



TANSZÉKVEZETŐ

## DIPLOMATERVEZÉSI FELADAT

**Szemenyei Márton**

szigorló villamosmérnök hallgató részére

### Ember-gép interfész kialakítása objektumfelismerés és -követés segítségével

Háromdimenziós felhasználói felületek esetében nehézséget jelent az objektumok virtuális megfogása (vagyis a szem látja az objektumot, de a tapintás során nem érzékeljük), ami kisebb mértékű idegrendszeri konfliktust jelent az agy számára. Ennek a problémának a megoldására számos megoldás létezik, de többségük igen drága, így a gyakorlati életben nem alkalmazható.

Egy egyszerűbb megoldást jelent, ha a virtuális objektumokhoz valós objektumok is tartoznak (más kialakítással, felületekkel stb.), ami – ha nem is tökéletesen – de képes rá, hogy az emberi agy számára egy hihető objektumot képezzen. Ennek a megoldásnak egyik nehézsége, hogy a környezet objektumait felismerjük és mozgásukat folyamatosan kövessük.

A hallgató feladata egy olyan megoldás elkészítése, melyben a rendszer képes a felhasználó által mozgatott objektumok felismerésére, követésére, illetve visszacsatolás segítségével a személlyel interaktív kapcsolat kialakítására.

A hallgató feladatának a következőkre kell kiterjednie:

- Tekintse át a téma szakirodalmát. Vizsgálja meg, hogy más műhelyekben milyen objektumfelismerési és -követési megoldásokat alkalmaznak.
- Készítsen rendszertervet egy megoldásra, amely alkalmas arra, hogy az objektumok pozíciója és orientációja folyamatosan követhető legyen.
- Készítse el a rendszert és egy demonstráló alkalmazást, amely segítségével a felhasználó a számítógéppel kapcsolatot tud tartani, a számítógépnek információkat tud küldeni.
- Végezzen tesztek az eszköz pontosságára és hatékonyságára vonatkozóan

**Tanszéki konzulens:** Dr. Vajda Ferenc, docens

**Külső konzulens:** -

Budapest, 2014. március 14.

/ Dr. Szirmay-Kalos László /  
egyetemi tanár  
tanszékvezető



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

Szemenyei Márton

# **EMBER-GÉP INTERFÉSZ KIALAKÍTÁSA**

Objektumfelismerés és -követés segítségével

KONZULENS

**Dr Vajda Ferenc**

BUDAPEST, 2014

## **Tartalomjegyzék**

|                                         |           |
|-----------------------------------------|-----------|
| <b>Összefoglaló .....</b>               | <b>1</b>  |
| <b>Abstract.....</b>                    | <b>2</b>  |
| <b>1 Bevezetés .....</b>                | <b>3</b>  |
| <b>2 A feladat specifikációja .....</b> | <b>5</b>  |
| <b>3 Előzmények.....</b>                | <b>9</b>  |
| 3.1 Kiterjesztett valóság.....          | 9         |
| 3.2 3D képfeldolgozás .....             | 12        |
| 3.3 Alakfelismerés .....                | 15        |
| 3.4 Tanuló rendszerek.....              | 20        |
| 3.5 Objektumkövetés .....               | 24        |
| <b>4 Megvalósítás .....</b>             | <b>27</b> |
| 4.1 A keretrendszer .....               | 27        |
| 4.1.1 Funkciók .....                    | 30        |
| 4.1.2 Osztályok .....                   | 37        |
| 4.1.3 Implementációs kérdések.....      | 43        |
| 4.2 Feltöltés algoritmusokkal.....      | 46        |
| 4.2.1 Rekonstrukció .....               | 46        |
| 4.2.2 Szegmentáció és alakleírás .....  | 48        |
| 4.2.3 Alakfelismerés .....              | 51        |
| 4.2.4 Követés .....                     | 60        |
| <b>5 Tesztelés, eredmények .....</b>    | <b>64</b> |
| 5.1 Keretrendszer .....                 | 64        |
| 5.1.1 Az alkalmazás profilozása .....   | 65        |
| 5.1.2 Memóriakezelés .....              | 71        |
| 5.2 Algoritmusok .....                  | 72        |
| 5.2.1 Alakfelismerés .....              | 72        |
| 5.2.2 Követés .....                     | 77        |
| <b>6 Összefoglalás.....</b>             | <b>79</b> |
| <b>Köszönetnyilvánítás .....</b>        | <b>81</b> |
| <b>Irodalomjegyzék.....</b>             | <b>82</b> |

# HALLGATÓI NYILATKOZAT

Alulírott **Szemenyei Márton**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2014. 12. 09.

.....  
Szemenyei Márton

# Összefoglaló

A virtuális és kiterjesztett valóság rendszerek célja, hogy új, hatékony módszereket fejlesszenek ki az ember-gép interakció megvalósítására. Ezen rendszerek egy csoportja a tapintható kiterjesztett valóság rendszerek (TAR – Tangible Augmented Reality), melyek a virtuális objektumok irányítását valós objektumok felhasználásával oldják meg, így a virtuális tárgyakat természetes módon lehet manipulálni.

A diplomamunkám témája egy TAR rendszer elkészítése, amely megkísérel egy előre ismeretlen valós teret intelligens módon virtuális objektumokkal berendezni. A jelenlegi megközelítésben az intelligens elrendezés a virtuális és az adott pozícióban lévő valós tárgy közti alakbeli hasonlóság alapján történik. Ily módon a felhasználók fizikai interakcióba léphetnek a virtuális objektummal, még hozzá oly módon, hogy az nem okoz idegrendszeri konfliktust.

Az elvégzendő munka legfőbb része egy keretrendszer létrehozása, amely az algoritmusokat fogja kezelni, valamint a keret feltöltésére szolgáló algoritmusok létrehozása. Ezen eljárások közül a legfontosabbak a virtuális és valós tárgyak alak alapján történő párosításának megvalósítása, és a valódi objektumok valósidejű követése. Az objektumfelismeréshez egy sztereó kamerapár segítségével meghatározott 3D jelenetet használok, az alakfelismerést pedig egy gráf kernel alapú SVM végzi. A követést képjellemzők alapján végzem el.

# **Abstract**

The goal of virtual and augmented reality systems is to provide new and efficient methods for the interaction between man and machine. A group of these systems, called Tangible Augmented Reality (TAR) achieves this goal by using real objects as an input device. These systems provide an intuitive, natural user interface by allowing users to manipulate virtual objects interacting with the aforementioned real objects.

The central subject of my master's thesis is the design and implementation of a TAR system that is able to insert virtual objects intelligently into a real scene of which the system has no a priori knowledge. The system is able to create an intelligent pairing of the real and virtual objects based on their similarity of shape. Using this method of pairing the system allows users to interact with virtual object physically without neural conflict.

The main part of my work was the creation of a framework responsible for handling the algorithms of the TAR system, and the implementation of the algorithms themselves. The most important algorithms are a shape matching method used to pair virtual and real objects, and the real-time tracking of real objects used to enable interaction. I use a 3D scene constructed from the images of a stereo camera pair, and a graph-kernel based SVM learning algorithm to implement the shape matching. The real-time tracking algorithm is based on natural image features.

# 1 Bevezetés

A számítástechnika számos forradalom kezdődött egy új felhasználói felület megjelenésével, éppen ezért ma is jelentős kutatás folyik új technikák kifejlesztésére. Az új felületek területén folyó kutatás egyik jelentős ágazata a Virtuális és Kiterjesztett Valóság (Virtual Reality/Augmented Reality; VR/AR) rendszerek területe.

A Virtuális Valóság lényege, hogy a felhasználó számára oly módon jelenítünk meg egy számítógép által generált környezetet, hogy a felhasználó úgy érezze, mintha valóban abban a környezetben lenne. A Kiterjesztett Valóság rendszerek esetében a felhasználó számára a valós környezetben helyezünk el virtuális objektumokat. Az ilyen rendszerekben a felhasználó valós időben lép kapcsolatba virtuális objektumokkal, valamint valós objektumokat is elérhet. A valós környezet miatt az AR rendszerek lényegesen könnyebben tudják a valódi világ tulajdonságait megvalósítani (pl.: fizika), míg ezek VR rendszerekben sokkal körülményesebbek. [1]

A legtöbb virtuális és kiterjesztett valóság rendszer azonban csak nehézségek árán képes a megjelenített virtuális objektumokhoz tapintásérzetet hozzárendelni, ami viszont használat közben az érzékszerveink ellentmondásához vezet. Ez az ellentmondás a felhasználói élményt jelentősen csökkenti, valamint a virtuális valóság rendszerek használata esetén fellépő kiberbetegség egyik fő okozója. Az idegrendszeri konfliktus elkerüléséhez szükséges a tapintásérzet hozzáadása a virtuális környezet eszköztárához. Ennek egy gyakori megoldása különféle kesztyűk használata, melyek képesek a tapintásérzet szimulálására. Az ember bőre azonban egy rendkívül összetett érzékszerv, így ennek mesterséges előállítása egy rendkívül bonyolult feladat, ráadásul a realisztikus szimulációhoz szintén szükséges erővisszacsatolás biztonsági kérdéseket is felvet.

A tapintásérzet bevezetésére egy másik jelentős irányzat a tapintható kiterjesztett valóság rendszerek (TAR) által szolgáltatott megoldás. Ezek a rendszerek működésük során a munkatérben elhelyezett valós tárgyakat rendelnek hozzá a megjelenített virtuális objektumokhoz, a felhasználók pedig a kiválasztott valós tárgyak segítségével léphetnek interakcióba a virtuális objektumokkal. [1] A legtöbb TAR rendszernek azonban két jelentős hiányossága van: Egyrészt, gyakran szükséges a munkatér előkészítése (pl.: a valós tárgyak markerekkel való ellátása a felismerést segítő), amely a hétköznapi

használatot jelentősen megnehezíti. Másrészt, a kiválasztott valós tárgy tapintása nem feltétlenül felel meg annak, amit a hozzárendelt virtuális objektumtól elvárnánk, így az idegrendszeri konfliktus megmarad.

A diplomamunkám témája egy Adaptív Kiterjesztett Valóság felület elkészítése, amely képes ezt a két hiányosságot kiküszöbölni. Egyrészt képes egy ismeretlen környezetet feltérképezésére, és berendezni virtuális objektumokkal, másrészt a berendezést intelligens módon képes elvégezni. Ez alatt azt értjük, hogy egy adott virtuális objektumhoz a leginkább hasonló valós tárgyat fogja hozzárendelni a munkatér objektumai közül.

A dolgozat során bemutatom egy prototípus rendszer elkészítését, melynek fő része egy keretrendszer, mely lehetővé teszi a legfontosabb funkciók indítását, beállítását és tesztelését. A munkám során több különböző algoritmust is implementáltam a főbb funkciók elvégzésére. Ezek közül a legfontosabbak a berendező funkció megvalósítása egy tanuló alakfelismerő algoritmus segítségével, valamint a valós tárgyak követésének implementációja egy hat szabadságfokú képjellemző-alapú algoritmus segítségével, mely az interakció megvalósítását szolgálja.

A dolgozat során következő fejezetében részletesen specifikálom az elvégzendő feladatot, ezt követően a 3. fejezetben pedig bemutatom más műhelyek a téma szempontjából releváns eredményeit. A 4. fejezetben részletesen leírom az implementáció kérdéseit, külön tárgyalva a keretrendszer létrehozását, illetve a benne felhasznált algoritmusok megvalósítását. Az ezt követő fejezetben a keretrendszer, illetve az algoritmusok tesztelését mutatom be, a 6. fejezetben pedig összefoglalom a dolgozat tartalmát, valamint a továbbfejlesztési elképzeléseimet mutatom be.

## 2 A feladat specifikációja

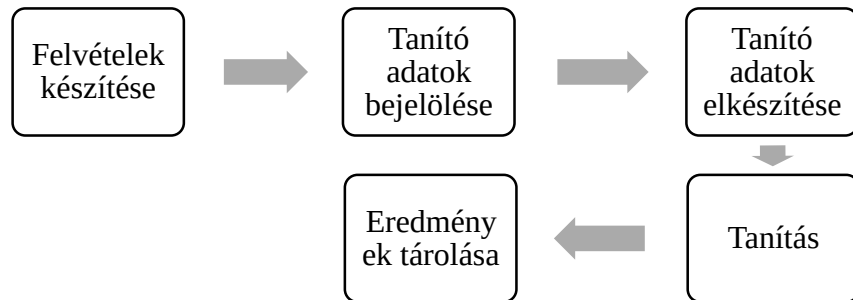
Ebben a fejezetben a diplomamunkám során elvégzendő feladatok részletes ismertetését végzem el. A feladatom egy az előző fejezetben felvetett rendszer prototípusának elkészítése volt, amely egy fix sztereó kamerapár segítségével térképezi el az előre ismeretlen jelenetet. Az elvégzendő feladatok két fő részre oszthatóak: az egyik egy, a rendszer használatához szükséges keretrendszer létrehozása, míg a másik ennek a keretnek a feltöltése a korábbi fejezetben leírt működést biztosító algoritmusokkal.

A keretrendszer alapvető feladata, hogy biztosítsa a hozzáférési felületet a rendszerhez, valamint, hogy az egyes algoritmusok futását vezérelje, így biztosítva a rendszer alapvető működését. Ehhez elérhetővé kell tenni a rendszer adminisztrációs funkcióit, például új virtuális objektumok felvétele, vagy az alakfelismerés tanítása, valamint az egyes algoritmusok tesztelését is. További fontos funkció az egyes algoritmusok paramétereinek beállítása és hangolása is, melyet szintén innen kell elvégezni.

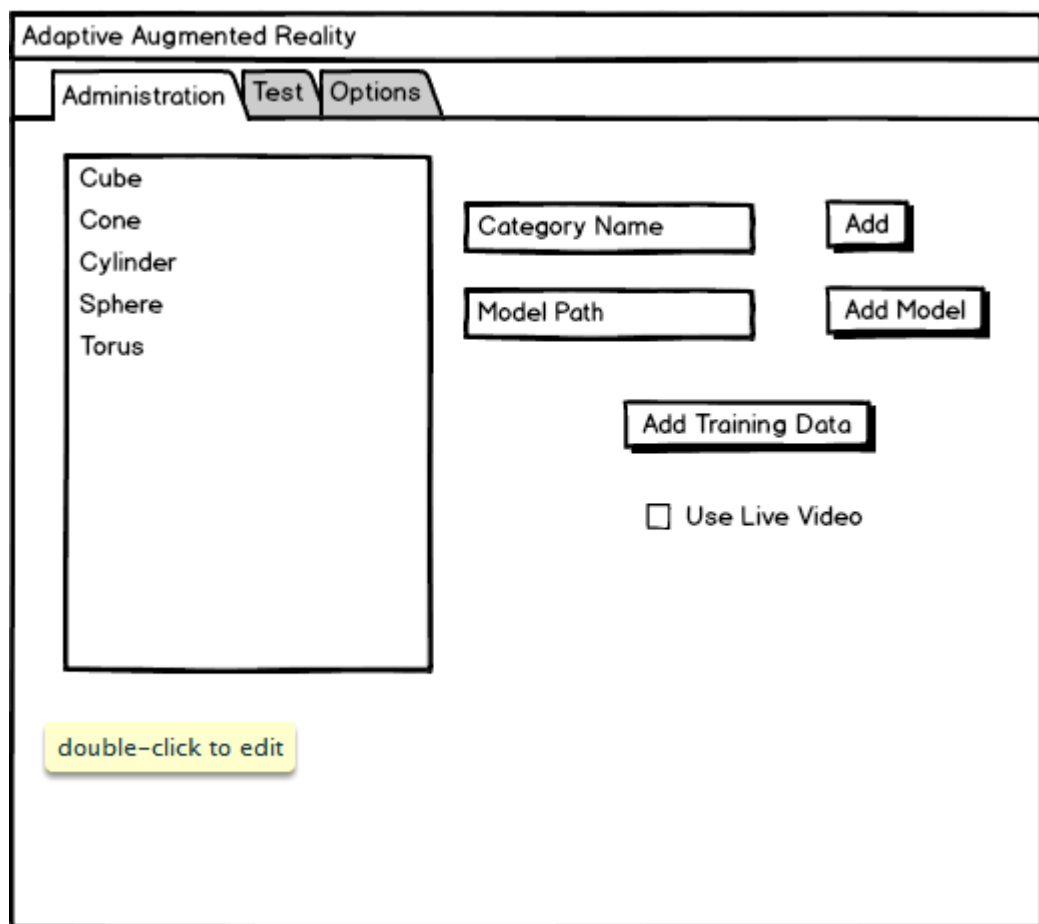
Ezeket a feladatokat négy fő funkcióba tömöríthetjük: Az első a virtuális környezet használat előtti adminisztrációja, melyet nem a végfelhasználó, hanem a környezet készítője végez el. Itt a legfontosabb részfeladatok a virtuális objektumok létrehozása, a megjelenésük megadása, valamint a tanuló alakfelismerés használata esetén az őket felismerni képes algoritmus tanítása. A második feladat a valós környezet feltérképezése, leírása, és berendezése virtuális objektumokkal. A harmadik a virtuális környezet folyamatos futtatása, mely az interakció megvalósításából és a kijelzésből áll. Az utolsó feladat a felhasznált algoritmusok paramétereinek beállítása és perzisztens tárolása.

Az adminisztrációs feladatok három fő részfeladatra oszthatóak: Új objektumkategória hozzáadása, 3D modell hozzárendelése a kategóriához, valamint tanító adatok hozzáadása a kategóriához. Ez utóbbi feladat egy kiemelten fontos része a munkának, így érdemes további részfeladatokra bontani (2.1 Ábra): Tanításra használt felvételek készítése, tanító adatok készítése, a tanítás elvégzése, és az eredmények perzisztens tárolása. Mivel ezt a feladatot nem a felhasználó végzi, ezért nincs szigorú

követelmény a futási időre vonatkozóan. Az adminisztrációs felület képernyőterve a 2.2 Ábrán látható.



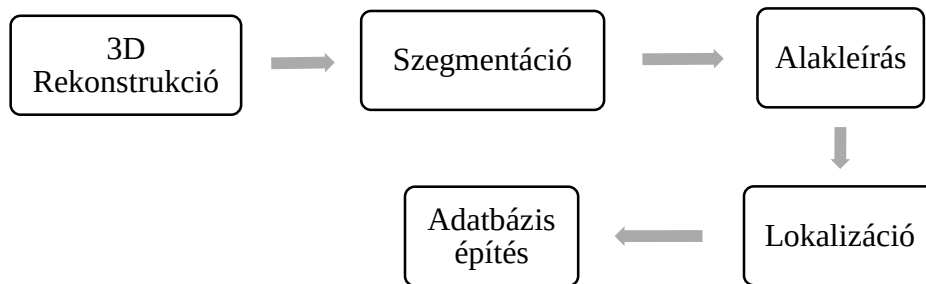
2.1 Ábra: A tanítás részfeladatai



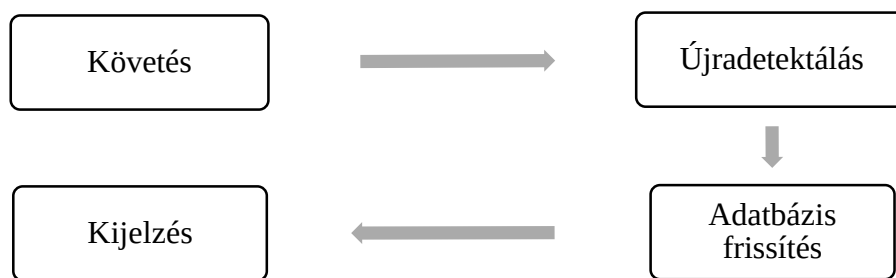
2.2 Ábra: Az adminisztrációs felület képernyőterve

A feltérképezési feladatot már a felhasználó indítja, azonban ezt a műveletet csak egyszer, a virtuális környezet indításakor fogja elvégezni, éppen ezért nincs kritikus követelmény a futási időre vonatkozóan. A felhasználói élmény megőrzésének érdekében azonban nem engedhetjük, hogy ez a lépés néhány (~10) másodpercnél tovább fusson. A berendezési feladat fő lépései (2.3 Ábra) az alábbiak: A kamerák képei alapján egy 3D

rekonstrukció elvégzése, melynek eredményképp előáll egy pontfelhő a valós jelentből. Ezt követően ebben a pontfelhőben szegmentálást végzünk, melyet egy alakleíró lépés követ. Ez után elvégezzük a lokalizációs lépést, melynek segítségével hozzárendelhetjük a virtuális objektumokat a jelenetben található tárgyakhoz. Végezetül pedig a rendszer későbbi futását előkészítendő egy követési adatbázist építünk a kiválasztott placeholder tárgyakból.



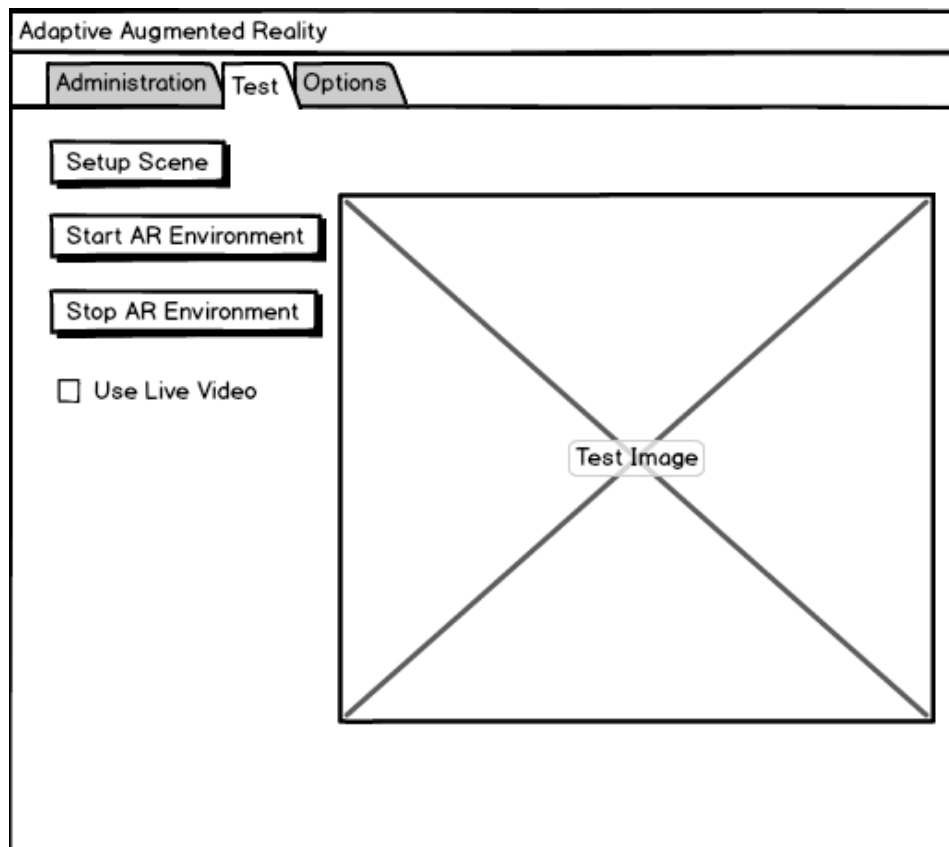
**2.3 Ábra: A berendezés lépései**



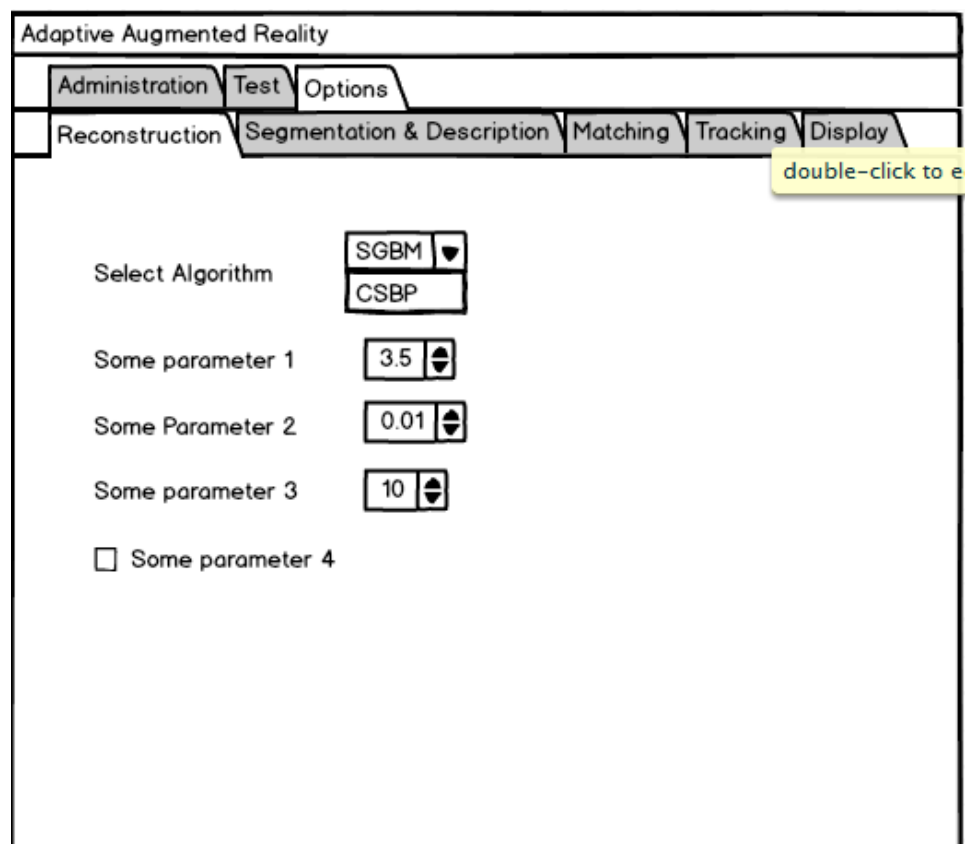
**2.4 Ábra: A környezet fenntartásának lépései**

A virtuális környezet futását a felhasználó indíthatja el a berendezés sikeres elvégzését követően. Ez a művelet már kritikus futási idő kritériummal rendelkezik, ugyanis valósidejű futást követelünk meg tőle. A futás során az interakciót és a kijelzést biztosító fő ciklus (2.4 Ábra) minden képkockára lefut. Az első lépése a valós tárgyak hat szabadságfokú követése, az objektumok elfordulása esetén pedig az előző funkció által előállított adatbázis frissítése. A második, és egyben utolsó lépés a kijelzés megoldása. Fontos követelmény még, hogy a rendszer legyen képes a követett objektum elvesztése esetén azt újra detektálni az adatbázis alapján.

A specifikációhoz még két további képernyőterv tartozik, a tesztelés, illetve a beállítások felületekhez. Az előbbi (2.5 Ábra) a berendezéshez, valamint a futáshoz szükséges funkciókat foglalja magába, az utóbbi (2.6 Ábra) pedig a beállítások funkció megvalósítására szolgál.



2.4 Ábra: A tesztelési funkció képernyőterve



2.5 Ábra: A beállítások funkció képernyőterve

## 3 Előzmények

Az alábbi fejezetben bemutatom diplomamunkám megvalósítása szempontjából releváns szakirodalom legfontosabb eredményeit. Az első alfejezetben bemutatom a tapintható kiterjesztett valóság alapvető kérdéseit és tulajdonságait, valamint a legfontosabb rendszereket. Ezt követően a sztereó rekonstrukció legelterjedtebb algoritmusait ismertetem. A fejezet harmadik részében a háromdimenziós alakfelismerés és az alakleírás alapvető kérdéseivel foglalkozok, a negyedik alfejezetben pedig a gráfokon történő tanulás problémakörére térek ki. Végezetül bemutatom az objektumok követésére használt módszerek közül a legfontosabbakat.

### 3.1 Kiterjesztett valóság

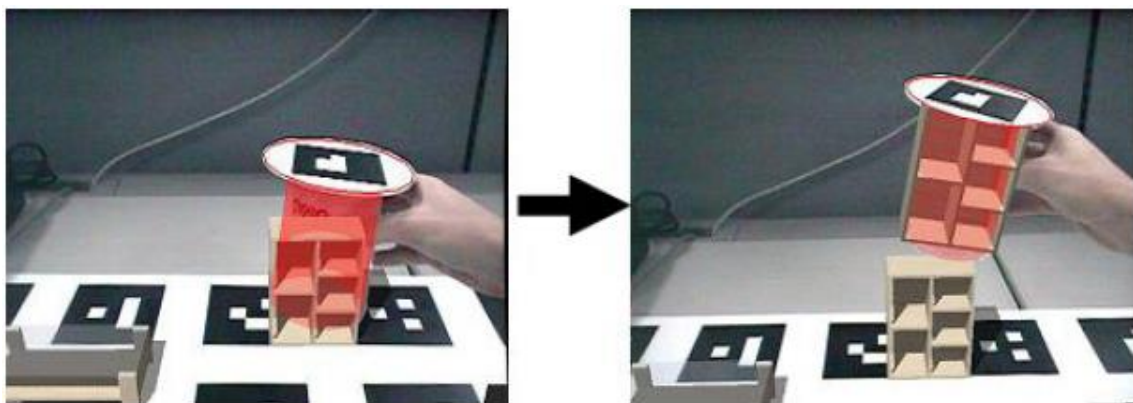
A kiterjesztett valóság rendszerek körében jelentős kutatási terület a különböző beviteli eszközök alkalmazása. Számos példát találhatunk radikálisan különböző rendszerekre, léteznek mobil alkalmazások [2], melyek elterjedését az okos mobil eszközök térhódításának köszönhetünk, gesztusvezérelt megoldások [3], lehetséges továbbá kitüntetett beviteli eszközök (pl.: toll, egyéb mutatószerszám [4]) alkalmazása. Érdeemes megemlíteni megoldásokat, melyek valós objektumokat is aktívan felhasználnak a virtuális környezet kialakításához: Léteznek rendszerek, melyek használata során virtuális textúrát vihetünk fel a környezetben lévő objektumokra [5], vagy éppen rajzolhatunk rájuk. [6] Ezenél lényegesen újszerűbb megoldás az IllumiRoom [7] technológia, mely a nappali falát használja fel, hogy egy vetítő segítségével kiterjeszthesse a TV készülék képét egy nagyobb felületre, így növelve a bemerülés érzetet.

A tapintható kiterjesztett valóság rendszerek egy külön kategóriát képeznek a különféle beviteli eszközök között. A valós objektumok felhasználása egy rendkívül fontos előnyt biztosít a TAR rendszerek számára: ezeknek az objektumoknak a használata ugyanis gyakran rendkívül intuitív, könnyen megérthető és kezelhető felhasználói felületeket eredményez. [8] A TAR rendszerek alapvető hátránya azonban, hogy csak nagy nehézségek árán képesek érintést, vagy erő kifejtést igénylő felhasználói felület elemek létrehozására. [9] További hátrány lehet, hogy kevés könnyen kiemelhető jellemzővel rendelkező objektumok követése nehézkes lehet, így az ilyen objektumok

használata ronthatja a rendszer hatékonyságát. [8] Ebben az esetben célszerű hibrid felületek használata, melyek a tapintható felületeket más, eltérő eszköztárat alkalmazó módszerekkel ötvözik, így helyettesítve a tapintható felületeket az olyan területeken, ahol azon használata nehézkes. [8] Erre rendkívül jó példa az eltakarásra alapú felületek alkalmazása. [9] A Tapintható Kiterjesztett Valóság rendszerek főbb tulajdonságai az alábbiak szerint foglalhatóak össze [10]:

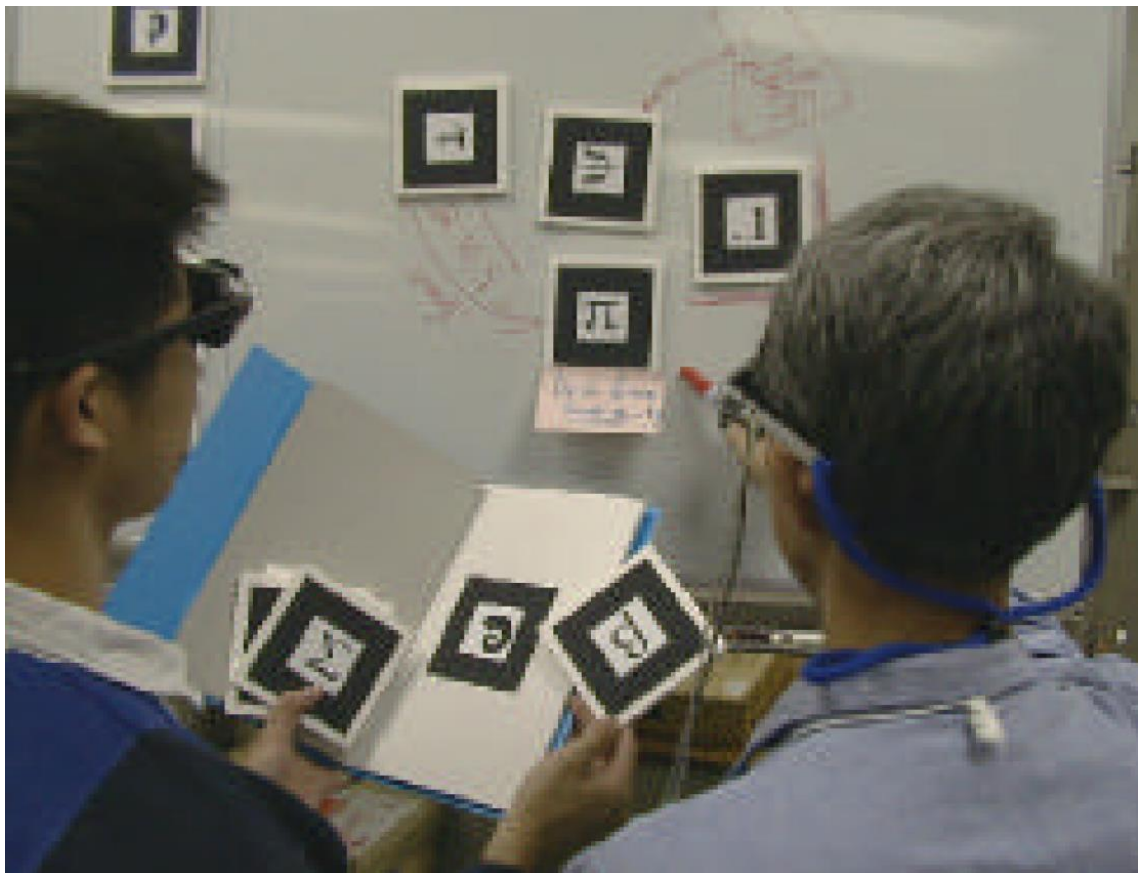
- A virtuális objektumok valós, fizikai objektumok segítségével manipulálhatók
- Térbeli interakciós megoldások használata (pl.: elforgatás, távolság)
- Több kezes használat támogatása
- Olyan objektumok kiválasztása, melyek fizikai tulajdonságai megfelelnek az elvégzendő feladat szempontjából
- Párhuzamos műveletvégzés támogatása több tárgy segítségével
- Több résztvevő kooperációjának támogatása

Kitűnő példa tapintható kiterjesztett valóság rendszerekre a MagicCup [10] felület. Ez a rendszer megfogható beviteli eszközként egy pohár alakú tárgyat használ, melyet a felismerhetőség segítésének céljából egy egyedi marker mintával láttak el. A felhasználók a virtuális objektumokat ezután ennek a pohárnak a segítségével manipulálhatják. Az erre használt művelet az „elfedés”, ami után az elfedett virtuális objektumot a pohár tartalmazni fogja, vagyis követni fogja a valós tárgy mozgását. Ez a rendszer kiváló példa a TAR rendszerek intuitív felületeire: Egy tárolóeszközbe való behelyezés és mozgatás rendkívül könnyen megérthető és tanulható művelet.



**3.1 Ábra: A MagicCup felület Forrás: [10]**

Egy másik érdekes felület a Tiles [11] rendszer, amely markerekkel ellátott kártyákat használ virtuális objektumok kezelésére. A kártyáknak két típusa van, az úgynevezett adat kártyák virtuális objektumok tárolására szolgálnak, míg az ezektől különböző operátor kártyák az egyes műveletek (másolás, törlés, stb.) elvégzését teszik lehetővé. A különböző virtuális objektumok egy könyvben találhatóak, innen lehet őket a kártyákra helyezni. A különböző kártyák interakcióját azok térbeli távolsága alapján valósítják meg, vagyis például egy adott virtuális objektum eltávolításához a törlő kártyát hozzá kell érintenünk a törölni kívánt objektumot tartalmazó adat kártyához.



**3.2 Ábra: A Tiles felület Forrás: [11]**

Az utolsó mélyebben bemutatott tapintható kiterjesztett valóság felület a virtuális kerekasztal (Virtual Round Table ) [12], mely bár egy meglehetősen régi megoldás, mégis releváns, mivel a saját elképzelésemhez rendkívül hasonlít. A VRT egy asztalon kíván létrehozni egy virtuális környezetet, az asztalon található, előre ismeretlen tárgyak segítségével. A működése során a felhasználók az egyes objektumokat kijelölhetik, és virtuális objektumokat tehetnek a helyükre. A rendszer innentől kezdve ezeket a valós tárgyakat követi, és a térbeli változásait a hozzájuk rendelt virtuális objektum másolni

fogja. Az általam elképzelt rendszerhez képest két fontos különbsége van a VRT felületnek. Egyrészt, a VRT a valós és virtuális objektumok összepárosítását nem automatikusan végzi, hanem a felhasználótól várja el a feladat megoldását. Ebből következően nem vesz figyelembe a rendszer semmilyen alakbeli hasonlóságot.



3.3. Ábra: A virtuális kerekasztal Forrás: [12]

## 3.2 3D képfeldolgozás

Az elkészítendő rendszer egyik legfontosabb alapfeladata az objektumok térbeli pozíciójának meghatározása, valamint az alakjukról valamiféle információ kinyerése. Ehhez szükséges egy rekonstrukciós eljárás választása. Ezek az algoritmusok két vagy több különböző nézőpontból készült képből próbálják minél nagyobb pontossággal visszaállítani a jelenet mélységét. A most következő részben leírt algoritmusok valamilyen módon az adott képpontok, vagy képrészek közötti távolságot – a diszparitást (disparity) - határozzák meg. Ahhoz, hogy ebből valóságos háromdimenziós információt nyerjünk ki a képek relatív pozícióját és orientációját ismerni, vagy becsülni kell.

Az első ilyen algoritmus a Semi-Global Matching [13], amely kölcsönös információk pixelenkénti párosításra, és egy globális 2D simasági kritériumra alapszik. A kölcsönös információt a két kép H entrópiájából számolhatjuk.

$$MI_{I_1, I_2} = H_{I_1} + H_{I_2} - H_{I_1, I_2} \quad (1)$$

HI1,I2 a kölcsönös entrópia, ami kicsi, ha az egyik képből könnyen megbecsülhető a másik, ami sztereó képpárnál igaz. Tehát minél jobban egyezik a két kép a kölcsönös információ annál nagyobb. Az algoritmus ezek után hierarchikus kölcsönös információ számítás segítségével ad egy kezdeti becslésből egy diszparitásképet.

Mivel a pixelenkénti költségfüggvények gyakorta találnak rossz párokat például a zaj miatt, ezért az eljárás során egy globális „energiafüggvényt” is alkalmaznak, mely a diszparitástól függ.

Egy másik eljárás az Optikai Áramlás (Optical Flow), amely egyenként becsli a képpontok elmozdulását állandó intenzitást feltételezve. Az optikai áramlás korlátja, hogy az elmozdulás iránya nem egyértelmű, mivel egy egyenletet kell két ismeretlennel megoldanunk kétdimenziós esetben. Ezért speciális feltételeket alkalmaznak a minél jobb meghatározásra. Ilyen feltétel például a közeli pontok „együttlmozgása”, aminek a figyelembe vételével olyan iteratív Optikai áramlás algoritmusokat kreálhatunk, melyek ezt a problémát sikeresen kezelik. Ha ezt az algoritmust sztereó rekonstrukcióra használjuk, akkor csoportosítani kell a hasonló eltéréssel rendelkező pixeleket, és a kiugró értékeket el kell vetni. [14] Az optikai áramlást gyakran használják mozgó kamerás rendszereknél (motion stereo), akár önkalibráló módon is. [15]

A harmadik eljárás a HBP [16] (Hierarchical belief propagation), amelyben a képpontok üzeneteket küldenek egymásnak egy négyszomszédságú képpont gráfban. Ezen üzeneteket a képpontok iteratívan számolják és segítségével egy globális költségfüggvényt minimalizálnak. Az algoritmust egy leskalázott képen hajtják végre először, majd egyre finomabb képen használják, kiindulási értéként felhasználva az előző hierarchia szinten becsült diszparitást, így jelentősen meggyorsítják a konvergenciát. A HBP algoritmusok általában jelentős mennyiségű memóriát használnak, de ez csökkenthető, valamint az algoritmus jól párhuzamosítható.

Végezetül megemlítendő a képjellemzőkre épülő (feature-based) sztereó rekonstrukció. Ezen eljárások lényege, hogy valamilyen eljárás segítségével a képeken található jellemzőket megkeressük, és későbbi azonosítás céljából leírjuk. Ezután a két képen egymástól függetlenül megtalált képjellemzőket a leírók alapján összepárosítjuk. Így a képjellemzők távolságából meghatározhatjuk a diszparitását az adott jellemző alatt. Az ilyen algoritmusok fontos jellemzője, hogy a megtalált jellemzőpárookra több követelményt állít. Ezek a követelmények lehetnek: [17]

- Epipoláris megkötés: A jellemzőpár helyezkedjen el a kamera epipoláris egyenes mentén. Ehhez az epipoláris egyenest ismerni, vagy becsülni kell.
- Kontinuitás megkötés: A diszparitás gyakran ugrik, ezért preferáljuk a környezethez hasonló találatokat.
- Egyediség megkötés: Csak egy találatot engedélyezzünk.
- Sorrend megkötés: Két képjellemző egymásutánisága az egyenes mentén ne változzon a másik képen!
- Eltakarás megkötés: Egy ugrás az egyik képen megfelel egy kitakarásnak a másikon.
- Maximum diszparitás: Előzetes ismeretek alapján becsülhető.
- A diszparitás irány menti deriváltja: A gyakorlatban általában megadható egy limit.

A következő fontos részfeladat a szegmentálás. A szegmentálás segítségével elkülöníthetjük egymástól a különböző objektumokat, hogy azokat egyesével vizsgálhassuk. Az objektumok szegmentálása az ember számára sem egyszerű feladat, főképp, mivel az eredménye nem önkényes: Mit tekintünk két külön objektumnak, és mit nem? További nehézség, hogy a kizárólag megjelenés alapján dolgozó szegmentáló algoritmusok nem megfelelően eredményesek a fejlesztendő rendszerhez. Éppen ezért csak olyan algoritmusokkal foglalkozok, amik valamilyen más információ alapján dolgoznak.

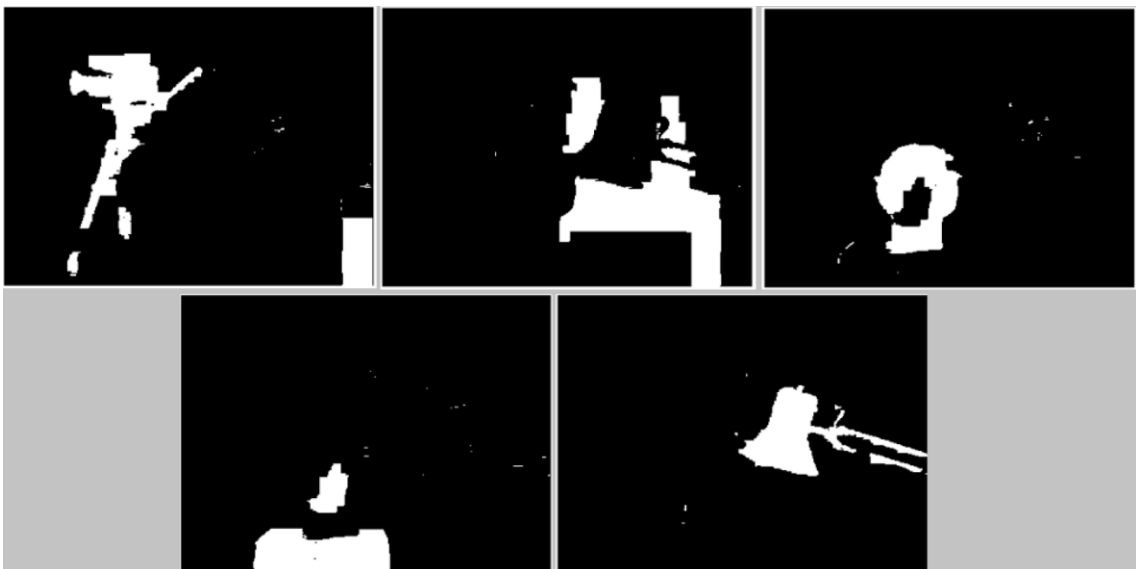
Az egyik lehetőség az előző feladatban előállított diszparitás kép alapján elvégezni a szegmentálást. Az elgondolás az, hogy a térben közel lévő objektumrészeket egy egységnek tekintjük, így ezeket el tudjuk választani egymástól.

Egy ilyen algoritmust használ Kounalakis és Triantafyllidis: [18]

- A diszparitás képet rétegekre bontjuk
- Minden rétegre meghatározzuk, hogy hány képpont tartozik hozzá
- Ha egy rétegbe egy előre megadott határnál több képpont tartozik, akkor az a réteg valószínűleg tartalmaz legalább egy objektumrészletet, innentől ezt jelölt rétegnek nevezzük. (3.4 Ábra)

- A kiválasztott jelölt rétegeket végignézzük, és összevonjuk azokat, amiknek a diszparitás értéke közel volt
- Az összevont rétegeken connected components algoritmussal megkeressük az objektumjelölteket
- Ha egy objektumjelölt elég nagy, akkor elfogadjuk

A fent leírt algoritmus képes objektumokat térbeli összetartozás alapján jó hatásfokkal szegmentálni, könnyen használható egymást kitakaró objektumokra. Előnye, hogy egyszerű és gyors. Hátránya, hogy a minősége függ a diszparitás kép minőségétől, valamint, hogy egymással összeérő objektumokat nem fog elválasztani egymástól. Ez probléma lehet asztalra helyezett objektumok esetén például, ahol az asztal helyét ismerni kell, és külön le kell választani az objektumokról, és csak ezután alkalmazni az algoritmust.



3.4 Ábra: A jelölt rétegek Forrás: [18]

### 3.3 Alakfelismerés

A gépi látás területén az objektumfelismerés, illetve ennek a forma alapján történő elvégzése kitüntetett jelentőségű probléma, melyre számos megoldás létezik. Ezen algoritmusok túlnyomó többsége azonban a felismerési feladatot kétdimenziós alak alapján végzi el, lévén, hogy a képfeldolgozás, illetve a gépi látás területén a képsíkon történő munkavégzés a jellemző. További nehézséget jelent, hogy ezen algoritmusok nem

terjeszthetők ki könnyen a három dimenziós esetre [19], a feladatomból szempontjából azonban ez szükséges lenne.

A fenti okból kifolyólag a háromdimenziós alakfelismeréshez külön ilyen esetre megalkotott eljárásokat kell használni. Erre az egyik lehetőség az alakjellemző alapú alakfelismerés használata. Ilyen eljárásra kiváló példa Golonivskiy et. al. [20] megoldása, ahol városi környezetről készült térbeli felvételeken végeztek alak alapú kategorizálást. Az eljárásuk első lépésében szegmentációt végeztek megkísérelve elválasztani egymástól a különálló objektumokat, így lehetővé téve az objektumjelöltek egyesével történő kategorizálását. Ezt követően minden egyes jelöltre kiszámoltak különböző formát leíró jellemzőket, és ezek alapján előre meghatározott szabályok alapján döntöttek. Ezen jellemzők között volt az objektum földől számított magassága, becsült térfogata, valamint a hozzá tartozó pontok száma. Külön jellemzőként használták a döntés során az egyes objektumok kontextusát is, mely bár nem alakjellemző, mégis nagyban segítheti a döntést.

Egy másik lehetőség alakfelismerés megvalósítására az alakzateloszlásra épülő kategorizálás. [19] Ennek az eljárásnak a lényege, hogy megkonstruálunk egy az objektumok formáját lehetőleg jól leíró alakfüggvényt, majd a függvény mintavételezése segítségével egy eloszlást generálunk az adott objektumhoz. Ezt az eloszlást más objektumok eloszlásával, vagy egy referenciával összehasonlítva hozhatunk döntést. Alakzateloszlásra kitűnő példa az objektumból két véletlenszerűen kiválasztott pont euklideszi távolságának az eloszlása.

A két fent bemutatott alakfelismerő eljárásnak azonban egy komoly hátránya akad: Mindkettő megköveteli, hogy használatuk előtt az egyes objektumokat elkülönítsük, akár valamilyen a priori információ alapján, vagy szegmentáció segítségével. Az általam elképzelt rendszerben azonban az előbbi nem áll rendelkezésre, mivel semmilyen feltételezést nem teszek a valós környezetre vonatkozóan. A szegmentáció elvégzése részben ugyanezen megkötés miatt problémás. A szegmentálás lépése ugyanis egy rendkívül komplex, és a probléma természetéből eredően önkényes lépés, ugyanis a szegmentáció elvárt helyes eredménye nem egyértelmű. További probléma, hogy a legtöbb szegmentáló algoritmus megköveteli bizonyos igen erős feltételezések (pl.: az objektum színére, mintázatára, térbeli érintkezésére vonatkozó

megkötések) teljesülését, ami az általános, ismeretlen környezetben való működési követelménynek ellentmond.

Erre a problémára kielégítő megoldás lehet az, ha a szegmentáció lépésében még nem külön objektumjelölteket kísérek meg előállítani, hanem csupán az alakleírás szempontjából ideális „építőkövekre” bontom le a jelenetet. A konkrét objektumjelöltek előállítása, vagyis annak eldöntése, hogy mely építőelemek alkotnak egy logikailag összetartozó objektumot pedig majd az alakfelismerést végző algoritmus feladata lesz.

Nyitott kérdés még az építőelemek kiválasztása. Célszerűnek tűnik azonban primitív formák (sík, gömb, henger, stb.) használata, ugyanis a primitív alakzatok képesek egy adott jelenetben a nagy, meghatározó formákat, valamint azok részleteit is egyformán reprezentálni, szemben a legtöbb alakleíró megoldással, ahol ez egy előre meghozandó tervezési döntés.

Primitív formák keresésére egy pontfelhőben számos megoldás született az évtizedek során. Az egyik legelterjedtebb megoldás a Hough-transzformáció használata, melyet leggyakrabban egyenesek keresésére szoktak használni a kétdimenziós képsíkon, azonban kiterjeszthető több dimenzióra és formára is. A transzformáció működése során a felhőben található pontokat egy olyan térbe transzformálja, amelyet a keresett alakzat paraméterei feszítenek ki. Egy pont képe ebben a térben egy olyan felület lesz, mely magába foglalja az összes lehetséges alakzatot, melynek az konkrét pont része lehet. (Egy kétdimenziós pont képe az egyenes paraméterei által kifeszített térben például egy szinusz görbe.) Miután az összes pontot transzformáljuk a Hough-térbe, megkeressük azokat a pontokat, ahol sok pont képe metszi egymást, vagyis azokat a potenciális alakzatokat, amelyeknek sok pont lehet része. Ezt a folyamatot nevezzük Hough-szavazásnak. A Hough-transzformációs eljárás rendkívül jól működik egyszerű alakzatok esetén, azonban minél több paramétere van a keresett formának, az algoritmus teljesítménye és erőforrásigénye annál rosszabb lesz. [21]

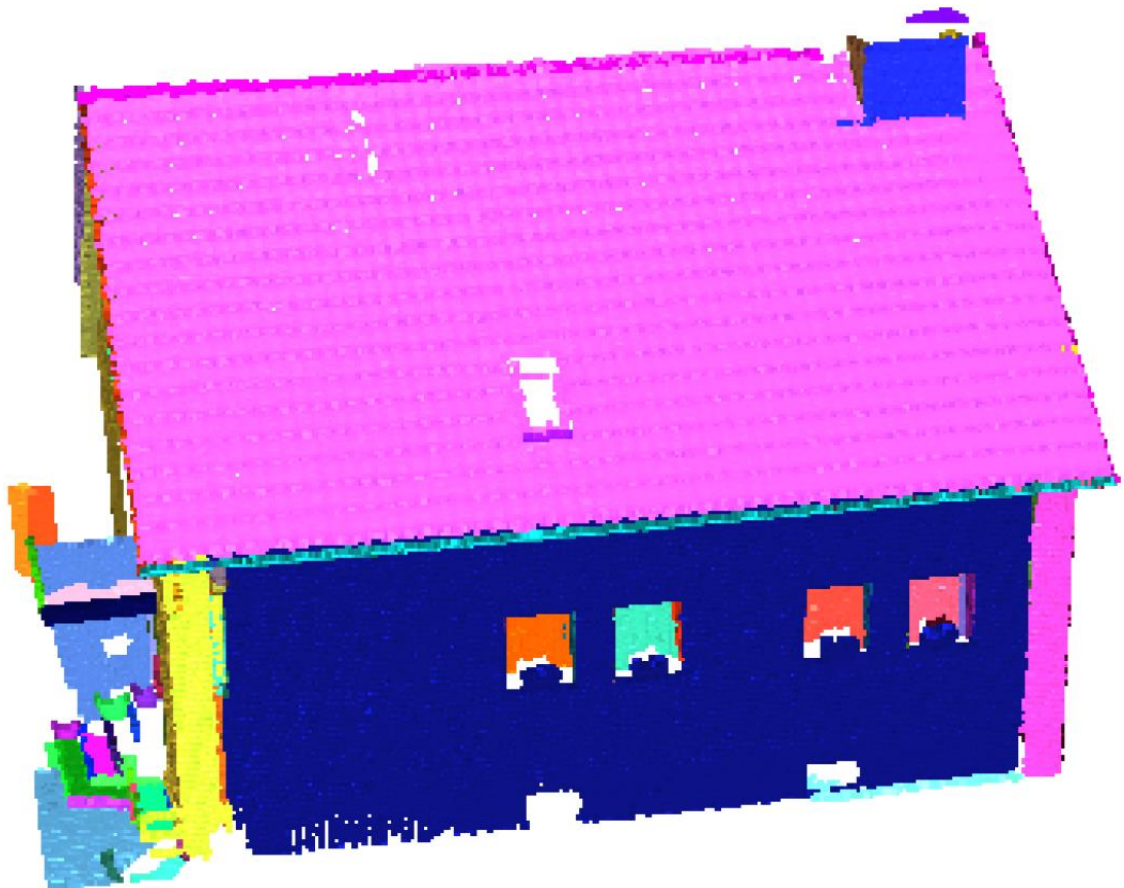
Különböző alakzatok pontfelhőben történő lokalizációjára kiváló algoritmus a RANSAC (Random Sample Consensus) [22] eljárás. A RANSAC működésének első lépése, hogy véletlenszerűen<sup>1</sup> kiválaszt a keresett forma leírásához éppen elegendő pontot

---

<sup>1</sup> Egyes variációk ettől eltérő technikákat is használhatnak.

a felhőből, majd meghatározza a kiválasztott pontokra ráillő alakzat paramétereit. Ezt követően még számos további jelöltet is előállít az algoritmus. A futás második lépésében az eljárás ezeket a jelölteket értékeli ki egyesével: Meghatározza, hogy a hátramaradó pontok közül mennyi illik rá az alakzatjelöltre, majd a ráillő pontok (inlierek) száma alapján kiválasztja a legjobb jelöltet. Az eljárásnak két jelentős előnye van [23]: Egyfelől, az alapvető működési elve egyszerű, hatékonyan megvalósítható, valamint a RANSAC algoritmus jelentős, 50%-nál nagyobb mértékű zaj előfordulásakor is képes az adott alakzatot megtalálni szemben más, hasonló módszerekkel.

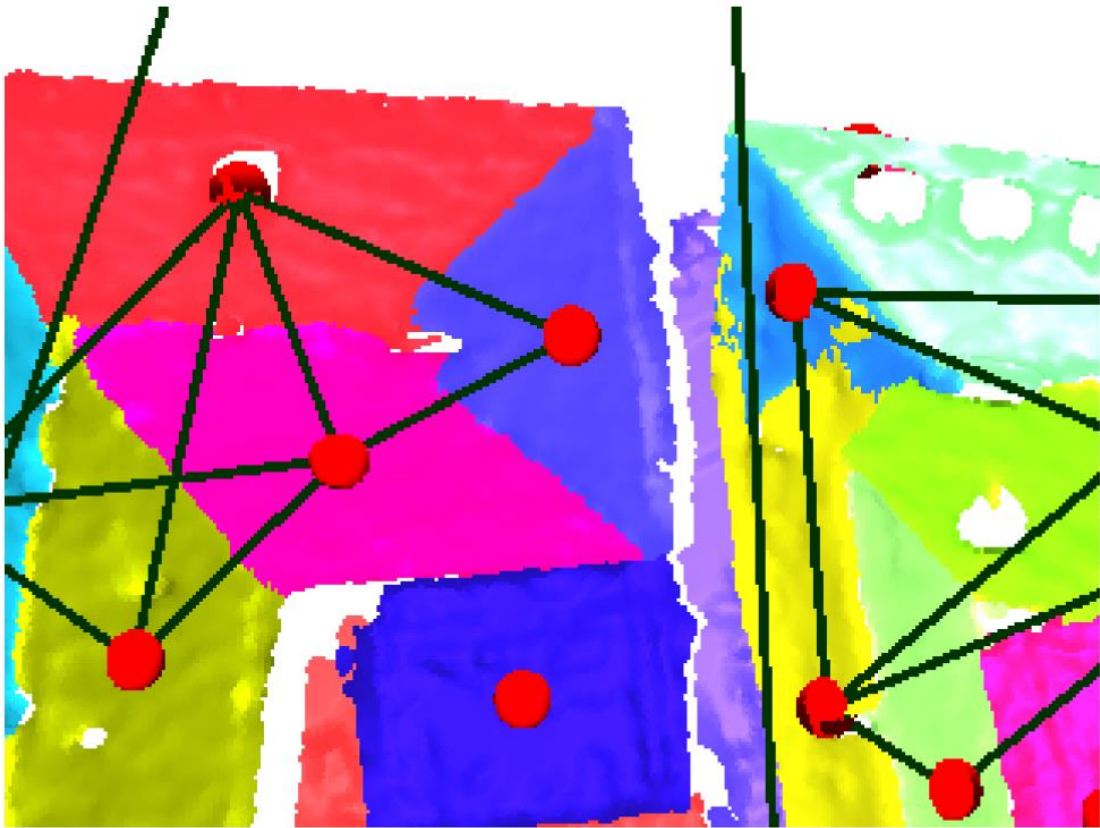
Schnabel, Wahl és Klein [23] kutatásuk során olyan továbbfejlesztését készítette el a RANSAC algoritmusnak, mely képes volt öt különböző primitívet (sík, gömb, henger, kúp és tórusz) keresni a bemeneti pontfelhőben. (3.5 Ábra) Az algoritmusuk fő előnye, hogy teljesen véletlenszerű mintavételezés helyett egy lokális mintavételező eljárást alkalmaznak, amely nagyobb eséllyel hoz létre jó objektumjelölteket. A lokális mintavételezés ezen felül erősíti az algoritmus szegmentáló jellegét az egész pontfelhőre értelmezett klasszifikáció helyett. Az algoritmusuk lépései a következők:



3.5 Ábra: Egy ház felbontva primitívekre Forrás: [23]

- Minden iterációban jelölt alakzatokat keres az algoritmus
- Az objektumjelöltek közül kiválasztja a legjobbat
- A legjobb jelölt alakzathoz tartozó pontokat törli a felhőből
- A felhőből törölt pontok miatt érvénytelené váló jelölteket elveti

A pontfelhő primitívekre történő felbontása után fontos lépés a jelenet formájának leírása. Erre egy kitűnő megoldás egy topológiai gráf megalkotása [24], melynek csomópontjai maguk a primitív alakzatok, míg a gráf élei a primitívek közti szomszédosságot adják meg (3.6 Ábra). Schnabel et. al. [24] octree struktúrát használt a primitívek szomszédosságának meghatározására.



3.6 Ábra: A konstruált topológiai gráf Forrás: [24]

A leírást követő lépés a leírás alapján történő alakfelismerés. Ehhez a ismertetett megoldás készítői algráf párosítást használtak, mely a csomópontok kis száma miatt kivitelezhető volt. Ehhez a keresett objektumkategóriához referenciagráfot hoztak létre, mellyel a felismerendő alakzatot írták le. Fontos kiegészítés volt, hogy a referenciagráfhoz hozzáadtak opcionális kontextus, vagy ismétlődő csomópontokat,

melyek segítettek bonyolultabb alakzatok felismerésében. Alapvető probléma azonban, hogy a topológiai leírás önmagában nem használható alakfelismerésre, mivel az egyszerűbb alakzatok gráfjai túlságosan általánosak lesznek a hatékony megkülönböztetéshez. [24]

A fenti problémának a megoldását különböző megkötések bevezetése jelentette. A megkötéseknek három fő típusa volt: csomópont, él, valamint gráf megkötések. A csomóponti megkötések tipikusan a csomópontok típusára, vagy egyéb tulajdonságára vonatkoztak (méret, arányok, stb.). Az él típusú megkötések az élekhez rendelt tulajdonságok korlátozására szolgáltak. Tipikus két szomszédos objektum távolságának, vagy bezárt szögének megkötését jelentette. A gráf megkötések leggyakrabban egymással nem kapcsolódó csomópontok viszonyának megszabásának eszköze volt. Tipikus példa gráf megkötésre két nem érintkező sík párhuzamossága. [24]

### 3.4 Tanuló rendszerek

Az előző fejezet végén bemutattam egy gráf párosításra épülő alakfelismerő algoritmust. Nagyméretű gráfok esetén azonban a brute force algráfkeresés futási ideje jelentősen menő, így nem feltétlenül alkalmazható a megvalósítandó rendszerben. Ráadásul az egyszerű párosításra épülő eljárás nem képes az öt megelőző lépések esetleges hibáit megfelelő flexibilitással kezelni, így célszerű lehet egy jobb adaptációs képességgel rendelkező párosító algoritmus használata (pl. tanuló megoldások). Az alábbi fejezetben ennek a lehetőségét fogom tárgyalni, belemerülve a gráfokon értelmezett tanuló algoritmusok irodalmának bemutatásával.

A gráfokon történő tanulás a gépi tanulás egy speciális területe, mely különös nehézséggel néz szembe. A legtöbb gyakorlatban bevált tanuló algoritmus ugyanis vektortereken értelmezett, és a gráfokra, vagy egyéb struktúrált objektumokra való átültetésük korántsem triviális feladat. A fejezet során két különböző módszert fogok részletesen bemutatni: Az egyik a gráfok vektoriális reprezentációjára épülő irányzat, míg a másik a kernel függvények alkalmazása.

A vektoriális módszerek célja, hogy a gráfból egy azt reprezentáló jellemző vektort konstruáljanak, majd ezt felhasználják a jellemző vektorokat használó tanuló algoritmusokban. Egy adott gráfból vektor konstruálásának legegyszerűbb módja a gráfhoz tartozó adjacencia mátrix vektorosítása, és konkatenálása az esetleges

csomóponti súlyokat tartalmazó vektorhoz. Ezzel a módszerrel viszont két kritikus probléma adódik. Az első, hogy a tanulás során felhasznált gráfok nem feltétlenül azonos méretűek, ami eltérő méretű vektorokat eredményez. Ez a nehézség még megoldható lenne a kisebb méretű gráfok nullákkal való kiegészítésével, azonban a másik probléma ezt nem teszi lehetővé.

A probléma ugyanis, hogy a gráfok struktúrált objektumok, amelyek invariánsak a csomópontok (önkényes) számozására, amely a vektor létrehozásához szükséges. Maga a megalkotott vektor viszont már változik a csomópontok számozásának változásával, az ebből származó többértelműséget a tanuló algoritmusok pedig nem tudják kezelni. Éppen ezért fontos a vektor számozásra invariáns megkonstruálása, ellenkező esetben minden felhasznált gráfhoz egy költséges gráfígazító algoritmust használata lenne szükséges. [25]

Az invariáns vektor létrehozására az egyik lényeges megoldás a gráfok spektrális reprezentációjának használata. [26]. A spektrális felbontás alapelve, hogy egy, a gráfot nem invariáns módon leíró reprezentáció mátrixának sajátfelbontása ( $A = VEV^T$ ) segítségével hozunk létre egy új, a csomópontok sorrendjére már invariáns reprezentációt. Ennek első lépése a sajátfelbontás során adódó sajátvektorok sorba rendezése a hozzájuk tartozó sajátértékek nagysága szerint, majd a sajátértékek valamint a megfelelő sajátvektorok konkatenálásával egy jellemző vektort hozhatunk létre. Fontos megjegyezni, hogy ez a módszer csak részleges invarianciát eredményez, így egy igazító algoritmus használatára még mindig szükség van. [26]

A spektrális reprezentációnak számos variánsa létezik, melyek a kiinduláshoz használt reprezentációs módszerben különböznek. [26] Az egyszerűbb változatok közé tartozik az, amikor a gráf adjacencia mátrixának sajátfelbontását használjuk, vagy valamivel bonyolultabb a gráf Laplace-mátrixát használni az adjacencia mátrix helyett. A Laplace mátrix  $L$  számítását a foksági mátrix  $D$  és az adjacencia mátrix  $A$  segítségével az alábbi egyenlet szerint számolhatjuk ki:

$$L = D - A \quad (2)$$

Egy valamivel komplexebb módszer a hőkernelre épülő eljárás, ami a Laplace mátrix dekompozícióját használja fel. A hőkernel szemléletesen a hő terjedését írja le a gráfban, amennyiben az egy valós struktúrát adna meg. [26]. A hőkernel fontos eleme az időparaméter  $t$ , amely segítségével megszabhatjuk, hogy a hőkernel a lokális vagy a

globális struktúrát vegye figyelembe nagyobb súllyal. A fent bemutatott módszerek közül Zhu et. al. [27] eredményei alapján a hőkernel biztosítja a legjobb leíróképességet. A hőkernelt az alábbi képlet segítségével számolhatjuk:

$$H = Ve^{-tE}V^T \quad (3)$$

A spektrális reprezentáció használatára kitűnő példa White [26] módszere, aki több spektrális reprezentáció keverésével hozott létre generatív tanuló algoritmust gráfokra. Az algoritmus generatív jellegének biztosításához egy a medián gráf meghatározására képes módszert használt [25]. Érdekes további eredményként megemlíteni, hogy gráf szegmentáló algoritmusok segítségével egy részgráfokra épülő, és azokból építkező generatív modellt hozott létre.

A másik fontos algoritmustípus, amely ebben a fejezetben bemutatásra kerül a kernel függvényekre épülő gráfokon történő tanulás. A kernel függvények használata először a vektorterekben történő tanulás körében került használatra, egy látszatra igencsak különböző probléma megoldására. A probléma az volt, hogy a tanuló algoritmusok használói szerették volna az egyes jellemzőkön felül azoknak a hatványait, vagy különböző szorzatait is felhasználni a döntés során, azonban ezek explicit felvétele a jellemzővektorba nagyságrendekkel megnövelheti a vektor méretét, ebből következően pedig a szükséges műveletek számát.

Erre a problémára jelentettek megoldást a kernel függvények, melyek olyan  $K(A, B)$  valósértékű, szimmetrikus<sup>2</sup>, pozitív szemidefinit függvények, hogy valamilyen módon az  $A$  és  $B$  jellemzővektorok hasonlóságát írják le. Bizonyítható, hogy a kernel függvények  $U = \Phi(A)$  és  $V = \Phi(B)$  vektorok skaláris szorzatát adják eredményül, ahol a két vektort  $A$  és  $B$  vektorokból konstruáltuk. Számos tanuló algoritmus leírható olyan alakban, hogy ne szerepeljen benne egy jellemző vektor sem explicit módon, hanem mindig csak egy másik vektorral skalárisan szorozva (az SVM algoritmus természetes módon ilyen alakban áll elő), ekkor viszont a képletben a skaláris szorzatokat a kernel függvénnyel helyettesítve elérjük azt, amit az előző bekezdésben megfogalmaztam, csupán lényegesen kevesebb számítással. Ezt nevezzük kernel trükknek.

---

<sup>2</sup>  $K(A, B) = K(B, A)$

A kernel függvények és a kernel trükk működéséhez azonban semmilyen esszenciális megkötés nincs  $A$  és  $B$  bemeneti paraméterekre, vagy azon formájára vonatkozóan. Ez azt jelenti, hogy  $A$  és  $B$  lehet vektor, vagy éppenséggel gráf, vagyis ha találunk egy függvényt, amely két gráfhoz hozzárendel egy valós számot, amely a két gráf hasonlóságát írja le, valamint, hogy erre a függvényre teljesül a kernel függvény definíciójának két feltétele, valamint invariáns a csomópontok sorrendjére, akkor ennek felhasználásával készíthetünk tanuló algoritmust.

A szakirodalomban számos gráf kernel létezik, ezek közül a véletlen séta kernel (random walk kernel) [28] fogom bemutatni. A kernel működésének alapvető gondolata, hogy a gráf csomópontjain véletlenszerű sétákat hajtunk vére, gráf éleinek súlyait úgy értelmezzük, mint adott élen történő továbbhaladás valószínűségét. Ezt követően elképzelhetjük, amint a két vizsgált gráfon egyszerre végzünk sétákat, és megszámloljuk, hogy hányszor végeztük el *ugyanolyan* sétát mindkét gráfon. Belátható, hogy hasonló szerkezetű gráfokon a megegyező séták száma nagyobb lesz, mint különböző gráfokon. [28]

Miután a véletlen séta kernelt szemléletesen bemutattam, ideje a matematikai leírást is megadni. Legyen  $U = \{Au, Nu\}$  és  $V = \{Av, Nv\}$  a két összehasonlítandó gráf, az adjacencia mátrixaikkal, illetve a csomóponti vektoraikkal megadva. Meghatározhatjuk a gráfok direkt szorzat gráfját:  $UV = \{Auv, Nuv\}$

$$Auv = Au \otimes Av \quad Nuv = Nu \otimes Nv \quad (4)$$

Ahol  $\otimes$  a mátrixok Korenecker szorzata. Fontos észrevenni, hogy a két bekezdéssel korábban leírt együttes sétálás matematikai módon reprezentálva nem más, mint a direkt szorzaton elvégzett véletlen séta. Így a kernel függvény végeredményét úgy számolhatjuk ki, hogy a direkt szorzaton megtett  $n$  hosszú séták számait összegezzük ( $n$  futó változó). Érdeemes megjegyezni, hogy ehhez a séta elkezdésének és befejezésének valószínűségét is definiálnunk kell, amire a direkt szorzat csomóponti vektorait használjuk fel. Csomóponti súlyok hiányában ez lehet egyenletes eloszlású, vagy más a priori információra épülő eloszlás. A véletlen séta végső képlete az alábbi:

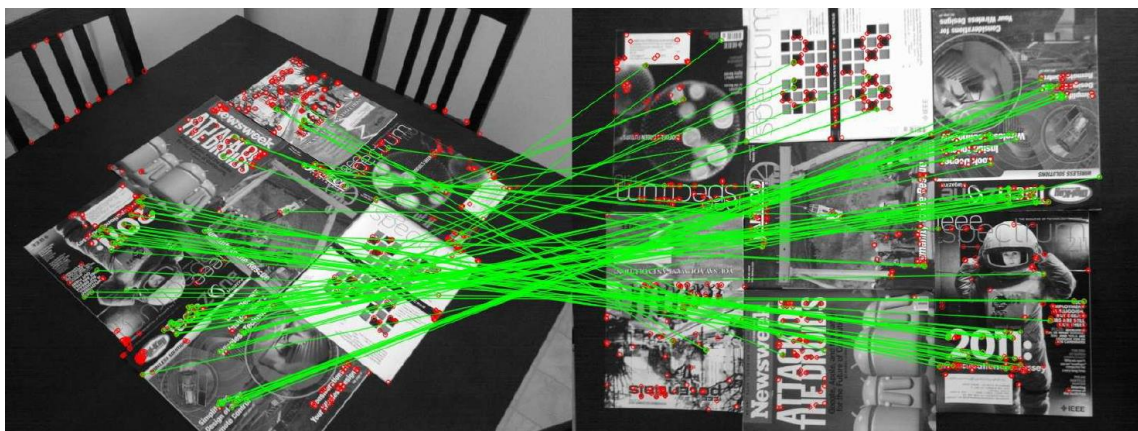
$$K = \sum_{n=1}^N W_n Nuv^T Auv^n Nuv \quad (5)$$

Ahol  $W_n$  a relatív súlya az  $n$  hosszú sétának. A véletlen séta kernel legfőbb erénye, hogy invariáns a csomópontok sorrendjére, így sikeresen képes a fejezet elején felvetett problémának orvosolni, szemben a vektoriális módszerek részleges invarianciájával. A kernel hátránya azonban a nagy számítási igény, amely a gráfok méretével, így a direkt szorzat adjacencia mátrixának méretével köbösen nő.

### 3.5 Objektumkövetés

A gépi látás problémakörének rendkívül fontos eleme az objektumkövetés, melynek számos ipari és kutatási alkalmazása van, melyek közül természetesen a kiterjesztett valóság rendszerekben történő alkalmazás a jelen munka szempontjából kitüntetett jelentőséggel bír. Az AR felületek három különböző követési módszert használhatnak. [8] Ezek közül elsőként a szenzorokra alapuló objektumkövetést említeném meg. Ez alapvetően nem gépi látásra alapul, hanem különböző nem kép alapú fizikai (optikai, akusztikus, mágneses, stb.) követést jelent. Második lehetséges módszerként a vizuális követést említeném meg, amely egy, vagy több kamera segítségével végzi el a követést. Ezen a kategórián belül megkülönböztethetjük a mesterséges, valamint a természetes jellemzőkre (3.7 Ábra) épülő követést. A harmadik lehetséges módszer a hibrid megoldású követés, mely az előző megoldások ötvözése.

A felsorolt követési módok közül a természetes vizuális követés kivételével mindegyik módszer igényli akár a követendő objektumok, vagy a munkatér előkészítését (szenzorok felszerelése), ami a megvalósítandó rendszer alapkövetelményeinek ellentmond, így az alábbi fejezetben kizárólag a természetes vizuális követéssel, valamint a követés és detektálás általános kérdéseivel fogok foglalkozni.



3.7 Ábra: Detektálás természetes képjellemzők alapján Forrás: [32]

A természetes képjellemzők alapján történő követéshez és detektáláshoz fontos ezekből a képjellemzőkből egy adatbázist építeni, és a jellemzőket az egyes objektumokhoz az adatbázison belül hozzárendelni. A különböző objektumok elkülönítésére használja Kounalakis és Triantafyllidis [18] a rétegekre alapuló 3D szegmentálást, amit korábban már bemutattem. Amint az objektumokat elválasztották egymástól, képjellemzőket keresnek rajtuk, majd ezeket egy neurális háló tanítására használják.

A természetes vizuális követés legegyszerűbb módja, hogy a keresendő objektumokat minden képkockán detektáljuk. A detektálás azonban egy meglehetősen lassú művelet, így ez a megoldás nem célszerű. A követés, vagyis az előző megtaláláskor feljegyzett jellemzők alapján történő keresés lényegesen egyszerűbb és gyorsabb, azonban ilyenkor fellép az akkumulálódó hibák problémája. A újabb képkockákon történő hibák hozzáadódnak az eddigi hibához, így a követés pontossága egyre csökken. Erre kézenfekvő megoldást nyújt, ha az objektumokat csak elvétele detektáljuk, a közbenső képkockákon csak követést végzünk, így az algoritmus segessége és pontossága is elfogadható lesz. [29]

Centrális témája a természetes vizuális követésnek a konkrét természetes képjellemzők típusának és detektálásának kérdése. Követésre használhatunk például természetes éleket, vagy sarokszerű pontokat. A legelterjedtebb, és leginkább robusztus megoldás azonban a sarokszerű pontokat körülvevő textúrált területek használata. Az ilyen képjellemzők, robusztus, fényerő, skálázásra és elforgatásra invariáns detektálása és leírása népszerű kutatási téma volt az utóbbi évtizedben. A jelenlegi irányzat kezdetét a SIFT (Scale Invariant Feature Transform) algoritmus jelentette, melyet Lowe [30] alkotott meg. A lépései az alábbiak:

- Az algoritmus első lépésében a képet különböző mértékben skálázott Difference of Gaussians (DoG) szűrők segítségével szűrjük, majd a háromdimenziós térben (x, y, skálafaktor) maximumot keresünk, így kiemelve a könnyen elismerhető részeket.
- A detektált sarokpontok közül a stabilitásuk (sarokság, kontrasztosság) alapján kiválasztjuk a legjobbakat, és feljegyezzük a skálafaktorukat is.

- A következő lépésben következetes módon orientációt rendelünk a képponthoz a lokális gradiens alapján. Az ezt követő műveletek a képjellemzők orientációját és skálafaktorát figyelembe veszik, biztosítva az invarianciát.
- A kulcspont környezetén kiszámoljuk a lokális gradienseket, és ezek segítségével egy leíró állítunk elő.

A SIFT óta számos új hasonló alapelvű és működésű algoritmus jelent meg, például a SURF [31], mely jelentős gyorsítást hozott csekély robusztusság csökkenés árán, valamint az ORB [32], és a BRIEF [33], melyek nagyságrendbeli gyorsítást hoztak, azonban mindezt jelentős áldozatok árán az ismétlődőképességet tekintve. Jelentősen újabb eredmény a KAZE [34] algoritmus, illetve annak gyorsított változata az AKAZE [35], melyek nemlineáris skálateret használnak a könnyen leírható pontok keresésére. Az AKAZE algoritmus nagyságrendbeli gyorsítást jelent a SIFT-hez képest, azonban a hatékonyság szempontjából is képes az elődjét elérni, illetve esetenként felülmúlni.

## 4 Megvalósítás

A jelenlegi feladatban a diplomatervem tárgyát képező prototípusrendszer létrehozásának gyakorlati kérdéseit vizsgálom részletesen. A fejezet első részében a megvalósítandó keretrendszer megvalósításával foglalkozom, külön kitérve a funkcionális követelményekre, valamint a követelményeket hatékonyan megvalósítani képes osztályszerkezet kialakítására. Az alfejezet végén az implementációhoz közvetlenül kapcsolódó kérdésekre is. A fejezet második felében kitérek a keretrendszer megfelelő részegységeinek algoritmusokkal történő feltöltésére, amelyben részben az előző fejezetben bemutatott munkákra és elvekre építkeztem. Külön részletesen bemutatom a rekonstrukcióra, a szegmentálásra, az alakleírásra és –felismerésre, valamint a követésre létrehozott módszereket.

### 4.1 A keretrendszer

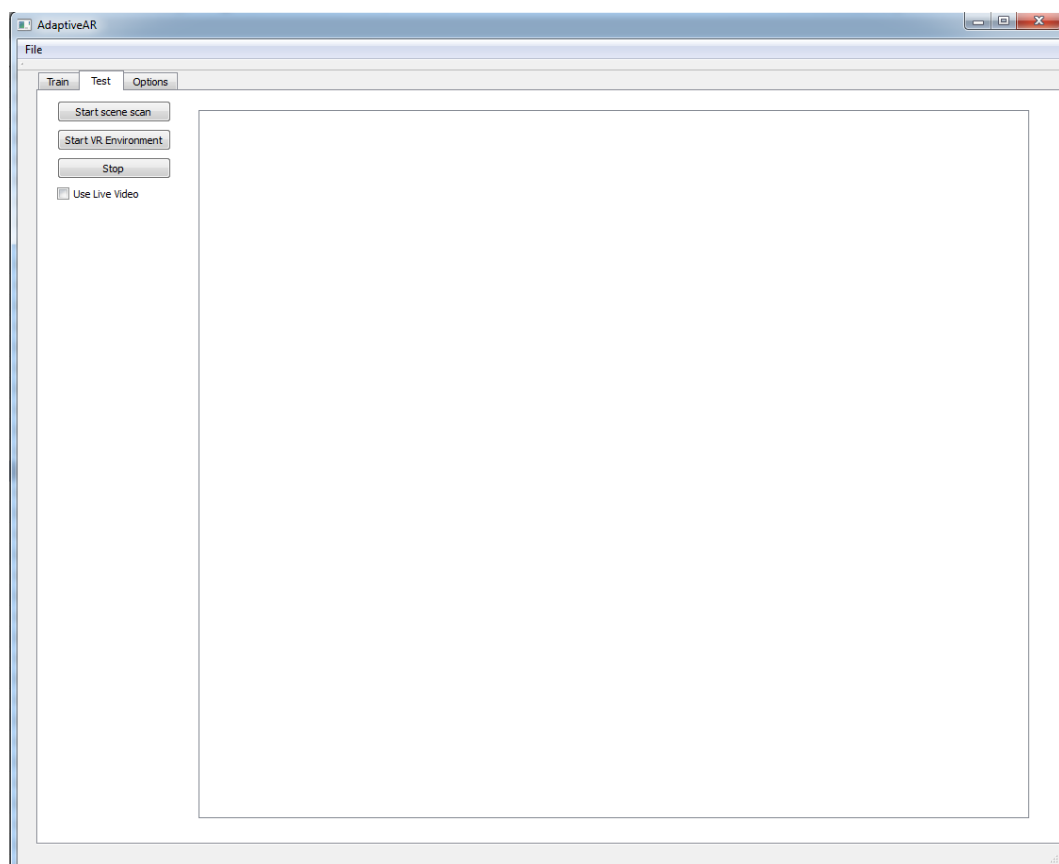
A keretrendszer implementálásánál az első fontos döntés a platform választása volt. Itt én nem szerettem volna egy adott operációs rendszerhez kötni az alkalmazást, ezért a grafikus felület kialakítására a Qt könyvtárat használtam, mert számos operációs rendszer alatt elérhető. Magát a felületet a Qt Creator környezetben készítettem, míg az elemek felprogramozását Visual Studio alatt végeztem. A Visual Studiohoz készült Qt plugin segítségével a két fejlesztőkörnyezet könnyedén együtt tudott működni.

A keretrendszer grafikus felülete három részre osztható fel: Az első a tanító nézet, a második a teszt nézet, a harmadik pedig a beállítások nézet. Tanító nézetben (4.1. Ábra) a rendszerbe felvett virtuális objektumok listáját láthatjuk, ezekhez adhatunk hozzá, adhatunk meg nekik virtuális modellt, ami a valós jelenetben történő megjelenítéshez szükséges. Az Add Training Set gomb segítségével a kiválasztott objektumhoz tanító adatokat adhatunk hozzá. Ekkor ki kell választanunk a tanításra használt képek forrását, majd egy felugró ablakban a megfelelő objektumot be kell kereteznünk (4.2. Ábra), a képek elfogyta után a tanítás elindul. Ezt meg lehet csinálni élő videó segítségével is, de ebben az esetben egyszerre csak egy objektumot taníthatunk.

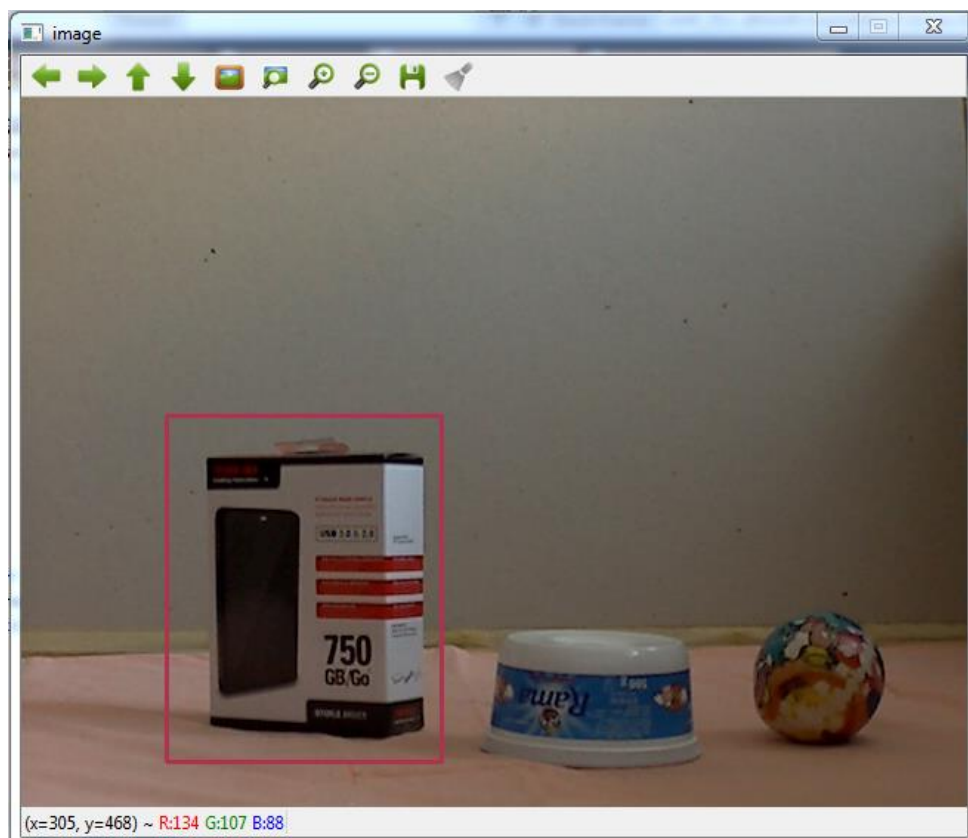
A teszt nézet (4.3 Ábra) nagy részét a végső jelenetet megjelenítő ablak foglalja el. Itt is természetesen kiválaszthatjuk, hogy a tesztet élő, vagy előre felvett videó

segítségével kívánjuk elvégezni. Innen indíthatjuk a Start scene scan gomb segítségével az objektumfelismerő algoritmust, amely a tesztképen látható jelenetet berendezi virtuális objektumokkal. A Start VR Environment gomb segítségével a már berendezett jelenetben elindítjuk a követést, és a rendszer megkezdí a működést.

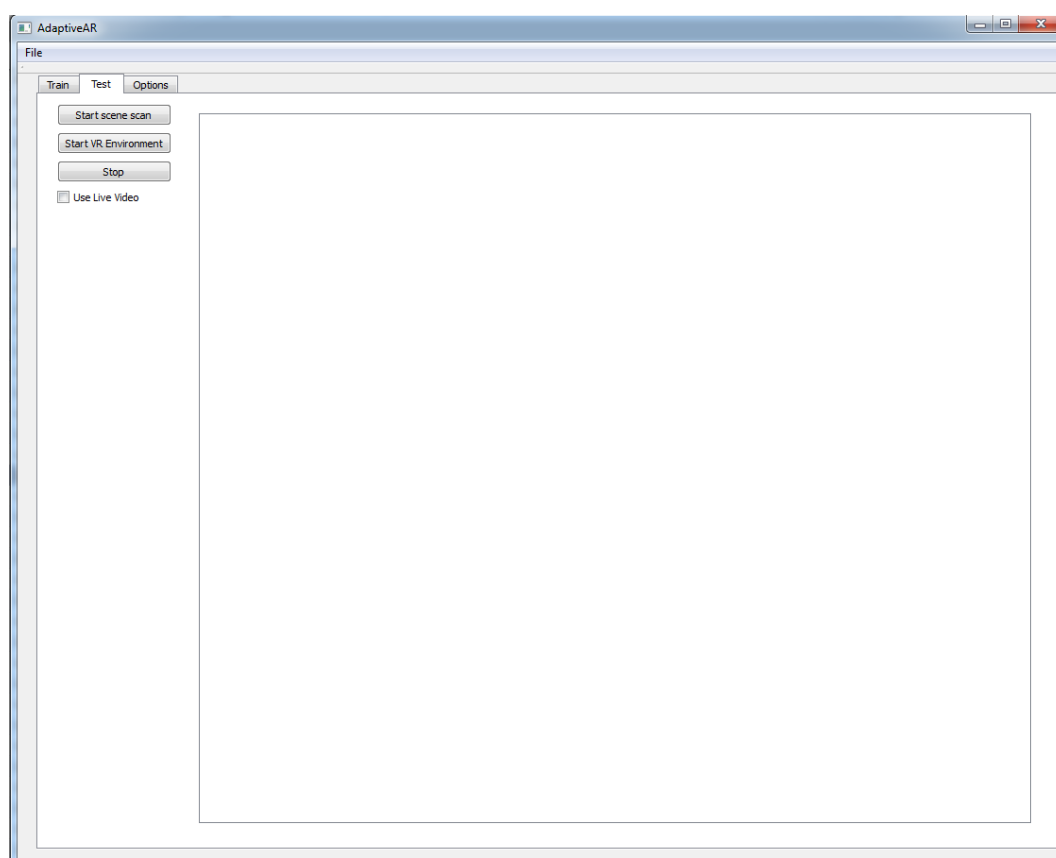
A beállítások nézetben (4.4 Ábra) a különböző algoritmusokkal kapcsolatos beállításokat láthatjuk és változtathatjuk meg. Ide tartoznak a konkrét algoritmusok és azok paramétereinek kiválasztása, valamint a kijelzés egyes paramétereinek a megválasztása. Szintén itt adhatjuk meg az élő videó forrásának címét. További lényeges megoldandó feladat itt az adott opciók perzisztens tárolása.



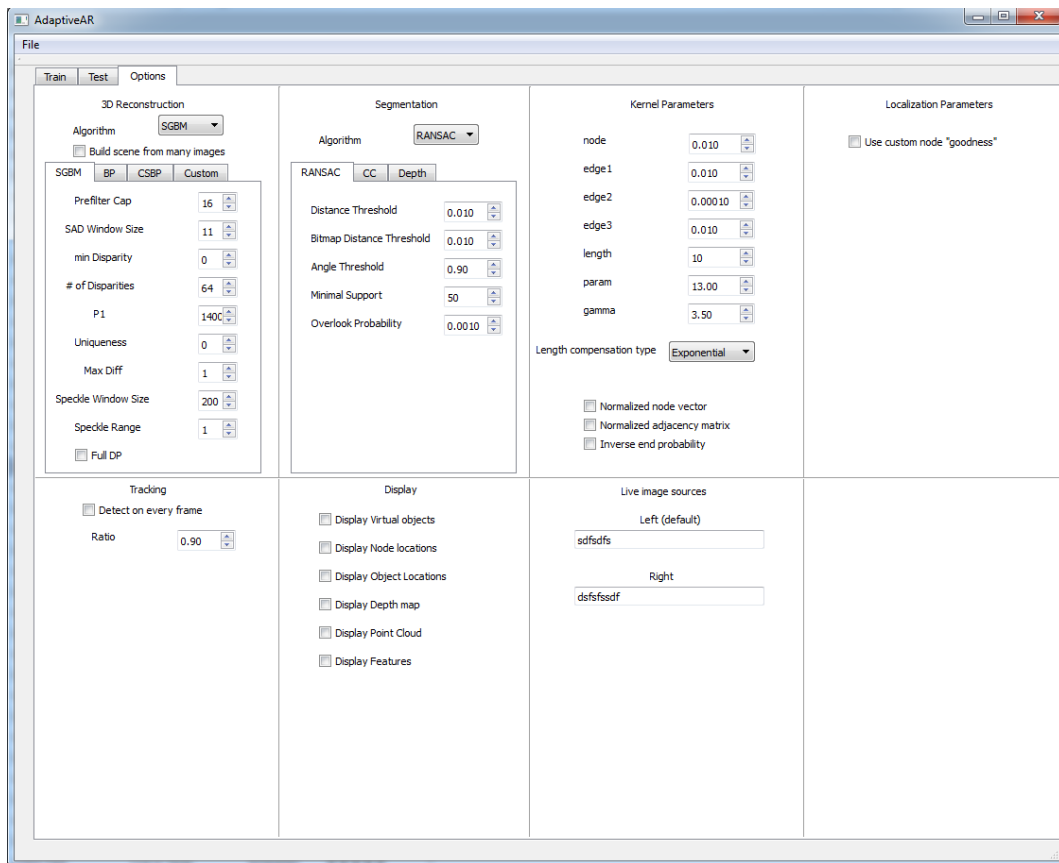
**4.1 Ábra: A Tanító nézet**



4.2 Ábra: A bekeretezés



4.3 Ábra: A Teszt nézet



4.4 Ábra: Az opció nézet

### 4.1.1 Funkciók

A keretrendszer megtervezéséhez fontos feladat a funkcionális követelmények alapos felállítása, mivel a főbb funkciókat kiszolgáló szerkezet a rendszer minőségét alapvetően képes meghatározni. A soron következő fejezetben a három legfontosabb use case-t fogom részletesen tárgyalni. Ezek a tanító adatok hozzáadását, a kiterjesztett valóság környezet létrehozását ismeretlen környezetben, valamint a környezet fenntartását fogják szolgálni. A fejezet során ezen use case-ek részletes leírását fogom megadni szekvencia diagramokkal kiegészítve.

A tanító adat elvégzése a legösszetettebb a három funkció közül. Alapvetően három fő feladatra osztható: Az elsőben felhasználó megadja a tanító adatként szolgáló képeket, majd egyesével bekeretezi a helyes objektumot a képeken. A második lépésben a rendszer automatikusan előállítja a tanító adatokat a befoglaló téglalapok alapján, végül a harmadik lépésben a rendszer elvégzi a tanítást, vagy a tanuló rendszerek helyes használata esetén elengedhetetlen kereszt-validációt a beállítások függvényében. A

funkció során lehetőség van az első két lépés átugrására, ha csak a tanítást szeretné a felhasználó újra elvégezni más paraméterekkel. A funkció szekvencia diagramja a 4.5 Ábrán látható, a részletes leírása pedig a következő:

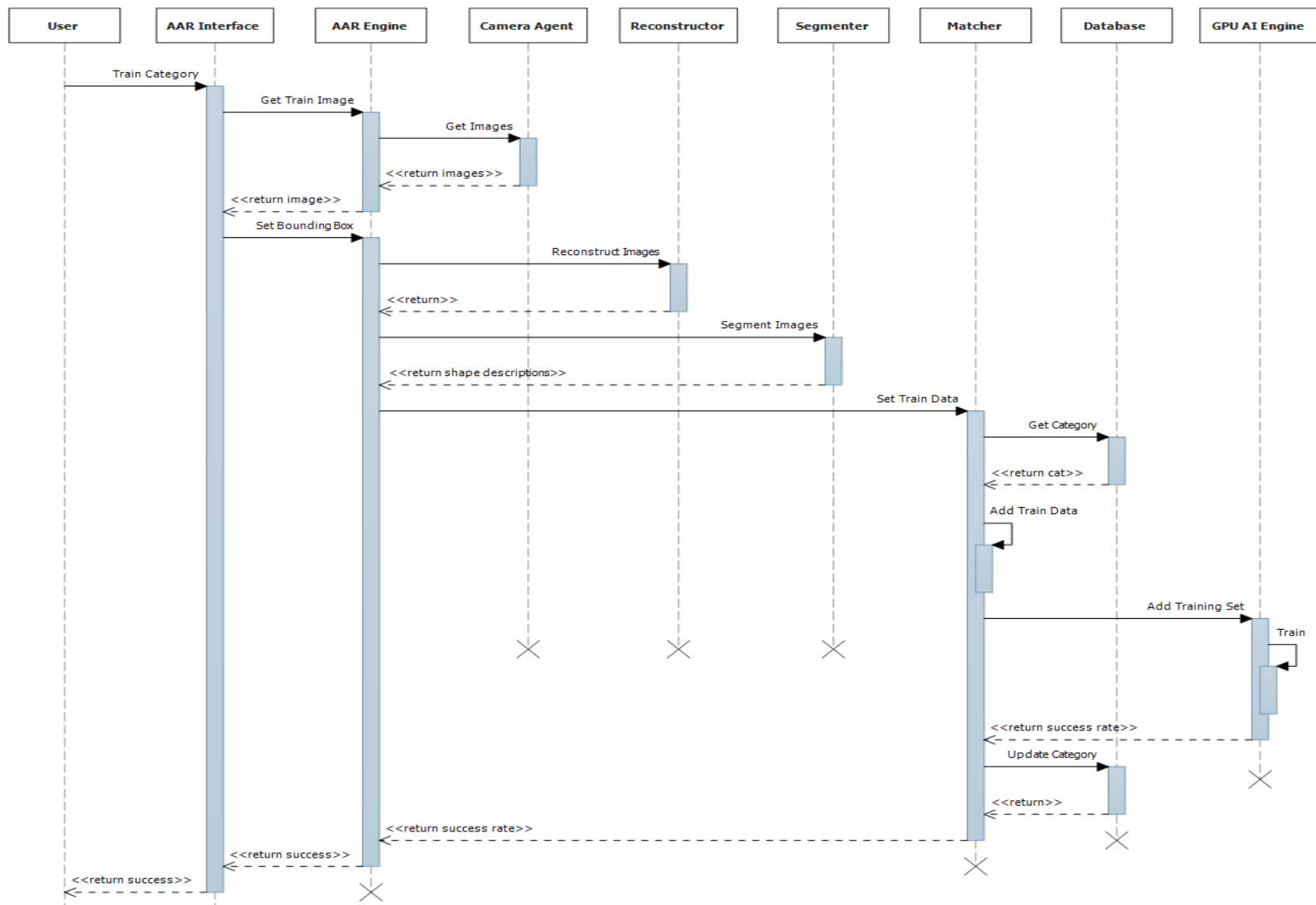
1. A felhasználó kéri az új tanító adatok hozzáadását a képek elérési címének megadásával
2. Amíg van kép:
  - 2.1. Az AAR rendszer motorja elkéri a képeket a kamerákat kezelő modultól és megkéri a felhasználót, hogy jelölje be a befoglaló téglalapot
  - 2.2. A felhasználó megadja a téglalapot, a rendszer pedig eltárolja azt
3. Amíg van kép és hozzá tartozó befoglaló téglalap:
  - 3.1. Az AAR motor kéri a sztereó képpár rekonstrukcióját az erre képes modultól, amely elvégzi a rekonstrukciót, és visszaadja a keletkezett pontfelhőt
  - 3.2. Ezt követően a motor megkéri a szegmentációt végző modult, hogy a pontfelhő szegmentálását és alakleírását végezze el, a szegmentáló modul pedig visszatér az alakleírás eredményével
  - 3.3. Ezt követően a motor elküldi az új adatokat a párosító modulnak,
    - 3.3.1. A párosító az adatbázistól elkéri a tanítani kívánt kategóriát
    - 3.3.2. A párosító modul létrehozza a tanító adatokat, és hozzáadja a kategóriához
    - 3.3.3. A párosító ezt követően a mesterséges intelligencia modul számára biztosítja az új adatokat, mire az MI modul elvégzi a tanítást, és visszatér a tanítás eredményeivel
    - 3.3.4. A párosító frissíti a kategóriát, és jelzi a változást az adatbázisnak
  - 3.4. Ezt követően jelzést ad a rendszer a felhasználónak a tanítás sikerességéről

A következő tárgyalandó funkció az AR környezet előállítás, amelyet az előre ismeretlen jelenet feltérképezésével és berendezésével oldok meg. Ez a lépés a környezet indításának első lépése, melyet csak egyszer szükséges elvégezni. Főbb lépései a háromdimenziós jelenet előállítása, az objektumjelöltek előállítása a háromdimenziós jelenetben, végezetük pedig a jelöltek segítségével egy optimális berendezés számítása. A use case feltételezi a primitív gráf alapú alakleírást és tanulást. A funkció szekvencia diagramja a 4.6 Ábrán látható. A részletes lépései a következők:

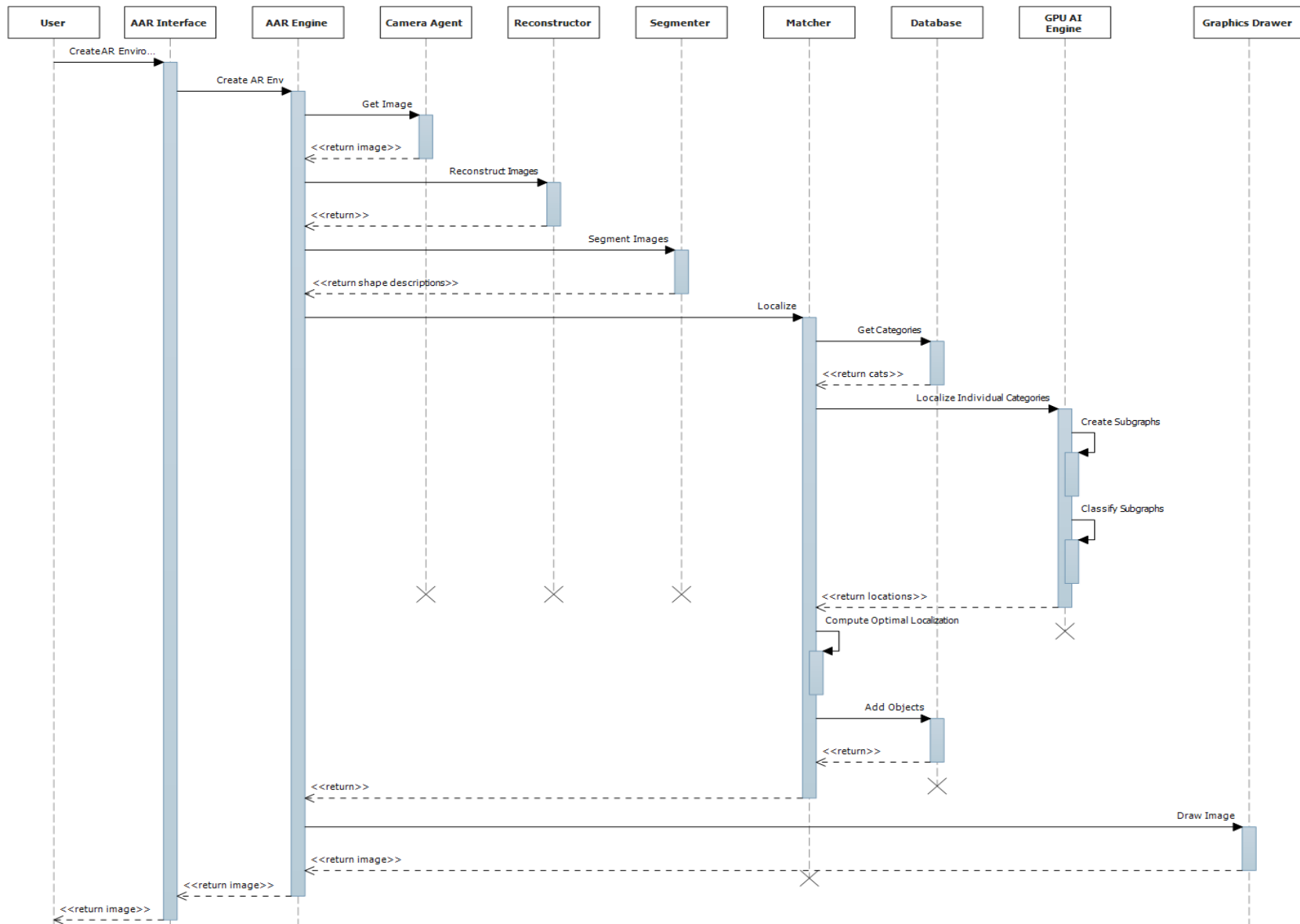
1. A felhasználó utasítja a rendszert a berendezés indítására, melyet a felhasználói felület továbbít az adaptív kiterjesztett valóság rendszert működtető motornak
  - 1.1. A motor ezt követően elkéri a jelenlegi sztereó képpárt a kamerát kezelő modultól, amely visszatér a képpárral
  - 1.2. A motor ezután a rekonstrukciós modult kéri meg a pontfelhő előállítására, amely visszatér az eredménnyel
  - 1.3. Ezt követően a motor utasítására a szegmentáló modul elvégzi a felbontást és az alakleírást, majd visszatér a leírt formákkal
  - 1.4. A motor utasítja a párosító modult, hogy végezze el a lokalizációt
    - 1.4.1. A párosító elkéri az adatbázistól az összes kategóriát, amit az adatbázis átad
    - 1.4.2. Az összes kategóriára:
      - 1.4.2.1. A párosító megkéri a mesterséges intelligencia modult, hogy állítsa elő az adott kategóriához tartozó objektumjelölteket
        - 1.4.2.1.1. Az MI motor először lehetséges jelölteket állít elő
        - 1.4.2.1.2. Ezt követően az MI motor a lehetséges részeket klasszifikálja, a jó eredményt elérő jelölteket pedig visszaadja
    - 1.4.3. A párosító ezt követően elkészíti az optimális berendezést, kezelve az esetleges átfedéseket.
    - 1.4.4. Az előző lépésben előállított objektumokat a párosító felveszi az adatbázisba
  - 1.5. A motor ezt követően jelzi a kijelző modulnak, hogy az objektumok alapján készítse el a kimeneti képet, amelyet a felhasználó számára megjelenít

Az utolsó feladat az AR környezet futtatása, amely egy folyamatosan, végtelen ciklusban futó művelet. A fő feladata, hogy a felhasznált valós tárgyakat kövesse, és a követés eredménye alapján a virtuális objektumok változásait kezelje. Fő lépései a követés, a detektálás az adatbázis frissítés, valamint a kijelzés megoldása. A funkcióhoz tartozó szekvencia diagramot a 4.7 Ábrán láthatjuk, a lépései pedig az alábbiak:

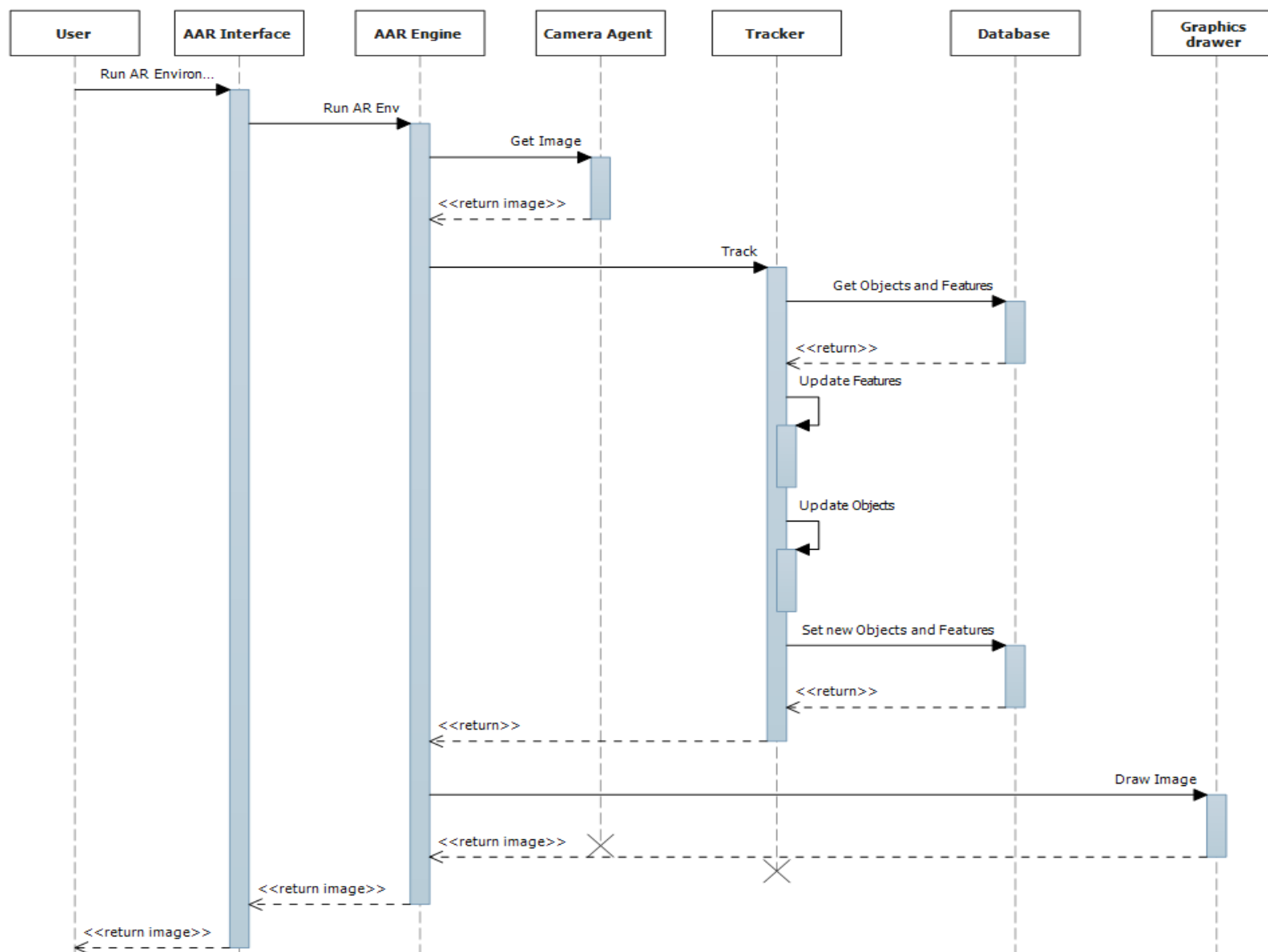
1. A felhasználó elindítja a virtuális környezet futását, melyet a rendszer továbbít az AAR motor számára, amely a felhasználó leállító parancsának megérkezéséig végtelen ciklusban működik
  - 1.1. A motor elkéri a soron következő sztereó képpárt a kamerát kezelő modultól, amely visszatér a képpárral
  - 1.2. A motor jelzi a követő modulnak, hogy új képpár áll rendelkezésre
    - 1.2.1. A követő modul elkéri a környezetben lévő modulokat és képjellemzőket az adatbázistól, melyeket az adatbázis átad számára
    - 1.2.2. A követő modul ezt követően a megkapott képpár alapján frissíti a képjellemzőket
    - 1.2.3. A frissített képjellemzők alapján a követő frissíti az objektumokat
    - 1.2.4. A követő modul jelzi az adatbázisnak a jellemzők és az objektumok frissülését, amit az adatbázis nyugtáz
  - 1.3. A motor kérést küld a kijelzést végző modulnak, hogy frissültek az objektumok, mire a kijelző modul elvégzi a kirajzolást, és a rendszer kijelzi a képet a felhasználó számára



4.5 Ábra: A tanítás szekvencia diagramja



4.6 Ábra: A lokalizáció szekvencia diagramja



4.7 Ábra: A környezet futtatásának szekvencia diagramja

### 4.1.2 Osztályok

A jelenlegi fejezet témája a keretrendszer működését biztosító osztálystruktúra bemutatása, melyet jelentős részben az előző fejezetben felvázolt funkcionális követelmények alapján állítottam össze. Már ott implicit módon sejtettem a tervezői döntést, miszerint a rendszer működése során megjelenő különálló feladatokat külön funkció osztályokban valósítom meg. Mielőtt azonban ezeket a funkció osztályokat bemutatnám fontos beszélni az alkalmazás adatstruktúrájáról, vagyis azon osztályok szerkezetéről, melyek a funkció osztályok számára paramétereket, adatokat szolgáltatnak.

A rendszer számára nélkülözhetetlen adatstruktúrák a képet, és a pontfelhőt reprezentáló osztályok, melyeket külső könyvtárak segítségével emeltem a rendszerbe. A képek előállításához és feldolgozásához szükséges struktúra a Camera osztály, mely egyetlen fizikai kamera paramétereit írja le. Részei az elérési út, a kamera- és projekciós mátrixok, valamint a kép rektifikálására szolgáló pixeltérkép, valamint a radiális torzítást leíró együtthatók. Ezeket a paramétereket főleg a kép beolvasása utáni képfeldolgozó lépések használják, valamint néhány paraméter az esetleges 3D=>2D vetületek számítása estén szükséges. Közvetlen kapcsolódó osztály a CameraPair, mely egy két kamerából álló sztereó párt ír le. A két Camera tagváltozóján kívül tárolja még a diszparitásképet pontfelhővé történő konvertálásához szükséges mátrixot, valamint a két kamera közti hat szabadságfokú transzformációt leíró paramétereket. A CameraPair osztályt főleg a rekonstrukció során használja a rendszer.

A rendszer szempontjából alapvető adat jellegű osztály a Shape, vagyis egy adott (jellemzően primitív) alakzatot leíró objektum. A Shape legfontosabb paraméterei a 3D pozícióját és orientációját leíró 1-1 vektor, valamint a forma paramétereit tartalmazó jellemző vektor. A Shape osztályra közvetlenül épül egy másik osztály a Node, mely a formákat leíró gráf egy csomópontját reprezentálja. A Node tartalmaz egy Shape típusú osztályt, valamint az adott formát befoglaló téglatestet. Fontos tulajdonsága a Node osztálynak, hogy tartozik hozzá egy egyedi azonosító (ID), melyet a konstruktorban autóinkrementálás segítségével tartok fenn. Ezt az inkrementálást azonban a Node osztály alapértelmezett paraméter nélküli konstruktora nem végzi el, ekkor ugyanis a különböző STL konténer osztályok beillesztő metódusai a konténerbe behelyezett objektumot különböző ID-val hoznák létre.

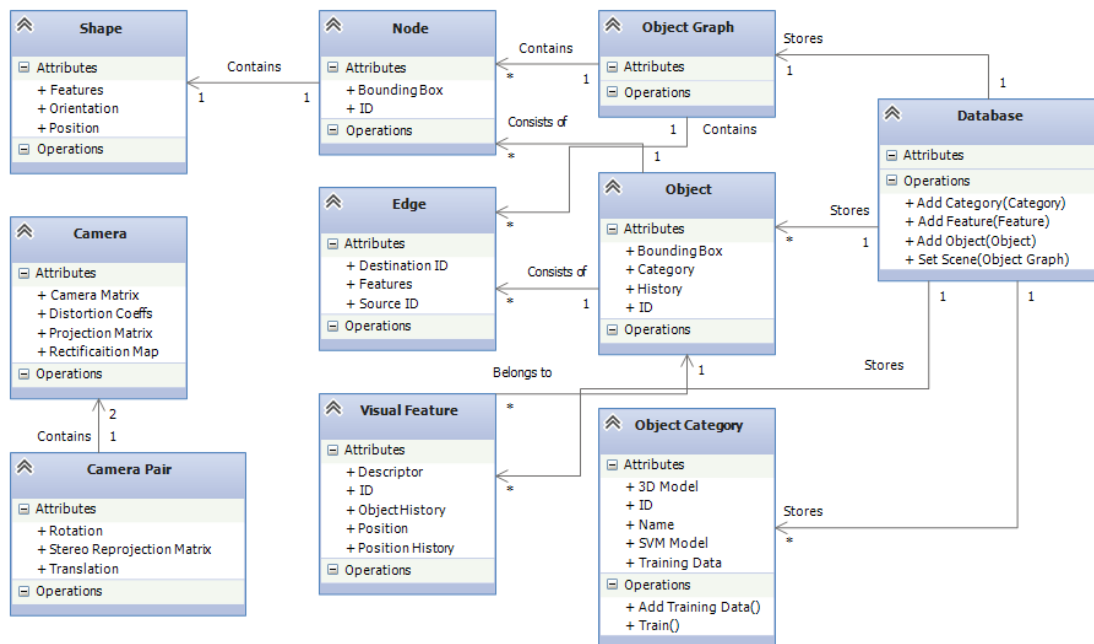
A primitív gráf létrehozásához szükséges osztály még az Edge, amely a gráf éleit reprezentálja. Az Edge osztály fontos paramétere a két csomópont orientációja közti elforgatást jellemző kvaternió, valamint az alakzatok középpontjai közti transláció. Lényeges továbbá, hogy minden él tartalmazza a két végén lévő csomópontok azonosítóját. A gráf objektum reprezentálására az ObjectGraph osztály szolgál. Az osztály maga meglehetősen egyszerű, két hash map objektumban tárolja a gráf éleit és csomópontjait. A csomópontok esetén a map kulcsa a csomópont azonosítója, míg élek esetén a kulcs a két élhez tartozó csomópont azonosítójából létrehozott pár.

Fontos alapvető adattípus a VisualFeature, amely a képjellemzők tárolására szolgál. Fontos részei a képjellemző leíró vektora, valamint a képen való elhelyezkedése. Mivel sztereó képen dolgozok lehetőség van mind a két képen megadni a jellemző pozícióját, valamint térbeli pozíció megadása is lehetséges. Minden képjellemzőhöz tartozik egy egyedi azonosító, amely a csomópont azonosítójával megegyező módon működik, valamint eltárolja annak az objektumnak az azonosítóját is, amelyhez az adott jellemző tartozik. Fontos tulajdonsága a VisualFeature osztálynak a history, ami a jellemző pozíciójának, illetve az őt tartalmazó objektum azonosítójának múltbéli értékeit tárolja.

A következő fontos osztály az Object, amely egy követendő valós tárgyat reprezentál. Korábban megemlítettem, hogy az Object osztályhoz is tartozik egy egyedi azonosító. Az objektumhoz tartozik továbbá több Node és VisualFeature is. Ezeket azonban az objektum nem tartalmazza, csupán hivatkozik rájuk. Az hivatkozott csomópontok befoglaló téglalapjából számolható egy összesített befoglaló doboz, amely az Object tagváltozója. Fontos tulajdonság még az adott objektum kategóriájának azonosítója, amely segítségével virtuális objektumot rendelhetek a valós tárgyhoz, valamint az Object 3D pozíciója, amit szintén history formájában tárolok.

Két fontos osztályt hoztam még létre az adatstruktúrához, melyek közül az egyik az ObjectCategory, amely egy virtuális objektumkategóriát tartalmaz. Ez az osztály egyrészt ad egy alapvető leírást a kategóriáról (név, modell, stb.), valamint lehetővé teszi a legfontosabb feladatok elvégzését. Az ObjectCategory tárolja a tanításhoz szükséges adatokat, valamint a betanított modellt, illetve képes ezen adatok kezelésére (tárolás, tanítás, adat manipuláció), továbbá a kategóriához is tartozik egy egyedi azonosító.

Az utolsó lényeges osztály az ObjectDatabase, vagyis az előző bekezdésekben bemutatott elemek tárolására szolgáló adatbázis osztály. Ez az osztály nem valósít meg fejlett adatbázis interfészeket és szolgáltatásokat, csupán egy tárolásra szolgáló osztály. Képes egy jelenet (ObjectGraph) tárolására, valamint tárolja a jelenetben található objektumokat, valamint a létező kategóriákat egy-egy hash map objektum segítségével. Ezen felül az adatbázis számon tartja a jelenetben megtalált képjellemzőket szintén egy map tároló segítségével. Az adat osztályok kapcsolatáról a tervezés során osztálydiagramot készítettem, amely a 4.8 Ábrán látható.



**4.8 Ábra: Az adat osztályok osztálydiagramja**

A következő lényeges tárgyalandó aspektusa a keretrendszer szerkezetének a különböző funkciókat ellátó osztályok bemutatása. Ezen osztályokat az előző fejezetben elkészített és bemutatott funkcionális leírások segítségével határoztam meg. Természetesen a funkció osztályok kapcsolatairól is készítettem osztálydiagramot, amely a 4.9 Ábrán látható.

Az első fontos funkció a bemeneti videó, vagy képek szolgáltatása volt. Ehhez egy külön osztályt hoztam létre (CameraAgent), amely képes megadott címről (legyen az fájlrendszerbeli, vagy webes cím) képsorozatot olvasni. A CameraAgent osztály tartalmaz egy CameraPair objektumot, melyet a kép olvasásra és feldolgozására használ.

Lehetőség van mono és sztereó képolvasásra is. Ez utóbbi esetén fontos, hogy élő videó használata esetén a két kamerából olvasott kép között minél kisebb legyen az időkülönbség, mivel a nem jól szinkronizált sztereó képek később problémákat okozhatnak. Ezért a kép készítését nem blokkoló műveletekkel kell kérni a két kamerától, a konkrét képek beolvasására csak eztuán kerülhet sor. További szolgáltatása a CameraAgent osztálynak, hogy egy nyers és egy előre feldolgozott képet is képes adni válaszképp. A feldolgozott képen különböző felhasználó által kért műveleteket végezhet: Ez lehet skálázás, színtérkonverzió (szürkeárnyalatosítás), vagy szűrés. Fontos megjegyezni, hogy a CameraAgent a kamera képeinek rektifikálását automatikusan elvégzi.

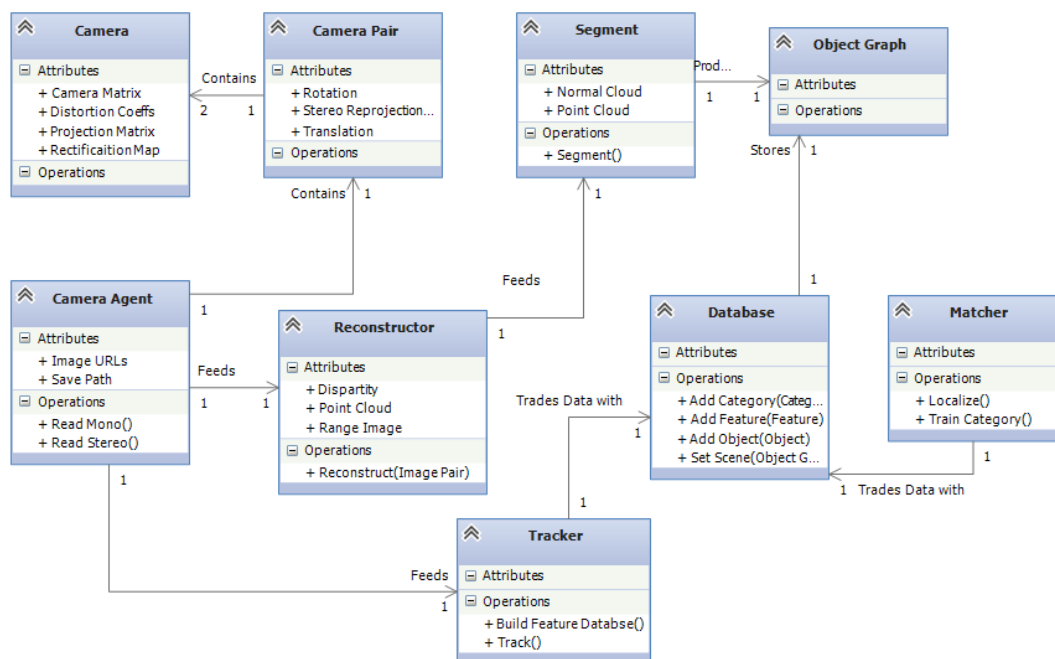
A következő fontos feladat a 3D rekonstrukció végzése, melyet a Reconstructor osztály segítségével végzek. Az osztály bemenete egy sztereó képpár, kimenete pedig az ezekből előállított pontfelhő. Lehetőség nyílik a Reconstructor osztálytól egy színes pontfelhő elkérésére is, ahol az egyes pontok színértéke megfelel a hozzájuk tartozó pixel értékének, valamint az osztály a rekonstrukció köztes eredményeit, a diszparitást, illetve a mélység képeket is elérhetővé teszi. A Reconstructor osztálynak szükséges referenciát szereznie a CameraAgent CameraPair típusú tagváltozójára, mivel ennek adataira szüksége van a rekonstrukció végzéséhez.

A soron következő bemutatandó osztály a Segmenter, amely a pontfelhő szegmentálását, illetve az így előállított részek alakjának leírását végzi el. A Segmenter osztályban a rendszer evolúciója során többféle algoritmus is alkalmazásra került, így az osztály felülete ezzel együtt változott. Habár a bemenet minden egyes algoritmus esetén egy pontfelhő volt, az algoritmus kimenete az algoritmustól függően több pontfelhő, vagy pedig ObjectGraph. A Segmenter osztály belül elvégzi a pontfelhő feldolgozását (szűrés, felületi normálisok számítása), ezt követően pedig belső algoritmusok meghívásával elvégzi a felosztást és a leírást.

A következő osztály a Matcher, amely a virtuális és valós objektumok párosítását végzi, valamint felületet biztosít a tanuló algoritmusok eléréséhez. A Matcher igen sokrétű osztály képes elvégezni a tanító adatok előállítását egy gráf és befoglaló téglalapok segítségével, valamint a kategóriák tanítását felügyelni. Képes továbbá a lokalizációs berendező eljárás elvégzésére. Éppen ezért a Matcher osztálynak hozzáférése van az adatbázishoz, mivel ő végzi a kategóriák kezelését, valamint az objektumok

előállítását. A Segmenter által előállított ObjectGraph típusú jelenet is az adatbázison keresztül kerül a párosító osztályhoz. A rendszer készítése során az Matcher is több különböző algoritmust valósított meg, melyeket bővebben a 4.2 fejezetben mutatok be.

A virtuális környezet futtatásához szükséges funkció osztály a Tracker, mely a követést és a detektálást végzi. A Matcher-hez hasonlóan a Tracker osztály is eléri az adatbázist, hogy az objektumokat, valamint a képjellemzőket frissíteni tudja. A követési és detektálási adatbázis felépítése is a Tracker osztály feladata. A környezet futásához a kijelzést elvégző GraphicsDrawer osztály is nélkülözhetetlen. A jelenlegi implementáció szerint a rajzoló osztály a megkapott objektumok pozíciójába meghatározott méretű primitív formákat rajzol az objektumok kategóriája alapján, a konkrét modellek megjelenítése még nincs implementálva. A kirajzoló osztály képes még a tesztelési folyamatot egyéb adatok vizualizációja által segíteni. Lehetőség van a képjellemzők, vagy az objektumok befoglaló dobozainak kirajzolására is.

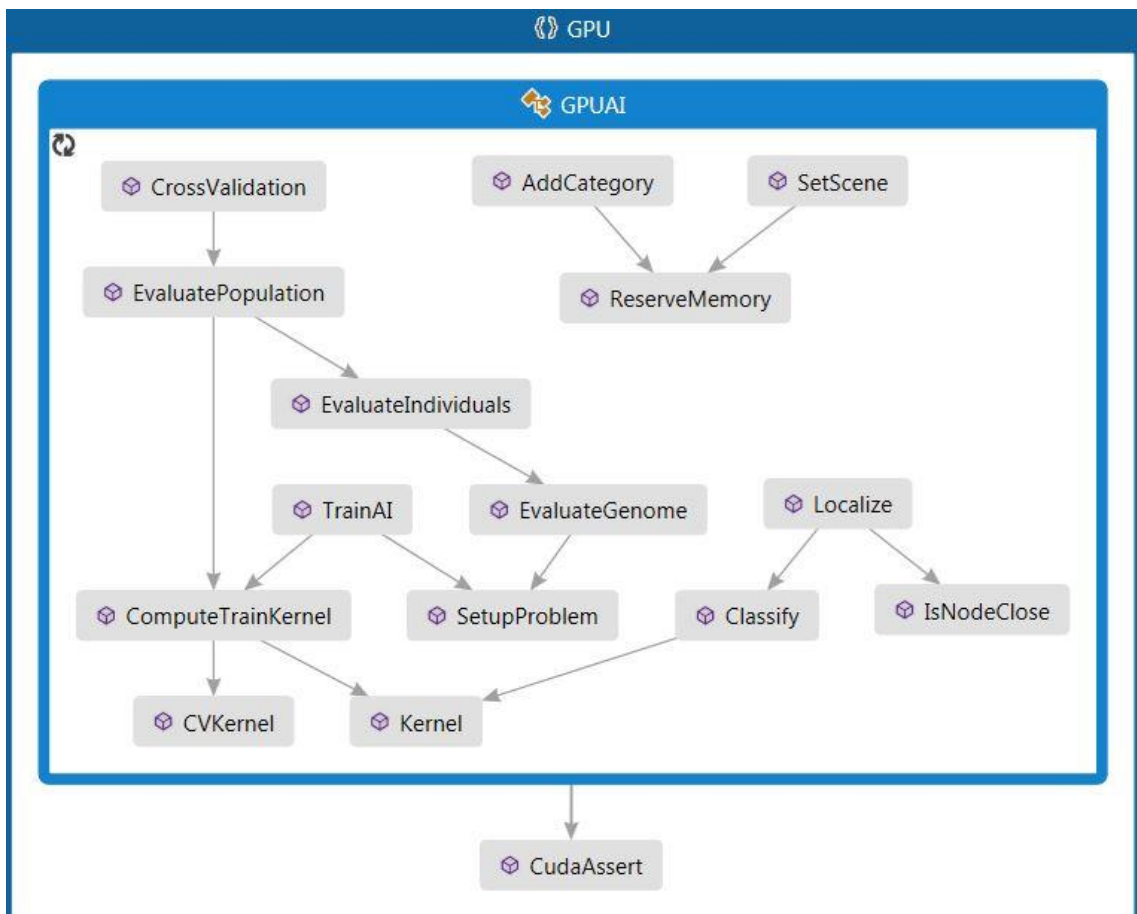


4.9 Ábra: A funkció osztályok osztálydiagramja

Az imént bemutatott osztályokon felül még három további egység bemutatása szükséges. Ezek egyike az Algorithm, amely egy gyakran használt, de egyik másik osztályhoz sem köthető funkciók megvalósítását tartalmazza. Ezekre jó példa a különböző fájlrendszerbeli írási, illetve olvasási műveletek. A másik fontos osztály az AAREngine, amely egyfajta fő vezérlőosztályként hívja a hozzá tartozó osztályok

funkcióit. Ez az osztály felel a korábban bemutatott osztályok példányosításáért, az algoritmusok paramétereinek tárolásáért és átadásáért, valamint a felhasználó által indítható műveletek megvalósításáért. További feladata a motor osztálynak a szálkezelés, mellyel a felhasználói felület válaszolását biztosítom a hosszú, blokkoló műveletek futása közben.

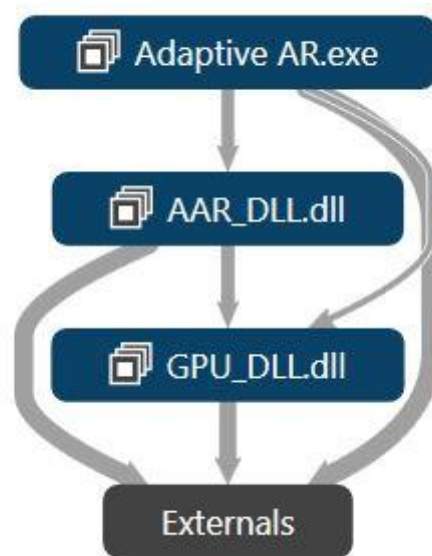
Az előző fejezetben a szekvencia diagramokon szerepelt egy GPUAI nevű osztály, amelynek a fő feladata, hogy a tanuló algoritmusokhoz tartozó nagy számításigényű műveleteket a grafikus kártya segítségével gyorsítsa. Az osztály fő feladatai közé tartozik a grafikus kártya memóriakezelése, valamint elérhetővé teszi a gépi tanuló algoritmus főbb szolgáltatásait: A tanítást, a lokalizációt, a kereszt-validációt, illetve néhány adminisztrációs függvényt. A GPUAI osztály algoritmikus felépítése a 4.10 Ábrán látható.



4.10 Ábra: A GPUAI osztály algoritmusai.

### 4.1.3 Implementációs kérdések

Miután a keretrendszer megtervezését részletesen tárgyaltam, ideje rátérni a megvalósítással kapcsolatos kérdések bemutatására. A megvalósítást C++ nyelven végeztem el a Visual Studio IDE segítségével. A keretrendszer struktúrája alapvetően három különböző szintre osztható fel: A legfelső, a felhasználóval közvetlenül kapcsolatot teremtő részegység (Adaptive AR) a grafikus felhasználó felület és a keretrendszer vezérlését végző egység, mely a másik két egységet dinamikus könyvtárakon keresztül éri el. Logikailag a fő modulnak alárendelve található az adaptív kiterjesztett valóság környezetet kezelő és vezérlő algoritmusok által alkotott részegység (AAR\_DLL). Ennek az egységnek alárendelve találhatjuk a tanuló algoritmusok grafikus magokon megvalósító modult (GPU\_DLL). Mindhárom modul természetesen érintkezik külső könyvtárakkal. A teljes struktúra a 4.11 Ábrán látható.



4.11 Ábra: A rendszer struktúrája

A fő modul elsődleges feladata a grafikus felhasználó felület kezelése, ezen belül is különösen fontos az egyes opciók feltöltése indításkor, valamint a memóriában lévő beállítás fájl folyamatos frissítése felhasználói beavatkozás esetén. Fontos feladat még a különböző felhasználói parancsok végrehajtása, valamint az alárendelt AAR\_DLL modul vezérlése ennek alapján. A fenti ábrán látható, hogy a Visual Studio által generált függőségi gráf esetén a GPU\_DLL modul is kapcsolódik a fő egységhez. Ennek oka, hogy a fő egység feladata az aleggységek által futtatott aszinkron feladatok státuszának

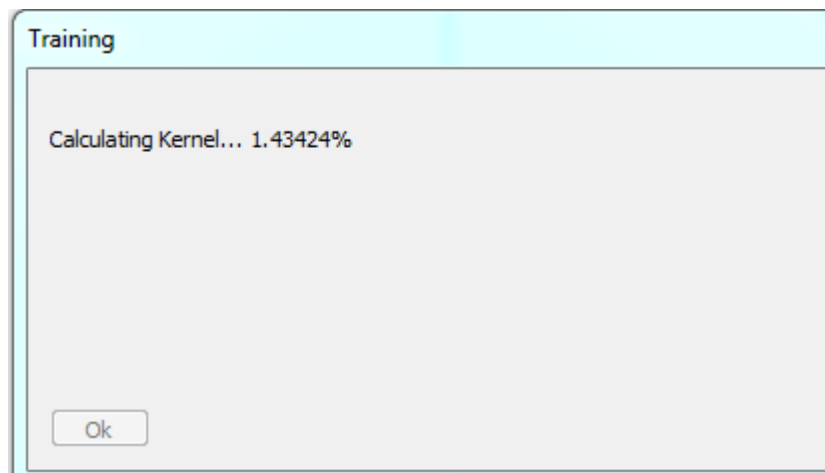
megjelenítése, vagyis a fő modul mindkét almodulból kaphat üzenetet. Az imént felsorolt feladatokat a fő modulban a Qt [36] nyílt forráskódú könyvtár segítségével valósítottam meg.

Az AAR\_DLL modult a fő modul egy dinamikus könyvtár formájában érheti el, melynek interfészét a korábban bemutatott AAREngine osztály adja. A modul megvalósítására számos külső könyvtárat felhasználtam. A képek kezelésére, illetve képfeldolgozási feladatokra az OpenCV [37] nyílt forráskódú képfeldolgozó könyvtárat használtam, mely biztosította a képek olvasásához, rekonstrukcióhoz, valamint a természetes képjellemzők kinyeréséhez szükséges metódusokat. A háromdimenziós pontfelhők manipulációjára a PCL (Point Cloud Library) [38] könyvtárat használtam, melynek segítségével a pontfelhők előzetes feldolgozását végeztem. Ennél a résznél nagy segítséget jelentett a 2.3 fejezetben bemutatott RANSAC variáns implementációja, melyet a cikk szerzői biztosítottak [39]. A modulhoz szükséges lineáris algebra műveletek elvégzésére pedig az Eigen [40] könyvtárat használtam. Jelentős segítség volt továbbá a gépi tanuló algoritmus implementálásához a LIBSVM [41] függvénykönyvtár.

A GPU\_DLL modul szintén egy dinamikus könyvtár, amelyet főleg az AAR\_DLL modul Matcher osztálya használt gyorsított tanuló algoritmusok elérésére. A grafikus processzoron történő programozáshoz a CUDA nyelvet használtam, valamint a szintén az Nvidia által szolgáltatott CUBLAS könyvtárat, amely GPU által gyorsított lineáris algebra műveleteket tesz elérhetővé. A GPU\_DLL modulhoz szintén felhasználtam a LIBSVM [41] könyvtárat, valamint a genetikus algoritmusokat tartalmazó GALIB [42] függvénykönyvtárat.

Az alkalmazás fontos eleme volt a szálkezelés megoldása, ugyanis a legtöbb használt algoritmus meglehetősen hosszú futási idejű, blokkoló művelet, ezek futása alatt a felhasználói felületnek azonban rezponzívnak kell maradnia, ezért ezeket a műveleteket külön szálban kell futtatni. Ekkor viszont külön meg kell oldani a külön szálban futó algoritmusok kommunikálását a fő modullal, valamint a kijelzés problémáját. A Qt könyvtár (és a legtöbb felhasználói felülettel operáló futtatási környezet) ugyanis nem engedi, hogy UI műveleteket végezzünk a fő száltól különböző szálból. Lényeges feladat továbbá az egyes algoritmusok megállítása vagy megszakítása felhasználói utasításra.

A fenti feladatokra az alábbi három rendszer implementáltam: A különösen hosszú futási idejű algoritmusok esetén (tanítás, kereszt-validáció) egy felugró ablakot hoztam létre, amely az algoritmusok állapotát volt képes kijelezni. A modulok kommunikációja a Qt könyvtár központi eseménykezelő rendszerén keresztül történt. Az algoritmusok különböző események jelzésével voltak képesek a felugró ablak létrehozására, üzenet kiírására, valamint az ablak bezáró gombjának engedélyezésére. A felugró ablak a megkapott üzeneteket három szinten képes kijelezni, így egymásba ágyazott algoritmusrészek állapotát egyszerre lehet megjeleníteni. (4.12 Ábra)



**4.12 Ábra: A felugró ablak**

Egy másik megoldandó probléma volt a kijelzés megoldása. A GraphicsDrawer osztály függvényei ugyanis a feldolgozó szálban futottak, így értesíteni kellett a fő szálát, ha új képek elkészültek. Erre a feladatra a boost függvénykönyvtár signal/slot kommunikációs megoldását használtam. A kirajzoló oldal signal függvényének a felhasználó felület képmegjelenítő slot metódusába való bekötést a mindkét osztályt elérő AAREngine végezte el. Az algoritmusok esetleges várakoztatását, illetve leállítását pedig UI gombok által vezérelt mutexek segítségével oldottam meg. Erre szintén a boost könyvtár named\_mutex szolgáltatását használtam.

További érdekes implementációs kérdés a GPU\_DLL modul memóriakezelésének megoldása. A grafikus processzorok általános célú használata esetén ugyanis köztudottan a szűk keresztmetszetet a grafikus kártya és a CPU közti kommunikáció jelenti, különös tekintettel az adatok másolására. Éppen ezért célszerű lenne az összes szükséges adatot a program futása elején felmásolni a GPU memóriájában, és csak a lehető legkritikább esetben kezdeményezni adatmásolást a futás

közben. Ez első megközelítésben nem tűnhet nagy nyereségnek, mivel a GPU funkciókat intenzíven használó feladatok (tanítás, lokalizáció) nem időkritikusak, így nem nyerünk ezzel a megoldással sokat. A rendszer későbbi működése során szükség lehet azonban új, a jelenetbe belépő objektumok kategorizálására, vagy egy korábbi meghozott döntés felülbírálására, amit a rendszer valósidejű futása közben kell elvégezni. Ilyenkor nagy mennyiségű adatmásolás szükségessé tétele hibás tervezői döntés lenne. Fontos eleme volt azonban a memóriakezelés közben felkészülni annak a lehetőségére, hogy a grafikus processzor memóriája megtelik adattal, amely a kategóriák, illetve a tanító adatok számának növekedésével egyre valószínűbb eshetőséggé válik.

## 4.2 Feltöltés algoritmusokkal

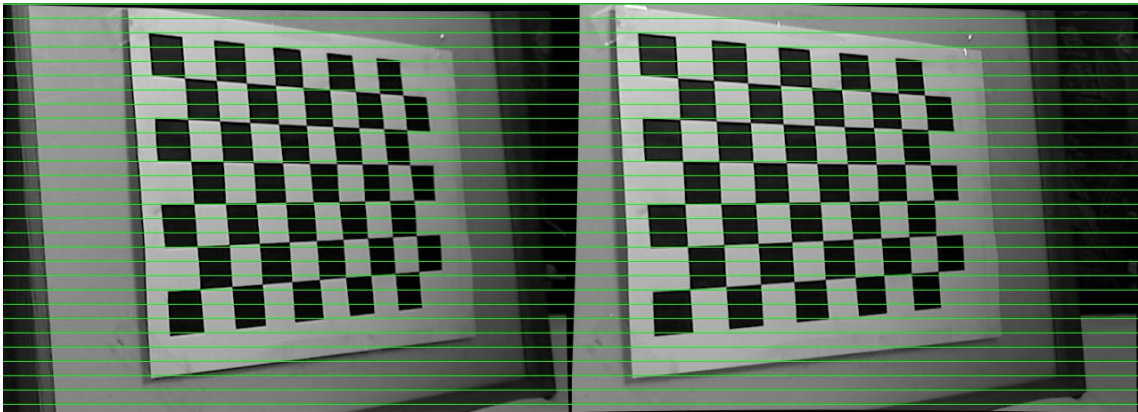
Miután a keretrendszer tervezésével és implementálásával kapcsolatos kérdéseket a jelenlegi fejezet első részében bemutattam, rátérhetek a megvalósítás második felére, amely a keretrendszer funkció osztályiban használt algoritmusokat tárgyalja. Az alábbi részfejezetben ezeket az algoritmusokat fogom bemutatni, kezdve a Reconstructor osztály felöltésével. Ezt követően a szegmentációra és az alakzatok leírására használt algoritmusokat fogom ismertetni, majd az alakfelismerésre használt algoritmusokat mutatom be. Ez az alfejezet a Matcher, illetve a GPUAI osztályok feltöltésével foglalkozik. Végezetül bemutatom a követésre használt algoritmust, amelyet a Tracker osztályba implementáltam.

### 4.2.1 Rekonstrukció

A jelenlegi fejezetben a Reconstructor osztály implementálásával fogok foglalkozni, amely a felhasznált sztereó rekonstrukciós algoritmus(ok) ismertetését jelenti. Mielőtt azonban erre rátérnék, ki kell tennem egy nélkülözhetetlen feladat elvégzésére: Ez a kamera kalibráció, illetve a sztereó kalibráció. Ez az a feladat, melynek során előállítom a Camera és a CameraPair osztályban található adatokat, melyeket a CameraAgent és a Reconstructor osztályok később felhasználhatnak a saját működésükhöz. A kalibrációt a sakktábla-módszer segítségével végeztem el.

A feladat elvégzéséhez egy előre elkészített programot használtam. A használatához először nyomtatni kellett egy sakktáblát, majd meg kellett mérni a négyzetek oldalainak a hosszát, és ismerni kellett a sorok és az oszlopok számát. Ezután a nyomtatott sakktáblát valamilyen síkfelületre felragasztottam és több különböző sztereó

képet is készítettem róla. A letöltött program ezeket a képeket használta bemenetként. Elméletileg egy kép is elég, de több, más nézőpontból készített kép növeli a pontosságot. A program ezeken a képeken beazonosítja a sakktáblán lévő négyzetek csúcsait, és ezeket a két képen összepárosítja. Mivel ismerjük a négyzetek méretét, és tudjuk, hogy térben egy egyenes köti őket össze, ezért kiszámítható az egyes kamerák principális pontja, fókusz távolsága, valamint a radiális és tangenciális torzítása, amiket fel lehet használni a képek torzításmentesítésére (rektifikálás). Az eljárás kiszámítja még a két kamera relatív 6DOF transzformációját, és az epipoláris egyeneseket. (4.13 Ábra)

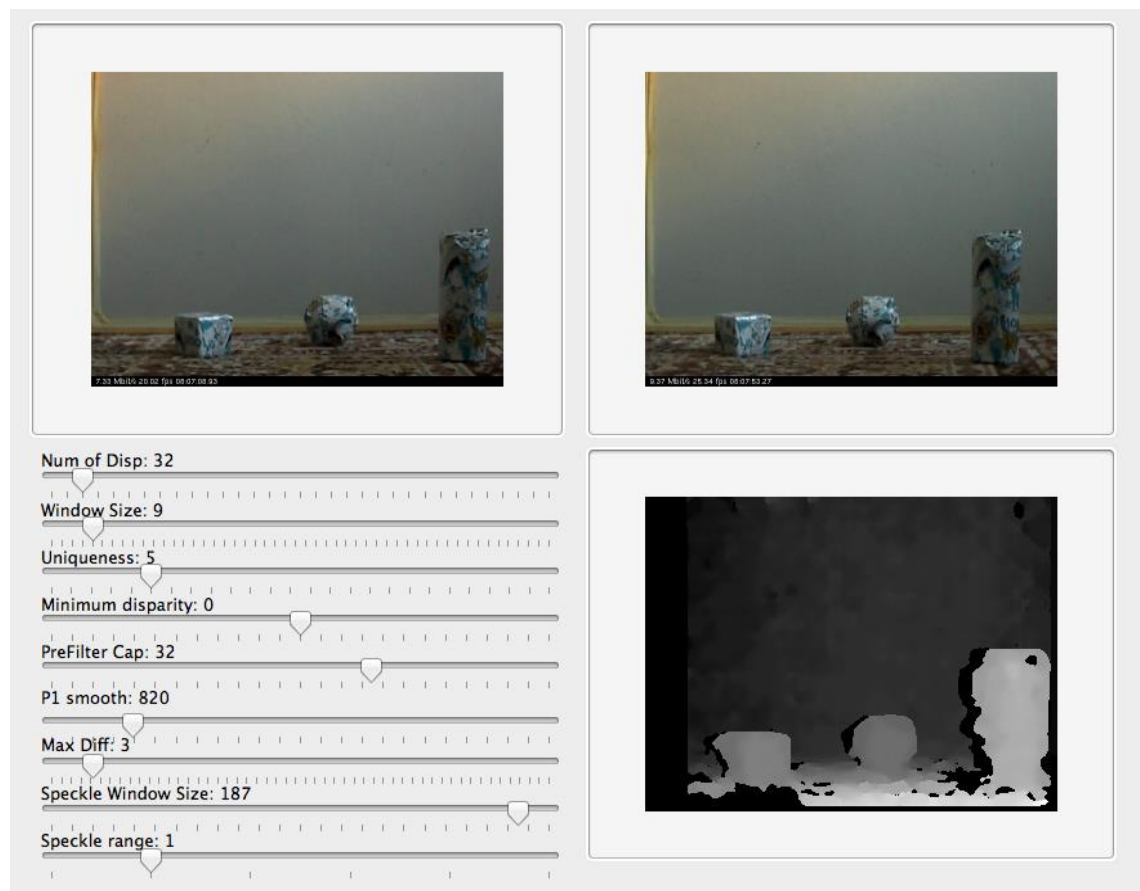


**4.13 Ábra: A sztereó kép a sakktábláról, és az epipoláris egyenesek**

Miután ezt a kérdést kitérőttem, rátérhetek a Reconstructor osztályban használt algoritmusok tárgyalására. A sztereó rekonstrukció megvalósításához segítségül hívtam az OpenCV könyvtárat, mely számos algoritmust biztosít a feladat elvégzésére. Lehetővé tettem, hogy a rekonstrukciót végző osztály az OpenCV által biztosított fő eljárásokat mind használni tudja, így végezetül három módszer került megvalósításra: Az SGBM (Semi-Global Block Matching), a HBP (Hierarchical Belief Propagation), valamint a CSBP (Constant-Space Belief Propagation) algoritmusok. Ezek közül a felhasználói a keretrendszer beállítások felületén választhat, az alapértelmezett az SGBM algoritmus. A különböző algoritmusok implementációja nem jelentett különösebb gondot, mivel a felületük megegyezik, leszámítva, hogy a második két algoritmushoz a OpenCV GPGPU implementációját használtam, mivel ezek az eljárások meglehetősen nagy számításigényűek.

Fontos megjegyezni, hogy az alapértelmezett SGBM algoritmus egyik kellemetlen velejárója, hogy számos paramétere van, amelyek jelentős mértékben befolyásolják a sztereó rekonstrukció minőségét. Ezért az algoritmus beépítése előtt készítettem egy

külön Macintosh alkalmazást (4.14 Ábra), melynek segítségével kikísérleteztem egy megfelelő alapértelmezett beállítást az algoritmus paramétereire.



4.14 Ábra: A rekonstrukció-hangoló alkalmazás

## 4.2.2 Szegmentáció és alakleírás

A jelenlegi fejezetben a szegmentáció és az alakleírás kérdéseivel fogok foglalkozni, bemutatva a Segmenter osztályt feltöltő algoritmusokat. A fejezet logikai felépítése meglehetősen hasonlít a 3.3 fejezethez. Először bemutatom a módszereket, amiket egy hagyományos objektumjelöltek előállítására használt szegmentáló osztályhoz használtam, majd a fejezet második részében bemutatom az alakzat alapú szegmnetáló és alakleíró algoritmust, amelyet a szegmentálás önkényességének elkerülése érdekében használtam.

### 4.2.2.1 Összefüggőség alapú szegmentáció

A munka korai szakaszában nem kívántam a szegmentálással külön foglalkozni, de a térben elkülönülő objektumokat mégis szét kellett választani. Ezért egy nagyon

egyszerű szétválasztást alkalmaztam, amely feltételezi, hogy különböző objektumok nem takarják ki egymást. Az eljárás lépései a következők voltak:

- A diszparitás képet küszöböztem, hogy a háttérben lévő falat kiszűrjem a képből
- A kapott bináris képen kontúrokat kerestem
- A kapott első szintű kontúrok területét meghatároztam (a bennfoglalt magasabb szintű kontúrokat nem vesszük figyelembe, mert azok valószínűleg a zaj eredményei)
- A kellően nagyméretű kontúrokat megtartom, és a segítségükkel maszkoltam a diszparitás képet
- A kontúrok által kimaszkolt objektumokat külön-külön az OpenCV `reprojectImageTo3D` függvényével pontfelhővé konvertáltam.

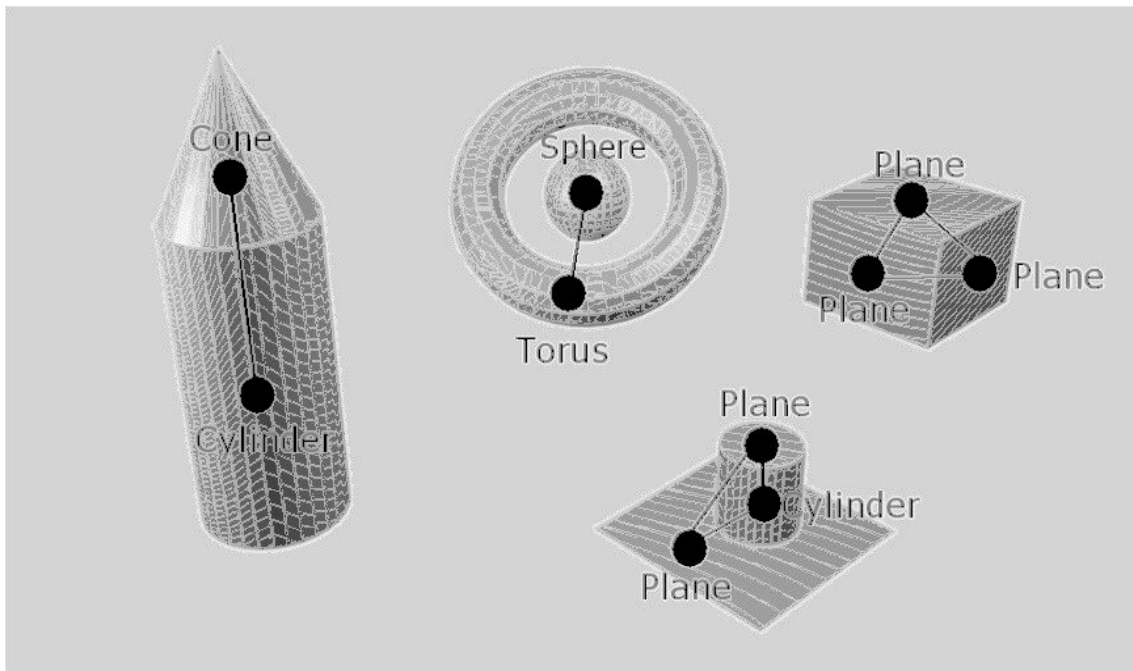
A munka későbbi részében szükségessé vált olyan szegmentáló módszer, amely képes az egymást átfedő objektumok szétválasztására, ekkor a korábban bemutatott mélység réteges [18] megoldást implementáltam. Fontos megjegyezni, hogy egyik szegmentáló eljárás sem képes egymáshoz érő, de különböző objektumok szétválasztására. Ez a probléma megoldható, ha az egyetlen objektum, ami a többihez hozzáérhet az asztal, mivel annak a pozícióját tudjuk, és kiszűrhetjük. Amint viszont kiterjesztjük a rendszert a prototípuson túl, már figyelembe kell vennünk ezt a problémát.

#### **4.2.2.2 Forma alapú szegmentáció**

Már korábban tárgyaltam a fenti szegmentáló módszerek hátrányát és elégtelenségét a megvalósítandó rendszer szempontjából. Ezért a munka későbbi fázisában szerettem volna egy olyan szegmentáló algoritmust létrehozni, amely nem tesz erős feltételezéseket a jelenetben előforduló objektumokra vonatkozóan, valamint nem próbálja ebben a fázisban eldönteni, hogy a pontfelhő mely részei tartoznak egy adott objektumhoz. Éppen ezért Schnabel et. al. Algoritmusát felhasználva [39] a pontfelhőt könnyen leírható építőelemekre bontottam, majd ebből a csoport munkájához hasonlóan [24] egy topológiai gráfot konstruáltam. (4.15 Ábra)

A módszer előnye, hogy az egyetlen szükséges feltételezés, hogy a munkatérben található objektumok többé-kevésbé leírhatóak néhány primitív forma segítségével,

valamint, hogy a konkrét objektumjelöltek építését rábízhatom egy későbbi alakfelismerő algoritmusra. További előny, hogy a primitív formák egyforma súllyal képesek reprezentálni az adott jelent, vagy objektum globális, illetve lokális struktúráját, így ezek relatív súlyát is elegendő lesz az alakfelismerő algoritmusnak meghatározni. A felhasznált algoritmus öt különböző primitív formát volt képes keresni: Síkot, gömböt, hengert, kúpot és tóruszt.



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

A felépített gráf csomópontjai maguk a primitív formák voltak, azonban ezekhez számos jellemző is tartozik, melyek az adott primitívek paramétereiből állnak. Ezeket a gráf minden csomópontjához egy vektor formájában eltároltam. Fontos azonban megjegyezni, hogy a különböző primitívekhez különböző jellemzők tartoznak, amelyek egymással nem feleltethetők meg, még akkor sem, ha azonos tulajdonságot takarnak. Éppen ezért az adott csomóponthoz eltárolt jellemző vektor az összes lehetséges primitív paramétereit tartalmazta, a nem értelmezett értékeket (pl. gömb esetén a henger, sík, tórusz és a gömb. paramétereit) nullára állítottam. Ezeket a paramétereket majd a későbbi algoritmusok fogják felhasználni az egyes csomópontok összehasonlítására, így a különböző típusú csomópontok mindig különbözni fognak, mivel az egyik formát leíró értékek a másik csomópontban mindig nullák lesznek. Az egyes formákhoz meghatározott jellemzőket a 4.1 Táblázat tartalmazza.

#### 4.1 Táblázat: A primitívek és a meghatározott jellemzőik

| Primitív | Jellemzők                                        |
|----------|--------------------------------------------------|
| Sík      | Átmérő<br>Terület<br>Befoglaló téglalap területe |
| Gömb     | Sugár                                            |
| Henger   | Sugár<br>Magasság                                |
| Kúp      | Sugár<br>Magasság<br>Oldal dőlésszöge            |
| Tórusz   | Sugár<br>A test sugara                           |

Fontos észrevenni, hogy a gömb kivételével az összes primitívhez tartozhat egy kitüntetett irány is: A sík normálisa, a kúp, a henger, illetve a tórusz esetén pedig forgástengely irányvektora. Ezen felül minden csomóponthoz tartozik egy kitüntetett pont is, a gömb, a sík, és a tórusz középpontja, a henger tengelyének felezőpontja, valamint a kúp talppontja. Ezen pontok és irányok segítségével definiálhatunk egy translációt és egy elfordulást az egyes primitívek között (a gömb esetén ez utóbbi mindig csupa nulla lesz). Ezek a térbeli transzformációk lesznek a gráf élei a korábban bemutatott implementációval ellentétben, ahol az élek kizárólag szomszédossági információt adtak meg. Észrevehetjük, hogy az élek hiánya az én megvalósításom esetén nem szolgáltat semmilyen információt, mivel az egyes csomópontok szomszédossága (távolsága) az élek paramétereinek között szerepel, ezért teljes gráfot hozok létre. Ennek további előnye, hogy a primitívek szomszédossága bináris helyett egy valós számmal leírható tulajdonság, ami robusztusabb leírást eredményez.

#### 4.2.3 Alakfelismerés

Az alábbi fejezetben az alakfelismerő algoritmus megvalósítását tárgyalom, bemutatva a Matcher, illetve a GPUAI osztályok magját képező algoritmusokat. Hasonlóan az ezt megelőző fejezethez, az első szekció témája az összefüggőségi alapú

szegmentációs algoritmus által szolgáltatott objektumjelöltek felismerése, míg a fejezet második, fajsúlyosabb részében a gráfokon működő tanuló alakfelismerő algoritmust mutatom be.

#### 4.2.3.1 Primitív felismerés

Az adaptív rendszer fejlesztésének korai fázisában a Matcher osztály fő eljárásának bemenete több objektumjelölt volt, a hozzájuk tartozó pontfelhő is rendelkezésre állt. Mivel ezt az algoritmust egyfajta proof-of-concept céllal fejlesztettem, megtehettem még egy egyszerűsítő feltételezést: Előírtam, hogy a szereplő objektumok formája csak henger, gömb, téglatest, vagy síkfelület lehet. Ekkor a RANSAC algoritmus egy egyszerűbb változata megfelelő lehet a formák klasszifikálására.

A PCL [38] könyvtárban elérhető egy RANSAC implementáció, amely paraméteres modelleket képes illeszteni egy pontfelhőbe, és visszaadja a legjobban illeszkedő becsült modell paramétereit. Az implementált eljárásnak három korlátja van. Az egyik, hogy egyszerre csak egyfajta modellt képes keresni, vagyis minden keresett modellfajtára külön kell keresni ugyanabban a pontfelhőben. Ez azt jelenti, hogy az implementáció nem képes dönteni az adott felhő formájáról, így ezt nekem kell elvégezni. A másik korlát, hogy ez a változata az algoritmusnak minden pontfelhőre pontosan egy modellt ad vissza, ezért ha a szegmentálás nem sikerül, akkor az megzavarhatja az algoritmust. A harmadik korlát, hogy téglatestet nem tud illeszteni, csak síkot, így ezt is nekem kell megoldani. A téglatest keresését síkok keresésének segítségével oldottam meg az alábbi módon:

- Keresek egy síkot a felhőben, majd a síkhoz tartozó pontokat elvesze a felhőből
- Ha a megmaradt felhő egy határértéknél kevesebb pontból áll, akkor úgy döntök, hogy az adott alakzat sík
- Ellenkező esetben a maradék pontok között egy újabb síkot keresek
- Ha második sík megfelelően nagy szöget (a gyakorlatban ez ritkán lesz pontosan 90 fok) zár be az első síkkal, akkor úgy döntök, hogy a talált objektum téglatest

Észrevehetjük, hogy a téglatest kereső algoritmus máris elvégez egy döntést a sík illetve a téglatest modell használata közt. A hátralévő döntéseket alapvetően az alapján hozom meg, hogy a három közül melyik modell illeszkedik jobban a pontfelhőre (melyik foglal magába több pontot).

- Megnézzük, hogy az objektum magassága közel van-e a szélességéhez. (+/- 25%) Ha nem akkor az objektum nem lehet gömb, és csak a henger modell keresést futtatjuk le
- Ha lehet gömb, akkor gömböt és hengert is keresünk, majd a fent leírt módon összehasonlítjuk az eredményeket

Fontos megjegyezni, hogy a RANSAC algoritmus nem ad vissza minden paramétert, amire az alakzatok meghatározásához szükségem lenne. Gömb esetén visszaadja a középpontot, és a sugarat, ami elégséges is. Henger esetén az algoritmus visszaadja a tengely egy pontját (hogy melyik pontját, azt nem tudjuk, ugyanis az algoritmus tengely mentén végtelen kiterjedésű modellt használ), a tengely irányvektorát, és a henger sugarát. A jelenetben természetesen egy véges henger található, ezért ki kell számolni a henger talppontját és magasságát. Téglalap esetén az egyik sík normálvektorát és egy pontját adja vissza az algoritmus (a másik sík paraméterei gyakran megbízhatatlanok, mivel általában kevés pontból áll), de ennek alapján először meg kell határozni egy kitüntetett pontot (a téglatest szemből nézve jobb felső sarka), az első sík normálvektora körüli forgatást, és a három iránybeli kiterjedéseket.

A henger paramétereinek kiszámítása relatíve egyszerű: A henger modelljébe illő pontokat elfogatom úgy, hogy a tengely irányvektora párhuzamos legyen az y tengellyel. Innen a henger magassága és talppontja megkapható az y irányban vett maximum és minimum segítségével.

A téglatest esete lényegesen bonyolultabb. Ebben az esetben először a forgatásokat kell meghatározni, amihez csak az első síkba illő pontokat használjuk fel. Az eljárás:

- Elforgatom a síkot úgy, hogy a normálvektora párhuzamos legyen a z tengellyel.
- Ezután kiszámolom az x és y irányban található minimum és maximum pontok koordinátáit, valamint az ezek által meghatározott téglalap területét.

- Ezután elforgatom a z tengely körül pozitív és negatív irányba is 5 fokkal a pontokat, majd újra kiszámolom a körbefoglaló téglalapok területét.
- Ha mindkét új téglalap kerülete nagyobb, mint az előzőé, akkor az algoritmus véget ért, mivel a körbefoglaló téglalap kerülete akkor minimális, ha valódi szintén téglalap alakú pontfelhőnk oldalai pont az x-y irányban állnak.
- Ha valamelyik kisebb, akkor abba az irányba indulva további 5 fokkal forgatom a felhőt egészen addig, amíg a minimális kerületet el nem érem.
- Ha mindkét irányba csökken a kerület, akkor a téglalap oldalai nagyjából 45 fokos szöget zárnak be a tengelyekkel, ezért mindegy, hogy merre indul az algoritmus. Ennek ellenére az algoritmus konzisztens módon abba az irányba indul, amerre nagyobbra csökkent a kerület, mivel az optimum jó eséllyel erre közelebb van.

Miután a téglalapot teljes orientációját meghatároztam, már a jobb felső sarka, és a méretei is egyszerűen számolhatóak. Ehhez a meghatározott orientációval el kell forgatni a teljes pontfelhőt (nem csak az első síkot), majd a hengerhez hasonló módon a minimum és a maximum pontok koordinátái segítségével a hiányzó paraméterek számolhatóak.

#### 4.2.3.2 Tanuló alakfelismerés

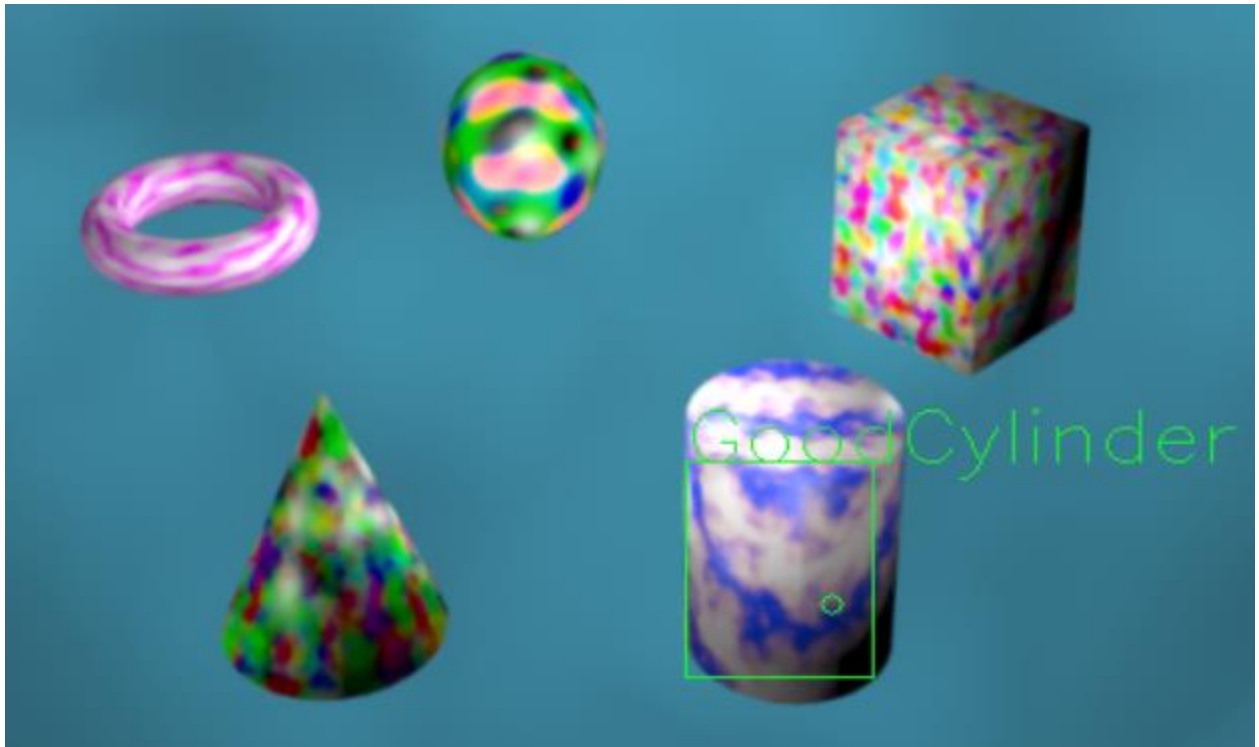
Az előző fejezetben bemutatott algoritmus meglehetősen gyors volt, azonban a helyes működéshez szükséges feltételezések túl erősek voltak ahhoz, hogy a bevezetésben és a specifikáció során leírt működést biztosítani tudja. Éppen ezért az alakleíró és –felismerő rendszert egy lényegesen flexibilisebb, intelligensebb változatra kellett cserélni. Erre használtam fel a 4.2.2.2 fejezetben bemutatott primitív gráf készítő algoritmust, a jelenlegi fejezetben pedig az ezt teljessé tevő tanuló alakfelismerésre épülő lokalizációs algoritmust fogom bemutatni. Először bemutatom a lokalizációs algoritmus működési elvét, ezt követően pedig részletezem a mögötte lévő tanuló eljárás működését. Az alfejezet végén a tanuló algoritmus implementációs kérdéseire fogok kitérni.

Korábban már többször említettem, hogy az objektumjelöltek előállításának feladatát áthelyeztem a szegmentáló algoritmusból a felismerő eljárásba. A lokalizációs algoritmus feladata lesz tehát a jelenetet leíró gráfból az egyes valós tárgyaknak megfelelő részgráfok kiválasztása, ezt követően pedig az algoritmus előállít számos

objektumjelöltet a részgráfok segítségével. A lokalizáció lényeges második lépése a jelöltek közti esetleges konfliktus feloldása, valamint a végleges objektumok előállítása. (4.16 Ábra) Az első feladatot, vagyis a jelöltek előállítását az eljárás egy részgráf növesztő eljárás segítségével végzi. A lokalizáció lépései az alábbiak:

1. Minden kategóriára külön:
  - 1.1. Az egyes csomópontokhoz jószág értékek hozzárendelése a tanuló alakfelismerő algoritmus segítségével
  - 1.2. Amíg van nagy jószág értékű csomópont:
    - 1.2.1. Egy csomópontból álló gráf készítése
      - 1.2.1.1. Újabb, közeli csomópont hozzáadása az eddig kiválasztott csomópontokhoz
      - 1.2.1.2. Az új gráf kiértékelése az alakfelismerő algoritmus segítségével
      - 1.2.1.3. Ha az alakfelismerés eredménye javult, akkor ugrás a 1.2.1.1. lépésre
    - 1.2.2. Az elkészült gráf validálása az alakfelismerés segítségével
  - 1.3. Az elkészült objektumjelöltekhez leginkább hasonló tanító gráf kiválasztása
  - 1.4. A jelöltek és a tanító gráfok megfelelő csomópontjainak alapján LS becslés segítségével a jelöltek térbeli pozíciójának és orientációjának meghatározása
2. Az átfedő objektumjelöltek közül az alakfelismerés eredménye szerinti legjobbak kiválasztása

A 2. lépés kapcsán felmerülhet a kérdés, hogy mi történik, ha a különböző kategóriákat felismerni képes tanuló modellek eredménye más értékek közt mozog, ekkor ugyanis egyes kategóriákat folyamatos előnyben részesítene a berendező lépés. Ennek elkerülésére a különböző modelleket mindig azonos hiperparaméterek mellett tanítom be, így nagyjából ugyanabban a tartományban mozognak az értékek, ám erre nincs matematikai garancia.



**4.16 Ábra: Egy henger lokalizációja**

A fenti leírásban több helyen hivatkoztam a tanuló alakfelismerő algoritmusra, így fontos ezt az algoritmust is részletesen ismertetni. Arra, hogy az egyes gráfokhoz jószág értékeket rendeljek hozzá bináris klasszifikáló algoritmust használok, egész pontosan annak a küszöbözés előtti értékét. A konkrét felhasznált módszer az SVM, vagyis a Support Vector Machine algoritmus, amely egy belső kernel alapú véletlen séta kernel függvény segítségével működik. Az SVM algoritmusra azért esett a választásom, mivel ez az algoritmus könnyen képes kernel függvényekkel működni, és szinte minden implementáció támogatja a saját kernel használatát.

Fontos azonban említést tenni a benne felhasznált véletlen séta kernelről. Ha visszaemlékezünk a 3.4 fejezetben történt bemutatásra, akkor láthatjuk, hogy a kernel a két összehasonlítandó gráf direkt szorzatát igényli a működés közben. A jelenleg használt gráfokban azonban sem a csomópontokhoz sem az élekhez nem rendelhetők valós számok, hanem vektorok írják le ezeket. Így a direkt szorzat számítás során, ahol két csomóponti vagy él súly szorzatára van szükség, ezt a valós számot egy belső kernel segítségével kell előállítanunk. A következő néhány bekezdésben röviden bemutatom a használt él és csomóponti kerneleket.

Az él kernel feladata tehát, hogy két élt a jellemzőik alapján összehasonlítsa, majd ezt egy valós számmal jellemezze. Az összehasonlításhoz az él jellemzői közül azonban csak kettőt veszek figyelembe: az élhez tartozó translációs vektor hosszénegyzetét, valamint a rotációt leíró kvaternió  $\cos(\frac{\alpha}{2})$  részét. A többi jellemző kihagyásának oka, hogy nem invariánsak olyan transzformációkra, amikre az alakfelismerésnek annak kellene lennie (pl.: elforgatás). Az él kernelt egy RBF kernel segítségével számolom az alábbiak szerint:

$$E = e^{-\gamma \Delta \alpha - \delta \Delta d - \mu d} \quad (6)$$

Ahol  $\mu$ ,  $\gamma$  és  $\delta$  az él kernel hiperparaméterei,  $\Delta \alpha$  a két élhez tartozó  $\cos(\frac{\alpha}{2})$  részek különbségének abszolút értéke,  $\Delta d$  az élekhez tartozó távolságnegyzetek különbségének abszolút értéke, míg  $d$  a távolságnegyzetek átlaga. Az átlagos távolság használata fontos eleme a kernelnek, holott az élek hasonlóságát nem befolyásolja. Ez azonban kiváló hely arra, hogy a távoli csomópontok közti kapcsolatokat büntessem, mivel alapvető feltételezésem, hogy az objektumok lokális jelenségek.

A másik belső kernel függvény a csomóponti kernel, mely két csomópont hasonlóságát adja meg. A csomópontokat leíró vektorokban található jellemzők azonban alapvetően különböznek a jellegükben (távolság, terület, vagy szög típusú jellemzők), ebből következően viszont a nagyságrendjükben is. Ez a vektorok összehasonlítását megnehezíti, így előbb egy normalizált eltérés értéket hozok létre a csomóponti vektorokból (7), majd utána erre alkalmazok egy RBF kernel függvényt (8).

$$W = \sum_{i=1}^M \frac{|a_i - b_i|}{a_i + b_i} \quad (7)$$

$$N = e^{-\vartheta * W} \quad (8)$$

Ahol az összehasonlítandó vektorok  $A = [a_1 \ a_2 \ \dots \ a_M]$  és  $B = [b_1 \ b_2 \ \dots \ b_M]$ ,  $\vartheta$  pedig egy újabb hiperparaméter. Egy fontos változtatást eszközöltem még a véletlen séta algoritmusban: Mivel azt tapasztaltam, hogy a gráfok méretétől jelentősen függött az kernel végeredménye, ezért egy extra normalizáló lépést eszközöltem. (9)

$$K_f = \frac{K}{(m * n)^\varepsilon} \quad (9)$$

Ahol  $m$  és  $n$  a két gráf mérete,  $K$  a véletlen séta eredménye,  $\varepsilon$  pedig egy újabb hiperparaméter. Az implementáció szempontjából két fontos hiperparamétere van még a véletlen séta kernelnek: Az egyik az  $N$ , vagyis a leghosszabb figyelembe vett séta hossza, a másik pedig  $s$ , vagyis a súlyfüggvény meredeksége. A más így is számos hiperparaméterhez még hozzáadódik az SVM algoritmus  $C$  paramétere, így összesen 8 paramétere lesz a tanuló alakfelismerő algoritmusnak.

Az előbb említett hiperparaméterek hangolása rendkívül fontos feladat a tanuló algoritmusok használatakor, ugyanis e nélkül az algoritmus könnyedén beleeshet a (gépi) tanulás alapvető hibáinak egyikébe. Az algoritmusok hiperparamétereinek célja ugyanis, hogy a tanuló algoritmust regularizálják, vagyis, hogy megteremtsék az egyensúlyt az elért tanítási hiba, és a megtanult modell bonyolultsága között. Ha ez az egyensúly nincs meg, akkor az alul-, vagy a túlillesztés (under-, overfitting) hibájával nézünk szembe.

A paraméterek hangolásához valamilyen optimalizáló eljárást célszerű alkalmazni, azonban a numerikus módszerek alkalmazása a bonyolult saját kernel függvény miatt rendkívül nehéz lenne. Alkalmazhatnám még az egyszerű paraméterhálón történő keresést is, azonban ez a hiperparaméterek száma miatt nem kivitelezhető. Éppen ezért egy genetikusan algoritmus használatát mellett döntöttem, mivel ez képes kevesebb kiértékelés árán nagyobb eséllyel az optimum közelébe kerülni.

Az optimumkeresés alapvető kérdése viszont, hogy mi legyen az a mérőszám, amit minimalizálni/maximalizálni szeretnénk. A gépi tanulás esetén akad egy kézenfekvő megoldás: a kereszt-validációs pontosság használata. Ezt a számot a kereszt-validáció segítségével lehet előállítani. Az eljárás kezdeti lépése a tanító adatok  $N$  (általában 5, vagy 10) egyelő részre osztása, majd az algoritmus  $N-1$  rész felhasználásával történő betanítása. Ezt követően a megtanult modell segítségével klasszifikálható a kimaradt rész. A tanítás és a klasszifikáció folyamatát eztán el kell végezni mindig egy különböző rész kihagyásával, így előáll egy becslés az összes tanító adatra, amelyek közül a helyes becslések aránya a kereszt-validációs pontosság. Látható, hogy a tanítási hibával szemben a kereszt-validációs pontosság a tanuló algoritmus általánosító képességét méri.

A fent bemutatott kernel függvényt először MATLAB segítségével valósítottam meg, mivel itt könnyedén lehet mátrix- és vektorműveleteket elvégezni. Az SVM implementálására a bioinformatika toolbox implementációját használtam. Miután a kernelt a MATLAB környezetben teszteltem, elkészítettem a C++ implementációt is a LibSVM [41] könyvtár felhasználásával. Az implementáció fontos lépése volt a kernel függvény párhuzamosítása a grafikus kártya segítségével, amit a számos lineáris algebra művelet tett lehetővé. A párhuzamosítással nagyjából 25-50-szeres gyorsítást értem el a MATLAB implementációhoz képest.

Éppen ezért a tanuló algoritmus alapvető szolgáltatásai a GPUAI osztályba kerültek implementálásra, amely egy teljesen statikus algoritmikus osztály. Implementálja a tanító, a lokalizáló, valamint a kereszt-validáció alapú optimalizálást végző algoritmust. Ez utóbbi implementálásához a GaLib [42] könyvtárat használtam fel, mely többféle testre szabható genetikus algoritmust bocsájt rendelkezésre. Lehetőség van többféle genom, szelekciós, keresztező és mutációs operátor, valamint fitness érték számítás választására, valamint természetesen felüldefiniálható az egyedekhez tartozó objektív függvény.

Az adott feladathoz egy közönséges genetikus algoritmust alkalmaztam, valós számokból álló genommal, univerzális sztochasztikus mintavételezés, valamint valós mutációs és keresztező operátorok felhasználásával. Rendkívüli előny, hogy nem csak az egyes egyedek, hanem az egész populáció kiértékelő függvényét is felülírhattam, így lehetőségem volt a kereszt-validáció egyedek közti párhuzamosítására. Erre két lehetőség is volt: Egyrészt a több egyed kereszt-validációja a kernel függvény többszöri kiértékelését teszi szükségessé ugyanazokkal a bemeneti adatokkal, csak különböző paraméterek mellett. Másrészt a későbbi, CPU-n történő tanulást is el lehet osztani a több processzormag közt.

A fejezet végén fontos még részletezni a grafikus kártyán párhuzamosított függvényeket. Az egyszerű tanítás esetére az él és a csomóponti kernelek implementációját végeztem el, míg a véletlen séta fő képletében szereplő műveleteket a CUBLAS könyvtár segítségével párhuzamosítottam. A kereszt-validációhoz elkészítettem a két belső kernel függvényen kívül a mátrixszorzó függvény különböző paraméterek szerint párhuzamosított változatát is.

Az él kernel esetén egy szál (thread) felelőssége a direkt szorzat gráf adjacencia mátrixában egyetlen elem kiszámolása volt az (5) képlet alapján. Fontos megjegyezni, hogy az él kernel esetén külön figyelmet kell fordítani a hurokélek kezelésére, melyek csupa nulla paramétereik miatt tökéletesen hasonlítanak egymásra, azonban ilyen esetben a kernel eredménye is nulla kell, hogy legyen. A csomóponti kernel esetén a megosztott memória segítségével a képletben (6) szereplő summát párhuzamosítottam. A jelenlegi implementáció során a summa 14 elemből áll, így egy fonatba (warp) két csomópont értékének kiszámítása fér bele az inaktív szálak minimalizálásával. A többparaméteres változat esetén a párhuzamos szálak száma megemelkedik, mivel minden paraméterhez külön adjacencia mátrixot, vagy csomóponti vektort kell számítani.

Mint már említettem, a kereszt-validációs eljárásához külön mátrixszorzó kernelt hoztam létre, amely a CUBLAS szorzó függvényével megegyező képletet számolt ki. (10)

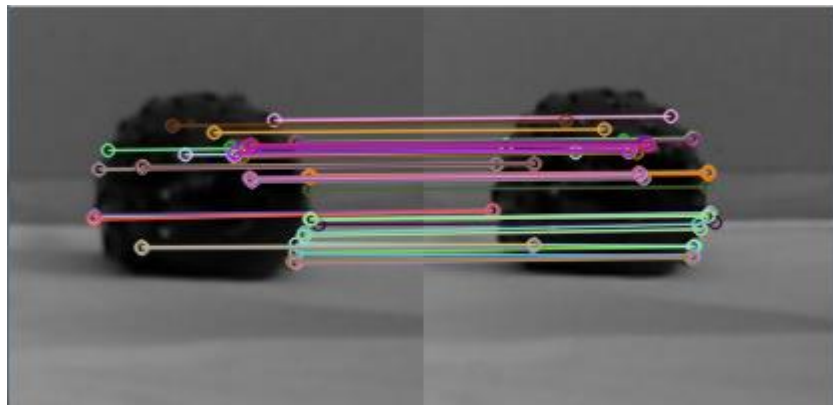
$$C = \alpha AB + \beta C \quad (10)$$

Ahol  $A, B$  és  $C$  mátrixok,  $\alpha$  és  $\beta$  pedig valós számok. Az én implementációm esetén azonban a grafikus kártya memóriájában több mátrix és hozzájuk külön paraméterek taroznak. A mátrixszorzó kernelt felhasználtam a mátrix-vektor, illetve a skaláris szorzás megoldásához is, lévén, hogy ezek a mátrix-mátrix szorzás speciális esetei.

#### 4.2.4 Követés

A jelenlegi fejezet célja, hogy a Tracker osztály feltöltésére használt algoritmust, vagyis a valós tárgyak követésére és detektálására szolgáló eljárást ismertessem. A feladat elvégzésére természetes képjellemző alapú vizuális követést fogok alkalmazni, amint az a 3.5, valamint a 4.1.2 fejezetekből sejthető volt. A képjellemzők detektálására az OpenCV könyvtár beépített eljárásait alkalmazom, lehetőség van többféle képjellemző használatára (ORB, BRIEF, KAZE, AKAZE), melyek közül az AKAZE eljárás az alapértelmezett. A különböző algoritmusok alapvetően különböznek az előállított leíró vektorok típusában, így erre a Tracker, illetve a VisualFeature osztályokat fel kellett készíteni. A követést a gépi látás területén megszokott módszerrel ellentétben azonban nem a képsíkon, hanem a háromdimenziós térben végzem el. Az algoritmus a következő:

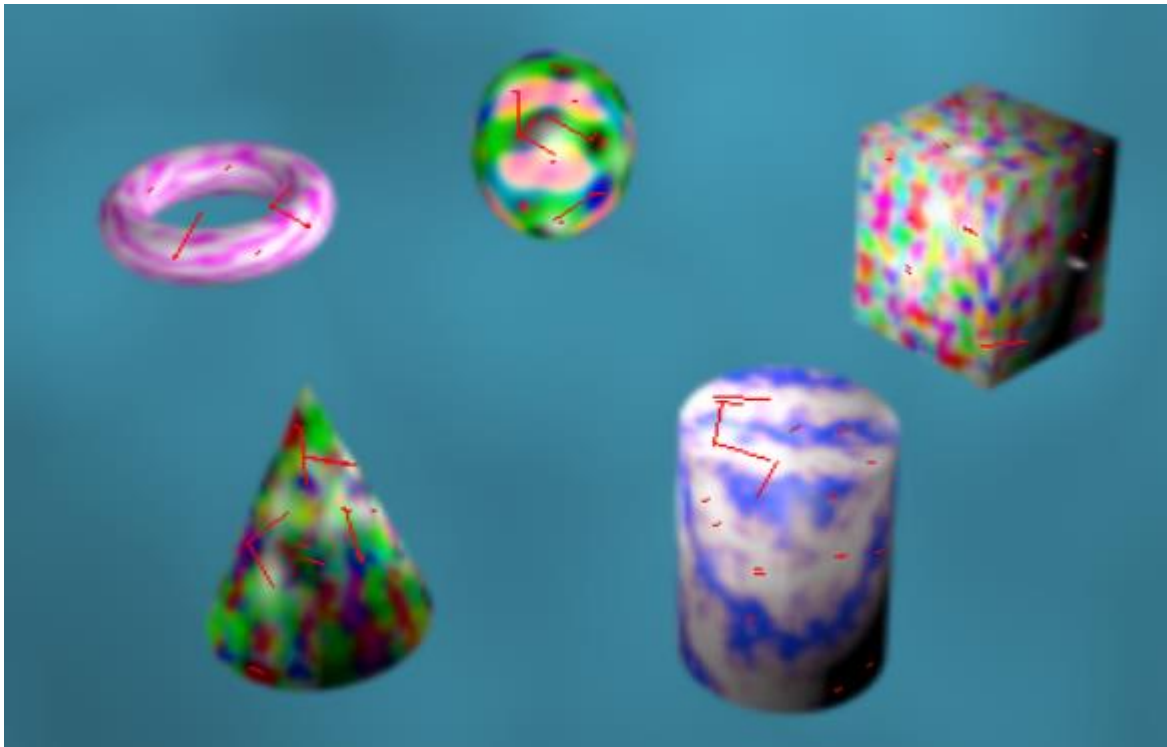
1. A kiválasztott jellemző detektor segítségével az algoritmus a sztereó képpár mindkét képén detektálja a képjellemzőket
2. Ezt követően megkísérel minden képjellemzőhöz egy sztereó párt találni a másik képkockán, és kiszámolja a 3D koordinátákat (4.17 Ábra)
3. Ezt követően a sztereó képjellemző párokhoz az algoritmus párt keres a képjellemző adatbázisban lévő jellemzők közül (4.18 ábra)
4. A detektált képjellemzőket elosztja az egyes objektumok között (4.19 Ábra), majd a korábban is megtalált jellemzők segítségével megbecsli az objektum által elszenvedett hat szabadságfokú transzformációt a két képkocka közt
5. Az objektum transzformációja és az egyes jellemzők relatív pozíciója alapján az algoritmus finomítja a képjellemzők párosítását a két képkocka közt
6. A több cikluson keresztül stabilan detektált képjellemzőket az adott objektumhoz tartozó detektálási adatbázisba elmenti



**4.17 Ábra: Sztereó képjellemző párok egy valós objektum esetében**

Az algoritmus több lépése is részletes tárgyalást igényel, elsőként a képjellemzők sztereó párosítását mutatom be. Erre az OpenCV Cookbook-ban publikált RobustMatcher [43] osztályt használom, amely többek között párosítási szimmetria, illetve epipoláris megkötés alapján teszi lehetővé a robosztus képjellemző detektálást. Szintén ezt az osztályt használom a két képkocka közti jellemző párosításra, azonban ekkor az epipoláris megkötést lecserélem két másik korlátozásra: Egyrészt, nem engedem a túlságosan nagy két képkocka közti elmozdulásokat, valamint az adott jellemző pozíció history-ja alapján megbecslem, hogy hol kéne lennie a jelenlegi kockán, és ettől sem engedek meg jelentős eltérést. A párosítás finomítása során további megkötés, hogy a

jellemzőhöz tartozó objektum transzformációja alapján is megbecslem a jellemző várható pozícióján, és ezt is előírom megkötésként.

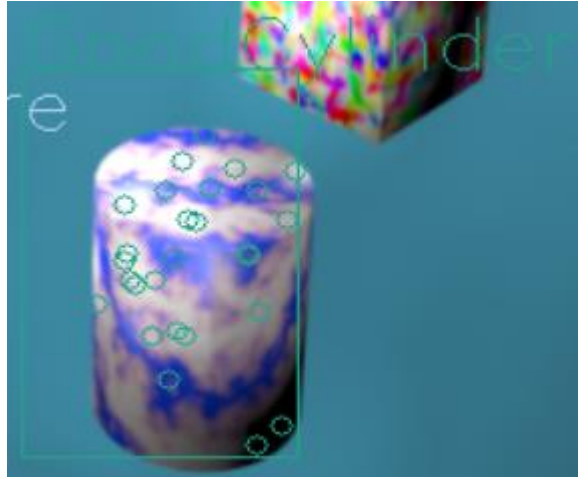


**4.18 Ábra: A képjellemzők elmozdulása két képkocka közt**

Fontos rész még az egyes objektumok által elszenvedett transzformáció megbecslése a párosított jellemzők segítségével. Ehhez először az objektum koordinátarendszerébe kell transzformálnunk a képjellemzőket. Szerencsére, mivel relatív mozgást kívánok meghatározni, a forgatás nem számít, így ez a transzformáció egy translációra egyszerűsödik. Ezt követően a hat szabadságfokú transzformációt egy saját RANSAC alapú becslő algoritmus segítségével végzem. A RANSAC algoritmus egy SVD felbontás alapú transzformáció becslő eljárást [44] használ a jelöltek előállítására, így biztosítva, hogy a transzformációban csak rigid elváltozások legyenek. További előnye a saját RANSAC használatának, hogy már a jelöltek előállításakor el tudok vetni különböző kritériumok alapján bizonyos transzformációkat (pl.: túl nagy elmozdulások, vagy elfordulások kizárása).

A fejezet végeztével fontos még szót ejteni a detektálás lépéséről. Ekkor az objektumokhoz tartozó folyamatosan karbantartott jellemző adatbázisban lévő jellemzőket próbálja az algoritmus a képen található jellemzőkkel párosítani. Erre szintén különböző megkötéseket alkalmaztam: Egyrészt az egyes jellemzők távolsága nem lehet

lényegesen különböző a jelenlegi képen, mint az adatbázisban. Másrészt, kizárom a keresésből azokat a képjellemzőket, melyek stabilan egy másik objektumhoz tartoztak, valamint, amennyiben ez engedélyezett, akkor az objektum utolsó ismert pozíciójától túl messze lévő jellemzőket is kizárom a párosításból.



**4.19 Ábra: Egy objektumhoz tartozó képjellemzők**

## 5 Tesztelés, eredmények

Az alábbi fejezetben az általam megtervezett és létrehozott rendszer, valamint a benne implementált algoritmusok tesztelését fogom ismertetni. A tesztelés eredményeinek értékelését is itt végzem el. A fejezet első részében a keretrendszer tesztelését fogom bemutatni, kitérve az elkészült rendszer összehasonlítására az előző fejezetben elkészült rendszerrel. A fejezet második felében a rendszer felöltésére használt algoritmusok tesztelését és értékelését végzem el, külön kitérve azok skálázhatóságára.

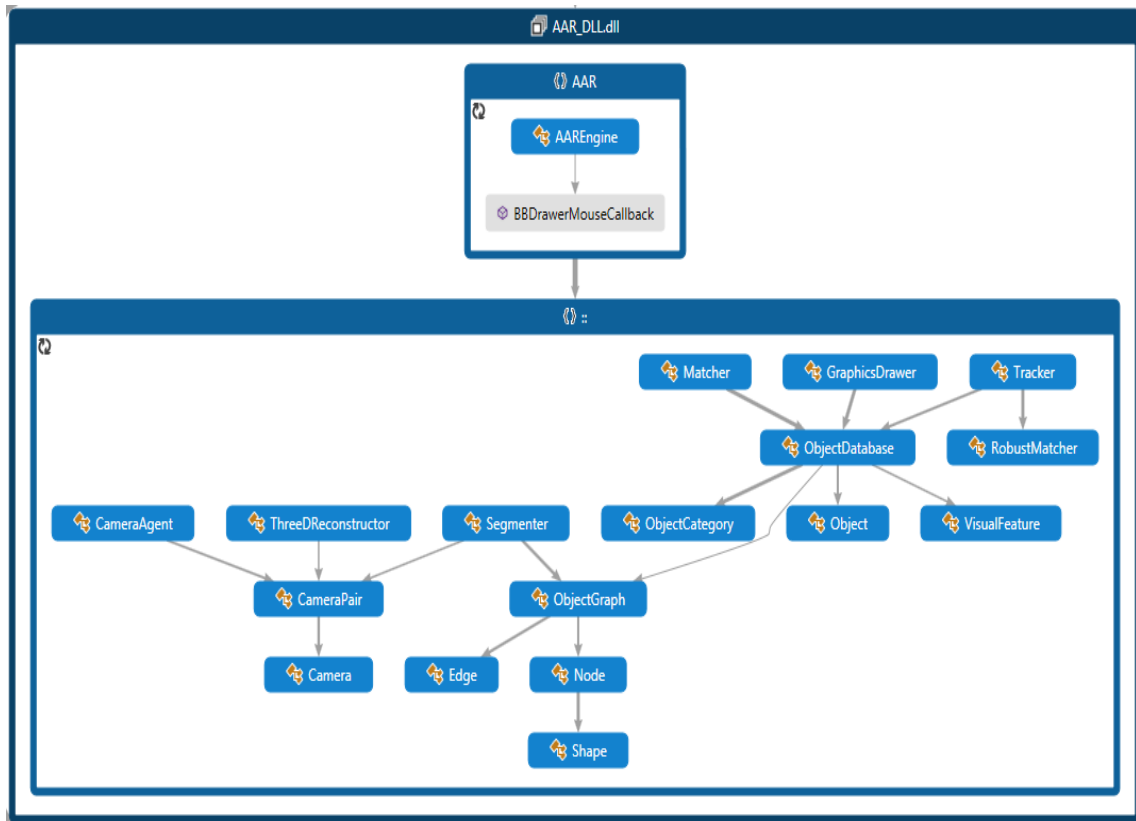
### 5.1 Keretrendszer

A jelenlegi alfejezetben a keretrendszer tesztelését és a tesztek eredményeit fogom bemutatni. A fejezet első részében az alkalmazás profilozásával foglalkozom, különösképpen a CPU, illetve az egyes algoritmusok GPU használata vizsgálatával. A fejezet második részében a keretrendszer memóriakezelésének tesztelését mutatom be.

Mielőtt azonban ebbe belekezdenék, fontos ellenőrizni, hogy a rendszer struktúrája mennyire felelt meg az előzetes terveknek. Ehhez a Visual Studio segítségével elkészítettem az alkalmazáskomponensek függőségi gráfját, amely részletesen feltérképezi az egyes osztályok függőségi viszonyait. Ez a gráf természetesen nem egyezik meg az osztálydiagrammal, mivel a közvetett függőségeket is közvetlenül ábrázolja. Azonban a közvetett függőségeket reprezentáló élek törlésével egy tiszta, áttekinthető diagramot kaphatunk (5.1 Ábra).

A gráfon látható, hogy az osztályok struktúrája nagyrészt megegyezik a 4.1.2 fejezetben mutatott osztálydiagramokkal, habár a két diagram egyszerre szerepel a függőségi gráfból készített ábrán. További fontos különbség, hogy a gráfon szerepel az AAREngine osztály, amely a modul többi osztályának kezelését és vezérlését végzi. A függőségi gráf feltüntette tovább a RobustMatcher osztályt, amely a Tracker osztályhoz tartozik. Egy további fontos eltérés az eredeti tervtől, hogy a Segmenter osztály is eléri a CameraAgent-hez tartozó CameraPair objektumot. Erre azért volt szükség, mert a jelenet elemeinek létrehozása során a Segmenter előállítja az egyes csomópontok a képek koordinátarendszerében értelmezett pozícióját, mivel ezt a képsíkon megadott befoglaló

téglalapok alapján történő tanítóadat készítés során tudni kell. A kamera síkjára való visszavetítéshez azonban szükség van az egyes kamerák projekciós mátrixaira.



5.1 Ábra: Az AAR\_DLL modul szerkezete

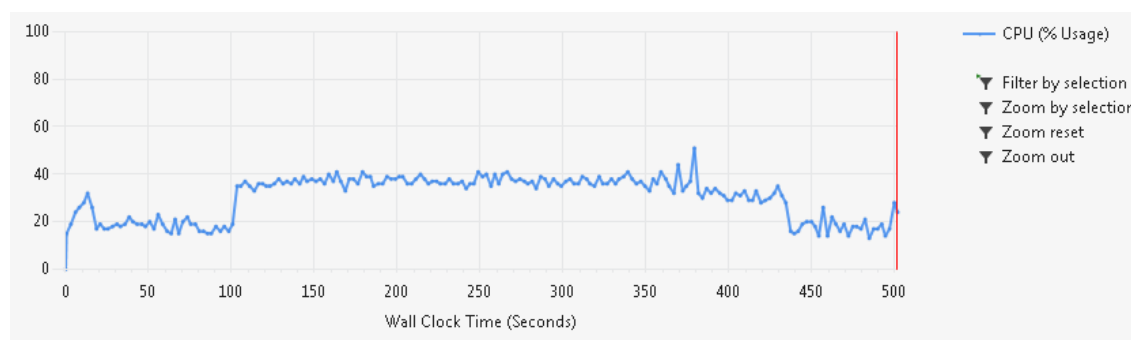
### 5.1.1 Az alkalmazás profilozása

Az alábbi fejezetben az alkalmazás különböző funkciók elvégzése közbeni profilozását fogom bemutatni. Erre a Visual Studio a profilozó eszközét használtam fel, amely részletes statisztikát képes készíteni a rendszer CPU használatáról. A profilozást három különböző funkció elvégzése közben végeztem el: Az egyik egy tanítás volt, míg a másik a virtuális környezet elkészítése, és fenntartása volt. A harmadik profilozott funkció a kereszt-validáció alapú genetikus paraméterhangoló eljárás volt. Mindhárom funkcióhoz külön bemutatom a CPU használat időfüggvényét, valamint a processzort leginkább igénybe vevő függvényeket.

A tanítás processzorhasználati időfüggvényén (5.2 Ábra) nagyjából megfelel az elvárásoknak. Látható, hogy a tanítás kezdetekor elvégzett kernel számítás (kb. 100.-200. másodperc közt) meglehetősen megterheli a processzort. Az ezt követő tanítás, amely nagyjából a 440. másodpercig tart szintén egy processzormagot foglal le teljesen. A

tanítás hosszú idejének fő oka, hogy szándékosan sok tanító adatot használok, hogy elég hosszú idősor keletkezzen. A tanítás során ráadásul az SVM C paraméterét egy paraméterháló segítségével optimalizálom.

A tanítás közben leginkább használt függvények (5.3 Ábra) nem okoznak meglepetést. A két leginkább használt függvénycsoport a CUBLAS, illetve a CUDA kerneleket megvalósító könyvtárakban vannak. A maradék három legaktívabb függvény pedig az SVMLIB-ben található optimalizáló algoritmus segédfüggvényei. Fontos látni, hogy az alkalmazás a legtöbb időt a GPU kernelekben, illetve harmadik fél által készített könyvtárakban tölti, így az esetleges sebességnövelést is ezek optimalizálásával kell elvégezni, a keretrendszer futási ideje ezekhez képest elhanyagolható.



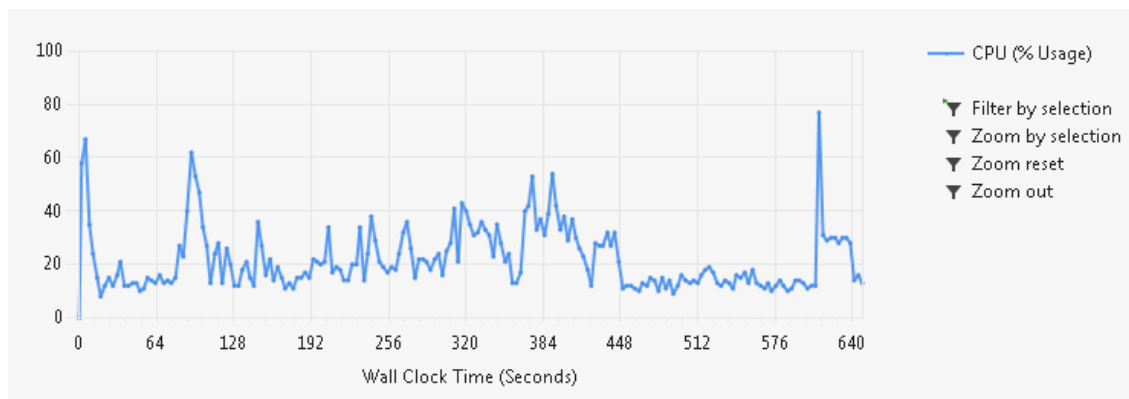
5.2 Ábra: A tanítás CPU használatának időfüggvénye



5.3 Ábra: A tanítás során leginkább aktív függvények

Rendkívül hasonló eredményeket láthatunk a paraméterhangolás CPU használata (5.4 Ábra) és a közben legaktívabb függvények listáján (5.5 Ábra). Bár a kernel számítása közben a terhelés kisebbnek tűnik, ez valószínűleg a jobb kihasználtság következménye, míg a tanítás végzése esetén láthatjuk, hogy a CPU kihasználtság megközelíti a 100%-ot, ami a egyedek közti párhuzamosítás eredménye. A leginkább használt függvények közül eltűnt a CUBLAS, mivel a kereszt-validáció esetén saját mátrixszorzó kernelt használtam. Látható, hogy a tanítás itt kevésbé fajsúlyos eleme a processzorhasználatnak, melynek oka, hogy a C paraméter szerinti optimalizálást is a genetikus algoritmus végzi el,

valamint, hogy kevés egyedből álló populációt alkalmaztam. Ezt azért tettem, mivel nem akartam, hogy az algoritmus túl sokáig fusson.



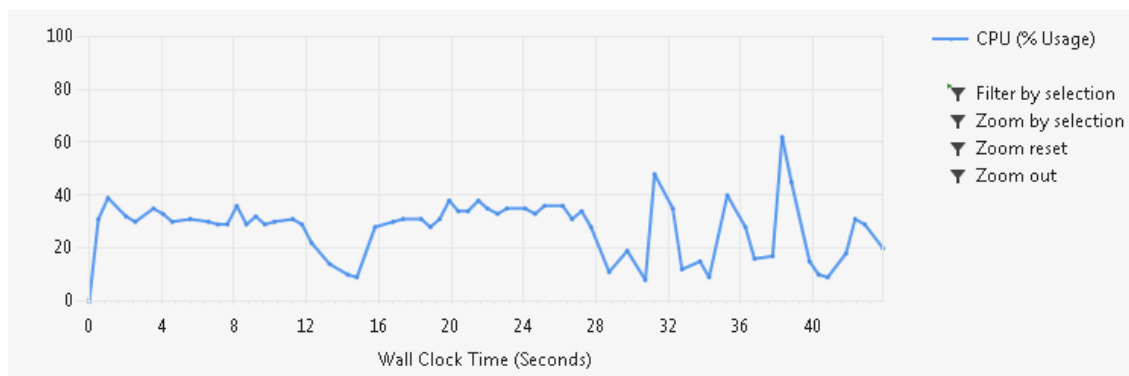
5.4 Ábra: A paraméterhangolás processzorhasználatának időfüggvénye

| Name                       | Exclusive Samples % |
|----------------------------|---------------------|
| [nvcuda.dll]               | 55.91               |
| Solver::select_working_set | 6.34                |
| __chkstk                   | 4.25                |
| Solver::is_upper_bound     | 2.71                |
| Solver::is_lower_bound     | 2.41                |

5.5 Ábra: A paraméterhangolás közben leginkább aktív függvények

A környezet létrehozása és fenntartása (5.6 és 5.7 Ábrák) során már eltérő eredményeket kapunk. A CPU használat során A berendező lépés során (16.-28. másodperc) azt látjuk, hogy egy magot leterhel a berendező algoritmus, míg az ezt követő futtatás (30-40 másodperc) során a terhelés egyenetlen, a képjellemző detektálás kiugró értékeket okoz. A legtöbbet dolgozó függvények közt is az OpenCV Features2D modulját találhatjuk, amely a képjellemzők detektálását végzi. A második legtöbb munkát ismét a GPU-n implementált algoritmus végzi, amely a berendezés során fut le. Megjelenik továbbá az OpenCV Calib3D modulja, amely a 3D rekonstrukciót végzi. Itt is megjegyzem, hogy a rendszer sebességét láthatóan nem korlátozza a keretrendszer.

A profilozás fontos második lépése volt, a GPU-n implementált algoritmusok vizsgálata hatékonyság és a grafikus kártya kihasználásának szempontjából. Ezt a vizsgálatot az Nvidia által készített NSight grafikus debugger és profilozó alkalmazás segítségével végeztem el. A profilozást elvégeztem mind a tanításkor és lokalizációkor, mind a paraméterhangoláskor használt véletlen séta kernel implementációra.



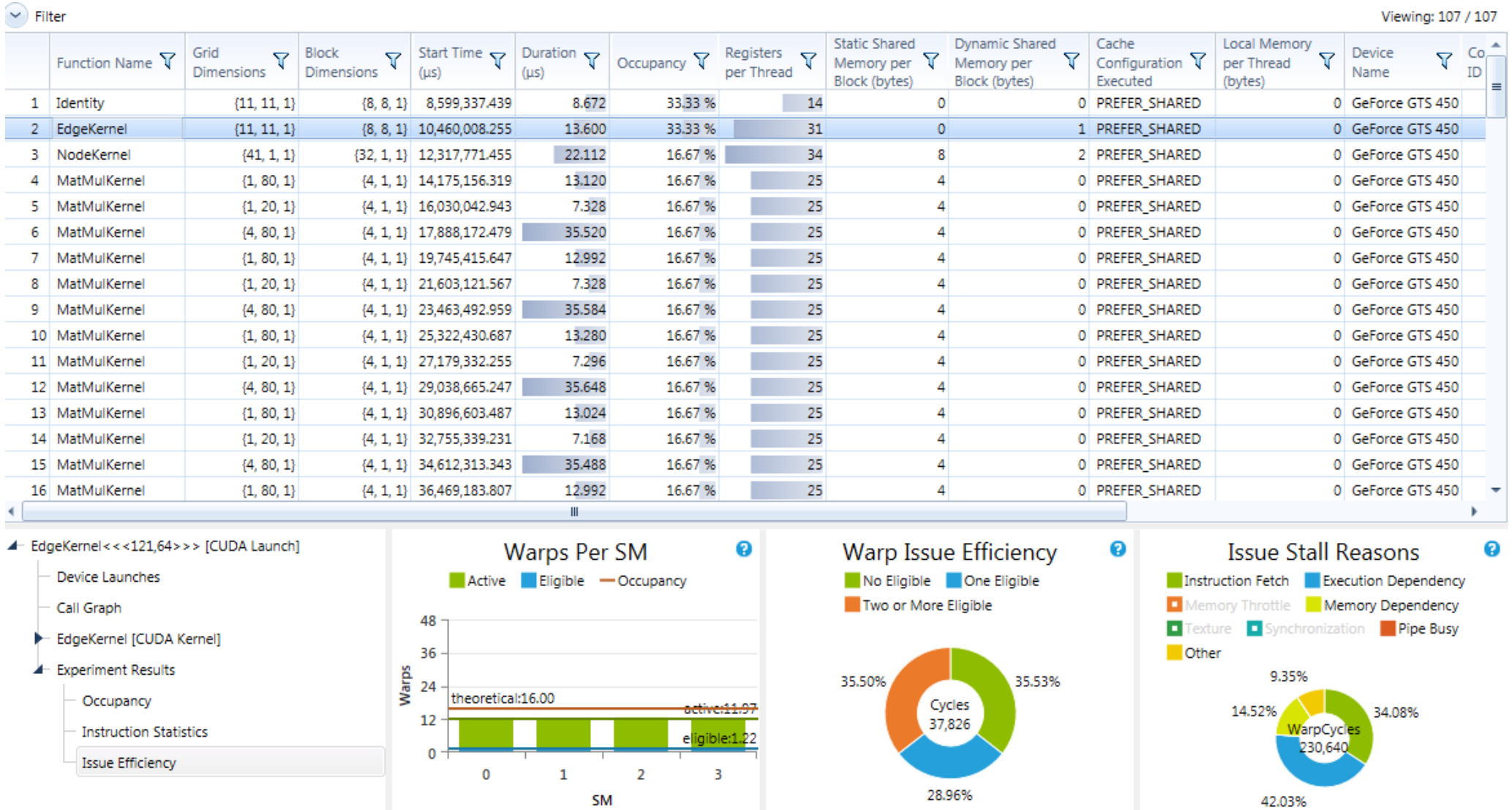
5.6 Ábra: A környezet működtetés processzorhasználatának időfüggvénye

| Name                        | Exclusive Samples % |
|-----------------------------|---------------------|
| [opencv_features2d300d.dll] | 13.54               |
| [nvcuda.dll]                | 13.27               |
| [ntdll.dll]                 | 4.15                |
| [opencv_calib3d300d.dll]    | 3.83                |
| _FDunscale                  | 3.15                |

5.7 Ábra: A környezet fenntartása közben legaktívabb függvények

A tanítás és a lokalizáció (5.8 Ábra) során használt kernel csupán a kiszámolandó mátrixok elemeinek szintjén végez párhuzamosítást. Látható, hogy ez elért kihasználtság meglehetősen alacsony, amelynek azonban nem a rossz implementáció, hanem a használt mátrixok kis mérete az oka. Éppen ezért fontos lehetőség az egyes kernelek tanító példák szintjén történő további párhuzamosítása a hatékonyság növelésének érdekében, ami természetesen jelentősen növelné a kernel komplexitását. Érdeemes megjegyezni, hogy a kernel futás idejének nagy részét a CUBLAS lineáris algebra kernelei tették ki, így az legnagyobb gyorsítást a véletlen séta kernel fő képletének hatékonyabb megvalósítása jelentené.

A paraméterhangolásra használt kernel függvény esetén (5.9 Ábra) az él és a csomóponti kernelek esetén a GPU kihasználtsága lényegesen jobban közelíti az elméleti maximumot, pedig az egyetlen különbség, hogy a párhuzamosítást a paraméterek szintjén is elvégeztem, így egyszerre több műveletet végez az algoritmus párhuzamosan. Látható azonban, hogy a saját mátrixszorzó kernel lényegesen rosszabb teljesítményű a CUBLAS-beli megfelelőjéhez képest, amelynek a legfőbb oka, hogy az implementáció során az áttekinthetőségre összpontosítottam, valamint, hogy nem használtam a grafikus kártya osztott memóriáját a gyorsítás érdekében, amit viszont (ahogy az az 5.8 Ábrán látszik) a CUBLAS megtesz.

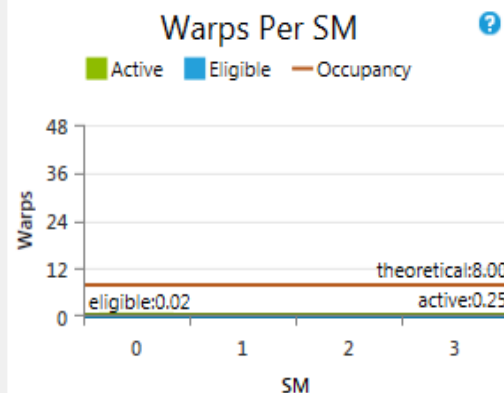


5.8 Ábra: A tanításra használt kernel függvény profilja

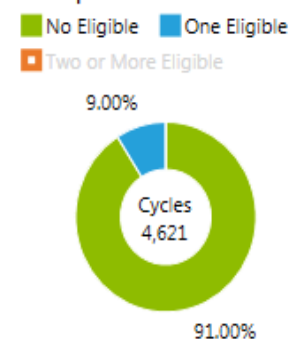
|    | Function Name                                                 | Grid Dimensions | Block Dimensions | Start Time (μs) | Duration (μs) | Occupancy | Registers per Thread | Static Shared Memory per Block (bytes) | Dynamic Shared Memory per Block (bytes) | Cache Configuration Executed |
|----|---------------------------------------------------------------|-----------------|------------------|-----------------|---------------|-----------|----------------------|----------------------------------------|-----------------------------------------|------------------------------|
| 1  | Identity                                                      | {1, 1, 1}       | {4, 4, 1}        | 16,940,333.070  | 3.392         | 16.67 %   | 8                    | 0                                      | 0                                       | PREFER_SHA                   |
| 2  | EdgeKernel                                                    | {1, 1, 1}       | {4, 4, 1}        | 17,144,317.934  | 6.688         | 16.67 %   | 27                   | 0                                      | 1                                       | PREFER_SHA                   |
| 3  | NodeKernel                                                    | {3, 1, 1}       | {32, 1, 1}       | 17,348,802.222  | 11.776        | 16.67 %   | 34                   | 8                                      | 2                                       | PREFER_SHA                   |
| 4  | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 17,552,635.214  | 3.904         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 5  | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 17,756,150.734  | 5.216         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 6  | gemm_kernel1x1_core<float,bool=0,bool=0,bool=0,bool=0,bool=0> | {1, 1, 1}       | {16, 16, 1}      | 17,959,827.214  | 2.816         | 83.33 %   | 22                   | 2176                                   | 0                                       | PREFER_SHA                   |
| 7  | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 18,162,455.438  | 3.840         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 8  | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 18,365,524.942  | 4.160         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 9  | gemm_kernel1x1_core<float,bool=0,bool=0,bool=0,bool=0,bool=0> | {1, 1, 1}       | {16, 16, 1}      | 18,568,770.510  | 2.784         | 83.33 %   | 22                   | 2176                                   | 0                                       | PREFER_SHA                   |
| 10 | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 18,770,637.774  | 3.840         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 11 | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 18,973,297.358  | 4.160         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 12 | gemm_kernel1x1_core<float,bool=0,bool=0,bool=0,bool=0,bool=0> | {1, 1, 1}       | {16, 16, 1}      | 19,176,660.878  | 2.784         | 83.33 %   | 22                   | 2176                                   | 0                                       | PREFER_SHA                   |
| 13 | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 19,379,318.222  | 3.840         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 14 | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 19,581,808.238  | 4.128         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |
| 15 | gemm_kernel1x1_core<float,bool=0,bool=0,bool=0,bool=0,bool=0> | {1, 1, 1}       | {16, 16, 1}      | 19,784,341.646  | 2.784         | 83.33 %   | 22                   | 2176                                   | 0                                       | PREFER_SHA                   |
| 16 | gemv2N_kernel_ref<float,int=128,int=1,int=4,int=4>            | {1, 1, 1}       | {128, 1, 1}      | 19,989,259.406  | 3.872         | 50.00 %   | 38                   | 2560                                   | 0                                       | PREFER_SHA                   |

#### EdgeKernel <<<1,16>>> [CUDA Launch]

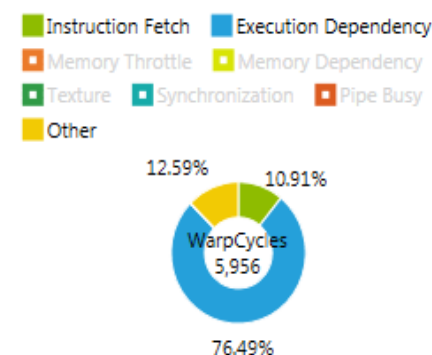
- Device Launches
- Call Graph
- EdgeKernel [CUDA Kernel]
- Experiment Results
  - Occupancy
  - Instruction Statistics
  - Issue Efficiency



#### Warp Issue Efficiency



#### Issue Stall Reasons

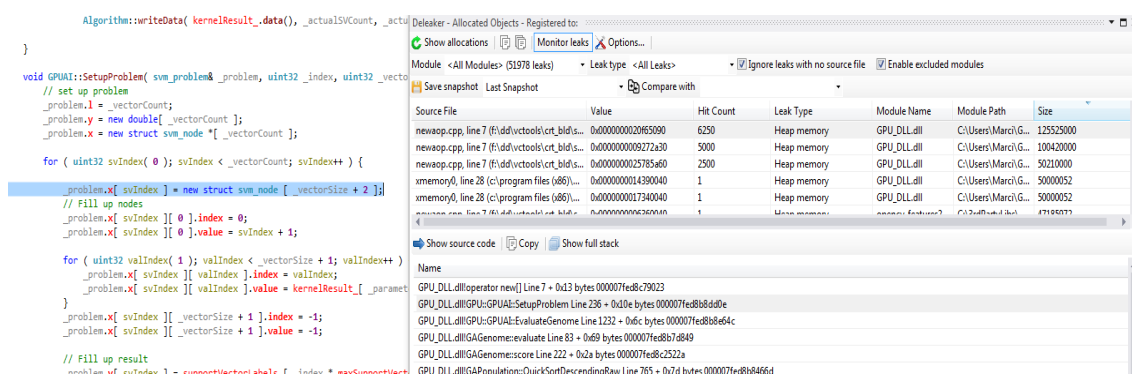


5.9 Ábra: A paraméterhangolásra használt kernel profilja

## 5.1.2 Memóriakezelés

Fontos eleme volt még a keretrendszer memóriakezelésének tesztelése, különösképp a memóriaszivárgások megkeresése. Ennek a feladatnak az elvégzésére egy külön eszközt, a DeLeaker tool-t (5.10 Ábra) használtam, amit a Visual Studioba lehetett beilleszteni. Ez az eszköz képes a memóriefoglalásokat az alkalmazás futása közben monitorozni, majd a futás befejeztével azokat kielemezni, majd a fejlesztőt a detektált szivárgásokat okozó kódrészlethez irányítja. Nagy segítség a call stack megjelenítése a szivárgások esetén.

A DeLeaker tool segítségével sikerült számos kisebb, illetve egy komolyabb szivárgást is felfedeznem. A GPUAI osztály SetupProblem nevű függvényében, amely a kernel függvény eredményiből állít elő egy optimalizálási problémát a LibSVM metódusai számára a létrehozott problémát nem szabadította fel a tanítás után. Emiatt minden egyes tanításkor elszivárgott több megabájtnyi memória, ami főleg a genetikus algoritmus futásakor volt képes hamar megtelíteni a memóriát.



5.10 Ábra: A DeLeaker tool működés közben

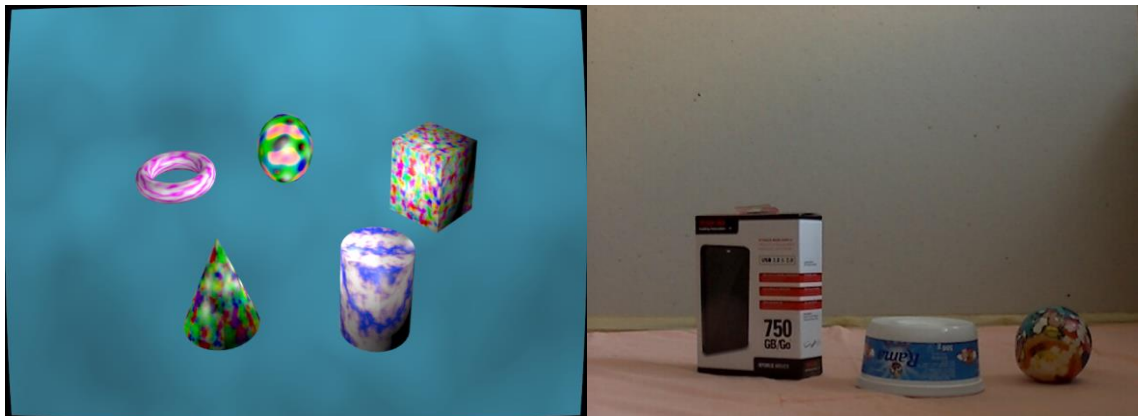
Fontos még megemlíteni egy gyakorlati jellegű problémát, melyet szintén hibás memóriefoglalás okozott. Már korábban említettem, hogy az alkalmazás a grafikus kártyán előre lefoglalja a tanító adatoknak szükséges tárhelyet. Azonban, ha a tanító adatok, vagy a kategóriák száma futás közben változik, akkor ezeket a memóriaterületeket újra kell foglalni a megváltozott méretekkel. Itt azonban véttem egy hibát, vagyis a régi terület felszabadítása után az új területet egy másik, hasonló nevű változó címére foglaltam, ami nem definiált működése miatt véletlenszerűen előforduló hibát eredményezett, és esetenként egy teljesen másik memóriaterületre mutató változó elállítását eredményezte. Ezt a hibát meglehetősen nehéz volt megtalálni, de a korrupt

változóra állított data breakpoint segítségével végül sikerült azonosítanom a hibás kódrészletet.

## 5.2 Algoritmusok

Az alábbi fejezetben a keretrendszer osztályait feltöltő algoritmusok tesztelését, valamint a tesztelés eredményeinek értékelését mutatom be. A felhasznált algoritmusok közül a tanuló alakfelismerő eljárás, valamint a követési módszer tesztelésére fogom kitérni a két alfejezetben.

Előtte fontos azonban említést tenni a teszteléshez használt adatok előállításáról. A tesztelés elvégzéséhez virtuális, valamint valós tesztsorozatok is készítettem. Az előbbi sorozatot egy Blenderben elkészített virtuális jelenetről készítettem. Ebben a jelenetben öt egyszerű formájú objektum volt, melyekről egy, a jelenet körül mozgó virtuális kamerapár készített felvételeket. A kamerák kalibrációját egy virtuális sakktábla segítségével végeztem el. A valós jelenethez egy Axis típusú webkamerák felhasználásával összeszerelt kamerapárt alkalmaztam. A jelenetben szereplő, szintén egyszerű formájú valós objektumokat ezt követően kézzel mozgattam. A tesztsorozatokból származó felvételek a 5.11 Ábrán láthatóak.



5.11 Ábra: A virtuális és a valós tesztsorozatok

### 5.2.1 Alakfelismerés

Az alakfelismerő algoritmus tesztelésére a virtuális tesztsorozatot használtam, mivel ennek segítségével kiküszöbölhettem a valódi kamerák hibáit, így lehetővé téve tisztán az alakfelismerés hatékonyságának a vizsgálatát. A tanító adatokat a korábban bemutatott bekeretező funkció segítségével állítottam elő (a felhasználó által meghatározott

befoglaló téglalapba beleeső csomópontok meghatározásával), így minden kép alapján egy pozitív, illetve egy negatív tanító gráfot adhattam hozzá az egyes kategóriákhoz. Ez összesen 500 tanító adatot jelentett, mivel a virtuális jelenetről 250 felvétel készült. A genetikus algoritmus alapú (a GA paraméterei az 5.1 Táblázatban láthatóak) paraméterhangolás futtatása alapján az 5.2 és az 5.3 Táblázatokban látható eredmények születtek.

**5.1 Táblázat: A genetikus algoritmus paraméterei**

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

**5.2 Táblázat: Az optimális hiperparaméterek**

| Paraméterek | $\vartheta$ | $\gamma$ | $\mu$   | $\delta$ | N  | $\sigma$ | $\varepsilon$ | C    |
|-------------|-------------|----------|---------|----------|----|----------|---------------|------|
| Értékek     | 0,0232      | 0,0097   | 0,00012 | 0,042    | 10 | 1,3      | 3             | 2,23 |

**5.3 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%                |

A szemmel láthatóan kiváló tanítás azonban egy hibás döntés eredménye: Engedtem ugyanis, hogy a genetikus algoritmus kernel végső eredményének skálázásának mértékét a gráfok méretétől függően (8) hangolando paraméterként kezelje, így az algoritmus végeredményképp a döntést szinte kizárólag a gráfok mérete alapján hozta meg, mivel a negatív példaként szolgáló „maradék” gráfok jellemzően nagyobbak voltak a pozitív példáknál. Így a tanítást az  $\varepsilon$  paraméter 1-be kényszerítve újra elvégeztem a hangolást. Az eredmények a 4.4, 4.5 és a 4.6 táblázatokban láthatóak. Fontos változtatás volt, hogy

a negatív példaként az egyes objektumokhoz a másik négy kategória pozitív példáit használtam fel, így a tanítást 1250, nagyjából egyforma méretű gráffal végeztem el. A újbóli tanítás eredményei az 5.4 és az 5.5 Táblázatokban láthatók. A hibát tartalmazó eredmények súlyozva vannak a pozitív és a negatív minták arányának megfelelően.

**5.4 Táblázat: Az optimális hiperparaméterek a javított kategóriák esetén**

| Paraméterek | $\vartheta$ | $\gamma$ | $\mu$   | $\delta$ | N | $\sigma$ | $\varepsilon^3$ | C    |
|-------------|-------------|----------|---------|----------|---|----------|-----------------|------|
| Értékek     | 1,78        | 0,0105   | 0,00514 | 0,117    | 9 | 2,25     | 1               | 0,57 |

**5.5 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%               |

Az eredményeken jelentős overfitting esete látható, melynek több oka is lehet: Ezek közül a legegyszerűbb a szuboptimális paraméterek használata. Ez azonban már csak az optimumkeresés miatt is valószínűtlen, ráadásul a genetikus algoritmus futása közben azt tapasztaltam, hogy az algoritmus eredménye a paraméterek értékére meglehetősen érzéketlen. Egy másik lehetőség, hogy az alakleíró lépésben előállított alakot leíró jellemzők nem elégségesek a kategóriák közti különbségtételre. Ezt okozhatják a jellemzők hiányosságai, vagy akár a jellemzők értékéhez képest jelentős mértékű zaj is.

Az algoritmus lépéseinek tüzetes vizsgálata után ez utóbbi okra sikerült a hibát visszavezetnem. Az alapvető probléma a sztereó rekonstrukció minőségével volt. Ezzel a módszerrel ugyanis előállíthatunk jó minőségű mélységi képeket, az egyes alakzatok finom formájának pontos visszaadására viszont már alkalmatlan az egymáshoz közel lévő

---

<sup>3</sup> A paraméter értéke 1-be volt kényszerítve

kamerákat alkalmazó rekonstrukciós eljárás. Ebből következően a RANSAC alapú szegmentálás és az alakleírás eredménye meglehetősen véletlenszerű lett.

A konkrét tanuló algoritmuson kívül teszteltem a lokalizációs lépés egészét. Itt a lokalizáció végeredményeként adódó objektumok pozícióját és kategóriáját hasonlítottam össze a tanítás során megadott befoglaló téglalapokkal. Bár a tanuló algoritmus túlillesztési hibája a lokalizáció eredményét lényegesen ronthatja, bíztam abban, hogy azonos képsorozaton ez a hiba (5.6 Táblázat) mérsékelt marad. A lokalizáció eredménye alapján látható, hogy a tanuló algoritmus hibája ellenére a lokalizáció elég megbízható maradt.

**5.6 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%             |

Fontos rész még az algoritmusok sebességének tárgyalása, amelyre nincs szigorú követelmény, de a lokalizáció esetén a felhasználói élmény szempontjából célszerű lenne ennek a néhány másodperces tartományban való tartása. A tanítás és a lokalizáció egy Negyedik generációs Intel i5 4670K négymagos processzor, valamint egy Nvidia GeForce Gtx 650 ti típusú grafikus kártyán mért futási ideje az 5.7 és 5.8 táblázatban láthatóak. Az eredményekből levonható az a következtetés, hogy a futási időt érintő követelmény tartásával nincs különösebb probléma.

**4.6 Táblázat: Az alakfelismeréshez kapcsolódó algoritmusok futási ideje**

|                        |                |
|------------------------|----------------|
| Tanítás (500x500 adat) | 23 154,836 ms  |
| Kereszt-validáció      | 157 688,250 ms |
| Osztályozás            | 14,237 ms      |

**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 |

Fontos kérdés volt még a skálázhatóság tárgyalása, vagyis a futási idők függése az adatok számától. Könnyen belátható, hogy a tanítás, valamint az alakfelismerés futási ideje a kategóriák számában lineáris. Fontos megjegyezni, hogy a különböző kategóriák száma az alakfelismerő algoritmus megkülönböztető képességét is befolyásolja. A kategóriák számának növelésével a köztük való döntés egyre nehezedik, és a hatékonyság csak újabb jellemzők hozzáadásával javítható. A tanítás sebességét a tanító adatok száma is befolyásolja, méghozzá négyzetes módon, míg a lokalizáció sebességét ez nem érinti közvetlenül, mivel az inkább a support vektorok számától függ lineárisan.

Fontos még az algoritmusok függése a kategóriák bonyolultságától (vagyis a csomópontok számától). A kernel számítása a csomópontok számától kubikus módon függ, ahogy azt már a korábbi fejezetekben megemlítettem. Ez az összefüggés azonban a tanítás futási idejében azonban nem fog jól látszani, mivel a csomópontok megnövekedett száma először a grafikus kártya kihasználtságát fogja növelni. A lokalizáció esetén hasonló helyzet áll elő kiegészülve azzal, hogy a különböző átlagos bonyolultságú gráfokból összeállított tanulási probléma más modellt fog eredményezni, különböző számú support vektorral, ami véglegesen elrejt a komplexitástól való függést. A mért eredmények az 5.9, az 5.10, valamint az 5.11 Táblázatokban láthatóak.

**5.9 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 |

**5.10 Táblázat: A lokalizáció futási ideje az objektumok komplexitása függvényében**

| Csomópontok | 4-5 csomópont | 10-12 csomópont | 25-30 csomópont |
|-------------|---------------|-----------------|-----------------|
| Lokalizáció | 3 024,12 ms   | 3 873,59 ms     | 6 118,56 ms     |

**5.11 Táblázat: Az algoritmusok futási ideje a kategóriák komplexitásának függvényében**

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

## 5.2.2 Követés

Az alábbi fejezet témája a követési algoritmus tesztelése és az eredmények értékelése. A követési algoritmus tesztelését a valós, valamint a virtuális képsorozat segítségével is elvégeztem. Fontos volt azonban a követés minőségét leíró mérőszámok megalkotása. Probléma ugyanis, hogy a követendő objektumoknak nem ismerjük a pontos pozícióját sem a képsíkon, sem a háromdimenziós térben. Ezért jobb híján a tanítás során megadott befoglaló téglalapok középpontjait használtam becslésként, és összeszámoltam azon képkockák számát, ahol a követett objektum középpontja a képsíkra visszavetítve a téglalap középpontjához egy határnál közelebb volt. A határ megválasztásánál viszont meglehetősen engedékenynek kellett lennem, a helyes eredményt jelző referenciapont bizonytalansága miatt.

A követő algoritmus eredményein (5.12 Táblázat) látszik, hogy a követés meglehetősen pontos, átlagosan több, mint 30 képkocka szükséges ahhoz, hogy az akkumulálódó hibák a követés eredményét elfogadhatatlanul pontatlanná tegyék. Mivel természetesen a követést szeretném minél pontosabban elvégezni, ezért a detektálást ennél gyakrabban végzem el.

**5.12 Táblázat: A követés eredménye**

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

Kiemelten fontos szempont az algoritmus futási idejének mérése, amely részfeladatokra lebontva az 5.13 Táblázatban látható. A követés futásával szemben

szigorú valósidejű követelményeink vannak, így a több, mint 600 ms futási idő elfogadhatatlanul nagy. A részeredményekből azonban látható, hogy ezt nagyrészt az AKAZE detektor futási ideje teszi ki. Ez a futási idő javítható az AKAZE gyorsabb alternatívára való cseréjével (pl.: ORB), valamint az AKAZE grafikus kártyán történő implementációjával. További csökkenés érhető el kisebb bementi kép használatával, ami viszont csökkentheti a detektált képjellemzők számát.

**5.13 Táblázat: A követő algoritmus futási ideje**

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

Az utolsó tárgyalandó szempont a követés tesztelésével kapcsolatban a skálázhatóság kérdése. A követés alapvetően egyetlen paramétertől függ, ez pedig a követendő objektumok száma. Az egyes objektumokra lefutott RANSAC alapú transzformáció becslő lépést ugyanis minden tárgyra külön kell futtatni, ami az objektumok száma szerinti lineáris függést jelentene. Azonban az objektumok frissítése a követés futási idejének csupán kis részét teszi ki, így nem számítottam jelentős lassulásra az objektum számának függvényében, amit a mért eredmények (5.14 Táblázat) vissza is igazoltak.

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

| Objektumok | 1-2 objektum | 4-5 objektum | 10-12 objektum |
|------------|--------------|--------------|----------------|
| Követés    | 632,33 ms    | 652,30 ms    | 660,23 ms      |

## 6 Összefoglalás

A diplomamunkám során létrehoztam és bemutattam egy adaptív kiterjesztett valóság rendszer prototípusát, mely képes egy előre ismeretlen munkateret intelligens módon berendezni virtuális objektumokkal, valamint a munkatérben található tárgyak követésével a felhasználó és a virtuális környezet interakcióját megvalósítani. A munkám részeként megterveztem, létrehoztam és teszteltem egy keretrendszert, amely hozzáférést biztosít a rendszer főbb funkcióihoz, valamint tesztelési, illetve hangolási lehetőségeket is biztosít. Ezen felül a keretrendszer vezérli a főbb feladatokat elvégző algoritmusok működését és kommunikációját. A munkám fontos részét képezte az előbb említett algoritmusok implementációja és tesztelése is, különös tekintettel egy tanuló alakfelismerő, valamint egy hat szabadságfokú követő algoritmusra.

A jelenlegi munkámat szeretném a jövőben a PhD képzés keretei között folytatni, ezért különösen fontos a továbbfejlesztési és további kutatási terveimet is bemutatnom. A legfontosabb továbbfejlesztési lépés a sztereó rekonstrukció lecserélése lesz, mivel ennek a módszernek a pontatlansága komoly problémákat okozott. Mivel a sztereó feltérképezés és rekonstrukció választása egy egyszerűsítő lépés volt ezért ez a lépés a rendszer természetes evolúciójának is része. Az új rekonstrukciós algoritmusként egy egykamerás motion stereo eljárást képzeletem el, mivel a végleges változatban a rendszer kijelző szemüveg segítségével fog működni, így célszerű az ilyen szemüvegeken általában megtalálható kamerát használni. Fontos megjegyezni, hogy ekkor a követő algoritmust is le kell cserélni, mivel nem állnak majd rendelkezésre sztereó képjellemző párok.

További fontos terv a kijelzés végleges megoldása a rajzolt primitív formák helyett a tervező által megadott modellek textúrák használatával. Ezen felül fontos még lehetővé tenni a keretrendszeren belül különálló virtuális környezet sémák létrehozását, amelyek az egyes elvégzendő feladatoknak felelnek meg. Így lehetővé válik többféle előre programozott felhasználási mód létrehozása, amelyek a rendelkezésre álló virtuális objektumkategóriák egy részét használnák csak fel.

További kutatási lehetőségek közül lényeges más tanuló algoritmusok kipróbálása az SVM helyett, és ezek eredményeinek összevetése a jelenlegivel. Fontos irányzatként

képzelem el az alakleírási, illetve magának a berendezési problémának megfogalmazását tanulási/optimalizálási probléma formájában. Ennek a problémának a kezelésére is többféle tanuló algoritmus kipróbálását tervezem, különös tekintettel a mély tanulás (deep learning) algoritmusokra, mint például az autoencoder, vagy az RBM (Restricted Boltzmann Machine). Lehetséges irányzat továbbá többszintű tanuló rendszerek létrehozása, ami a feladat jellegéből adódik.

A tanuló algoritmus milyenségén felül fontos irányzat a döntés során figyelembe vett jellemzők körének bővítése. A jelenlegi állapotban a rendszer a formát leíró jellemzők alapján működik, melyek bővítése további feature-ökkel lehetséges. Ezen felül lehetőség nyílik további szempontok, például kontextus, vagy vizuális jellemzők hozzáadására is.

Korábban említettem, hogy fontos lehetőség a berendező lépés megfogalmazása optimalizálási problémaként, így lehetővé téve optimum kereső, vagy tanuló algoritmusok használatát a megoldáshoz. Ennél a lépésnél fontos az optimalizálási kritérium tárgyalása, ami a jelenlegi rendszerben az alakbeli hasonlóság (a tanuló alakfelismerés eredménye). Lehetséges azonban újabb kritériumokat bevezetni, melyek a berendezés minőségét írják le. Ezek közül az egyik a visszaillesztési pontosság: Ekkor a berendezés végeredménye szerint visszaillesztjük a behelyezendő virtuális objektum 3D modelljét a pontfelhőbe, és egy illeszkedési pontosságot számolunk. A másik fontos lehetőség az alkalmazás igényeinek figyelembevétele. Egyes virtuális objektumok ugyanis az adott feladat szempontjából sokkal fontosabbak lehetnek, míg más objektumok esetleg csupán dekoratív jellegűek. Ebben az esetben az előbbi kategóriát előnyben kell részesíteni a berendezés során.

## **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.

## Irodalomjegyzék

- [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] P. Rojtberg és A. Olwal, „Tangible Interfaces using Handheld Augmented Reality,” in *SIGRAD*, Västerås, 2010.
- [3] V. Buchmann, S. Violich, M. Billinghurst és 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.
- [4] H. T. Regenbrecht, G. Baratoff, I. Poupyrev és M. Billinghurst, „A Cable-less Interaction Device for AR and VR Environments,” in *Proceedings of ISMR*, 2001.
- [5] K. Matkovic, T. Psik, I. Wagner és D. Gracanin, „Dynamic Texturing of Real Objects in an Augmented Reality System,” in *VR 2005. IEEE Virtual Reality*, Bonn, 2005.
- [6] D. Bandyopadhyay, R. Raskar és H. Fuchs, „Dynamic Shader Lamps : Painting on Movable Objects,” in *IEEE and ACM International Symposium on Augmented Reality*, New York, NY USA, 2001.
- [7] B. R. Jones, H. Benko, E. Ofek és A. D. Wilson, „IllumiRoom: Peripheral Projected Illusions for Interactive Experiences,” Microsoft Research, Redmond, WA USA, 2013.
- [8] F. Thou, H. B.-L. Duh és 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] G. A. Lee, M. Billinghurst és 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.

- [10] M. Billinghurst, H. Kato és S. Myojin, „Advanced Interaction Techniques for Augmented Reality Applications,” in *Lecture Notes in Computer Science*, Berlin, Springer, 2009, pp. 13-22.
- [11] M. Billinghurst, H. Kato és 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 és 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] H. Hirschmüller, „Stereo Processing by Semi-Global Matching and Mutual Information,” *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, %1. kötet30, %1. szám2, pp. 328 - 341, 2007.
- [14] M. Hatzitheodorou, E. A. Karabassi, G. Papaioannou, A. Boehm és T. Thouaris, „Stereo matching using optic flow,” *Real-Time Imaging*, %1. kötet6, %1. szám4, pp. 251 - 266, 2000.
- [15] M. Zuchelli, *Optical Flow Based Structure from Motion*, Stockholm: Royal Institute of Technology, 2002.
- [16] Q. Yang, L. Wang és 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.
- [17] H. Sadeghi, P. Moallem és S. A. Monadjemi, „Feature Based Dense Stereo Matching using Dynamic Programming and Color,” *Interational Journal of Infomation and Mathematical Science*, %1. kötet4, %1. szám3, pp. 179-186, 2008.

- [18] T. Kounalakis és G. A. Triantafyllidis, „3D scene's object detection and recognition using depth layers and SIFT-based machine learning,” *3D Research*, %1. kötet2, %1. szám6, pp. 1-11, 2011.
- [19] R. Osada, T. Funkhouser, B. Chazelle és Dobkin David, „Matching 3D Models with Shape Distributions,” in *SMI 2001 International Conference on Shape Modeling and Applications*, Genova, 2001.
- [20] A. Golonivskiy, V. G. Kim és T. Funkhouser, „Shape-based Recognition of 3D Point Clouds in Urban Environments,” in *IEEE 12th International Conference on Computer Vision*, Kyoto, Japan, 2009.
- [21] F. Tombari és 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.
- [22] M. A. Fishler és R. C. Bolles, „Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Magazine Communications of the ACM*, %1. kötet24, %1. szám6, pp. 381-395, 1981.
- [23] R. Schnabel, R. Wahl és R. Klein, „Efficient RANSAC for Point-Cloud Shape Detection,” *Computer Graphics Forum*, %1. kötet26, %1. szám2, pp. 214-226, 2007.
- [24] R. Schnabel, R. Wahl, R. Wessel és 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.
- [25] M. Ferrer, F. Serratosa és A. Sanfeliu, „Synthesis of median spectral graph,” in *Second Iberian Conference, IbPRIA 2005*, Estoril, 2005.
- [26] D. H. White, *Generative Models for Graphs*, Department of Computer Science, The University of York, 2009.
- [27] P. Zhu és R. C. Wilson, „Stability of the Eigenvalues of Graphs,” in *11th International Conference, CAIP 2005*, Versailles, 2005.

- [28] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor és K. M. Borgwardt, „Graph Kernels,” *Journal of Machine Learning*, %1. kötet11, pp. 1201-1242, 2010.
- [29] Y. Park, V. Lepetit és W. Woo, „Multiple 3D Object Tracking for Augmented Reality,” in *7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Cambridge, 2008.
- [30] D. G. Lowe, „Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, %1. kötet60, %1. szám2, pp. 91-110, 2004.
- [31] H. Bay, T. Tuytelaars és L. Van Gool, „SURF: Speeded Up Robust Features,” *Computer Vision and Image Understanding*, %1. kötet3, %1. szám110, p. 346–359, 2008.
- [32] E. Rublee, V. Rabaud, K. Konolige és G. Bradski, „ORB: an efficient alternative to SIFT or SURF,” in *2011 IEEE International Conference on Computer Vision*, Barcelona, 2011.
- [33] M. Calonder, V. Lepetit, C. Strecha és P. Fua, „BRIEF: Binary Robust Independent Elementary Features,” in *11th European Conference on Computer Vision*, Heraklion, 2010.
- [34] P. F. Alcantarilla, A. Bartolli és A. J. Davison, „KAZE Features,” in *12th European Conference on Computer Vision, Part VI*, Florence, Italy, 2012.
- [35] P. F. Alcantarilla, J. Nuevo és A. Bartoli, „Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces,” in *In Proceedings British Machine Vision Conference*, Bristol, UK, 2013.
- [36] „Qt Project,” Open Source, [Online]. Available: <http://qt-project.org/>.
- [37] „OpenCV,” Open Source, Intel, [Online]. Available: <http://opencv.org/>.
- [38] „PCL - Point Cloud Library,” Open Source, [Online]. Available: <http://pointclouds.org/>.

- [39] R. Schnabel, R. Wahl és R. Klein, „RANSAC Implementation,” 2007. [Online]. Available: <http://cg.cs.uni-bonn.de/en/publications/paper-details/schnabel-2007-efficient/>.
- [40] „Eigen,” Open Source, [Online]. Available: [http://eigen.tuxfamily.org/index.php?title=Main\\_Page](http://eigen.tuxfamily.org/index.php?title=Main_Page).
- [41] C.-C. Chang és C.-J. Lin, „LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, %1. kötet2, %1. szám3, pp. 27:1-27:27, 2011.
- [42] M. Wall, „GAlib A C++ Library of Genetic Algorithm Components,” MIT, 1999. [Online]. Available: <http://lancet.mit.edu/ga/>.
- [43] R. Laganière, „OpenCV Cookbook Source Code - Robust Matcher,” 2011. [Online]. Available: <https://code.google.com/p/opencv-cookbook/source/browse/trunk/Chapter%2009/matcher.h>.
- [44] F. L. Markley, "Attitude Determination using Vector Observations and the Singular Value Decomposition," *Journal of the Astronautical Sciences*, vol. 38, no. 3, pp. 245-258, 1988.