Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Optikai alapú Motion Capture rendszer fejlesztése

TDK Dolgozat

Készítette
Devecseri Viktor

Konzulensek
Dr. Umenhoffer Tamás
Dr. Tamás Péter

2011. október 28.

Tartalomjegyzék

Kivonat

Mozgás követő rendszerek használatával élőlények, tipikusan emberek mozgását rögzítik, amelyeket animált virtuális karakterekre lehet átültetni. Ezért a mozgás követő rendszereket főleg a film, és játék iparban alkalmazzák valósághű mozgás létrehozásához. Az animálás megvalósítható hagyományos technikával is, azonban hasonló minőségű mozgás létrehozása időigényes, és nagy szakértelmet igénylő feladat.

Többfajta mozgás követő rendszer létezik, a dolgozatban egy látható fény tartományban működő aktív optikai alapú mozgás követő rendszer fejlesztését mutatom be.

A szereplőkre, az ízületeik közelébe, homogén, világító markereket helyezünk, amelyeket a terem szélein elhelyezett 6 nagy sebességű kamera néz. Ezután a kamerák képén a markerek pozícióját kell meghatározni. A kamerák által másodpercenként 60-szor frissített kép nagy adatmennyiséget generál, a feldolgozásához elosztott, több számítógépen történő képfeldolgozás szükséges. A képfeldolgozás nagy része jól párhuzamosítható, ezért a számítógépben lévő videókártyát is használom hozzá.

A képfeldolgozás után a központi számítógép a megkapott 2 dimenziós marker pozíciók alapján határozza meg a markerek 3 dimenziós pozícióit. Animáció létrehozásához szükséges a markereket azonosítani, ezért a markerek 2 dimenziós, és 3 dimenziós pozícióit követni kell.

Az így megkapott pont felhőben a pontok között definiálható egy hierarchikus viszony, amelyből meghatározható a karakter csontvázában lévő csontok közötti hierarchikus transzformációs lánc.

A jó eredmények eléréséhez fontos a pontos kalibráció, amelynek során a kamerák pozícióját, és orientációját kell meghatározni. A jó kalibrálás a pontos 3 dimenziós pozíciókon kívül nagy szerepet játszik a pontok követésében is.

Abstract

The motion of humans, or other living beings can be recorded with motion capture systems. The recorded data can be used to animate a virtual character. Motion capture systems are mainly used in the movie, and video game industry. Similar quality character animation can be achieved by hand animation techniques, but this requires a lot of time, and competence.

There are many methods used by motion capture systems. In this paper I present the development of an active optical motion capture system.

The performer wears homogenous, illuminating markers attached to each relevant joint. The performer movement is captured by 6 high speed cameras in order to determine the markers' position in the cameras' image. The cameras operate at 60 frame per second, which generates a lot of data. The image processing is distributed among multiple computers. Luckily most of the image processing task is highly parallel, so the high parallel computational capacity of modern programmable GPUs can be exploited.

The main computer calculates the markers' 3D positions after it receives the 2D data from image processing. Each marker needs to be identified, so the markers' 2D and 3D positions are tracked.

A hierarchy can be defined between the points in the point cloud, which can be used to determine the bones hierarchical transformation chain in the character's skeleton.

To achieve satisfying results precise calibration is required. During calibration the cameras' position and orientation are determined. Good calibration is also crucial for accurate tracked point identification and labeling.

1. fejezet

Bevezetés

1.1. Célkitüzés

Számos tudományterület van ahol a mozgáskövető és mozgás rögzítő eszközök fontos szerepet játszanak. Ilyen például az automatizálás, robotika és számítógépes animáció is. A különböző alkalmazási területek más-más speciális követelményeket támaszthatnak, más igények lehetnek fontosak. Van ahol elegendő egy-két független pont követése, máshol pl. a karakter animációnál, a követett pontok komplex transzformációs hierarchiát alkotnak, ahol az egyes elemek teljes merev transzformációjának előállítására szükség van.

A mozgáskövető rendszerekre általánosan jellemző a bonyolultságuk. Általában speciális eszköz igényeik vannak (kamerák, speciális érzékelők, elosztott számításért felelős számítógép hálózat, stb.), és a követés helyszínével szemben is támaszthatnak komoly követelményeket (méret, megvilágítás, falak színe, helyiség bútorzata, stb.). Minél több pontot és minél bonyolultabb összefüggéseikben szeretnénk követni, annál nagyobbak ezek az igények és jelennek meg a korlátok.

Az automatizálás és robotika területén az igények igazán változóak lehetnek, általában az adott feladathoz speciális mozgáskövető rendszert építenek, ami a szükséges feltételeket a leghatékonyabban teljesíti mind költség, mind gyorsaság, mind minőség tekintetében. A számítógépes animáció területén viszont elég általánosan meghatározhatók a követelmények, általában egy vagy több emberi karakter fő ízületi pontjait kell követni, a csontok vagy testrészek hierarchikus transzformációs rendszerét számítani. Erre a feladatra több komplex mozgáskövető rendszer is megvásárolható, melyek a legapróbb részletig kidolgozottak, tartalmazzák a szükséges hardver és szoftver eszközöket illetve a terméktámogatást is. Ezek a rendszerek bonyolultságuk miatt rendkívül drágák.

A dolgozat célja egy olyan mozgáskövető és mozgásrögzítő rendszer kidolgozása, mely elég flexibilis ahhoz, hogy egyszerű eszközökön egyszerű feladatokra, és komplex berendezéssel bonyolultabb feladatokra is használható legyen. A rendszer optikai marker alapú megközelítést használ, ami az eszközigényt tekintve elég tág teret nyújt. Feladataim közé tartozott a mozgáskövető rendszer eszközeinek, és a helyszín igényeinek meghatározása és kielégítése, a mozgáskövetés algoritmusainak kiválasztása és hatékony implementálása, illetve a felhasználói kezelőprogram megtervezése és implementálása. Igyekeztem több

különböző alkalmazással demonstrálni a rendszerem használhatóságát és sokoldalúságát.

A fejezet következő alfejezeteiben a mozgáskövetés történelmi áttekintése és a fő mozgáskövető rendszer típusok ismertetése következik, majd ismertetem röviden a megoldott főbb problémákat.

1.2. Történelem

A következő rész röviden összefoglalja a motion capture történelmét [14] könyvből.

1872-ben Kalifornia egykori kormányzója, Leland Stanford, felbérelte Eadweard Muybridge-et, a híres tájkép fotóst, egy fogadás eldöntéséhez. A fogadás tárgya az volt, hogy egy vágtató lónak mind a 4 lába a levegőben van-e. Ezt 6 évvel később Muybridge bebizonyította úgy, hogy 24 db fényképezőgéppel egymás után lefényképezett egy vágtató lovat (1.1. ábra).

1.1. ábra. *Mahomet Running,* Eadweard Muybridge*, 1879*

1883-ban Etienne-Jules Marey készített egy kronofotográfiás kamerát, amelynek a zárszerkezete időzítve volt. Így egy mozgásból egymást követő képeket tudott rögzíteni. A mozgás rögzítéséhez ő egy kamerát használt, míg Muybridge többet.

1926-ban Harold Edgerton felfedezte, hogy egy működő motor forgó alkatrészét úgy tudja megfigyelni, mintha az ki lenne kapcsolva. Ehhez egy stroboszkóp villanását a motor forgásához igazította. 1931-ben kifejlesztette az elektromos stroboszkópot, amellyel gyorsan mozgó objektumokat tudott filmre rögzíteni.

1915-ben Max Fleischer lefilmezte testvérét David-et bohóc ruhába öltözve, és ebből

közel egy év alatt elkészítették az első animációs filmjüket *rotoszkóp* alkalmazásával. A rotoszkópot 1917-ben szabadalmaztatta. 1918-tól kezdve az „Out of the Inkwell" animációs sorozaton dolgozott, amelyben az animációs és élő felvételeket vegyítette.

1937-ben, majdnem 4 évnyi munka után, Walt Disney bemutatta az első egész estés animációs filmjét a Hófehérke és a hét törpét. Az animációs film elkészítéséhez rotoszkópot használtak, amely az élő felvétel egy kép kockáját kivetíti egy üvegre, és ezt rajzolja át egy animátor. Ezzel élethű mozgást lehet elérni.

Az 1970-es években kezdődtek meg a kutatások a digitális mozgás rögzítés területén. A CGI ipar hamar felfedezte a technológia lehetőségeit az 1980-as években. 1985-ben a Super Bowl közben sugározták a „Brilliance" című reklámot, amelyet Robert Abel reklám ügynöksége készített. A reklámban egy számítógépes női robot mozgott úgy, mint egy ember. A valósághű mozgáshoz egy modell 18 ízületi pontjára rajzoltak fekete pontokat, és a mozgást több kamera nézetből rögzítették. Ezután a film képeit Silicon Graphics számítógépekre importálták, és feldolgozták a robot mozgatásához szükséges animáció elkészítéséhez.

1988-as SIGGRAPH-on reklámozta a Silicon Graphics az új 4D sorozatú munkaállomását, a „Mike the talking head" produkcióval. Mike Normal fejét beszkennelték, majd a pontokból poligon modellt készítettek. Ezután Mike fejére érzékelőket rögzítettek, és az arc mozdulataival valós időben mozgatta a virtuális világban lévő arcot.

Az 1995-ben kiadott „FX Fighter" nevű játék volt az első valósidejű verekedős játék, amelyben 3D-s karakterek 3D-s környezetben mozogtak. Ezen kívül az első volt a játékok között, amelyben mozgás rögzítést használtak a karakterek mozgásához. A karakterek mozgását valós időben a felhasználó irányította, és ez alapján a rögzített mozgás darabok (futás, séta, ütés) lettek lejátszva úgy, hogy a felhasználó nem látta a mozgás darabok közötti átmenetet.

Az elkövetkező években mozgás rögzítést alkalmaztak az orvoslásban, katonaságnál, szórakoztatásban. Több sportágban is használják, amellyel a sportolók mozgását elemzik, és ez alapján javítják a teljesítményüket, vagy sérüléseket előznek meg.

1.3. Típusok

Ez az alfejezet összefoglalja a különböző motion capture rendszereket, és a piacon elérhető megoldásokat [14], [11], [7].

1.3.1. Optikai

Az optikai motion capture rendszerek több, tipikusan 4 – 32 kamerát alkalmaznak, amelyek a színtér körül vannak elhelyezve. A kamerák 30 és 2000 FPS között működnek. A szereplőkre markereket helyeznek, amelyek pozícióját a kamerák által látott képek alapján állapítják meg háromszögeléssel. A markerek típusa alapján több féle rendszer különböztethető meg.

Passzív markeres

A markerek fény visszaverő anyaggal vannak bevonva, amelyek vagy közvetlenül a színészre vannak felhelyezve, vagy tépőzárral egy motion capture ruhára.

A markerek a kamerákhoz közel helyezett fényforrások fényét verik vissza a kamerákba. Passzív markeres rendszerekben általában infra fénnyel világítják meg a markereket, és a kamerákon olyan szűrő van, amely az infra fényt engedi át. A megoldás előnye, hogy így a rendszer nem érzékeny a látható fénytartományra.

Ilyen rendszert gyárt a Vicon[1], a MotionAnalysis[2], a Qualisys[3], illetve a NaturalPoint[4].

A NaturalPoint OptiTrack nevű teljes test motion capture rendszere, 6 kamerával, szoftverrel, és kiegészítőkkel 6 718$-ba kerül. A kamerák látószöge 46 fok, felbontásuk 640×480, és 100 FPS-el működnek [4].

Aktív markeres

Aktív markeres esetben a markerek bocsátják ki a fényt, ehhez LED-eket alkalmaznak. A markerek folyamatosan világíthatnak, illetve villoghatnak, vagy pulzálhatnak is. A markerek gyors ki- és bekapcsolásának a markerek azonosításában van szerepe, amelyhez nagyobb képkocka sebességgel működő kamerákra van szükség. Ennek előnye a passzív markeres rendszerhez képest, hogy a markerek nem cserélődhetnek meg.

Aktív markeres rendszert gyárt a PhaseSpace[5]. A kamerák 3600×3600 pixel felbontásúak, és 480 Hz-en működnek. A markerek azonosításához minden LED saját frekvencián változik [5].

Marker nélküli

Az utóbbi években a gépi látás rohamos fejlődésének köszönhetően egyre nagyobb szerephez jutnak a tisztán képfeldolgozás alapú technológiák. A marker nélküli (*markerless*) rendszerekhez a szereplőknek nem kell markereket viselniük, a mozgás tipikusan több nézetből felvett videó folyamokból lesz visszaállítva speciális, gyakran tanításon alapuló algoritmusok alkalmazásával, melyek képesek felismerni az emberi alakokat. A bonyolultságuk miatt ezek a rendszerek általában nem valós időben működnek. Ilyen rendszert fejlesztenek például a Stanford Egyetemen[6].

Kereskedelemben is kapható rendszert kínál az iPi Soft[7], vagy az OrganicMotion[8]. Marker nélküli technológiát használ a Microsoft Kinect rendszere, mely egy hagyományos és egy mélység kamera segítségével valós idejű emberi mozgásdetektálást végez. Minősége azonban nem összemérhető a komplex rendszerekével.

[1] http://www.vicon.com/
[2] http://www.motionanalysis.com/
[3] http://www.qualisys.com/
[4] http://www.naturalpoint.com/
[5] http://www.phasespace.com/index.html
[6] http://www.stanford.edu/group/biomotion/
[7] http://www.ipisoft.com/
[8] http://organicmotion.com/

1.3.2. Mechanikus

A mechanikus rendszereknél a testre egy fém vázat, külső csontvázat, csatolnak, amely az ízületi pontokban lévő potenciométerekkel méri az elfordulásokat. Ez alapján szoftveresen rekonstruálható a test helyzete. Az optikai rendszerekhez képest előnye, hogy így mindig minden adat megvan, nem lehet markereket kitakarni. Hátránya, hogy a szereplő pozícióját, például ugrásnál, pontatlanul határozza meg. Ehhez más megoldást is kell alkalmazni.

Az Animazoo[9] Gypsy 7 rendszere így működik, és 8 000$-ba kerül.

A Measurand ShapeWrap III[10] rendszere hasonló elven működik, az elfordulásokat a külső csontvázban lévő szál optikákban a fény interferenciája alapján határozzák meg.

1.3.3. Mágneses

A szereplőkre 12 – 20 db mágneses érzékelőt helyeznek, amelyek egy külső adó által generált alacsony frekvenciás mágneses térhez képest mérik a pozíciójukat, és irányukat. Az irány meghatározásához 3 egymásra merőleges tekercset alkalmaznak.

Előnye, hogy nem lehet kitakarni a szenzorokat, mint az optikai esetben, azonban érzékenyek a mágneses, és elektromos interferenciára.

Ilyen megoldásra épülő rendszer az Ascension Technology[11] által készített MotionStar.

1.3.4. Inercia rendszeres

A szereplők helyzetét a testre elhelyezett gyorsulás mérőkkel és giroszkópokkal mérik. A módszer előnye, hogy nem igényel kamerákat, nem korlátozza a mozgásteret és könnyen hordozható. Hátrány azonban, hogy az érzékelők csupán a paraméterek változásait továbbítják, az abszolút értéket idő szerinti integrálással kaphatjuk meg. Ez az integrálás folyamat azonban akkumulálódó hibát okoz a rendszerben, melyet más mozgáskövető módszerekkel korrigálni kell.

Az XSens[12] MVN rendszere ezen a módszeren alapul. Az átlagos felhasználóknak is elérhető eszközök közül ide sorolhatjuk a Wii kontrollert is, mely egy ízület, a kéz helyzetét és pozícióját követi, mely szintén aktív markeres módszerrel egészül ki.

1.4. Áttekintés

A dolgozatban bemutatom egy aktív markeres optikai motion capture rendszer fejlesztését, és felépítését. A markerek állandóan világítanak, nincs szerepük a marker azonosításban. A rendszer fejlesztése során több problémát is meg kellett oldanom:

- A rendszer tervezése során fontos volt a moduláris felépítés, hogy a jövőben könnyen bővíthető legyen. A komponensek közötti lazán csatoltság eredménye a képfeldolgozás elosztott megvalósítása. Továbbá fontos volt az objektum orientált felépítés, és a

[9] http://www.animazoo.com/
[10] http://www.motion-capture-system.com/
[11] http://www.ascension-tech.com/
[12] http://www.xsens.com/

tervezési minták alkalmazása, úgy hogy a teljesítmény is megfelelő legyen. A rendszer felépítését a 2. fejezetben ismertetem.

- Első lépésként a kamerák képein meg kell határozni a markerek pozícióit. Magas másodpercenkénti képkocka sebességű kamerák esetén erre nincs sok idő. Például 120 FPS esetén mindössze ~ 8 ms telik el két képkocka érkezése között. A képfeldolgozás ideje a GPU-k alkalmazásával csökkenthető, így elérhetővé válik a 120 FPS valós idejű feldolgozása. Elosztott képfeldolgozás esetén fontos, hogy a képkockák exponálásának ideje ugyanabban az idő keretben legyen ismert. A képfeldolgozást a 3. fejezetben mutatom be.

- Miután ismertek a kamerák képein a markerek pozíciói, háromszögeléssel meghatározható a marker 3D-s pozíciója. Ehhez először a különböző kamerák képein lévő markereket egymáshoz kell rendelni. A pontok 2D-s, és 3D-s követésével az időben azonosíthatóvá vállnak a markerek, és egyszerűsítik a marker vetületek egymáshoz rendelését. Ezzel a 4. fejezetben foglalkozok.

- A markerek 3D-s pozícióinak meghatározásához fontos pontosan ismerni a kamerák helyzetét, illetve a belső felépítésüket leíró paramétereket. Ilyen például a kamera torzítás, amelyet szintén korrigálni kell. A kalibráláshoz több módszert kipróbáltam, amire megtaláltam azt, amely jó eredményeket ad. Ezeket a 5. fejezetben részletezem.

- Egy merev testre helyezve a markereket, a markerek térbeli pozíciójából meghatározható a merev test pozíciója, és iránya. Továbbá, ha a markerek egy emberre vannak helyezve, csontok hierarchikus rendszere mozgatható ezek alapján. Így a rendszer használatával számítógépes animációhoz is lehet mozgást rögzíteni. Ezt a 6. fejezetben ismertetem.

- A megvalósított funkciók eléréshez egy könnyen kezelhető felhasználói felület szükséges. Technikai kihívást jelentett, hogy a rendszert natív C++-ban fejlesztettem, de a felhasználói programot .NET-ben írtam. A programot a 7. fejezeben mutatom be.

- Végül a 8. fejezetben a mozgás rögzítésen kívül bemutatok más megvalósított felhasználási lehetőséget is. A rendszeren még lehet javítani, a tovább fejlesztési lehetőségeket is itt ismertetem.

2. fejezet

Tervezés

2.1. Követelmények

A rendszerrel szemben a következő követelményeket fogalmaztam meg:

Kamerák: Több különböző típusú kamera kezelése, azaz működjön a rendszer web kamerákkal is, illetve nagyobb tudású ipari kamerákkal is.

Sebesség: Képes legyen a 120 FPS-el működő Basler ipari kamerák képeinek valós idejű feldolgozására.

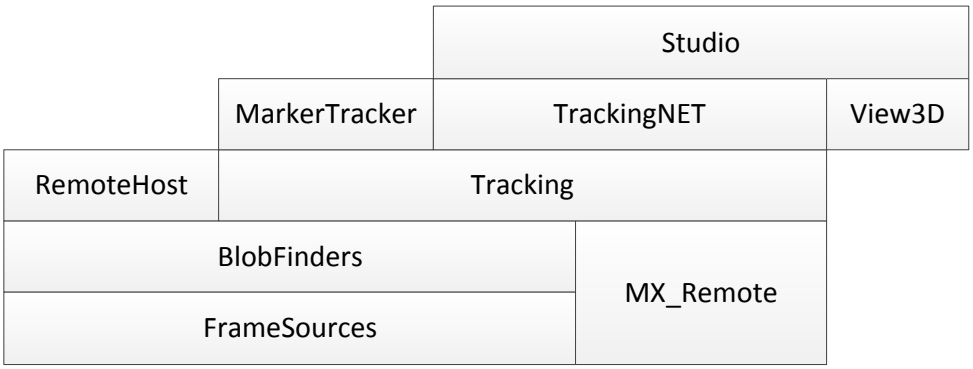
Skálázhatóság: Tudjon működni a rendszer egy számítógépen például néhány web kamerával, illetve sok kamera esetén elosztottan is, úgy, hogy ez a felhasználó felé átlátszó módon történjen.

Moduláris felépítés: Komponensek közötti lazán csatolás, amelyek egymást előre definiált interfészeken keresztül érik el. A kamerákat, és folt keresőket kezelő osztályok külön DLL-ekbe kerülnek, amelyeket a rendszer futás időben tölt be. Ennek előnye, hogy például hiányzó függőségek esetén is elindul a rendszer, maximum egy kamera típus nem lesz elérhető.

Hatékony előnézeti kép: A kamerák képeinek megjelenítése OpenGL-el történik, így a GPU-s képfeldolgozás során a kép gyorsan elérhető. Hálózati feldolgozás esetén a megjelenítés előtt a képet a GPU-ra fel kell tölteni.

Minimalista design, feladatok szétválasztása: Minden osztály csak egy feladatot lát el, és azt is a lehető legegyszerűbben. Azaz például egy kamera kezelő objektum nem tudja, hogy melyik szálon fut. Annyit tud, hogy blokkol, amíg egy új frame nem érkezik, vagy a timeout bekövetkezik. Ha callback mechanizmusra van szükség, akkor egy külön szálat kell indítani, amely futtatja a frame source-ot, és új frame esemény bekövetkezésekor meghívja a beregisztrált callback objektumot.

2.2. Áttekintés



2.1. ábra. *Komponensek*

A megvalósított rendszer a következő komponensekből áll (2.1. ábra):

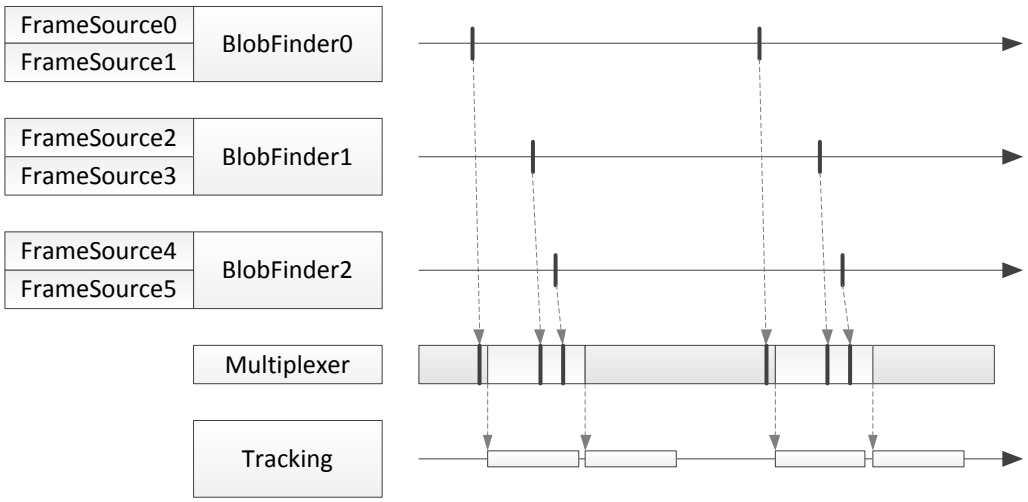
- **FrameSources:** Kamerák kezelését végző modulok. A modulok, kamera típusonként, DLL-eket jelentenek. A következő típusú kamerák támogatottak: DirectShow, uEye, Basler GigE.
- **BlobFinders:** Folt keresést végző modulok, melyek feladata a kamerák képén megkeresni a beállított szűrési paraméterek alapján a markereket. A folt keresés történhet a videókártyán CUDA-val, illetve a processzoron.
- **RemoteHost:** Kamera, és folt kereső objektumokat futtat, amelyeket távolról, hálózaton keresztül, elérhetővé tesz. Az elosztottságot az *Internet Communications Engine* (ICE)-al valósítottam meg [1].
- **MX_Remote:** A *RemoteHost* által elérhetővé tett távoli kamerákat, és folt keresőket kezeli. Kívülről lokális kamerának, vagy folt keresőnek látszik[1].
- **Tracking:** A 2D-s marker pozíciókból előállítja a markerek 3D-s pozícióját. A markereket követi, illetve lehetőség van a mozgás felvételére, és visszajátszására is. Magasabb szintű funkciókat is megvalósít: pontokba merev test illesztés, és a pontok alapján csontváz mozgatása.
- **MarkerTracker:** A *Tracking* alacsonyabb szintű funkciói köré felhasználói felület. Lehetőség van a kamerák kalibrálására, a szűrési paraméterek beállítására, a képfeldolgozás állapotainak ellenőrzésére. Továbbá a rekonstruált pontok megtekinthetőek egy 3D-s nézetben.
- **TrackingNET:** A *Tracking* fontosabb funkcióit becsomagoló .NET osztály könyvtár.

[1]MX, mint MiXed, mert kamerát, és folt keresőt is tartalmaz

- **View3D:** Windows Presentation Foundation-hez (WPF) Direct3D 9-et használó 3D megjelenítő. Hasonlít a beépített Viewport-hoz, de annál alacsonyabb szintű, és könnyebben bővíthető.
- **Studio:** .NET alapú WPF kliens program, amelyben lehetőség van a mozgás rögzítésére, visszanézésére, szerkesztésére. 3D-s nézetben a pontokba illesztett merev test, és csontváz megtekinthető.

2.3. Működés

2.3.1. Időbeli működés



2.2. ábra. *Tracking időbeli működése*

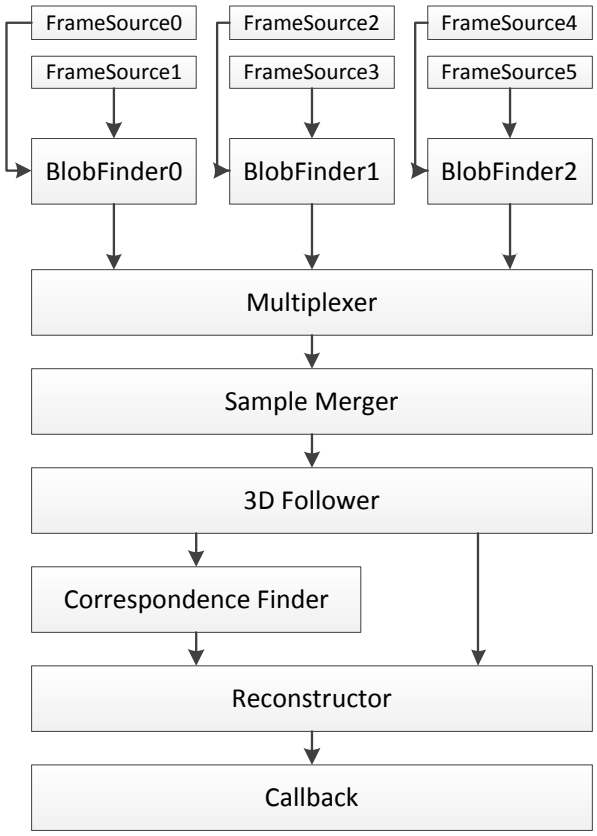
A működés időbeli lefolyása a 2.2. ábrán látható. A frame source-ok képein a markerek pozícióit a folt keresők határozzák meg. Minden folt kereső külön szálon fut. Miután egy folt kereső feldolgozott egy képet, az eredménnyel (marker pozíciók, és timestamp) jelez a *Multiplexer*-nek. A *Multiplexer* feladata a minták összegyűjtése a folt keresőktől. A *Tracking* egy külön szálon várakozik arra, hogy érkezzen új minta, amelyet töröl a *Multiplexer*-ből. Ezután elvégzi a markerek 3D-s pozícióinak rekonstruálását.

Amíg a *Tracking* feldolgozást végez, a mintákat a *Multiplexer* tárolja. A feldolgozás befejeztével a *Tracking* ellenőrzi, hogy érkezett-e közben új minta. Ha igen, akkor ezeket is feldolgozza. Tehát az új mintákat a *Tracking* az első adandó alkalommal feldolgozza.

A kamerák alapesetben nem szinkronizáltak, ezért működik a *Tracking* „mohón". Triggerelt kamerák esetében a minták közel egyszerre érkeznek meg, ilyenkor a *Tracking* megvárja, amíg az összes minta összegyűlik, és csak ezután kezdi ezek feldolgozását.

2.3.2. Rekonstrukció menete

A rekonstrukció menete a 2.3. ábrán látható. A folt keresők által megtalált marker pozíciókat a *Multiplexer* összegyűjti, és az ismert kamera paraméterek alapján a pontok pozícióján



2.3. ábra. *Rekonstrukció folyamata*

elvégzi az inverz torzítást. Az új 2D-s pozíciókat, a régi 2D-s pozíciókkal a *SampleMerger* egyesíti, elvégzi a pontok 2D-s követését, és ez alapján meghatározza a pontok 2D-s sebességét.

Mivel a jelenlegi rendszerben a kamerák nem szinkronizáltak, ezért egy új kép feldolgozása után, a pontosság, és helyesség érdekében a többi kamera képén lévő pontokat a sebességük alapján előre lehet tolni az eltelt idő alapján. Így a pontok valóságbeli helye közelíthető, amíg nem érkezik meg a kamerákból az újabb eredmény.

A *3D follower* az előzőleg meghatározott 3D-s pontok alapján azonosítja a kamerák képein lévő pontokat. Ehhez minden kamerán megkeresi a kamerára visszavetített 3D-s ponthoz legközelebbi 2D-s pontot. Ha egy 3D-s ponthoz legalább 2 kamerán tartozik egyértelműen azonosított 2D-s pont, akkor rekonstruálható a pont 3D-ben.

Megjelenhetnek új pontok is, például takarásból láthatóvá vállnak, ezeket a *3D follower* nem tudja azonosítani. A maradék pontokat egymáshoz kell rendelni, azaz meg kell határozni, hogy az egyik kamera képén lévő pont a többi kamera képén melyik pontnak felel meg. Ezt végzi a *Correspondence Finder*.

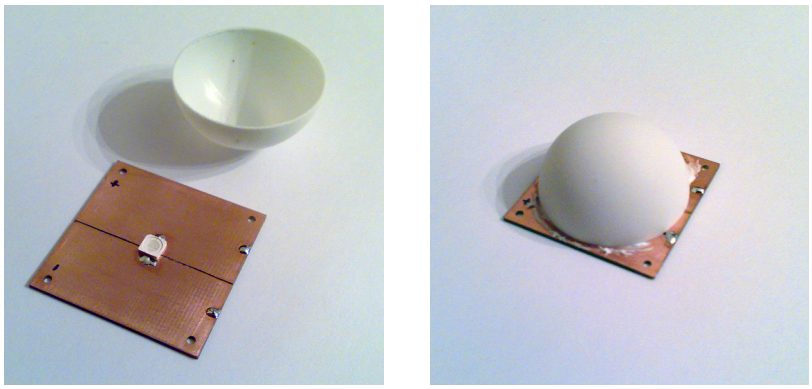
Az előző lépésekben kialakult pont összerendelésekből a *Reconstructor* számolja ki a marker 3D-s pozícióját.

Végül a meghatározott 3D-s pozíciókkal a felhasználó által beállított objektumok kerülnek meghívásra (*Callback*).

3. fejezet

Képfeldolgozás

A képfeldolgozás célja a kamerák képein meghatározni a markerek pozícióit. A markerek fényes LED-ek, körülöttük egy fél ping-pong labdával (3.1. ábra). Így nagy, homogén, világító fényes pontoknak látszanak, és a kamerák expozíciós idejét lecsökkentve elérhető, hogy szinte csak a markerek látszódjanak.



3.1. ábra. *A marker*

A megvalósított rendszerben a következő kamera típusokat lehet használni:

- **DirectShow:** Windows-os web kamerák, videó digitalizáló kártyák,
- **uEye:** uEye ipari kamerák,
- **Basler GigE:** Basler ipari kamerák, gigabit Ethernet-el csatlakoznak a számítógéphez.

A rendszerben alkalmazott 6 db Basler scA640-120gc kamera felbontása 658×492 pixel, YUV422 pixel formátumban kódolják a képeket, és maximálisan 120 FPS-el működnek. Egy kamera által előállított maximális adatmennyiség:

$$\underbrace{658}_{\text{wres}} \cdot \underbrace{492}_{\text{hres}} \cdot \underbrace{2}_{\text{YUV422}} \cdot \underbrace{120}_{\text{FPS}} \approx 74\,\text{MiB/s}$$

A 6 kamera által generált adatmennyiség 120 FPS-nél 444 MiB/s. Két képkocka érkezése között pedig mindössze 8 ms telik el.

3.1. Áttekintés

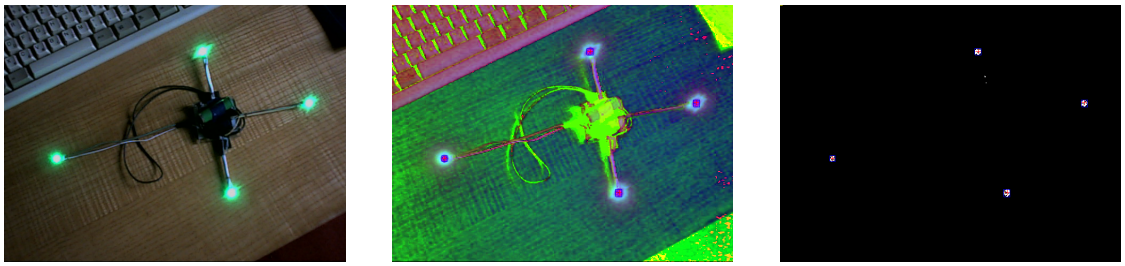
Általánosságban a markerek pozíciójának meghatározása a kamerák képén a következő lépésekből áll (3.2. ábra):

1. **A kamera képének konvertálása HSV színrendszerbe.** A HSV színrendszer a színeket az RGB színrendszerhez képest természetesebben, intuitívabban írja le. Egy színt a színárnyalat (*Hue*), telítettség (*Saturation*) és világosság (*Value*) ad meg.

2. Minden beállított szűrőre

   (a) **Szűrés.** Ennek a lépésnek a bemenete a HSV-be alakított kép, és a HSV szűrési tartomány. A kimenete egy bináris – fekete-fehér – kép. Ezen a szűrt képen a fehér pixelek jelölik azokat a pixeleket, amelyek a megadott HSV tartományon belül vannak, azaz a markerek helyét a képen.

   (b) **Foltok pozíciójának meghatározása.** A szűrés során előállított bináris képen az összetartozó foltokat kell megtalálni, amelyek súlypontja fogja megadni a markerek pozícióját a kamerák képein. Azaz a fehér pixeleket fel kell címkézni, úgy hogy az egymás melletti pixelek ugyanazt a címkét kapják (*connected component labeling*).



(a) RGB **(b)** HSV **(c)** Szűrt

3.2. ábra. *Képfeldolgozási lépések*

3.2. Képfeldolgozás a GPU-n

A képek HSV színtartományba alakítása, és a szűrés során minden pixelre ugyanazt a műveletet kell elvégezni, így a képfeldolgozás nagy része erősen párhuzamosítható, amely lehetővé teszi a GPU-kban (*Graphics Processing Unit*) lévő hatalmas számítási kapacitás kiaknázását. A GPU-n történő képfeldolgozást CUDA 3.2-ben implementáltam.

3.2.1. CUDA

2006 novemberében mutatta be az Nvidia a GPU-iban a CUDA-t (*Compute Unified Device Architecture*), egy általános célú számításokra alkalmazható párhuzamos architektúrát. Programozása a *C for CUDA*-n keresztül történik, amely a C nyelven alapul Nvidia specifikus kiterjesztésekkel.

3.2.2. Működés

Egy folt kereső objektumba több frame source objektumot is be lehet regisztrálni. Mindegyik frame source-hoz tartozik egy szál, amely a frame source-ot futtatja. Miután egy új kép érkezett, azt bemásolja egy bufferbe. A buffer a kép számára fenntartott CPU-n és GPU-n található memória területekre tárol pointereket. Ezután a buffer bekerül egy sorba, jelezve, hogy készen áll egy új kép a feldolgozásra.

A folt keresőt futtató szálon a folt kereső várakozik a buffer sorra, hogy érkezzen egy új feldolgozásra váró kép. Miután ez megtörtént, a buffer tartalmát feltölti a GPU memóriába, ahonnan átalakítja HSV színrendszerbe. Ezután a beállított HSV szűrök mindegyikére elvégzi egyesével a szűrést, majd az így kapott bináris kép sorainak RLE tömörítését. A tömörített kép letöltése után az RLE sorok fentről lefelé történő egyesítésével elvégzi a foltok, és azok súlypontjának meghatározását.

A folt kereső a megkapott foltokon ezután a súlyuk (fehér pixelek száma) alapján még egy szűrést végez. A felhasználó megadhat egy minimális, és maximális súlyt, amellyel például a néhány pixeles zaj okozta foltokat lehet kiszűrni.

Végül a felhasználó által beregisztrált objektum megkapja a megtalált foltokat. Az egész folyamat a 3.3. ábrán látható.

Bufferelés

Egy frame source-hoz több buffer-t is létre lehet hozni, így dupla bufferelést is meg lehet valósítani. Azaz az egyik buffer-ben történik a képfeldolgozás, miközben a másik bufferbe egy új kép kerül be. Így elérhető a 120 FPS-el érkező képek feldolgozása is. Egy darab buffer alkalmazása esetén néha kimaradtak képek, és a feldolgozás 60 és 120 FPS között ugrált.

RGBA

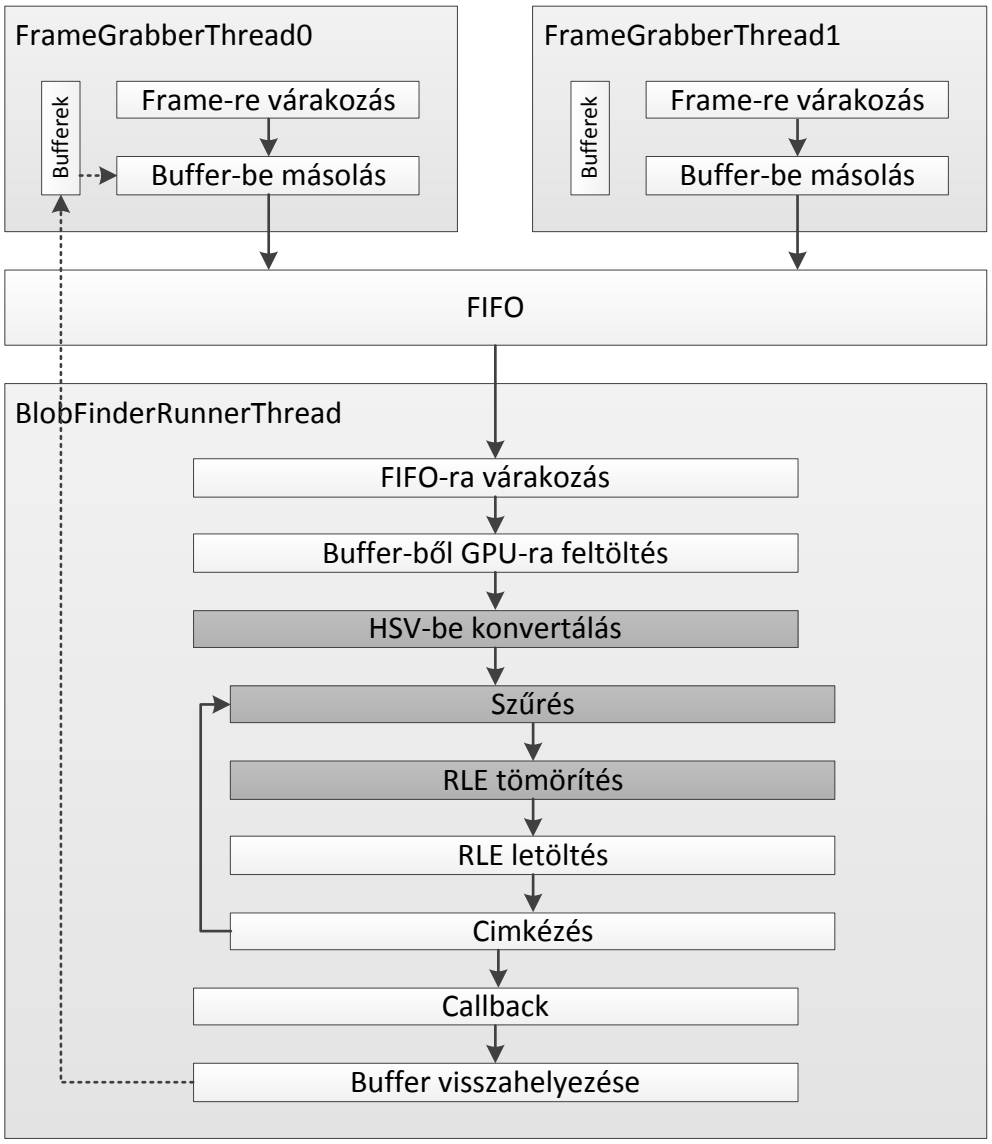
Az RGB és HSV színrendszerben lévő képek pixeleit 3 byte-al le lehet írni, azonban a videókártyák a 32 bites címhatárra igazított változókat gyorsabban tudják írni, és olvasni. Ezért a GPU-n a kamera kép feltöltése után, egy új lépés közbe iktatása árán a folt kereső a képet átalakítja RGBA formátumba, azaz a pixeleket 32 bites címhatárra igazítja.

Így a GPU memóriában lévő kép négycsatornássá vált, és elérhetővé váltak a pixelek olvasására a textúra mintavételező utasítások is, amelyek 1,2 vagy 4 csatornás képeken működnek. Ennek nagy előnye, hogy a GPU-kba épített mintavételező hardware-t használja, amely teljesítmény növekedéssel jár (mérések a 3.2.3. részben).

RLE tömörítés

GPU-n a pixelek címkézését hatékonyan megvalósítani nehéz feladat, egyszerűbb a CPU-n megoldani. Így a bináris képre szükség van a memóriában, amely GPU-ról való visszamásolása időigényes művelet. A visszamásolandó adatmennyiség csökkenthető RLE tömörítéssel.

A *Run-Length Encoding* tömörítés alkalmazásával egyszerűbb ábrák tömöríthetőek. Az



3.3. ábra. *GPU-s folt keresés*

egy sorban egymás mellett lévő ugyanolyan pixeleket, futamokat, nem kell egyesével tárolni, hanem elég a futam kezdetén megadni egyszer a pixelek színét, és a futamban lévő pixelek számát.

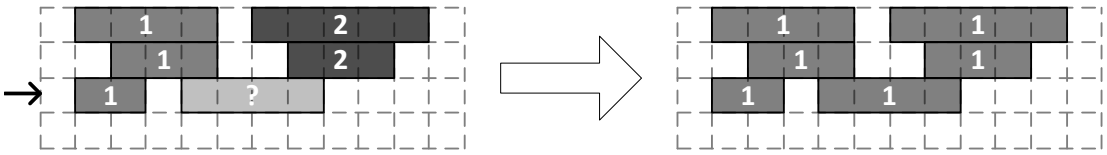
Az implementációban a futamok a fehér pixelek kezdő és végpontját tartalmazzák. Egy 1024×1024 méretű bináris kép letöltése tömörítve, soronként maximum 16 futammal 128 KiB, tömörítetlenül 1 MiB.

Folt keresés

Az RLE-vel tömörített sorokon fentről lefelé végig kell iterálni, és az aktuális sort a megelőző sorral kell egyesíteni [25].

Minden futamra meg kell vizsgálni, hogy az előző sorból mely futamok vannak felette.

Az aktuális sorban az éppen aktuális futam a felette lévő futam azonosítóját kapja meg. Probléma akkor van, ha több futam is van felette. Ilyenkor a futam azonosítók minimuma lesz az új azonosító, azonban az előző sor többi futamát is át kell nevezni, és az eddigi 2 külön foltot egyesíteni kell (3.4. ábra).



3.4. ábra. *Sorok egyesítése*

3.2.3. Eredmények

A különböző folt kereső módszereket a 3.1. táblázat tartalmazza. A táblázat *OpenCV* oszlopa az OpenCV, és a cvBlobsLib[1] könyvtárakkal készült méréseket tartalmazza. A cvBlobsLib könyvtár lineáris idejű algoritmussal végzi a pixelek címkézését [10]. A *CUDA naiv* oszlop, egy naiv implementációt jelent, amely közvetlenül a 3 byte-os pixeleken dolgozik. A *CUDA textúra* oszlop a korábban kifejtett módon a képek pixeleihez textúrázó műveleteken keresztül fér hozzá. A táblázatban a feltöltés a CPU-ról GPU-ra másolást, a letöltés a GPU-ról CPU-ra másolást jelenti.

A mérés során a bemeneti kép ugyanaz az 1024×1024 méretű RGB kép volt, és minden iteráció között 50 ms-t várt a teszt program (20 Hz-es kamerát szimulálva). A teszteket a következő konfigurációjú számítógépen végeztem el: Intel Core 2 Quad Q9550 (2.83 Ghz, 2×6 MiB L2), Nvidia 9800GT 512 MiB, Windows XP (x86) SP3.

3.1. táblázat. *Képfeldolgozási módszerek összehasonlítása*

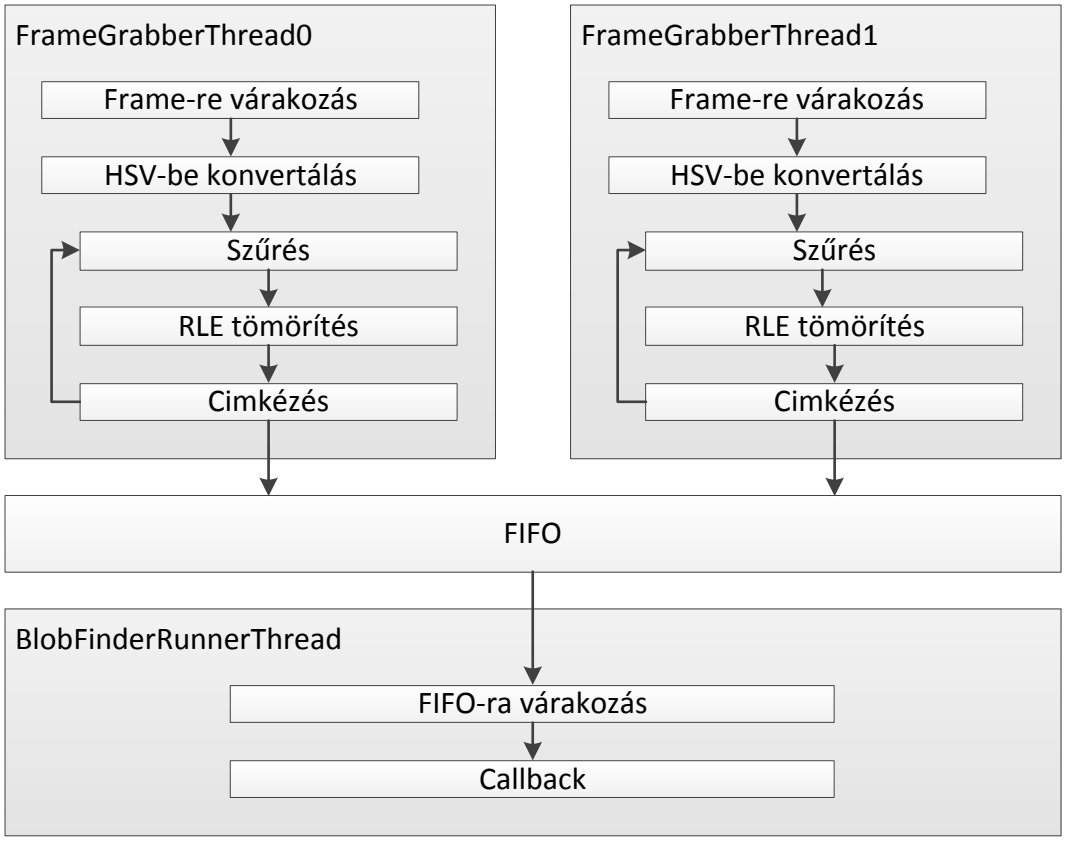
	OpenCV	CUDA naiv	CUDA textúra
Feltöltés	-	1.2 ms	1.2 ms
RGB - RGBA	-	-	0.7 ms
RGB - HSV	14.5 ms	5.5 ms	0.3 ms
Szűrés	4.5 ms	3.9 ms	0.3 ms
RLE	-	0.9 ms	0.9 ms
Letöltés	-	0.2 ms	0.2 ms
Cimkézés	12.1 ms	0.3 ms	0.3 ms
Összesen:	31.1 ms	12.0 ms	3.9 ms

Az eredményekből jól látszik, hogy a GPU-n egy extra lépés közbe iktatásával, az RGBA-ba történő konvertálással, és ezután a textúra mintavételező hardware használatával a naiv megoldáshoz képest több, mint 300%-os gyorsulás érhető el. Továbbá így a 120 FPS jelentette 8 ms alatt történik meg egy kép feldolgozása.

[1] http://opencv.willowgarage.com/wiki/cvBlobsLib

3.3. Képfeldolgozás a CPU-n

A 3.2 részben leírt módszert kis módosításokkal implementáltam a CPU-n is. Minden frame source-hoz tartozik egy szál, amely várakozik az új kép érkezésére. Miután ez megtörtént, a szálon a folt kereső minden pixelt, helyben, HSV-be alakít, elvégzi a szűrést, és az RLE futamok létrehozását. A sorok végén, az RLE futamokat egyesíti az előző sor futamaival. Végül az előállított eredményt behelyezi egy sorba. Erre a sorra várakozik a folt kereső, a saját szálján, hogy végül vissza hívja a felhasználót az eredményekkel (3.5. ábra).



3.5. ábra. *CPU-s folt keresés*

3.3.1. Összehasonlítás a GPU-val

Az 3.2. táblázat az ismertetett GPU-s és CPU-s képfeldolgozás eredményeit hasonlítja össze a kamerák számának függvényében. A számok egy darab kép feldolgozására vonatkoznak, a bemenet egy 640x480-as RGB kép. A méréseket a következő konfigurációjú számítógépen végeztem el: Intel Core i5-750 (2.67 GHz), Nvidia GeForce GTX260+, Windows 7 (x64).

3.2. táblázat. *CPU-s és GPU-s módszerek összehasonlítása*

kamerák száma	1	2	3	4	5
GPU	4 ms	6 ms	7 ms	8 ms	10 ms
CPU	16 ms	17 ms	18 ms	18 ms	19 ms

A táblázatból jól látszik, hogy a CPU-s képfeldolgozás több idő, viszont kevésbé függ a kamerák számától, és elég egy átlagos web kamera által másodpercenként 30-szor küldött képeinek feldolgozására.

Érdekesség, hogy ez a fajta CPU-n futó megvalósítás gyorsabb (16 ms), mint az OpenCV és cvBlobsLib alkalmazása (31.1 ms).

3.4. Elosztott képfeldolgozás

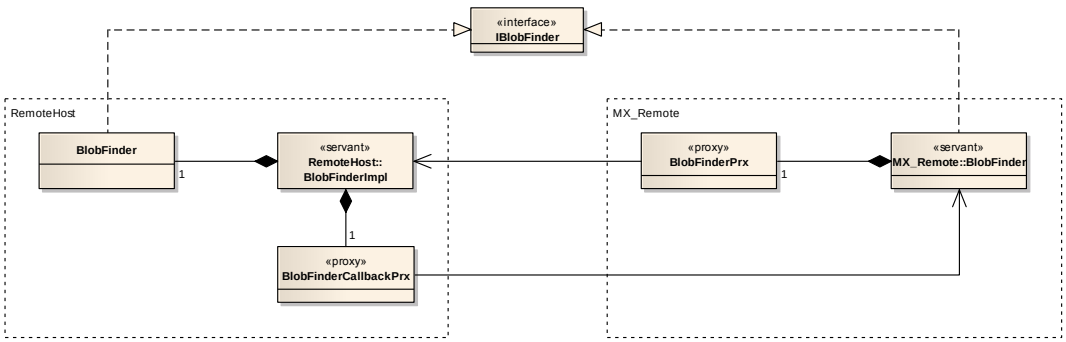
A hálózati kommunikációt az *Internet Communications Engine* (Ice) nevű keretrendszerrel valósítottam meg [1]. Az Ice a CORBA-hoz nagyon hasonló felépítésű rendszer, néhányan a CORBA specifikációt írók közül is részt vettek a megalkotásában. Azonban a CORBA-nál kisebb, és egyszerűbb használni. Támogatja a C++, Java, C#, Objective-C, Python, PHP, és Ruby nyelveket.

A szerver oldali kiszolgáló objektumot *servant*-nak hívják. A szervant metódusait távolról egy *proxy*-n keresztül lehet elérni. Egy proxy objektum a szervant eléréséhez szükséges paramétereket tárolja, ilyen a szervantot futtató szerver címe, és a szervant egyedi azonosítója. A metódus hívások során a paraméterek, visszatérési értékek, kivételek szerializálását az Ice keretrendszer végzi, a felhasználó felé átlátszó módon.

3.4.1. Működés

A kép feldolgozást végző gépen fut a *RemoteHost*, amely tényleges folt kereső objektumokat futtat, és ezeket a *decorator* tervezési mintához [12] hasonlóan becsomagolja egy másik objektumba, a szervantba. A szervant a külső kéréseket a folt kereső objektumhoz delegálja. A rekonstrukciót végző oldal egy folt keresőnek látszó objektumot használ, az *MX_Remote*-ot, amely egy proxy-n keresztül éri el a távol futó tényleges képfeldolgozó objektumot.

A tényleges képfeldolgozást végző objektum az eredményeivel az őt futtató szervant-ot hívja meg, amely egy másik proxy objektumon keresztül meghívja az *MX_Remote*-ban lévő folt keresőnek látszó objektumot. Így ez az objektum is távolról elérhető kell legyen, azaz egy szervant. A felépítést a 3.6. ábrán lévő osztály diagram szemlélteti.



3.6. ábra. *Elosztott működést szemléltető osztály diagram*

Ezzel a megoldással a felhasználó számára átlátszó módon kezelhetőek a lokális, és távoli

folt kereső objektumok, hiszen ő csak egy interfészt használ. Ugyanígy, bármilyen folt keresőt elosztottá lehet tenni, mert a *RemoteHost* oldalon is csak egy interfész kell. Így a rendszer könnyen bővíthető, és konfigurálható.

3.4.2. Idő

A rekonstrukció, és kalibrálás során fontos pontosan ismerni egy közös idő keretben a kamera képek exponálásának idejét.

Rendszer idő

Egyik lehetőség, hogy a számítógépek óráit például NTP (*Network Time Protocol*) használatával szinkronizáljuk, és a kép exponálási időnek az éppen aktuális rendszer időt vesszük. A rendszer időt Windows-on látszólag ezred másodperces pontossággal lehet lekérdezni, azonban Windows XP-ig a rendszer idő csak 15 – 16 ezred másodpercenként frissül, amikor lefut a windows ütemezője [19].

Kamera timestamp

Az ipari kamerákban általában van egy nagy felbontású óra, amely például Basler kameráknál a bekapcsolás óta eltelt időt méri, és az exponáláskori értékét le lehet kérdezni. A kamerák jelenleg nem szinkronizáltan futnak, de egyszerre kerülnek bekapcsolásra, tehát elvileg közel egyszerre indulnak az óráik. A 3.3. táblázat a legutolsó exponálás idejét, és az eltelt időből számított FPS-t mutatja a kamerák saját órái szerint egy idő pillanatban.

3.3. táblázat. *Kamera idők*

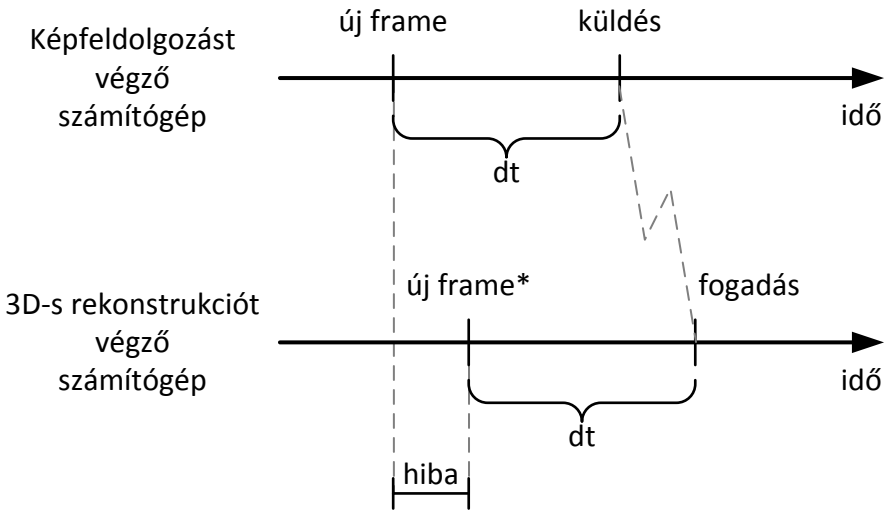
kamera	1	2	3	4	5	6
timestamp	5:37.722	5:37.724	5:37.709	5:37.722	5:37.709	5:37.688
fps	59.99	59.99	59.99	59.99	59.99	59.99

A táblázatban a minimális és maximális érték között 36 ms a különbség, amely 60 FPS-nél körülbelül 2 képkockányi eltérést jelent. Ez az eltérés rekonstrukciónál, és a kalibrálásnál problémát jelent.

Idő eltolás

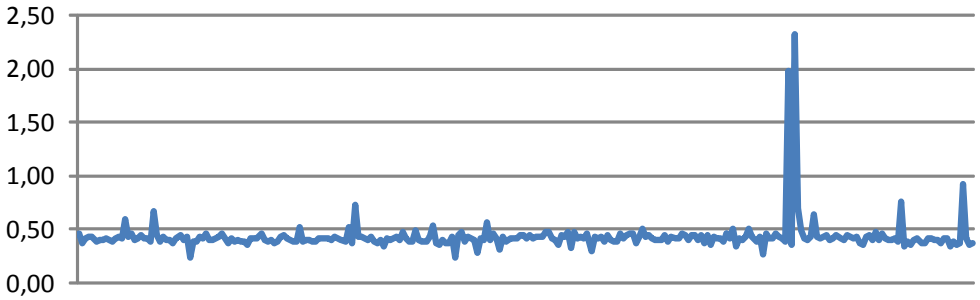
A Windows operációs rendszertől nagy pontossággal lekérdezhető a számítógép indítása óta eltelt idő. Így lekérdezhető a kép érkezésekor az idő, illetve a képfeldolgozás eredményének hálózaton való átküldése előtt is. Ezután a hálózaton ennek a két időnek a különbségét küldöm át. A távoli gépen a metódus meghívásának pillanatában lekérdezem a saját nagy pontosságú órájának az értékét, és ebből kivonom a megkapott eltelt időt. Így a távoli gép idő keretébe került közelítőleg a kamera exponáláskori idő. Az így kiszámolt időben, a hálózati átvitelhez szükséges idő nincs figyelembe véve (3.7. ábra).

Az átviteli idő meghatározására egy szinkron távoli metódus híváshoz szükséges időt mértem le. Ez magába foglalja a távoli oldal meghívását, majd egy nyugtázás vissza küldé-



3.7. ábra. *Idő eltolás*

sét (*round-trip time, RTT*). A mérés eredményét, amely gigabit Ethernet hálózaton készült, a 3.8. ábra tartalmazza. Átlagosan 0,433 ms az RTT, a szórás 0,161 ms.



3.8. ábra. *Round-trip times*

Ezzel a módszerrel a 3D-s rekonstrukciót végző oldalon a timestamp-ek közötti időben lesz egy kicsi eltérés, így a másodpercenkénti képkocka szám kis mértékben ingadozik (3.4. táblázat), de cserébe átlagosan kevesebb, mint 1 ms pontossággal ismert a kép exponálási ideje. A táblázatban a több ms-nyi eltérést az magyarázza, hogy a kamerák nem egyszerre kezdték meg az exponálást. Viszont a számokból látszik, hogy 60 Hz-es képkocka időn belüliek az eltérések.

3.4. táblázat. *Eltolással számított kamera idők*

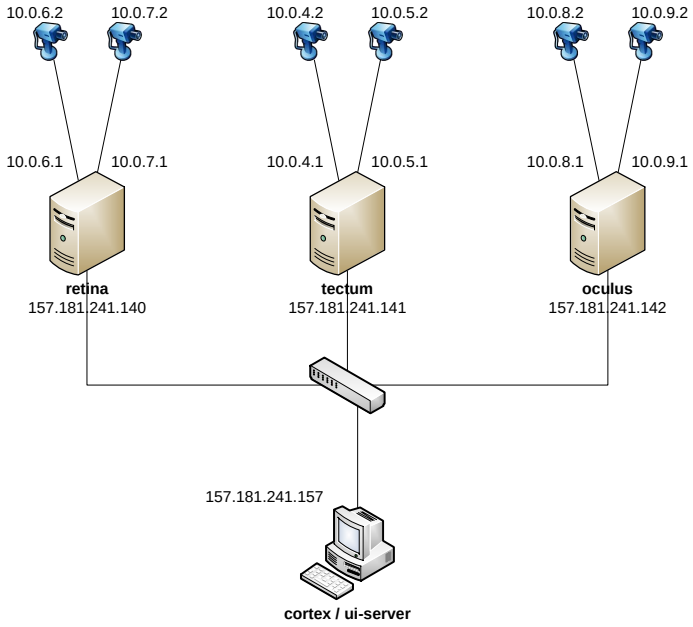
kamera	1	2	3	4	5	6
timestamp	18:57.282	18:57.284	18:57.276	18:57.283	18:57.278	18:57.281
fps	59.99	60.91	60.11	60.05	60.22	60.00

Kompromisszum

Végül mindkét feljebb vázolt megoldást implementáltam, és konfigurálható, hogy a kamera timestamp, vagy az eltelt idő legyen a hálózaton átküldve.

3.4.3. Több interfész

Egy proxy objektum a szervant eléréshez szükséges paramétereket tartalmazza, többek között a címeket, ahol elérhető a szerver. Alapértelmezetten egy szervanthoz létrehozott proxy az őt futtató számítógép összes hálózati interfészének a címét tartalmazza. Így azokat is, amelyekre a gigabites kamerák csatlakoznak (3.9. ábra).



3.9. ábra. *Hálózati topológia*

Egy proxy-ban több cím esetén az Ice véletlenszerű sorrendben próbálja végig a címeket, és mivel a "kamerás hálózatok" kívülről nem érhetőek el, a következő próbálkozás csak a hosszú, több másodperces timeout után történik meg. Ezért például a callback proxy beregisztrálásakor erre figyelni kell, hogy olyan proxy legyen átadva, amely csak a kommunikációra használandó interfész címét tartalmazza.

3.5. Előnézet

A képfeldolgozás során a szűrők beállításához előnézeti képet kell biztosítani. Két fajta callback regisztrálható be egy folt keresőbe az előnézeti képekhez való hozzáféréshez:

- **CPU callback**: A kép a rendszer memóriába kerül, erre kap a callback objektum egy pointert.
- **OpenGL callback**: A kép valahol a GPU memóriában van, amely egy *Pixel Buffer Object* (PBO) objektumon keresztül érhető el [3], másolható textúrába.

CUDA és OpenGL együttműködéssel lehetőség van egy OpenGL-ben létrehozott PBO-ba CUDA memóriából adatokat másolni.

Hálózati képfeldolgozás esetén a *RemoteHost*-beli szervant CPU callback-el regisztrál be a folt keresőhöz, és a megkapott pixelekkel hívja meg az *MX_Remote*-ban lévő callback szervantot, amely szintén a CPU callback-en keresztül értesíti a képek rajzolását kezelő objektumot.

Az előnézeti képek rajzolása a UI szálon történik OpenGL-el, azonban a folt keresők külön szálon futnak, és innen hívnak vissza az elérhető új kamera képpel. Windows operációs rendszeren az OpenGL kontextusok egy időben legfeljebb egy szálon lehetnek aktívak, azonban lehetőség van egy processzen belül több OpenGL kontextus között az OpenGL-es objektumok megosztására. Így a UI szálon lévő OpenGL kontextusban létrehozott textúrába egy másik szálról be lehet másolni a kamera képét.

4. fejezet

3D-s rekonstrukció

4.1. Kamera modell

A kamerák fizikai modelljeit Gary Bradski és Adrian Kaehler könyvének 11. és 12. fejezete tárgyalja részletesen [9]. Az alábbiakban a legfontosabb paramétereket ismertetem.

A valós kamerákat a tűlyuk (*pin-hole*) kamera modellel lehet egyszerűen közelíteni. A kamera egy képzeletbeli falból áll, a közepén egy kisméretű lyukkal. A fal minden fénysugarat blokkol, kivéve azokat, amelyek a tűlyukon keresztül haladnak. Egy igazi tűlyuk kamera igazából nem túl jó, mert a gyors exponálásokhoz nem kap elegendő fényt. Ezért vannak lencsék a szemünkben, és a kamerákban, hogy több fénysugár érkezzen be, mint ami egy kis méretű lyukon keresztül lehetséges. Cserébe a lencsék torzítják a képet.
Egy kamera paraméterei két fő csoportba oszthatók:

- **Intrinsic**: A kamera belső felépítését leíró paraméterek.
- **Extrinsic**: A kamera világbeli helyzetét leíró paraméterek.

4.1.1. Intrinsic paraméterek

- **Fókusz távolság**: f_x, f_y A kamera pozíció és a kép távolsága pixelekben kifejezve. Azért van két fókusztávolság, mert a pixelek az olcsóbb kamerákon téglalap alakúak lehetnek a négyzet helyett.
- **Optikai középpont (*principle point*)**: c_x, c_y A kamerában lévő képalkotó chip optikai tengely menti középpontja pixelekben. Ez általában nem a kép középpontja, mert a kamera gyártásakor a chip kicsit arrébb csúszhat.
- **Lencse torzítás**: Két fő komponense van:
  - **Radiális**: k_1, k_2, k_3 Főleg a kép széleihez közel vehető észre, az optikai középpontnál ennek a mértéke 0. Ez okozza a hordó, vagy párna torzítást. Az optikai középpont körüli Taylor sorba fejtett számtani sor első 2, vagy 3 páros számú együtthatójával jellemezhető. A 3. együttható nagy látószögű kamerák esetében érdekes.

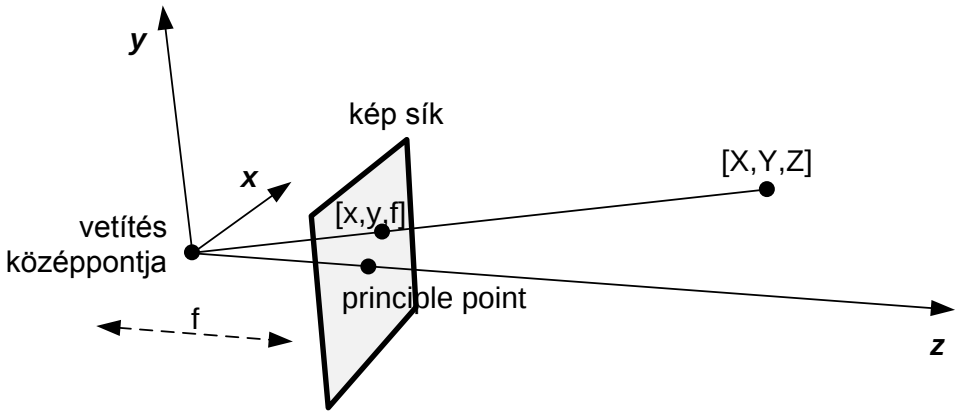
– **Tangenciális**: p_0, p_1 Az okozza, hogy a gyártás során a lencse nem lesz teljesen párhuzamos a kép síkkal.

A középpontos vetítés ezekkel a paraméterekkel a következő módon írható le (x, y a kamera képére vetített pont, X, Y, Z a pont világbeli pozíciója, a kamera az origóban van, és a Z tengely irányába néz, 4.1. ábra):

$$x = f_x \left(\frac{X}{Z} \right) + c_x$$
$$y = f_y \left(\frac{Y}{Z} \right) + c_y$$

Ugyanez mátrixos formában is felírható, az eredmény homogén koordinátákban lesz:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$



4.1. ábra. *Intrinsic kamera paraméterek*

A kamerába érkező képet a lencse torzítja. Egy nem torzított pixel (x, y) torzított koordinátáinak meghatározásához első lépésként normalizált kamera koordinátákba ($\hat{x}$, $\hat{y}$) kell alakítani.

$$\hat{x} = \frac{x - c_x}{f_x}$$
$$\hat{y} = \frac{y - c_y}{f_y}$$

Az $\hat{x} = 0$ és $\hat{y} = 0$ pont az optikai középpont, és a 4.1. ábrán a $z = f$ helyett a $z = 1$ síkon vannak ezek a pontok.

A radiális torzítás okozta elmozdulás (4.2. ábra) a következő módon határozható meg

(r az optikai középponttól való távolság, azaz $r = \sqrt{\hat{x}^2 + \hat{y}^2}$):

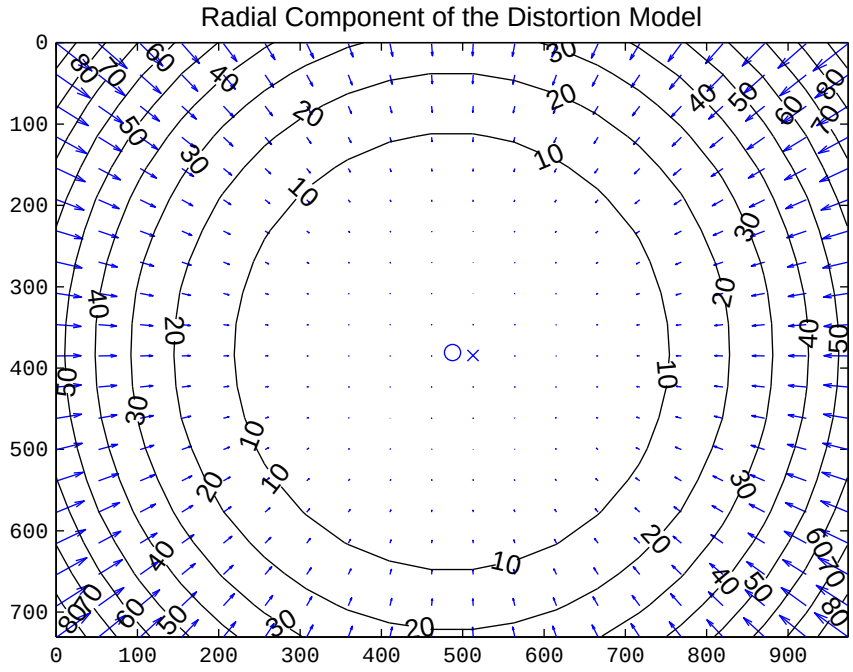
$$dr_x = \hat{x}\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right)$$
$$dr_y = \hat{y}\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right)$$

A tangenciális torzítás okozta elmozdulás a következő módon határozható meg:

$$dp_x = 2p_1\hat{y} + p_2\left(r^2 + 2\hat{x}^2\right)$$
$$dp_y = p_1\left(r^2 + 2\hat{y}^2\right) + 2p_2\hat{x}$$

A marker torzított pozíciója normalizált kamera koordinátákban:

$$\hat{x'} = \hat{x} + dr_x + dp_x$$
$$\hat{y'} = \hat{y} + dr_y + dp_y$$




```
Pixel error                  = [0.2604, 0.2501]
Focal Length                 = (841.805, 842.938)  +/- [19.09, 19.02]
Principal Point              = (486.581, 379.925)  +/- [3.551, 3.637]
Skew                         = 0                   +/- 0
Radial coefficients          = (-0.379, 0.1608, -0.[0.01738, 0.01755, 0]
Tangential coefficients  = (0.002665, 0.00209)+/- [0.0011, 0.0007891]
```

4.2. ábra. *Radiális torzítás*

A képfeldolgozás eredménye a markerek pozíciója torzítás után, ezért torzítás függvény inverzére van szükség. Ez iteratívan a Newton módszerrel közelíthető.

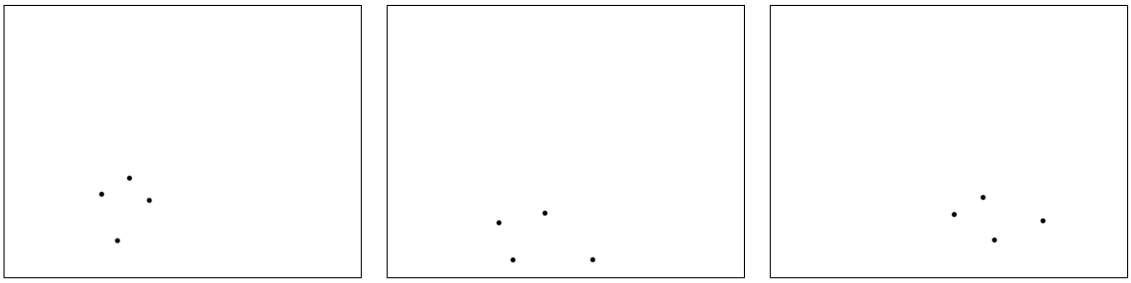
4.1.2. Extrinsic paraméterek

- **Pozíció**: **t** a kamera világbeli pozíciója

- **Orientáció**: **R** a kamera világbeli nézeti iránya, amely megadható egy forgatás mátrixszal.

4.2. Pont összerendelés

A képfeldolgozás során meghatározásra kerültek a markerek 2D-s pozíciói. A 3D-s pozíció meghatározás első lépéseként a kamerák képén lévő pontokat egymáshoz kell rendelni (4.3. ábra). Ez a pont összerendelési probléma (*correspondence problem*).

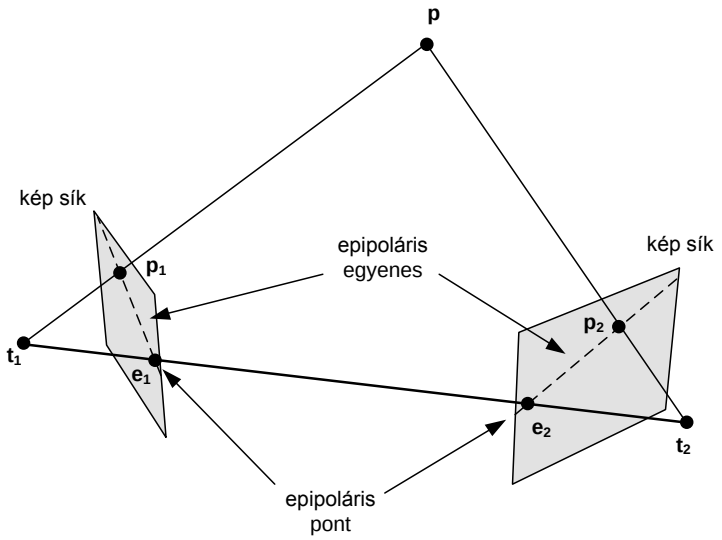


(a) Nézet 1 **(b)** Nézet 2 **(c)** Nézet 3

4.3. ábra. *Foltok*

Az összes lehetőség végigpróbálása időigényes feladat, de erre nincs is szükség, mert az epipoláris geometria egyszerűsíti a problémát.

4.2.1. Epipoláris geometria



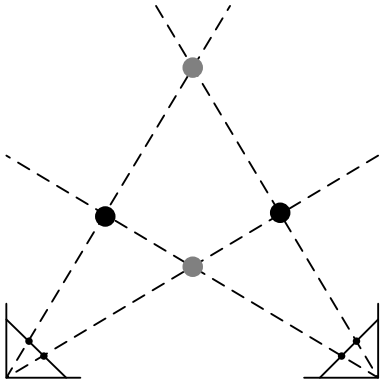
4.4. ábra. *Epipoláris sík*

A kamerák ($\mathbf{t_1}$, $\mathbf{t_2}$), a marker ($\mathbf{p}$) és a marker vetületi pontjai ($\mathbf{p_1}$, $\mathbf{p_2}$) egy síkon, az epipoláris síkon vannak (4.4. ábra). A kamerákat összekötő egyenes a képsíkokat az epipoláris pontokban (*epipole*) metszi ($\mathbf{e_1}$, $\mathbf{e_2}$). Az epipoláris pontot a marker vetületi pontjával ($\mathbf{p_1}$, $\mathbf{p_2}$) összekötő egyenes az epipoláris egyenes. Minden epipoláris egyenes az

epipoláris pontban metszi egymást. Az egyik kamera képén lévő pont ($\mathbf{p_1}$) a másik kamera megfelelő epipoláris egyenese ($\mathbf{e_2}$ $\mathbf{p_2}$) mentén van valahol. Ez az epipoláris kényszer [9].

Az epipoláris kényszer felhasználásával csökkenthető a vizsgálandó pontok száma, mert az egyik nézetbeli ponthoz a másik nézetbeli epipoláris egyenes mentén lévő (vagy az epipoláris egyeneshez közel lévő) pontokat kell csak vizsgálni.

Speciális esetben 2 pont is kerülhet ugyanarra az epipoláris síkra, ilyenkor többféleképpen is összerendelhetőek a pontok (4.5. ábra). Azonban ha egy harmadik kamera képén is ismertek a pont vetületei, akkor már egyértelmű az összerendelés. Ezért egy pontot legalább három kamerának kell látnia, hogy az összerendelési algoritmus sikeresen lefusson.



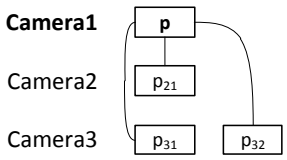
4.5. ábra. *Epipoláris probléma felül nézetben*

4.2.2. Összerendelési algoritmus

Mások optikai motion capture rendszer fejlesztésénél a pontokat úgy rendelik össze, hogy először veszik az egyértelműen összerendelhető pont párokat, és ezeket rekonstruálják. Egyértelműen azok a pontok rendelhetőek össze, amelyekre teljesül az epipoláris kényszer, és az epipoláris egyenesen nincs más pont. Az így kapott pont akkor tekinthető jól rekonstruáltnak, ha még legalább 1 kamerára visszavetíthető [13].

Az így történő pont összerendelés páronkénti próbálkozáson alapul, amely sok kamera, és sok pont esetén időigényes lehet.

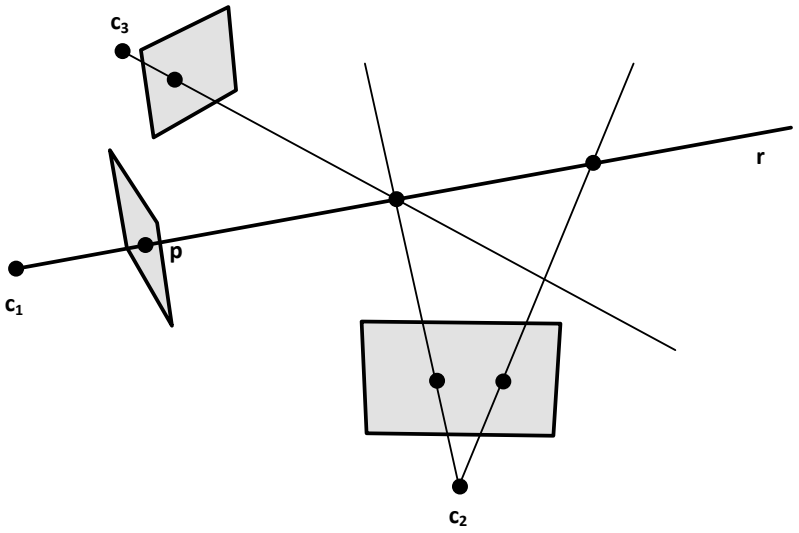
A megvalósított algoritmus vesz egy szabad pontot ($\mathbf{p}$ pont, $\mathbf{c_1}$ kamerában), amelyhez meghatározom a többi kamera képén a szóba jöhető pontokat, azaz azokat, amelyek a $\mathbf{p}$-hez tartozó epipoláris egyenesen, vagy annak néhány pixeles környezetében vannak.



4.6. ábra. *Kezdeti összerendelés*

Ebben a kezdeti összerendelésben (4.6. ábra) kell megkeresni azt a pont összerendelést, amelyek ugyanabban a pontban fogják egymást metszeni.

Minden pont vetítő egyenese és az eredeti **p** ponthoz tartozó vetítő egyenes (**r**) metszés pontját meghatározom. Az így kapott metszés pontokat az **r** vetítő sugárra vetítve a sugáron csomók alakulnak ki. Ahol a pontok elég közel vannak egymáshoz sugár paraméterben mérve, egy csomóba tartoznak. Ezekből a csomókból a kamerák számában a maximálist kell megkeresni, amely megadja az összerendelendő pontokat (4.7. ábra).



4.7. ábra. *Csomók a vetítő egyenesen*

Ha ez az összerendelés **p**-vel együtt legalább 3 kamerából áll, akkor a pont rekonstruálható, és a pontokat törlöm a kamerákból. Ha nincs legalább 3 kamerából meg az összerendelés, akkor a kamerákból csak az eredeti **p** pontot törlöm. Így a többi pont még más összerendelésben felhasználható. Az algoritmus addig fut, amíg el nem fogynak a pontok.

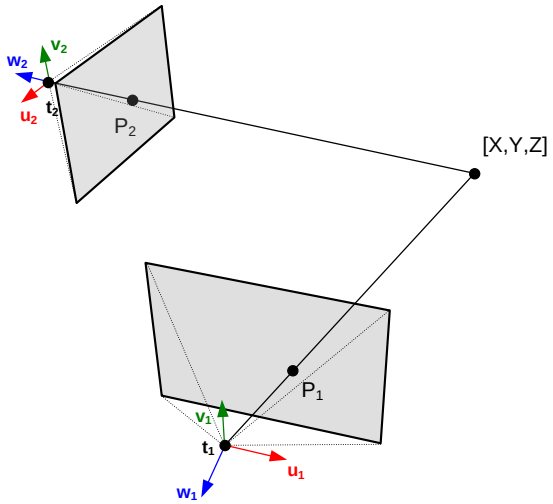
4.3. Rekonstrukció

A feldolgozás ezen szakaszában már ismertek a pont összerendelések, azaz egy marker vetületi pontjai a különböző kamerákban egymáshoz lettek rendelve. Ezek alapján a marker 3D-s pozíciója meghatározható.

A kamerákból ($\mathbf{o_n}$) a kamerán lévő pont irányába ($\mathbf{v_n}$) indított sugarak metszés pontjában lesz a marker 3D-s pozíciója (**p**) (4.8. ábra). Három kamera esetén a következő egyenletek írhatóak fel:

$$\mathbf{o_1} + t_1 \cdot \mathbf{v_1} = \mathbf{p}$$
$$\mathbf{o_2} + t_2 \cdot \mathbf{v_2} = \mathbf{p}$$
$$\mathbf{o_3} + t_3 \cdot \mathbf{v_3} = \mathbf{p}$$

Ezt átrendezve, és mátrixos formára alakítva egy $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ alakú túlhatározott egyenletrendszert kellene megoldani. Mivel a sugarak kitérőek lehetnek kalibrálási pontatlanság,



4.8. ábra. *Vetítő egyenesek*

vagy 2D-s mérési pontatlanság miatt, ezért az egyenletrendszernek nincs megoldása. Helyette a 3D-s pozíció azzal az $\mathbf{x}$ vektorral közelíthető, amelynek négyzetes hibája a legkisebb. Ez geometriailag az a pont lesz, amely a vetítő egyenesekhez a lehető legközelebb van.

4.4. Pont követés

Az 4.2 részben leírt módon minden új kép feldolgozása után összerendelhetőek a pontok, és ez alapján meghatározható a 3D-s pozíciójuk. Azonban a pontok így az időben nincsenek azonosítva, illetve így egy markert mindig legalább három kamerának kell látnia, pedig elég kettő kamera is a 3D-s rekonstrukcióhoz.

Minden pontnak kell adni egy egyedi azonosító számot, és a pontokat követni kell. A követés történhet 2D-ben a kamerák képein, és 3D-ben is.

Lehetséges lenne egy kezdeti összerendelés után a markerek vetületeit külön-külön követni a kamerák képein, azonban ha kettő marker közel kerül egymáshoz, előfordulhat, hogy az azonosságuk megcserélődik. Később, miután eltávolodtak, ez problémához vezethet, mert a kamerák egy részén megmaradna az eredeti azonosítás, a kamerák többi részén a felcserélődött. Így a vetítő egyenesek széttartóak lennének.

Ezért a követés elsődlegesen 3D-ben történik. Az előző állapotok alapján a sebességekből jósolt pozíciót a kamerák képeire visszavetítve a hozzá legközelebbi marker vetület lesz az adott markerként azonosítva. Ebben az esetben is előfordulhat, hogy ha kettő marker közel kerül egymáshoz, akkor a vetületeik felcserélődhetnek. Azonban így az idő előre haladtával, amikor elég messze kerültek egymástól, a visszavetítés a megfelelő marker vetületeket fogja azonosítani, és a vetítő egyenesek nem fognak széttartani. Azonban az előfordulhat, hogy a két marker azonossága felcserélődik.

Mivel a kamerák nem feltétlenül szinkronizáltak, ezért egy új képfeldolgozás eredményének megérkezése után, a még nem frissült kamerákon a marker vetületek pozícióját a sebességük alapján lehet közelíteni. A 2D-s követésre a marker vetületek sebességének meghatározása miatt van szükség.

5. fejezet

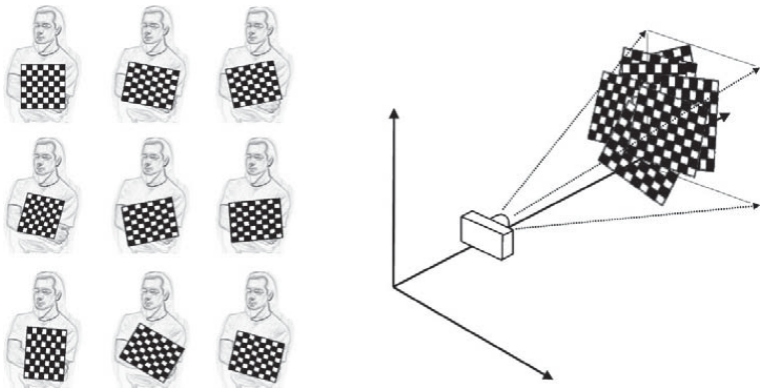
Kalibrálás

A kalibrálás során a 4.1 részben ismertetett kamera paraméterek pontos meghatározása a cél. A fejezetben bemutatom a különböző általam kipróbált kalibrálási módszereket, és a fejezet végén ezeket összehasonlítom.

5.1. Kalibrálás OpenCV-vel

Az OpenCV tartalmaz beépített függvényeket a kamerák intrinsic, és extrinsic paramétereinek meghatározására.

Az intrinsic kalibrálás során egy síkbeli objektum néhány képbeli és lokális pontját kell megadni több nézetből. A síkbeli objektum egy sakktábla, amely belső sarkainak megtalálásához az OpenCV szintén tartalmaz függvényt. Minden nézetre vissza lehet kapni az extrinsic paramétereket a sakktáblához képest (5.1. ábra). Az intrinsic kalibrálás során az OpenCV egy kezdeti közelítő értékből kiindulva a *Levenberg-Marquardt* algoritmussal optimalizációt végez a kamera paraméterekre a visszavetítési hiba (*reprojection error*) minimalizálásával. A visszavetítési hiba a képen lévő pontok és a kép síkra vetített pontok közötti távolság négyzetes összege [9] [6].



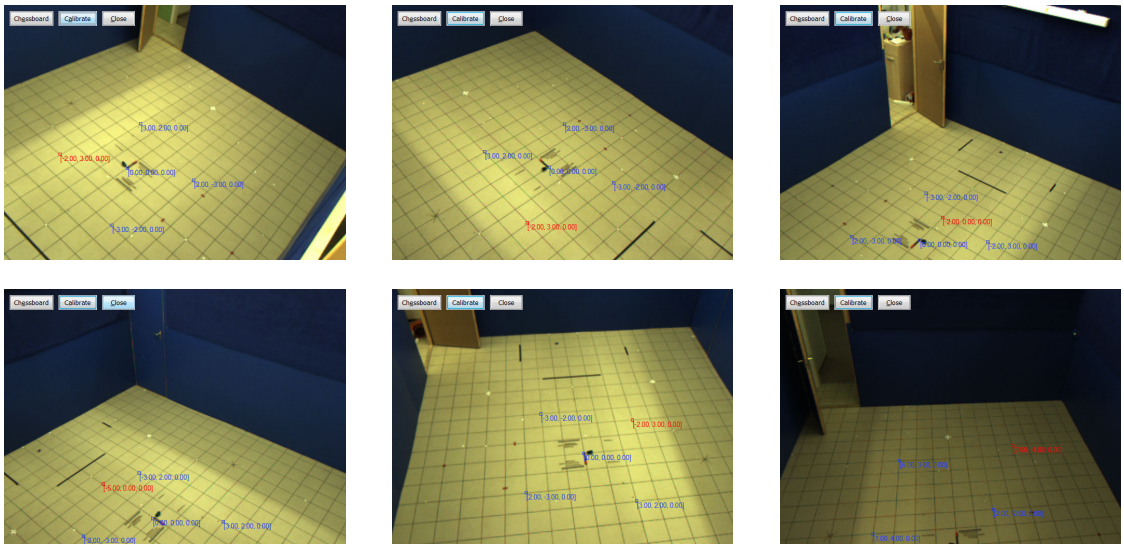
5.1. ábra. *Intrinsic kalibrálás OpenCV-vel [9]*

Az extrinsic paraméterek meghatározása hasonlóan történik, meg kell adni képbeli pontokat, és a világbeli helyzetüket. A már ismert intrinsic paraméterek felhasználásával meghatározható a kamera helye, és orientációja.

5.1.1. Kalibrálás kézzel

Az intrinsic paraméterek meghatározása az OpenCV által javasolt módon történik. Azaz egy sakktáblát a kamerának kb 10 nézetből megmutatva, az OpenCV minden nézeten megkeresi a sakktábla belső csúcsait, és ezekből meghatározza az intrinsic paramétereket.

Az extrinsic paraméterek meghatározásához a kamera képén kézzel meg kell jelölni legalább 4 pontot, és ezekhez meg kell adni a világbeli helyzetüket. Fontos, hogy az origó ugyanott legyen, és a világ koordináta rendszer tengelyei is ugyanabba az irányba álljanak. Ehhez az origó, és a tengelyek padlóra helyezett tárgyakkal voltak megjelölve. A kalibrálás során az 1 egység meghatározásában sokat segített, hogy a padló csempézett (5.2. ábra).



5.2. ábra. *Extrinsic kalibrálás kézzel*

A megoldás hátránya, hogy kézzel kell elvégezni, így könnyű eltéveszteni a koordináták meghatározását, és zajt is könnyű bele vinni, amely pontatlanabb kamera paramétereket eredményez. Mivel az extrinsic kalibrálás során a pontok egy síkban, a padlón lettek megadva, ezért az OpenCV úgy optimalizálta a kamerák elrendezését, hogy a padló síkjában legyen a visszavetítési hiba minimális. Ahogy a markerek a padlótól elemelkedtek, a vetítő sugarak egyre jobban kitérnek, amely közeli markerek esetén hibát visz a pont összerendelésbe, és követésbe.

5.1.2. Intrinsic kalibrálás++

A sakktáblás intrinsic kalibrálás során a többször elvégzett kalibrálások mindig kicsit más eredményt adtak, illetve néha irreális eredményt is. Ennek oka a zajos sakktábla csúcs kereséssel, illetve a kevés ponttal magyarázható.

Ezért szükség volt egy pontosabb, automatizáltabb, és könnyebben reprodukálható módszerre, amellyel pontosan meg lehet határozni, hogy hova kerüljön a kalibráló objektum egy pontja az objektum térben, illetve a képen is pontosan meg lehet határozni a pont helyét.

Ha a pont egy fényes kerek dolog, akkor használható a képen lévő pozíció meghatározásra a rendszerben alkalmazott folt kereső (3.2. rész). Ha a kamera nem mozog, akkor nem

szükséges egyszerre meghatározni a kalibráló objektum pontjait, hanem elég mindig csak egy pontot. Így például falra vetített, és mozgatott lézer ponttal elvégezhető a kalibrálás.

Lézer helyett a kalibrálandó kamerát egy monitor elé helyeztem, amelyen fekete háttéren egy fehér kör jelent meg néhány másodpercig. A kör pozícióját egy folt kereső meghatározta a kamera képén, és a végleges folt pozíció több mintából lett átlagolva. Ezután a kör arrébb jelent meg a képen, és ott is meg lett határozva a kamera képén lévő pozíció. Ez ismétlődött, amíg a kör végig ért a monitoron (5.3. ábra). A kalibráló objektum modellbeli pontjai a kör pixel koordinátái lettek. A kör, monitoron való végig iterálását az eredeti módszerhez hasonlóan több nézetből el kell végezni.

5.3. ábra. *Basler kamera intrinsic kalibrálása*

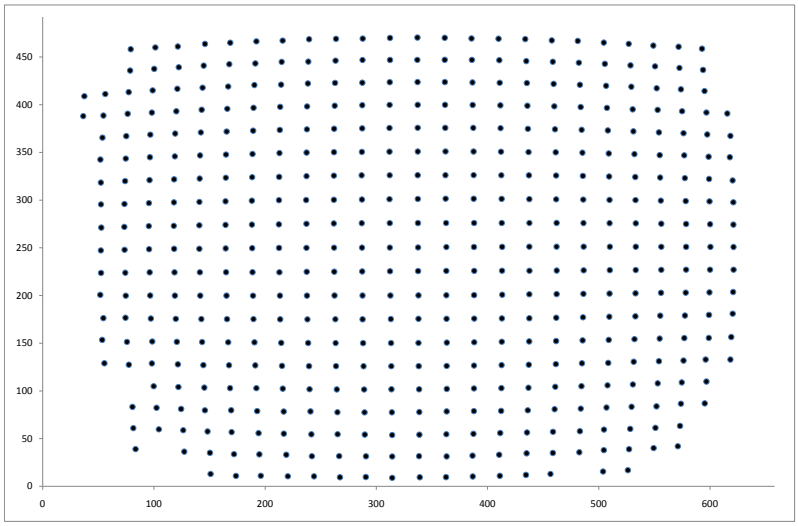
A módszer előnye, hogy több pontot ad bemenetként az intrinsic paramétereket meghatározó függvénynek a sakktáblás módszerhez képest (5.4. ábra), és részben automatizált. Azonban figyelni kell a környezetből jövő becsillanásokra, mert így a kör helyett lehet, hogy a csillanás pozíciója lesz meghatározva.

Hátránya, hogy nagyon időigényes. Ha a kamera 60 FPS-el működik, 20 minta összegyűjtéséhez kell minimum ∼333 ms. Ezután a pontot ki kell kapcsolni, és megvárni, hogy a folt keresőben lévő bufferekből is kifussanak a pontot még tartalmazó képek. Ezután 50 pixellel arrébb kell helyezni, majd a sor végén 50 pixellel lefelé. Ez 1280×1024 pixeles felbontásban $25 \times 20 = 500$-szor fut le. Azaz, nézetenként 3–4 perc. 9 nézet esetén 1 kamera kalibrálásához kb fél óra szükséges.

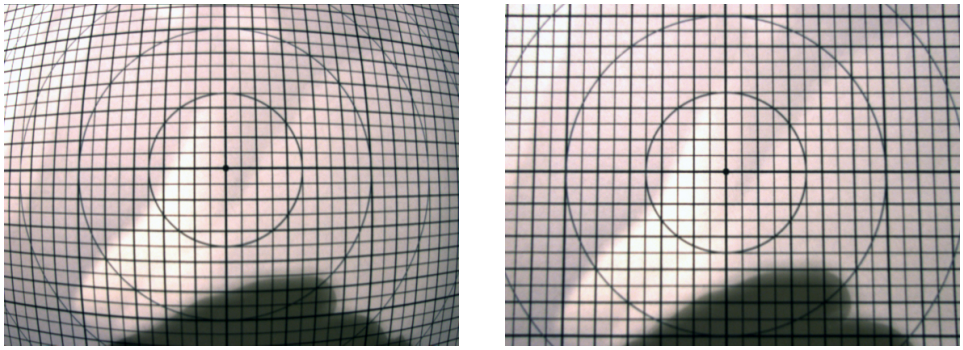
Az eredmények a sakktáblás kalibráláshoz képest jobbak lettek, a kamera képét inverz torzítva ránézésre egyenesebbek lettek az egyenesek (5.5. ábra), mint a sakktáblás módszerrel.

5.1.3. Projektoros extrinsic kalibrálás

A monitoros intrinsic kalibrálás ötlete alapján az extrinsic kalibrálás is pontosítható, gyorsítható. Egy plafonra szerelt projektor a padlóra vetít egy kört, amelyek pozícióját a kamerák képén folt keresők meghatároznak. Hasonlóan, egy ponthoz több minta lesz összegyűjtve, majd átlagolva lesznek. Ezután a padlóra vetített kör arrébb kerül, és végig iterál a padlón.



5.4. ábra. *Egyik Basler kamera torzítása*



(a) Torzított kép **(b)** Inverz torzított kép

5.5. ábra. *Kamera torzítás*

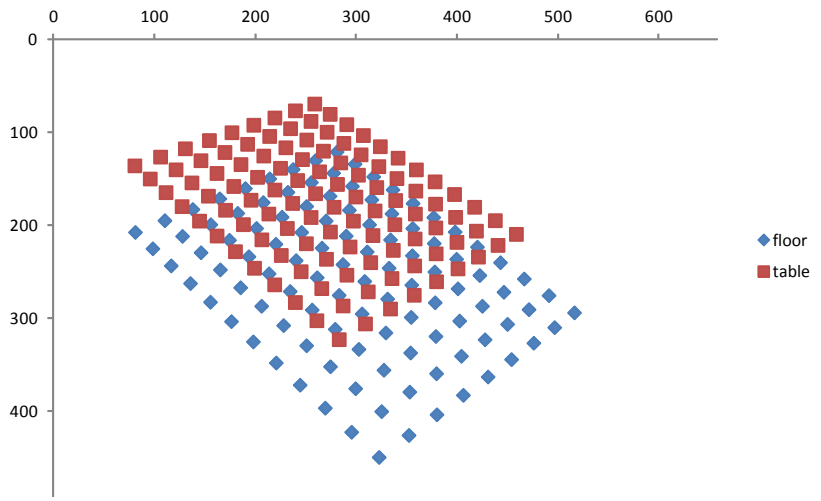
Az intrinsic kalibráláshoz hasonlóan itt is nagyon figyelni kell a csillanásokra, a kalibrálást célszerű sötétben elvégezni. Illetve, a projektoron lévő pixel koordinátákat át kell transzformálni a padlón lévő világ koordináta rendszerbe.

A kézzel történő extrinsic kalibráláshoz képest sokkal jobb eredményeket adott ez a módszer. A padló közelében a vetítő egyenesek szinte egy pontban találkoztak. Azonban a padlótól felemelkedve a vetítő egyenesek kezdtek kitérni, megnőtt közöttük a távolság.

5.1.4. Asztalos projektoros extrinsic kalibrálás

A projektoros kalibrálás a padló közelében nagyon jó eredményeket adott, de a padló felett nőtt a hiba. Ezért egy asztal síkjára vetítve is elvégeztem a projektoros kalibrálást. Az extrinsic paraméterek a padló, illetve asztal pontjaiból lettek meghatározva (5.6. ábra).

Figyelni kellett arra, hogy a padlón, illetve asztalon lévő pontok azonos koordináta rendszerbe kerüljenek.



5.6. ábra. *Egyik Basler kamera torzítása*

Így a vetítő egyenesek már nem csak a padlón, hanem a padló felett is közel kerültek egymáshoz, fej magasságban is viszonylag pontos volt a markerek 3D-s pozíciójának meghatározása.

5.2. Kötegelt behangolás

A kötegelt behangolással (*bundle adjustment*) 3D-s pontok 2D-s vetületeinek megadásával finomhangolhatóak egy előzetes becslésből a kamera paraméterek. Így az előzőleg bemutatott kalibrálási módszerek által adott eredmények tovább pontosíthatóak. A megvalósításhoz az *sba* [18] könyvtárat használtam. A kötegelt behangolás a Levenberg-Marquard algoritmussal a visszavetítési hiba négyzetét nem lineárisan minimalizálja.

A Levenberg-Marquardt minimalizálási eljárás egy iteratív módszer, amely egy nemlineáris függvény lokális minimumát keresi meg, a négyzetes hiba minimalizálásával. A Levenberg-Marquardt eljárás a Gauss-Newton és a gradiens módszerek keveréke [17]. Amikor az aktuális megoldás messze van a lokális minimumtól, az algoritmus gradiens módszerrel működik. Amikor az aktuális megoldás közel kerül a lokális minimumhoz, akkor Gauss-Newton módszert alkalmazva gyorsan konvergál a minimumhoz [18].

A kalibrálás során egy kezdeti kalibrálásban, az aktív térben egy markert körbe kell mozgatni úgy, hogy egy időben legalább 2 kamera lássa (*wanding*). A kamera paraméterek azokon a részeken lesznek optimalizálva, ahol járt a marker, ezért a markert az aktív tér alján, közepén, tetején, szélén is mozgatni kell. Az így rögzített mozgásból újra mintavételezéssel megkaphatóak a marker 3D-s koordinátái, és 2D-s vetületi, mért pontjai. Ezekkel az *sba*-t meghívva tovább pontosíthatóak a kamera paraméterek.

Az újra mintavételezés miatt fontos a kamerák exponálási idejének pontos meghatározása (3.4.2 rész), mert néhány frame-nyi különbség a kamerák idejében megnöveli a vetítő egyenesek közötti távolságot, így bizonytalanságot visz a kalibrálásba.

5.3. Összehasonlítás

A fejezetben vázolt módszerek összehasonlítását a 5.1. táblázat tartalmazza. A mérés során 1 darab markert mozgattam az aktív térben kb 2 percen keresztül, az 5.2 részben leírtak szerint. Ezután a különböző kalibrálási módszerekkel előállított kamera paraméterekkel a marker teljes mozgására le mértem a pontosságot: a visszavetítési hibát, és a vetítő sugarak közötti átlagos távolságot.

5.1. táblázat. *Kalibrálás módszereket összehasonlító táblázat*

	visszavetítési hiba (px)	átlagos sugár távolság (cm)
kézi	15,598603	4,066854
padló	4,077808	1,324165
padló + asztal	2,282632	0,780656
kötegelt behangolás	0,705685	0,233849

A táblázatból látszik, hogy kötegelt behangolással a sugarak közötti távolság, a pontosság, átlagosan fél centiméter alatt van az aktív terület teljes terében. Így embernyi mennyiségű és alakú marker esetén is megbízhatóan, ugrálás nélkül, működik a pontok 3D-s rekonstruálása.

6. fejezet

Merev test, csontváz

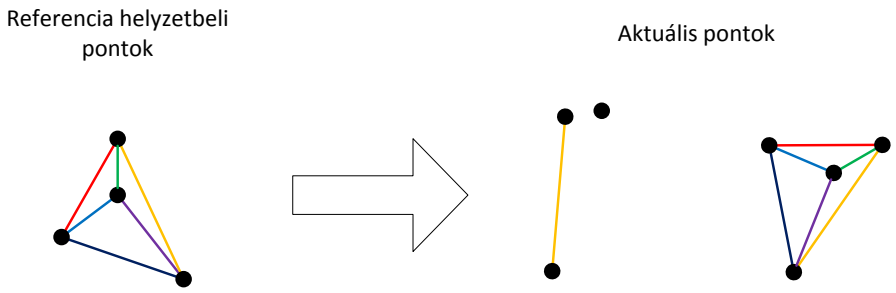
6.1. Pontokba merev test illesztés

A térben legalább 3 különböző pont meghatároz egy merev testet. Ha ismert a pontok helyzete egy referencia helyzetben, akkor meghatározható egy eltolás, és egy forgatás, amely a referencia helyzetben lévő pontokat áttranszformálja az aktuális állapotba.

6.1.1. Pontok összerendelése

A transzformáció meghatározása előtt meg kell határozni, hogy a referencia helyzetben lévő pontokhoz mely aktuális pontok rendelhetőek. Egy merev testen a pontok közötti távolság az időben nem változik, maximum a mérési hiba vihet be zajt.

Az összes új pont között meg kell határozni az élek hosszát, majd a merev test élei ezekhez az élekhez rendelhetőek a hosszuk alapján (6.1. ábra). Ezért fontos, hogy a merev test pontjai között az élek különböző hosszúságúak legyenek.



6.1. ábra. *Élek egymáshoz rendelése*

Az így kialakult gráfban az összefüggő komponenseket kell megvizsgálni, hogy az eredeti merev testtel megegyezik-e topológiailag. Ehhez az éleket be kell járni például szélességi kereséssel. Az egyértelmű pozíció, és orientáció meghatározásához legalább 2 élnek (3 pontnak) kell egyeznie. Az ilyen egyezések közül a legkisebb hibájú adja meg az egyező pontokat. A hiba a referencia helyzetbeli élek, és a hozzájuk tartozó élek közötti különbség összege.

6.1.2. Merev test illesztés

Össze lett rendelve n darab pont az aktuális helyzetben ($\{\mathbf{q_1}, \ldots, \mathbf{q_n}\}$), és a hozzájuk tartozó pontok a referencia helyzetben ($\{\mathbf{p_1}, \ldots, \mathbf{p_n}\}$). A cél meghatározni az $\mathbf{R}$ forgatás mátrixot, és a $\mathbf{d}$ eltolás vektort, amely a referencia helyzetbeli pontokat áttranszformálja az aktuális helyzetbe [8]:

$$\mathbf{q_i} = \mathbf{R}\mathbf{p_i} + \mathbf{d}$$

Azonban a mérés során hibák is előfordulhatnak, ezért a legkisebb négyzetes hibájú transzformációt keressük:

$$\min \sum_{i=1}^{n} |\mathbf{R}\mathbf{p_i} + \mathbf{d} - \mathbf{q_i}|^2$$

Algoritmus

A transzformáció meghatározásának lépései [21]:

1. A két ponthalmaz origóba tolásával ($\bar{\mathbf{p}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p_i}$, $\bar{\mathbf{q}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{q_i}$)

   $$\mathbf{A} = [\mathbf{p_1} - \bar{\mathbf{p}}, \ldots, \mathbf{p_n} - \bar{\mathbf{p}}], \qquad \mathbf{B} = [\mathbf{q_1} - \bar{\mathbf{q}}, \ldots, \mathbf{q_n} - \bar{\mathbf{q}}]$$

   a forgatás mátrix meghatározása a következőre egyszerűsödik:

   $$\min |\mathbf{R}\mathbf{A} - \mathbf{B}|$$

2. A pontokat tartalmazó mátrixot össze kell szorozni:

   $$\mathbf{C} = \mathbf{B}\mathbf{A}^T$$

3. Majd a mátrixot SVD szerint felbontani:

   $$\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{C}$$

4. Az $\mathbf{U}\mathbf{V}^T$ adja meg a forgatás mátrixot, amely zaj miatt tükrözés is lehet, amelyet korrigálni kell:

   $$\mathbf{R} = \begin{cases} \mathbf{U}\mathbf{V}^T & \text{ha } \det\left(\mathbf{U}\mathbf{V}^T\right) = 1 \\ \mathbf{U}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\mathbf{V}^T & \text{ha } \det\left(\mathbf{U}\mathbf{V}^T\right) = -1 \end{cases}$$

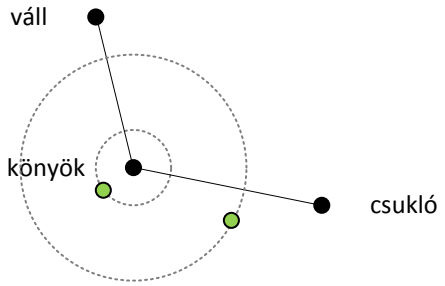
5. Végül az eltolás a következő lesz:

   $$\mathbf{d} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}$$

6.2. Csontváz

Az emberi test, csontok hierarchikus rendszerével modellezhető. A csontok hossza időben nem változik, ezt a csontváz létrehozásakor kell meghatározni. A cél a csontok szüleikhez viszonyított forgatásainak meghatározása a megfigyelt marker pozíciók alapján.

A markerek azonban nem az ízületi pontban vannak, hanem a bőr felületen. Így egy marker az ízületi pont körül egy gömbön (pl. váll), vagy egy körön (pl. könyök) mozog (6.2. ábra). A mozgás közben a markerek a bőrön elcsúszhatnak, amely zajként jelenik meg.



6.2. ábra. *Marker az ízületi pont körül mozog egy körön*

A csontváz kalibrálása során a kalibrálási mozdulatok alapján (*gym motion*) a markerek pozíciójából kell meghatározni az ízületi pont pozícióját, és ekörül a markerek sugarát. Azaz a mért pontokra egy gömböt kell illeszteni. Ez közelíthető a legkisebb négyzetes hiba alapján [20] [22], vagy meghatározható zárt formulával is [15].

A csontvázban csak a gyökér csontnak lehet eltolása, amely meghatározza a csontváz világbeli helyzetét. A gyökér csont iránya is fontos, mert ehhez képest lesznek a gyerek csontok elforgatva. Tehát a gyökér csontot legalább 3 marker határozza meg, amelyek egy merev testnek tekinthetőek. Ezután a gyerek csontok markereinek betranszformálásával meghatározható a gyerek csontok lokális iránya. Ez történhet a 6.1.2 részben leírt módszerrel. A hierarchia bejárásával ez elvégezhető minden pontra.

A csontváz és a markerek egy rugó rendszernek is tekinthetőek. A markerek és a hozzájuk rendelt csontok közé egy rugót képzelünk, amely húzza, és így forgatja a csontot a marker felé. A fizikai rendszerben a rugók az energiájuk minimalizálására törekednek, így globális minimumot kapunk eredményül [26].

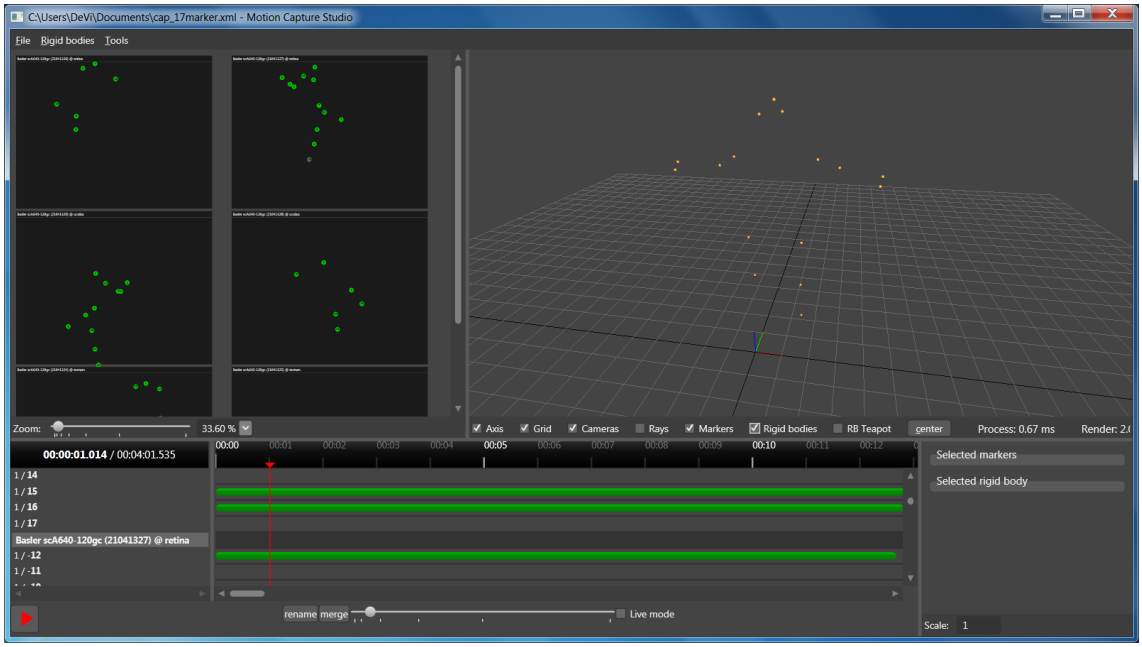
Idő hiányában a rendszer még nem kezeli a csontvázakat.

7. fejezet

Motion Capture Studio

A rendszer komponenseit (kamera kezelő modulok, foltkereső, tracking) natív C++-ban írtam. Ehhez szintén natív C++-ban készítettem egy grafikus felhasználói felülettel rendelkező programot (*MarkerTracker*). A programmal az „alacsonyabb szintű funkciókat" lehet elérni, azaz a kamerák intrinsic, és extrinsic kalibrálását (5.1.1 rész), illetve a képfeldolgozás során alkalmazandó szűrőket lehet beállítani, és a képfeldolgozás lépéseit ellenőrizni. A rekonstruált pontok 3D-ben megtekinthetőek.

Azonban a „magasabb szintű funkciók", például mozgás rögzítés, visszajátszás, merev test létrehozása, pontok azonosítása hiányzik belőle. A fejlesztés során azt tapasztaltam, hogy natív C++-ban bonyolultabb felhasználói felületet nehézkes létrehozni. Ezért született meg .NET-ben Windows Presentation Foundation (WPF) alapon a *Motion Capture Studio* (7.1. ábra).



7.1. ábra. *Motion Capture Studio*

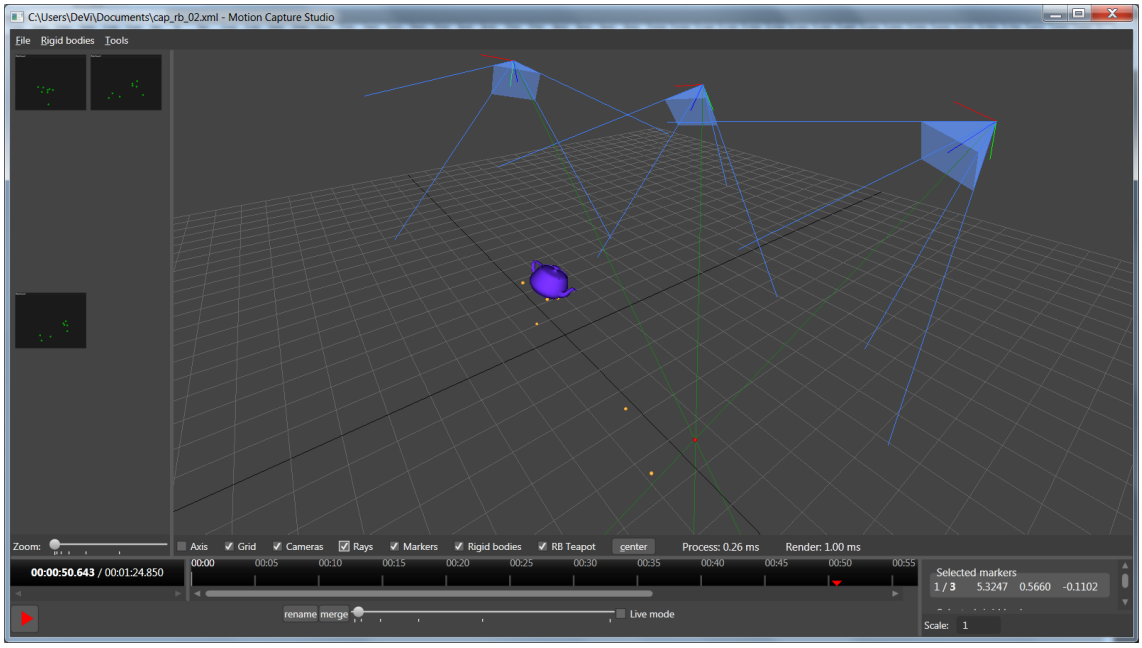
A képernyő 3 fő részre osztható:

- **2D nézet:** a kamerák képein lévő markerek láthatóak azonosítójukkal együtt,

- **3D nézet:** a rekonstruált pontok látszanak 3D-ben,
- **Idővonal:** kameránként láthatóak, hogy az egyes marker vetületek az időben mikor láthatóak.

Lehetőség van az adatok szerkesztésére, így például a hibás pont összerendelések is kijavíthatóak. Az egybefüggő 2D-s mintákat (7.1. ábrán zöld részek), szegmenseket, át lehet nevezni, szét lehet vágni, kettőt egyesíteni, és meglévőt törölni is lehet. Az időskála változtatható, azaz akár ezred másodperces részletességgel is lehet elemezni a problémákat.

A 3D nézetben legalább 3 pont kijelölése után létre lehet hozni egy merev testet, amely orientációját, és pozícióját egy 3D-s modellen lehet követni (7.2. ábra).



7.2. ábra. *Merev test*

7.1. Technikai részletek

Az új kliens programot C#-ban írtam, azonban a funkciók natív C++-ban érhetőek el. Ezért C++/CLI-ben készült egy osztály könyvtár (*TrackingNET*), amely becsomagolja a natív osztályokat.

A 3D-s rekonstrukció eredményét 3D-ben meg kell jeleníteni a felhasználónak. Azonban a WPF-ben lévő 3D megjelenítésre használható Viewport nem teljesít minden követelményt: nem lehet vonalakat rajzolni, illetve nem lehet saját vertex és pixel shader-eket használni a mesh-eken, így a későbbiekben a karakter animáció nehezen oldható meg. Ezért úgy döntöttem, hogy a .NET Framework 3.5 SP1-ben bevezetett [24] Direct3D interop-al megkerülöm a Viewport használatát. Így lehetőség van közvetlenül egy Direct3D surface-re renderelni, amelyből a WPF végzi majd a képernyőre rajzolást. Ezért készítettem C++/CLI-ben egy menedzselt környezetből elérhető minimális grafikus engine-t (*View3D*), amely Direct3D 9 (Ex) felett működik.

A WPF egyik nagy erőssége a Data Binding, amellyel a megjelenítendő adatok, és a megjelenítést végző vezérlők között lehet kötést létrehozni. A vezérlők az adatok frissülése esetén automatikusan frissítik tartalmukat. A strukturált felépítést a *Model-View-ViewModel* (MVVM) tervezési minta alkalmazása is elősegítette [23]. A WPF alkalmazásának másik kellemes tulajdonsága, hogy az alkalmazás DPI-aware lesz, így nagy pixel sűrűségű kijelzőn is olvashatóak maradnak a betűk.

Azonban a data binding naiv alkalmazásának is van hátránya. A gyakran frissülő vagy sok elemű listás adatok megjelenítése erőforrás igényes. Ilyen például a 2D-s előnézet, vagy az idő vonalon a szegmensek megjelenítése. Ezért saját vezérlőket készítettem a kritikus helyeken, amelyek mindig csak a megfelelő részeket jelenítik meg. Így elérhetővé vált nagy adatmennyiségek (például több perces 17 markeres ember felvétel) gördülékeny kezelése.

8. fejezet

Értékelés

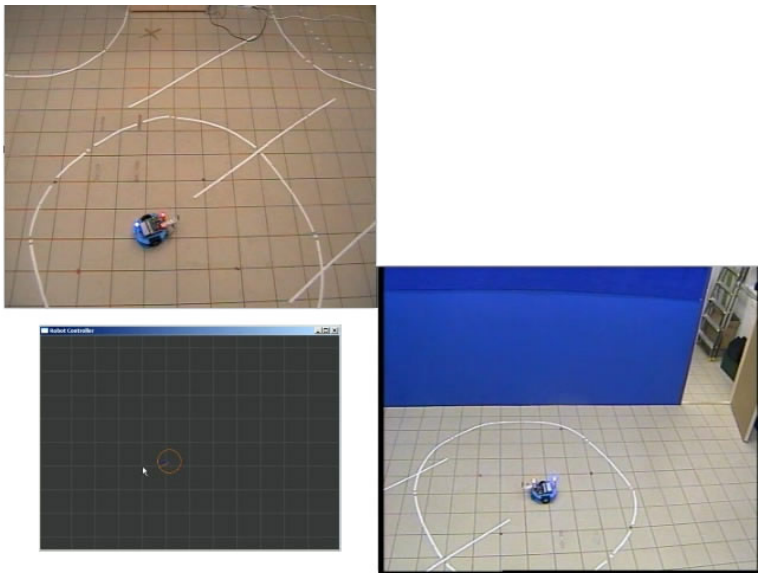
A dolgozatban bemutattam az általam készített motion capture rendszert, amely rugalmasan alkalmazható különböző környezetekben. Használható például egyetlen számítógépről, a környező bútorzatra helyezett 3 db web kamerával. Illetve használható nagy képkocka sebességű ipari kamerákkal is, több gépen, elosztottan.

8.1. Alkalmazás

Az elkészült rendszert mozgás rögzítésen kívül másra is lehet használni. Röviden bemutatok néhány ötletet, amelyeket megvalósítottam.

8.1.1. Robot mozgatás

A rendszer legelső változata arra a célra lett kifejlesztve, hogy egy LED-ekkel megjelölt robotot felül nézetből egy kattintással kijelölt helyre lehessen irányítani. A kamerákból ismert a robot pozíciója, és iránya, ez alapján Bluetooth-on át küldött parancsokkal a robotot az adott pozícióba lehet irányítani (8.1. ábra).

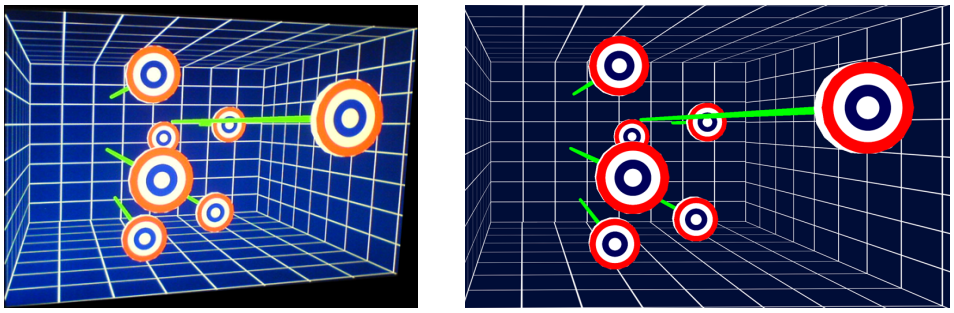


8.1. ábra. *Robot mozgatás*

8.1.2. Fej követés

Egy baseball sapkát markerekkel megjelölve (8.2. ábra) meghatározható a fej helyzete a monitorhoz képest. Ez alapján a felhasználó szerinti nézőpontból rajzolva ki a színteret a mozgás parallaxisnak köszönhetően növelhető az immerzió hatása (8.3. ábra). A rendszer képes sztereó 3D-ben működni Nvidia 3D Vision-el.

8.2. ábra. *Fej követésnél alkalmazott baseball sapka*



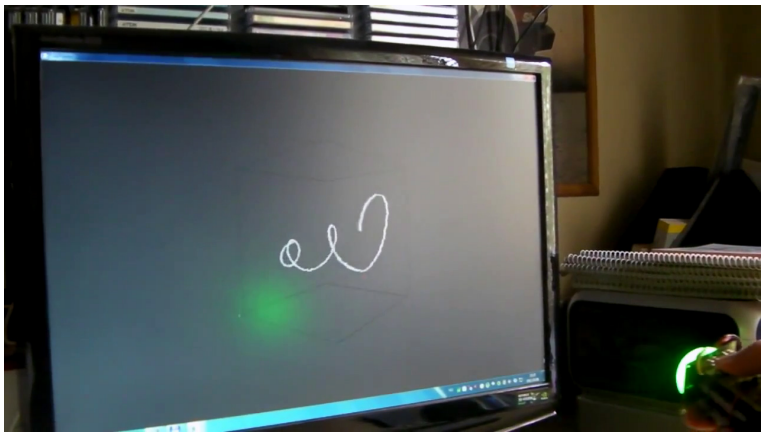
(a) Nézőpontból látható kép **(b)** A monitoron megjelenő kép

8.3. ábra. *Fej követés képe*

Johnny Chung Lee a Wii játék konzol perifériáit felhasználva valósított meg fej követést [16], amelyhez 1 kamerát (Wii Remote) használt, és 2 LED-et (Sensor Bar).

8.1.3. 3D-s rajzolás

A Rátai Dániel által kifejlesztett Leonar3Do-hoz hasonlóan [2] megvalósítottam egy egyszerű 3D-s rajzoló programot. A beviteli eszköz egy RGB LED köré helyezett fél ping-pong labda volt, amely fehér, piros, zöld, és kék színen világított különböző gombok hatására (8.4. ábra). Minden színre be volt állítva egy szűrő, és ez alapján tudta a program, hogy milyen funkciót kell ellátnia. A piros jelentette a rajzolást, a zöld a nézet forgatását, a kék egy menüt hozott elő, ahol az ecset méretet lehetett kiválasztani, vagy radír funkcióra lehetett váltani.



8.4. ábra. *CubePainter*

8.2. Továbbfejlesztési lehetőségek

A dolgozat leadásáig idő hiányában nem készült el, hogy a markerek alapján mozogjon egy csontváz. A tervezett megvalósítás egyszerűsítésekkel él. Feltételezem, hogy minden pont mindig látszik. Továbbá, hogy a bőrfelületen lévő markerek párhuzamosan állnak a belül lévő csontokkal. Így egy csont iránya közelíthető a megfelelő markerekből képzett merev test irányával.

Távlati terv a csontváz pontosabb kalibrálása, és modellezése. Illetve a felhasználótól kevésbé függő automatizáltabb működés, beleértve a hiányzó markerek kezelését, és az újra megjelenő markerek azonosítását.

A képfeldolgozás most több számítógépen történik, a 6 kamerához 3 számítógépre van szükség. A képfeldolgozás implementálható például FPGA-n, így a költségek csökkenthetőek. A képfeldolgozó hardware közvetlenül kezelheti a CCD chip-eket, vagy egy szabványos interfészen keresztül kaphatja meg a kamera képeit. Például a Basler kamerák esetében Ethernet-en keresztül.

Jelenleg folyamatosan világító LED-ek vannak a markerekben, így a szereplő akkumulátort kell viseljen, és kábelek hálózzák be. A markereket le lehet cserélni fény visszaverő anyaggal bevont gömbökre, amelyeket a kamerák köré helyezett infra LED-ek világítanak meg, így passzív markeressé alakítható a rendszer. Ehhez a rendszer szoftveres részén nem is kell módosításokat végezni.

Irodalomjegyzék

[1] *Internet Communications Engine*. URL: http://www.zeroc.org/.

[2] Leonar3do. URL: http://leonar3do.com/.

[3] Arb_pixel_buffer_object, December 2004.
URL: http://www.opengl.org/registry/specs/ARB/pixel_buffer_object.txt.

[4] October 2011. URL: http://www.naturalpoint.com/.

[5] October 2011. URL: http://www.phasespace.com/.

[6] *Camera Calibration and 3d Reconstruction*, October 2011.
URL: http://opencv.willowgarage.com/documentation/cpp/
calib3d_camera_calibration_and_3d_reconstruction.html.

[7] Motion capture, October 2011. URL: http://en.wikipedia.org/wiki/Motion_capture.

[8] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(5):698 – 700, sept. 1987.

[9] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[10] Fu Chang, Chun jen Chen, and Chi jen Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220, 2004. URL: http://www.iis.sinica.edu.tw/ fchang/paper/component_labeling_cviu.pdf.

[11] Maureen Furniss. Mocap mit - an essay on mocap from mit, October 2011. URL: http://www.motioncapturesociety.com/resources/articles/miscellaneous-articles/91-mocap-mit-an-essay-on-mocap-from-mit.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Programtervezési minták*. Addison-Wesley, 1995.

[13] Lorna Herda, Pascal Fua, Ralf Plaenkers, Ronan Boulic, and Daniel Thalmann. Using skeleton-based tracking to increase the reliability of optical motion capture, 2001.

[14] Midori Kitagawa and Brian Windsor. *MoCap for Artists - Workflow and Techniques for Motion Capture*. Elsevier Inc, 2008.

[15] Jonathan Kipling Knight. *Rotation points from motion capture data using a closed form solution*. PhD thesis, Colorado Springs, CO, USA, 2008. Adviser-Semwal, Sudhanshu Kumar.

[16] Johnny Chung Lee. URL: http://johnnylee.net/projects/wii/.

[17] Hajder Levente. Kötegelt behangolás (bundle adjustment), December 2007. URL: http://vision.sztaki.hu/ hajder/rek/jegyzet/kotegelt_behangolas/ba.pdf.

[18] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[19] Johan Nilsson. Implement a continuously updating, high-resolution time provider for windows. *MSDN Magazine*, March 2004. URL: http://msdn.microsoft.com/en-us/magazine/cc163996.aspx.

[20] James F. O'Brien, Robert E. Bodenheimer, Jr., Jr. Gabriel, Gabriel J. Brostow, and Jessica K. Hodgins. Automatic joint parameter estimation from magnetic motion capture data, 2000.

[21] Inge Söderkvist. Using svd for some fitting problems. URL: http://www.ltu.se/cms_fs/1.51590!/svd-fitting.pdf.

[22] Marius-Calin Silaghi, Ralf Plänkers, Marius c Lin Silaghi, Ronan Boulic, Pascal Fua, and Daniel Thalmann. Local and global skeleton fitting techniques for optical motion capture. In *In Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, pages 26–40. Springer, 1998.

[23] Josh Smith. Wpf apps with the model-view-viewmodel design pattern. *MSDN Magazine*, February 2009. URL: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx.

[24] Tim Sneath. Introducing the third major release of windows presentation foundation, May 2008. URL: http://blogs.msdn.com/b/tims/archive/2008/05/12/introducing-the-third-major-release-of-windows-presentation-foundation.aspx.

[25] J. Trein, A. T. Schwarzbacher, B. Hoppe, K. Noffz, and T. Trenschel. Development of a FPGA based real-time blob analysis circuit. In *Irish Systems and Signals Conference*, pages 121–126, Derry, N. Ireland, September 2007. URL: http://www.electronics.dit.ie/postgrads/jtrein/ISSC2007Blob.pdf.

[26] Victor Brian Zordan and Nicholas C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 245–250, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.