és $P_r = R_r P + T_r$ a jobb kamerára, ahol P_l az adott pixel koordinátái a bal kamerából P_r a jobb kamerából R_x, T_x a megfelelő kamerára vonatkoztatott forgatómátrix, ill. eltolási vektor, P pedig a 3D pont. Ezek ismeretében a következő egyenletrendszert kell megoldanunk:

$$R = R_r (R_l)^T$$

$$T = T_r - R T_l$$

Rektifikáció: Tökéletesen illesztett képeken a legkönnyebb diszparitást számolni, azonban, ahogy fentebb említettük hibák mindig adódnak. A rektifikáció során ezeket a hibákat próbáljuk kiküszöbölni. A két képen megkeressük a hasonló tulajdonsággal rendelkező régiókat, ezeket próbáljuk egymáshoz illeszteni.

2.2 Gesztusfelismerés, kézdetektálás

2.2.1 Arcfelismerés

A módszer lényege, hogy a kameraképen felismerjük a felhasználó arcát, ebből bőrszín mintát tudunk venni, ehhez képest már könnyebben meg tudjuk határozni a felhasználó kezeinek helyzetét. Ehhez fokozott osztályozókból (boosted classifier) álló hálót, és a meanshift algoritmust használjuk, a következő képkockán az arc elhelyezkedésének előrejelzésére.

2.2.2 Kézfelismerés

A kéz geometriailag meglehetősen bonyolult objektum (több mint 25 szabadsági foka van), dinamikus környezetben, változó világítás mellett detektálása nehéz feladat. Erre több rendszert fejlesztettek már, korai módszerek jelölők, vagy színezett kesztyűk hordását követelték meg a felhasználójuktól. Az újabb módszerek bőrszín-, ill. alakdetektáláson alapulnak, bár ezek a rendszerek nem elég robusztusak dinamikus környezetben való felhasználásra. Általában homogén háttérrel követelnek meg, és az egész képen csak egy középre helyezett kezet. A fokozott osztályozók sokkal robusztusabb, és gyorsabb kézdetektálást tesznek lehetővé, továbbá ezek az osztályozók használhatók statikus gesztusok felismerésére. Dinamikus gesztusokat általában rejtett Markov modellekkel vizsgálunk.

2.2.3 Statikus gesztusok felismerése

A kézdetektor kimenetét egy fokozott osztályozókból álló hálóra küldjük, majd a megfelelő gesztusra tanított osztályozó igaz értéket ad vissza (én betanítanám a rendszert, amikor már megfelelően működik, a tanítás eredményét adatbázisba menteném, a gyorsabb működés érdekében). A külső zajok, illetve az adott gesztus bizonytalansága miatt több gesztusfelismerő osztályozó adhat egyszerre igaz értéket. Az ezáltal okozott pontatlanság kiküszöbölésére több osztályozó tömböt alkalmazunk, majd annak a kimenetét használjuk fel, amelyik a legtöbb igaz értéket adta.

2.2.4 Dinamikus gesztusok felismerése

Dinamikus gesztusfelismerés közben egy sor egymást követő detektált kezet vizsgálunk. Az eltárolható detektálások száma fix, ha az erre szánt memória megtelik, a legrégebbit eldobjuk, majd egy új veszi át a helyét. A teljes eltárolt detektálásokat eldobjuk, ha az inaktivitási feltétel teljesül (a kéz kimegy a kameraképből, vagy álló kéz van a képen). Ha egy új detektálás érkezik, azzal együtt az összes eddigi letárolt detektálást kiértékeljük. Ez az analízis egy jellemzővektort számít a vizsgált sorozat geometriai és kinematikai adataiból. Minden részsorozatból számított jellemzővektor egy naiv Bayes osztályozóba kerül, amely az összes lehetséges gesztusra meghatároz egy pontszámot, amely az adott gesztus valószínűségét fejezi ki. Mindig azt a gesztust feltételezzük, amelynek az adott időpillanatban a legnagyobb a pontszáma (valószínűsége). Minden képkockára csak a legnagyobb pontszámot tartjuk meg. A legnagyobb pontszámok használhatók az éppen végrehajtott gesztus meghatározására, de, hogy a robusztusságot növeljük. egy

időintervallumon belül a pontszámoknak egyenletesnek kell maradniuk. Minden gesztusra a legutolsó k darab legnagyobb pontszám mozgóátlagát tartjuk meg. Bármely adott időpillanatban a legnagyobb mozgóátlag pontszámmal rendelkező gesztust határozzuk meg. Mivel a gesztus végét nem tudjuk meghatározni bármely képkockából, a szegmentálás problémás lesz. A mozgóátlagok küszöbölése egy lehetséges megközelítése a problémának. A küszöbértéket tanítás útján határozzuk meg.[7]

2.2.5 Boosted classifier

A boosting a felügyelt tanítás egy formája, amely több gyenge osztályozóból próbál egy erős osztályozót alkotni. A gyenge osztályozó olyan osztályozó, amely a bemenetére adott mintákat csak gyenge korrelációval képes osztályozni (de jobb eredménnyel, mintha csak véletlenszerűen „találgatna”), ezzel szemben az erős osztályozó a valóságnak jobban megfelelő értéket ad. A legtöbb boosting algoritmus esetében iteratívan tanítjuk a gyenge osztályozókat, majd amikor elérték egy bizonyos pontossági szintet a végleges erős osztályozóhoz adjuk. Ezután a tanítómintákat újrásúlyozzuk, a hibákat büntetjük (növeljük a súlyt), a helyes osztályozásokat jutalmazzuk (csökkentjük a súlyt). Így az új gyenge osztályozók azokra a tanítómintákra fognak összpontosítani, amelyeket az előzők hibásan osztályoztak [8].

2.2.6 Meanshift algoritmus

A meanshift algoritmust egy sűrűségfüggvény maximumainak megkeresésére használjuk, ha a függvényt nem ismerjük, de diszkrét felvett értékek rendelkezésünkre állnak. Ez a módszer hasznos ilyen sűrűségfüggvények móduszának megkeresésére. Adott pontokra a módusz a következőképpen számítható ki:

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

ahol az x_i a sűrűségfüggvény ismert pontja, $K(x)$ az eloszlásfüggvény, $N(x)$ x olyan környezete, ahol $K(x) \neq 0$. Az $m(x)$ -et egy adott x -re kiszámítjuk, majd visszahelyettesítjük az eredeti képletbe, és addig ismétljük, míg megfelelően pontos eredményt nem kapunk [9].

2.2.7 k legközelebbi szomszéd (kNN) algoritmus [10]

Legyen S n pont halmaza a síkon, amelyet felosztunk egy R piros pontokból, és egy B kék pontokból álló halmazra. A legközelebbi szomszéd döntési szabály egy új q pont színét úgy határozza meg, hogy a hozzá legközelebb eső S -beli pont színét kapja. Ezt a módszert gyakran alkalmazzák a mintafelismerésben, mint a példaalapú tanítási módszerek egyike. Ezért az S -t gyakran szokás tanítóhalmaznak nevezni. Ez az egyik legegyszerűbb tanuló algoritmus.

A legközelebbi szomszéd döntési szabály közvetett módon osztja fel a síkot egy piros, és egy kék halmazra, amelyek a piros-kék döntési határnál találkoznak. Gyakran lehetőség van az S halmaz csökkentésére a döntési határ változtatása nélkül.

2.2.7.1 Egydimenziós kNN algoritmus

Egydimenzióban az S halmaz n darab valós számból áll. Rendezzük S -t úgy, hogy $S = \{s_1 \dots s_n\}$ ahol $s_i < s_{i+1}$ minden $1 \leq i \leq n$ -re. A döntési határ minden (s_i, s_{i+1}) rendezett párból áll, ahol s_i piros, s_{i+1} pedig kék, és fordítva. Így a probléma véges időn belül megoldható, ha S rendezett. A döntési határt úgy keressük meg, hogy vesszük S mediánját ($m = s_{\lfloor n/2 \rfloor}$), felosztjuk S -t $S_1 = \{s_1 \dots s_{\lfloor n/2 \rfloor - 1}\}$ és $S_2 = \{s_{\lfloor n/2 \rfloor + 1} \dots s_n\}$ partíciókra, majd megvizsgáljuk, hogy $(s_{\lfloor n/2 \rfloor - 1}, m)$, és/vagy $(m, s_{\lfloor n/2 \rfloor + 1})$ részei-e a döntési határnak, majd rekurzívan ismétéljük ezt S_1 -re, és S_2 -re.

2.2.7.2 Kétdimenziós kNN algoritmus

Kétdimenzióban S elemei Voronoi cellák. A célunk minden olyan Voronoi élt megtalálni, amely két olyan Voronoi cellát köt össze, melyeknek definiáló pontjai eltérő színűek. Feltételezzük, hogy S általános helyzetű, azaz a pontok eloszlása véletlenszerű, és nem eshet négy tetszőleges pont egy szabályos körre.

Az algoritmust a Delaunay-háromszögelés fogalmaival mutatjuk be. S -ben egy Delaunay-háromszög egy olyan háromszög, melynek csúcsai (v_1, v_2, v_3) S elemei, valamint a v_1, v_2, v_3 pontokat érintő körben nincs S eleme. A Delaunay-háromszögelés S -t Delaunay háromszögekre particionálja. A Delaunay-él olyan szakasz, amelynek csúcspontjai olyan körre illeszkednek, melyben nincsenek S elemei. Ha S általános helyzetű, pontosan egy Delaunay-háromszögelése létezik, és tartalmaz minden olyan háromszöget, melynek élei Delaunay-élek. A Delaunay-háromszögelés a Voronoi diagram duálisa, mert S két pontját csakis akkor kötheti össze Delaunay-él ha szomszédos Voronoi cellában vannak.

Bikromatikusnak hívunk egy Delaunay-háromszöget, ha legalább egy piros, és egy kék csúcsa van. Így a probléma a bikromatikus Delaunay-élek megkeresésére vezethető vissza.

2.2.8 Dynamic time warping [11]

A dynamic time warping (DTW) egy jól ismert technika arra, hogy optimális illeszkedést találjunk két adott időben változó szekvenciára bizonyos feltételek mellett (5. ábra).

Eredetileg a DTW-t automatikus beszédfelismerésre fejlesztették ki. Az adatbányászat, és az információkinyerés terén szintén sikerrel alkalmazzák.

A DTW célja, hogy összehasonlítsa két időben változó szekvenciát. Legyen

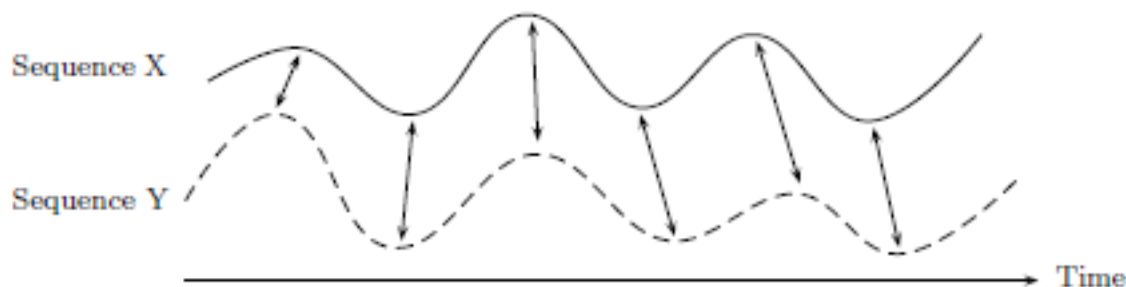
$X := (x_1, x_2, \dots, x_N)$, melynek hossza $N \in \mathbb{N}$, és $Y := (y_1, y_2, \dots, y_M)$, melynek hossza

$M \in \mathbb{N}$. Ezek a szekvenciák lehetnek diszkrét értékek, vagy valamilyen folyamatos jelből egyenlő időközönként vett minták. Ha rögzítünk egy F jellemzőteret, akkor $x_n, y_m \in F$ minden $n \in [1, N]$ és $m \in [1, M]$ -re. $x, y \in F$ Jellemzőinek összehasonlítására tudnunk kell mérni valamilyen lokális távolságot, amelyet a következőképpen definiálunk:

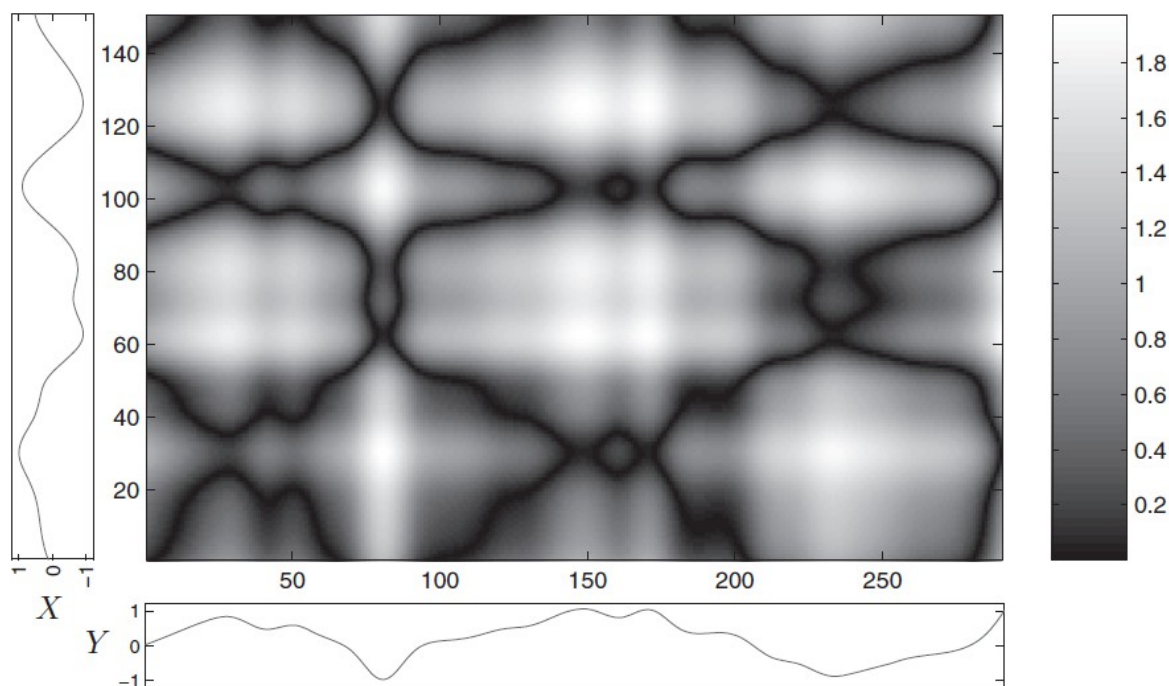
$$c: F \times F \rightarrow \mathbb{R}_{[0, \infty]}.$$

Általában $c(x, y)$ kicsi, ha x és y hasonlóak, egyébként nagy. Minden (x_n, y_m) párra (ahol $n = 1 \dots N$ és $m = 1 \dots M$) kiszámítjuk a távolságot, így megkapjuk a

$C \in \mathbb{R}^{N \times M}$ távolságmátrixot, ahol $C(n, m) := c(x_n, y_m)$ (6. ábra). A célunk, hogy X és Y között megtaláljuk a legkisebb össztávolságú illeszkedést.



5. ábra: Két szekvencia illesztése. A nyilak az illesztett pontokat jelölik [11].



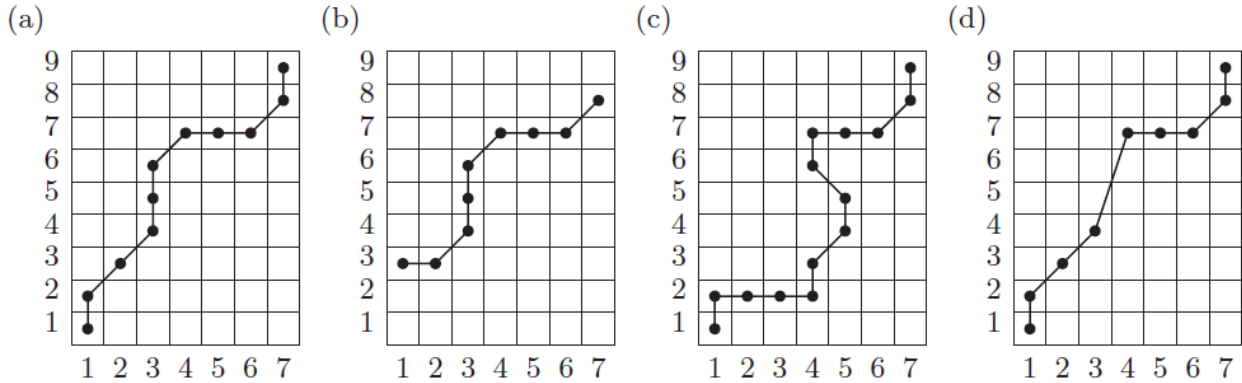
6. ábra: Két valós szekvencia közötti távolsági mátrix. A vízszintes tengelyen az X , a függőleges tengelyen az Y szekvenciát ábrázoltuk, kis értékek sötét, nagy értékek világos szürkeárnyalattal vannak jelölve [11].

Definíció: Az (N, M) illeszkedési útvonal egy $p = (p_1, \dots, p_L)$ szekvencia, ahol $p_l = (n_l, m_l) \in [1, N] \times [1, M]$ $l \in [1, L]$ -re, amely a következő feltételeknek felel meg:

- i. Határfeltétel: $p_1 = (1, 1)$, és $p_L = (N, M)$.
- ii. Monotonitási feltétel: $n_1 \leq n_2 \leq \dots \leq n_L$, és $m_1 \leq m_2 \leq \dots \leq m_L$.
- iii. Lépésméret feltétel: $p_{l+1} - p_l \in \{(1, 0), (0, 1), (1, 1)\}$ $l \in [1, L-1]$ -re.

Jegyezzük meg, hogy a lépésméret feltétel (iii.) maga után vonja a monotonitási feltételt (ii.), amelyet a jobb érthetőség érdekében külön említettünk. A $p = (p_1, \dots, p_L)$ illeszkedési útvonal az $X = (x_1, x_2, \dots, x_N)$, és az $Y = (y_1, y_2, \dots, y_M)$ szekvencia közötti lehetséges illeszkedést ad meg, úgy, hogy az x_{n_l} elemet X -hez, az y_{m_l} elemet Y -hoz rendeli.

A határfeltétel kikényszeríti, hogy X és Y első és utolsó elemei illeszkedjenek egymásra, vagyis az illeszkedés a teljes szekvenciára vonatkozik. A monotonitási feltétel tükrözi a pontos időzítés szükségességét. Ha egy X -beli elemre van rákövetkező elem, akkor Y megfelelő elemének is meg kell lennie, és fordítva. Végül a lépésméret feltétel egyfajta folyamatosságot fejez ki. X -ből, és Y -ből nem hagyható el elem, és nincs bennük ismétlődés (a 7. ábra szemlélteti a feltételeket).



7. ábra: Két szekvencia közötti lehetséges útvonalak szemléltetése. (a): megfelelő illeszkedési útvonal, mindhárom feltételt kielégíti. (b): a határfeltételt (i.) nem elégíti ki (c): a monotonitási feltételt (ii.) nem elégíti ki (d): a lépésméret feltételt (iii.) nem elégíti ki [11].

A p illeszkedési útvonal $c_p(X, Y)$ teljes távolsága X és Y között a lokális távolságokat figyelembe véve a következőképpen definiálható:

$$c_p(X, Y) := \sum_{l=1}^L c(x_{n_l}, y_{m_l}) \quad ,$$

továbbá a $\hat{p}$ optimális illeszkedési útvonal X és Y között az az illeszkedési útvonal, amely az összes lehetséges illeszkedési útvonal közül a legkisebb távolságokat tartalmazza. Az X és Y közötti DTW távolság ($DTW(X, Y)$) $\hat{p}$ teljes távolsága:

$$DTW(X, Y) := c_{\hat{p}}(X, Y) = \min \{c_p(X, Y)\}$$

ahol p egy (N, M) illeszkedési útvonal.

2.2.9 Spline interpoláció

Felmerült egy ötlet, hogy a szoftver a felhasználó által levegőben leírt görbék bevitelére legyen alkalmas. Ehhez bizonyos időközönként mintavételezzük a felhasználó keze koordinátáit, majd a keletkezett pontokra spline interpolációval görbét illesztünk.

A spline interpoláció az $f(x)$ függvény ismert x_i (ahol $i=0 \dots n$) pontjaira ún. spline görbét illeszt, amely általában a következő feltételeknek felel meg:

pl. harmadfokú spline-ra $S_3(\Delta_n, x)$ ahol Δ_n olyan partíció, hogy $a = x_0 \leq x_1 \leq \dots \leq x_n = b$ ahol $[a, b]$ szakaszonként harmadfokú polinomokból áll, második deriváltja folyamatos, továbbá $S_3(\Delta_n, x_i) = f(x_i)$, és a következő feltétel mindkét végpontban: (pl.

$S_3'(\Delta_n, a) = y_0'$ és $S_3'(\Delta_n, b) = y_n'$ vagy $S_3''(\Delta_n, a) = y_0''$ és $S_3''(\Delta_n, b) = y_n''$) Ha az $f(x_i)$ értékek az $[a, b]$ intervallumon periodikusak, az illesztendő spline görbének

is periodikusnak kell lennie. Polinomiális spline görbék esetén az illesztett görbe fokszáma $2k+1$, a két végpont feltételen túl további k darab feltételnek kell teljesülnie. $2k$ -adfokú interpoláció esetén a spline csomópontjait (a $2k$ -adik derivált nem értelmezett) az x_i pontok közötti távolság felénél vesszük fel, a -ra, és b -re további k darab feltételt rendelünk [12].

2.3 Beépülők fejlesztése Blenderhez

A Blender funkcióinak kibővítésére Python szkripteket használhatunk, mint ahogy sok hasznos eszközt fejlesztettek már így a programhoz. Sajnos a Blendernek nincs beépített fejlesztőkörnyezete, csak egy szintaxiskiemelő szövegszerkesztőt, ill. a beépített Python konzolt tudjuk használni erre a célra. Ez elég ahhoz, hogy egyszerű szkripteket írjunk, de nagyobb, komolyabb projektekhez már kevés. Különösen problémás a hibakereső hiánya, ezért külső eszközök használata szükséges.

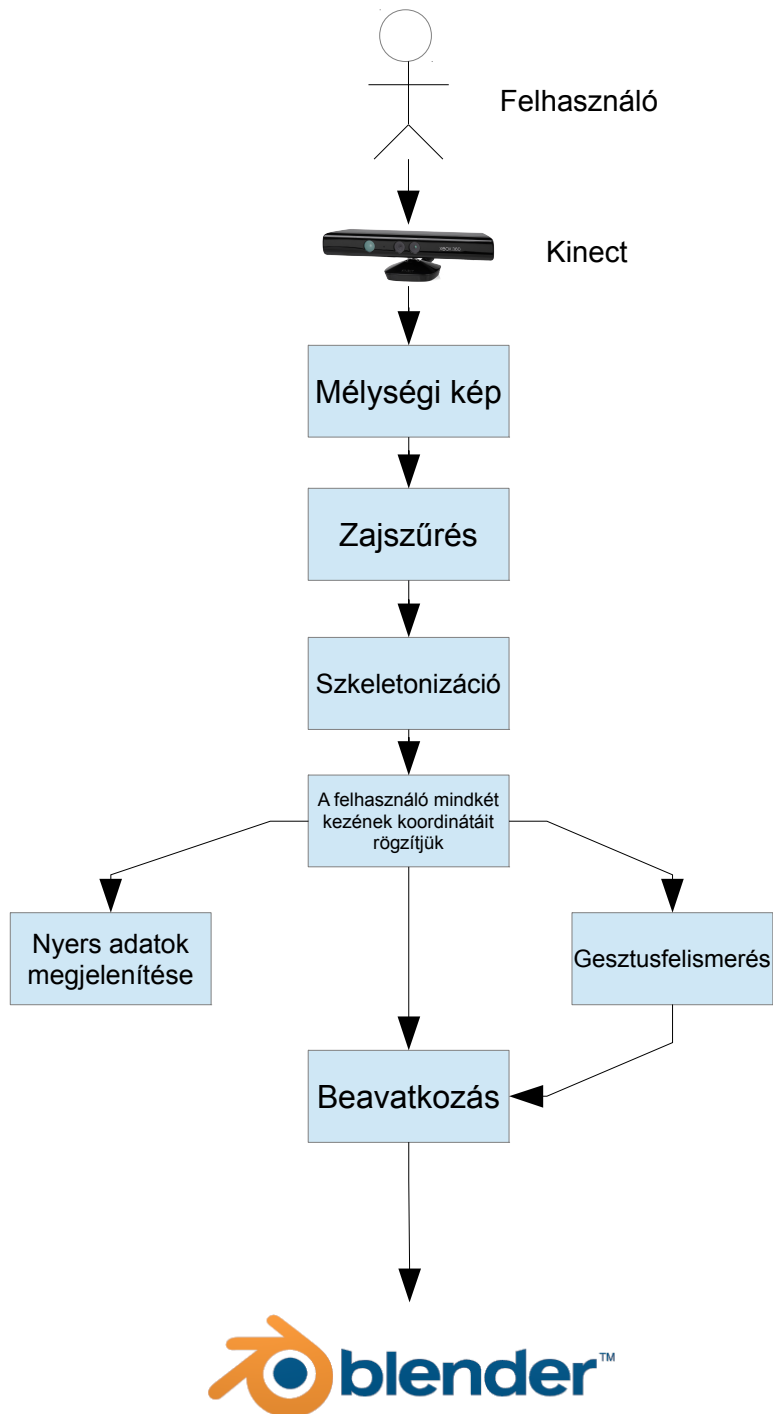
2009-ben a Blender fejlesztői úgy döntöttek, hogy az új alapoktól újraírt 2.5-ös verzió teljesen új Python API-t kap. Sőt, az új verzióba a 3.x-es Pythont integrálták, azaz szakítottak az előző verziókkal való visszafele kompatibilitással, amelyek a 2.x-es Python verziót használták. Ez főleg a GUI fejlesztésnél okozhat problémát, mert néhány eszköz nem támogatja a Python 3.x-et [13].

3 A rendszer bemutatása

3.1 Cél

A rendszer célja, hogy megkönnyítse a 3D modellezés folyamatát azáltal, hogy a felhasználó kézmozdulatait a modelltérben leképezve 3D objektumok manipulálást teszi lehetővé.

3.2 Rendszervázlat



3.3 A rendszer felépítése

3.3.1 Mélységi kép

A Kinect szenzor egy látszólag véletlenszerű, változó intenzitású pontokból álló hálót vetít ki, majd ennek torzulásaiból következtet az előtte elhelyezkedő objektumok távolságára, minden pixelhez egy 11 bites távolsági értéket rendel. Bár közvetlenül nem dolgozzuk fel, de alapja a rendszer működésének.

3.3.2 Szkeletonizáció

A szkeletonizáció segítségével egyszerűen tudjuk figyelni az egyes testrészek mozgását. Bár kevesebb információt hordoz, mint a mélységi kamera képe, az adatok gyorsabban, és egyszerűbb algoritmusokkal feldolgozhatók, és a Kinect SDK, és az OpenNI függvénykönyvtár beépített megvalósítást tartalmaz, amely valamilyen alakfelismerés alapján próbálja meghatározni az ember helyzetét a mélységi képből. A beépített algoritmus képes álló, vagy ülő felhasználó detektálására.

3.3.3 Zajszűrés

Mivel a Kinect nyersen elég zajos képet szolgáltat, mielőtt feldolgozzuk szűrni kell. Erre a beépített zajszűrő algoritmust alkalmazzuk, amely képes a kamera által észlelt zajok kiszűrésére, valamint a követés elvesztése esetén interpolációra. A zajszűrő paramétereinek beállítása függ a szoba méretétől, a megvilágítástól, de pontos beállítása *„inkább művészet mint tudomány”* [14], ezért a paraméterek egy konfigurációs állományból adhatóak meg.

A szkeleton lekövetés által közölt adatokban előforduló zajok szűrhetők, hogy az időbeli akadozást minimalizáljuk, és az ízületi pontok helyzetét időben stabilizáljuk (ezzel nem csak a zajokat, de a kézremegést is képesek vagyunk kompenzálni).

A Kinect SDK az ilyen zajok kiszűrésére beépített megvalósítást tartalmaz. Ez a megoldás a gazdasági adatok elemzésére is használt Holt-féle exponenciális simítást alkalmazza, mert más simító eljárásokkal ellentétben kisebb a késleltetése. A szűrő öt paraméterrel állítható:

- `Smoothing`
Ennek a paraméternek a növelése simítja az szkeleton pontjainak mozgását, de azzal fordítottan arányosan csökkenti az érzékenységet.
- `Correction`
Ha alacsony értéket adunk meg, lassabban korrigál, de egyenletesebb lesz a mozgás, ha nagy értéket adunk meg gyorsabban korrigál.
- `Prediction`
A pontok jövőbeli helyzetét próbálja megbecsülni arra az esetre, ha valamelyik pont elvész. Ha nagy értéket állítunk be gyors mozgásoknál nagy lehet a túllövések száma.
- `JitterRadius`
Egy gömb sugarát adja meg, amelyen belül bármely pont mozgását a rendszer

remegésnek számítja, és kiszűri.

- `MaxDeviationRadius`

Annak a gömbnek a sugarát adja meg, amelyben a szkeleton pontjai elhelyezkedhetnek, a többit kiszűri [15].

A későbbiekben több környezetből való tesztelésből előre beállított paraméterek közül lehet majd váltogatni, (valamint felmerült az ötlet, hogy a Kinect RGB kamerájának hisztogramja alapján egy tanuló algoritmus segítségével az előre eltárolt paraméterek közül automatikusan választjuk ki a legmegfelelőbbet), így a felhasználónak nem kell foglalkoznia a „belövással”.

3.3.4 Gesztusfelismerés

Ha azt szeretnénk elérni, hogy a rendszer a felhasználó által megadott parancsokat végezze el, neki különböző gesztusokkal kell jeleznie szándékát. Elindítás után a rendszer egy alapállapotba kerül, amelyben figyeli a felhasználó kezének mozgását, de az alkalmazás állapotterébe nem avatkozik be. Ha valamilyen gesztust észlel, elkezd a felhasználó szándékának megfelelően, kezének mozgásával arányosan módosítani az alkalmazás állapotterét, majd ha a kéz „megáll” (legalább egy másodpercig koordinátái 0,1 egységnél kevesebbet ingadoznak) visszaáll alapállapotba.

Egyszerű gesztusokat (pl. karlendítés valamelyik irányba) könnyen le tudunk kezelni, bonyolultabbakhoz valamilyen tanuló algoritmust kell használni. A felhasználó keze helyzetének megállapítására a szkeleton által jelzett koordináták egy környezetében (a környezet sugarát Z koordinátától függően kell megválasztani, mert minél messzebb van a kamerától, annál kisebbnek látszik, és fordítva) a mélységi képet vizsgáljuk, a detektálásra tanuló algoritmusokat használunk. A rendszer tanítása során képkockánként (másodpercenként 60-szor) rögzítjük a felhasználó által elvégzett mozdulatot. Gesztusfelismerés közben a DTW algoritmust alkalmazzuk a felhasználó mozdulatának időbeli változásait figyelve. Ez egy adott időpillanatra képes meghatározni, az éppen végzett mozdulat melyik felvett gesztusra hasonlít leginkább. Mivel ekkor még nem tudjuk meghatározni, milyen módon kell beavatkozni, csak egy számértéket kapunk, a különböző időpillanatokban kiszámított DTW távolságokból egy kNN osztályozó dönti el a végzett gesztus típusát, egy esemény kiváltásával jelzi a detektálás sikerességét, az eseménykezelő pedig megkapja a detektált gesztus azonosítóját.

3.3.5 Nyers adatok megjelenítése

Erre hibakeresési okokból került sor. A rendszer kiírja a felhasználó kezeinek koordinátáit, az épp aktuális működési módot, és a felismert gesztus azonosítóját.

3.3.6 Beavatkozás

A felismert gesztus alapján a modellező szoftver megfelelő funkciójának meghívása

3.4 Az elkészült példaprogram

A megvalósított algoritmusok demonstrálására elkészítettem az XNA egyik példaprogramjának módosított változatát, valamint a rendszer tanítására elkészült egy külön alkalmazás. A programmal egy 3D objektumot lehet térben, gesztusfelismerés segítségével mozgatni.

Ha a felhasználó a jobb kezét vízszintes irányba mozdítja a rendszer mozgató módba vált, majd a jobb kéz mozgási iránya, és sebessége alapján mozdítja el a képernyőn megjelenő objektumot, majd ha legalább egy másodpercig a kéz „áll” visszál alapállapotba.

Ha a felhasználó kört ír le a jobb kezével a forgató mód aktiválódik, majd a rendszer a kéz mozgási iránya és sebessége alapján fordítja el a képernyőn megjelenő objektumot.

A rendszer tanítása egy külső alkalmazás segítségével történik, úgy, hogy ha a felhasználó a rögzítés gombra kattint egy pár másodperces késleltetés után elkezd képkockáként rögzíteni a szkeleton pontjainak koordinátáit, majd ha bármely pont esetén 0,1 egységnél kevesebb változást észlel leáll a rögzítéssel.



8. ábra: Az elkészült példaprogram. Középen látható a mozgatható objektum, a bal felső sarokban a hibakeresés érdekében megjelenített információk.

3.5 Továbbfejlesztési lehetőségek

3.5.1 Megvalósítás egyéb eszközzel

Tanulmányoztam a megvalósítás lehetőségeit Time-of-Flight kamerával, és sztereó kamerával, az általánosabb felhasználhatóság érdekében, de a félév során a megvalósításra nem jutott idő, ezért a Kinect szenzort választottam.

3.5.2 Integrálás a Blenderbe, vagy más modellező szoftverekbe

A Blender programozható a beépített Python API-ján keresztül, ez azonban nem alkalmas interaktív, a felhasználótól érkező adatok feldolgozására. Erre más eszközt kell használni, valamint felmerült a Blender forráskódjába való implementálás lehetősége is.

3.5.3 Zajszűrő paraméterek automatikus beállítása

A rendszer használhatóságának növelése érdekében különböző fényviszonyok mellett optimálisan beállított zajszűrő paramétereket rögzítünk, majd a Kinect RGB kamerájának hisztogramjából egy tanuló algoritmus fogja megállapítani melyik előre beállított paraméterhalmazt kell alkalmazni.

3.5.4 Dinamikus gesztusok kezelése

Dinamikus gesztusok kezelése még nem megoldott, ez fontos lenne pl. nagyításnál, kicsinyítésnél, kamera mozgatásánál. Erre fokozott osztályozók kerülnek alkalmazásra.

3.5.5 Görbe bevitele dinamikus gesztussal

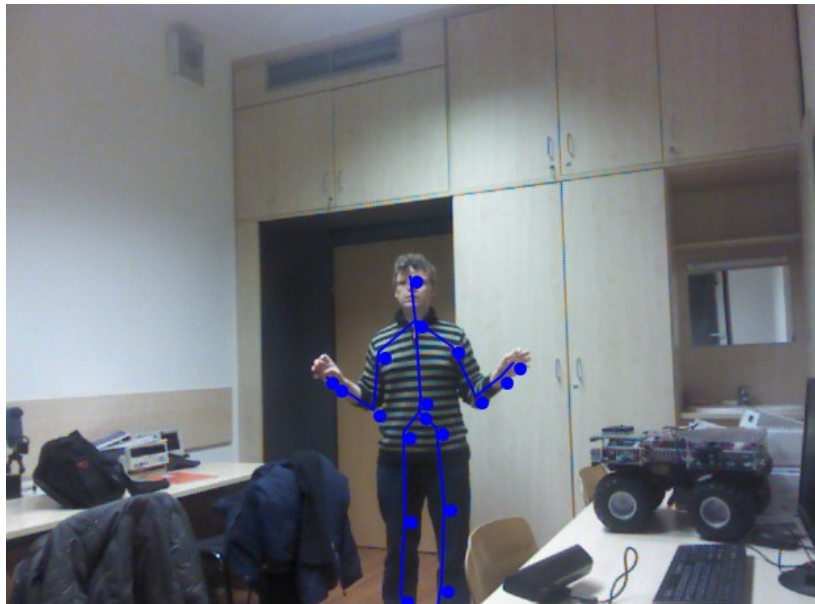
Felmerült egy ötlet görbék felvitelére gesztusfelismerés segítségével. Az eljárás a felhasználó kezének koordinátaiból bizonyos időközönként mintát vesz, majd spline interpolációval ún. spline görbét illeszt a pontokra. A választás azért került a spline interpolációra, mert a spline görbék gyakoriak a 3D modellezésben (egyéb eljárások esetén spline vagy Bézier görbére való konvertálásra lenne szükség).

A Kinect-től kapott adatok feldolgozására alkalmazható a Blender Python API-ja.

3.5.6 A gesztusfelismerő algoritmus robusztusságának növelése

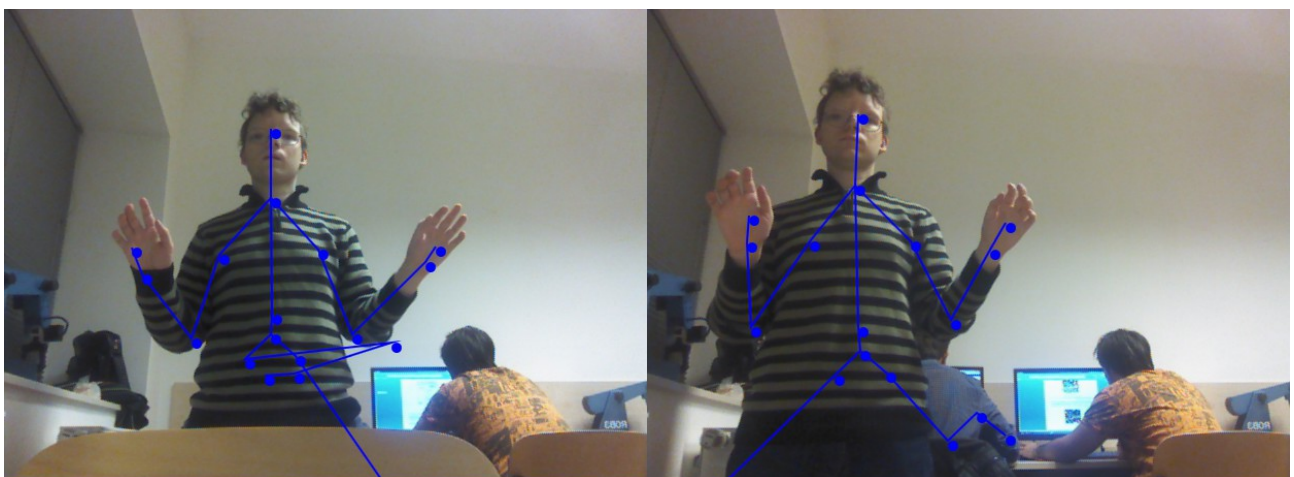
Tesztelés során nyilvánvalóvá vált, hogy a gesztusfelismerő algoritmus erősen függ a környezettől, és a felhasználótól. A lehetőségeket vizsgálom a kNN lecserélésére, valamilyen erősebb algoritmusra, illetve fokozott osztályozókra (boosting).

4 Eredmények



9. ábra: Ha a teljes test látszik, és a Kinect látómezőjében nincs akadály, a rendszer megfelelően működik

A Blender egy bárki számára ingyen elérhető, és bármilyen célra felhasználható profi modellező képességekkel rendelkező programcsomag, ám a sok billentyűkombináció (amely egyben erőssége is, mert nagyon hatékonyan lehet vele dolgozni) sok kezdőt elriaszt a kipróbálástól. A projektemmel arra próbáltam megoldást keresni, hogy egy felhasználóbarátabb, természetesebb kezelőfelületet alkossak, a hatékonyság elvesztése nélkül.



10. ábra: A detektálás hibái

A gesztusfelismerő algoritmus erősen függ a környezettől, a felhasználó ruházatától, a felhasználótól. Tesztelésre került különböző környezetekben, természetes-, illetve lámpafénynél, ezen a téren a gesztusfelismerő algoritmus még pontosításra szorul. Arra jutottam, hogy ha új környezetben telepítjük a rendszert új tanítómintákat is kell rögzíteni, hogy a megbízható működést biztosíthassuk. A rendszer pontatlansága nagyobb, ha nem

„látja” a felhasználó egész testét, a szenzor túl közel van a felhasználóhoz, vagy a felhasználón kívül más személyek, vagy mozgó objektumok vannak a Kinect látómezőjében (10. ábra). Amennyiben tudjuk biztosítani, hogy a Kinect szenzor legalább 1,5-2 méterre legyen a felhasználótól a rendszer kb. 90%-ban helyesen becsli a felhasználó által végzett gesztusokat. Ha ez nem lehetséges, a rendszert különféle megvilágítások melletti tanítással kell a környezetre szabni.

Irodalomjegyzék

- 1: Matthias Rehm, Nikolaus Bee, Elisabeth André: Wave Like an Egyptian – Accelerometer Based Gesture Recognition for Culture Specific Interactions, University of Augsburg, 2007, http://mm-werkstatt.informatik.uni-augsburg.de/files/publications/199/wave_like_an_egyptian_final.pdf
- 2: Andrew Davison: Java Programming Techniques for Games, 2011, Prince of Songkla University, <http://fivedots.coe.psu.ac.th/~ad/jg/nui13/KinectImaging.pdf>
- 3: Stephanie Crawford: How Microsoft Kinect Works, 2010, HowStuffWorks, <http://electronics.howstuffworks.com/microsoft-kinect.htm>
- 4: Mark Wilson, Matt Buchanan: Testing Project Natal: We Touched the Intangible, 2009, gizmodo.com, <http://gizmodo.com/5277954/testing-project-natal-we-touched-the-intangible>
- 5: Bernhard Büttgen, Thierry Oggier, Michael Lehmann, Rolf Kaufmann, Felix Lustenberger: CCD/CMOS Lock-In Pixel for Range Imaging: Challenges, Limitations and State-of-the-Art, Swiss Center for Electronics and Microtechnology, , http://www.mesa-imaging.ch/dlm.php?fname=pdf/RIM_Lock_In_Challenges_Limitations_5.pdf
- 6: Gary Bradsky, Adrian Kaehler: Learning OpenCV, O'Reilly, 2008, ISBN: 978-0-596-51613-0
- 7: M. Correa, J. Ruiz-del-Solar, R. Verschae, J. Lee-Ferng, N. Castillo: Real-Time Hand Gesture Recognition for Human RobotInteraction, Universidad de Chile, <http://bender.li2.uchile.cl/papers/GestureRecognition.pdf>
- 8: Robert E. Schapire: The Strength of Weak Learnability, Kluwer Academic Publishers, Boston, 1990, <http://www.cs.princeton.edu/~schapire/papers/strengthofweak.pdf>
- 9: Yizong Cheng: Mean Shift, Mode Seeking, and Clustering, IEEE, 1995, <http://users.isr.ist.utl.pt/~alex/Resources/meanshift.pdf>
- 10: David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, Godfried Toussaint: Output-Sensitive Algorithms for Computing Nearest-Neighbour Decision Boundaries, 2005, Springer Science+Business Media, Inc, http://download.springer.com/static/pdf/456/art%253A10.1007%252Fs00454-004-1152-0.pdf?auth66=1351942196_a7becbe10a302d78bfc106903af2c951&ext=.pdf
- 11: Muller M: Information Retrieval for Music and Motion, Springer, 2007, ISBN: 978-3-540-74047-6
- 12: Michiel Hazewinkel: Spline Interpolation, 2001, Springer, <http://www.encyclopediaofmath.org/index.php?title=p/s086820>
- 13: Witold Jaworski: Programming Add-Ons for Blender 2.5, 2011, ISBN: 978-83-931754-2-0

14: Jarrett Webb, James Ashley: Beginning Kinect Programming with the Microsoft Kinect SDK, Apress, 2012, ISBN: 978-1-4302-4104-1

15: Microsoft: Joint Filtering, 2012, MSDN, <http://msdn.microsoft.com/en-us/library/jj131024.aspx>