BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

Villamosmérnöki és Informatikai Kar

Irányítástechnika és Informatika tanszék

Tapintható Ember-Gép Interfész Kialakítása Valós és Virtuális Objektumok Illesztésével

Készítette: Szemenyei Márton
Villamosmérnök szakos hallgató

Konzulens: Dr. Vajda Ferenc
Docens

Tudományos Diákköri Konferencia

Budapest, 2014

Összefoglaló

Az informatika története során számos forradalom kezdődött egy új felhasználói felület létrehozásával. Példaként említhetjük a személyi számítógépek megjelenését a hétköznapi életben, ami a grafikus felhasználói felület kereskedelmi forgalomban való megjelenéséhez köthető, vagy az érintőképernyős „okos” mobil eszközök rohamos elterjedését. Ma is széleskörű kutatás folyik új kezelőfelületek terén, melynek talán a legjelentősebb irányzata a virtuális és kiterjesztett valóság rendszerek fejlesztése. Ezen rendszerek egy csoportja a tapintható kiterjesztett valóság (TAR – Tangible Augmented Reality), melyek a virtuális objektumok irányítását a valós tárgyak felhasználásával oldják meg, így biztosítva a virtuális tárgyak természetes módon történő manipulálásának lehetőségét.

Használhatóság szempontjából a TAR rendszerek egyik legnagyobb hátránya, hogy a felhasznált valós objektumok előzetes ismerete szükséges a fejlesztésükhöz. Ebből következően a felhasználónak a használat előtt a saját munkaterét be kell rendeznie, hogy a rendszer a felhasználandó tárgyakat felismerje. Ez a tulajdonság a rendszer hétköznapi használhatóságát rendkívül nehézkessé teszi.

A TDK dolgozatom témája egy olyan intelligens TAR rendszer elkészítése, amely képes egy előre ismeretlen teret „berendezni” virtuális objektumokkal alakbeli hasonlóság felhasználásával. Így a rendszer képes egy olyan környezetet létrehozni, melyben a virtuális objektumokat természetes módon manipulálhatjuk, mégsem szükséges ezeket speciális markerekkel ellátni. A használathoz elegendő, ha a felhasználó a környezetéből kiválaszt olyan tárgyakat, amik az adott feladat elvégzéséhez hasznosak.

A dolgozat központi témája az alakfelismerés, melyet egy tanuló algoritmus segítségével implementálok. A párosított valós tárgyakon ezután hat szabadságfokú követést végzek, így biztosítva, hogy a felhasználók a virtuális objektumokat valós tárgyak segítségével manipulálhassanak.

Abstract

In the history of IT several revolutions started with the development of a new user interface. The appearance of personal computers in everyday life, which can be attributed to the introduction of graphical user interfaces, or the rapid conquest of touchscreen-based “smart” mobile devices are great examples. Even today, there is extensive research in the area of new user interfaces, virtual and augmented reality systems being one of the most significant directions. One subgroup of these systems called Tangible Augmented Reality (TAR) systems allow users to control virtual objects with the help of physical objects, thus providing a natural method of interaction.

When it comes to usability, the greatest disadvantage of TAR systems is that knowledge about the real objects used is required for their development. Therefore users need to set up their own workspace before usage in order to enable the recognition of the physical objects to be utilized. This property is a significant barrier in the way of simple everyday usage.

In this essay, I will propose an intelligent TAR system that attempts to place virtual objects into a previously unknown scene using shape similarity. This way the system can create an environment that allows nature manipulation of virtual objects without having to equip them with special markers. Users will only have to choose a set of objects from their environment that are useful for the given task.

The central theme of my work is shape recognition, which I will implement using a learning algorithm. Then, I use 6-DOF tracking on real object in order to ensure that users could interact with virtual objects by manipulating physical ones.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani konzulensemnek, Dr. Vajda Ferencnek, aki a munkám során szakértelmével, tapasztalataival és hasznos tanácsaival a kutatás eddigi eredményeihez és a dolgozatom elkészüléséhez hatalmas segítséget nyújtott, valamint inspirációt nyújtott a tudományos kutatómunka végzéséhez.

Tartalomjegyzék

Ábrajegyzék

Táblázatjegyzék

1. Bevezetés

A számítástechnika rohamos fejlődésével együtt számos, a hagyományostól eltérő módszer fejlődött ki a számítógépekkel történő kommunikáció megoldására. Ezen megoldások jelentős része kapcsolható Kiterjesztett Valóság (Augmented Reality; AR) rendszerekhez. A Kiterjesztett Valóság rendszerek körében a legfontosabb kutatási témák a megjelenítés eszközei, a virtuális objektumok minél realisztikusabb előállítása, valamint az interakciós technikák. Ez utóbbi területen egy lehetséges megoldás a Tapintható Kiterjesztett Valóság (Tangible Augmented Reality; TAR).

A TAR rendszerek a Tapintható Felhasználói Felület (Tangible User Interface; TUI) Kiterjesztett Valóság rendszerekbe való átültetése. A tapintható felületek alapvető működési elve, hogy egy valós környezetbe, fizikai tárgyak helyére virtuális objektumokat helyez egy megjelenítő szemüveg segítségével, és a virtuális objektumok kezelését a valós tárgyakhoz köti, így valósítva meg a felhasználó és a környezet közti interakciót. [1] A tapintható felhasználói felületek esetében gyakori megoldás, hogy néhány, a rendszer tervezője által meghatározott kinézetű valós objektumot használnak fel a virtuális objektumok manipulálására. Ezeket a valós objektumokat a legtöbb esetben különféle markerekkel látják el, amely használat előtti preparációt igényel, jelentősen megnehezítve a rendszer hétköznapi használatát. További hátránya ennek a megoldásnak, hogy az egymáshoz tartozó virtuális és valós objektumok a felhasználó számára logikus manipulálási módja különbözhet, ami idegrendszeri konfliktushoz vezet, és a felhasználói élményt jelentősen csökkenti.

A dolgozatom témája egy olyan tapintható kiterjesztett valóság rendszer fejlesztése, amely igyekszik ezt a komoly hátrányt áthidalni intelligens objektumpárosítás segítségével. Ez az adaptív kiterjesztett valóság rendszer képes egy előre ismeretlen környezetet feltérképezni, majd a környezetben található valós objektumokhoz virtuálisakat rendelni úgy, hogy az összepárosított objektumok alakja hasonlítson. Ezzel a megoldással nem szükséges többé a használt valós tér előkészítése, így a rendszer hétköznapi körülmények között is használható, az egyetlen használat előtt elvégzendő feladat a környezet feltérképezése, amelyet a felhasználó a megjelenítő szemüvegeken megtalálható kamera segítségével könnyedén elvégezhet. A megoldás további

előnye, hogy az alakbeli hasonlóság miatt a felhasználók magától értetődő módón képesek a virtuális objektumokat kezelni.

A dolgozatom során az elképzelt adaptív AR rendszerhez legfontosabb feladatok megvalósítását és vizsgálatát végeztem el, egy prototípusrendszer létrehozásával. A rendszer által elvégzendő feladatok közül a legfontosabb és a legkomplexebb az intelligens párosítás volt, melynek fejlesztése a dolgozat központi témája. Alapvetően ez a feladat öt részfeladatra bontható:

- A környezetről készült felvételek alapján történő 3D rekonstrukció
- Az elkészült 3 dimenziós térben található objektumok szegmentációja
- Az objektumok formájának leírása
- Az ez alapján történő párosítás
- A detektált objektumok lokalizációja (skála, helyzet, orientáció)

A szegmentációs lépés során a 3 dimenziós jelenetben primitív alakzatokat keresek, majd az alakzat leírásához ezekből egy gráfot készítek. A gráf csomópontjai maguk a primitív formák, míg az élek a primitív formák térbeli kapcsolatát írják le. A párosítás és a lokalizáció implementálásához pedig egy SVM alapú tanuló algoritmust használok.

A második lényeges feladat a hat szabadságfokú, valósidejű követés megvalósítása. Ez a feladat elengedhetetlen ahhoz, hogy a valós objektumok változásait a rendszer észlelni tudja, és így az interakciót megvalósíthassa. Az objektumok detektálása után egy adatbázist építek, amely az objektumok követéshez és detektáláshoz szükséges tulajdonságait tárolja, majd ez alapján képkockáról képkockára követni kell. Rendkívül fontos, hogy ha az objektum követés közben változik (például elfordul), akkor az első lépésben épített adatbázist frissíteni kell. Szintén lényeges, hogy a követett objektumok esetleges elvesztését észlelni kell, és ekkor az objektumot újra detektálni kell a képen.

A dolgozat soron következő részében egy részletes áttekintést kínálok más műhelyek a témához kapcsolódó eredményeiről, kitérve a tapintható kiterjesztett valóság rendszerekre, az alakfelismerésre, a tanuló rendszerekre, illetve a valósidejű háromdimenziós követésre is. Ezután a harmadik fejezetben részletezem a fejlesztés során meghozott fontos döntéseket, ezt követően pedig a tesztelés eljárását és az e során szerzett tapasztalatokat és eredményeket mutatom be. A

dolgozat végén egy rövid összefoglalást és értékelést adok, valamint részletezem a továbbfejlesztési terveimet is.

2. Más műhelyek eredményei

Ebben a fejezetben a feladatok megoldásához szükséges szakterületekhez kapcsolódó irodalom áttekintését végzem el. Először a virtuális és a kiterjesztett valóság rendszerekhez kapcsolódó kutatásokkal foglalkozom, különösképpen a tapintható/megfogható kiterjesztett valóság rendszerek és felhasználói felületek területével. Ezt követően a gráfokon értelmezett gépi tanuló algoritmusok területén történő kutatásokat mutatom be, majd pedig az alakzatok leírásának és felismerésének módszereit tekintem át. Befejezésképpen az objektumok követésének néhány módszerét tárgyalom.

2.1.Kiterjesztett Valóság

A kiterjesztett valóság rendszerekhez számos különböző beviteli módszert fejlesztettek ki az elmúlt két évtizedben. Gyakoriak a gesztusvezérelt alkalmazások [2], vagy a különböző egyszerű beviteli eszközöket alkalmazó megoldások, például egy LED-ekkel ellátott toll [3], az okostelefonok elterjedésével pedig a mobil kiterjesztett alkalmazások [4] kezdtek rohamosan fejlődni. Szintén érdemes megemlíteni olyan eseteket, melyben egy beviteli eszköz segítségével a környezetben elhelyezkedő valós objektumokat lehet manipulálni. Léteznek megoldások, amelyben lehetőség nyílik valós objektumok felületére történő rajzolásra [5], vagy pedig virtuális textúrák a felületre történő felvitelére. [6] A valós felületekre történő rajzolás már az iparban is megjelent, amire jó példa a Microsoft által fejlesztett IllumiRoom rendszer, amely egy projektor segítségével képes a televízió képét kiegészíteni az adott szoba falát használva vetítővászon gyanánt. [7]

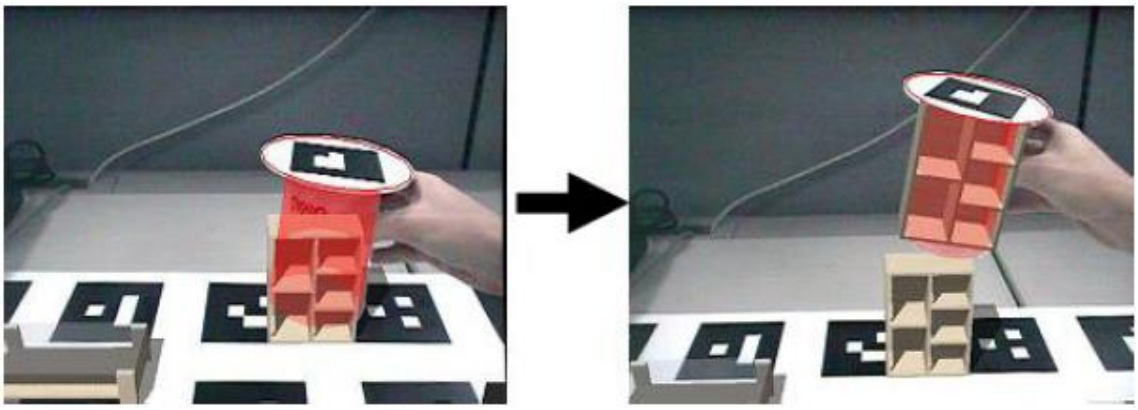
A kiterjesztett valóság kutatás egyik fő területe a tapintható felhasználói felületek alkalmazásának vizsgálata. A digitális tartalom fizikai objektumokkal történő kezelése egy rendkívül könnyen megtanulható kezelési módszert nyújt a felhasználók számára. [8] A Tapintható Kiterjesztett Valóság felületek főbb jellemzőit így foglalhatjuk össze [9]:

- A virtuális objektumok fizikai objektumokkal történő manipulációja
- Térbeli interakciós technikák használata (pl.: távolság, elforgatás)
- Több kéz használatának a támogatása
- Olyan valós tárgyak választása, melyek fizikai korlátai összeegyeztethetők az elvégzendő feladattal

- Párhuzamos aktivitás támogatása több objektummal
- Több résztvevő együttműködésének a támogatása

A TAR interfészek fő előnyei a könnyű kezelhetőség, és a természetes manipuláció lehetősége. Hátrányuk viszont, hogy nehéz TAR módszerekkel érintést, nyomást tartalmazó felhasználói felület elemek (pl.: nyomógomb) megvalósítása [10], valamint, hogy ha a követendő objektumon nincs elég könnyen követhető rész, akkor nehéz meghatározni az alakját és a helyzetét. [8] Gyakori, hogy a TAR felületeket nem önmagukban, hanem más kiegészítő beviteli módszerekkel együtt használják, melyeket hibrid felhasználói felületnek nevezzük. Ilyen megoldások segítségével ki lehet egészíteni egy TAR felületet olyan műveletekkel, amiket a TAR módszerekkel nehéz megoldani. [8]

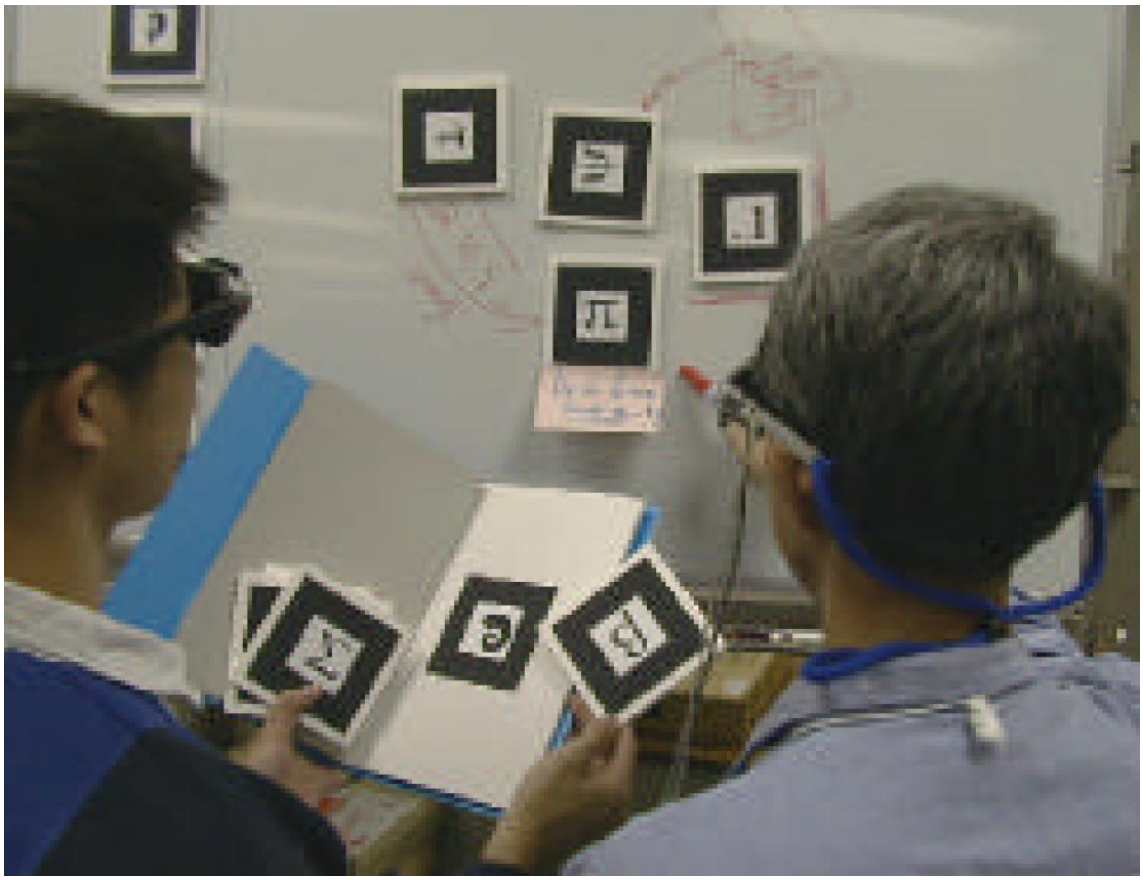
Az egyik legjobb példa az TAR interfészekre a MagicCup. [9] A MagicCup (*2.1 Ábra*) tapintható vezérlője egy pohár alakú tárgy, amin hat szabadságfokú követést végeznek. A pohár képes virtuálisan tartalmazni a környezet objektumait, aminek segítségével kiválasztható az éppen manipulálandó virtuális tárgy. Ahhoz, hogy beleillesszünk egy virtuális objektumot, el kell fednünk a pohárral. Ezek után azt az objektumot át tudjuk helyezni egy másik helyre más orientációval. A rendszer képes még más műveletek érzékelésére is, ilyen a pohár rázása, csúsztatása, forgatása, stb.



2.1 Ábra: A MagicCup felület. Forrás: [9]

Egy további említésre méltó megoldás a Tiles felületet [11], ami mágneses kártyákra nyomtatott markereket használ, melyeket egy fém táblára lehet felhelyezni. (*2.2 Ábra*) Ezt a felületet már komplexebb műveletek elvégzésére is használják (pl.: másolás, törlés). A működés

során felhasználható virtuális objektumok egy könyvben találhatóak (a TILES esetében ezek az objektumok repülőgépek műszerei). Az előbb említett kártyák különböző feladatot látnak el: Vannak adat kártyák, amelyek az objektumok tárolására szolgálnak, illetve operátor kártyák, melyek a különböző műveletek elvégzésében segítenek. A kártyák közti interakciót a kártyák távolsága alapján lehet elvégezni (egymásra helyezés).



2.2 Ábra: A Tiles felület. Forrás: [11]

A saját feladatom szempontjából a legrelevánsabb TAR megoldás a VRT (Virtual Round Table). [12] A rendszer elérni kívánt célja, hogy egy asztal körül ülő felhasználók könnyedén, valós tárgyak segítségével hozhassanak létre egy mesterséges környezetet. A felhasználók HMD-t viselnek, amely az asztalon lévő valós objektumok helyére a felhasználó által választott virtuális objektumot vetíti. (*2.3 Ábra*) A rendszer fontos összetevője, hogy nem ismerjük előre az asztalon lévő tárgyakat, ami a saját rendszerem egyik legfontosabb összetevője. A legfőbb különbség a két rendszer közt, hogy a VRT fejlesztőinek nem volt célja a valós és a virtuális objektumok hasonlóság alapján történő illesztése, itt a párosítás elvégzése a felhasználó dolga.

2.3 Ábra: A Virtual Roud Table felület. Forrás: [12]

2.2. Alakfelismerés

Egyes tárgyak alak alapján történő felismerése a gépi látás szakterületének egy fontos területe, amely a kiterjesztett valóság kutatásában külön jelentőséggel bír. Azonban a legtöbb létező módszer kétdimenziós alakzatok felismerésére lett kifejlesztve, mivel a leggyakrabban az alakfelismerést a képsíkon végzik el. Ezen algoritmusok többsége nem terjeszthető ki egyszerűen három- vagy többdimenziós esetre. [13]

Ennek ellenére számos módszert találhatunk a háromdimenziós esetre is, melyek többsége a háromdimenziós tér pontfelhős reprezentációjának segítségével működik. Ezek közül az egyik az alakzat eloszlásra épülő párosítás. [13] Ez az eljárás poligonos alakzatok párosítását eloszlások összehasonlításával igyekszik megoldani. Az algoritmus első lépése egy alakfüggvény létrehozása, melyről feltesszük, hogy kellőképp leírja az adott formát. A következő lépésben az algoritmus az alakfüggvény mintavételezésével egy eloszlást (az alakzat eloszlást) készít. Ilyen eloszlásra jó példa az alakzatból véletlenszerűen kiválasztott pontpárok euklideszi távolságának eloszlása.

Egy másik irányzat az alakjellemzők egy csoportjának létrehozására alapul, amivel leírhatjuk az adott formát. Erre rendkívül jó példa a városi alakzatfelismerés. [14] Itt először egy szegmentációs lépés során objektumjelölteket állítanak elő, majd meghatároznak minden jelölthöz

néhány alakjellemzőt. Ezen jellemzők egy része az alakzatot írta le (pl.: a pontok száma, becsült térfogat, átlagos földtől számított magasság), egy másik része pedig a kontextust, amiben az objektumjelölt elhelyezkedett. A kontextus jellemzők ugyan nem tartoznak az alakfelismerés körébe, mégis fontos részei lehetnek egy felismerő algoritmusnak, hiszen képesek az algoritmus hatékonyságát nagymértékben javítani.

A fenti módszerek legnagyobb hátránya, hogy egy külön szegmentáló lépés szükséges a felismerendő objektumoknak a környezetüktől való elválasztásához. Ez problémát okoz, ugyanis a szegmentálás általában rendkívül önkényes (a szegmentáció helyes megoldása ugyanis nem egyértelmű) és nagy számításigényű lépés. Ez a probléma elkerülhető, ha egy külön szegmentáló algoritmus alkalmazása helyett a szegmentációt maga a felismerő algoritmus végzi el. Ekkor az alakfelismerő algoritmus egy egész jelenetben keres olyan részeket, amelyek az általa keresett formának megjelennek (alakfelirmerésre alapuló szegmentáció).

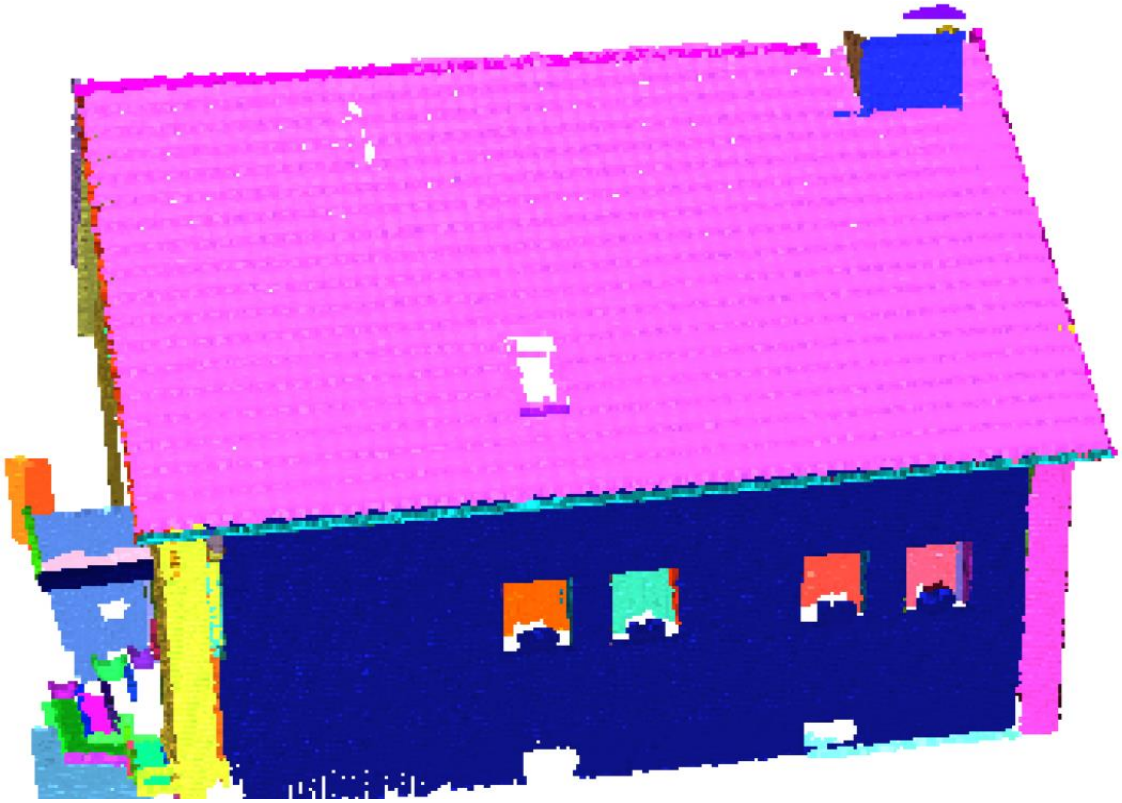
Az alak alapú szegmentáció egyik legismertebb módja a Hough-transzformáció, mely a különböző alakzatokat a szabadon megválasztható paramétereik által kifeszített térben ír le. A transzformáció a pontfelhőben lévő pontokat először a paramétertérbe transzformálja, ahol minden ponthoz valamilyen görbe/felület fog tartozni, amely tartalmazza az összes olyan keresett formát, amelyhez az adott pont tartozhat. Ahol sok görbe vagy felület metszi egymást, nagy valószínűséggel a keresett alakzat egy példánya található, így a pontok szavaztatásával ezeket az algoritmus megkeresi. A Hough-transzformációt leggyakrabban 2D egyenesek megtalálására használják, de kiterjeszthető más alakzatokra is. Korlátozó tényező viszont, hogy minél bonyolultabb a keresett alakzat (minél több paraméterrel írható csak le), a módszer annál rosszabb teljesítményű és nagyobb erőforrás igényű lesz. [15]

Egy másik módja különböző formák pontfelhőben történő keresésének a RANSAC (Random Sample Consensus) [16] algoritmus. A RANSAC algoritmus futása során véletlenszerűen kiválaszt annyi pontot a pontfelhőből, amennyi minimálisan szükséges a keresett alakzat paramétereinek kiszámolásához. Ez után az algoritmus kiszámolja, hogy hány pont illik rá az adott jelölt alakzatra a pontfelhő megmaradt részéből. Az algoritmus ezek után ezt az alapműveletet nagyon sok véletlenszerűen kiválasztott ponthalmazra elvégzi, minden jelölthöz egy pontszámot hozzárendelve. Az algoritmus végeredménye a legtöbb illeszkedő pontot tartalmazó, vagyis a legvalószínűbb alakzat. Az algoritmusnak számos előnye van [17] melyek közül az egyszerűségét,

illetve a robusztusságát emelném ki. Az algoritmus elvileg akármekkora mennyiségű zaj esetén is képes megtalálni az alakzatot, míg a legtöbb hasonló módszer 50%-nál nagyobb zajarány esetén csődöt mond.

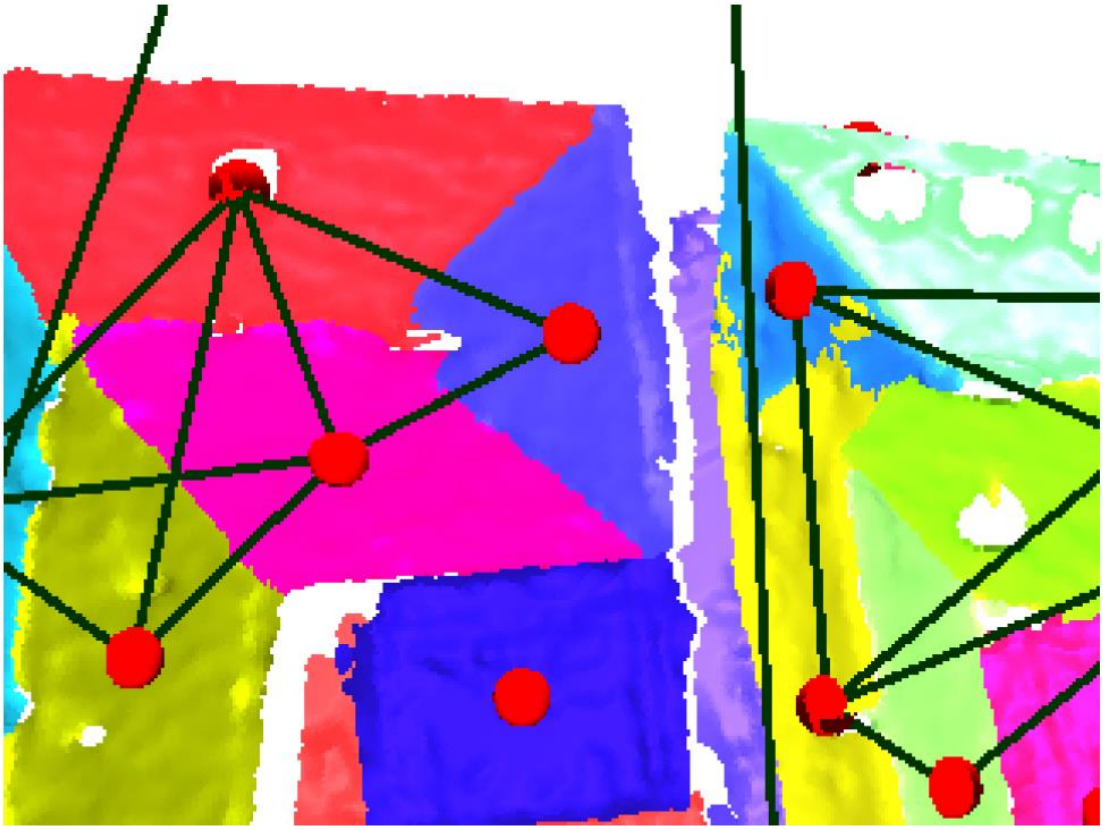
Schnabel, Wahl és Klein [17] sikerrel alkalmazták a RANSAC algoritmust arra, hogy egy 3D pontfelhőben egyszerre többfajta primitívet is detektáljanak, ráadásul egyszerre több példány detektálására is lehetőség van. (*2.4 Ábra*) Az eljárásukhoz a RANSAC algoritmust módosították, többek között úgy, hogy nem véletlenszerű, hanem lokális mintavételezést használnak. Ennek oka az, hogy az alakzatok lokális jelenségek, így pontok lokális kiválasztásával lényegesen nagyobb eséllyel választanak ki egyazon alakzathoz tartozó pontokat. Az algoritmusuk a következő:

- Minden egyes iterációban új jelöltek keresése lokális mintavételezéssel
- A jelöltek közül a legjobb kiválasztása
- A legjobb alakzathoz tartozó pontok törlése felhőből
- A törölt pontok által érvénytelenített jelöltek elvetése
- Az iteráció folytatása, amíg van elfogadható jelölt



2.4 Ábra: Egy ház lebontva primitív formákra. Forrás: [17]

Az alakzat leírásához Schnabel et. al. felhasználták ennek az algoritmusnak a végeredményét. A RANSAC algoritmus segítségével meghatározott primitív formák közötti érintkezéseket kihasználva létrehoztak egy topológiai gráfot [18], mely leírja, hogy az adott primitív alakzatok hogyan kapcsolódnak egymáshoz. (*2.5 Ábra*) A gráf pontjai az alakzatok lesznek, míg az élek reprezentálják az alakzatok kapcsolatát. Ha a térben két primitív forma távolsága egy küszöbérték alá esik, akkor a két csomópont között létrehoztak egy élet, ellenkező esetben az az él nem fog szerepelni a gráfban.



2.5 Ábra: A konstruált topológiai gráf. Forrás: [18]

Ahhoz, hogy az objektumokról készített gráfokat párosításra használhassák, a referencia objektumokhoz is elő kell állítaniuk az alakot leíró topológiai gráfot. Ezután a konkrét párosítást algráf illesztéssel végzik el. Bár az algráf illesztés NP teljes probléma, ez a gráfok relatíve kis mérete miatt nem okoz problémát. Fontos megjegyezni, hogy a kizárólag topológiai alapú leírás nem elégséges a megfelelően jó megkülönböztető képességgel rendelkező leíráshoz. Egy téglatest gráfja például három összekötött pontból áll, de ez a gráf annyira egyszerű, hogy nagy valószínűséggel lesz más olyan alakzat is, amelyik ezt a gráfot adja. [18]

Ennek a problémának a megoldásához a gráfhoz különböző megkötéseket, kiegészítéseket készítettek. Ezeket három típusba sorolták: csomóponti, él, és gráf típusú megkötések. Csomóponti típusú megkötés lehet például az alakzat típusa, mérete, él típusú megkötés a két alakzat egymással bezárt szöge, gráf típusú megkötés pedig például két közvetlenül nem kapcsolt sík párhuzamossága. A csomóponti és az él megkötések jelentősen meggyorsítják az algráf párosító algoritmust, valamint a gráf megkötésekkel együtt megoldják tisztán topológiai reprezentáció elégtelenségét. [18]

2.3.Tanuló algoritmusok

Ebben a fejezetben áttekintem a tanuló gráf párosításra használatos algoritmusok irodalmát. Először a vektoriális gráfpárosító módszereket mutatom be, valamint a gráfok spektrális reprezentációját. A második részben a kernel módszerekről és a véletlen séta (random walk) kernelről számolok be.

A vektoriális módszerek elég gyakori megoldások a tanuló gráfpárosítás területén. Mivel a legtöbb gépi tanuló algoritmus vektortereken definiált tanulási problémákkal működik, ezért adja magát az ötlet, hogy valamilyen eljárás segítségével egy vektort konstruáljunk a gráfból, és ezt felhasználjuk, mint jellemző vektort az algoritmushoz. Ennek a legegyszerűbb módja, ha valós csomóponti és él súlyokkal ellátott gráf esetén a csomópontok súlyaiból vektort készítünk, majd ezt a vektort kiegészítjük a gráf adjacencia mátrixának egymás alá helyezett oszlopaival. A módszerrel azonban két súlyos probléma akad. Először is, különböző méretű gráfok összehasonlításakor különböző méretű vektorokkal kell számolni. Ez a probléma önmagában még orvosolható, mivel a kisebbik vektort nullákkal könnyedén kiegészíthetjük. A második probléma az, hogy a gráf egy strukturált objektum, így a csomópontok sorszámozása (ami szükséges a vektor megkonstruálásához) önkényes. A csomópontok sorrendjének megváltoztatásával egy teljesen más jellemző vektort kapunk ugyanahhoz a gráfhoz. Ennek a problémának a megoldásához valamilyen gráf igazító algoritmust kellene használni, mint például [19].

Egy másik, részleges megoldás erre a gráfok spektrális reprezentációjának használata [20], mely a mátrixok sajátfelbontásán alapszik. A módszer alapelve, hogy a gráf mátrixos reprezentációjának a sajátértékeit és sajátvektorait kiszámoljuk, majd a sajátértékeket és a hozzájuk tartozó vektorokat nagyság szerint sorba rendezzük. A jellemző vektor ezután a sajátértékek és a hozzájuk tartozó sajátvektorok egymás alá helyezésével (konkatenálásával)

keletkezik. A módszer előnye, hogy részleges invarianciát biztosít a csomópontok számozására, de egy igazító lépés még mindig szükséges. [20]

A gráfok spektrális reprezentációjának több variációja is van, melyek a felhasznált mátrixos reprezentációban különböznek. A legegyszerűbb mód az adjacencia mátrix használata *(A)*. Egy komplexebb megoldás a Laplace-mátrix használata, amelyet a gráf foksági *(D)* és adjacencia mátrixából lehet konstruálni. (1)

$$L = D - A \tag{1}$$

Egy még komplexebb megoldás a hőkernel *(H)* használata, amely a Laplace-mátrix sajátfelbontását $(L = VEV^T)$ használja. A hőkernel azt adja meg, hogy a hő milyen módon terjedne a gráfon, amennyiben az valamilyen valós struktúrát írna le [20]. A kernelhez tartozik egy idő paraméter *(t)*, amellyel azt lehet megadni, hogy a kernel a globális vagy a lokális struktúrát írja le nagyobb súllyal. Zhu és Wilson [21] szerint a három módszer közül a hőkernel adja a legjobb eredményt. A kiszámítása a következő:

$$H = Ve^{-tE}V^T \tag{2}$$

A vektoriális gráfpárosítás kitűnő példája White [20] módszere, aki egy generatív eljárást fejlesztett ki. Az algoritmus többfajta spektrális reprezentáció együttes felhasználásával konstruálja meg a gráfhoz tartozó jellemző vektort. A modell generatív jellegének biztosítására egy medián gráf számoló módszert [19] is alkalmaztak. Ezen felül kifejlesztettek egy részgráf-alapú generatív modellt is, amelyhez egy gráf szegmentáló algoritmust alkalmaztak.

A gráfpárosító tanuló algoritmusok egy másik nagy csoportja a kernel módszerekre alapul. A kernel módszerek nagy népszerűségnek örvendenek a gépi tanulás területén, nem csupán gráfok esetén. Ennek a népszerűségnek oka a „kernel trükk”, amely lehetővé teszi nemlineáris függvények illetve szeparáló felületek megtanulását elhanyagolható számításigény mellett egy kernel függvény segítségével. A kernel függvényeket leggyakrabban Support Vector Machine (SVM) algoritmusba építik bele, amely egy olyan tanuló algoritmus, melyet egyáltalán nem kell módosítani ahhoz, hogy kernel függvényekkel működjön.

Először definiáljuk a kernel függvényt: A kernel függvény *K(A,B)* alapvetően az *A* és *B* objektumok közti „hasonlóság” mértéke, ahol *A* és *B* általában vektorok, de ez nem megkötés. A

kernel függvénynek két matematikai megkötést is teljesíteni kell: Egyrészt pozitív szemi-definitnek és szimmetrikusnak[1] kell lennie, akkor bizonyítható, hogy létezik olyan $X = \Phi(A)$ és $Y=\Phi(B)$ vektor, hogy $K(A,B) = <X,Y>$. A hagyományos alkalmazások során egy jól megválasztott kernel függvény segítségével kiszámolhatjuk a skaláris szorzatát két magasabb dimenziós vektornak (X és Y), melyeket A-ból és B-ből lehet konstruálni anélkül, hogy X-et és Y-t explicit módon reprezentálnunk kéne. A gráfpárosítás során a kernel trükk lehetőséget nyújt két gráf hasonlóságának kiszámítására anélkül, hogy explicit módon vektort kéne konstruálnunk belőlük.

Az egyik népszerű gráfokon értelmezett kernel függvény a véletlen séta (random walk) kernel. Legyen adott két gráf valós súlyokkal a csomópontjain és az élein. A csomópontok súlyait értelmezzük úgy, mint annak a valószínűségét, hogy azon a csomóponton kezdünk, vagy fejezünk be egy véletlenszerű sétát, míg az élek súlyait az adott élen való továbbhaladásnak a valószínűségeként értelmezzük. Ezt követően képzeljük el, hogy a két összehasonlítandó gráfon szimultán módon sétálunk, és szeretnénk, ha a két séta *ugyanolyan* lenne. Egy adott séta megtételének a valószínűsége kiszámítható bármelyik gráfra, az adott séta mindkét gráfon való együttes megtételének a valószínűsége ebből származtatható. Ha a két gráf hasonló, akkor ugyanannak a sétának a valószínűsége magas lesz, ha különbözőek, akkor pedig alacsony. [22]

A magyarázat után már bemutathatom az algoritmus matematikai képletét. Legyen $X = \{Ax, Nx\}$ és $Y = \{Az, Ny\}$ a két gráf, melyeket a csomóponti vektorokkal és az adjacencia mátrixokkal adjunk meg. A séták együttes megtételének reprezentációjához ki kell számolnunk a két gráf direkt szorzatát. (3)

$$XY = \{Axy, Nxy\}: \qquad Axy = Ax \otimes Ay \qquad Nxy = Nx \otimes Ny \tag{3}$$

$$A \otimes B = \begin{bmatrix} a_{11} * B & \cdots & a_{1n} * B \\ \vdots & \ddots & \vdots \\ a_{m1} * B & \cdots & a_{mn} * B \end{bmatrix}$$

Ahol $\otimes$ két mátrix Korenecker szorzatát jelenti. Ahhoz, hogy az egyszerre történő séták valószínűségét az eljárás kiszámolja, igazából a direkt szorzat gráfon való i hosszú séták valószínűségeit kell összegezni (i futó változó). Fontos megjegyezni, hogy ennek kiszámolásához

[1] $K(A,B) = K(B,A)$

értelmeznünk kell az séta abbahagyásának valószínűségét is az adott csomóponton, amit a kezdés valószínűségével megegyezőnek választható. A fenti megfontolások alapján kernel végeredménye az alábbi: [22]

$$K = \sum_{i=1}^{N} W_i N x y^T A x y^i N x y \tag{4}$$

$$W_i = e^{-\sigma i}$$

Ahol σ a súlyfüggvény meredeksége, N pedig a maximális figyelembe vett séta hossza. Az algoritmus nagy előnye, hogy teljesen invariáns a csomópontok számozására, és különböző méretű gráfokat is össze tud hasonlítani nullákkal való kiegészítés nélkül is. Egy hátrány, hogy egy *n* és egy *m* méretű gráf összehasonlításának esetén a direkt szorzat gráf már *m x n* méretű lesz, annak adjacencia mátrixa pedig $n * m \; x \; n * m$ méretű lesz, ami nagy gráfok esetén lényegesen lelassíthatja az algoritmust.

2.4. Objektumkövetés

Különböző tárgyak valósidejű követése egy rendkívüli fontossággal bíró eleme számos kiterjesztett valóság rendszernek. A közelmúltban fejlesztett AR rendszerek alapvetően három különböző módszert használnak ennek a feladatnak az elvégzésére. [8] Az első a szenzor alapú követés, amihez valamilyen (mágneses, akusztikus, optikai, stb.) szenzort használnak, majd ezek segítségével követik a tárgyakat. A második megoldás a vizuális követés, mely esetben az objektumokat egy kamera segítségével követik az objektumok mesterséges (marker based), vagy természetes (natural feature) jellemzőit kihasználva. A harmadik lehetőség a hibrid követés, amely az előző két módszer együttes használata. Ebben a fejezetben a vizuális követési eljárásokat mutatom be részletesebben.

A vizuális követéshez első sorban könnyen azonosítható és követhető képi elemekre, képi jellemzőkre van szükség. Gyakori megoldás, hogy ezek biztosítása végett az objektumokra különböző markereket helyeznek fel, majd ezeket követik. Ez az én feladatom esetén nem kivitelezhető, mivel a markerek használata ellentmond a tetszőleges, előre nem ismert tárgyak felhasználásának. A másik megoldás, hogy az objektumokon természetesen előforduló képjellemzőket kövessük.

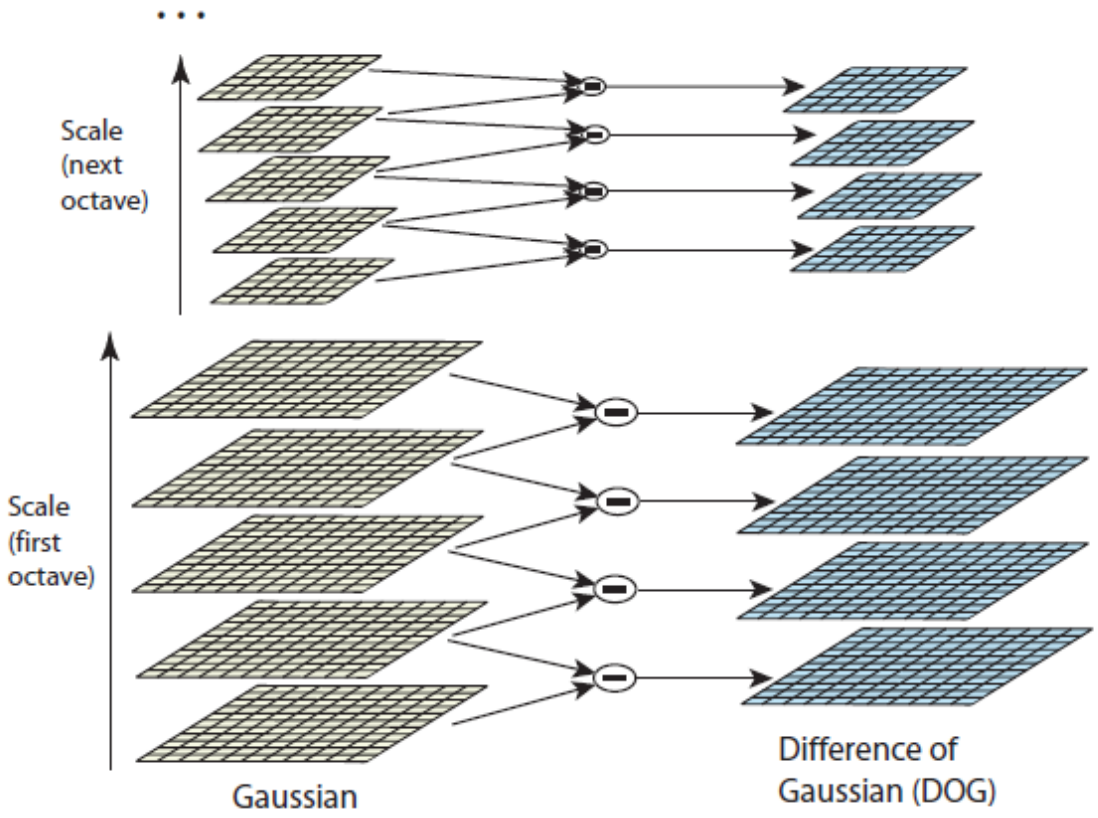
A tárgyak követéséhez először a rendszer indításakor az azokon található képi jellemzőket detektálni kell, és egy adatbázisban el kell tárolni, majd később a felépített adatbázisban található objektumokat kell a képen keresni. Az eltárolt képjellemzők alapján ezután az adott objektumot észlelni lehet minden egyes képkockán, ami viszont több tárgy három dimenzióban történő detektálása esetén lassú lehet. Park et. al. [23] a módszer gyorsítása érdekében egy adott képkockán csak a követendő objektumaink egy részét detektálta, a többit ideiglenes képjellemzők segítségével követte. A követés előnye, hogy gyorsabb, mivel a jellemző párokat csak egymáshoz közel kell keresni, azonban gyors mozgások esetén az objektumokat az algoritmus könnyen elveszítheti. Egy további hátránya az ideiglenes jellemzőkre alapuló követésnek, hogy a képkockáról képkockára akkumulálódó követési hibák miatt a követés egyre pontatlanabbá válik. Ez a probléma megoldható a néhány képkockánként történő detektálás eredménye alapján történő korrekció segítségével.

További fontos lépés a követést segítő képi jellemzők tárgyalása. Gyakori lehetőség az élek, sarokpontok, vagy kontúrok követése, de az utóbbi időben a legelterjedtebb módszer a textúrált régiók használata. Az efféle képjellemzők elterjedésének az első löketet Lowe [24] adta a SIFT (Scale Invariant Feature Transform) algoritmussal. Az algoritmus főbb lépései a következők: [24]

- Az eljárás különböző skálafaktorok mellett Difference of Gaussians (DoG) szűrőket alkalmaz a képen (*2.6 Ábra*), így megtalálhatja a kép karakteres részeit, valamint minden ilyen ponthoz hozzárendeli azt a skálafaktort, amelyik esetén maximális a szűrő kimenete.
- A stabilitásuk alapján kiválasztja a megfelelő pontokat (kontrasztosság, sarokság)
- A lokális gradiens alapján egy következetes orientációt ad a pontnak. Az ez utáni műveletek már mind figyelembe fogják venni a megtalált pont méretét, lokációját és orientációját, így biztosítjuk a képjellemző invarianciáját.
- A pont környezetében kiszámolt gradiensek alapján összeállít egy leírót a képjellemzőhöz, amely invariáns a geometriai torzításokra, és a fényviszonyok változására.

A SIFT algoritmus után számos hasonló alapötletből kiinduló képjellemző detektáló és leíró eljárás született, én ezek közül a KAZE [25], illetve az AKAZE [26] eljárásokat emelném ki. Ezek az algoritmusok nemlineáris skálateret építenek, majd ebben keresnek sarokszerű, könnyen követhető kulcspontokat. Fő előnye az eljárásoknak, hogy mindkét algoritmus képes a SIFT

pontosságát elérni, illetve esetenként meghaladni, azonban mindezt kisebb futási idő mellett. Az AKAZE eljárás esetében a futási idő már lényegesen alacsonyabb, így az már valósidejű követésre is alkalmazható. Más, konkurens algoritmusok, például az ORB [27] vagy a BRIEF [28] általában gyorsabbak, mint a SIFT, de a hatékonyság, vagy robusztusság árán.)



2.6 Ábra: A SIFT algoritmusban alkalmazott DoG szűrő. Forrás: [24]

3. Tervezés és megvalósítás

Az alábbi fejezetben a megvalósítandó prototípusrendszer megtervezésének legfontosabb kérdéseit mutatom be. A legelső részben az alakleíró algoritmus implementálási kérdéseivel foglalkozom, amelyek az ismeretlen jelenet feltérképezéséhez szükségesek, majd ezt követően a tanuló algoritmus létrehozását tárgyalom. Ezt követően a tanulás fontos velejárójával, vagyis a hiperparaméterek hangolására szolgáló kereszt-validációról számolok be, amit a tanuló algoritmus predikciós képességeit felhasználó lokalizációs lépés követ, melynek segítségével a feltérképezett jelenetben a rendszer placeholdernek használható objektumokat keres. Végezetül a valós idejű követéshez használt algoritmust ismertetem.

3.1. Alakleírás

Ebben a fejezetben az alakzat leírására használt algoritmus részletes leírását találhatjuk. Az első részben a rekonstrukció és a szegmentáció algoritmusáról teszek említést, majd az alakhoz tartozó jellemző vektor előállítása következik. A prototípusrendszer esetén bemenetként egy sztereó képpárt használtunk. Ahhoz, hogy ebből egy 3D jelenetet konstruáljunk, a képeket előállító kamera pár előzetes kalibrációja szükséges. Ezt a kalibrációs lépést sakktábla-módszerrel előre elvégeztem, így az algoritmus a kamerák és az elrendezés paramétereit is megkapja. A 3D rekonstrukciót a rendszer három különböző algoritmussal is el tudja végezni: Az egyik a Semi-Global Block Matching (SGBM) [29], a másik a Belief Propagation (BP), a harmadik pedig a Constant-Space Belief Propagation (CSBP) [30] algoritmus. Mindhárom algoritmushoz az OpenCV könyvtár létező implementációit használtam. Az alapértelmezett módszer az SGBM volt, mivel a másik kettőhöz az OpenCV csupán GPGPU implementációt szolgáltat az algoritmusok nagy számításigénye miatt.

Fontos megemlíteni, hogy az SGBM algoritmus egyik nagy hátránya, hogy számos paraméterrel rendelkezik, melyek helyes beállítása szükséges a jó minőségű rekonstrukció eléréséhez. Ehhez írtam egy külön alkalmazást (*3.1. Ábra*), amellyel az algoritmus paramétereit grafikus felület segítségével tudtam valós időben hangolni.

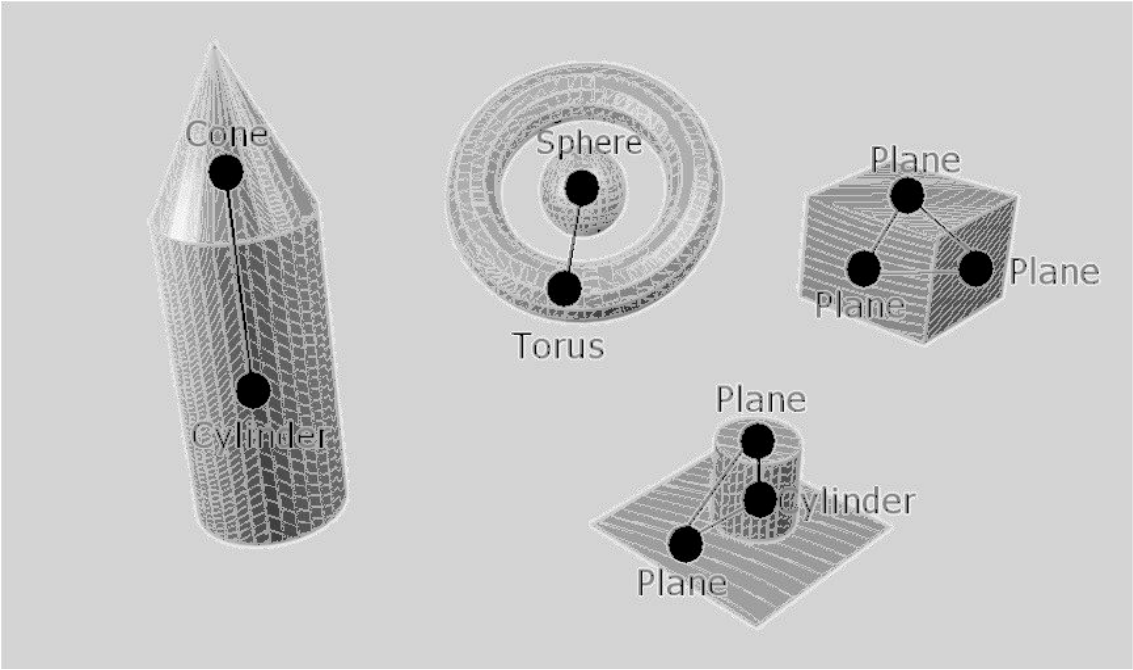
3.1 Ábra: A rekonstrukció-hangoló alkalmazás

A következő megvalósítandó lépés a szegmentáció, ahol egy fontos választási lehetőségem volt, melyről már az előző fejezetben is írtam. Az egyik lehetőség az, hogy már ennél a lépésnél megpróbálom az egyes valós objektumokat elválasztani egymástól, az alakfelismerő algoritmusnak pedig ezekről az objektumokról kell majd eldöntenie, hogy melyik kategóriába tartoznak. A másik lehetőség, hogy a szegmentálás elvégzése során a jelenetet nem objektumokra bontom le, hanem a teljes objektumoknál feltehetően kisebb építőelemekre, amelyeknek könnyű leírni az alakját, és az egyes építőelemek objektumokhoz való hozzáadását pedig az alakfelismerő algoritmusra bízom.

Az első módszer hátránya, hogy az ilyen szegmentálás rendkívül önkényes, mivel nem minden esetben egyértelmű, hogy mi a helyes eredmény. Éppen ezért a második megoldás mellett döntöttem: A szegmentáló lépésben csupán primitív alakzatokat keresek a pontfelhőben: síkokat, hengereket, gömböket, kúpokat, és tóruszokat. Eztán ezekből egy gráfot konstruálok és majd a tanuló algoritmus ebben a gráfban fog az objektumokhoz tartozó részgráfokat keresni. Ennek a

megoldásnak a legfontosabb előnye, hogy így a szegmentációt az algoritmus az alapján végzi el, hogy milyen objektumokat tud beilleszteni az adott jelenetbe. A szegmentáció helyes eredményének megválasztása ugyan még így is önkényes marad, viszont a feladat elvégzése szempontjából ideális lesz.

A primitívek megkeresésére [17] RANSAC algoritmusát használtam, amely többféle primitívet képes egyszerre keresni, valamint a forráskódja online elérhető [31], így nem kellett ezt a lépést külön implementálni. Az algoritmus futási sebességének és pontosságának növelése érdekében a használt pontfelhőt szűrtem és mintavételeztem.



3.2 Ábra: Primitívekből készült gráf (csak a lényeges élek jelölve)

A következő lépés a gráf felépítése, melynek csomópontjait a RANSAC által talált primitívek alkotják (*3.2 Ábra*). Azonban a gráfban minden csomópontnak van meghatározott számú jellemzője (hasonlóan a Schnabel et. al. [18] által használt gráfhoz), melyek a csomóponthoz tartozó primitív tulajdonságait írják le. Azonban különböző fajta primitíveknek más tulajdonságaik, és más jellemzőik vannak, ami miatt a csomóponthoz tartozó jellemző vektorok különböző méretűek lehetnek. Ezt elkerülendő csak egy jellemző vektor tartozik minden csomóponthoz, ami az egyes primitív típusokhoz tartozó vektorok konkatenációja. Természetesen a nem értelmezett jellemző értékeket (például ha az adott primitív henger, akkor a gömb, sík, kúp,

stb. jellemzőit) nullára állítottam. A különböző primitívekhez meghatározott jellemzőket a 3.1 Táblázatban láthatjuk.

3.1 Táblázat: A primitívek és a meghatározott jellemzőik

Primitív	Jellemzők
Sík	Átmérő Terület Befoglaló téglalap területe
Gömb	Sugár
Henger	Sugár Magasság
Kúp	Sugár Magasság Oldal dőlésszöge
Tórusz	Sugár A test sugara

Ezen felül minden csomóponthoz tartozik egy koordinátarendszer, amit a primitív egy kitüntetett pontja, és annak egy kitüntetett iránya (általában a tengely iránya, vagy egy felületi normális). Az egyetlen kivétel a gömb, amelyhez nem tartozik semmilyen irány egyértelműen. Ennek köszönhetően definiálhattam egy transzformációt a primitívek (vagyis a gráf csomópontjai) közt. Ez a transzformáció a speciális pontok közti transzláció, és a kitüntetett vektorok közti elforgatás együttese. Ezek a transzformációk lesznek a gráf éleinek, vagyis a csomópontok kapcsolatainak a jellemzői, egy kvaternióval és egy elmozdulás vektorral jellemezve. Ily módon mindig teljes gráfot fogok létrehozni, hiszen az az információ, hogy a csomópontok szomszédosak-e már az élhez tartozó transzformációban szerepel. A teljes gráf konstruálásának legfontosabb előnye, hogy így az egyes csomópontok szomszédosságának mértéke egy folytonos, valós számmal jellemzett tulajdonság, és nem egy önkényes küszöbértékkel meghatározott bináris jellemző, így sokkal robusztusabb megoldást kaphatunk.

3.2. Tanulás

A következő lépés a gráf beillesztése a véletlen séta kernelbe, ahol egy kisebb nehézségbe ütköztem. A probléma, hogy a véletlen séta kernel valós súlyokkal ellátott súlyozott gráfokat vár, a jelenlegi alkalmazás pedig vektorok által súlyozott gráfokat használ. A probléma áthidalására definiáltam egy egyszerűbb kernel függvényt két csomópont illetve egy második kernelt két él között is, majd úgy írtam át a véletlen séta kernel direkt szorzat számító algoritmusát, hogy ahol két él, vagy csomóponti súly szorzatát kellene kiszámítani, ott e helyett az adott kernelt használom fel arra, hogy két jellemző vektorhoz egy számot hozzárendeljek.

Ez azt jelenti, hogy a direkt szorzat adjacencia mátrixát egy az élek között definiált kernel függvény segítségével számolom ki. Ehhez a kernelhez az egész transzformáció helyett annak csak két jellemzőjét használom fel: A kvaternió $cos(\frac{\alpha}{2})$ részét, valamint a két csomópont közti távolság négyzetét. A többi kihagyásának oka az, hogy a csomópontok pontos orientációja nem egyértelmű (csak egy irányt tudtunk hozzájuk köti és nem egy egész keretet), így ennek a felhasználása hibás működéshez vezethet. Az él kernel pontos értéke az alábbiak szerint határozható meg (RBF kernel felhasználásával):

$$E = e^{-\gamma * \Delta\alpha - \delta * \Delta d - \mu * d} \tag{5}$$

Ahol μ, γ és δ az él kernel hiperparaméterei, $\Delta\alpha$ a két élhez tartozó $cos(\frac{\alpha}{2})$ részek különbségének abszolút értéke, Δd az élekhez tartozó távolságnégyzetek különbségének abszolút értéke, míg d a távolságnégyzetek átlaga. A távolságok átlagát azért vesszük számításba, mivel közeli objektumok nagy valószínűséggel ugyanahhoz az objektumhoz tartoznak, míg a távoliak nem. Ezért egy olyan hasonlósági függvényt definiáltam, amely a távoli éleket bünteti a hasonlóságuktól függetlenül, ily módon szőve bele a szomszédossági információt a kernel működésébe.

A következő lépésben a séta kezdési és befejezési valószínűségeit tartalmazó vektort állítom elő a csomópontok közt definiált kernel függvény segítségével. Itt egy újabb problémával álltam szemben: A jellemző vektor, amit a csomópontokhoz hozzárendeltem, olyan jellemzőket tartalmaz, amelyek értékei különböző nagyságrendekben mozognak. Vannak terület, távolság és

szög típusú jellemzők, amiket egyszerű RBF kernellel nem lehet összehasonlítani, mivel a nagy abszolút értékű jellemzők (leginkább a területszerűek) kicsi relatív eltérése is lényegesen nagyobb abszolút hibát okoz, mint egy szög típusú jellemző nagy relatív eltérése. Megoldásképp egy jóság értéket konstruáltam, egész pontosan egy „rosszság" értéket, ahol külön figyeltem, hogy minden jellemzőt a *[0-1]* tartományba skálázzak. A skálázás implementációjára az abszolút különbséget az összehasonlítandó értékek összegével osztottam. Fontos megjegyezni, hogy a jellemzők típusa, illetve a primitívek geometriai megkötései miatt az egyes jellemzők mindig pozitív számok. Ezután a kiszámolt *W* rosszság értékkel már megvalósíthattam az RBF kernel függvényt. (6-7)

$$W = \sum_{i=1}^{M} \frac{|a_i - b_i|}{a_i + b_i} \tag{6}$$

$$N = e^{-\vartheta * W} \tag{7}$$

Ahol az összehasonlítandó vektorok $A = [\, a_1\ a_2\ \dots\ a_M\,]$ és $B = [\, b_1\ b_2\ \dots\ b_M\,]$, ϑ pedig egy újabb hiperparaméter. A véletlen séta kernelnek két további egyértelmű hiperparamétere van. Az első az *N*, vagyis a maximális figyelembe vett séta hossza, míg a másik az *n* hosszú séták relatív súlyát meghatározó súlyfüggvény meredekségét meghatározó paraméter. A fejlesztés során én bevezettem egy újabb hiperparamétert. Ennek oka, hogy a kernel értéke erősen függött attól, hogy milyen hosszú gráfokat hasonlítottunk össze. Éppen ezért a kernel függvény végeredményét az alábbi módon módosítottam:

$$K_f = \frac{K}{(m * n)^{\varepsilon}} \tag{8}$$

Ahol *m* és *n* a két gráf mérete, K a kernel eredménye, ε pedig a hetedik hiperparaméter. A kernel függvényt először MATLAB-ban valósítottam meg, mivel ebben a környezetben mátrixműveletek könnyen megvalósíthatóak, illetve ezek debugolása is lényegesen egyszerűbb, mint C++ környezetben. A MATLAB implementáció további előnye, hogy a bioinformatika toolboxban egy beépített SVM algoritmus is található. A fenti érvek miatt ezt a környezetet használtam a kernel algoritmus működésének ellenőrzésére. Természetesen a végleges verzióhoz C++ nyelven is implementáltam az algoritmust, hogy a prototípus környezetében is hatékonyan

működhessen. A C++ verzió a LibSVM [32] nyílt forráskódú függvénykönyvtár SVM algoritmusát használja fel.

Fontos szempont volt a kernel függvény futási idejének jelentős csökkentése is. Már korábban említettem a véletlen séta kernel számításigényének jelentős emelkedését a gráfok méretétől függően, amely összetett objektumkategóriák esetén jelentősen csökkentheti a futási sebességet. Az tanítás azonban kisméretű gráfok esetén 4-500 tanító példa esetén is elfogadhatatlanul lassú volt, nagyjából 2-3 perces futási idővel. Természetesen a tanításnak nem szükséges valós időben futni, de ekkora kernel futási idő már a klasszifikáció esetén is gondot okozhat, aminél viszont megköveteljük a valósidejű futást.

Mivel a kernel függvény számos mátrix és vektorműveletet tartalmaz, melyek könnyedén párhuzamosíthatóak, célszerűnek tűnt a kernel függvény GPGPU implementációja által gyorsítani a program futásán. Ehhez először létrehoztam egy keretrendszert, mely a program indításakor betölti a tanításhoz, illetve az osztályozáshoz szükséges változókat, valamint a tanításra használt adatokat a grafikus kártya memóriájába, lévén, hogy a grafikus és az általános célú processzorok közti kommunikáció a grafikus kártya használatának szűk keresztmetszete. Ügyelni kellet viszont a grafikus kártya memóriájának véges méretére, túlságosan sok tanító adat (nagyjából 15 000) esetén egy 1GB-os memóriával rendelkező grafikus kártya memóriája megtelik a kernel függvény eredményeivel.

A grafikus kártya segítségével történő gyorsítás során a kernel függvényhez tartozó él és csomóponti kerneleket implementáltam le CUDA kernel formájában. Az él kernel esetén az egyetlen grafikus magon futó program a keletkező direkt szorzat gráf egyetlen élének súlyát számolta ki. Ehhez tulajdonképpen az (5) egyenlet implementációja volt szükséges, egyetlen változtatással. Mindkét kiindulási gráf adjacencia mátrixa tartalmaz hurokéleket, melyek jellemzői az alapértelmezett nulla értéket veszik fel. Az (5) egyenlet szerint viszont ekkor a kernel eredménye 1 kellene, hogy legyen. Ez viszont nem szerencsés, mivel így a csupán a matematikai ábrázolás kényelmessége miatt reprezentált élek jelentősen befolyásolnák a kernel eredményét, így az ilyen esetekben az eredményt nullára kell állítanunk.

A csomóponti kernel esetén a párhuzamosítást meglehetősen bonyolultan oldottam meg. Ennek oka az, hogy ha a gráfok méretei *n* és *m*, akkor az él kernel esetén *n*m x n*m* párhuzamosítható számítási feladatunk van, míg a csomóponti kernel esetén csak *n*m*, amivel valószínűvel nem sikerül egyetlen fonatot (warp) sem kitölteni. Ráadásul a csomóponti kernel képletében van egy summa is (6), amit párhuzamosíthatunk. Mivel egy fonat 32 grafikus magból áll, ebbe két csomópont kiszámolása fér bele úgy, hogy a kihasználatlan grafikus magok száma minimális legyen. Az összegzés megoldására osztott memóriát használtam.

Természetesen további párhuzamosítható lépés volt a véletlen séta kernel fő képlete is, amely mátrix-mátrix, mátrix-vektor, illetve skaláris szorzatot tartalmaz. Ehhez viszont nem készítettem külön kernel függvényt, mivel a CUDA nyelvhez elérhető a CUBLAS lineáris algebra függvénykönyvtár, melynek segítségével a végső képletet (4) egyszerűen implementálhattam. A GPGPU gyorsítás meghozta az eredményét, mivel a tanítás futási ideje, néhány percről 10-20 másodpercre esett le, így biztos lehettem abban, hogy a klasszifikáció is csak 20-40 milliszekundum idejű lesz.

3.3.Kereszt-validáció

Az előző fejezet során bemutattam a használt tanuló algoritmust, és annak legfontosabb tulajdonságait és az implementációhoz kapcsolódó kérdéseit. Ebben a fejezetben a tanuló algoritmusok egy rendkívül fontos tulajdonságához kapcsolódó kérdéskört tárgyalok tovább, amely a hiperparaméterekhez kapcsolható. Egy tanuló algoritmus a tanulás során mindig egy költségfüggvény minimalizálását végzi el, melyen eredményeképp egy modell paramétereit állítja elő. Ez a modell a tanításra használt adatokon valamilyen hibaszázalékkal képes elvégezni azok klasszifikációját, vagy egyéb más műveletet, amire a tanuló rendszert használni akarjuk. A tanulás előtt azonban egy tervezői döntést kell meghoznunk, amellyel az eljárás hibája és a megtanult modell bonyolultsága között próbálunk egyensúlyt teremteni.

Ez az egyensúly azért lényeges, mivel túl hibatűrő tanulás esetén a megtanult modell jó eséllyel alulillesztett (underfitted/high bias) lesz, vagyis mind a tanulás, mind a későbbi predikció esetén hagy hibával fog dolgozni. Ellenkező esetben, ha alacsony a hibatűrés, de nem büntetjük a rendszer bonyolultságát, akkor a modell túlillesztett (overfitted/high variance) lesz, és bár a tanulási hiba alacsony lesz, a rendszer mégis elveszti a predikciós képességét, vagyis nem lesz

képes a tanult adatok alapján általánosítani (emberi tanulás esetén a magolás rendkívül találó hasonlat erre a jelenségre).

Ennek a második optimalizálási feladatnak egy népszerű megoldási eszköze a kereszt-validáció, melynek során a tanító adatainkat N (általában 5, vagy 10) részre osztjuk, és a modellt minden részhez külön betanítjuk a maradék N-1 rész felhasználásával. Eztán a tanításból kihagyott egy résznyi adatra futtatjuk a predikciós algoritmust, és meghatározzuk, hogy az algoritmus mennyire képes a másik N-1 rész alapján helyesen megbecsülni a kimaradt rész értékét. Ha ezt mind az N részhalmazra elvégezzük, és a helyes becsléseket összegezzük, akkor megkapjuk a kereszt-validációs pontosságot. Ezt a pontosságot maximalizálva határozzuk meg a hiperparaméterek optimális értékét.

Ezt az eljárást alkalmaztam a saját tanuló algoritmusom 8 hiperparaméterének meghatározására (7 kernel paraméter, továbbá az SVM C paramétere). Az optimalizálásra általában egy egyszerű paraméterhálót szoktak használni, vagyis több értékkel kipróbálják az algoritmust, majd ezek közül a legjobbat kiválasztják. Ez azonban az én esetemben nem járható út, ugyanis itt egy nyolc dimenziós paraméterhálót kellene létrehozni, ahol csupán minden paraméterhez tíz különböző érték megvizsgálásához a tanítást 10^8-szor kéne elvégezni, ami 10 másodperces futási idővel egy milliárd másodpercig, vagyis több évtizedig tartana.

Éppen ezért a kereszt-validáció futtatásához egy genetikus algoritmust használtam. A genetikus algoritmus előnye a paraméterhálóval szemben, hogy egy sokdimenziós tér esetén sokkal kevesebb kiértékelés árán lényegesen nagyobb területet képes bejárni, így sokkal kevesebb kiértékelésből nagyobb valószínűséggel és pontossággal állapíthatom meg a hiperparaméterek optimális értékét. A genetikus algoritmus beillesztéséhez a GALib [33] nevű nyílt forráskódú könyvtárat használtam fel, mely többfajta algoritmust és genomot is tartalmaz, valamint lehetővé teszi az egyedek, vagy akár az egész populáció kiértékelő függvényének (az objektív függvénynek) a felüldefiniálását.

Ez utóbbi különösen szerencsés volt számomra, ugyanis egy egész populáció kiértékelése során gráf kernel függvényt kell meghívnom ugyanazokra a gráfokra csak éppen különböző paraméterekkel. Éppen ezért a kereszt-validáció esetén egy újabb SIMD (Single Instruction Multiple Data) típusú problémát kaptam, amit tovább párhuzamosíthattam a grafikus kártya segítségével. Felmerülhet, hogy ezzel az újabb párosítással a grafikus magok véges száma miatt

már nem érhetek el számottevő sebességnövekedést, emlékezzünk azonban, hogy a gráfok mérete meglehetősen kicsi (1-5 csomópont), így a grafikus processzor meglehetősen gyakran nincs megfelelően kihasználva.

Az újabb GPGPU implementációhoz először az él és a csomóponti kerneleket módosítottam. Ezek a kernel függvények most már egyetlen paraméter helyett egy paramétereket tartalmazó tömböt kapnak bemenetként, illetve a kimenetük is egyetlen vektor/mátrix helyett egy olyan tömb, amely egymás után a különböző paraméterekkel kiszámolt vektorokat/mátrixokat tartalmazza. A kereszt-validációs gráf kernel függvényhez egy saját mátrixszorzó kernelt is implementáltam, mivel itt is szerettem volna a paraméterek szerint párhuzamosítani, a CUBLAS könyvtár erre viszont nem ad lehetőséget. A mátrixszorzó kernelt a CUBLAS-ban található szorzófüggvénnyel megegyező működéssel implementáltam, vagyis az alábbi műveletet valósítottam meg:

$$C = \alpha AB + \beta C \tag{9}$$

Ahol *A*,*B* és *C* mátrixok, α és β pedig valós számok. Az én implementációmban természetesen a mind a mátrixokból, mind az α és β paraméterekből több különböző példány követi egymást a memóriában. A kereszt-validációhoz külön mátrix-vektor és skaláris szorzó kernelt nem fejlesztettem, mivel mindkettő a mátrix-mátrix szorzás speciális esete.

3.4.Lokalizáció

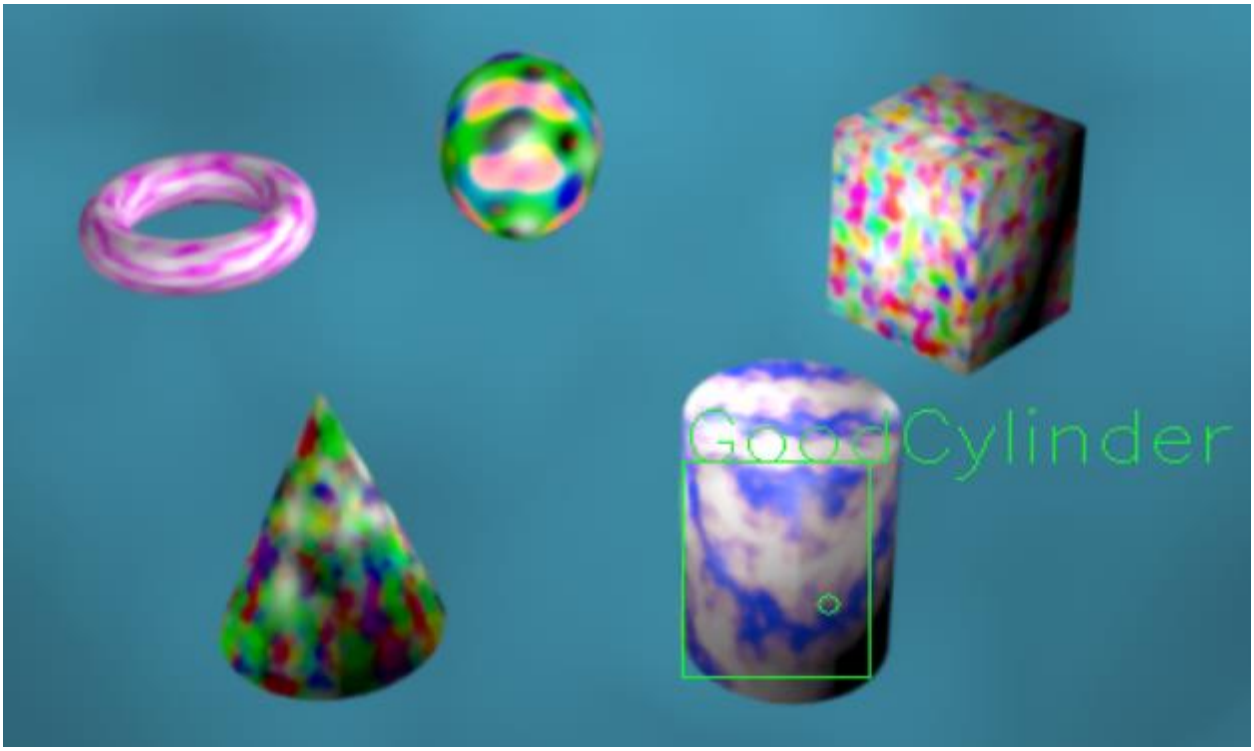
Korábban említettem, hogy a szegmentációs lépés során nem döntjük el, hogy mely részletek mely objektumokhoz tartoznak, hanem ezt majd a tanuló algoritmusra bízzuk. A fent bemutatott tanuló algoritmus viszont egy kategorizáló eljárás, amely nem képes egy külön részgráfot kiemelni egy egész jelenetből, így a lokalizációhoz egy külön eljárás szükséges. Az általam megvalósított eljárásnak két fő lépése van: az első, hogy minden egyes objektumkategóriához meghatározzuk azokat a jelölteket, amelyek az adott kategória példányai lehetnek az adott jelenetben, míg a második, hogy ezek közül kiválasztunk egy olyan részhalmazt, amely nem tartalmaz átfedést, és lehetőleg a jelenetbe leginkább illeszkedő példányokat tartalmazza.

A lokalizációs kezdeti lépéseként az adott jelenet gráf minden egyes csomópontját, mint különálló részgráfot kategorizáljuk az adott objektumkategóriához tartozó két osztályú SVM modellje segítségével. Ez a lépés minden egyes csomóponthoz előállít egy számot (a bináris

klasszifikáció eredménye küszöbözés előtt), amelyet az adott csomópont jóságának fogunk tekinteni. Ezt követően a csomópontokat jóság alapján sorba rendezzük, majd kiválasztjuk az első (legjobb) csomópontot. Ezután iteratív módon újabb csomópontokat próbálunk az elsőhöz hozzáadni jóság szerint sorban haladva, figyelembe véve az újabb csomópont kapcsolatát a jelenlegi részgráffal, meggátolva a túlságosan messzi csomópontok hozzáadását. Legvégül ellenőrizzük, hogy az új csomóponttal (és az ahhoz tartozó élekkel) kiegészült részgráf jobb eredményt ér-e el a klasszifikáció során, és amennyiben ez nem teljesül, a hozzáadást visszavonjuk.

Ha már több csomópontot nem tudunk a gráfhoz adni, akkor ellenőrizzük a megtalált részgráf helyességét úgy, hogy a kategorizáló tanuló algoritmussal kiértékeljük. Ha az algoritmus a részgráf esetében is úgy dönt, hogy az az adott objektumkategóriába tartozik, akkor a lokalizáció sikeres. (*3.3 Ábra*) Ellenkező esetben a részgráfot elvetjük. Ezután a megtalált objektumhoz elmentjük a hozzá tartozó csomópontokat, illetve minden egyes csomóponthoz keresünk egy párt a leginkább illeszkedő tanító példa gráfban, majd ezek középpontjait pontpárok formájában elmentjük. A legjobban illeszkedő példa megtalálására a kernel függvény eredményét használjuk (ezt a klasszifikáció során úgyis minden felhasznált tanító gráfhoz kiszámoltuk), míg az egyes csomópontokhoz tartozó párt a csomóponti kernelt használjuk fel. Az így meghatározott pontpárokat később fel lehet használni, hogy a megtalált objektum és a példa gráf közti hat szabadságfokú transzformációt és a skálázást megbecsüljük.

Amint a lokalizáció első fázisa az összes létező objektumkategóriához meghatározta a lehetséges példányokat, akkor a következő lépés azon jelöltek kiválasztása, melyek a lehető legjobbak valamilyen kiértékelési szempont szerint, azonban nincs köztük átfedés (vagyis nem használjuk fel ugyanazokat a csomópontokat kétszer). A jelenlegi megoldás során az egyes jelöltek jóságát a kategorizálásuk konfidenciája alapján végezzük el. Itt probléma lehet, hogy a különböző kategóriákhoz tartozó SVM modellek küszöbözés előtti kimenete más-más tartományokban mozoghat, így adott kategóriák jelöltjei konzisztens módon előnyösebb helyzetben lehetnek. Ezt a problémát jól kezeli, hogy a különböző kategóriákat azonos hiperparaméterekkel tanítjuk, így az SVM klasszifikáció eredményei ugyanabba a nagyságrendbe fognak esni.



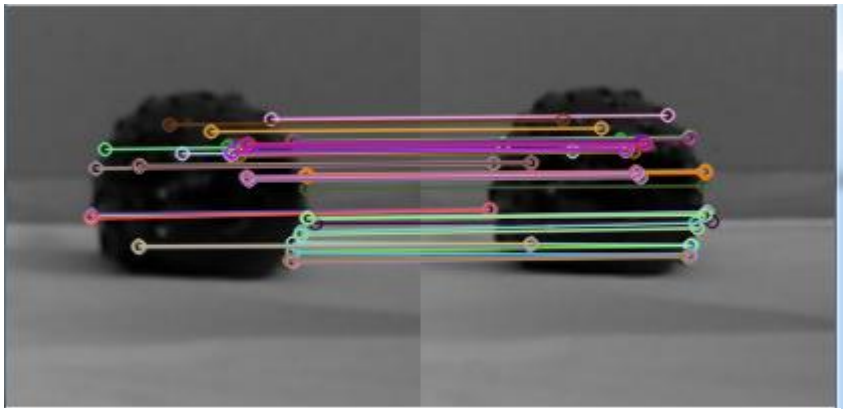
3.3 Ábra: Egy henger lokalizációja

3.5.Követés

Az utolsó alfejezetben a követési lépéssel kapcsolatos implementációs kérdéseket tárgyalom. Az első lépés természetesen a követési adatbázis elkészítése, hiszen ez esszenciális a további lépésekhez. Ezt követi magának a követésnek az implementációja, majd az újradetektáláshoz használatos jellemző felhő építése.

Az adatbázis építés első lépése a követendő objektumokat beazonosítása. Ehhez a lokalizációs lépés kimenetét használom, amely megadja, hogy az adott virtuális objektumhoz mely valós primitív formák tartoztak. A szegmentációs lépés során minden primitívhez meghatároztam az őt befoglaló téglatestet, így ennél a lépésnél csupán ezeket a téglatesteket kell összegeznünk, hogy a követendő térrészt megtaláljuk. A megtalált térrészhez azonban még meg kell határoznunk az ahhoz tartozó képrészletet is, mivel a képjellemző detektálást a képen kell elvégezni. Ehhez a kalibráció során meghatározott kameramátrix segítségével visszavetítettem a befoglaló téglalap nyolc sarkát, majd meghatároztam az ezeket befoglaló téglalapot. Ezután a téglalap által adott képrészleten megkeressük a könnyen követhető jellemző területeket. Ehhez az AKAZE képjellemző detektáló és leíró algoritmust használtam.

Fontos azonban, hogy az objektumokat (lévén, hogy kiterjesztett valóság alkalmazásról van szó) a 3D térben szeretném követni, nem pedig csupán a képsíkon. Éppen ezért a sztereó képpár mindkét képén megkeresem az AKAZE képjellemzőket (*3.4 Ábra*), egy párosító eljárás segítségével pedig ezekből sztereó jellemzőpárokat készítettem. Ez az epipoláris megkötéseket felhasználó párosító eljárás az OpenCV cookbookból származó Robust Matcher algoritmus módosításából született. [34] Az adatbázisban minden egyes így megtalált képjellemzőt eltárolunk. A későbbi követés érdekében a képjellemzők mellé azoknak a múltbéli követési lépésekkor detektált 3D pozícióit is eltároltam, amit innentől history-nak fogok nevezni.

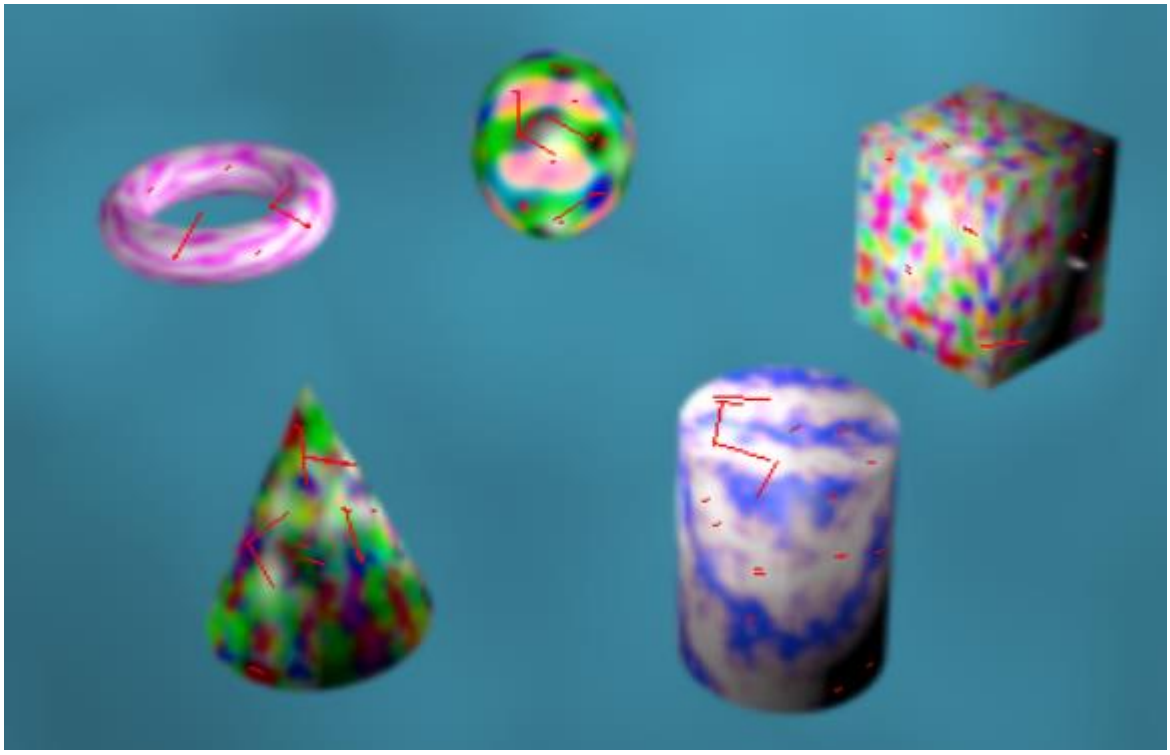


3.4 Ábra: Sztereó képjellemző párok egy valós objektum esetében

A második fázis a valós idejű követés megvalósítása. Követés során szintén az első lépés, hogy a soron következő képkocka páron sztereó képjellemző párokat keresünk a fent leírt módszerrel. Ezután a megtalált képjellemző párokat igyekszünk a korábban megtalált párokkal összevetni. (*3.5 Ábra*) Itt is a RobustMatcher algoritmus egy módosított változatát használom, de más megkötésekkel. Míg sztereó becslés esetén a fő megkötés, hogy a párok az epipoláris egyenesek mentén helyezkedjenek el, itt a megkötés, hogy a képjellemző párok a háromdimenziós térben relatíve közel helyezkedjenek el, valamint, hogy a képjellemző pozíciója az eddigi mozgása alapján megbecsülhető pozíciójától se helyezkedjen el messze.

Az összepárosított képjellemzők előállítása után azokat az egyes objektumpéldányokhoz kell sorolni, amit egyrészt az objektumok befoglaló téglalapja, másrészt a képjellemzők és az objektumok mozgástörténetének összevetése alapján végeztem el. Ezt követően az egy adott objektumhoz tartozó jellemzők (*3.6 Ábra*) koordinátáiból két vektort állítottam elő: az egyik a képjellemzők előző időpillanatban felvett koordinátáit, a másik a jelenlegi koordinátákat tartalmazza. Ezek segítségével meg tudtam becsülni a 3D homogén transzformációt, amelyet az

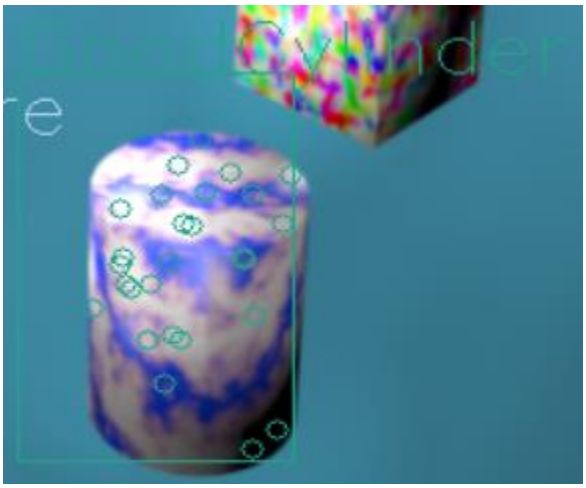
objektum a két időpillanat közt elszenvedett. Erre egy saját RANSAC algoritmusra épülő becslőt használok. Ennek fő tulajdonsága, hogy egy SVD felbontásra épülő algoritmust [35] használ a rotációs mátrix megbecslésére, miután a transzformáció elmozdulás komponensét meghatároztuk, így ez az algoritmus csak olyan jelölteket fog előállítani, amelyek rigid transzformációt írnak le. A saját RANSAC algoritmus megírásának további oka az volt, hogy így könnyen megoldható volt olyan módosítást eszközölni, mely bizonyos transzformáció jelölteket még a kiértékelés előtt elvet. Így meggátolhattam, hogy olyan transzformációkat adjon vissza az algoritmus végeredményképp, amelyek túl nagy mozgást, vagy elfordulást írnak le, így két képkocka között nem történhettek meg.



3.5 Ábra: A képjellemzők elmozdulása két képkocka közt

Miután az egész objektumra értelmezett transzformációt meghatároztam, az objektumhoz tartozó egyes csomópontok három dimenziós befoglaló téglatesteit ennek értelmében transzformáltam. Ezt követően a csomópontok befoglaló téglatesteiből az objektumét újra kiszámoltam. Ezt követően a jelenlegi és az azt megelőző képpárokon detektált képjellemzők párosítását újra elvégzem, immár figyelembe véve azt is, hogy az objektum elmozdulását követően az ahhoz tartozó képjellemzőknek hova kellett kerülnie a három dimenziós térben.

A követés utolsó fontos eleme az újradetektálás és az ahhoz szükséges jellemző felhő építése, amely alatt képjellemzőket és azok koordinátáit tartalmazó struktúrát értek. Az objektumokhoz minden képkockán meghatározom a hozzá tartozó képjellemzőket, azonban ezek jelentős része nem stabil, így a későbbi detektálásra sem alkalmas. Éppen ezért a detektálásra használt jellemző felhőbe csak azokat a jellemzőket illesztem be, amelyeket már több korábbi esetben is sikeresen detektáltam és követtem. Ily módon az objektum elvesztése esetén egy egyszerű jellemző párosítás és transzformáció becslés segítségével az objektumot újra detektálhatom.



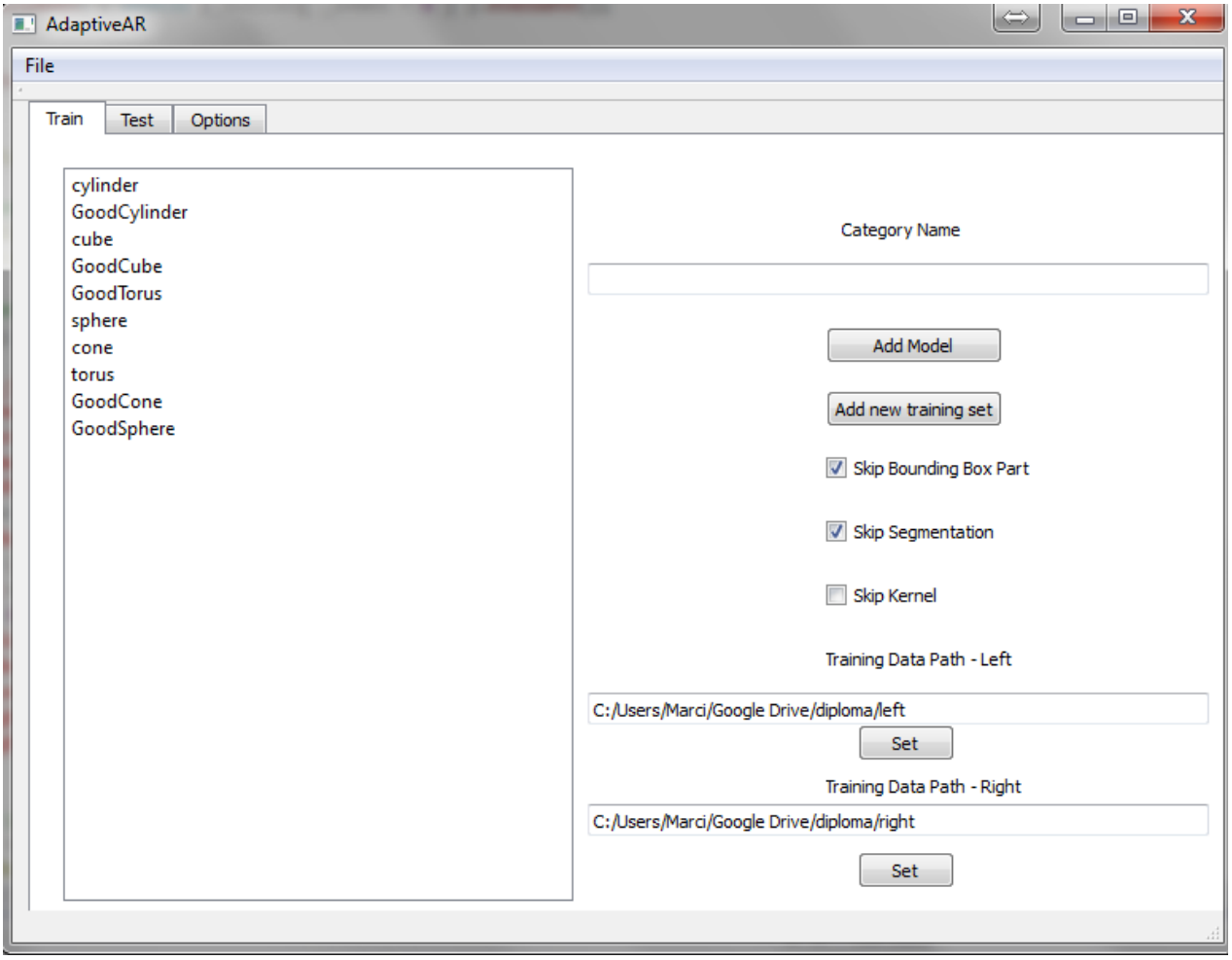
3.6 Ábra: Egy objektumhoz tartozó képjellemzők

4. Tesztelés, eredmények értékelése

Az alábbi fejezetben a prototípusrendszer elemeit magába foglaló tesztkörnyezet és a teszteléshez használt adatok és módszerek bemutatását végzem el, valamint a tesztelés során kapott eredményeket értékelem. Az első három alfejezetben legfontosabb elemeket, vagyis az alakfelismerést, a lokalizációt, és a követést elemzem. Az utolsó alfejezetben a skálázhatóság kérdésére térek ki külön, lévén, hogy ez az egyes algoritmusok felhasználhatóságához elengedhetetlen.

A teszteléshez használt keretrendszert Visual Studio környezetben fejlesztettem a Qt platformfüggetlen grafikus felhasználói felületeket tartalmazó könyvtár segítségével. (*4.1 Ábra*) Ennek a környezetnek a segítségével a különböző algoritmusok paramétereit, és egyéb beállításait grafikus felületről adtam meg, az algoritmusok indítását és az eredmények (esetleges grafikus) ellenőrzését is innen végeztem el. Az alkalmazás többszálas vezérlésű, így biztosítva a grafikus felület válaszolását az eljárások futása alatt, míg a szálak közti kommunikációt a Qt által szolgáltatott signal-slot kommunikációval, illetve az eseménykuldő szolgáltatás segítségével oldottam meg.

A megvalósított algoritmusok teszteléséhez több teszt sztereó képsorozatot is készítettem, mind valós, illetve virtuális eszközökkel. (*4.2 Ábra*) Azért esett a sztereó képpárokra a választásom, mert ezek segítségével könnyedén juthattam mélység információhoz, míg egy teljes több oldalról történő rekonstrukció megvalósítása lényegesen bonyolultabb lett volna, és a jelen kutatásnak nem fókusza. Valós teszt szettet két megegyező típusú Axis biztonsági kamera segítségével készítettem hétköznapi, egyszerű formájú, egy íróasztalon előforduló objektumok felhasználásával. A virtuális teszt sorozatot Blender segítségével hoztam létre, egy virtuális kamerapár felhasználásával. Ezeken a képsorozatokon még egyszerű formák szerepelnek, könnyen felismerhető textúrákkal. Az objektumok mozgását a valós sorozatok esetében kézzel, míg a virtuális képek esetében a kamera elforgatásával értem el. A virtuális képsorozatra külön radiális torzítást tettem, majd egy virtuális sakktábla segítségével a kamerapárt kalibráltam.



4.1 Ábra: A használt teszt alkalmazás

4.2 Ábra: A használt valós és virtuális jelenetek

4.1. Alakfelismerés

A tanuló alakfelismerő algoritmus teszteléséhez először a virtuális tesztsorozatot használtam, mivel igyekeztem a valós kamerák különböző hibáiból eredő pontatlanságokat az algoritmus teszteléséből kihagyni, a tesztelés jelenlegi fázisánál a kérdés ugyanis az, hogy képes-e az algoritmus egy könnyen megtanulható adatsor alapján eredményesen tanulni, ezért fontos, hogy feleslegesen nehéz tanító adatokat az első tesztelés során ne adjunk.

A teszteléshez az első lépés a tanító adatok előállítása. Ehhez először minden egyes képhez egy grafikus felületen megadtam az algoritmus számára az egyes objektumkategóriák példányai két dimenziós befoglaló téglalapját. Ezután elvégeztem a sztereó rekonstrukciót, majd az alak alapú szegmentáló és gráf készítő lépést. Ezután minden kategóriához két gráfot készítettem: Azok a csomópontok, melyek középpontjai a két dimenziós képre visszavetítve az adott kategória befoglaló téglalapjában voltak a köztük lévő élekkel együtt a pozitív gráfba kerültek, míg a többi csomópont, és a köztük lévő élek a negatívba. A pozitív és a negatív gráf csomópontjai közt lévő élek egyik gráfban sem lettek reprezentálva.

Az így elkészült kategóriánként 500 tanító adat (a virtuális képsorozatban 250 kép található) segítségével aztán lefuttattam a genetikus algoritmust, amely a kereszt-validáció segítségével hivatott megkeresni az optimális hiperparamétereket. A genetikus algoritmus paraméterei, a megtalált optimális paraméterek, illetve az öt kategória kereszt-validációs pontossága és tanítási hibája rendre a 4.1, 4.2 és 4.3 Táblázatokban találhatóak.

4.1 Táblázat: A genetikus algoritmus paraméterei

Paraméterek	Generációszám	Populációszám	Mutáció esélye	Keresztezés esélye
Értékek	50	50	0,01	0,05

4.2 Táblázat: Az optimális hiperparaméterek

Paraméterek	ϑ	γ	μ	δ	N	σ	ε	C
Értékek	0,0232	0,0097	0,00012	0,042	10	1,3	3	2,23

4.3 Táblázat: Az egyes kategóriák tanítási hibái és validációs pontossága

Kategória	Tanítási hiba	Validációs pontosság
Kocka	0,4%	98%
Henger	0,2%	98,8%
Kúp	0%	97,2%
Gömb	0%	98,8%
Tórusz	0,4%	98,4%

Az eredmények szemmel láthatóan kimagaslóan jók, azonban kis vizsgálódás után látható, hogy az oka az, hogy a tanuló algoritmus egy olyan jellemző alapján végzi el a pozitív és negatív gráfok szétválasztását, ami egy tervezési hiba miatt került a rendszerbe. A hiba az volt, hogy megengedtük, hogy a véletlen séta kernel végén történő gráf méret alapján történő súlyozás (8) erősségét (az ε hiperparamétert) hangolandó paraméterként kezelje az algoritmus. Ráadásul olyan tanító adatokat generáltunk, amelyek méret alapján könnyen szétválaszthatóak: a negatív „maradék” gráf általában négyszer annyi csomópontból áll, mint a pozitív.

Éppen ezért a kernel ε hiperparaméterét fixen egyre állítottuk, mivel a kernel gráf mérettől való függése az (4) egyenlet alapján jó közelítéssel lineáris. A másik fontos változtatás az volt, hogy a tanításhoz negatív példákként többé nem a „maradék” gráfot, hanem más kategóriák pozitív mintáit használtuk fel, így többé nincs jelentős méretkülönbség a pozitív és negatív minták közt. Így a genetikus algoritmus a korábbival megegyező paraméterek mellet újra futtatva már lényegesen más paramétereket (*4.4 Táblázat*), és lényegesen rosszabb eredményeket (*4.5 Táblázat*) kapunk.

4.4 Táblázat: Az optimális hiperparaméterek a javított kategóriák esetén

Paraméterek	ϑ	γ	μ	δ	N	σ	ε^2	C
Értékek	1,78	0,0105	0,00514	0,117	9	2,25	1	0,57

[2] A paraméter értéke 1-be volt kényszerítve

4.5 Táblázat: A javított kategóriák tanítási hibái és validációs pontossága

Kategória	Tanítási hiba	Validációs pontosság
Kocka	4,24%	74,32%
Henger	3,32%	76,40%
Kúp	4,56%	74,00%
Gömb	4,40%	74,16%
Tórusz	3,36%	75,76%

Ahogy az az eredményekből látszik komoly túlillesztés jelensége lép fel a tanítás során, melynek több oka is lehet. Az egyik, hogy rosszak a megbecsült paraméterek, ami már csak az optimalizálás miatt is valószínűtlen. További érv emellett, hogy az eredményül kapott paraméterektől jelentősen eltérő értékek mellett az „optimálishoz" nagyon hasonló eredményeket kaptam, vagyis az algoritmus érzéketlen a paraméterek megválasztására, ezért feltételeztem, hogy a probléma máshol rejtőzik.

További lehetőség, hogy az alakleírás során előállított jellemzők nem megfelelőek a gráfok közti helyes különbségtétel elvégzéséhez. Ennek oka lehet egyszerűen a jellemzők elégtelensége, vagy az, hogy a mérésben végzek nagy hibát. Miután megfigyeltem a mért jellemzők értékeit ugyanazon a sztereó képpáron többször egymás után megmérve, kijelenthetem, hogy a túlillesztés problémáját nagy valószínűséggel ez utóbbi okozza. A probléma ugyanis, hogy a sztereó képalkotás csupán egy felületet szolgáltat, így az egyes alakzatok nagy része hiányzik, ami a RANSAC algoritmus működését rendkívül megnehezíti. További probléma, hogy a sztereó rekonstrukció esetén könnyedén meghatározhatjuk az egyes elemek kamerától számított távolságát, azon finom felületi struktúrája viszont már relatíve nagy zajjal terhelt, e két tényező együttes hatása a RANSAC algoritmus végeredményét teljesen véletlenszerűvé teszi. A probléma áthidalásához egy bonyolultabb és pontosabb 3D rekonstrukciós eljárás alkalmazása szükséges.

További fontos szempont volt az algoritmusok sebességének tesztelése. A 4.6 Táblázatban számos mérést találhatunk a különböző algoritmus elemek futási sebességéről. Látható, hogy néhány elem futása bőven az egy másodperces határ fölött van, azonban ezek az elemek csak a jelenet feltérképezéséhez és berendezéséhez szükségesek, amit csak egyszer a rendszer indításakor

kell elvégezni, így a néhány másodperces válaszidő a felhasználói élményt nem akadályozza. A legfontosabb eredmény, hogy az osztályozás futása csupán néhány ms, melyek tisztán látható, hogy az eljárásunk alábbi része valósidejű működésre képes.

4.6 Táblázat: Az alakfelismeréshez kapcsolódó algoritmusok futási ideje

Tanítás (500x500 adat)	23 154,836 ms
Kereszt-validáció (1 generáció, 10 fős populáció)	157 688,250 ms
Osztályozás (72 szupport vektor)	14,237 ms

4.2.Lokalizáció

A lokalizáció teszteléséhez szintén a virtuális képsorozatot, illetve az az alapján tanított klasszifikáló algoritmust használtam fel. Bár a klasszifikáció sikerességét a tanító algoritmus túlillesztési hibája jelentősen ronthatja, azonban a túlillesztés jellegzetessége, hogy a legtöbb esetben az algoritmus a tanító adatokat képes helyesen kategorizálni, így bíztam abban, hogy emiatt ugyanazon az adatsoron a lokalizáció is fog működni. Ezzel ugyan az általánosító képességet nem sikerül bizonyítanom, azonban azt nem is itt, hanem az alakfelismerő résznél kell biztosítani.

A lokalizáció sikerességét az alábbi módon számszerűsítettem: a tanító szett előállítása során meghatároztam, hogy az egyes képeken a kategóriák hol helyezkednek el, amit a lokalizáció végeredményével összehasonlítottam, megkapva a helyes lokalizálások számát az összes előfordulás[3] arányában. Külön előállítottam a hamis pozitív és negatív lokalizációs hibák arányát is, ahogy az a 4.7 Táblázatban látszik. A lokalizációs algoritmus esetén is mértem futási időt is külön számítva az egyes kategóriák lokalizációs lépését, illetve az optimális elrendezést számító kikeverő lépést is. (*4.8 Táblázat*)

[3] A virtuális képsorozaton ez a készített képek száma (250) volt minden kategória esetében, a részleges elfedéseket ugyanis nem vontam le.

4.7 Táblázat: A lokalizáció eredményessége az egyes kategóriák esetén

Kategória	Helyes (%)	Fals pozitív (%)	Fals negatív (%)
Henger	95,2%	5,5%	4,8%
Kúp	87,7%	13,8%	12,3%
Tórusz	95,6%	3,2%	4,4%
Gömb	85,8%	7,6%	14,2%
Kocka	95,2%	3,3%	4,8%

A táblázatban található eredmények a korábban tárgyalt túlillesztési probléma ellenére meglehetősen pontosak. Ennek ellenére megfigyelhetjük, hogy a hibaszázalékok a tanuló algoritmushoz képest valamivel nagyobbak. Ez a jelenség részben a túlillesztés, illetve a lokalizációhoz használt részgráfok építésének hibái rovására írható. További megjegyzés, hogy a helyes és a hamis lokalizációk összege azért nem 100%, mivel a százalékos eredmények az összes kép és nem az összes lokalizáció arányában értendőek.

4.8 Táblázat: A lokalizáció futási ideje

Egy kategória lokalizációja	587,625 ms
A kikeverő lépés	0,109 ms
Teljes lokalizáció (5 kategória)	2 938,235 ms

A futási időre vonatkozó adatokból tisztán látszik, hogy öt kategória esetén az algoritmus teljesíti a sebességre vonatkozó követelményeinket, vagyis képes az adott jelenetet objektumokkal néhány másodperc alatt berendezni. Ehhez azért nem szükséges gyorsabb végrehajtás, mivel a lokalizációs lépést csak a rendszer használatának kezdetekor végezzük el.

4.3.Követés

A követési algoritmust mind a virtuális mind a valós képsorozaton teszteltem, segítségképp a lokalizáció helyes eredményét az első képkocka esetén megadva (így helyes pontból indul az algoritmus). A helyességet a tanításhoz megadott kétdimenziós befoglaló téglalapok segítségével ellenőriztem, meghatározva azon képkockák arányát, ahol a követés eredménye ezektől jelentősen

eltér (*4.9 Táblázat*). Természetesen a futási időt a követési fázis esetében is megmértem (*4.10 Táblázat*).

4.9 Táblázat: A követő algoritmus eredményessége

Helyes követés aránya	98,43%
Átlagos helyes követési időtartam	32,14 képkocka

Az eredmények alapján az algoritmus meglehetősen pontos, mivel nagyjából 30 képkocka szükséges ahhoz, hogy az objektum követés segítségével meghatározott középpontja jelentősen elmozduljon az előre megadott helyes eredményhez képest. Az újradetektálási lépést azonban ennél lényegesen gyakrabban kell elvégezni, mivel a teszt során a követési hibát egy egyébként is zajos referenciához (kézzel megadott befoglaló téglalapok) képest mértem, így a követési hiba meghatározásánál megengedő voltam.

4.10 Táblázat: A követő algoritmus futási ideje

Sztereó jellemző detektálás és párosítás	532,81 ms
Objektumok frissítése	84,16 ms
Jellemző párosítás pontosítása	5,38 ms
Teljes	622,34 ms

Az eredmények alapján látható, hogy az AKAZE detektor kétszeri futása miatt a valósidejűségi kritériumunk itt enyhe mértékben sérül, mivel a jellemző detektálásnak legfeljebb 50-100 ms alatt végre kéne hajtódnia, amelyen kisebb képméret használata, valamint az AKAZE detektor GPGPU implementációja segíthet.

4.4.Skálázhatóság

Az utolsó fontos tárgyalandó szempont a rendszer skálázhatóságának a kérdése. Ezt két külön aspektusból volt szükséges vizsgálnom: egyrészt a valós jelenetben található követendő objektumok száma, valamint a virtuális objektumokhoz használt objektumkategóriák száma és bonyolultsága alapján. Az előbbi a bonyolultabb, összetett valós jelenetek használásánál fordulhat elő, és alapvetően két algoritmusra van jelentős hatással. Az egyik a lokalizáció, a másik pedig a követés.

A lokalizáció során a jelentből készült gráf lényegesen több csomópontból fog állni, ami a kernel számításigényét négyzetesen befolyásolja. A követés során a sok objektum a követés második fázisának megnövelt számú kiértékelését vonja magával, ami objektumok számától függő részek esetén e szerinti lineáris növekedést jelent. Itt további gond lehet, hogy a legtöbb képjellemző detektor, köztük az AKAZE implementációja is felső korlátot szab a megtalált képjellemzők számának, ami ha túl sok objektum közt oszlik el, elégtelen lehet az elmozdulás hat szabadságfokú meghatározására. Ez a felső korlát azonban szerencsére 500, így nagyjából 30-50 objektumnak kéne egyszerre látszania ehhez, így ennek a problémának a fellépése valószínűtlen. A 4.11 Táblázatban láthatjuk a lokalizáció és a követés futási idejét az objektumok/csomópontok száma függvényében.

4.11 Táblázat: Az algoritmusok futási ideje az objektumok száma/komplexitása függvényében

Objektumok Csomópontok	1-2 objektum 4-5 csomópont	4-5 objektum 10-12 csomópont	10-12 objektum 25-30 csomópont
Lokalizáció	3 024,12 ms	3 873,59 ms	6 118,56 ms
Követés	632,33 ms	652,30 ms	660,23 ms

Az eredményekből látszik, hogy az előzetes jóslataim nagyjából beigazolódtak, figyelembe véve természetesen, hogy a lokalizáció esetén a kernel algoritmus GPGPU implementáció miatt a jelenetben lévő csomópontok száma csak azután kezdi el lényegesen növelni a futási időt, miután a grafikus kártya processzorainak kihasználtsága eléri a lehetséges maximumot. A követő algoritmus esetén a függés szinte elhanyagolható, mivel az algoritmus jelentős része nem függ az objektumok számától.

A kategóriák bonyolultsága a lokalizáció, valamint a kernel algoritmus futási idejét érinti, míg a kategóriák száma a lokalizáció futási idejét érinti, méghozzá lineárisan. Fontos azonban megjegyezni, hogy ez a jellemző nem csak a futási időre van hatással. A tanuló algoritmus megkülönböztető képessége ugyanis nem végtelen, ezt minél jobban kihasználjuk, annál kevésbé lesz képes a rendszer az egyes kategóriákat megkülönböztetni. A hatékonyan megkülönböztethető kategóriák száma természetesen újabb, jól megválasztott jellemzők felvételével növelhető. A kategóriák bonyolultsága alapvetően ugyanazzal a hatással bír, mint az objektumok száma a jelenetben, mivel mind a kettő kifejezhető a kernelben lévő gráfok csomópontjainak számával. A

kategóriák tulajdonságainak hatása a tanulás és a lokalizáció futási idejére a 4.12 és 4.13 Táblázatokban látható.

4.12 Táblázat: A lokalizációs algoritmus futási ideje a kategóriák számának függvényében

Kategóriák száma	1	5	10
Lokalizáció	608,91 ms	2 547,44 ms	5 098,27 ms

4.13 Táblázat: Az algoritmusok futási ideje a kategóriák komplexitásának függvényében

Kategóriák átlagos csomópontszáma	1-3	4-6	10-12
Véletlen séta kernel (500x500 adatra)	18 065,81 ms	22 372,99 ms	30 623,71 ms
Lokalizáció (5 kategória)	3 054,25 ms	1 841,91 ms	2 115,94 ms

A táblázat adatai alapján feltűnhet, hogy a kernel algoritmus esetén nem látható a gráf mérettől feltételezett négyzetes függés, aminek az oka szintén a GPGPU implementáció. Három, vagy annál kevesebb csomópont esetén ugyanis nem lehet már a grafikus kártya lehetőségeit teljesen kihasználni, így a csomópontok számának növelésével a kihasználtság is nő. A lokalizáció esetén viszont egyáltalán nem várt eredményeket láthatunk, ugyanis a futási idő nagy részét itt is a kernel futása teszi ki, tehát az előbbihez hasonló függést kellene kapnunk. Fontos azonban azt észrevenni, hogy a kategóriák átlagos csomópontjainak számával a tanulási problémát is megváltoztattam, így azok a modellek, amikkel a lokalizációt végeztem a mérések során elkerülhetetlenül megváltoztak. A tanulás sikerességétől függően viszont a felhasznált szupport vektorok száma is jelentősen megváltozik, ami a lokalizáció futási idejét alapvetően meghatározza. Így rávilágítottam egy fontos aspektusra: Habár látható, hogy 10, vagy több kategória esetén a lokalizáció futási ideje már kezd az elfogadhatatlan kategóriába tartozni, viszont a túlillesztési probléma megoldásával ez a futási idő könnyedén csökkenthető.

5. Összefoglalás

A dolgozat során bemutattam egy kiterjesztett valóság rendszert mely egy előre ismeretlen környezet felhasználásával képes tapintható felhasználói felületet létrehozni, így lehetővé téve az egyszerű hétköznapi használatot. Részletesen ismertettem egy prototípusrendszer fejlesztési lépéseit a legfontosabb algoritmusokra koncentrálva, valamint a fejlesztés eddig elért eredményeit. A munkám során kifejlesztettem egy alakfelismerésre épülő tanuló algoritmust, melynek segítségével egy előre ismeretlen teret virtuális objektumokkal berendezhetünk, illetve egy eljárást, mellyel a térben lévő valós objektumokat képes valós időben követni. Ezeket az eredményeket sikeresnek tekintem, bár fontos elismerni, hogy több téren továbbfejlesztésre szorulnak az algoritmusok, viszont nem szükséges azok alapvető átgondolása, vagy elvetése. A jelenlegi hibák nagy valószínűvéggel javíthatók a prototípus az eredeti elképzeléshez való közelítésével.

Végzős mesterképzéses hallgató lévén a jelenlegi kutatásomat szeretném a doktori képzés keretein belül folytatni. Szerencsére bőven akad továbbfejlesztési, illetve további kutatási lehetőség is. Ezen lehetőségek közül a lehető legfontosabb a 3D jelenet teljes rekonstrukciója. Ezt nagy valószínűséggel egykamerás motion stereo eljárással fogom végezni, mivel a különböző kiterjesztett valóság szemüvegek gyakran csak egy kamerával rendelkeznek. Ez a megoldás nagy valószínűséggel javítani fogja a tanuló algoritmus problémáinak nagy részét, viszont a követő algoritmus módosítása szükséges hozzá, mivel nem állnak majd rendelkezésre sztereó képjellemzők. A másik fontos továbbfejlesztési lépés a környezet végleges összeállítása, melyhez első sorban a különböző feladatok és virtuális környezetek adminisztrációja, illetve a virtuális objektumok kijelzésének megoldása szükséges.

A további kutatás szempontjából az elsődleges cél az objektumok párosításának a fejlesztése. Az ember számára logikus berendezés létrehozásához szükséges az objektumok kontextusának, illetve bizonyos könnyen felismerhető támpontokhoz (falak, padló, ajtó, ablak, stb.) viszonyított helyzetének figyelembevétele is. Fontos lehet további jellemzők hozzáadása, melynek segítségével a felismerhető kategóriák számát gyarapíthatjuk. Ezek közül fontosak a különböző vizuális jellemzők (jellemző régiók, textúra) alapján történő döntések bevezetése, mellyel

elkerülhetjük veszélyes objektumok felhasználását (vezetékek, konnektor, éles tárgyak, stb), vagy éppen meghatározhatjuk a valós tárgyak anyagát, így segítve a párosítást.

További fontos területe lesz a jövőbeli kutatásomnak a lokalizációs lépés továbbfejlesztése. Fontos, hogy a rendszer az egyszerű kategorizáláson felül próbáljon az adott alkalmazás igényeinek is megfelelni, bizonyos objektumok például nélkülözhetetlenek egy adott feladathoz, míg mások csupán dekoráció céljául szolgálnak, így az előbbieket a lokalizáció során előnyben kell részesíteni. Az optimális berendezésnek még számos további szempontja is lehet, melyeket a lokalizáló algoritmusba mind be kell építeni. A további szempontokra jó példa lehet a visszaillesztési pontosság: az algoritmus figyelembe veheti, hogy egy adott virtuális tárgy modellje mennyire illik bele az adott térrészbe, és ez alapján a pozíciót, az orientációt, illetve a skálát pontosíthatja, vagy rendkívül rossz illeszkedés esetén a párosítást felülbírálhatja.

Hivatkozások

[1] J.-S. Kim, *Tangible User Interface for CAVE based on Augmented Reality Technique,* Faculty of the Virginia Polytechnic Institute and State University, 2005.

[2] V. Buchmann, S. Violich, M. Billinghurst and A. Cockburn, "FingARtips – Gesture Based Direct Manipulation in Augmented Reality," in *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, Singapore, 2004.

[3] H. T. Regenbrecht, G. Baratoff, I. Poupyrev and M. Billinghurst, "A Cable-less Interaction Device for AR and VR Environments," in *Proceedings of ISMR*, 2001.

[4] P. Rojtberg and A. Olwal, "Tangible Interfaces using Handheld Augmented Reality," in *SIGRAD*, Västeräs, 2010.

[5] D. Bandyopadhyay, R. Raskar and H. Fuchs, "Dynamic Shader Lamps : Painting on Movable Objects," in *IEEE and ACM International Symposium on Augmented Reality*, New York, NY USA, 2001.

[6] K. Matkovic, T. Psik, I. Wagner and D. Gracanin, "Dynamic Texturing of Real Objects in an Augmented Reality System," in *VR 2005. IEEE Virtual Reality*, Bonn, 2005.

[7] B. R. Jones, H. Benko, E. Ofek and A. D. Wilson, "IllumiRoom: Peripheral Projected Illusions for Interactive Experiences," Microsoft Research, Redmond, WA USA, 2013.

[8] F. Thou, H. B.-L. Duh and M. Billinghurst, "Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR," in *7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Cambridge, 2008.

[9] M. Billinghurst, H. Kato and S. Myojin, "Advanced Interaction Techniques for Augmented Reality Applications," in *Lecture Notes in Computer Science*, Berlin, Springer, 2009, pp. 13-22.

[10] G. A. Lee, M. Billinghurst and G. J. Kim, "Occlusion based Interaction Methods for Tangible Augmented Reality Environments," in *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, Los Angeles, CA USA, 2004.

[11] M. Billinghurst, H. Kato and I. Poupyrev, "Tangible Augmented Reality," 2001. [Online]. Available: http://www.csie.nuk.edu.tw/~ayen/teach/ar/ref/Tangible%20Augmented%20Reality.pdf.

[12] W. Broll, E. Meier and T. Schardt, "The Virtual Round Table - a Collaborative Augmented Multi-User Environment," in *Proceedings of the ACM Collaborative Virtual Environments*, San Fransisco, CA USA, 2000.

[13] R. Osada, T. Funkhouser, B. Chazelle and Dobkin David, "Matching 3D Models with Shape Distributions," in *SMI 2001 International Conference on Shape Modeling and Applications* , Genova, 2001.

[14] A. Golonivskiy, V. G. Kim and T. Funkhouser, "Shape-based Recognition of 3D Point Clouds in Urban Environments," in *IEEE 12th International Conference on Computer Vision*, Kyoto, Japan, 2009.

[15] F. Tombari and L. Di Stefano, "Object recognition in 3D scenes with occlusions and clutter by Hough voting," in *Fourth Pacific-Rim Symposium on Image and Video Technology*, Singapore, 2010.

[16] M. A. Fishler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Magazine Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[17] R. Schnabel, R. Wahl and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," *Computer Graphics Forum,* vol. 26, no. 2, pp. 214-226, 2007.

[18] R. Schnabel, R. Wahl, R. Wessel and R. Klein, "Shape Recognition in 3D Point Clouds," in *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*, Bory, 2008.

[19] M. Ferrer, F. Serratosa and A. Sanfeliu, "Synthesis of median spectral graph," in *Second Iberian Conference, IbPRIA 2005*, Estoril, 2005.

[20] D. H. White, *Generative Models for Graphs,* Department of Computer Science, The University of York, 2009.

[21] P. Zhu and R. C. Wilson, "Stability of the Eigenvalues of Graphs," in *11th International Conference, CAIP 2005*, Versailles, 2005.

[22] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor and K. M. Borgwardt, "Graph Kernels," *Journal of Machine Learning,* vol. 11, pp. 1201-1242, 2010.

[23] Y. Park, V. Lepetit and W. Woo, "Multiple 3D Object Tracking for Augmented Reality," in *7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Cambridge, 2008.

[24] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[25] P. F. Alcantarilla, A. Bartolli and A. J. Davison, "KAZE Features," in *12th European Conference on Computer Vision, Part VI*, Florence, Italy, 2012.

[26] P. F. Alcantarilla, J. Nuevo and A. Bartoli, "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces," in *In Proceedings British Machine Vision Conference*, Bristol, UK, 2013.

[27] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *2011 IEEE International Conference on Computer Vision*, Barcelona, 2011.

[28] M. Calonder, V. Lepetit, C. Strecha and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *11th European Conference on Computer Vision*, Heraklion, 2010.

[29] H. Hirschmüller, "Stereo Processing by Semi-Global Matching and Mutual Information," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 30, no. 2, pp. 328 - 341, 2007.

[30] Q. Yang, L. Wang and N. Ahuja, "A Constant-Space Belief Propagation Algorithm for Stereo Matching," in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA USA, 2010.

[31] R. Schnabel, R. Wahl and R. Klein, "RANSAC Implementation," 2007. [Online]. Available: http://cg.cs.uni-bonn.de/en/publications/paper-details/schnabel-2007-efficient/.

[32] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology,* vol. 2, no. 3, pp. 27:1-27:27, 2011.

[33] M. Wall, "GAlib A C++ Library of Genetic Algorithm Components," MIT, 1999. [Online]. Available: http://lancet.mit.edu/ga/.

[34] R. Laganiére, "OpenCV Cookbook Source Code - Robust Matcher," 2011. [Online]. Available: https://code.google.com/p/opencv-cookbook/source/browse/trunk/Chapter%2009/matcher.h.

[35] F. L. Markley, "Attitude Determination using Vector Observations and theSingular Value Decomposition," *Journal of the Astronautical Sciences,* vol. 38, no. 3, pp. 245-258, 1988.