# 3D pontfelhő regisztrációs folyamatok és jellegzetességleírók vizsgálata

Doktori értekezés

Varga Dániel
Témavezető: Dr. Laki Sándor
DOI: 10.15476/ELTE.2022.228


Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

Informatikai Doktori Iskola
Iskolavezető: Prof. Csuhaj-Varjú Erzsébet
Doktori Program: Információs Rendszerek
Programvezető: Prof. Benczúr András

Budapest, 2022

# Köszönetnyilvánítás

Szeretnék köszönetet mondani Laki Sándor témavezetőmnek, aki segítséget nyújtott a cikkek elkészülésében, valamint minden lehetséges módon támogatta a kutatásokat.

Köszönetet mondok az Információs Rendszerek tanszék minden munkatársának, akik hasznos tanácsokkal segítették a mindennapjaimat és segítenek a mai napig. Külön köszönöm Vörös Péter, Gombos Gergő, Szalai-Gindl János Márk, Fejes Ferenc és Rudas Ákos kollégáimnak, Frankó Gáborné Mariannak és Kiss Attilának a tanszék vezetőjének.

Köszönettel tartozom szüleimnek, testvéreimnek és barátnőmnek, Deák Júliának hogy segítettek a tanulmányaim során.

*„...mert minden maga útján járó ember kötelességének tartom, hogy tudtára adja a közösségnek, mit talált felfedező útján: hűvös vizet-e a szomjazónak, vagy a terméketlen tévedés homoksivatagját. Az első megsegít, a második intésül szolgál.„*

- Carl Gustav Jung, Bevezetés a tudattalan pszichológiájába

# Tartalomjegyzék

# Ábrák jegyzéke

# Táblázatok jegyzéke

# Algoritmusok és példák jegyzéke

# Rövidítések jegyzéke

ABKR - adatbázis-kezelő rendszer
CatMat - Catenarian Matching (láncolat alapú objektumkeresés)
FPFH - Fast Point Feature Histogram
GiST - generalized indext structure (általánosított keresőfa)
ICP - Iterative Closest Point
ISS - Intrinsic Shape Signature
LIDAR - Light Detection and Ranging (lézer alapú távérzékelés)
LRF - Locale Reference Frame (lokális vonatkoztatási rendszer)
NMS - Non-Maximum Supression
NNDR - Nearest Neighbor Distance Ratio
PCA - Principal Component Analysis (főkomponens analízis)
PCL - Point Cloud Library
PFH - Point Feature Histogram
PRC - Precision-Recall Curve
QBB - Quantile Based Binarization
RDP - Ramer–Douglas–Peucker
ROC görbe - Receiver Operating Characteristic Curve
SHOT - Signature of Histograms of OrienTations
SI - Spin Images jellegzetességleíró
SSI - Smooth Shrink Index
T-LESS - adathalmaz
ToF - Time-of-Flight
TYO-L - Toyota Light adathalmaz

# 1. fejezet

# Bevezetés

A számítógépes látás célja, hogy a számítógép képes legyen megérteni a képeket, videókat és egyéb felvételeket, automatizálva elvégezni azokat a feladatokat, amit az emberek a látásuk segítségével. Az elmúlt évtizedekben a terület rohamosan fejlődött. A fejlődést az motiválta, hogy a kutatók megoldást találjanak gyakorlati és hasznos problémák elvégzésére: emberi arcok felismerésére [20], kézzel írt számok felismerésére a postai levelek automatikus osztályozásához [21] stb. A számítógépes látással szorosan együtt fejlődtek a gépi tanulást használó módszerek is. Az utóbbi években a számítógépes látás az informatika egyik legfontosabb területévé nőtte ki magát, ahol rendszeresen jelennek meg korábban megvalósíthatatlannak hitt megoldások. Az algoritmusok meg tudják mondani, hogy az egyes képeken milyen objektumok láthatóak, az emberi arcok felismerése már magas szintű biztonságos azonosítást is lehetővé tesz.

A területen egyre nagyobb figyelem fordult a háromdimenziós adatok feldolgozására, amiben két fontos tényező játszott közre. Először is megjelentek és egyre olcsóbbá váltak a mélységadatot felvenni képes szenzorok, a számítógépes számítási kapacitások növekedésével pedig lehetővé vált a háromdimenziós adatokat feldolgozó algoritmusok akár valós időben történő futtatása. Másodszor pedig megjelentek olyan alkalmazási területek, ahol fontossá vált a mélységadatok megértése is.

Míg másfél évtizeddel ezelőtt a háromdimenziós adatok feldolgozásának a fő motivációja az autonóm robotok működésének elősegítése volt, ma már más célok motiválják a kutatókat, ipari szereplőket. Az egyik ilyen téma a kiterjesztett valóság, amely meghódította a felhasználók mobileszközeit. A kiterjesztett valóság alkalmazások alapja, hogy az eszköz ismerje a környezetét, így képes legyen virtuális objektumokat elhelyezni ebbe a környezetbe. A világon végig söpörtek a kiterjesztett valóság élményt nyújtó alkalmazások, pedig a technológia még közel sem tökéletes. A kiterjesztett valóság mellett a virtuális valóságban és a való életben is fontos szerepet játszanak a háromdimenziós

adatok. A felvételekkel a virtuális térben rekonstruálhatunk valós helyszíneket és objektumokat, így megőrizve az utókornak és eljuttatva az élményt a világ távoli részeire. Különböző műemlékek és épületek szkennelése segíthet a restaurálásban.

Egy másik népszerű terület, amely szoros kapcsolatban áll a mélység adatok érzékelésével az autonóm járművek világa. Bár egyes elképzelések szerint az autonóm járművek képesek pusztán kétdimenziós felvételek alapján megfelelően működni, a legtöbb megközelítés 3D adatokra is hagyatkozik. Az olyan rendszerek, amelyek csak kétdimenziós képekre támaszkodnak bizonyítottan átverhetőek [113]. Természetesen ezeket a hibákat lehet orvosolni és a 3D adatokat használó rendszerekben is találhatunk ilyen hibákat.

Az értekezésemben háromdimenziós adatokkal dolgozok, főleg különböző szkennerek által begyűjtött 3D pontfelhőkkel. A 3D pontfelhők feldolgozása költségesebb, mint a kétdimenziós képeké, viszont a korábban említett felhasználási területek fejlődése miatt egyre gyorsabb és jobb megoldásokra van szükség ezen a területen. A 3D pontfelhők egymásra illesztése olyan feladat, amely számos alkalmazásban komoly szerepet kap: objektumok felismerésénél az ismert objektumokat illesztjük az objektumokat tartalmazó adatokra, objektumok szkennelésekor több nézőpontból felvett oldalakat kell egymásra illeszteni, hogy azok egy összefüggő objektumot alkossanak, épületek felvételekor a különböző felvételeket pontosan kell illeszteni, hogy az elkészült reprezentáció megfeleljen a valóságnak.

A 3D pontfelhők illesztése, vagy regisztrációja már évtizedek óta ismert probléma. Ugyanúgy, mint minden más területen, itt is megjelentek megközelítések, amiket később felváltottak újabb, korszerűbb irányok. Történelmileg, az első munka, amely 3D adatok regisztrációjával foglalkozott 1986-ban született [18]. Azóta sok idő eltelt, és a ma is jól ismert és széles körben használt (továbbfejlesztésekkel [89]) Iterative Closest Point algoritmus [23] is már 30 éve jelent meg, 1992-ben.

Az általam mélységeiben feldolgozott téma a jellegzetességleíró alapú regisztrációs folyamatok. Ez a megközelítés az elmúlt másfél évtizedben nyert teret és ért el kiváló eredményeket. Mind a mai napig jelennek meg új jellegzetességleírók, amelyek egyre jobb eredményeket érnek el. A jellegzetességleíró alapú regisztrációs folyamatok lépéseit részletesen bemutatom a későbbiekben. A munkám során a folyamat minden lépésével foglalkoztam így teljes egészében bemutatom a 3D pontfelhők regisztrációjának ezt a fajta megközelítését.

## 1.1. Főbb hozzájárulások

Az értékezésemben a jellegzetességleíró alapú regisztrációs folyamatok egyes lépéseit vizsgáltam. A főbb hozzájárulásaimat és a hozzájuk kapcsolódó munkákat lentebb összefoglalom.

### 1.1.1. Jellegzetességleírók dimenziócsökkentésnek a vizsgálata [5]

A jellegzetességvektorok általában sokdimenziós, valós értékű vektorok. A leggyakrabban végzett művelet a jellegzetességvektorokon a legközelebbi szomszéd keresések. Ezeket különböző indexstruktúrák segítségével lehet gyorsan és hatékonyan elvégezni (pl. $Kd$-fa). Azonban egyes jellegzetességvektorok dimenziószáma elérheti a több százat is (pl. a SHOT leíró 352 dimenziós). Ilyen magas dimenziószám esetén az indexstruktúrák már nem tudnak hatékonyan működni, egyes esetekben a szekvenciális szkennelés jobb teljesítményre képes.

A munkám célja az volt, hogy megvizsgáljam, hogyan változnak a jellegzetességleírók leíróképességei a dimenziócsökkentés hatására. A dimenziócsökkentést kétféle módon végeztem el: főkomponens analízis használatával és a jellegzetességleírók belső paramétereinek a módosításával. A vizsgálat során kiderült, hogy egy bizonyos dimenziószámig a belső paraméterek módosítása előnyösebb, azonban alacsony dimenziószámnál érdemesebb a főkomponens analízist használni. Az is kiderült továbbá, hogy mekkora dimenziószámnál esik nagyot a leíróképesség, és meddig érdemes még elmenni a csökkentéssel. A vizsgált jellegzetességleírók összehasonlítása alapján arról is levonható következtetés, hogy melyik jellegzetességleíró teljesít a legjobban. Az így nyert tapasztalat összhangban áll más munkák hasonló összevetéseinek az eredményeivel [66].

### 1.1.2. Jellegzetességleírók binarizációja [7]

A jellegzetességleírók legközelebbi szomszéd kereséseinek a gyorsítására a dimenziócsökkentésen kívül egy másik népszerű módja a valós értékű jellegzetességvektorok binarizációja. A binarizált jellegzetességvektorok összehasonlítása gyorsan elvégezhető bitvektorok közötti Hamming távolság kiszámításával, szemben a valós értékű vektorok közötti különböző távolságmetrikákkal (Mahnattan vagy euklideszi távolság).

A hozzájárulásom a területhez egy új jellegzetességleíró binarizációs eljárás, amely tetszőleges valós értékű jellegzetességvektort képes binarizálni. A javasolt módszert több ismert jellegzetességleíró módszerre alkalmaztam, és összehasonlítottam más korszerű binarizációs algoritmusokkal. A kiértékelésem alapján az általam javasolt módszerrel készí-

tett bináris leírók leíróképessége meghaladja a más binarizációs módszerekkel binarizált leírókét. Továbbá, sokkal kevesebb tárigénnyel jobb teljesítményt nyújt az önálló bináris jellegzetességleíróktól is. Bemutatom a javasolt módszer különböző változatait is, amelyek nagyban javítják a leíróképességet, bár a legközelebbi szomszéd keresések lassulása mellett. A bemutatott változatok közötti választással eldönthető, hogy az alkalmazások során a leíróképesség vagy a futási idő a fontosabb.

### 1.1.3. Adatbázis-kezelő rendszert használó sablonillesztési folyamat [4]

A sablonillesztés egy olyan objektumfelismerési feladat, amely esetében egy objektumnak több előfordulása is lehet a helyszínen. A munkámban meghatároztam a sablonillesztési folyamatnak egy speciális feladatát, amikor feltételezzük, hogy a helyszínfelhőt előre ismerjük és offline feldolgozható, a lekérdezésfelhő pedig online érkezik. Egy olyan folyamatot javaslok, amely adatbázisban tárolja a helyszínfelhőt, előre kiszámolja a jellegzetességleíróit és indexstruktúrát hoz létre a legközelebbi szomszéd keresések elvégzéséhez.

A jellegzetességleírók közötti távolságok által meghatározott megfeleltetések pontjain sűrűségalapú klaszterezést végzünk, így kiszűrve a lehetséges hibás megfeleltetéseket. A klasztereket előfordulásnak tekintve új megfeleltetéseket keresünk, a feladatot minimális súlyú párosításként értelmezve. Az így kapott új megfeleltetések alapján meghatározott durva transzformációk további finomításon mennek keresztül.

A javasolt folyamat sikeresen megoldja a sablonillesztési feladatot. A folyamathoz PostgreSQL adatbázist használó implementációt készítettünk, és ezen végeztük el a kiértékelést. A folyamat bemutatása során rámutatok, hogy milyen lépésekben tud előnyös lenni az adatbázis-kezelő rendszer használata.

### 1.1.4. Láncolatalapú objektumfelismerő algoritmus [6]

A mélységadatokat használó objektumfelismerési feladat lényege, hogy előre ismerünk bizonyos objektumokat (rendelkezünk róluk valamilyen reprezentációval, pl. 3D pontfelhő) és az online érkező, tipikusan 2.5D pontfelhőben felismerjük az ott található objektumokat. A feladat motivációja az autonóm robotok érzékelésén alapszik, ugyanis egy robotnak érzékelnie kell a körülötte lévő világot és rendelkeznie kell magas szintű tudással arról, hogy az általa érzékelt világ mit tartalmaz. A legáltalánosabb feladat az autonóm robot előtt lévő objektumok mozgatása. Az objektumok mozgatásához szükséges ismerni az objektumok helyzetét és orientációját, azt, hogy milyen messze van, kitakarja-e más

objektum stb.

Az értekezésemben bemutatok egy új objektumfelismerő algoritmust, amely pontláncolatokat határoz meg és azok megfelelőit keresi a helyszínen. A javasolt módszerben éldetektáló algoritmus felhasználásával meghatározom a felületek érdekes részeit és gráfot építek az objektumot reprezentáló pontokon. A gráfok éleit súlyozom, hogy a későbbi útvonalkeresés közben a gráf érdekes részei kerüljenek be a láncolatokba. Az így kapott gráfon kijelölök kezdő és végpontokat és minimális súlyú útvonalat keresek közöttük, amelyeket köztes pontok elhagyásával egyszerűsítek.

Egy helyszínfelhő beérkezésekor a feladat az objektumokon meghatározott láncolatokhoz hasonló láncolatokat keresni a helyszínfelhőn. Ehhez jellegzetességvektorok közötti legközelebbi szomszéd keresések történnek és geometriai megszorítás kerülnek ellenőrzésre. A hasonló láncolatok megkeresése után megkapjuk az objektumokhoz tartozó transzformációkat.

A módszer összehasonlításra került más objektumfelismerő algoritmussal és a kiértékelések alapján a módszerünk robusztusabb, bár pontosságban elmarad. A javasolt módszer képes a sablonillesztés feladatának megoldására is.

## 1.2. A dolgozat felépítése

Az értekezésem 3. fejezetében bemutatom a jellegzetességleírók dimenziócsökkentésének a vizsgálatát. A rákövetkező fejezetben (4. fejezet) rátérek a jellegzetességleírók binarizációjára, mivel a két munka célja hasonló: a legközelebbi szomszéd keresések felgyorsítása. Ezután, az 5. fejezetben bemutatom a javasolt adatbázis-kezelő rendszert használó sablonillesztési folyamatot. Végül pedig, a 6. fejezetben bemutatom az objektumfelismerési feladatot megoldó láncolatalapú objektumfelismerő algoritmust. Az értekezésemet egy összegzéssel zárom.

# 2. fejezet

# Elméleti áttekintés

Ebben a fejezetben szeretném megismertetni az olvasót a doktori értekezésem megértéséhez elengedhetetlenül szükséges fogalmakkal. A bevezetőben említett fogalmak nagy része ismerős lehet az olvasó számára, azonban az egyes témák tárgyalásakor szerepelnek olyan jelölések és meghatározások, amelyek bár a téma ismerőinek alapfogalom, a köznyelvben más jelentéssel is előfordulhatnak, így nem várható el ezek pontos ismerete. Csak olyan fogalmak fognak itt szerepelni, amelyek több témakörhöz kapcsolódóan is szerepelnek az értekezésben, ezért az egyes fejezetek előtti bemutatásuk redundáns lenne. Kezdjük az áttekintést a legelején, a bevezetőben is többször említett pontok és pontfelhők bevezetésével. Az áttekintés megírásához felhasználtam Radu Bogdan Rusu Semantic 3D Object Maps for Everyday Robot Manipulation [58] című disszertációjának a jelöléseit és megállapításait.

## 2.1. 3D pontfelhők

A pontfelhők pontok halmaza a térben. Amikor pontfelhőkről beszélünk, akkor általában háromdimenziós pontokról beszélünk az euklideszi térben. Legyen $\mathbf{p} \in \mathbb{R}^3$ egy pont az euklideszi térben, amely a Descartes-féle koordináta-rendszerben a következőképpen írható fel:

$$\mathbf{p} = [x, y, z]^T,$$

ahol $x, y, z \in \mathbb{R}$ a pont koordinátái, amelyek a pont távolságát jelölik a koordináta-rendszer origójától. Egy 3D pontfelhőt (vagy egyszerűen pontfelhőt), a következő módon definiálunk:

$$\mathcal{P} = \{\mathbf{p_1}, \mathbf{p_1}, ..., \mathbf{p_n}\},$$

ahol $\mathbf{p_i} \in \mathbb{R}^3, i \in [1..n]$ háromdimenziós pontok, $n$ pedig a pontfelhőben lévő pontok száma, azaz a pontfelhő számossága. A pontfelhőkkel való munka során az egyik leggyakrabban használt művelet egy pont szomszédságának meghatározása. Mivel egy pont szomszédsága a későbbiekben sokszor fog szerepelni, ezért célszerű most bevezetnünk. Legyen $\mathcal{P}$ egy pontfelhő, $\mathbf{p_q} \in \mathcal{P}$ pedig egy pont, amelynek meg szeretnénk határozni a szomszédságát (lekérdezési pont). Ekkor a $\mathbf{p_q}$ lekérdezési pont szomszédsága ($\mathcal{N}_r$) a következő:

$$\mathcal{N}_r = \{\mathbf{p} \mid d(\mathbf{p}, \mathbf{p_q}) \leq r\} \quad (\mathbf{p} \in \mathcal{P}),$$

ahol $d(\cdot)$ két pont távolságát megadó függvény (3D pontfelhőben lévő pontok esetén az euklideszi-távolság: $||\cdot||_2$), és az $r \in \mathbb{R}$ a sugár, amely megadja, hogy milyen távol lévő pontok számítanak bele a szomszédságba. Hasonló módon megadható egy $\mathbf{p_q}$ lekérdezési pontra a pont $k$-szomszédsága, amit $\mathcal{N}_k$-val jelölünk és igazak rá a következő feltételek:

$$|\mathcal{N}_k| \leq k$$

$$\forall \mathbf{p} \in \mathcal{N}_k,\ \forall \mathbf{t} \in \mathcal{P} - \mathcal{N}_k :\ d(\mathbf{p_q}, \mathbf{p}) \leq d(\mathbf{p_q}, \mathbf{t}),$$

ahol $|\mathcal{N}_k|$ a $\mathcal{N}_k$ számossága, azaz halmazba eső pontok száma.

### 2.1.1. Pontfelhők begyűjtése

Pontfelhők többféle módon keletkezhetnek. Általában valamilyen szenzor segítségével kerülnek felvételre, de akár számítógépes modellek felületéről pontok mintavételezésével is készülhetnek. Az utóbbi során keletkezett pontfelhőkre szintetikus adatként hivatkozunk. A pontfelhők felvételére használt különböző szenzorok az alábbi típusba sorolhatóak:

1. Time-of-Flight (ToF): Az ilyen elven rendszerek kibocsájtanak valamilyen jelet (fény vagy hang), amely egy felületbe ütközve visszatér az eszközhöz. A kibocsátás és a fogadás közötti időt lemérve és ismerve a jel sebességét meghatározható az eszköz és a jelet visszaverő felület közötti távolság. Ilyen technológiát használnak általában az önvezető járművekre szerelt lézer alapú távérzékelő (Light Detection and Ranging - LIDAR) eszközök vagy az utóbbi években a végfelhasználói mobileszközökben használt szenzorok (pl. a Google Tango projekt által támogatott eszközök).

2. Sztereó rekonstrukció: A sztereó képalkotás során két kamera által felvett kép között megfeleltetéseket keresünk, amelyeket felhasználva háromszögeléssel meghatározható a felületek távolsága. Ezt a technológiát használják általában az Intel RealSense [1] kamerák.

3. Strukturált fény: Ebben a folyamatban az eszköz egy ismert mintát vetít a felvenni kívánt helyszínre vagy objektumra és egy kamerával felvételt készít. A kivetített minta alapján kiszámíthatóak a felülethez tartozó mélységinformációk. Ilyen technológiát használnak az Apple cég által gyártott mobileszközökben használt TrueDepth szenzorok, amit a felhasználók azonosításához használnak.

A szakirodalomban a pontfelhők megnevezésére különböző terminológiák terjedtek el. Ezek a megnevezések gyakran egymással nagyon hasonló fogalmakat írnak le, kis eltérésekkel. Feldolgozás szempontjából ezek az adatok gyakran ugyanúgy kezelhetőek, ezért a fogalmakat gyakran felcserélik és ritkán definiálják pontosan. Az alábbiakban röviden ismertetem a szakirodalomban előforduló megnevezéseket.

Az egyik legismertebb 3D pontfelhő adatokkal dolgozó nyílt forráskódú algoritmusgyűjtemény (könyvtár) a Point Cloud Library (PCL [52]). A készítők a könyvtár megalkotásakor létrehoztak egy új adatformátumot a pontfelhő adatok tárolására (PCD - Point Cloud Data). A PCD adatformátum képes megkülönböztetni az ún. rendezett (organized) és rendezetlen (unorganized) pontfelhőket. A rendezett pontfelhőkre táblázat vagy mátrixként tekinthetünk és ilyen formában is kerülnek tárolásra. Ugyanis megadható egy szélesség ($width$) és egy magasság ($height$) érték, amely meghatározza a pontfelhőben lévő pontok számát ($width \cdot height$). Rendezett pontfelhőket képesek begyűjteni a sztereó és TOF szenzorok. Az előnye a rendezett pontfelhőknek, hogy plusz információ nyerhető ki belőlük a pontok kapcsolatai alapján (egymás melletti vagy alatti pontok), így egyes algoritmusok kihasználva ezt az előzetest tudást, hatékonyabb futásra képesek (pl. legközelebbi szomszéd keresések a pontok között). A rendezetlen pontfelhők esetében nincs semmilyen megszorítás adva a pontok helyzetére vagy szomszédságára. A rendezetlen pontfelhők esetén a PCD adatformátum magasság attribútumának az értéke mindig 1, míg a szélesség attribútuma megadja a pontfelhőben lévő pontok számát. Az ilyen pontfelhő valóban pontok halmaza, ugyanis semmilyen információnk nincs a pontok egymás közötti viszonyáról. A szakirodalomban fellelhető munkák nagy része rendezetlen pontfelhőkkel foglalkozik.

A PCD adatformátum képes a pontfelhőben lévő pontok koordinátái mellett más értékek tárolására is. A pontok rendelkezhetnek RGB színértékekkel, intenzitással vagy

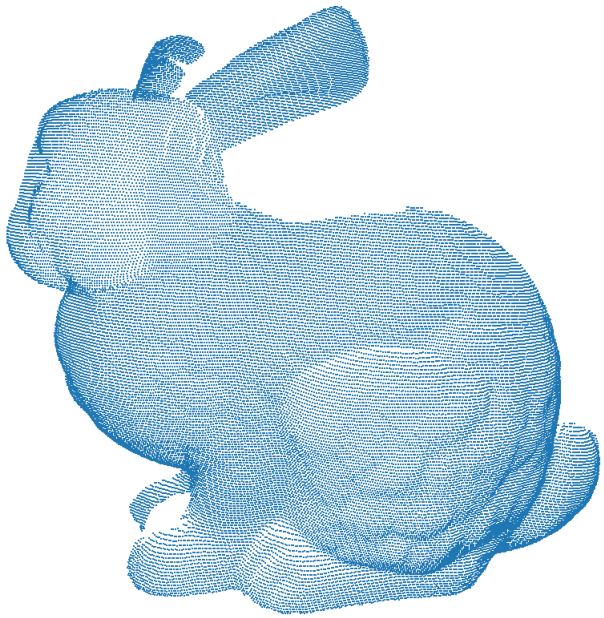
[1] https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html

normálvektorokkal is. A pontokról tárolt egyéb értékeket a PCD fejlécében van lehetőség felsorolni más információkkal együtt (tárolás módja, a pontok attribútumainak tárolásához szükséges tárhely stb.). További információk a PCD adatformátumról megtalálhatóak a PCL dokumentációjában [2].

Egy nagyon gyakori megkülönböztetése a pontfelhőknek, a 3D vagy 2.5D pontfelhők közötti különbség. Ha közérthetően szeretnénk megfogalmazni a különbséget a két típusú pontfelhő között, akkor azt mondhatjuk, hogy 2.5D pontfelhőnek hívjuk azokat a pontfelhőket, amely pontjait egy síkra vetítve a pontok nem kerülnek egymásra. Vagy akár úgy is mondhatjuk, hogy a pontfelhő egy nézőpontból volt felvéve, a szenzor és helyszín pozíciójának módosítása nélkül. Ha egy nézőpontból veszünk fel egy helyszínt vagy egy objektumot, akkor a felületek hátoldala nem látszik, így arról nem lesz információnk. Általában, a szenzorok nagy részével 2.5D pontfelhőt vagyunk képesek begyűjteni egy felvétel során. 3D pontfelhőt a 2.5D pontfelhők összeillesztésével (regisztrációjával) tudunk készíteni. Ennek egy tipikus esete, amikor a szkennerünk rendelkezik egy tálcával, amely 360°-ban forog körbe és az erre helyezett objektumról forgás közben megadott időközönként felvételt készít, amelyet végül összeillesztve 3D pontfelhő keletkezik (pl. HP 3D Structured Light Scanner [3]). Azonnal, összeillesztés nélkül készíthetünk 3D pontfelhőt, ha az utóbbi években egyre inkább elterjedt és önvezető járművekben gyakran használt 360°-os LIDAR szenzort használunk. Ilyen szenzorok a Velodyne által készített különböző szkennerek (pl. VLP-16 [4]). A 360°-os horizontális láthatárral rendelkező LIDAR szenzorok egy tengely körül körbe forognak, így a körülöttük lévő minden felület felé képesek sugarakat kibocsájtani. Könnyen látható, hogy ezáltal a pontfelhőben lévő pontok nem vetíthetőek le a koordináta-rendszer tengelyei által meghatározott síkokra anélkül, hogy a pontok egymásra kerülhessenek, azonban a felületeket így is csak egy irányból látjuk.

A 2.1. ábrán a szakirodalomban gyakran illusztrációs céllal felhasznált Stanford Bunny [5] látható. Az objektumot különböző nézőpontokból szkennelték. Az ábrán csak egy nézőpontból látható és ez a pontfelhő 40 256 pontból áll. Ez a pontfelhő 2.5D pontfelhőnek tekinthető.

[2] https://pcl.readthedocs.io/projects/tutorials/en/latest/pcd_file_format.html
[3] https://www.hp.com/us-en/campaign/3Dscanner/overview.html
[4] https://velodynelidar.com/products/puck/
[5] http://graphics.stanford.edu/data/3Dscanrep/



2.1. ábra: A Stanford Bunny objektumról készült pontfelhő felvétel.

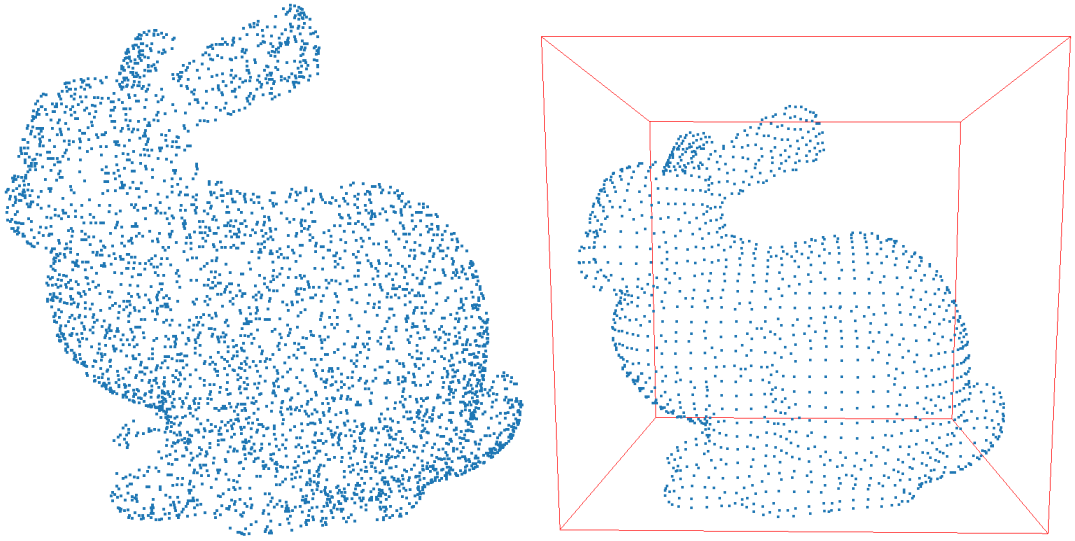
## 2.2. Előfeldolgozási algoritmusok

A különböző típusú szenzorok eltérő számosságú pontfelhők felvételére képesek. Napjainkban egy felvétel több százezer pontból is állhat, az egymásra illesztések után pedig nem ritka a több millió vagy milliárd pontból álló pontfelhők előfordulása sem. A pontfelhő adatokat feldolgozó algoritmusok költsége a bemeneti pontfelhő számosságától függ, így gyakran fontos a pontok számának csökkentése. Az algoritmusok futási idejének gyorsítása az egyik legfontosabb szempont, ezért a pontfelhők alulmintavételezése gyakran alkalmazott eljárás a területen.

### 2.2.1. Véletlen mintavételezés

Az egyik legkézenfekvőbb megoldás egy pontfelhő számosságának csökkentésére a véletlen mintavételezés. Véletlen mintavételezés alatt általában az egyenletes eloszlású véletlen mintavételezést értjük, amikor a pontfelhőben lévő minden pont azonos valószínűséggel kerül kiválasztásra. A PCL-ben implementált *RandomSample* osztályt Jeffrey Scott Vitter [17] munkája alapján készítették el, melynek futási ideje $\mathcal{O}(n)$. Véletlen alulmintavételezés során úgy csökkentjük a pontfelhő számosságát, hogy nem teszünk kivételt a pontok között, amely egyes esetekben előny, máskor hátrány tud lenni. Vegyük észre, hogy egyszerű geometriai felületek reprezentálásához egyes feladatok elvégzésekor nincs szükség sok pontra (pl. egy sík akár három pont segítségével megadható). A 2.2/(a).

ábrán a Stanford Bunny véletlen mintavételezéssel alulmintavételezett változata látható. A pontok 10%-át választottuk ki az eredeti felhőből.



a) b)

2.2. ábra: A Stanford Bunny pontfelhő a) véletlen alulmintavételezéssel és b) voxelrács alulmintavételezéssel (a piros vonalak a pontfelhő befoglaló dobozát mutatják). A véletlen alulmintavételezés során a pontok 10%-át választottuk ki, a voxelrács esetében pedig 0.005 volt a voxelméret. A voxelekbe eső pontokat a pontok centroidjával helyettesítettük.

### 2.2.2. Voxelrács mintavételezés

A voxelrács a háromdimenziós tér felosztása szabályos ráccsal. Elemei a voxelek, amelyek a voxel élei által meghatározott térfogattal rendelkeznek. A voxelek elképzelhetőek a pixelek háromdimenziós megfelelőiként. Gyakran használják a voxeleket megjelenítésére. Általában a voxelek kockák, azaz minden oldaluk egyforma hosszúságú, de a különböző oldalhosszúságú téglatestekből álló rácsot is voxelrácsnak hívják. A voxelek oldalhosszát nevezzük a voxelrács felbontásának vagy voxelméretnek.

A voxelrács alulmintavételezés során a bemeneti pontfelhőre voxelrácsot építünk egy megadott $v$ voxelfelbontással. A voxelrács kiterjedését a pontfelhő minimális területű befoglaló doboza határolja. Általánosan definiálva egy minimális területű befoglaló doboz egy olyan $N$ dimenziós hipertéglalap, amely magába foglalja az adathalmazban lévő összes $N$ dimenziós pontot, és a lehető legkisebb. Esetünkben ez egy háromdimenziós téglatest. Jelölje a bemeneti pontfelhőnk minimális térfogatú befoglaló dobozának szélességét $w$, magasságát $h$ és hosszát $l$. Ekkor a pontfelhőre készített $v$ felbontású voxelrácsban lévő voxelek száma:

$$N = \left\lceil \frac{w}{v} \right\rceil \cdot \left\lceil \frac{h}{v} \right\rceil \cdot \left\lceil \frac{l}{v} \right\rceil$$

Az egy voxelbe eső pontokat a későbbiekben egy ponttal reprezentáljuk, így egy tetszőleges számosságú pontfelhő mérete $N$ számúra csökkenthető. Vegyük észre, hogy nem lehetséges olyan voxelrács létrehozása, amely a pontfelhőben lévő pontok számát megnöveli. A voxelekbe eső pontok reprezentálásra egy ponttal több lehetőségünk is van. A legkevésbé költséges megoldás az, hogy a voxelekbe eső pontokat helyettesítjük a voxel közepével (vagy valamelyik sarkával), a többi pontot pedig elhagyjuk. Az üres voxeleket kihagyva így egy új pontfelhőt kapunk. Fontos megjegyezni, hogy az így kapott új pontfelhő pontjai nem lesznek azonosak a bemeneti pontfelhő pontjaival (azokat az eseteket kivéve, amikor egy pont pontosan a voxel közepén található, ami a számábrázolás miatt valós esetben szinte sosem fordul elő). Egy fokkal költségesebb megoldás, ha a voxelbe eső pontokat helyettesítjük a pontok súlypontjával, amely megegyezik a pontok koordinátájának számtani közepével. Legyen $\mathcal{V} = \{\mathbf{p_1}, ..., \mathbf{p_n}\}$ a voxelbe eső pontok halmaza. Ekkor a súlypontot $\overline{\mathbf{p}}$-vel jelöljük és a következő módon számoljuk ki:

$$\overline{\mathbf{p}} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{p_i}$$

ahol $n$ a voxelbe eső pontok száma, és $\mathbf{p_i} \in \mathcal{V}$ pedig a voxelbe eső $i$. pont. Ebben az esetben a kimenetként kapott pontfelhőnk jobban reprezentálja az eredeti pontfelhőt, azonban itt is olyan pontokból áll az új felhőnk, amelyek valószínűleg nem léteztek a korábbi pontfelhőben. A legköltségesebb megoldás az, ha a voxelbe eső pontokat a pontok medoidjával helyettesítjük:

$$medoid = \arg\min_{m \in \mathcal{V}} \sum_{i=n}^{n} d(\mathbf{m}, \mathbf{p_i}),$$

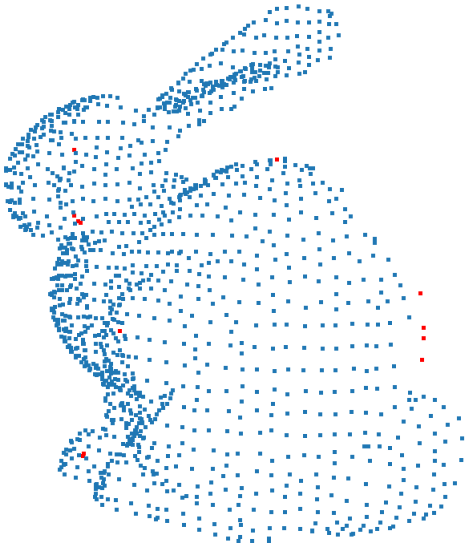
ahol $n$ szintén a voxelbe eső pontok száma, a $\mathbf{p_i} \in \mathcal{V}$ pedig a voxelbe eső $i$. pont, a $d(\cdot)$ pedig két pont közötti távolságot megadó függvény (a távolságmetrikákról későbbi fejezetben lesz szó részletesebben). Ez a megoldás azért költségesebb, mert a medoid kiszámításához minden voxel esetén $(n * (n-1))/2$ távolság kiszámítása szükséges. A fő különbség az előző megközelítésekhez képest, hogy medoidot használva a kimenetként kapott pontfelhő pontjai az eredeti pontfelhő pontjai közül kerülnek kiválasztásra. Ez a megoldás őrzi meg a felületekről a legtöbb információt, azonban ez a legköltségesebb is. Az értekezés későbbi részeiben a voxelrácsot többször is használom alulmintavételezésre. Az egy voxelbe eső pontok reprezentációjához választott módszert a megfelelő részeknél

külön kiemelem. A 2.2/(a). ábrán a Stanford Bunny pontfelhő voxelrács mintavételezéssel alulmintavételezett felhője látható. A voxelméret 0.005, a voxelbe eső pontokat pedig a pontok súlypontjával kerültek helyettesítésére. A piros vonalak a pontfelhő befoglaló dobozát mutatják. Ez a befoglaló doboz nem minimális területű, hanem a koordinátatengelyekkel párhuzamos élű.

### 2.2.3. Kiugró értékek eltávolítása

A pontfelhők felvétele során zajos és kiugró értékek keletkezhetnek. Ennek számos oka lehet, szenzortípustól függően. Például, a ToF típusú szenzorok esetében a tükröződő, fényvisszaverő felületek felvétele okozhat gondot vagy akár a kültéri felvételek során a levegőben szálló kisebb objektumok (falevél, szemét stb.) is zajszerű pontokat okozhatnak. Továbbá, a kibocsátott sugár visszaérkezési idejének mérése sem tökéletes, így a szenzorok specifikációinál is meg szokták adni a pontokhoz tartozó hibahatárt.

Általánosságban elmondhatjuk, hogy a pontfelhőben lévő, más pontoktól távol eső, néhány pontból álló ponthalmazokat kiugró pontoknak nevezzük és nemkívánatosnak tekintjük. Az ilyen pontok eltávolítása csökkenti a pontfelhőben lévő pontok számát, így a későbbiekben gyorsabb futási időket eredményezhet, továbbá megakadályozza, hogy egyes algoritmusok "értelmetlen" eredménnyel térjenek vissza (ilyen lehet a normálvektor becslés vagy jellegzetességleíró kiszámítás, amikről a fejezet későbbi részében írok). A kiugró pontok eltávolítása nem alulmintavételezés, viszont csökkenti a pontfelhő méretét, ezért úgy döntöttem, hogy ebben a fejezetben tárgyalom.



2.3. ábra: A Stanford Bunny pontfelhő oldalról. A piros pontok mutatják a kiugró pontokat.

A későbbiekben bemutatott munkáim során a kiértékeléshez használt pontfelhő adathalmazokon gyakran alkalmaztam előfeldolgozási lépéseket, amelynek általában része a kiugró értékek eltávolítása is. A feladat elvégzéséhez a Rusu és társai [39] által javasolt sugáralapú kiugró érték eltávolító módszert használtam.

A sugáralapú kiugró érték eltávolító algoritmusnak két paramétere van: egy $r \in \mathbb{R}$ sugár és egy $n \in \mathbb{N}$ nullától nagyobb egész szám, amely az $r$ sugarú környezetbe eső pontok számosságára ad egy határértéket. Egy pont akkor számít kiugró pontnak (eltávolíthatónak), ha a ponthoz tartozó $r$ sugarú szomszédság számossága kisebb, mint a paraméterül megadott szám, azaz $|\mathcal{N}_r| < n$.

A 2.3. ábra mutatja egy sugáralapú kiugró érték eltávolító algoritmus egy lehetséges eredményét. A bemeneti pontfelhő a Stanford Bunny voxelráccsal mintavételezett változata volt, az algoritmus paraméterei pedig a következők: $r = 0.01, n = 7$. Az ábrán jól látszik, hogy a pontfelhő szélén lévő magányos pontok kerülnek eltávolításra.

### 2.2.4. Pontfelhő felbontás

A kiugró értékek eltávolításakor láthattuk, hogy a módszer fontos paramétere volt az $r$ sugár, amely megadta, hogy egy lekérdezési pontnak mekkora környezetét vizsgáljuk. A későbbiekben látni fogjuk, hogy egy lekérdezési pont környezetének meghatározása számos algoritmus fontos lépése és így a sugár a különböző módszerek egyik legfontosabb paramétere. Mivel a különböző pontfelhőkben lévő pontok koordinátái eltérő skálán mozoghatnak, nem létezik olyan érték, amelyet használva mindig jó eredményt kapunk. A sugár beállítása számos dologtól függhet: az algoritmus céljától, a pontfelhő által reprezentált felületek tipikus méreteitől, a megoldandó feladattól, a felvételre használt eszköz tulajdonságaitól stb. Az egyik ilyen fontos tényező a ún. pontfelhő felbontás vagy pontfelhő sűrűség. A pontfelhő felbontása alatt egy olyan számot értünk, amely jellemzi a pontfelhő pontjai között lévő távolságokat. A munkámban leggyakrabban előforduló ilyen jellemzőt pontfelhő felbontásnak hívom (*pcr* - point cloud resolution) és $\rho$-val jelölöm. A pontfelhő felbontás kiszámolása a következő módon történik:

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{n} d(\mathbf{p_i}, \mathbf{p_i^1}),$$

ahol $n$ a pontfelhő számossága, $p_i$ a pontfelhő egy pontja, $d(\cdot)$ két pont távolságát megadó függvény, $\mathbf{p_i^1}$ pedig az $i$. pont legközelebbi szomszédja a pontfelhőben. Az eredeti Stanford Bunny pontfelhő (2.1. ábra) esetén a $\rho$ értéke 0.00058, míg a voxelrács mintavételezés után kapott pontfelhő (2.2/(b). ábra) esetén már 0.0031. Mivel az alul-

mintavételezés során ritkítottuk a pontfelhőt, így a pontok közötti átlagos távolságok is megnőttek. A későbbi fejezetekben gyakran előfordul, hogy az algoritmusok paramétereinek a beállításához a $\rho$ értékét vesszük alapul.

### 2.2.5. Normálvektor becslés

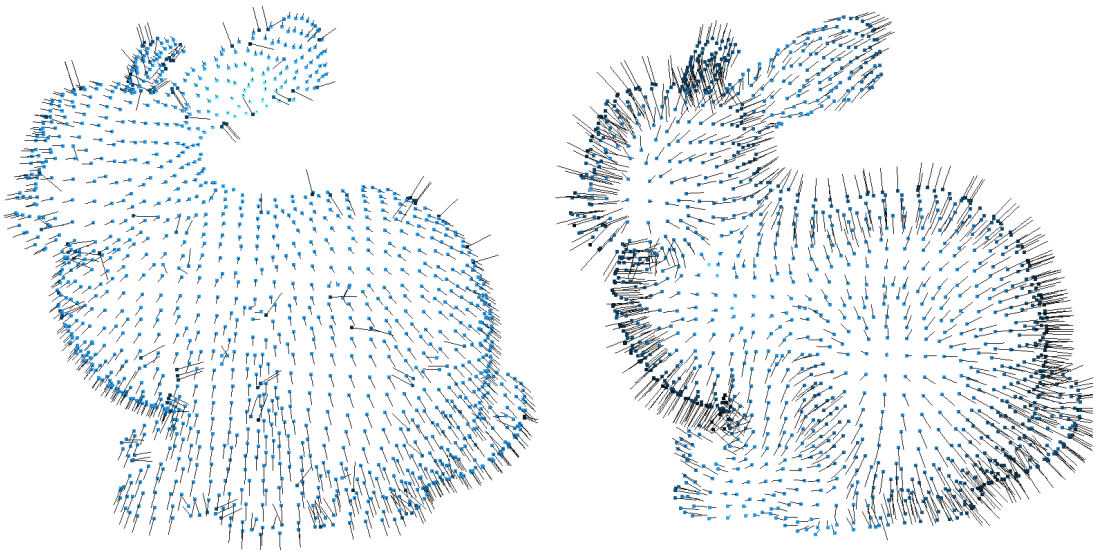
A későbbiekben bemutatott algoritmusok nagy része feltételezi, hogy a pontfelhő pontjai rendelkeznek normálvektorokkal. Ezért a pontfelhők feldolgozásánál mindig az első lépések között szerepel a pontok normálvektorainak a meghatározása (ha az adathalmaz nem rendelkezik ezzel az információval). Az egyik legelső és máig népszerű megoldást Hoppe és társai [24] javasolták. Ők a következő módon határozták meg egy pontfelhőben lévő pont normálvektorát. Először meghatározták a pont $k$-szomszédságát: $\mathcal{N}_k$ (a $k$-szomszédság definíciója fentebb megtalálható). Későbbi munkák [34] érveltek amellett, hogy a $k$-szomszédság helyett előnyösebb a pont $r$ sugarú környezetét venni. A következő lépésben elkészítették a $\mathcal{N}_k$ halmazban lévő pontok kovariancia mátrixát:

$$\mathcal{C} = \frac{1}{n} \cdot \sum_{i=1}^{n} (\mathbf{p_i} - \overline{\mathbf{p_i}}) \cdot (\mathbf{p_i} - \overline{\mathbf{p_i}})^T$$

Feltéve, hogy $\mathcal{C} \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, j \in \{0, 1, 2\}$ és $\vec{v_j}$ és $\lambda_j$ a kovariancia mátrix sajátvektorai és sajátértékei, továbbá $\lambda_0$ a legkisebb sajátérték ($0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$), akkor a kiválasztott pont normálvektora a legkisebb sajátértékhez tartozó sajátvektor, vagy annak negálása, azaz: $\vec{v_0}$ vagy $-\vec{v_0}$. A normálvektor iránya nem egyértelmű és előfordulhat nem konzisztens orientációja a normálvektoroknak, amikor ugyanazon a felületen lévő akár szomszédos pontok normálvektorai ellenkező irányba mutatnak. Azt, hogy az inkonzisztens normálvektor orientáció problémát okozhat már Hoppe és társai is észlelték és kidolgoztak egy módszert a globálisan konzisztens orientáció eléréséhez. A módszer során a szomszédos pontok normálvektorai úgy vannak negálva, hogy azok végül azonos irányba mutassanak. A módszer részletesebb leírása megtalálható a kapcsolódó publikációban [24].

A normálvektorok becslésére a későbbiekben új módszerek jelentek meg, például a szomszédságban lévő pontok távolsága alapján bevezetett súlyozás segítségével jobb eredményeket lehetett elérni [35]. A területen számos módszer ismert [40] és ma is jelennek meg új módszerek. A PCL-ben lévő normálvektorok becslését végző függvény Hoppe és társai normálvektor becslő és orientáló módszerét alkalmazza és az Open3D könyvtárban is megtalálható az algoritmusuk implementációja.

A 2.4. ábrán a Stanford Bunny voxelráccsal mintavételezett változata látható. A fekete vonalak a pontok normálvektorait mutatják. A bal oldali részábrán a szomszédság



a) b)

2.4. ábra: A Stanford Bunny pontfelhő voxelráccsal mintavételezett változata, a pontok normálvektoraival. a) a szomszédság meghatározásához használt sugár $r = \rho$ b) míg ebben az esetben $r = \rho \cdot 5$.

meghatározásához használt sugár $r = \rho$, míg a jobb oldali részábrán a sugár $r = \rho \cdot 5$. Látható, hogy a bal oldali ábrán használt sugár nem tette lehetővé a stabil normálvektor becslést, néhány pont normálvektora iránya inkonzisztens a szomszédaihoz képest. Ezzel szemben a jobb oldali ábrán ezek az anomáliák már eltűntek.

## 2.3. Pontfelhő regisztráció

A pontfelhő regisztráció (illesztés, összeillesztés) egy folyamat, melynek során két egymással átfedő pontfelhőt úgy transzformálunk a térben (eltolás és forgatás segítségével), hogy az átfedő részeik a lehető legjobban illeszkedjenek egymásra. A feladat formális meghatározása már nehezebb feladat. Tegyük fel, hogy $\mathcal{S}$ és $\mathcal{M}$ két pontfelhő. Ekkor keressük azt a $T \in \mathcal{T}$ térbeli transzformációt, amely minimalizálja a két felhő eltéréséből számolt hibát:

$$T = \arg\min_{T \in \mathcal{T}} E(T(\mathcal{S}), \mathcal{M}),$$

ahol $E(\cdot)$ a két felhő eltérését megadó függvény, $\mathcal{T}$ pedig az összes lehetséges transzformáció halmaza. Azért kell ennyire általánosan megfogalmazni a problémát, mert egyes speciális pontfelhők esetén nem használhatóak azok a módszerek, amelyek az általános esetekben. A $E(\cdot)$ hibát meghatározó függvényt gyakran a következő módon definiálják:

$$E(T(\mathcal{S}), \mathcal{M}) = \sum_{\mathbf{p} \in \mathcal{S}} d(T(\mathbf{p}), \mathbf{p_m}),$$

ahol $\mathbf{p_m} \in \mathcal{M}$ a $\mathbf{p}$ legközelebbi szomszédja az $\mathcal{M}$ pontfelhőből. Ebben az esetben a hiba akkor lesz a legkisebb, ha az egyik pontfelhőben lévő összes pont a lehető legközelebb esik a legközelebb lévő szomszédjához a másik pontfelhőből. Viszont az, hogy melyik a legközelebbi szomszédja attól függ, hogy milyen pozícióban vannak a felhők, így ez transzformációk alkalmazásával változhat. Ebből következik, hogy két olyan pontot fogunk egymáshoz közel transzformálni, amelyek egyébként nem ugyanazon felület azonos részeit reprezentálják. Vagy az is előfordulhat, hogy egy pontnak nincs megfelelő párja a másik pontfelhőben, így az mindenképp távol lesz a legközelebbi szomszédjától. Az ilyen és ehhez hasonló problémák megoldására komplexebb hibafüggvényeket is bevezettek, például ahol kiszűrésre kerülnek a távoli legközelebbi szomszéddal rendelkező pontok.

A pontfelhő regisztrációról szóló fejezet elkészítéséhez felhasználtam Fitzgibbon [29] és Holz és társai [67] munkáit, jelöléseket és megállapításokat.

### 2.3.1. Jellegzetességleíró alapú regisztrációs folyamat

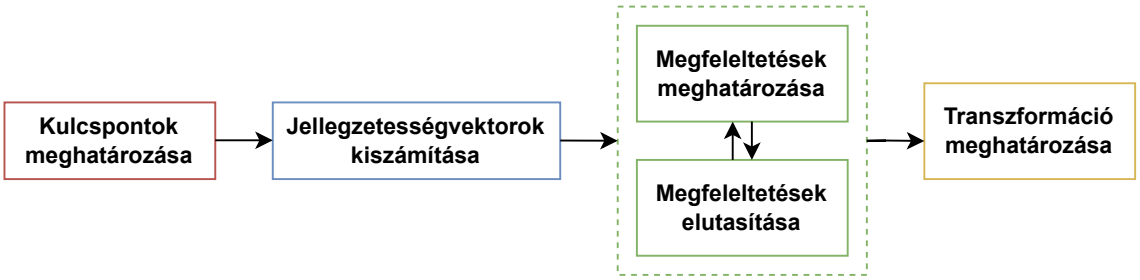
Az előző bekezdésben említett pontpárokat, amelyeket közel szeretnénk vinni egymáshoz transzformáció segítségével megfeleltetéseknek hívjuk. A megfeleltetések meghatározásának legegyszerűbb módja a fent vázolt eset, amikor megnézzük egy adott pont legközelebbi szomszédját a másik felhőből és a két pont egy megfeleltetést fog alkotni. A megfeleltetések létrehozásának egy korszerűbb módja a pontfelhő pontjainak jellemzése és a pontok jellemzése alapján hasonló pontokból megfeleltetéseket készíteni. A pontfelhőkben lévő 3D pontok jellemzését a jellegzetességleíró módszerek végzik. A jellegzetességleíró módszerek jellegzetességvektorokat adnak meg kimenetként, amelyek általában sokdimenziós, valós értékű vektorok. A jellegzetességleíró módszerek célja egy ponthalmaz oly módon való jellemzése, hogy az megkülönböztethető legyen más ponthalmazoktól. A jellemzéshez a ponthalmazban lévő pontok geometriai tulajdonságait használják fel, például a ponthalmazban lévő pontok egymáshoz való távolságát, normálvektoraik által bezárt szöget stb. Formálisan, egy jellegzetességleíró módszer megadható egy $F(\cdot)$ függvényként, amely egy $\mathbf{p_q}$ lekérdezési pontot és a lekérdezési pont szomszédságát ($\mathcal{N}$) kapja meg bemenetként és egy $n$ dimenziós jellegzetességvektort ad vissza kimenetként:

$$F(\mathbf{p_q}, \mathcal{N}) = \{x_1, x_2, ..., x_n\},$$

ahol $x_i, i \in [1..n]$ a jellegzetességvektor $i$. eleme. A jellegzetességleíró módszerekről egy későbbi fejezetben részletesebben lesz szó. Az alapfogalmak bevezetése után rátérhetünk a regisztrációs folyamatok egy speciális területére.

A jellegzetességleíró alapú regisztrációs folyamat egy olyan pontfelhő regisztrációs módszer, amely a pontok közötti megfeleltetések meghatározásához jellegzetességleírókat és az általuk készített jellegzetességvektorok közötti távolságokat használják.

A jellegzetességleíró alapú regisztrációs folyamat lépései megadhatóak általánosan. Ettől az egyes módszerek eltérhetnek, például egy lépést kihagyhatnak, vagy valamelyiket többször végrehajtják. Az egyes lépéseket részletesen kifejtem a megfelelő fejezetekben. A folyamat lépéseit tekinti át az 2.5. ábra.




2.5. ábra: Jellegzetességleíró alapú regisztrációs folyamatok főbb lépései. Az egyes módszerek ettől eltérhetnek.

### 2.3.2. Kulcspontok meghatározása

Vezessük be először a kulcspontot, mint fogalmat. Kulcspontoknak [6] nevezzük a pontfelhő azon pontjait, amelyeket valamilyen kulcspontdetektáló módszer kiválaszt. A kulcspontdetektáló módszerek próbálnak a pontfelhőből valamilyen szempontból érdekes pontokat kiválasztani. Ilyen érdekes pontok lehetnek egy pontfelhőben az objektumok sarkait vagy éleit reprezentáló pontok. A kulcspontok célja az, hogy a gyakran költséges jellegzetességleíró számítást ne kelljen a pontfelhő összes pontjára elvégezni. A jellegzetességleírók számára megadott bemeneti pontok számosságának a csökkentésére a legegyszerűbb mód a pontfelhő alulmintavételezése (pl. véletlen vagy voxelrács mintavételezés). Ha valóban a pontfelhőben létező pontokat akarunk kiválasztani, a voxelrács mintavételezés medoidok használatával megfelelő lehet. Ezeket a módszereket azonban nem lehet valós kulcspontdetektáló algoritmusnak tekinteni. Bár a feldolgozandó pontok számát csökkentik, nem teljesítik azt a feltételt, hogy a pontfelhő érdekes pontjait válasszák ki. Mian és társai

[6] keypoints, interest points

[41] munkája alapján a kulcspontoknak lehetőség szerint eleget kell tenniük a következő követelményeknek:

1. Ugyanarról a helyszínről vagy objektumról készült felvételek esetén a kulcspontoknak megismételhetőnek kell lennie, azaz ugyanazokat a pontokat kell kulcspontként meghatározni az eltérő felvételeken.

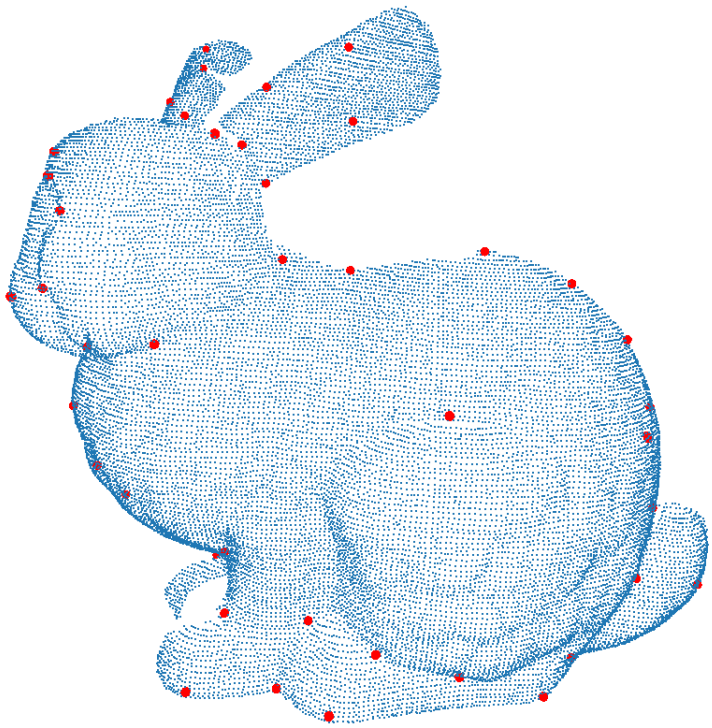
2. A kulcspont környezetébe eső pontok alapján lehetséges legyen egy stabil lokális koordináta-rendszert meghatározni, amelyet felhasználva forgatásra invariáns jellegzetességleírók készíthetőek.

3. A kulcspont környezetének jól megkülönböztethetőnek kell lennie, hogy a környezetből számolt jellegzetességleírók egyediek legyenek.

A követelmények megfogalmazása általános és talán kicsit homályos is. Jól látható, hogy a kulcspontokkal szemben támasztott követelmények szoros kapcsolatban vannak a jellegzetességleírókkal. A három követelményből kettő is azzal kapcsolatos, hogy a későbbiekben a kulcspontokra kiszámolt jellegzetességleírók jók legyenek. Emiatt a kulcspontdetektáló módszerek kiértékelése is nehézségekbe ütközik. Megjelentek munkák [59] [53], amelyek kiértékelték a kulcspontdetektáló algoritmusokat a megismételhetőség szempontjából eltolások, elforgatások és zaj hozzáadása mellett. Később megjelentek olyan munkák, amelyek a jellegzetességleírókkal való szoros kapcsolat miatt jellegzetességleíró specifikus kulcspontokat detektáltak [71], az utóbbi években pedig a gépi tanulás alapú módszerek is egyre nagyobb népszerűségre tesznek szert [88]. Az egyes kulcspontdetektáló algoritmusok részletesebb leírása megtalálható a hivatkozott cikkekben. Az 2.6. ábrán az Intrinsic Shape Signature (ISS [43]) kulcspontdetektáló által kiválasztott kulcspontokat láthatjuk a Stanford Bunny pontfelhőn.

### 2.3.3. Jellegzetességleírók

Korábban már általánosan bevezettük a jellegzetességleíró módszereket. Akkor úgy hangzott a meghatározás, hogy a jellegzetességleíró módszerek célja egy ponthalmaz jellemzése olyan módon, hogy az megkülönböztethető legyen más ponthalmazoktól. Az általános megfogalmazás azért volt fontos, mivel a jellegzetességleírók csoportosítása alapján egyes típusok között eltérhetnek a bemeneti adatok. Han és társai [92] összegyűjtötték az ismert jellegzetességleírókat és elkészítették az osztályozásukat. Az ő osztályozásuk alapján megkülönböztethetünk globális és lokális jellegzetességleírókat. A globális jellegzetességleírók



2.6. ábra: Az ISS módszer által kiválasztott kulcspontok a Stanford Bunny pontfelhőn. Pirossal láthatóak a kulcspontok.

bemenete egy teljes felvétel (helyszín vagy objektum) és erre készítenek egy jellegzetességvektort. Az ilyen leírókat főleg helyszín-lokalizációra szokták használni. Ide tartozik a teljesség igénye nélkül a Scan Context [95], Global Structure Histogram [55], The Global Orthographic Object Descriptor [75] stb.

Ezzel szemben a lokális jellegzetességleírók bemenete egy lekérdezési pont és általában egy sugár (vagy egy $k$ érték), amely meghatároz egy szomszédságot a pont körül, így ezt a szomszédságot tartalmazó ponthalmaz tekinthetjük a módszerek bemeneteként. Azonban a lokális jellegzetességleírók esetében fontos szerepe van a lekérdezési pontnak is, és annak viszonya a szomszédságában lévő pontokhoz, ugyanis egyes módszerek súlyozzák a szomszédságban lévő pontokat a lekérdezési ponttól való távolságuk alapján. Han és társai [92] és más jellegzetességleírókat összehasonlító munkák [66] is tovább osztályozzák a lokális jellegzetességleírókat térbeli eloszlás és geometriai tulajdonságokon alapuló módszerekre.

A térbeli eloszláson alapuló módszerek lényege, hogy a lekérdezési pont körüli szomszédságot térbeli részekre (láda, cella) osztják és az adott cellába eső pontok tulajdonságai határozzák meg a cella által reprezentált jellegzetességet. Erre egy egyszerű példa egy olyan módszer, amely felosztja a lekérdezési pont környezetét megadó gömböt 8 egyforma részre és megszámolja az egyes részekbe eső pontokat. Ha felsoroljuk az egyes cellákba eső pontok arányát, akkor olyan hosszúságú jellegzetességvektort kapunk, amely hossza

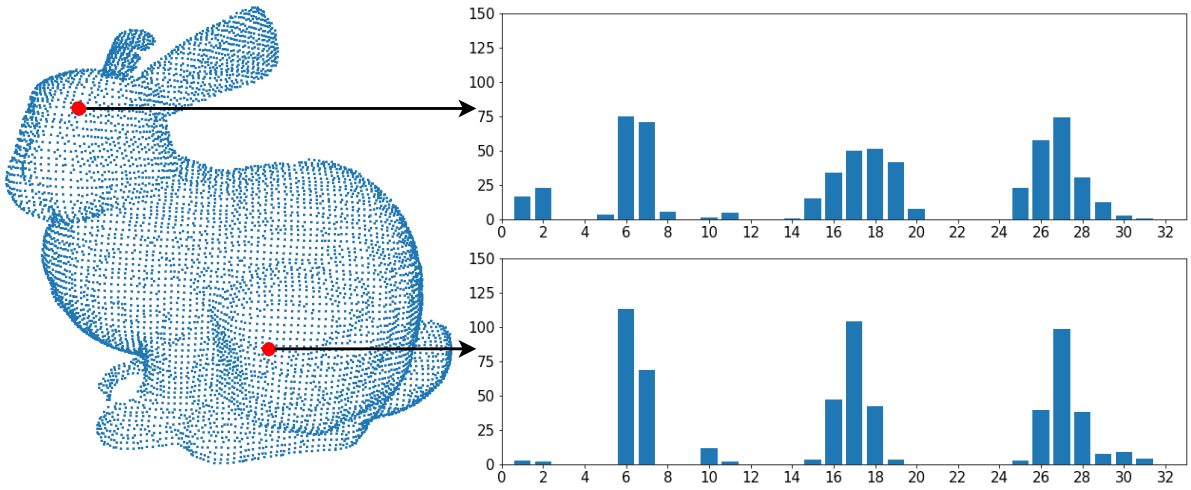
egyenlő a cellák számával, az elemeinek az értéke pedig attól függ hány pont esett az adott cellába. Egy fontos kiegészítést kell tennünk ennél a résznél, hiszen a bemeneti pontfelhő tetszőleges elforgatással rendelkezhet, így a vázolt egyszerű jellegzetességleíró nem lesz invariáns az elforgatásra. Ennek megoldására a jellegzetességleíró algoritmusok első lépésben egy lokális vonatkoztatási rendszert (LRF) készítenek a szomszédság alapján. Természetesen a szakirodalomban fellelhető módszerek általában komplexebb megoldásokat használnak a cellák jellemzésre, mint a cellába eső pontok megszámolása. Guo és társai [66] munkája alapján térbeli eloszláson alapuló módszernek tekinthető a Spin Image [94], Unique Shape Context [48], Rotational Projection Statistics [57] stb.

A geometriai tulajdonságokon alapuló módszerek általában hisztogramokat készítenek a szomszédságba eső pontok geometriai tulajdonságai alapján (normálvektorok iránya, görbület, pontok közötti távolságok, normálvektorok által bezárt szögek). A kiszámolt értékeket hisztogramládákba rakják és ez alapján készítik el a jellegzetességvektorokat. A jellegzetességleírók e csoportjának a legismertebb képviselői a Point Feature Histogram [38] és a Fast Point Feature Histogram [42]. Ide sorolhatóak még a szomszédság kovarianciamátrixának a sajátértékeiből és sajátvektoraiból információt kinyerő leírók is [104].

A munkám során kizárólag lokális jellegzetességleírókkal foglalkoztam, amelyek között volt térbeli eloszláson és geometriai tulajdonságokon alapuló módszer is. Kiemelt figyelmet fordítottam a Fast Point Feature Histogram (FPFH) módszer vizsgálatára, mivel összehasonlítások alapján ez bizonyul az egyik legjobb jellegzetességleírónak [66]. A 2.7. ábrán látható a Stanford Bunny pontfelhő két tetszőlegesen kiválasztott pontja (piros). Látható, hogy a két pont környezetében lévő pontok eltérő felületeket reprezentálnak. Az ábra jobb oldalán láthatóak a két pontra kiszámolt 33 dimenziós FPFH jellegzetességvektorok. Szemmel láthatóan a két jellegzetességvektor különbözik: a felső pont esetén az 1. és 2. elemnek megfelelő hisztogramládákba több érték esett, és 6., 17. és 27. elemek magasabbra nyúlnak az alsó jellegzetességvektor esetében.

### 2.3.4. Megfeleltetések

A pontok közötti megfeleltetések létrehozásának a célja, hogy különböző felvételeken megtaláljunk olyan részeket, amelyek a felületek azonos részeit reprezentálják, így a megfeleltetéseket alkotó pontpárok egymásra illesztése a térben a felületeket reprezentáló pontfelhők egymásra illesztését eredményezi. Könnyű belátni, hogy háromdimenziós pontok esetén legalább 3 helyes megfeleltetésre van szükségünk ahhoz, hogy két pontfelhőt egymásra illesszünk.




2.7. ábra: A Stanford Bunny pontfelhő két kiválasztott pontja (piros) és a pontok FPFH jellegzetességvektora

A jellegzetességleíró alapú regisztrációs folyamatokban a megfeleltetések meghatározásához a korábban kiszámolt jellegzetességvektorokat használjuk fel. Ahhoz, hogy megtaláljuk a felületek azonos részeit reprezentáló pontokat a két pontfelhő pontjainak a jellegzetességvektorait hasonlítjuk össze. Legyen $F_1$ és $F_2$ két jellegzetességvektor. Ekkor a két jellegzetességvektor leírótávolsága:

$$\Gamma = d(F_1, F_2),$$

ahol a $d(\cdot)$ két vektor közötti távolságot megadó függvény. Azt mondjuk, hogy két pont hasonló, ha a pontok jellegzetességvektorainak a leírótávolsága nullához közelít és két pont különböző, ha a leírótávolság minél nagyobb. Vegyük észre, hogy tetszőleges szkennerrel begyűjtött különböző valós pontfelhők esetében két pont jellegzetességvektorának a leírótávolsága sosem lesz nulla, még akkor sem ha a két pont valóban egyazon felületen lévő azonos részeket reprezentálnak.

A jellegzetességvektorok közötti távolság mérésére a legelterjedtebb választás az euklideszi távolság (L2):

$$d_{L2} = \sqrt{\sum_{i=1}^{n} (\mathbf{p_i} - \mathbf{q_i})^2},$$

ahol $F_1$ és $F_2$ két $n$ elemből álló jellegzetességvektor és $\mathbf{p_i} \in F_1, \mathbf{q_i} \in F_2$. Az euklideszi távolságon kívül használják még a Manhattan távolságot (L1) és a Kullback-Leibler-divergenciát (KL):

$$d_{L1} = \sum_{i=1}^{n} |\mathbf{p_i} - \mathbf{q_i}|$$

$$d_{KL} = \sum_{i=1}^{n} |\mathbf{p_i} - \mathbf{q_i}| \cdot \ln \frac{\mathbf{p_i}}{\mathbf{q_i}}$$

A legkézenfekvőbb nyers-erő algoritmus a megfeleltetések meghatározására az, hogy egy ciklussal végig iterálunk az egyik pontfelhő pontjain és minden ponthoz megkeressük a jellegzetességvektorának legközelebbi szomszédjához tartozó pontot (1. algoritmus). Az algoritmus végén egy halmazt kapunk, amelyekben megfeleltetések (pontpárok) vannak.

**Algoritmus 1** Megfeleltetések meghatározása

INPUT: $\mathcal{F}_1, \mathcal{F}_2$ ▷ két jellegzetességvektor-halmaz
OUTPUT: Megfeleltetések halmaza
1: $\mathcal{C} \leftarrow \{\}$
2: **for** $i = 1, 2, \ldots, N$ **do**
3: $F_1 \leftarrow \mathcal{F}_1(i)$
4: $j \leftarrow \mathit{IndexOfNearestNeighbor}(\mathcal{F}_2, F_1)$
5: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(i, j)\}$
6: **end for**
7: **return** $\mathcal{C}$

A javasolt nyers-erő algoritmus a megfeleltetések meghatározására egy olyan megfeleltetéshalmazt ad meg kimenetként, amelynek a számossága egyenlő az egyik pontfelhő számosságával. Ha ezt a megfeleltetés halmazt bemenetként adjuk a transzformációt meghatározó algoritmusnak, akkor nagy eséllyel nem az elvárt transzformációt fogjuk kapni, mivel a megfeleltetések között sok hibás is előfordulhat. Korábban említésre került, hogy egy transzformációhoz elegendő három megfeleltetés, így a megfeleltetéseinkből bátran eltávolíthatunk hibás megfeleltetéseket.

A megfeleltetések eltávolítását különböző megszorítások, feltételek és algoritmusok végzik, gyakran a meghatározásuk közben. Egy gyakran használt megszorítás [79] a megfeleltetések szűrésére a reciprocitás teszt, amely csak akkor engedi egy megfeleltetés megtartását, ha a megfeleltetés két pontjának jellegzetességvektora kölcsönösen legközelebb szomszédjai egymásnak. A reciprocitás teszttel kiegészített algoritmust a 2. algoritmus mutatja.

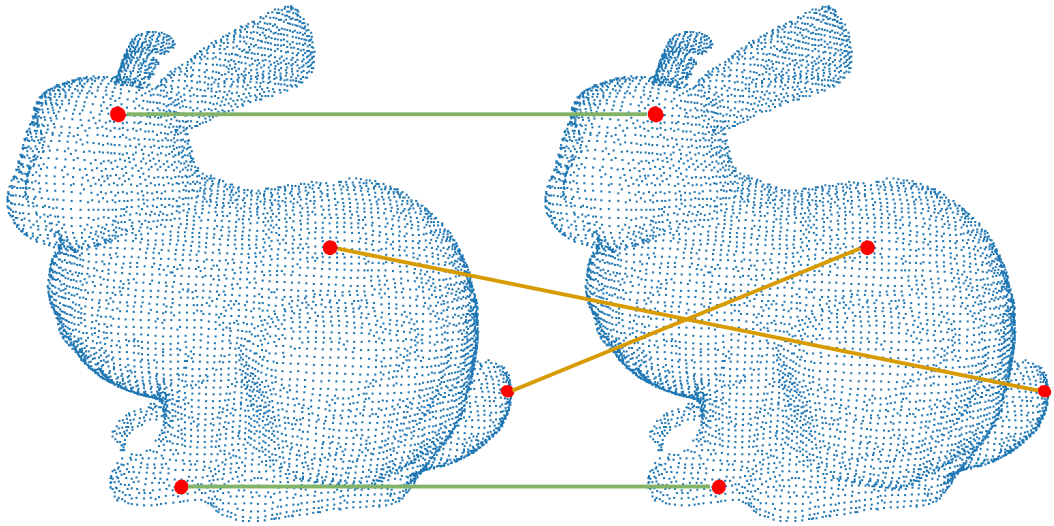
Az 2.5. ábrán a megfeleltetéssekkel kapcsolatban két lépés is szerepel: megfeleltetések meghatározása és megfeleltetések elutasítása. A két doboz közötti nyilak azt jelzik, hogy ez a két lépés egymás után többször is szerepelhet. A 2.8. ábrán megfeleltetések illusztrációja látható a Stanford Bunny pontfelhőn. A piros pontok a kulcspontokat jelölik a

**Algoritmus 2** Megfeleltetések meghatározása reciprocitás teszttel

INPUT: $\mathcal{F}_1, \mathcal{F}_2$ ▷ két jellegzetességvektor-halmaz
OUTPUT: Megfeleltetések halmaza

$\mathcal{C} \leftarrow \{\}$
**for** $i = 1, 2, \ldots, N$ **do**
    $F_1 \leftarrow \mathcal{F}_1(i)$
    $j \leftarrow IndexOfNearestNeighbor(\mathcal{F}_2, F_1)$
    $F_2 \leftarrow \mathcal{F}_2(j)$
    $k \leftarrow IndexOfNearestNeighbor(\mathcal{F}_1, F_2)$
    **if** $i = k$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(i, j)\}$
    **end if**
**end for**
**return** $\mathcal{C}$

pontfelhőkön, a zöld vonalak a helyes míg a sárga vonalak a hibás megfeleltetéseket. A megfeleltetések visszautasításának a célja a hibás megfeleltetések kiszűrése.



2.8. ábra: Megfeleltetések illusztrációja. Helyes megfeleltetések zölddel és hibás megfeleltetések sárgával.

### 2.3.5. Transzformáció meghatározása

A transzformáció becslő algoritmusukról részletes összefoglalót ad Zhao és társai munkája [117]. Ők összegyűjtötték a legnépszerűbb módszereket, csoportokba sorolták őket és összehasonlították. Az összehasonlítás során nem csak jellegzetességleírók által meghatározott megfeleltetéseket, hanem szintetikus megfeleltetéseket is használták.

Ennél a lépésnél fontos megjegyeznünk, hogy a jellegzetességleíró alapú regisztrációs folyamat végén kapott transzformáció gyakran nem tökéletes, és nem is ez a célja. Az

így kapott transzformációt durva transzformációnak [7] hívják, amely jelzi, hogy általában még egy lépés következik, amely nem szerepel a 2.5. ábrán. Ezt a lépést transzformáció finomításnak hívják [8] és a célja az, hogy a nem tökéletes kezdeti illesztést iteratívan minél pontosabbá tegye. Ennek megoldására az Iterative Closest Point algoritmust és annak különböző változatait szokták használni [23, 22, 33, 78, 32, 89].

Pont megfeleltetések alapján a transzformációs mátrixok meghatározására a legelterjedtebb megoldás Kabsch algoritmusa [14], amely szinguláris felbontást használ (SVD) a megfeleltetésekben szereplő pontok távolságainak a minimalizálására [19]. Az ICP és Kabsch algoritmusa részletesebb kifejtésre kerül a B függelékben.

[7] coarse transformation
[8] transformation refinement

# 3. fejezet

# Jellegzetességleírók dimenziócsökkentésének a vizsgálata

## 3.1. Bevezetés

Az egyik legfontosabb alkalmazása a pontfelhőknek (2D és 3D) a regisztráció. A regisztráció során olyan transzformációkat keresünk, amelyek kettő vagy több pontfelhőt egymásra illesztenek úgy, hogy a távolság a felületeket reprezentáló pontok között minimális legyen. A regisztráció bizonyos eseteiben (nagy méretű felhők, globális regisztráció, sablonillesztés) a jellegzetességleíró alapú regisztrációs módszerek széles körben elterjedtek [52]. Az ilyen folyamatok felépítését részletesen kifejtettem a 2 fejezetben.

A jellegzetességleíró alapú regisztrációs módszerek egyik legfontosabb eleme - ahogy a neve is mutatja - maga a jellegzetességleíró. Azonban nem csak a regisztrációhoz használhatóak a jellegzetességleírók, hanem az objektumok felismeréséhez [72], pontfelhő szegmentáláshoz és geometriai primitívek (sík, él, sarok) detektálásához is. Ebből is látható, hogy a jellegzetességleírók számos feladatra alkalmazhatóak, mivel a céljuk alapvetően egy pont környezetében lévő, a pontok által reprezentált felület geometriájának a jellemzése.

Az elmúlt években számos különböző leírót javasoltak és megjelentek kiváló összefoglaló cikkek is a témában [66]. Későbbi fejezetekben részletesebben írok majd az általunk megvizsgált jellegzetességleírók működéséről. A jellegzetességleírók általában sokdimenziós vektorok (akár több száz elemből is állhatnak). A pontfelhők pontjainak jellegzetességvektorain leggyakrabban végzett művelet a a $k$-legközelebbi szomszéd ($k$-NN [1]) keresés. Ugyanis, a $k$-NN lekérdezések elengedhetetlenek, ha különböző pontfelhők pontjai között megfeleltetéseket akarunk meghatározni.

[1] $k$-nearest neighbor

Beyer és társai már klasszikussá vált tanulmányukban [27] megmutatták, hogy számos felhasználási esetben egy pont legközelebbi és legtávolabbi szomszédjaihoz vett távolságok hajlamosak egyre közelebb kerülni egymáshoz, ahogy a dimenziók száma növekszik. Ezekben az esetekben a legközelebbi szomszéd fogalma értelmetlenné, semmitmondóvá válik. Arra is rámutatnak, hogy már nem túl nagy dimenziószám esetén (12-15) is értelmét vesztheti az indexstruktúrák használata, mivel a szekvenciális szkennelés gyorsabb eredményt ad. A munkájukban arról is értekeznek, hogy egyes esetekben a sokdimenziós vektorok a sokdimenziós térnek csak egy részhalmazába esnek, így az adatok ún. belső dimenzionalitása [2] jelentősen alacsonyabb, mint az eredeti. A belső dimenzionalitás az adatok leírásához szükséges attribútumok minimális száma [12]. A pontfelhők felvételének tulajdonságai alapján feltételezzük, hogy a jellemzésükre szolgáló jellegzetességleírók belső dimenzionalitása alacsonyabb lehet, mint az eredetileg használtak.

Más munkák különböző dimenziószámú és eloszlású adatok esetében vizsgálták az olyan indexstruktúrák mint az R-fák teljesítményeit legközelebbi szomszéd keresések során (pl. [26], [30]). Ha egy adathalmaz rendelkezik az önhasonlóság [3] tulajdonsággal, akkor a keresések gyorsasága az adathalmaz belső dimenzionalitásától ("fraktáldimenzió") függ [30]. Samet szerint [37, Chapter 4.] ha egy adathalmaz belső dimenzionalitása alacsonyabb, mint az eredeti, az jó ok arra, hogy valamilyen dimenziócsökkentési módszert alkalmazzunk rajta, mielőtt eltárolnánk az adatbázisban.

Egy másik ok, amit a könyvben említenek, hogy mivel a dimenzionalitás növekedésével az indexstruktúrák gyerekcsúcsainak tárhelyigénye megnő, így egyre kevesebb gyerekcsúcs fér el az adott kapacitású helyen, így a fák magassága nő, ami tovább csökkenti a struktúrák teljesítményét. A fenti okok miatt főkomponens analízis (PCA) segítségével dimenziócsökkentést végzünk a jellegzetességvektorokon és megpróbáljuk a jellegzetességleírók belső paraméterinek módosításával csökkenteni a jellegzetességvektorok hosszát. Megvizsgáljuk hogyan változik a leíróképességük a dimenziócsökkentési módszereket alkalmazva.

### 3.1.1. Kapcsolódó munkák

Guo és társainak [66] munkája alapján a jellegzetességleírók legfontosabb tulajdonsága a leíróképesség, amely azt mutatja, hogy a jellegzetességleíró milyen jól képes egységbe foglalni és reprezentálni a pontok által meghatározott felületek jellegét. Számos munka megjelent [66] [60] [107], amelyek a 3D jellegzetességleírókat hasonlították össze. Guo és társai *precision* és *recall* értékeket számoltak a legközelebbi szomszéd keresések ál-

[2] intrinsic dimensionality
[3] self similarity

tal meghatározott megfeleltetésekre, és ezeket hasonlították össze egymással. Azonban ezek a munkák általában a jellegzetességleírók alapértelmezetten megadott paramétereit használták.

Ezzel szemben Prakhya és társai [85] a jellegzetességleírók dimenzionalitását főkomponens analízis segítségével csökkentették és megvizsgálták hogyan változik a leíróképességük. Az általuk bevezetett RRR metrikát használták a leíróképesség mérésére, amely kis mértékben eltér a korábbi módszerek mérési módszertanától, de ugyanúgy a megfeleltetések helyességét vizsgálja.

### 3.1.2. A vizsgált jellegzetességleírók bemutatása

Ebben a munkában három jól ismert jellegzetességleírót vizsgálunk meg: Point Feature Histogram (PFH [38]), Fast Point Feature Histogram (FPFH [42]) és Spin Iamge (SI [94]). Az alapértelmezett paraméterekkel kiszámolt jellegzetességvektorok hossza 125 (PFH), 33 (FPFH) és 153 (SI).

A PFH jellegzetességleíró a lekérdezési pont szomszédságában lévő összes pontpárra három jellemzőt számol ki ($d = 3$). Az idézett cikkben egy további jellemzője is szerepel a pontpároknak, de később kiderült, hogy a pontok közötti távolság nem növeli a leíró teljesítményét, így azt később elhagyták. A kimeneti jellegzetességvektorok hossza attól függ, hogy a három különböző jellemzőt hány hisztogramládába osztjuk fel ($b$). Mivel minden pontpár mindhárom jellemző szerint felvesz egy értéket ezért összesen $b^d$ hisztogramládát kapunk. Az alapértelmezett implementációban a hisztogramládák száma 5, így $b^d = 5^3 = 125$ elemből áll a jellegzetességvektor. Ez egy $5 \cdot 5 \cdot 5$ részre felosztott háromdimenziós kockaként képzelhető el.

Az FPFH jellegzetességleíró nagyon hasonlít a PFH-hoz, mivel annak továbbfejlesztése. Ugyanazt a három jellemzőt számolja ki a pontpárokra, azonban a pontpárok meghatározása máshogy történik. A másik különbség a két leíró között, hogy az FPFH esetében az egyes jellemzők hisztogramládái, ahelyett, hogy egy teljesen korrelált teret alkotnának, csak össze vannak fűzve, így $b \cdot d$ hosszúságú jellegzetességvektorokat kapunk. Az alapértelmezett implementációban minden jellemző egyforma számú hisztogramládára van felosztva (11), így az alapértelmezett hossza a jellegzetességvektoroknak $11 \cdot 3 = 33$. A PFH és FPFH jellegzetességleírók és azok létrehozása részletesen ki van fejtve az értekezés A. függelékében.

Végezetül, az SI jellegzetességleíró esetében a képszélesség[4] ($i$) és a ládaméret ($b$) paraméterek határozzák meg a jellegzetességvektor hosszát. A képszélesség megadja azt,

[4]image width

hogy hány darab ládából álljon a négyzet, amely felosztja a lekérdezési pont környezetét. Így a környezet meghatározásához használt sugár $i \cdot b$, míg a jellegzetességvektor hossza $(i+1) \cdot (2i+1)$ lesz. További információk az egyes jellegzetességleírókról megtalálhatóak a hivatkozott munkákban.

## 3.2. Dimenziócsökkentés

A munkánkban kétféle módon csökkentettük a jellegzetességleírók dimenzionalitását. Az egyik mód a jellegzetességleírók belső paraméterei módosításával való csökkentés. Ahogy azt a korábbi fejezetben láttuk, a jellegzetességleírók különböző módszereket használhatnak a vektorok elkészítéséhez, azonban sok leíróban közös, hogy valamilyen kiszámolt értékek alapján, kosarazással alakítják ki a vektor dimenziószámát. Ezért szinte minden jellegzetességleíró által elkészített jellegzetességvektor dimenziószáma csökkenthető az algoritmus belső paramétereinek a módosításával. Azért hívjuk ezeket belső paraméternek, mert az implementációkban ezek általában nincsenek kivezetve, mint beállítható paraméter. Az általunk vizsgált lehetséges dimenziócsökkentések egyik módja a belső paraméterek módosításával történő dimenziócsökkentés.

A belső paraméterek módosításával nem érhető el tetszőleges dimenziószám, mivel az egyes algoritmusok megszorításait figyelembe kell vennünk. Az előző fejezetben megadtam, hogy az általunk vizsgált módszerek által létrehozott jellegzetességvektorok dimenziószámát hogyan lehet meghatározni. Az PFHF jellegzetességleíró az algoritmusban meghatározott pontpárokra három jellemzőt számol ki. Ha egyformán szeretnénk felbontani a pontpárok jellemzőit, akkor mindig hárommal osztható dimenziószámú jellegzetességvektort kapunk, így a lehetséges dimenziószámai $3q, (q \in \mathbb{N})$.

A másik módszer a dimenziócsökkentésre, amelyet használtunk a vizsgálatunkhoz a széles körben ismert főkomponens analízis. Ahhoz, hogy az adathalmazban lévő összes pontfelhő pontjainak a jellegzetességleírói ugyanazzal a transzformációval legyenek transzformálva, az főkomponens analízist az összes pontot bemenetként megadva kell elvégeznünk. A főkomponens analízis segítségével lehetőségünk van tetszőleges dimenziószámú vektorokat készíteni az eredeti jellegzetességvektorokból.

### 3.2.1. A kiértékelés módszertana

Guo és társai alapján [66] a jellegzetességleírók leíróképességének a kiértékeléséhez a $precision-recall$ görbét [5] használjuk. Szerintük a PRC alkalmasabb a 3D pontfelhők jel-

[5] precision-recall curve (PRC)

legzetességleíróinak kiértékelésére, mint az adatbányászati módszerek kiértékelésére széles körben használt ROC görbe [6]. A PRC elkészítéséhez az alábbi lépéseket végezzük el.

Első lépésben normális eloszlású véletlen mintavételezés segítségével pontokat választunk a pontfelhőkből. Ezek a pontok lesznek a kulcspontjaink. Fontos megjegyezni, hogy ebben a lépésben használhattunk volna valós kulcspontdetektáló algoritmust, amelyek valóban jól megkülönböztethető, egyedi, megismételhető és stabil kulcspontokat detektálnak. Azonban valamilyen kulcspontdetektáló módszer használatával fennáll annak az esélye, hogy egyes jellegzetességleírók jobban működnek az adott kulcspontdetektálóval, mint mások. Az elfogultság elkerülése érdekében választottuk a véletlen kulcspontkiválasztást. Természetesen az is előfordulhat, hogy valamelyik jellegzetességleíró pont a véletlenszerűen kiválasztott kulcspontok esetében tesz szert előnyre, de ezzel a módszerrel tudjuk a legjobban elkerülni az efféle eseteket.

A kiválasztott kulcspontokra kiszámítjuk a jellegzetességleírókat. Amikor főkomponens analízist használunk, akkor az összes pontra kiszámítjuk a jellegzetességleírókat, hogy megkapjuk a transzformációt. Ahhoz, hogy ki tudjuk számítani a jellegzetességleírókat be kell állítanunk az algoritmushoz szükséges sugár paramétert, amely meghatározza, hogy egy pont körül mekkora környezetet vizsgálunk. A paraméter beállítása azonban nem triviális feladat. A tapasztalataink azt mutatják, hogy különböző típusú felhők esetén eltérő a sugár optimális értéke. Sőt, egyes típusú felhők esetében azt tapasztaltuk, hogy minél nagyobb a sugár, annál jobb a jellegzetességleíró leíróképessége. Viszont a sugár növelésével a jellegzetességleírók kiszámításának futási ideje növekszik. Alkalmazásokban történő felhasználás esetén meg kell határoznunk a paraméterek egy olyan beállítását, amelyek még elfogadható futási időt eredményeznek. A mérésekhez számunkra a legfontosabb az volt, hogy egyenlő feltételeket biztosítsunk a jellegzetességleíróknak, ezért a jellegzetességleírók kiszámításához a sugarat egy konstans értékre állítottuk be minden esetben.

A következő lépésben $Kd$-fát építettünk a felhők kulcspontjainak jellegzetességleíróira a $k$-legközelebbi szomszédok ($k$-NN [7]) gyors megtalálása érdekében. A pontmegfeleltetések elkészítéséhez az NNDR [8] módszert alkalmaztuk [36]. A lényege, hogy az egyik pontfelhő kulcspontjaihoz tartozó jellegzetességleíróknak megkeressük az első és második legközelebbi szomszédját a másik felhő kulcspontjaihoz tartozó jellegzetességleírók között. Megnézzük azt, hogy első és második legközelebbi szomszéd hogyan aránylanak egymáshoz, azaz kiszámítjuk a két távolság hányadosát. Ha a hányados kisebb, mint

[6] receiver operating characteristic curve
[7] $k$-nearest neighbor
[8] nearest neighbor distance ratio

egy megadott $\tau$ határérték, akkor a jellegzetességleírónak és a legközelebbi szomszédjának megfelelő pontok egy megfeleltetést fognak alkotni. A határérték használatával arra törekszünk, hogy csak azokat a megfeleltetéseket vegyük számításba, amelyek nagyobb eséllyel lehetnek helyes megfeleltetések.

A helyes megfeleltetések meghatározásához a kiértékeléshez használt adathalmaz biztosan helyes [9] transzformációt alkalmazzuk a két felhő egymásra illesztésére. Az illesztés után megnézhetjük, hogy a megfeleltetések pontjai az euklideszi térben milyen távolságra vannak egymástól. Ha egy megfeleltetés két pontja közötti távolság kisebb, mint $\varepsilon$, akkor azt mondjuk, hogy a megfeleltetés helyes. Vegyük észre, hogy az $\varepsilon$ paraméter értéke alapvetően befolyásolja, hogy mely megfeleltetéseket tekintünk helyesnek. Ha túl magas értéket választunk, akkor akár minden megfeleltetés is helyes lehet, ha pedig túl alacsony értéket, akkor akár egy megfeleltetés sem (mivel valós adathalmazok esetében nem fordul elő az, hogy összeillesztés után pontosan egymásra kerüljön két pont).

Ha meghatároztuk, hogy a megfeleltetéseink közül melyek a helyes megfeleltetések, akkor kiszámíthatjuk a *prcision* és *recall* értékeket. Jelöljük $TP$-vel a helyes megfeleltetések számát, $FP$-vel pedig a téves megfeleltetések számát. Az összes lehetséges helyes megfeleltetést jelölje $GT$. Ekkor a *precision* és *recall* értékek kiszámíthatóak a következő módon:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{GT}$$

A $precision - recall$ görbe (továbbiakban PRC) meghatározásához kiszámítjuk a *precision* ás *recall* értékeket különböző $\tau$ értékekre. A $\tau$ az értékeit a $[0.5, 1]$ intervallumon veszi fel 0.1-es lépésekkel. Ha $\tau = 0.5$ az azt jelenti, hogy a második legközelebbi szomszéd kétszer akkor távolságra van, mint az első. Ez csak nagyon ritka esetekben fordulhat elő (valós adatoknál szinte sosem), azaz nincs értelme a $\tau < 0.5$ értékeknek, mivel ilyenkor egy megfeleltetést sem találunk. Ahhoz, hogy a PRC által megmutatott eredményeket egy kompaktabb formában is meg tudjuk vizsgálni kiszámítjuk a görbe alatti területet (AUC [10]). Az AUC maximális értéke 1, és minél nagyobb annál jobb egy jellegzetességleíró leíróképessége.

[9] ground truth
[10] area under curve

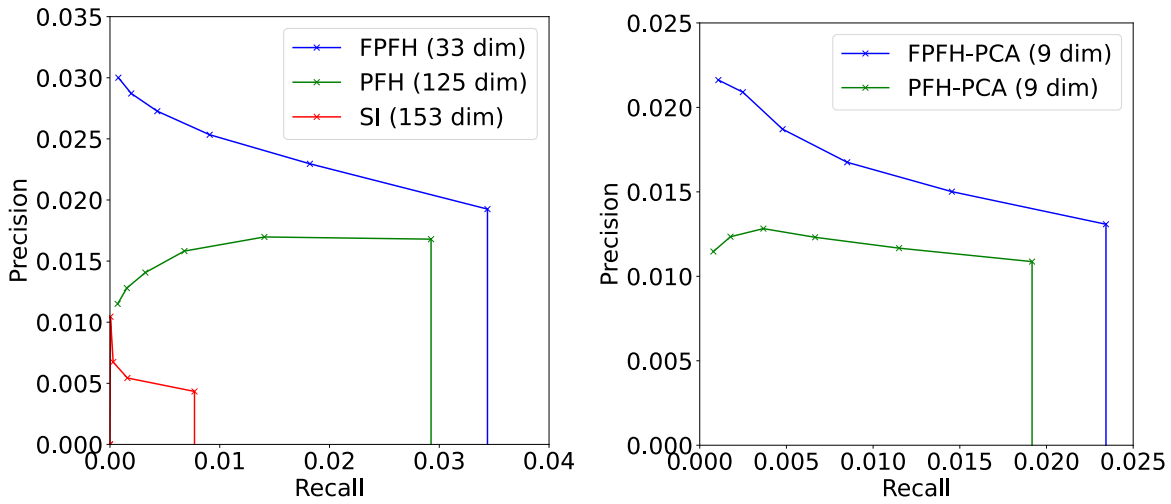
### 3.2.2. Adathalmaz

A jellegzetességleírók leíróképességének kiértékeléshez csökkentett dimenziószámmal a "7-Scenes RGB-D" adathalmazok *redkitchen* felvételeit használtuk [56]. Az adathalmaz egy szobáról (konyha) tartalmaz felvételeket, és minden felvétel $\sim$ 250.000 pontot tartalmaz. A kiértékelés előtt az adathalmazon előfeldolgozási lépéseket végeztünk. Voxelrács mintavételezést használva alulmintavételeztük a pontfelhőt, 0.01-es voxelmérettel. Az alulmintavételezés eredményeképpen az egyes pontfelhő felvételek pontjainak a száma $\sim$ 100.000-re csökkent. Ezután kiszűrtük a kiugró pontokat sugár alapú kiugróérték eltávolító módszerrel. Egy pont akkor került eltávolításra, ha kevesebb mint 10 szomszédja volt 0.04 sugarú környezetében. A kiugró értékek kiszűréséhez használt paramétereket vizuálisan ellenőriztük. Mivel a célunk a jellegzetességleírók kiértékelése volt, és nem egy automatizált alkalmazás létrehozása ezért az adathalmazok ilyen módú tisztítása megfelelő volt számunkra. A kiugró értékek kiszűrése $\sim$ 100 pontot távolított el felvételenként.

Az utolsó lépés, amit előfeldolgozásként végeztünk a normálvektorok becslése volt, mivel az szükséges a jellegzetességleírók kiszámításához. A normálvektor becsléshez 0.04-es sugarat használtunk. A jellegzetességleírók számára pedig a sugarat 0.06-ra állítottuk be minden esetben. A kiértékeléshez 10 egymással átfedő felhőt választottunk az adathalmazból. A kulcspontok száma pedig minden felhő esetében 5000 volt. A véletlenszerű kulcspontkiválasztás miatt minden egymással átfedő pontfelhő pár esetében a kiértékelést tízszer futtattuk le, és az eredmények átlagát tekintettük. A későbbi ábrákon ezeket az aggregált eredményeket láthatjuk.

### 3.2.3. Eredmények

Elsősorban kiértékeltük a jellegzetességleírók leíróképességét az alapértelmezetten megadott dimenziószámmal. Alapértelmezett értéknek azt tekintettük, amelyet a szerzők a munkájukban megadtak vagy az általuk készített implementációban feltüntettek. Az FPFH esetében a cikkben bemutatott dimenziószám és az implementációban szereplő eltér egymástól és nem találtunk egyértelmű utalást arra, hogy melyik a javasolt. Mivel a későbbi munkákban mindig az implementációban használt dimenziószám (33) szerepelt, ezért mi is azt használtuk. A 3.1. ábra (bal) három jellegzetességleíró $precision - recall$ görbéjét mutatja: Point Feature Histogram (PFH), Fast Point Feature Histogram (FPFH) és Spin Image (SI). A kiértékelésünk alapján kijelenthető, hogy a Spin Image jellegzetességleíró teljesítménye jelentősen elmarad a másik kettőtől. Érdemes megjegyezni, hogy a Spin Image jellegzetességleíró által készített jellegzetességvektorok rendelkeznek a legnagyobb dimenziószámmal (153). Az FPFH pontossága alacsony $\tau$ értékek esetében sokkal



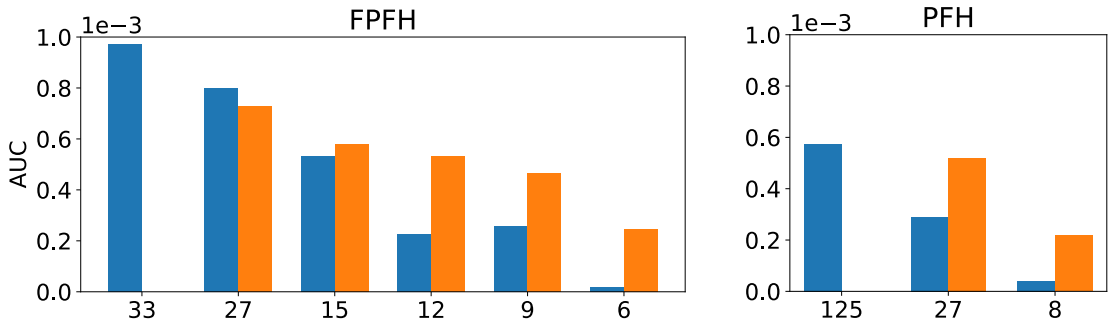

3.1. ábra: Bal oldalon: 3 jellegzetességleíró $precision - recall$ görbéje: FPFH, PFH, SI. Jobb oldalon: a PFH és FPFH jellegzetességleírók $precision-recall$ görbéje, 9 dimenzióra csökkentve a főkomponens analízis módszerrel.

jobb mindkét módszertől, azonban ahogy nő a $\tau$ értéke úgy a különbség az FPFH és a PFH között csökken. Azonban magas $\tau$ érték esetében is az FPFH *recall* értéke magasabb a PFH *recall* értékétől, ami azt jelenti, hogy több helyes megfeleltetést képes megtalálni.

Mivel a Spin Image jellegzetességleíró teljesítménye ilyen mértékben elmaradt a másik kettőtől, úgy döntöttünk, hogy a következő lépésekben csak az FPFH és PFH jellegzetességleírókat vizsgáljuk. A 3.1. ábra (jobb) az FPFH és PFH jellegzetességleírók görbéit mutatják, PCA-val 9 dimenzióra csökkentve. Ezt a mérést azért készítettük, hogy megnézzük a leíróképességek közötti különbséget csökkentett dimenziószámmal is. Az ábra alapján látható, hogy csökkentett dimenziószámok esetén is a különbség a két jellegzetességleíró között továbbra is megmarad.

A 3.2. ábra a PFH és FPFH jellegzetességleírók AUC értékeit mutatja különböző dimenziószámokkal. A bal oldali oszlopok (kék) a jellegzetességleírók belső paramétereivel történő dimenziócsökkenést mutatják. Az első oszlop a jellegzetességleírók eredeti dimenziójával vett leíróképességét mutatja mindkét esetben. A jobb oszlopok mutatják a PCA-val történő dimenziócsökkentett jellegzetességleírók leíróképességét. A PCA-t az eredeti dimenziószámmal kiszámolt jellegzetességleírókra alkalmaztuk. Az FPFH esetében, ha az algoritmus által mindhárom kiszámolt jellemző esetében egyforma kosarazást választunk, akkor a következő dimenziószámú jellegzetességvektorokat kapjuk: 33, 27, 15, 12, 9 és 6.

A 3.2. ábra (bal) alapján az FPFH esetében 27 dimenziónál még jobb eredményt kapunk, ha a belső paraméterekkel csökkentjük a dimenziószámot. Azonban, ha 15 di-




3.2. ábra: Az FPFH (bal ábra) és PFH (jobb ábra) jellegzetességleírók görbe alatti területének értékei (AUC) különböző dimenziószámmal. A bal oldali oszlopok (kék) esetében a dimenziószámok a jellegzetességleírók belső paramétereinek módosításával lettek csökkentve, míg a jobb oldali oszlopok (sárga) főkomponens analízis használatával.

menzió alá megyünk, akkor a PCA-val jobb eredményt kapunk. 15-ről 12-re lépés során a leíróképesség PCA-t használva nem csökken nagy mértékben, azonban a belső paraméteres dimenziócsökkentésnél itt már jelentős esést láthatunk. 6 dimenzió esetében már a PCA-val csökkentett jellegzetességvektor teljesítménye is jelentősen rosszabb, mint korábban. A PCA esetében a 15, 12 és 9 dimenziós jellegzetességvektorok között nincs jelentős különbség. Azonban vegyük észre, hogy az eredeti leíróhoz képest a leíróképesség felére csökkent. A következtetést levonva, 15 dimenzió fölött érdemes a belső paraméterek beállításával csökkenteni a dimenziószámot, azonban, ha 15 vagy az alatti dimenziót szeretnénk, akkor a PCA-val jobb eredményeket kapunk.

Jegyezzük meg, hogy a PCA használata több időbe kerül, mivel ebben az esetben a jellegzetességleírót az eredeti paraméterivel ki kell számolni, és utána tudjuk transzformálni az adatot. Ezért tehát, ha a futási idő kritikus a jellegzetességleíró alkalmazása során, akkor még 15 dimenzió esetében is érdemes lehet a belső paraméterek módosításával csökkenteni a dimenziószámot.

A PFH jellegzetességleíró esetében a belső paraméterek módosításával 3 különböző dimenziószámot vizsgáltunk meg: 125, 27 és 8 (mivel csak köbszámokat adunk meg, ezért egyedül a 64 maradt ki). A 3.2. ábra (jobb) a PFH jellegzetesség leíróképességét mutatja a 3 különböző dimenziószámmal. Először is vegyük észre, hogy az eredeti dimenziószámmal kiszámolt PFH jellegzetességleírók leíróképessége az FPFH 15 dimenziót használó változatával egyenlő (mindkét dimenziócsökkentő módszer esetében). A PFH esetében 27-es dimenziószámnál már jelentősen jobban teljesít a PCA-val csökkentett változat, mint a belső paraméterek módosításával. Érdemes megfigyelni, hogy a PCA-val 27 dimenzióra lecsökkentett változat szinte azonos leíróképességgel rendelkezik, mint az eredeti. Ez azért

történhet, mert az eredeti PFH-nak 125 dimenzióval a legtöbb eleme 0 értéket vesz fel. A vizsgálataink alapján az FPFH minden esetben jobb eredményt ad, mint a PFH ha ugyanannyi dimenziószámot használunk a két jellegzetességleíró esetében. Mivel a PFH kiszámítása költségesebb, ezért úgy látjuk, hogy a PFH használata nem előnyös.

## 3.3. Összegzés

A jellegzetességleírók tárolása és legközelebbi szomszédaik megkeresése egy nehéz feladat. A jelenleg létező jellegzetességleírók dimenziószáma olyan magas, hogy az indexstruktúrák rosszabbul teljesíthetnek, mint a szekvenciális szkennelés. Ebben a munkában megvizsgáltuk hogyan változik a jellegzetességleírók leíróképessége dimenziócsökkentés esetén. A dimenziócsökkentésre két módszert használtunk: a jellegzetességleírók belső paramétereinek a módosítását és a széles körben használt főkomponens analízist. A kiértékelésünk alapján az FPFH esetében egy bizonyos dimenziószám alatt a PCA használatával jobb eredményeket kapunk, PFH esetében pedig minden dimenziószámnál. Azt a következtetést is levonhatjuk, hogy az FPFH leíróképessége jelentősen nagyobb, mint a Spin Image és PFH leíróké.

Jövőbeli terveink között szerepel más dimenziócsökkentő módszerek hatásának a vizsgálata a jellegzetességleírókra, és a közelmúltban megjelent jellegzetességleírók bevonása a vizsgálatba.

# 4. fejezet

# 3D pontfelhők jellegzetességleíróinak kvantilis-alapú binarizációja

## 4.1. Bevezetés

A 3D pontfelhő adatokon végzett feldolgozási folyamatok során gyakran használunk lokális jellegzetességleírókat. Ahogy azt a korábbi fejezetben is kifejtettem, a leggyakoribb művelet, amelyet a jellegzetességleírókon alkalmaznak az a legközelebbi szomszéd keresések. A probléma ezekkel a klasszikus lokális jellegzetességleírókkal, hogy valós értékűek és sok dimenziósak (a SHOT jellegzetességleíró által készített vektornak 352 dimenziója van), ezért sok helyet foglalnak és a legközelebbi szomszéd keresések lassúk lehetnek valós idejű alkalmazásokhoz. Az olyan alkalmazási területeken, mint a mobil eszközök esetén egyre népszerűbbé váló kiterjesztett valóság alkalmazások fontos a valós idejű futási sebesség. Ahhoz, hogy a legközelebbi szomszéd keresések költségét (futási idő és tárhely) csökkenteni tudják, egyes munkák a leírók dimenziócsökkentését használták [85] [5]. A dimenziócsökkentéssel lehetséges úgy csökkenteni a jellegzetességvektorok méretét, hogy a leíróképességük ne csökkenjen jelentősen. Egy másik módszer a probléma megoldására a valós értékű jellegzetességleírók átalakítása bináris értékű jellegzetességleíróvá. Egy bináris jellegzetességleíró sokkal kevesebb helyet foglal, mint a valós értékűek. Továbbá, a bitvektorok közötti Hamming-távolság bitműveletekkel elvégezhető, így a legközelebbi szomszéd keresések ideje jelentősen lecsökken [70].

A binarizációs módszerek célja a valós értékű jellegzetességvektorok átalakítása bitvektorrá, úgy, hogy a jellegzetességleíró leíróképessége számottevően ne csökkenjen. Általában, egy vagy több bittel helyettesítik az eredeti jellegzetességleírók minden valós értékű elemét. Vannak olyan binarizációs módszerek, amelyek általánosak és bármely va-

lós értékű leírót képesek binarizálni, viszont olyan módszerek is, amelyek kihasználják az egyes jellegzetességleírók kiszámításának speciális tulajdonságait. Léteznek továbbá olyan jellegzetességleírók, amelyek azonnal bináris jellegzetességvektort készítenek, ezeket önálló [1] bináris jellegzetességleírónak hívjuk. A bináris jellegzetességvektorokkal jelentősen felgyorsítható a pontfelhő alapú lokalizáció és az objektumfelismerés. Prakhya és társai [70] munkája alapján a binarizált jellegzetességvektorok használatával a pontok közötti megfeleltetések keresését ötöd annyi idő alatt el lehet végezni.

Ebben a munkában bemutatunk egy binarizációs módszert, amely az eredeti jellegzetességleírók dimenzióinak eloszlásainak kvantilisei alapján csoportokat hoz létre. Ezeket a csoportokat bitsorozatokkal reprezentálja, és ezeket összefűzve elkészíti a bináris jellegzetességvektorokat. A javasolt módszerünk neve QBB [2], amely egy általános, paraméter nélküli binarizációs módszer. Összehasonlítjuk az elérhető binarizációs módszerekkel és az önálló bináris jellegzetességleírókkal, és megmutatjuk, hogy jobb eredményt tudunk elérni egy valós adathalmazon bemutatva. Bemutatjuk a QBB lehetséges variációit, amelyek egyes esetekben nagyobb leíróképességgel rendelkeznek, mint az eredeti jellegzetességleíró. A QBB módszer és a kiértékeléséhez használt kód teljes egészében nyilvánosan elérhetőek az interneten.

## 4.2. Kapcsolódó munkák

2015-ben megjelent a B-SHOT [70], az első módszer, amelynek célja valós értékű 3D jellegzetességleíró binarziációja volt. Azóta több munka is megjelent, amelyek szintén ezt a feladatot próbálták megoldani. Alapvetően a bináris jellegzetességleírókat két osztályba lehet sorolni ( 4.1. ábra). Az egyik fajta megközelítés, hogy már létező valós értékű leírót alakítsunk át binárissá. Ezen módszerek előnye, hogy a legtöbb esetben tetszőleges valós értékű jellegzetességleíróra lehet alkalmazni. Általában a binarizálás után egy gyengébb leíróképességű, de sokkal kevesebb helyet foglaló leírót kapunk. Ennek a megközelítésnek az egyik hátránya, hogy előbb ki kell számolni a valós értékű jellegzetességvektort, és utána át kell alakítani binárissá, ami plusz időt vesz igénybe.

A Prakhya és társai [70] által készített binarizációs módszer célja a SHOT valós értékű jellegzetességleíró binarizációja. Az algoritmus egyébként bármilyen valós értékű vektort képes binarizálni. A módszer két fontos paramétere a kódolási hossz ($L$) és a kódolási arány ($E_r$). Lényege, hogy a valós értékű vektor L darab egymás utáni értékét kiválasztva, megnézi, hogy az $L$ darab elem értékének az összegének mely elemek adják az $E_r$

[1] standalone
[2] Quantile-Based Binarization

százalékát. Ezen elemek értéke 1 lesz a bináris vektorban, míg a többié 0. Ez a módszer minden valós értékű elemet egy bináris értékkel helyettesít. A szerzők mérései alapján, a SHOT esetében a legjobb teljesítményt akkor nyújtotta az algoritmus, amikor a kódolási hossznak 4-et választottak, a kódolási arány értéke pedig 0.9 volt. Később összehasonlították a B-SHOT-ot az ugyanazzal a módszerrel elkészített B-FPFH és B-ROPS-al is [86]. Az eredményeik alapján, míg a B-SHOT leíróképessége közelít az eredeti leíróéhoz, addig a másik két binarizált leíró messze elmarad ettől. A szerzők elismerik, hogy bizonyos esetekben az információveszteség nagy lehet.

Később újabb munkák jelentek meg, amelyek a B-SHOT-hoz hasonlóan a SHOT valós értékű jellegzetességleírót binarizáltak [103] [91]. Lin és társai elkészítették a B-SHOT algoritmus egy általános változatát (Gray-SHOT), amelyben a kódolási hossz és az egyes dimenziókhoz használt bitek száma bemeneti paraméterré vált. Az egy dimenzió reprezentálásához használt bitek számának növelésével lehetővé vált kevesebb információveszteséggel binarizálni a valós értékű leírót, viszont így a tároláshoz szükséges tárhely is megnőtt. Több bit használata esetén bevezették a Gray-kódolást, hogy az egy dimenzióban levő egymás melletti csoportok Hamming-távolsága minden esetben 1 legyen. Méréseik alapján a legjobb eredményt akkor kapják, ha az kódolási hossz 352 és az egy dimenzió reprezentálásához használt bitek száma pedig 2. A CI-SHOT nagyon hasonló az előző megközelítéshez. Az algoritmus ugyanúgy kódolási hossz darabszámú dimenziót vizsgál egyszerre, és több bitet képes használni egy dimenzió reprezentálásához. A fő különbség a korábbi módszerekhez képest, hogy az algoritmus a Chebisev-egyenlőtlenség következtetései alapján dönti el, hogy az adott érték milyen kategóriába kerüljön. A paraméter vizsgálatok alapján ők 11-et választanak kódolási hossznak és 2 bitet egy dimenzió reprezentálásához. A szerzők mindkét esetben a binarizációs módszerüket csak a SHOT valós értékű leíróra készítették el.

Yang és társai kifejlesztettek egy valós értékű jellegzetességleírót, amelyet RCS-nek hívnak [90]. A munkájukban megadnak három módszert, amellyel a leíró átalakítható bináris leíróvá. Azonban ezek a módszerek, kisebb módosításokkal, tetszőleges valós értékű leíróra is alkalmazhatóak. Az első módszer a küszöbérték-alapú [3], melynek a lényege, hogy kiszámítanak egy határértéket, és egy dimenzió attól függőn kap 1-es vagy 0-ás bitet, hogy elérik-e ezt a határértéket. A cikkükben ők az adott dimenzió medián értékét vették határértékként. A második megközelítésük a vektorkvantálás, amikor egy dimenziót több bittel reprezentálnak, így $2^N$ kategóriát kapnak. A módszer előnye, hogy több információt képes tárolni, azonban több tárhelyet foglal és érzékenyebb a zajra. A harmadik módszer a

[3] thresholding

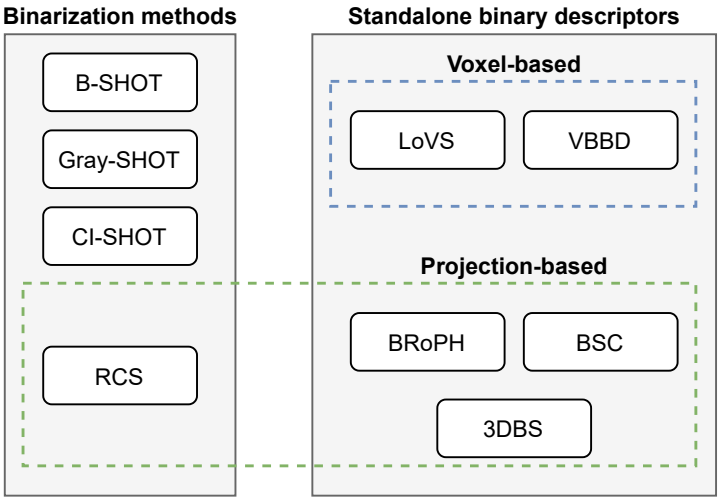
geometria-alapú. Ez kihasználja az RCS leíró algoritmusának egy speciális részét, ezért ez nem feltétlenül alkalmazható más módszereknél. Tetszőleges valós értékű vektort is képes binarizálni, de a geometriai megfontolások nélkül az eredménye elmaradhat a várttól. A lényege, hogy az egymás után következő valós értékeket összehasonlítja, és az összehasonlítás eredménye alapján kap egy bit értéket (az utolsó értéket pedig az elsővel hasonlítja össze, így minden valós dimenziónak megfog felelni egy bit). Az eredményeik alapján a legjobb binarizációs módszernek a vektorkvantálás bizonyult, főleg a 2 és 4 bitet használó változatok. Egyes esetekben a binarizált változat majdnem olyan jó eredményeket ért el mint maga a valós értékű leíró.

A bináris leírók másik nagy csoportjába tartoznak az önálló bináris jellegzetességleírók. Ezek olyan leírók, amelyek azonnal bináris leírót készítenek. Előnye az önálló módszereknek, hogy nincs szükség külön binarizációs lépésre. Hátránya a másik megközelítéshez képest, hogyha megjelenik egy új, minden szempontból jobb jellegzetességleíró, akkor a binarizációs módszerekkel azokból is készíthető bináris változat. Az önálló módszerek azonban elavulttá válhatnak. A jelenlegi önálló jellegzetességleírókat vizsgálva, két csoportra lehet őket osztani. Az első csoportba (voxelrács-alapú) tartoznak azok a módszerek, amelyek voxelrácsot készítenek a pont lokális környezetében, és minden voxelhez egy bitet rendelnek. A másik csoportba olyan módszerek tartoznak, amelyek levetítik a pontokat egy síkra vagy tengelyre, és az így kapott síkokat és értékeket dolgozzák fel a leírók kiszámításához.

A két legismertebb voxelrács-alapú önálló bináris jellegzetességleíró a LoVS (Local Voxelized Structure) [98], a másik pedig a VBBD (Voxel-Based Buffer-Weighted Binary Descriptor) [110]. A két módszer nagyon hasonló. Első lépésben mindkét módszer meghatároz egy lokális vonatkoztatási rendszert (LRF [4]) a kiválasztott pont körül, bár ezt különböző módszerrel teszik meg. Ezután az LRF-be transzformált felhőre voxelrácsot készítenek. A leírók annyi bitből állnak, ahány voxel van. A LoVS esetében akkor lesz egy bit értéke 1, ha a hozzá kapcsolódó voxelbe esik pont, egyébként pedig az értéke 0 lesz. A módszer paramétere a voxelek száma egy tengely mentén ($m$), ebből adódóan pedig a leíró hossza $m^3$ lesz. Minél több voxelt használunk, annál több információt tudunk tárolni, azonban túl sok voxel esetében már a zajt is megőrizhetjük. A szerzők a paraméterhangolás alapján az $m = 9$ beállítást ajánlják. A VBBD esetében ugyanúgy a voxelek száma határozza meg a bitek számát, azonban a bitek értékének kiszámítása komplexebb módszerrel történik. Egy voxel középpontjának $h$ sugarú környezete az adott voxel pufferrégiója. Minden voxel pufferrégiójára kiszámolják a Gauss-féle kernelsűrűsé-

[4] local reference frame

get. Egy bit értéke akkor lesz 1, ha a hozzá tartozó pufferrégió kernelsűrűsége nagyobb, mint az átlagos kernelsűrűség.




4.1. ábra: Binarizációs módszerek és önálló bináris jellegzetességleírók osztályozása. Az RCS egy önálló, valós értékű leíró, de a szerzők megadtak három binarizációs módszert is, amelyekkel binarizálható a leírójuk.

A projekció-alapú módszerekben közös, hogy a pont lokális környezete alapján lokális vonatkoztatási rendszert készítenek és ebbe transzformálják a szomszédos pontokat. A 3D Binary Signatures (3DBS [77]) a lokális környezetet nem sugár alapján határozza meg, hanem az $N$ legközelebbi szomszédság alapján, amelyeket különböző szögmegszorításokkal szűrnek meg. Ez azért fontos számukra, mert ebben a módszerben a szomszédok száma határozza meg a jellegzetességvektor hosszát. A módszer minden pont normálvektorát levetíti az $X$, $Y$ és $Z$ tengelyekre. Rendezett párokat alkot belőlük és összehasonlítja a pontpárok normálvektorainak tengelyekre vetített értékeit. Minden pontpár 3 bitet fog adni a jellegzetességvektorba, összesen pedig $3 * \frac{N*(N-1)}{2}$ bit hosszú lesz a vektor. A Binary Shape Context (BSC [83]) és a Binary Rotational Projection Histogram (BRoPH [80]) a lokális környezetben lévő pontokat az $XY$, $XZ$ és $YZ$ síkokra vetítik le. Az ötletük az, hogy a 3D pontok binarizációját visszavezetik kétdimenziós binarizációra. A BRoPH hasonló a RoPS [57] valós értékű leíróhoz. Az algoritmus $X$, $Y$ és $Z$ tengelyek körül $\theta$ szöggel elforgatja a lokális pontfelhőt és minden ilyen elforgatás után levetíti a pontokat a három síkra. Azután ezeket a kétdimenziós képfoltokat $L \times L$ darab ládára osztják. A ládákba eső pontokat megszámolják és átlagolják a mélység értéküket. Ezen ládák alapján készül el a bináris leíró. A BSC módszer elforgatás nélkül vetíti le a pontokat a három síkra, és ezeket ugyanúgy ládába rakja a pontok eloszlása és a mélységük alapján.

Összességében elmondható, hogy a már létező binarizációs módszerek túl speciálisak,

csak egy konkrét valós értékű jellegzetességleíró esetén működnek jól. A B-SHOT módszer esetében a szerzők is elismerik, hogy egyes esetekben az információveszteség nagy lehet.

## 4.3. A javasolt binarizációs módszer

### 4.3.1. Motiváció

Guo és társai [66] kiemelik, hogy a legtöbb jellegzetességleíró módszer hisztogramot használ a jellegzetességvektorok előállításához. Ezek között jelentős eltérések vannak, de közös bennük, hogy a hisztogramládákba eső értékek számai vagy arányai határozzák meg a jellegzetességvektorok elemeit.

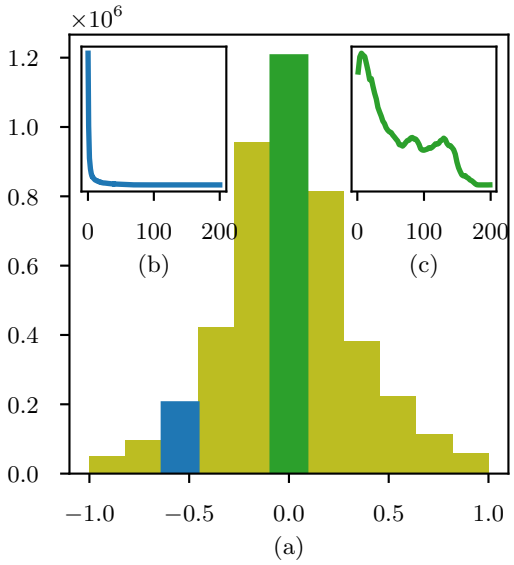
Ebben a bekezdésben a Fast Point Feature Histogram (FPFH) jellegzetességleíró felépítéséről lesz szó. Az FPFH módszert részletesen bemutatom az értekezés A. függelékében. Az FPFH jellegzetességleíró módszer alapjául három jellemző szolgál: $\theta$, $\alpha$ és $\phi$. Ezeket a jellemzőket a módszer által meghatározott pontpárokra számolja ki. Az FPFH esetében a három jellemző mindegyikére hisztogramot készítenek úgy, hogy minden jellemzőt 11 egyenlő részre osztanak (a 11-es felosztás az alapértelmezett beállítás a szerzők által készített PCL implementációban [52]). Ebből következik, hogy az FPFH $1-11.$, $12-22.$, $23-33$ dimenziói a $\theta, \alpha, \phi$ jellemzők által vannak meghatározva. Ha egyetlen pontpárra számolt jellemző érték sem esik egy adott intervallumba, akkor az intervallum által meghatározott hisztogramládának megfelelő koordinátaérték 0 lesz, ha mindegyik odaesik, akkor pedig 200 (az Open3D FPFH implementációja alapján [100]). Mivel a $\phi$ jellemző a legérdekesebb, ezért a 4.2. ábrán csak ezt mutattuk meg. Ha egy hisztogramládába kevés érték esik, akkor a jellegzetességvektor megfelelő eleme az esetek többségében 0 lesz és csak ritkán tér el ettől. Ez látszik a Fig. 4.2 (b) részábráján. Ha egy hisztogramládába sok érték esik, akkor e dimenzió mentén a koordinátaértékek "érdekesebb" sűrűségfüggvényt adhatnak, mint az előző esetben, amelyet a (c) részábra illusztrál. Tapasztalataink alapján általánosságban annyit lehet mondani a különböző jellegzetességleíró módszerekről, hogy különböző okok miatt az a tipikus, hogy sok dimenzió mentén lesz általában sok 0 érték és csak kevésnél lesz a koordinátaértékek sűrűségfüggvénye igazán "érdekes".

Binarizációs módszerünknél arra fogunk törekedni, hogy a jellegzetességleírók koordinátaértékeit egy dimenzió mentén csoportokra bontsuk. Egy csoportot alkotó értékek között nem fogunk különbséget tenni a későbbiekben. A csoportosításnak tehát úgy kell megtörténni, hogy minél jobban meg tudjuk őrizni az eredeti jellegzetességleíró módszer leíróképességét. Egy dimenzió mentén a csoportosításnál az lesz az egyik cél, hogy a csoportokba nagyjából azonos mennyiségű elem kerüljön, mivel azt szeretnénk, hogy

mindegyik csoport ugyanolyan fontos legyen. A csoportok határainak kijelölése kvantilisek alapján történik. Feltételezve, hogy pontfelhő-típusonként egy rögzített jellegzetességleíró módszerre egy dimenzió mentén a koordinátaértékek hasonló sűrűségfüggvényt határoznak meg, kellően nagy tanítóhalmazzal előre meg lehet határozni a csoportok határait. A legfontosabb kérdés, hogy hány csoportot érdemes venni egy dimenzió mentén. Ezt részletesebben ki fogjuk fejteni az alábbiakban. Most csak annyit jegyzünk meg, hogy

- Az FPFH azon dimenzióinál, amelyek mentén sok a 0 koordinátaérték, előfordulhat, hogy még a medián érték is (vagy akár még a harmadik kvartilis is) 0 lesz, ami azt jelenti, hogy nem fogunk tudni két csoportnál többet definiálni és ebben az esetben még az se fog teljesülni feltétlenül, hogy a csoportokba azonos mennyiségű koordinátaértékek kerülnek.

- Ha sok csoportra szeretnénk bontani egy dimenzió mentén az értékeket, akkor igazából az "érdekesebb" dimenzióknál is előfordulhat, hogy egy csoport határpontjai már "túlságosan" közel lesznek egymáshoz, ezért megadunk egy leállási feltételt.




4.2. ábra: Az FPFH jellegzetességleíró alapjául szolgáló $\phi$ jellemzőnek a vizsgálata: (a) a jellemző hisztogramja (b) az FPFH 25. dimenziójához tartozó empirikus sűrűségfüggvény, amelyhez tartozó hisztogramláda kék színnel lett jelölve (c) az FPFH 28. dimenziójához tartozó empirikus sűrűségfüggvény, amelyhez tartozó hisztogramláda zöld színnel lett jelölve.

A csoportokat ezután olyan bitsorozatokkal fogjuk reprezentálni, amelyek a csoportok közti közelséget valamilyen szinten figyelembe veszik. A teljes jellegzetességleíróra a

binarizáció a dimenziók mentén kapott bitsorozatok összefűzésével valósul meg. A módszerünket kvantilis-alapú binarizációnak [5] (QBB) hívjuk.

### 4.3.2. A kvantilis-alapú binarizáció (QBB) bemutatása

Ebben a részben szeretném részletesen bemutatni az általunk javasolt binarizációs módszert. Először is vezessünk be jelöléseket. Legyen $D$ a jellegzetességleírók dimenziószáma (a jellegzetességvektorok hossza). Legyen $X_1, \ldots, X_n$ $n$ darab jellegzetességvektor, az $i$. jellegzetességvektor ($i = 1, \ldots, n$) $d$. koordinátáját jelölje $X_i^{(d)}$ és a jellegzetességvektorok $d$. koordinátaértékeinek listáját pedig: $\mathbf{X}^{(d)} = (X_1^{(d)}, \ldots, X_n^{(d)})$ ($d = 1, \ldots, D$). Legyen $\mathbf{X}_{\min}^{(d)} = \min\{X_1^{(d)}, \ldots, X_n^{(d)}\}$ és $\mathbf{X}_{\max}^{(d)} = \max\{X_1^{(d)}, \ldots, X_n^{(d)}\}$.

Legyen $Q^{(d)}(p)$ az empirikus kvantilisfüggvénye $\mathbf{X}^{(d)}$-nek ($0 \leq p \leq 1$). Legyen továbbá $(f(k) \mid k = 0, \ldots, K)$, egy lista, ahol $f(k)$ függvénye $k$-nak és a lista első eleme f(0), második eleme f(1) stb. Jelölje $round(x)$ a legközelebbi egész függvényt, amely megadja az $x \in \mathbb{R}$ értékhez legközelebbi egész számot. A továbbiakban az egyszerűség kedvéért feltesszük, hogy kettő hatvány számú csoportot fogunk képezni.

Nem feltétlen igaz az, hogy több csoportot definiálva a dimenziók mentén, jobban megőrizzük az adott jellegzetességleírónak a leíróképességét, mivel a nagyon hasonló értékek külön csoportba rakása inkább hátránnyal jár, mint előnyökkel. (pl. ha a sűrűségfüggvénynek van egy viszonylag keskeny csúcsa valahol, azt nem szeretnénk ketté vágni). A csoportok számának megválasztása némileg kapcsolódik a hisztogramládák számának megválasztásához, amikor egy eloszlást hisztogrammal szeretnénk közelíteni. Ott is igaz, hogy elegendő hisztogramládának kell lenni ahhoz, hogy egy hisztogramláda ne rejtsen el semmi lényeges információt az eloszlás alakjáról, de hagyja figyelmen kívül a véletlenszerű fluktuáció miatti részleteket [101].

Ha el szeretnénk kerülni a fent említett értékek két külön csoportba sorolását, akkor figyelni kell arra, hogy egy csoport határa ne kerülhessen akárhová, hanem csak valamilyen egységben mérve. Az egységet a koordinátaértékek hisztogramjának egyenletes hisztogramláda szélessége adja, amelynek meghatározására több módszer létezik. A népszerűsége okán a Freedman-Diaconis szabályt használtuk [16] azzal a kiegészítéssel, hogy megadunk egy korlátot, amely alá nem mehet a hisztogram ládaméret a számítás felgyorsítása érdekében. Jelölje $bw$ a hisztogramláda szélességet, ekkor:

$$bw = \max\left(2 \cdot \frac{IQR\left(\mathbf{X}^{(d)}\right)}{\sqrt[3]{n}}, \frac{\mathbf{X}_{\max}^{(d)} - \mathbf{X}_{\min}^{(d)}}{10^4}\right) \tag{4.1}$$

[5] quantile-based binarization

ahol $IQR\left(\mathbf{X}^{(d)}\right)$ az interkvartilis terjedelem, azaz $Q^{(d)}(0.75) - Q^{(d)}(0.25)$.

Az 3. algoritmus, a fent leírtakkal összhangban, meghatározza egy adott $d$. dimenzióra egyrészt, hogy melyik az a maximális csoportszám, amelyre még (várhatóan) figyelmen kívül lesznek hagyva a véletlenszerű fluktuáció miatti részletek. Másrészt kiszámolja a maximális csoportszámig minden csoportra a határokat. Az algoritmus leírásában a $\gamma$ a csoportszámot jelöli. Egy adott $\gamma$ csoportszámra ez azt jelenti, hogy a jellegzetességvektorok sorba rendezett $d$. koordinátaértékeit $\gamma$ darab egyenlő részre osztjuk, azaz a $\gamma$-kvantiliseket határozzuk meg. Azzal a kiegészítéssel, hogy csak oda lehet tenni a határokat, ahol az egységhatárok vannak. Tulajdonképpen csak addig fogunk menni az algoritmusban megadott ciklussal, ameddig nem lesz olyan csoport, amelyhez tartozó intervallum hossza 0 lesz. Tehát összességében vissza fog térni egy $\mathcal{G}_d$ halmazzal, amelynek egy eleme egy adott csoportszámot és a hozzátartozó határokat reprezentálja. Azért nem csak a maximális csoportszámhoz tartozó határokkal tér vissza az algoritmus, mert adott esetben kapacitáskorlátot is szeretnénk figyelembe venni. A kapacitáskorlátról a 4.3.3. fejezetben lesz bővebben szó. Az algoritmusban az *endps* tartalmazza a csoporthatárok végpontjait, a $\delta$ a szomszédos csoportok határainak egymáshoz való távolságának minimumát. Ha túl kis intervallumot fed le egy csoport, akkor a ládaméret korlát miatt a távolsága a szomszéd végponttól 0 lesz, azaz az algoritmus leáll.

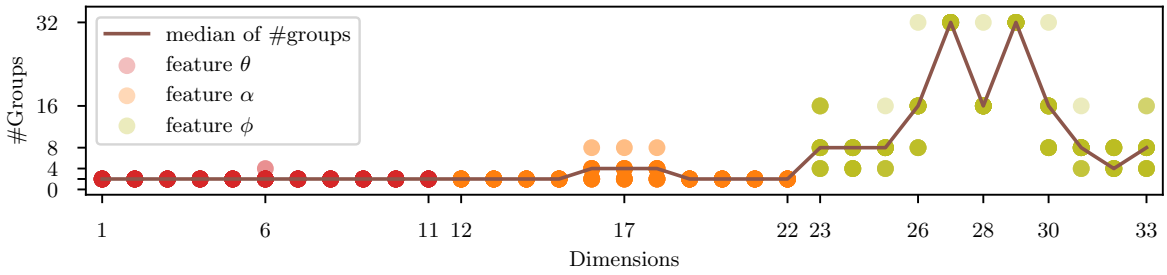
**Algoritmus 3** Csoportok meghatározása a $d$ dimenzió számára

INPUT: $\mathbf{X}^{(d)}, bw$ ▷ a $bw$ definícióját a 4.1. képletben adtuk meg
OUTPUT: a $\mathcal{G}_d$ halmaz, amely tartalmazza a csoportok számát és határaikat, a legnagyobb megengedett csoportszámig

$\mathcal{G}_d \leftarrow \{\}$
$\gamma \leftarrow 2$
**repeat**
  $endps \leftarrow (\text{round}(Q^{(d)}(k/\gamma)/bw) \cdot bw \mid k = 0, \ldots, \gamma)$
  $\delta \leftarrow \min(\{endps[k+1] - endps[k] \mid k \in [0, \gamma - 1]\})$
  **if** $\delta > 0 \vee \gamma = 2$ **then**
    $\mathcal{G}_d \leftarrow \mathcal{G}_d \cup \{(\gamma, endps)\}$
  **end if**
  $\gamma \leftarrow 2 \cdot \gamma$
**until** $\delta > 0$
**return** $\mathcal{G}_d$

Az FPFH módszer esetén a dimenziók mentén adódó maximális csoportszámokat külön-külön meghatároztuk több pontfelhőre is, amelyet a 4.3. ábra mutat meg. Látszik, hogy mindegyik dimenzió esetén az 3. algoritmus nagyjából hasonló csoportszámot ad a különböző pontfelhők esetén. Emiatt előre meg lehet határozni egy olyan ésszerű csoportszámot a dimenziók mentén, amivel a későbbi pontfelhők esetén is nagyjából

hasonlóan lehet megőrizni az FPFH leíróképességét.

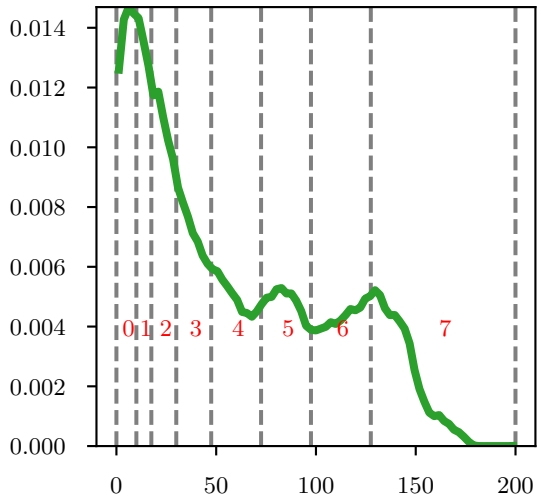



4.3. ábra: Az FPFH jellegzetességleíró dimenzióihoz tartozó csoportszámok. A különböző dimenziók aszerint vannak színezve, hogy melyik jellemzőhöz tartoznak a három közül. Az ábra megmutatja, hogy miért a harmadik jellemző ($\phi$) a legérdekesebb a jellegzetességleíró szempontjából.

Többféle módon el lehet járni a csoportokra adódó bitsorozatok meghatározásánál, azaz a kódolással. Amit mindenképpen érdemes figyelembe venni, hogy a binarizált jellegzetességleírók között a hasonlóság a Hamming-távolság (illetve annak módosított változata) alapján lesz meghatározva. Célunk az, hogy a binarizált leírók közötti távolság, minél jobban tükrözze a valós-értékű leírók közötti távolságot. Azt is szeretnénk, hogy az információt minél kevesebb biten kelljen tárolni. $\gamma$ csoport kódolásához szükségünk van minimum $\log_2 \gamma$ darab bitre. Pontosság szempontjából a legjobb az lenne, hogy ha két csoport között éppen $k$ darab csoport van, akkor a Hamming-távolságuk $k+1$, de azt azért mindenképpen szeretnénk, hogy a szomszédos csoportok között 1 legyen a távolság. Ha csak ennyit szeretnénk biztosítani, akkor erre alkalmas a Gray-kód [8] és ez $\log_2(\gamma)$ bitet használ. Ha viszont szeretnénk a csoportok közti távolságot pontosan tartani, akkor meggondolható, hogy ehhez $\gamma - 1$ bitre lesz szükségünk és az $M_0, M_1, \ldots, M_{\gamma-1}$ Mersenne-számok [9] bináris alakjára (a bináris számrendszerben kifejezett Mersenne-számokra). Erre innentől Mersenne-kódként hivatkozunk. A 4.4. ábra és a 4.1. táblázat illusztrációként szolgál az előbbi kódolásokra.

### 4.3.3. Kapacitáskorlát

Hátra van még annak vizsgálata, hogy hogyan kell eljárnunk abban az esetben, hogy ha kapacitáskorlát is van, azaz egy binarizált jellegzetességleíró tárigénye legfeljebb $C$ bit lehet. Ebben az esetben mindenképpen a Gray-kódot választjuk, mivel egyrészt ez a lehető legtömörebb bináris reprezentálása az információnak, másrészt a tapasztalatok azt mutatják, hogy általában így is hasonló pontosságot lehet elérni, mint a Mersenne-kóddal. A különböző dimenziókra a csoportszámot úgy szeretnénk meghatározni, hogy mindegyik




4.4. ábra: Az FPFH jellegzetességleíró 28. dimenziójának sűrűségfüggvénye a meghatározott csoportokkal. A 4.1. táblázat az ábrán piros színnel jelölt csoportindexeknek megfelelően foglalja össze, hogy melyek lesznek a Gray-kód és a Mersenne-kód szerinti bitsorozatok.

| Csoport index | Gray-kód | Mersenne-kód |
|---|---|---|
| 0 | 000 | 0000000 |
| 1 | 001 | 0000001 |
| 2 | 011 | 0000011 |
| 3 | 010 | 0000111 |
| 4 | 110 | 0001111 |
| 5 | 111 | 0011111 |
| 6 | 101 | 0111111 |
| 7 | 100 | 1111111 |

4.1. táblázat: Gray és Mersenne-kódok 8 csoport esetén. Két szomszédos Gray-kód esetén a Hamming-távolság 1. A Mersenne-kód esetén, minden kód között akkora a Hamming-távolság, mint amekkora az általuk reprezentált csoportok között.

dimenzió legalább 1 bitet kapjon. Jelölje $C$ a kapacitáskorlát szerint megengedhető bitek számát, $D$ pedig az eredeti jellegzetességvektor hosszát (azaz $D$ számú valós értékű elemből áll). Feltehető, hogy $D \leq C$ (mert egyébként a jellegzetességleírónak a belső paramétereit kellene változtatni, pl. hisztogram-alapú módszer esetén a hisztogramládák számát lehetne csökkenteni). A maradék $C - D$ számú bitet az alapján szeretnénk elosztani a dimenziók között, hogy az egyes dimenziók mentén mennyi volt a maximális csoportszám. Úgy is el lehet képzelni, hogy a maximális csoportszám alapján a dimenziók leadják az igényüket a maradék bitekre és az igényeket minél jobban tükrözve osztjuk szét közöttük ezeket. Ehhez jelölje $r_d$ a $d$. dimenzió által igényelt bitek számát, ami az alábbival egyezik meg:

$$r_d = \log_2(\max\{\gamma \mid (\gamma, endps) \in \mathcal{G}_d\}). \tag{4.2}$$

Továbbá legyen az összes igényelt bit:

$$R = \sum_{d=1}^{D} r_d. \tag{4.3}$$

Ha $R \leq C$, akkor mindenki megkapja az igényelt biteket, hiszen belefér a kapacitáskorlátba. Tehát az érdekes eset az $R > C$. Ekkor a $d$. dimenzió $l_d$ bitet fog kapni:

$$l_d = 1 + \lfloor (C - D) \cdot (r_d - 1)/(R - D) \rfloor \tag{4.4}$$

mert legalább 1 bitet minden dimenzió kapni fog és a maradék $C - D$ biteket kell az igényelt bitek $(r_d - 1)/(R - D)$ súlyában elosztani. A $D \leq C < R$ teljesül a konstrukció miatt, így $l_d$ mindenképpen pozitív egész lesz.

## 4.4. Kiértékelés

A módszerünket összehasonlítottuk az általunk ismert 3D jellegzetességleíró binarizációs módszerekkel (B-SHOT [70], CI-SHOT [91], Gray-SHOT [103]) és önálló bináris jellegzetességleírókkal is (VBBD [110], LoVS [98]). A binarizációs módszereket azokra a leírókra alkalmaztuk, amit a szerzők is ajánlottak, így nem minden binarizációs módszert alkalmaztunk minden jellegzetességleíróra. A kiértékeléshez az Open3D-ben [100] és a Point Cloud Library-ben [52] elérhető valós értékű leírókat használtuk (FPFH, SHOT, RoPS, SI), a binarizációs módszereket (B-SHOT, CI-SHOT, Gray-SHOT) és az önálló bináris jellegzetességleírókat (VBBD, LoVS) pedig Python nyelven implementáltuk. Az implementációk és a kiértékelést végző kód nyilvánosan elérhető a GitHub forráskódmegosztó

| Jellegzetességleírók | Paraméterek | Méret |
|---|---|---|
| FPFH [42] | PCL-ben alapértelmezett [52] | 33 (f) |
| SHOT [62] | PCL-ben alapértelmezett | 352 (f) |
| Spin Image [94] | PCL-ben alapértelmezett | 153 (f) |
| RoPS [57] | PCL-ben alapértelmezett | 135 (f) |
| VBBD [110] | $g = 4, 7, 9$<br>$h = 4 * radius * g$ | $64, 343, 729$ (b) |
| LoVS [98] | $m = 4, 7, 9$ | $64, 343, 729$ (b) |
| Gray-SHOT [103] | $L = 352, N = 2$ | 704 (b) |
| CI-SHOT [91] | $L = 11, N = 2$ | 704 (b) |
| B-FPFH | $L = 4, 11$ | 33 (b) |
| B-SHOT [70] | $L = 4$ | 352 (b) |

4.2. táblázat: Jellegzetességleírók és azok paraméterei. Minden jellegzetességleírót 0.06-os sugárral számoltunk ki. A valós és bináris értékeket (f) és (b) betűkkel jelöljük.

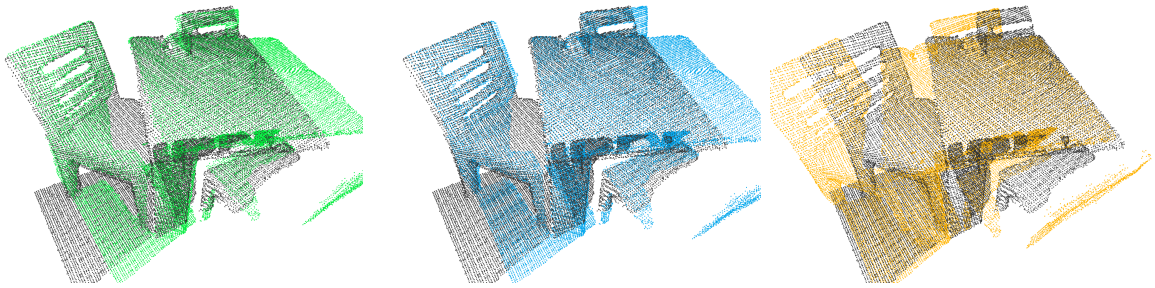
weboldalon [6].

A valós értékű és önálló bináris jellegzetességleírók fontos paramétere a sugár, amely alapján kiválasztjuk a jellemezni kívánt pont szomszédságát. Minél nagyobb a sugár, annál több információt tud magába foglalni egy leíró. A sugár mérete függ a felhő által reprezentált felületek méretéről is. Guo és társai [66] munkáját követve, ahhoz, hogy hasonló feltételeket biztosítsunk a különböző leíróknak, ugyanolyan méretű sugár paramétert használtunk minden esetben.

A kiértékeléshez a nyilvánosan elérhető 7-Scenes RGB-D Redkitchen [56] és a Redwood Livingroom [65] adathalmazokat használtuk. Ebben az adathalmazban minden pontfelhő nagyjából 250000 pontot tartalmaz. A gyorsabb feldolgozás érdekében a pontfelhőket voxelrács módszerrel alulmintavételeztük, 0.01-es voxelméretet használva, így a felhők tipikus mérete 100000 pont lett. Ebben az adathalmazban 60 darab, egymással átfedő felhő található előre megadott biztosan helyes transzformációval. Hogy a kiértékelésünk pontos legyen (azaz a nyilvánvalóan esélytelen regisztrációk ne zavarják a kiértékelést), kiválasztottuk azokat a felhőpárokat, amelyek legalább 65%-ban átfednek egymással. 45 pontfelhő párt kaptunk, amely megfelelt ennek a kritériumnak. Az 4.5. ábra egymásra illesztett felhőpárt mutat az adathalmazból, különböző jellegzetességleírókat használva (a pontfelhők egy része le lett vágva a jobb megjelenítés érdekében).

A jellegzetességleírók leíróképességének az összehasonlításához széles körben elterjedt a Precision-Recall Curve (PRC) használata [66]. A görbék elkészítéséhez mind a 45, egymással átfedő felhőpáron végig iterálunk. Mindkét felhőből kiválasztunk $\sigma$ számú pontot

[6] `https:\\github.com\ELTE-IK-Point-Cloud-Group\QBB`



a) b) c)

4.5. ábra: 3D pontfelhő regisztráció különböző jellegzetességleírók használatával: (a) FPFH, (b) QBB-FPFH, (c) B-SHOT. A regisztrációhoz használt transzformációs mátrix az Open3D [100] könyvtárban implementált RANSAC algoritmussal lett meghatározva. A jobb megjelenítésért a pontfelhők egy részét levágtuk.

véletlenszerűen (ebben a kiértékelésben $\sigma = 5000$, ami általában a pontok 5%-a). Ezekre a pontokra kiszámoljuk a pontok jellegzetességleíróit és az egyik felhő minden pontjának jellegzetességleíróihoz megkeressük a két legközelebbi szomszédot a másik felhő pontjainak jellegzetességleírói közül. Ha az arány az első és a második legközelebbi szomszéd között nagyobb, mint egy $\tau$ határérték, akkor a pont és a legközelebbi szomszéd egy megfeleltetést fog alkotni (a leírt módszert "nearest neighbor distance ratio"-nak hívják [36]). Ezután meg kell vizsgálnunk, hogy a megfeleltetések között hány helyes van. Ehhez transzformáljuk a felhőpár egyik tagját az előre megadott helyes transzformációval, hogy egymásra illesszük őket. Ha egy megfeleltetés két pontja a transzformáció után közelebb van egymáshoz, mint egy megadott határérték, akkor ez helyes megfeleltetésnek fog számítani. Jelöljük $TP$-vel a helyes megfeleltetések számát, $FP$-vel pedig a téves megfeleltetések számát. Az összes lehetséges helyes megfeleltetést jelölje $GT$. Ekkor a *precision* és *recall* értékek kiszámíthatóak a következő módon:

$$precision = \frac{TP}{TP + FP} \tag{4.5}$$

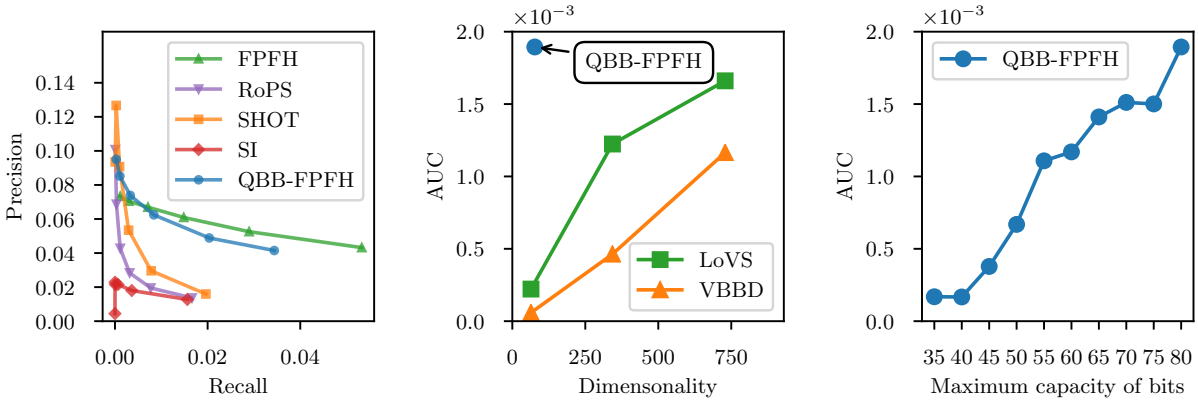
$$recall = \frac{TP}{GT} \tag{4.6}$$

Ahhoz, hogy megkapjuk a PRC-t a $\tau$ határértéket módosítjuk, 0.5-től 1-ig. Azért kezdjük 0.5-től, mert ez azt jelenti, hogy a második legközelebbi szomszéd kétszer akkora távol van, mint az első legközelebbi szomszéd, ami valós pontfelhők esetében nagyon ritkán fordul elő (azonban ilyenkor a *precision* és *recall* szélsőséges értékeket vehetnek fel). Ha kompakt módon akarjuk szemléltetni egy módszer leíróképességét, akkor összehasonlítjuk a PRC görbék alatti terület (AUC) értékeit (az AUC értéket a scikit-learn [51]

metrics.auc függvényének segítségével számoltuk ki). Mivel véletlenszerűen választunk pontokat a kiértékeléshez, ezért minden felhő pár esetén 5-ször lefuttatjuk a kiértékelőt, és az eredmények átlagát tekintjük valós eredménynek.

## 4.5. Eredmények

Ebben a részben szeretnénk bemutatni a kiértékelésünk eredményeit, amiben összehasonlítottuk az általunk javasolt binarizációs módszert más releváns módszerekkel. Ahol nem jelezzük, ott a QBB módszer az alapértelmezett Gray-kódolást használja hagyományos Hamming-távolságot használva távolságmetrikaként, kapacitás korlát nélkül. Az 4.6/(a). ábrán a valós értékű jellegzetességleírók görbéjét láthatjuk. Az ábra szintén tartalmazza az FPFH jellegzetességleíró QBB által binarizált változatát. A Spin Image jellegzetességleíró teljesített a legrosszabbul a valós értékű leírók közül. Ez az eredmény megfelel Guo és társai [66] által végzett kiértékeléseknek, ahol szintén a Spin Image szerepelt a legrosszabbul más jellegzetességleírókkal szemben (FPFH, SHOT, RoPS). Amikor a $\tau$ határérték alacsony, a SHOT és RoPS jellegzetességleírók magas pontosságot tudnak elérni, de magasabb határértékeknél a pontosság szignifikánsan csökkent. Csak az FPFH képes konzisztensen magas pontosságot elérni. Szintén látható az ábrán, hogy a QBB-FPFH majdnem olyan jó leíróképességgel rendelkezik, mint az eredeti valós értékű leíró, de mindenképp jobb, mint a többi valós értékű jellegzetességleírók. A későbbiekben látni fogjuk, hogy a QBB módszer képes az FPFH leírót a legjobb leíróképességgel binarizálni.




4.6. ábra: (a) A QBB-FPFH és a valós értékű leírók PRC görbéje. (b) Dimenziószámok és AUC értékek összehasonlítása a QBB-FPFH és az önálló bináris jellegzetességleírók között. (c) A kapacitáskorlát bevezetésének a hatása a QBB-FPFH leíróképességére.

### 4.5.1. Önálló bináris jellegzetességleírók

A 4.6/(b) ábra azt mutatja, hogyan teljesít a QBB-FPFH az önálló bináris jellegzetességleírókkal szemben. Azért választottuk a QBB-FPFH-t az összehasonlításhoz, mivel az FPFH jellegzetességleíró binarizálása adta számunkra a legjobb eredményt. A VBBD és LoVS módszerek szerzői leírása alapján a módszereik akkor érték el a legjobb eredményt, amikor 729 bit hosszúságú leírót készítettek (9 voxel minden tengely mentén). A QBB-FPFH mérete kapacitáskorlát nélkül 77 bit. Az ábrán látható, hogy az alacsony bitszám ellenére magasabb AUC értékkel rendelkezik, mint a 729 bitből álló önálló bináris jellegzetességleírók. Ha lecsökkentjük az önálló bináris leírók bitszámát 343-ra és 64-re (csak köbszámok lehetséges a módszerek felépítéséből adódóan, akkor láthatjuk, hogy a leíróképességük jelentősen csökken. A VBBD és a LoVS nagyon hasonló módszerek, viszont a lokális vonatkoztatási rendszereket (LRF) máshogy számolják ki. Feltehetően nagyrészt emiatt láthatjuk a különbséget a két leíró AUC értékei között. Az 4.6/(c) ábrán láthatjuk hogyan változik a QBB-FPFH leíróképessége a kapacitáskorlát egyre szigorúbbá válásával. Ahogyan várható volt, a kapacitáskorlát bevezetése negatívan befolyásolja a leíróképességet. A legnagyobb csökkentés az AUC értékekben akkor történik, amikor a kapacitáskorlátot 50-re állítjuk be. Az ábra alapján, ha a jellegzetességleírónkat olyan eszközön szeretnénk használni, amelynek erősen korlátozott tárhelye és számítási kapacitása van, akkor a QBB-FPFH még 65 bit esetében is alkalmas lehet, mivel leíróképessége nem csökken jelentősen.

### 4.5.2. A QBB lehetséges változatai

A bináris jellegzetességleírók előnye, hogy 1) kevés tárhelyet foglalnak és 2) a bitműveletekkel kiszámolt Hamming-távolság sokkal gyorsabb, mint valós értékű jellegzetességleírók közötti távolságok számolása. Ahogy a 4.3.2 fejezetben írtuk, többféle módszert is kipróbáltunk a csoportok kódolására. A 4.1. táblázat megmutatja a Gray-kódokat és Mersenne-kódokat 8 csoport esetén. A QBB esetében az eredeti leíróhoz tartozó dimenziókat különböző számú csoportokra bontjuk fel. Ebből következik az, hogy az egyes dimenziók csoportszáma eltérhet egymástól. Ekkor azonban a hagyományos Hamming-távolság esetében a több csoportot kapott dimenziók súlya megnőhet az eredeti állapothoz képest. Ez a jelenség érinti a Gray-kóddal létrehozott bináris leírót is, azonban a Mersenne-kód esetében erősebb. Például, ha két dimenziót 4 és 8 csoportra kell bontanunk, akkor Gray-kód esetében ezek a dimenziók 2 és 3 bitet kapnak, míg a Mersenne-kódnál 4 és 7 bitet. Hagyományos Hamming-távolságot használva a Gray-kódos változat esetében a második dimenzió $3/2 = 1.5$-ször nagyobb súllyal fog a távolságba számítani, mint az első. A

Mersenne-kód esetében pedig $7/4 = 1.75$-szer lesz nagyobb a súly. Ennek a problémának a megoldására bevezettük a módosított Hamming-távolság távolságmetrikát (Modified Hamming Distance). Tegyük fel, hogy $B = b_1 b_2 ... b_n$ és $B' = b'_1 b'_2 ... b'_n$ két $n$ bitből álló bitvektorok. Ekkor a módosított Hamming-távolságot a következő módon kapjuk meg:

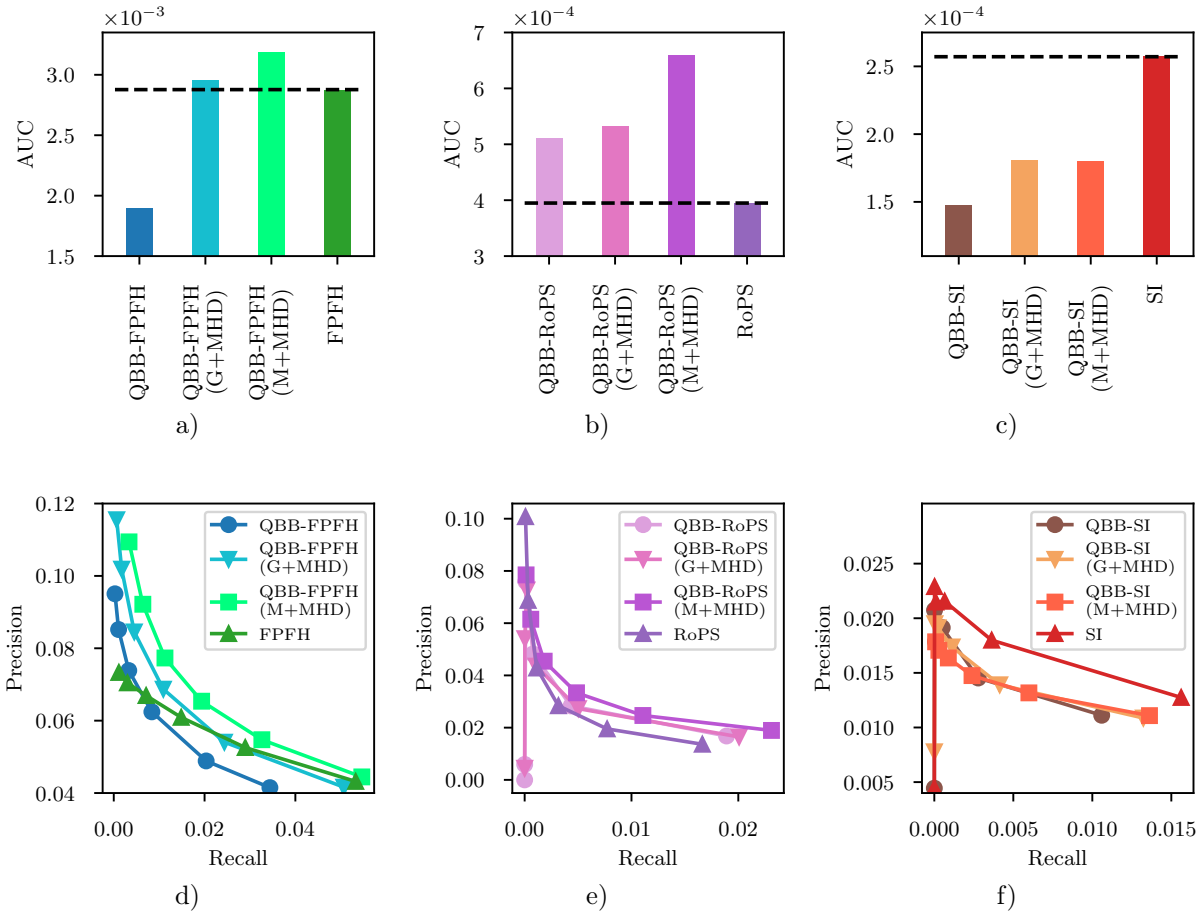
$$MHD(B, B') = \sum_{d=1}^{D} \frac{HD(s_B(d), s_{B'}(d))}{l_d} \tag{4.7}$$

$$s_B(d) = b_{i(d)} ... b_{i(d)+l_d-1} \tag{4.8}$$

$$i(1) = 1 \text{ and } i(d) = i(d-1) + l_{d-1}, 1 < d \leq D \tag{4.9}$$

ahol $HD$ két bitvektor Hamming-távolságát megadó függvény, $D$ az eredeti leíró dimenzióinak a száma, $l_d$ a $d$. dimenzióhoz tartozó bitek száma a binarizált leíróban, $s_B(d)$ pedig a $d$. dimenzióhoz tartozó bitsorozat a binarizált leíróban. A MHD számításakor minden eredeti dimenzióhoz tartozó rész-bitvektorra kiszámoljuk a Hamming-távolságot, aztán ezt elosztjuk a rész hosszával. Így minden rész 0 és 1 közötti értéket fog adni, azaz minden eredeti dimenzióhoz tartozó bitsorozat egyforma súllyal vesz részt a távolságban. Vegyük észre, hogy ehhez számon kell tartani azt, hogy az eredeti leíró egyes dimenziói hány biten vannak reprezentálva a binarizált változatban. Ezért a módosított Hamming-távolság nem tud olyan hatékony lenni, mint a hagyományos Hamming-távolság, mivel azt pusztán bitműveletekkel el lehet végezni. Ha eltekintünk a távolságszámítások futási idejétől, akkor a módosított Hamming-távolságot használva a binarizált leírók akár jobb eredményeket is elérhetnek, mint az eredeti valós értékűek.

Fig. 4.7 megmutatja, hogyan teljesítenek a különböző változatai a QBB-nek. Ezen az ábrán szerepel a QBB-FPFH, ami Gray-kódot használ, a QBB-FPFH (G+W), ami Gray-kódot és módosított Hamming-távolságot használ, és a QBB-FPFH (M+W), ami Mersenne-kódot és módosított Hamming-távolságot. Jól látható, hogy a módosított Hamming-távolságot használó QBB változatok kicsivel jobb eredményt tudnak elérni az eredeti leírótól, miközben sokkal kevesebb tárhelyet foglalnak. A QBB változatok a többi valós értékű leírónál is hasonlóan viselkednek, egyes esetekben az eredeti jellegzetességleírónál is jobb eredményeket tudnak elérni. A sejtésünk az, hogy a módosított Hamming-távolsága számítási költsége kevesebb, mint az euklideszi távolságé, így futási idő és leíróképesség szempontjából és jobb választás lehet a QBB-FPFH módosított Hamming-távolság használatával. Fontos megjegyezni, hogy a sejtésünk alátámasztására nem végeztünk méréseket, ez a jövőbeli munkánk részét képzi. Azt viszont egyértelműen




4.7. ábra: Az AUC értékek (a, b, c) és a megfelelő PRC görbék (d, e, f) a QBB módszernek és változatainak. Jelölések: G - Gray-kód, M - Mersenne-kód, MHD - módosított Hamming-távolság. A QBB-FPFH, QBB-RoPS és a QBB-SI Gray-kódot és hagyományos Hamming-távolságot használnak, míg az FPFH, RoPS és SI euklideszi távolságot. Fontos megjegyezni, hogy az FPFH jellegzetességleíró AUC értéke egy nagyságrenddel nagyobb, mint más jellegzetességleírók értékei.

kijelenthetjük, ha nem a jellegzetességleírók megfeleltetésének keresési idejét szeretnénk minimálisra csökkenteni, hanem fontosabb számunkra a pontosság és az alacsony tárhelyigény, akkor mindenképp a QBB-FPFH-t érdemes használni módosított Hamming-távolsággal.

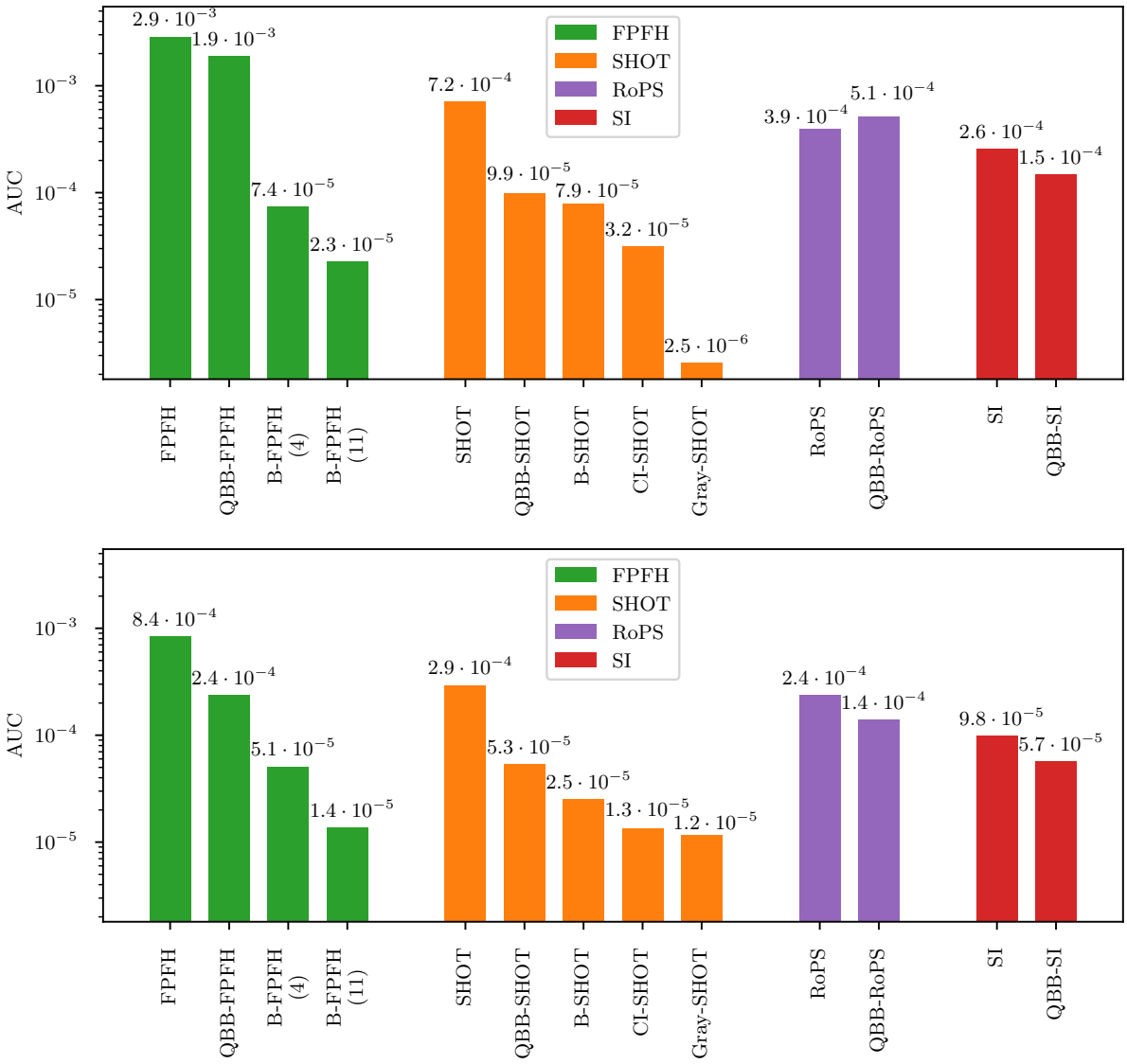
| Módszer | Összehasonlítások száma (#) | Előfeldolgozás | Alkalmazva |
|---|---|---|---|
| QBB (miénk) | $D \cdot 2^N$ | Igen | SHOT, FPFH, RoPS, SI |
| B-SHOT | $\frac{D}{L} \cdot (L \cdot \log L + L)$ | Nem | SHOT, FPFH |
| CI-SHOT | $\frac{D}{L} \cdot 2^N$ | Igen | SHOT |
| Gray-SHOT | $\frac{D}{L} \cdot L \cdot 2^N$ | Igen | SHOT |

4.3. táblázat: A vizsgált binarizációs módszerek tulajdonságai. A második oszlop mutatja az szükséges összehasonlítások számát egy jellegzetességvektor binarizációjához. $D$ a bemeneti jellegzetességvektor hossza, $L$ a kódolási hossz és $N$ az egy valós érték binarizációjához használt bitek száma. A harmadik oszlop azt mutatja, hogy van használ-e előfeldolgozást az algoritmus. Az utolsó oszlop pedig azokat a valós értékű jellegzetességleírókat sorolja fel, amelyre a szerzők alkalmazták a módszerüket.

### 4.5.3. Binarizációs módszerek összehasonlítása

A 4.3. táblázat bemutatja a vizsgált binarizációs módszerek tulajdonságait. Az 4.8. ábrán összesítettük az eredeti, valós értékű jellegzetességleírók és a különböző binarizációs módszerek által kapott jellegzetességleírók AUC értékeit. A jobb megjelenítés érdekében az AUC értékek logaritmikus skálán szerepelnek. Erre azért van szükség, mert egyes módszerek bizonyos leírókkal nagyon rosszul működnek, és másképp megjelenítésük nem lenne lehetséges. Ahol QBB szerepel ott mindig a hagyományos Hamming-távolságot és Gray-kódot használtunk.

A B-SHOT, CI-SHOT és Gray-SHOT algoritmusok célzottan a SHOT leíró binarizálására készültek. A módszerek általánosak, azaz bármilyen valós értékű leírót képesek binarizálni, azonban az algoritmus a SHOT leíró értékeinek eloszlása esetében működnek jól. Ezért a szerzők munkája alapján mi is a SHOT jellegzetességleíróra alkalmaztuk őket. Ezzel szemben Prakhya és társai [86] a módszerüket a SHOT mellett az FPFH jellegzetességleíróra is alkalmazták, ezért mi is így tettünk. Az ábrán látható, hogy a SHOT




4.8. ábra: A valós értékű jellegzetességleírók és a binarizált változataik különböző binarizációs módszerekkel. Az AUC értékek logaritmikus skálán vannak ábrázolva. A felső ábra a *redkitchen* adathalmazon történő kiértékelés eredményeit mutatja, míg az alsó a *livingroom* adathalmazét.

esetében az eredeti jellegzetességleíró leíróképességét egyik módszernek sem sikerült elérnie. A B-SHOT és a QBB-SHOT 352 bitet használ - ahány dimenziós az eredeti leíró. A Gray-SHOT és a CI-SHOT 704 bitet használ, mivel minden eredeti dimenziót négy csoportra osztanak és ezeket két biten kódolják. A B-SHOT és a QBB-SHOT hasonlóan teljesítenek, azonban mindkettő egy teljes nagyságrenddel elmarad az eredeti leírótól, így kétséges, hogy mennyire lennének használhatóak valós esetekben.

Az FPFH esetében a QBB-FPFH áll az eredeti leíróhoz legközelebb. A *livingroom* adathalmaz esetén a QBB-FPFH nem tudja annyira megközelíteni az eredeti leírót. A B-FPFH-t a szerző is kipróbálta, és ő is azt tapasztalta, hogy az FPFH esetében a módszer nem tud jó eredményt elérni. A B-FPFH módszert 4 és 11-es kódolási hosszal is kipróbáltuk, és hasonló eredményeket kaptunk. Vegyük észre, hogy a *redkitchen* esetében a QBB-FPFH jobban teljesít, mint az eredeti SHOT leíró. Ez az előny a *livingroom* esetében nem áll fenn. A CI-SHOT és Gray-SHOT algoritmusokat nem alkalmaztuk az FPFH-ra, mivel a szerzők a cikkükben sem ajánlják.

A RoPS és a Spin Image leírók esetében, csak a QBB-t alkalmaztuk. A *redkitchen* adathalmazon a RoPS esetében a binarizált változat jobb eredményt tudott elérni, mint az eredeti, míg a Spin Image-nél az eredeti maradt a győztes. A *livingroom* adathalmaz esetében a RoPS-nál is gyengébben teljesít a binarizált változat. Az oka annak, hogy a *livingroom* adathalmaz esetén minden leíró rosszabbul teljesít (és a binarizált változataik is), hogy ez az adathalmaz egyszerűbb, mint a *redkitchen*, csak néhány érdekes objektumot tartalmaz, így nehezebb a jellegzetességleíróknak megkülönböztetni a pontok nagy részét.

Összességében az látszik, hogy a QBB módszer minden leírónál jobban teljesít, mint más binarizációs megoldások. A legjobb eredményt a QBB-FPFH érte el, amely jobb eredményt tud elérni, mint más valós értékű leírók.

### 4.5.4. Költségbecslés

Az általunk javasolt módszerhez a B-SHOT, CI-SHOT és Gray-SHOT binarizációs módszerek állnak a legközelebb. Azonban van egy fontos különbség a felsorolt módszerek között. Ahogy a 4.3.2. fejezetben bemutattuk, a QBB számára előzetesen meg kell határozni a binarizációhoz szükséges csoportszámokat. Ezt a lépést nem lehet egyes jellegzetességleírók binarizációja közben elvégezni, mivel abban az esetben képes jól működni, ha több jellegzetességvektort használunk fel egyszerre. A kiértékelés során több pontfelhő jellegzetességvektorait is felhasználtuk a csoportszámok meghatározásához. Ez a lépés hasonló ahhoz, amikor gépi tanulás során a modellt tanítjuk. Ennek megfelelően a cél

az, hogy minél eltérőbb jellegzetességvektorokat elemezzünk. Annak a meghatározása, hogy mekkora és milyen tanítóhalmazra van szükségünk nem triviális és további munkát igényel. A tapasztalataink azt mutatják, hogy egy jellegzetességleíró esetében elegendő egyszer meghatározni a csoportszámokat egy adathalmazra, mivel ezek a csoportszámok jól használhatók más adathalmazok esetében is. Az általunk meghatározott csoportszámok megtalálhatóak a módszer forráskódjánál.

A CI-SHOT és Gray-SHOT módszerek esetében ugyanúgy nem kell az egyes jellegzetességvektorok binarizációja közben elvégezni a számításokat, hanem elegendő egyszer ezt megtenni.

A fenti megfontolás következtében a költségbecslés során egyedül az online feldolgozási idő költségeit vettük számításba. Egy binarizációs módszer esetén a bemenet egy jellegzetességvektor, így a futási idő a bementi jellegzetességvektor hosszától függ (jelölje $n$). A binarizáció során a leggyakrabban előforduló művelet a valós értékek összehasonlítása, és mivel a számításokat nem kell minden vektor esetén elvégezni, ezért úgy gondoljuk elegendő csak azokat vizsgálni. Két fontos állandó még a kódolási hossz ($L$) és az egy dimenzió kódolásához használt bitek száma ($N$). Jelölje $T(D)$ egy binarizációs algoritmus összehasonlításainak a költségeit. Ekkor

$$T_{QBB}(D) = D \cdot 2^N$$

$$T_B(D) = \frac{D}{L} \cdot (L \cdot \log L + L)$$

$$T_{CI}(D) = \frac{D}{L} \cdot 2^N$$

$$T_{Gray}(D) = \frac{D}{L} \cdot L \cdot 2^N$$

megadják a QBB, B-SHOT (B-FPFH), CI-SHOT, Gray-SHOT módszerek költségeit. Az $L$ értéke általában 4 vagy 11, míg az $N$ értéke 4 vagy 5, így ezek állandónak tekinthetők. Jól látható, hogy mind a négy módszer futási ideje $\mathcal{O}(n)$, azaz lineárisan nő a bementi jellegzetességvektor méretével.

## 4.6. Összegzés

Ebben a munkában javasoltam egy binarizációs módszert, a QBB-t, amely képes tetszőleges 3D jellegzetességleírót binarizálni. Bemutattam a módszer lehetséges változatait és

azok teljesítményeit. Összehasonlítottam a módszerünket más binarizációs módszerekkel és önálló bináris jellegzetességleírókkal is. Valós pontfelhőkön végzett kiértékelések szerint a módszer jobb leíróképességgel rendelkezik, mint az önálló binráis leírók és más binarizációs módszerek által létrehozott bináris jellegzetességleírók, továbbá kevesebb tárhelyet igényel. A módosított Hamming-távolság és Mersenne-kód bevezetésével, néhány esetben jobb eredményeket képes elérni, mint az eredeti leírók.

A binarizációs módszerek használatával a pontfelhő alapú lokalizáció, objektumfelismerés, mintaillesztés, regisztráció futási ideje jelentősen csökkenhet. Jövőbeli terveink között szerepel a közelmúltban megjelent jellegzetességleírókra is alkalmazni a módszerünket, megvizsgálni egyes dimenziók fontosságát és felhasználni az így nyert tudást a csoportszámok meghatározásánál.

# 5. fejezet

# Sablonillesztés nagy méretű 3D pontfelhőben adatbázis-kezelő rendszer használatával

A LIDAR és mélység szenzorok óriási fejlődésen mentek keresztül az elmúlt években, ezzel lehetővé téve a nagy mennyiségű 3D pontfelhő adatok begyűjtését. A nagy méretű 3D pontfelhők feldolgozása új módszerek és algoritmusok megjelenését igényli. A jelenleg létező pontfelhő feldolgozási folyamatok nagy része úgy lett kidolgozva, hogy feltételezik, hogy az egész pontfelhő befér a memóriába. Azonban, ha a pontfelhő mérete meghaladja a memória méretét, akkor a feldolgozáshoz új módszerek szükségesek. Ebben a fejezetben javaslok egy adatbázis-kezelő rendszer alapú 3D pontfelhő felfoldozó folyamatot, amely megoldja a sablonillesztés feladatát, azaz megkeresi egy pontfelhő egy vagy több előfordulását egy jóval nagyobb pontfelhőben, amely adatbázis-kezelő rendszerben kerül tárolásra. Az adatok tárolásához egy kompakt és újszerű reprezentációt használok, hogy kiaknázzuk az indexstruktúrák lehetőségeit. A lekérdező algoritmus korábbi munkákban használt módszereket használ fel újszerű összeállításban. A legjobb tudásom szerint ez az első olyan sablonillesztést megoldó folyamat, amely kihasználja az adatbázis-kezelő rendszerek (a továbbiakban ABKR) előnyeit.

## 5.1. Bevezetés

A LIDAR, Time-of-Flight és sztereó szenzorok elterjedése lehetővé tette egyre nagyobb 3D ponthalmazok egyszerű begyűjtését. Míg a korábban felhasználói eszközökbe épített szenzorok alacsony felbontásúak és másodpercenként 5 képkocka felvételére voltak képe-

sek, addig jelenlegi a jelenlegi szenzorok magasabb felbontással és másodpercenként 90 képkocka sebességgel való rögzítést is elérik. Az így gyűjtött adatok felhasználhatóak különböző feladatok megoldására: robot navigáció, lokalizáció, virtuális és kiterjesztett valóságot (VR és AR) használó feladatok [114, 50, 108]. A begyűjtött adatok, a méretük miatt egy idő után nem kezelhetőek kizárólag a memóriába, ami új kihívások elé állítja a pontfelhők feldolgozását végző módszereket.

A sablonillesztési feladat célja, hogy megtaláljuk egy relatív kis méretű 3D pontfelhő egy vagy több előfordulását egy jóval nagyobb 3D pontfelhőben. A kis méretű 3D pontfelhőt lekérdezésfelhőnek hívjuk és általában egy objektumot vagy a nagy felhő egy kisebb részét értjük alatta. A nagy méretű 3D pontfelhőt pedig helyszínfelhőnek hívjuk. A létező sablonillesztési feladatot megoldó módszerek feltételezik, hogy a helyszínfelhő vagy annak reprezentációja teljes egészében befér a számítást végző eszköz memóriájába, így a lekérdezések gyorsan végrehajthatóak. Ebben a fejezetben egy olyan folyamatot javaslok, amely megoldja a sablonillesztési problémát, miközben a bemeneti 3D pontfelhőket adatbázis-kezelő rendszerben tárolja és kihasználja az adatreprezentáció által alkalmazható hatékony indexstruktúrákat.

A javasolt folyamat felbontható offline és online fázisokra. Az offline fázis tartalmazza a helyszínfelhő előfeldolgozását (zajszűrés, mintavételezés, kulcspontdetektálás, jellegzetességleírók kiszámítása, dimenzióredukció), az adatok betöltését az adatbázisba és az indexstruktúrák létrehozását. Az offline fázis időigényes, mivel feltételezzük, hogy a helyszínfelhő nagy számosságú. Az online fázisban a beérkező lekérdezésfelhő előfeldolgozása történik. Ebben a fázisban felhasználjuk az adatbázis-kezelő rendszer által nyújtott lehetőségeket, hogy az online lekérdezés minél gyorsabban történjen. A kiértékelést egy prototípus implementáció segítségével végezzük, amely a PostgreSQL adatbázis-kezelő rendszer használatával készítettünk el. A legjobb tudásunk szerint ez az első munka, amely azt vizsgálja, hogy az adatbázis-kezelő rendszerek hogyan tudják felgyorsítani a sablonillesztés feladatát megoldó folyamatokat.

Főbb hozzájárulásaink a következők:

1. Definiáljuk a sablonillesztés egy speciális feladatát, melynek lényege, hogy a nagyméretű helyszínfelhő előzetesen elérhető és offline feldolgozható, míg a lekérdezésfelhő online érkezik.

2. Javaslunk egy újszerű jellegzetességleíró alapú regisztrációs folyamatot, amely megoldja a sablonillesztési feladatot.

3. Implementáltuk a javasolt módszer prototípusát a PostgreSQL adatbázis-kezelő

rendszer segítségével.

4. A kiértékelésünk megmutatja, hogy a javasolt módszer számos gyakorlati előnnyel jár: triviálisan skálázható, képes memóriába nem beférő méretű adatokkal dolgozni, képes indexstruktúrákat használni a hatékony és gyors keresésekhez stb.

## 5.2. Kapcsolódó munkák

Bár a legtöbb munka, amely 3D pontfelhő feldolgozással foglalkozik feltételezi, hogy a bementi adatok befférnek a memóriába, léteznek megoldások, amelyek javasolják az adatbázis-kezelő rendszerek használatát pontfelhők tárolására és bizonyos feldolgozási műveletekre.

Oosterom és társai [69] szerint az ABKR egy megfelelő és jól használható eszköz pontfelhők tárolására. A munkájukban kiértékelnek különböző rendszereket abból a szempontból, hogy mennyire megfelelő pontfelhő tároláshoz. A kiértékelt rendszerek a következők: PostgreSQL, MongoDB, Oracle Database és LASTools. A kiértékeléshez Hollandiában felvett légi felvétellel begyűjtött pontfelhőt használtak (2.5D pontfelhők). A pontfelhők tárolására a szerzők az ún. lapos tábla modellt javasolják, amiben minden egyes pont egy sorban foglal helyet az adatbázis egy táblájában. Így az adat egyszerűen szűrhető és rendezhető a legtöbb felhasználási eset számára. Például a pontok rendezése végbe mehet a Hilbert-kódjuk vagy Morton-kódjuk alapján. Ezután könnyedén blokkokat képezhetünk a rendezett sorok fölött, melyeknek számos előnye lesz: a) térbeli módon összefüggők, b) használhatók egyszerű tömörítésre (pl. az összes koordinátának az első számjegyei azonosak és blokk szinten tárolhatók), c) kevesebb sortöbbletköltség lesz a sima tábla modellhez képest, d) gyorsítótárazásra is használhatók. Ebben a munkában egy lekérdezés-algoritmust is meghatároznak, amely felhasználja a Morton-kódot, illetve annak közvetlen kapcsolatát a negyedelő fákkal.

Cura és társai cikkükben [82] a nagy (több terabájt méretű) adathalmazok miatt a tömörítést hangsúlyozzák, és pontblokkokkal dolgoznak egyedi pontok helyett. Ebben az esetben fontos, hogy a pontblokkoknak kompatibilisnek kell lennie a releváns és kapcsolódó lekérdezésekkel. Hangsúlyozzák továbbá, hogy az összes blokknak nem kell ugyanazt a szabályt követni a méret szempontjából, így a blokkméretet a pontfelhő lokális jellemzőinek megfelelően lehet adaptálni. Ez jelentheti azt, hogy a csoportosítás a pontsűrűség alapján vagy a pontoknak valamilyen előre adott osztályozása szerint történik. Munkájukban olyan pontfelhőkkel foglalkoznak, ahol a sűrűsége a pontoknak homogén, mint pl. a légi LIDAR felvételek esetében, amely általában nem áll fenn a beltéri adatok-

ra. A lekérdezések válaszidejének felgyorsítására B-fát és R-fát, illetve függvényindexelést alkalmaznak.

Meyer és Brunn munkájukban [105] a 3D-s pontfelhők PostgreSQL/PostGIS adatbázisba való integrálással foglalkoznak használva a PostgreSQL-hez készült Pointcloud nevű kiterjesztést és a Point Data Abstraction Library (PDAL) függvényeit. A térben közeli pontokat blokkokba szervezik, és egy blokkot egy tábla sorában tárolják. A pontok csoportosításához a pontfelhő adatbázisba történő importálása során használják a PDAL-t. A PDAL két alapfüggvényt nyújt, amelyekre igaz, hogy valójában 2D-s csempézések a 3D-s pontfelhőn, mivel a Z tengelyt nem veszik figyelembe. Az egyikkel irreguláris csempézést lehet megvalósítani, ahol minden blokk térbelileg folytonos és nem fednek át egymással. Továbbá mindegyik blokknak meg van adva a kapacitása. A másik függvénnyel reguláris csempézést lehet végrehajtani, ahol a pontok reguláris négyzetű blokkokba vannak szervezve, a felhasználó által meghatározott rácsméret alapján. Ennél a módszernél a blokkok XY síkbeli kiterjedésére nézve ugyanakkorák, de változó számú pontokat tartalmaznak. A cikk szerzői a 2D-s csempézést úgy próbálják javítani, hogy egy további PDAL függvényt is felhasználnak, amivel korlátozni tudják mindegyik csoportra a maximális Z kiterjedést, és ezt az előző kettő függvénnyel kombinálva már 3D-s csempézéseket kapnak. Olyan lekérdezéseket tekintenek, amelyek egy tetszőleges poliéderbe eső pontokat adnak vissza. Viszont erre önmagában a PostGIS és a Pointcloud kiterjesztés nem lenne alkalmas. Emiatt kerülő megoldáshoz folyamodnak és a fent említett csempézési módszereket összehasonlítják. Igazából csak annyit állapítanak meg, hogy a 3D-s kerülő megoldásukat mindenképpen érdemes alkalmazni, de arról egyértelműen nem vonnak le következtetést, hogy a reguláris vagy az irreguláris csempézés jobb-e.

A kapcsolódó munkák alapos áttekintése alapján megállapítható, hogy bár vannak a munkánk szempontjából lényeges megközelítések és ötletek, nem találtunk olyan cikket, amely a bevezetésben említett lekérdezésfelhőnek a helyszínfelhőben való megkeresésének speciális feladatával foglalkozott volna az adatbázis-kezelő rendszerek kontextusában. Mivel nem egyértelmű a pontok blokkokba szervezésének előnye, így lapos tábla modellt alkalmazunk, és a Pointcloud kiterjesztés használatától is eltekintünk.

## 5.3. A sablonillesztés feladat jellegzetességleíró alapú általános megoldása

### 5.3.1. A feladat definiálása

Jelölje $\mathcal{S}$ a helyszínfelhőt, amely egy nagy méretű 3D pontfelhő (pl. magas felbontású felvétel egy vagy több szobáról, épületek, városok vagy akár annál is nagyobb területek), $\mathcal{Q}$ pedig jelölje a lekérdezésfelhőt, amely kisebb, mint a helyszínfelhő és általában egy objektumot reprezentál (pl. háztartási objektum, gépjármű, ipari objektumok stb.). A célja a sablonillesztésnek, hogy megtalálja a $\mathcal{Q}$ lekérdezésfelhő összes előfordulását a $\mathcal{S}$ helyszínfelhőben és megadja $T_1, ..., T_n$ transzformációkat, ahol $n \geq 0$ az előfordulások száma és $T_i$ ($i \in [1..n]$) az $i$.-edik előforduláshoz tartozó transzformáció. Az általunk definiált sablonillesztési feladat a ponthalmaz regisztráció egy speciális esete, ahol igazak a következő feltételek:

1. $\mathcal{S} >> \mathcal{Q}$, azaz a helyszínfelhő nagyságrendileg nagyobb, mint a lekérdezésfelhő.
2. A $\mathcal{Q}$ lekérdezésfelhő előfordulásainak a száma előre nem ismert.
3. A $\mathcal{S}$ helyszínfelhőt előre ismerjük, így a költséges és időigényes előfeldolgozási lépéseket offline végezhetjük, míg a $\mathcal{Q}$ lekérdezésfelhő online kerül feldolgozásra.

### 5.3.2. Általános megoldás

Ebben a fejezetben a sablonillesztés általános megoldásának lépéseit szeretnénk leírni, miközben az egyes lépések adatbázisba való integrálását is kiemeljük. A probléma fenti feltételeit figyelembe véve az általános folyamat is feloszthető offline és online fázisra.

1. **Offline fázis.** Mivel a helyszínfelhőt előre ismerjük, azért a költséges előfeldolgozási lépéseket offline is elvégezhetjük, így az online fázisban nem kell időigényes számításokat végeznünk. Az előfeldolgozás részei lehetnek például a mintavételezés, kiugró értékek detektálása és eltávolítása, normálvektorok becslése és orientációja és egyéb adattisztító és előre kiszámítható számítások elvégezése. A jellegzetességleíró alapú regisztrációs folyamatok egyéb műveleteket is elvégezhetnek, amelyek nagy méretű felhőkre költségesek lehetnek, úgymint a kulcspontok detektálása, jellegzetességleírók kiszámítása, a jellegzetességleírók dimenziócsökkentése vagy akár pont n-esek létrehozása. Az offline fázis eredménye az előfeldolgozáson átment előre ismert helyszín pontfelhő. Ebben a fázisban van egyúttal a legnagyobb szerepe az ABKR-nek.

Mivel a műveletek offline történnek, ezért ez a megfelelő hely a pontfelhők és jellegzetességleírók adatbázisba való betöltésének és az indexstruktúrák felépítésének.

2. **Online fázis.**

   (a) **A lekérdezésfelhő előfeldolgozása.** Általánosan nem feltételezhetjük azt, hogy a beérkezett lekérdezésfelhőt az illesztéshez megfelelő állapotban kapjuk. Ebben a lépésben végezhetjük el az illesztéshez szükséges előfeldolgozási lépéseket. Ebbe beletartozhatnak az offline fázisnál felsorolt műveletek, de akár egyéb szükséges feladatok is. Ebben a lépésben azonban különös figyelmet kell fordítanunk arra, hogy az alkalmazott módszerek számítási költsége alacsony maradjon, hiszen ez már hatással van az online feldolgozási időre. Ha a sablonillesztési folyamat a gyors online feldolgozásra szeretne optimalizálni, akkor megfontolandó kevésbé költséges előfeldolgozási lépéseket alkalmazni. Például, a normálvektor becslésére különböző algoritmusok léteznek, ezért érdemes lehet az offline részben a helyszínfelhőhöz a legpontosabb módszert alkalmazni, míg az online fázisban feldolgozott lekérdezésfelhőhöz gyorsabb, de esetenként pontatlanabb módszert.

   (b) **Megfeleltetések keresése és elutasítása.** A lekérdezésfelhőnk összes előfordulásának megtalálásához pontok közötti megfeleltetéseket kell keresnünk (pontok közötti, pont n-esek közötti megfeleltetések stb.). Általában a megfeleltetések keresése a jellegzetességleírók legközelebbi szomszédjainak megkeresésével történik. Ebben a lépésben az ABKR-nek és a létrehozott indexstruktúráknak fontos szerep jut. A pontok és jellegzetességleíróik megfelelő tárolásával és indexelésével a legközelebbi szomszéd keresések gyorsan végrehajthatók nagy mennyiségű pont esetén is. A becsült megfeleltetések között sok hibás szerepelhet, ezért ez a lépés általában tartalmazza a megfelelések visszautasítását is. A visszautasítás során különböző tesztek (vagy megszorítások) alkalmazásával a hibás visszautasítások kiszűrhetőek (pl. ponthármas-teszt és reciprocitás teszt [79]). A lépés célja, hogy a lehető legtöbb helyes megfeleltetést tudjuk továbbadni a következő lépésnek.

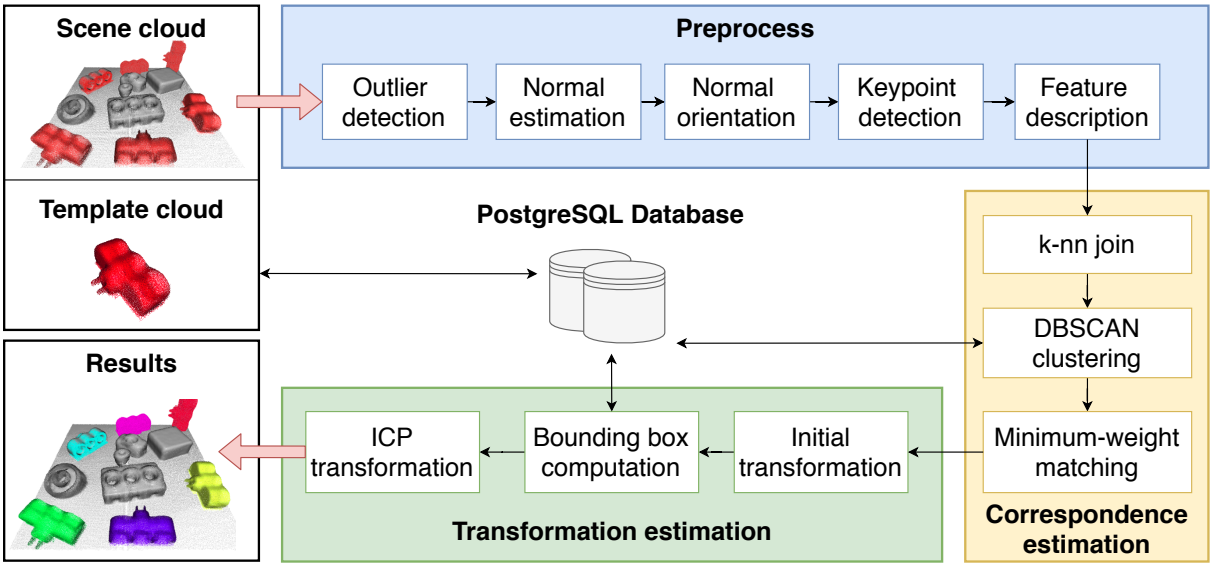
   (c) **Transzformáció becslés és validáció.** Ebben a lépésben az előzőleg becsült megfeleltetések alapján egy vagy több transzformációt számolunk, amely illeszti a lekérdezésfelhőt a helyszínen való előfordulásaira. A jellegzetességleíró alapú regisztrációs folyamatok által megadott durva, kezdeti transzformációt gyakran használják egy iteratív módszer bemeneteként, amely elkészíti a végső,

finomított transzformációt. Ha van egy kezdeti, durva transzformáció, akkor a feladat olyan ponthalmaz regisztrációs feladatként értelmezhető, ahol jóval kevesebb ponttal kell dolgozni, mint az eredeti feladat esetében (csak a megfeleltetések pontjai, vagy azoknak a környezete). Így tehát bátran használható iteratív regisztrációs módszer. Az egyik legnépszerűbb az Iterative Closest Point (ICP) algoritmus és annak különböző változatai ([63, 118]). A transzformációk becslése után következik a transzformációk validálása. A validálás során hozzá kell rendelnünk az egyes transzformációkhoz egy "jóság" értéket, amely megmondja, hogy az adott transzformáció milyen közel van a helyes megoldáshoz. Általában a jóságot egy olyan értékkel mérjük, amely megpróbálja megmondani, hogy a transzformáció után a helyszínfelhő és a lekérdezésfelhő mekkora átfedésben van és a pontjaik milyen messze esnek egymástól (átfedés, fitness stb.). A megfelelő jóság érték meghatározása nehéz feladat, hiszen előfordulhat, hogy egy objektum, amelyet a lekérdezésfelhő reprezentál nem szerepel teljes egészében a helyszínfelhőben, így sosem lehetséges 100%-os átfedés. Egy transzformációt akkor tudunk elfogadni helyes illesztésnek, ha a választott jóság érték elér egy határértéket. A folyamat szempontjából kulcsfontosságú a helyes határérték meghatározása. Ehhez figyelembe kell venni a pontfelhők zajosságát, kitakarást stb. A validációnál fontos meghatározni, hogy a helyszínfelhőben az transzformált lekérdezésfelhő közelében milyen környezetet vizsgálva szeretnénk elvégezni a validációt. A szükséges területbe eső pontok hatékony lekérdezéséhez szintén indexstruktúrák használatára van szükség.

## 5.4. Javasolt módszer

A korábbi feltételeknek megfelelően feltesszük, hogy a helyszínfelhőnk nem fér el a számítást végző eszköz memóriájában, így a jelenleg létező sablonillesztési folyamatok nem alkalmazhatóak. A probléma megoldására egy ABKR-t használó jellegzetességleíró alapú sablonillesztési folyamatot javaslunk, amely a következő lépésekből áll (lásd 5.1. ábra):

1. Előfeldolgozás
   (a) Kiugró értékek kiszűrése és normálvektor becslés és orientáció (5.4.1. fejezet)
   (b) Kulcspont detektálás (5.4.2. fejezet)
   (c) Jellegzetességleírók kiszámítása (5.4.3. fejezet)




5.1. ábra: A javasolt adatbázis-kezelő rendszert használó jellegzetességleíró alapú sablonillesztési folyamatunk

2. Megfeleltetések becslés (5.4.5. fejezet)

3. Transzformációk becslése (5.4.6. fejezet)

A javasolt folyamat online és offline fázisokból áll. Az offline fázis tartalmazza a helyszínfelhő előfeldolgozását: az adatok tisztítását, a kulcspontok detektálást, és a jellegzetességleírók kiszámítását. Fontos megjegyezni, hogy a 3D pontfelhőben lévő koordinátákon túl, az előfeldolgozás eredményeként kapott adatokat is adatbázisban tároljuk (pl. a kiszámított jellegzetességleírókat) az online fázis gyorsítása érdekében. Az online fázisban a lekérdezésfelhőt bemenetként kapjuk és alkalmazzuk rá az előfeldolgozási lépéseket, hogy a helyszínfelhő és a lekérdezésfelhő tulajdonságai megegyezzenek. Végezetül a megfeleltetés és transzformáció becslés lépések következnek, amelyek választ adnak a lekérdezésre, azaz visszaadják a lekérdezésfelhő előfordulásaihoz való illesztések transzformációit. Ebben a fejezetben részletesen bemutatjuk a javasolt folyamat egyes elemeit. Más munkánkban, ahol a problémafelvetés azonos, az offline és online fázisokat máshogy osztják fel ([108]). Pontosabban, a helyszínfelhőt tekintik online érkezőnek a lekérdezésfelhőket pedig offline feldolgozhatónak.

### 5.4.1. Kiugró értékek kezelése és normálvektor számítás

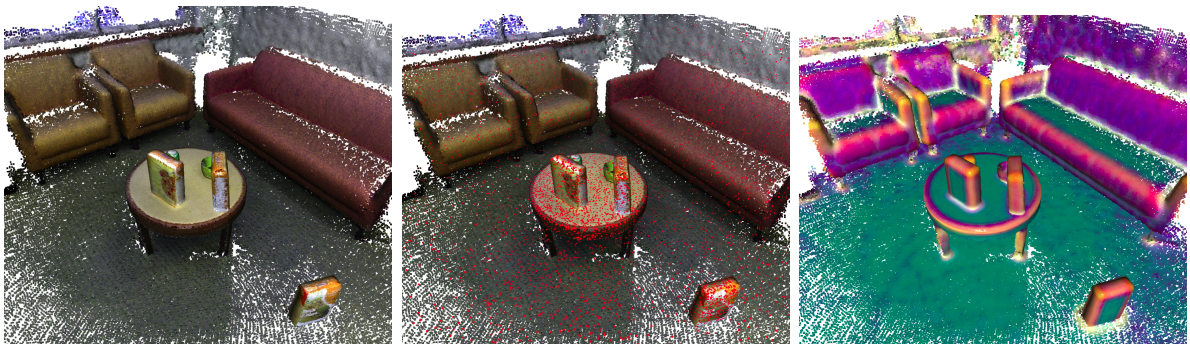
Különböző mélység adatokat felvevő szenzorok eltérő tulajdonságú 3D pontfelhőket készíthetnek. Nagy eltérés figyelhető meg egy LIDAR és egy ToF szenzor által felvett pontfelhőben, azonban két hasonló technológiát használó szenzor esetében is különböző

felhőket kaphatunk (pl. eltérő sűrűség, zajosság stb.). Ezekre az eltérésekre érdemes odafigyelni az előfeldolgozás során.

A 3D pontfelhők jellemzésére széles körben használt jellemző a pontfelhőfelbontás ($\rho$). A pontfelhőfelbontás az átlaga a pontok és a legközelebbi szomszédjuk közötti távolságoknak (2.2.4. fejezet). Ez a jellemző a pontfelhő sűrűségével van kapcsolatban. Az előfeldolgozásban használt algoritmusok paramétereit célszerű a pontfelhőfelbontás érték alapján beállítani, hogy különböző bemeneti felhők esetében is jó eredményeket kapjunk. Jegyezzük meg, hogy a pontfelhő felbontás használata erre a célra nem feltétlenül megfelelő, ha a pontfelhő sűrűsége nem egyenletes. Ebben az esetben a paraméterbeállításhoz más jellemzők figyelembevételére is szükség lehet (pl. a pontok és legközelebbi szomszédjuk távolságainak a szórása).

Mivel minden nem szintetikus, szenzor által felvett adat tartalmaz valamilyen szintű zajosságot, ezért első lépésben a kiugró értékek kiszűrését végezzük el. Az algoritmus, amit használunk egy egyszerű megoldás a kiugró értékek kiszűrésére. Minden pontban egy síkot illeszt a pont környezete alapján, amelyet meghatározhatunk sugár vagy legközelebbi szomszédok száma alapján. Az algoritmus eltávolítja azokat a környezetben lévő pontokat, amelyek egy bizonyos határértéktől távolabb esnek ettől az illesztett síktól. Az algoritmus könnyen el tudja távolítani az izolált pontokat, azonban a határérték beállításánál gondosan kell eljárni. Rossz határérték esetén (pl. túl alacsony) számunkra fontos pontok is eltávolításra kerülhetnek, vagy egyáltalán nem tűnnek el az izolált pontok (pl. túl nagy határérték). A kiugró értékek kiszűrésének lépése elhagyható, ha feltételezhetjük, hogy az adatunk csak minimális kiugró értéket és zajt tartalmaz.

A jellegzetességleírók nagy része használja a pontok normálvektorát a jellegzetességvektorok kiszámításához. Ezért a következő lépés a normálvektorok becslése a pontfelhő minden pontjára. A normálvektorok becsléséhez használt algoritmus fontos paramétere a pont szomszédságának meghatározásához szükség sugár. Továbbá, a jellegzetességleíróknál és későbbi lépéseknél is szükséges egy pont szomszédságát meghatározni, amelyekhez ugyanúgy paraméterként kell megadnunk a sugárt. Vegyük észre, hogy a különböző algoritmusok (normálvektor számítás, jellegzetességleíró stb.) sugarai hatással vannak egymásra. Az általunk használt jellegzetességleíró esetében (FPFH) a normálvektorok kiszámításához használt sugárnak kisebbnek kell lennie a jellegzetességleíróhoz használt sugártól. Az ilyen és ehhez hasonló megszorítások miatt célszerű az algoritmusok paramétereinek értékét azonos jellemzők alapján beállítani. A paraméterek beállítása nehéz probléma és még nem született rá széles körben használt és elfogadott megoldás. Úgy döntöttünk, hogy a paramétereket a felhő sűrűségét mérő pontfelhő felbontás ($\rho$) alapján



a) b) c)

5.2. ábra: Helyszínfelhő: (a) Az *indoor* adathalmaz (b) Véletlenszerűen kiválasztott kulcspontok (piros pontok az ábrán) a helyszínfelhőn (c) Színtér az első három dimenzió alapján (a három legnagyobb sajátértékhez tartozó sajátvektornak megfelelően)

számítjuk ki.

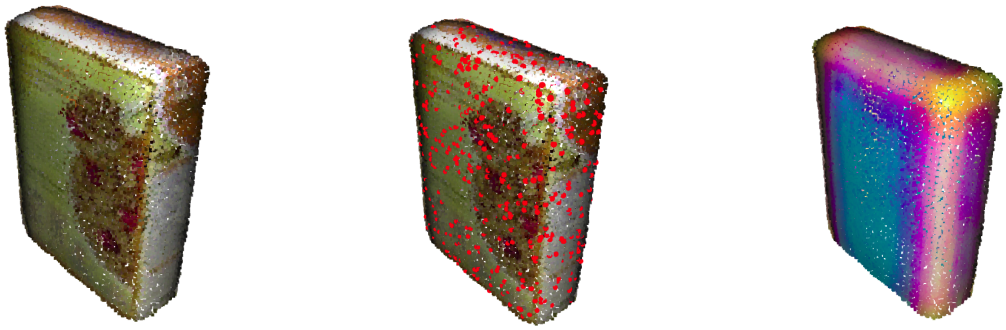
Egy másik megoldás lehet a paraméterek optimális beállítására a rácskeresés a megszorítások figyelembevételével [2]. Azonban ez költséges számítás és a különböző tulajdonságú felhők esetében újra el kell végezni, ugyanis a sugár paraméter (és más paraméterek is) értéke függ a pontfelhő tulajdonságaitól (sűrűség, zajosság, méret stb.).

A normálvektorok kiszámítása után fontos, hogy orientációjuk globálisan konzisztens legyen. A normálvektorok globálisan konzisztens orientációjának elérése nehéz probléma, és sok javaslat született a megoldására [106]. A probléma lényege, hogy egy felületen lévő egymás melletti pontok normálvektorai is állhatnak ellenkező irányba, a jellegzetességleírók pedig sokszor felhasználják a normálvektorokat a jellegzetességvektorok kiszámításához. Az jellegzetességleíró algoritmusok pedig érzékenyek a nem konzisztens orientációra. A globálisan konzisztens orientáció eléréséhez Hoppe és társai által javasolt [24] minimális feszítőfákat használó algoritmust alkalmaztuk.

Az előfeldolgozás után a pontfelhőt betöltjük az adatbázisba. Az előfeldolgozási lépések során eltávolítottuk a kiugró értékeket és globálisan konzisztens normálvektorokat kaptunk. Feltesszük továbbá, hogy a pontfelhőknek egyenletes és egyforma a sűrűségük. Ha a pontfelhők sűrűsége eltérne, akkor további előfeldolgozási lépések lehetnek szükségesek, úgymint az alulmintavételezés vagy sűrítés.

### 5.4.2. Kulcspontok detektálása

Mivel feltételezzük, hogy nagy méretű helyszínfelhővel rendelkezünk, ezért alapvető fontosságú, hogy a kiszámítási bonyolultságot csökkenteni tudjuk. Ehhez a nagy méretű felhők mintavételezése szükséges. A legtöbb ponthalmaz illesztő és pontfelhő regisztráló



a) b) c)

5.3. ábra: Lekérdezésfelhő: (a) A doboz objektum, amely az *indoor* adathalmazban lévő asztalon helyezkedik el (b) Véletlenszerűen kiválasztott kulcspontok (piros pontok az ábrán) a doboz objektumon (c) Színtér az első három dimenzió alapján

módszer [68] olyan algoritmusokat alkalmaz, amelyek a pontfelhő valamilyen szempontból érdekes pontjait kiválasztják. Ez ilyen algoritmusokat hívjuk kulcspontdetektálóknak, a kiválasztott érdekes pontokat pedig kulcspontoknak. Számos kulcspontdetektáló algoritmus készült az elmúlt években. Korábban az ún. hand-crafted (kézzel készített) algoritmusok voltak népszerűek és elterjedtek [87, 36], míg napjainkban a gépi tanulást használó algoritmusok nyernek egyre nagyobb teret [102, 99]. A célja a kulcspontdetektáló algoritmusoknak, hogy kiválasszanak a pontfelhőből jól megkülönböztethető, egyedi és érdekes pontokat, amelyeket lehetséges különböző felvételeken azonosítani. A kulcspontok használatával a pontfelhő regisztrációt végző módszereknek elég csak a kulcspontokkal dolgozniuk az egész felhő helyett. Ha a kulcspontok száma eléri a teljes felhő számosságának 1-2%-át az már elég lehet megfelelő regisztrációra [2].

A javasolt folyamatunk elkészítéséhez megvizsgáltuk a korszerű és széles körben ismert kulcspontdetektáló módszereket [87, 36, 44]. A tapasztalataink alapján egyik módszer sem adott konzisztensen megbízható eredményeket az általunk használt bemeneti adathalmazokra. A fő probléma a kipróbált kulcspontdetektálókkal szembe az volt, hogy egyes esetekben a kulcspontok távol estek egymástól, így a folyamatunk későbbi lépésében használt sűrűség alapú klaszterezés rossz eredményekre vezetett (egy objektumhoz használt klasztereket vágott szét több klaszterre). Ezen tapasztalatok birtokában úgy döntöttünk, hogy véletlenszerűen kiválasztott kulcspontokat fogunk használni. Ez a módszer nem különbözik egy normális eloszlású véletlen mintavételezéstől. Jegyezzük meg a pontosság kedvéért, hogy szigorúan véve az így kiválasztott pontok nem tekinthető kulcspontoknak, mivel a véletlen kiválasztás miatt ezek a pontok nem lesznek egyedibbek vagy lényegesebbek a nem kiválasztott pontoktól. Azonban az elnevezést azért tartjuk indokoltnak, mert

bár az implementációnkban véletlenszerűen választottunk ki pontokat, elképzelhető egy olyan kulcspontdetektáló algoritmus, amely valódi kulcspontokat választ ki és megfelel az általunk támasztott követelményeknek is. Egy ilyen módszerre mindenféle gond nélkül kicserélhető a jelenlegi algoritmus.

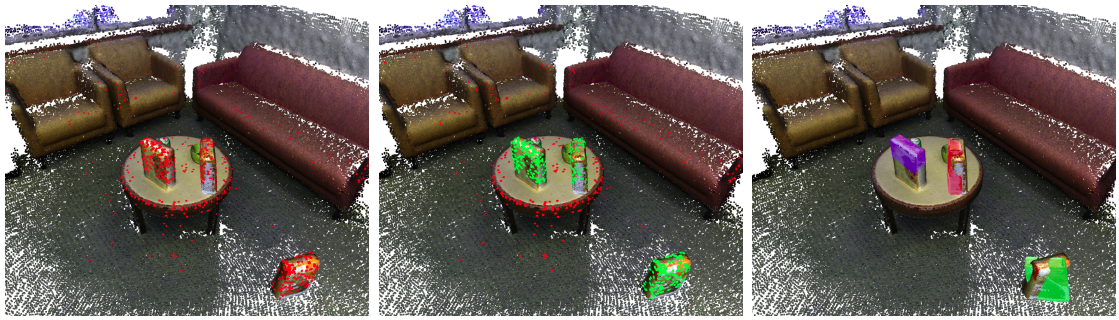
A bemeneti pontfelhőktől függően a véletlenszerűen kiválasztott kulcspontok 2-5%-a elég a helyes illesztéshez [2]. Mivel a kulcspontok kiválasztását nem csak a helyszínfelhőre kell elvégezni, hanem a lekérdezésfelhőre is, ezért a véletlenszerű kulcspontkiválasztás nagy előnye a sebessége. Az 5.2. ábrán látható a kiértékeléshez használt adathalmazok egyike. A 5.2/(a) ábrán a 3D pontfelhőben lévő összes pont látható, míg a 5.2/(b) ábrán pirossal kiemeltük a véletlen mintavételezéssel kiválasztott kulcspontokat ugyanazon a pontfelhőn.

Velünk ellentétben, Vock és társai [108] más megközelítést használtak ahhoz, hogy a feldolgozni kívánt pontok számosságát csökkentsék. Ők a 3D pontfelhőn élpontokat detektáltak és csak azokat vették figyelembe a későbbi számításaik során. A tapasztalataink szerint az élpontok használata jobb illesztéseket tud eredményezni bizonyos esetekben. Azonban az éldetektálással való kulcspontkiválasztás hátránya, hogy egyes objektum esetében az élpontok távol esnek egymástól (pl. egy pohár objektum esetében alul és fent helyezkednek el élpontok, középen pedig nem), így a folyamatunk későbbi lépésében használt sűrűség alapú klaszterezést ez is rossz eredményre vezetheti.

### 5.4.3. Jellegzetességleírók kiszámolása

A javasolt folyamatunknak, ahogy más jellegzetességleíró alapú regisztrációs folyamatoknak is, szüksége van jellegzetességvektorok kiszámításához a kiválasztott kulcspontok környezete alapján. A jellegzetességleíró algoritmusok egy jellegzetességvektort adnak meg a bemenetként megadott pont és annak környezete alapján. Ezt a vektort használhatjuk később, hogy a helyszínfelhő és a lekérdezésfelhő hasonló pontjai között megfeleltetéseket találjunk. Pl., ha van egy sarokpontunk a lekérdezésfelhőben, akkor a jellegzetességvektorokat felhasználhatjuk arra, hogy megtaláljuk a hasonló sarokpontokat a helyszínfelhőben. Számos különböző jellegzetességleíró létezik, amelyeket 3D pontfelhők pontjainak leírására készítettek. Az utóbbi években megjelent cikkek alapján, amelyek szemlézik és összehasonlítják a különböző módszereket, a választásunk a Fast Point Feature Histogramra (FPFH [42]) esett. Az FPFH módszer, a szerzői által ajánlott paraméterezés szerint 33 dimenziós jellegzetességvektort készít, amely más módszerekhez képest alacsonynak tekinthető (pl. a SHOT leíró 352 dimenziós). Egy FPFH jellegzetességleíró kiszámítása gyors, és az összehasonlítások alapján leíróképessége a legjobbak között van.

Azonban, a 33 dimenzió még mindig túl sok ahhoz, hogy hatékony lekérdezéseket



a) b) c)

5.4. ábra: (a) A $k$-NN összekapcsolás után maradt kulcspontok (b) A DBSCAN klaszterezés után kiválasztott kulcspontok (zöld: valós előforduláson lévő pontok, piros: nem valós előforduláson lévő pontok) (c) A minimális súlyú párosítás során kapott megfeleltetésekből számolt transzformáció által transzformált lekérdezésfelhő.

tudjunk végrehajtani az adatbázisban. Prakhya és társai szerint a főkomponens-analízis (PCA) módszer segítségével a jellegzetességvektorok hossza csökkenthető úgy, hogy a leíróképességük ne csökkenjen számottevően [85]. Ezért a javasolt folyamatunk főkomponens analízist használ a jellegzetességvektor dimenziószámának a csökkentésére. A főkomponens analízis után kiválasztjuk az 5 legnagyobb sajátértékhez tartozó sajátvektort, és transzformáció után 5 dimenzióra csökkentjük a jellegzetességvektorokat. Az 5.2/(c). ábra egy színtérben illusztrálja az első három főkomponens értékét. A pontok színeit úgy kaptuk meg, hogy egy szín RGB kódjához tartozó piros, zöld és kék értékek megfelelnek az első, második és harmadik főkomponens értékeinek a $[0, 1]$ intervallumra normalizálva. Fontos megjegyezni, hogy az ábrán az egész pontfelhőt ábrázoltuk a jobb megjelenítés érdekében, azonban a folyamat során csak a kiválasztott kulcspontokra számoljuk ki a jellegzetességvektorokat, és csak azok jellegzetességvektorait tároljuk az adatbázisban. Az ábrát megvizsgálva észrevehető, hogy az első három főkomponenshez tartozó értékek képesek különbséget tenni a sík-, él- és sarokpontok között. Az 5.3. ábra megmutatja ugyanezt egy doboz objektumot reprezentáló pontfelhő esetében: (a) az objektum teljes 3D pontfelhője, (b) piros színnel jelölve láthatóak a pontfelhőn kiválasztott kulcspontok, (c) a főkomponensek alapján kiszínezett lekérdezésfelhő.

Vock és társai [108] 4 dimenziós pontpár jellegzetességvektort használtak a pontfelhők közötti megfeleltetések létrehozásához. 4 dimenzió esetében nem szükséges főkomponens-analízist használni, azonban az ő megközelítésüknek más hátrányai vannak.

### 5.4.4. Adatbázis előkészítés

A Generalized Search Tree (GiST - általánosított keresőfa), ahogy a neve is utal rá egy olyan sablon (vagy struktúra), amely lehetővé teszi keresőfák implementálását, mint a B-fa vagy az R-fa. A legismertebb implementációja a PostgreSQL releciós adatbázis-kezelő rendszerben készült el. A GiST használatával lehetséges indexstruktúrát építeni tetszőleges, saját típusú adatokra, és az indexstruktúrához szükséges ún. támogató függvényeket (support functions) is elkészíthetjük (vannak köztük kötelezőek és opcionálisak is). A GiST-t használva lehetőségünk van felkészülni olyan lekérdezések megválaszolására is, amelyek terület-specifikusak. Egy ún. operator class [1] (operátor osztály) elkészítésekor megadhajtuk, hogy a felhasználó által megadott adattípushoz milyen támogató függvények tartoznak, és a GiST segíthet a lekérdezések gyorsításában is.

Mind a pontfelhőkben található 3D pontok, mind pedig a pontok jellegzetességvektorai a `cube` adattípusban [2] kerülnek tárolásra. Ez az adattípus számos hasznos függvényt tartalma és többdimenziós pontot vagy kockát reprezentál. Számunkra a két legfontosabb operátor a `@>`, amely megmondja, hogy egy kocka tartalmaz-e egy másik kokckát és a `<->`, amely megadja az Euklideszi távolságot két kokcka között. Továbbá, egy GiST operátor osztály is implementálva van a `cube` adattípushoz, amely használható a fent említett két operátorral a `WHERE` és `ORDER BY` klózokban, így képes a legközelebbi szomszéd keresések gyors végrehajtására. A PostgreSQL `cube` kiterjesztésének a GiST implementációja hagyományos R-fának tekinthető többdimenziós kockákra. Érdemes megemlíteni, hogy a PostgreSQL adatbázis-kezelő rendszernek létezik egy `Pointcloud`[3] nevű kiterjesztése is, amely szintén használható pontfelhő adatok tárolásához. Az általunk is használt sima tábla modellel ellentétben a `Pointcloud` kiterjesztésben a pontok blokkokban vannak tárolva, amelyeket felhasználva index készíthető a térbeli adatokon. Azonban más munkák megmutatták [96], hogy a blokkalapú megközelítésnek magas számítási többletköltsége van a legtöbb felhasználási esetben. A `Pointcloud` kiterjesztés PostGIS integrációval támogatja a a hagyományos R-fa indexelést is [4]. Mi úgy látjuk, hogy a szempontunkból nem lenne előnye a `Pointcloud` kiterjesztés használatának, így a munkánkban a sima tábla modellt használjuk és a `cube` adattípus implementált függvényeit.

A helyszínfelhő minden pontja a `scene_points_table (id number, coords cube)` táblában kerül tárolásra, míg a jellegzetességvektorai a helyszínfelhő kulcspontjainak a

[1] See `https://www.postgresql.org/docs/current/sql-createopclass.html` és `https://www.postgresql.org/docs/current/xindex.html`

[2] `https://www.postgresql.org/docs/current/cube.html`

[3] `https://github.com/pgpointcloud/pointcloud`

[4] `https://postgis.net/workshops/postgis-intro/indexing.html`

`scene_keypoints_table (p_id number, feats cube)` táblában, ahol az `p_id` oszlop idegen kulcs megszorítás, amely a `scene_points_table` tábla `id` oszlopára hivatkozik. A $k$-legközelebbi szomszéd ($k$-NN) összekapcsolások gyorsításához GiST indexstruktúrát készítünk a `scene_points_table` tábla `coords` oszlopára és a `scene_keypoints_table` tábla `feats` oszlopára. a lekérdezésfelhőkhöz szintén elkészítünk két táblát, `query_points_table (id number, coords cube)` és `query_keypoints_table (p_id number, feats cube)` a megfelelő GiST indexstruktúrákkal. Fontos megjegyezni, hogy a `query_points_table` táblában a kezdeti transzformáció utáni felhő kerül tárolásra, mielőtt még a finomított transzformációt kiszámítottuk volna (lásd lentebb).

### 5.4.5. Megfeleltetések keresése

A helyszínfelhőt tároló tábla (`scene_keypoints_table`) és a lekérdezés felhőt tároló tábla (`query_keypoints_table`) közötti $k$-NN összekapcsolás művelet elvégzéséhez elkészítettük a `get_feat_neighbors` nevű tárolt eljárást. A tárolj eljárás egy virtuális táblával tér vissza, melynek minden sora a `(qf_id int, sf_id int, nn_idx int)` értékeket tartalmazza, ahol

- `qf_id` a lekérdezésfelhő egy pontja,
- `sf_id` a helyszínfelhő egy pontja,
- `nn_idx` a helyszínfelhő pontjának sorszámát reprezentálja a lekérdezésfelhő pontjának legközelebb szomszédai között.

Jegyezzük meg, hogy a `get_feat_neighbors` eljárás végigmegy a `query_keypoints_table` minden során és egy ciklusban megkeresi a $k$ legközelebbi szomszédját az adott sor `feats` értékeinek, az `<->` operátor használatával az `ORDER BY` klózban és a `LIMIT` $k$ klóz használatával kihasználja a GiST által nyújtott előnyöket. Egyébként, ha a fent említett ciklust elhagynánk és valódi összekapcsolást hajtanánk végre a táblákon a PostgreSQL lekérdezésoptimalizálója lehet szekvenciális szkennelést alkalmazna.

A $k$-NN összekapcsolás után klaszterezést végzünk a lekérdezésfelhő kulcspontjainak legközelebbi szomszédjait bemeneti adatként használva. A klaszterezést a DBSCAN sűrűségalapú klaszterező módszerrel végezzük a pontok $X$, $Y$ és $Z$ koordinátáit használva, azok Euklideszi távolsága alapján. Az algoritmus egyik legfontosabb paramétere az $\varepsilon$ (*epsilon*). Ezzel megadhatunk egy távolságot, amely egy pont szomszédságát adja meg. Alapvetően ez a paraméter dönti el, hogy mely pontok kerülnek azonos klaszterbe. Mivel azt szeretnénk, hogy a klaszterezés után minden klaszter egy lekérdezésfelhő előfordulásnak feleljen

meg, ezért a kulcspontok nem lehetnek egymástól távolabb, mint az $\varepsilon$ értéke. Ez az oka annak, hogy miért van szükségünk normális eloszlású véletlen mintavételezésre a széles körben használt kulcspontdetektáló algoritmusok helyett, amelyek gyakran egymástól $\varepsilon$ távolságtól messzebb lévő pontokat választanak ki kulcspontként. A klaszterezés után olyan klasztereket kapunk, amelyek mindegyikében a pontok hasonlóak a lekérdezésfelhő pontjaihoz a jellegzetességleíróik alapján. Mindegyik klaszter egy lehetséges előfordulása a lekérdezésfelhőnek.

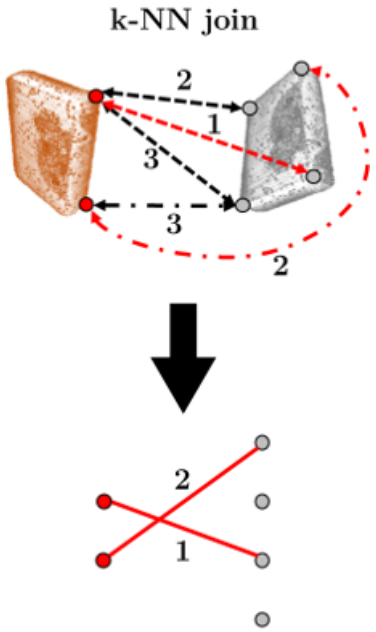
A $k$-NN összekapcsolás és a klaszterezés után a lekérdezésfelhő kulcspontjai és egy klaszterben lévő pontok közötti legjobb megfeleltetések megtalálásának érdekében egy $(V_s, V_q, E)$ páros gráfot hozunk létre, ahol $V_s$ a helyszínfelhő kulcspontjait, a $V_q$ pedig a lekérdezésfelhő kulcspontjait reprezentálja. Akkor és csakis akkor létezik $e$ él $w_e$ súllyal az $sn \in V_s$ és $qn \in V_q$ csúcsok között, ha az $sn$ által reprezentált helyszínfelhő pontjának jellegzetességvektora a $qn$ által reprezentált lekérdezésfelhő pontjának jellegzetességvektorának a $w_e$-edik legközelebbi szomszédja. Ennek az illusztrációja látható az 5.5. ábrán. A $w_e$ élsúly 1 és $k$ közötti értékeket vehet fel. Minimális súlyú párosítás kereséséhez a magyar módszert használjuk [10]. A legjobb jellegzetességleíró párosítás megkeresése minimális súlyú párosítással páros gráfokon nem teljesen új ötlet, hasonló megoldást már alkalmaztak korábban [97]. A párosítás során kapott élek lesznek az adott klaszterhez tartozó megfeleltetések.

### 5.4.6. Transzformációk becslése

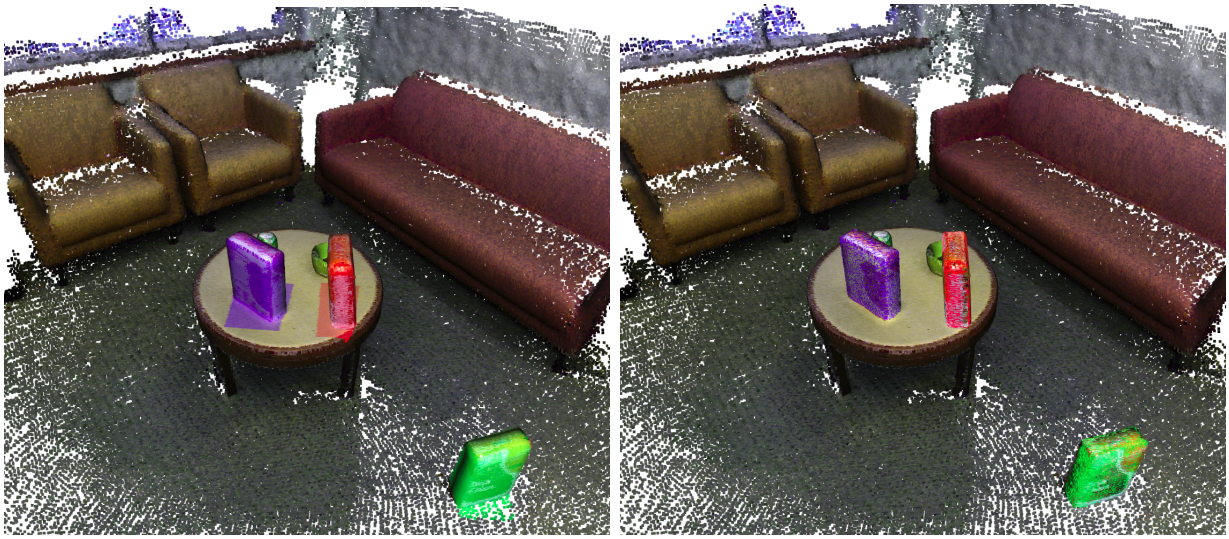
Az előző lépés után rendelkezésünkre áll annyi megfeleltetés halmaz, ahány klasztert talált a sűrűség alapú klaszterezés. Ezután a megfeleltetések alapján Kabsch [14] algoritmusának a segítségével megkapjuk a transzformációkat [5]. Az így kapott transzformációkat hívjuk kezdeti vagy durva transzformációknak. Ezeket a transzformációs mátrixokat arra használjuk, hogy a lekérdezésfelhőt transzformáljuk és eltároljuk az adatbázis `query_points_table` táblájában. A 5.4. ábra bemutatja az eredményét ezeknek a transzformációknak. Az ábrán látható, hogy a lekérdezésfelhő nem illeszkedik tökéletesen a helyszínfelhőben talált megfeleltetésekre. Általában, a kezdeti transzformációk, amelyeket a megfeleltetések alapján kapunk nem képesek nagyon pontos illesztésre. Ezért ahhoz, hogy megkapjuk a végső transzformációt iteratív módszerrel finomítjuk az illesztést [23].

A módszerünk, a kezdeti transzformációk finomításához a széles körben ismert iteratív módszert, az Iterative Closest Pointot (ICP) használja [22]. Az ICP célja, hogy

[5] A pygeostat Python csomag `affine_matrix_from_points(...)` függvény implementációját használtuk




5.5. ábra: Páros gráf minimális súlyú párosítása. Piros csúcsok: a lekérdezésfelhő kulcspontjai. Szürke csúcsok: a helyszínfelhő egy klaszterében lévő kulcspontok. Akkor van él egy piros és egy szürke csúcs között, ha a szürke csúcsokhoz tartozó jellegzetességleírók a piros csúcsokhoz tartozó jellegzetességleírók $k$ szomszédságában vannak. A súlya az élnek *1* ha a szürke csúcs az első legközelebbi szomszédja, a súly $i$, ha a szürke csúcs a $i$. legközelebbi szomszédja a piros csúcsnak a jellegzetességleírók alapján. Az ábrán a két piros él lesz kiválasztva megfeleltetésként.



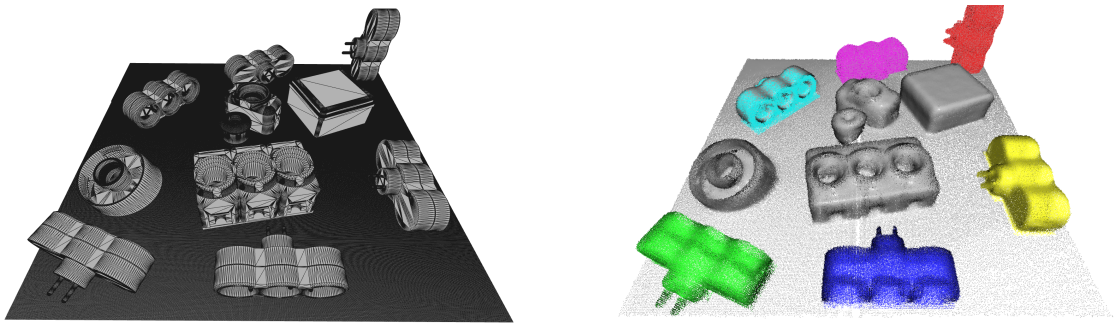
a) b)

5.6. ábra: (a) A transzformált lekérdezésfelhők környezetei a helyszínfelhőben (b) A finomított transzformációk eredményei (ICP utáni állapot)

minimalizálja a távolságot a két pontfelhő pontjai között. Az algoritmus két pontfelhőt kap bemenetként és kimenetként megadja a finomított transzformációt. Az algoritmus három lépésből áll: (1) az egyik felhő minden pontjához megkeresi a legközelebbi pontot a másik felhő pontjai közül, (2) az algoritmus transzformációt készít, amely minimalizálja a távolságot az előző lépésben meghatározott megfeleltetések pontjai között (ezt a távolságot különböző módszerekkel lehet meghatározni, és nagyban befolyásolja az algoritmus eredményét), (3) alkalmazza a transzformációt és elölről kezdi a folyamatot a pontok egymáshoz rendelésével. Az algoritmusnak megadhatjuk, hogy mennyi legyen a maximális iterációszám vagy az a hibahatár, ami után megállhat.

Az eredeti ICP algoritmus megjelenése óta számos továbbfejlesztése jelent meg [32, 78, 33, 89]. Az egyik legnagyobb probléma az eredeti algoritmussal, hogy lokális optimumban ragadhat. Mivel az ICP ma is széles körben használt, számos implementációja létezik és a továbbfejlesztéseknek köszönhetően jól teljesít, úgy döntöttünk, hogy a folyamatunkban ezt a módszert használjuk. A munkánkban az eredeti pont-pont távolságot használó módszer [23] helyett, a pont-sík távolságot minimalizáló algoritmust választottuk [22]. Rusinkiewicz és társa [31] megmutatták, hogy a pont-sík távolságot használó ICP gyorsabban konvergál, mint a pont-pont távolságot használó. Pulli [28] munkájában alátámasztotta a korábbi eredményt és megmutatta, hogy a pont-sík távolságot használó ICP kevésbé érzékeny a hibás megfeleltetésekre és tizedannyi iterációra van szüksége.

Mivel az ICP algoritmushoz szükséges a bemeneti pontfelhőket a memóriába betölteni, ezért a teljes helyszínfelhő használata nem lehetséges és nem is lenne hatékony. Ezért kivágjuk azt a részt a helyszínfelhőből, amely a kezdeti transzformációval transzformált lekérdezésfelhő környezetében található a (`scene_points_table` és `query_points_table`), a `coords` oszlopra készített index segítségével és az `@>` operátor használatával. A lekérdezésfelhő környezetének a meghatározásához meghatározzuk a felhő befoglaló dobozát és ezt kiterjesztjük. A kiterjesztéshez a befoglaló doboz átmérőjét növeljük meg egy felhasználó által megadott paraméterrel ($\mu$). Minél nagyobb a paraméter annál biztosabb, hogy nem hagyunk el semmilyen fontos részt a helyszínfelhőből, azonban annál lassabb lehet a futási idő. Miután kivágtuk a kiterjesztett befoglaló doboznak megfelelő részt a helyszínfelhőből az ICP algoritmust alkalmazzuk a kivágott részt és a lekérdezésfelhőt megadva bementi adatként. Kimenetként megkapjuk a finomított transzformációt. Jegyezzük meg, hogy ez a lépés végrehajtódik minden klaszterre, amit a sűrűségalapú klaszterezés közben kaptunk.



a) b)

5.7. ábra: (a) A T-LESS adathalmazból összeállított helyszínfelhő CAD modellje 6 darab lekérdezésfelhő előfordulással, különböző pozíciókban (b) A megtalált előfordulások a helyszínfelhőben.

## 5.5. Eredmények

A kiértékelést PostgreSQL 10.12 adatbázis-kezelő rendszeren végeztük, amely egy virtuális gépen futott (Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz (4 mag) és 11GB RAM) Ubuntu 18.04.4 operációs rendszert használva.

### 5.5.1. Kiértékelés

A kiértékeléshez használt két adathalmaz tulajdonságai a 5.1. táblázatban vannak összefoglalva. Az első adathalmaz a "RGB-D Scenes Dataset v2" [61] egy valós adathalmaz. Több helyszínt is tartalmaz, amelyek mindegyike szobáról készült felvételek összessége. A helyszínek különböző beltéri berendezéseket tartalmaznak (szék, kanapé, asztal) és kisebb hétköznapi objektumokat (bögre, doboz stb.). Ez az adathalmaz kiváló a módszer robusztusságának a kiértékelése, mivel valós adat, annak velejáróival együtt. Sok kiugró értéket tartalmaz, szemmel is észrevehető mértékű zajt, különböző nézőpontokból készített felvételek összeillesztése során készült és a felhő sűrűsége is változik. Erre az adathalmazra röviden *indoor*-ként hivatkozunk. Az *indoor* helyszínen a lekérdezésfelhőnek három előfordulása van.

A másik adathalmaz a T-LESS [84]. A T-LESS adathalmaz ipari objektumok számítógépes modelljeit tartalmazza. Ezekből hat, méretükben és formájukban különböző objektumokat választottunk ki, egy síkra helyeztük őket a MeshLab szoftver segítségével. A 5.7/a ábrán látható, hogy a lekérdezésfelhő által reprezentált objektum hatszor szerepel a helyszínen, különböző pozíciókban (különböző oldalon fekve, elforgatva, élére állítva stb.). A T-LESS modellekből összeállított helyszínből a Point Cloud Library (PCL [52]) könyvtárban implementált **mesh2pcd** virtuális szkenner segítségével 3D pontfelhőt

készítettünk. A **mesh2pcd** virtuális szkenner a modellek körül elhelyez virtuális kamerákat, amelyek a modell közepére néznek és sugárkövetés által pontokat mintavételez a modellek felületéről (így szimulálva a valóságban használt Time-of-Flight szenzorokat). A **mesh2pcd** úgy lett elkészítve, hogy az eredményül kapott felhő szintén tartalmazzon bizonyos fokú zajt. Így bár az elkészített adat szintetikus, a különböző pontokról felvett felvételek összeillesztése miatt itt is változik a sűrűség és zajosság is megjelenik, csakúgy, mint a valós adatok esetén. A T-LESS adatokból elkészített helyszínfelhő a 5.7/b ábrán látható. Fontos megjegyezni, hogy a fent bemutatott adathalmazok bár adatbázisban voltak tárolva, méretük nem érte el azt a szintet, hogy ne férjen be a kiértékeléshez használt konfiguráció memóriájába.

Amikor a fentebb bemutatok lépések véget érnek minden klaszterhez kapunk egy finomított transzformációt. Azonban nem lehetünk biztosak abban, hogy minden klaszter egy valós előfordulása a lekérdezésfelhőnek. Előfordulhat, hogy egyes klaszterek csak egy nagyon hasonló objektumot reprezentálnak. Ezért a klasztereket két csoportba soroljuk: igaz pozitív és hamis pozitív klaszterek. Igaz pozitívnak hívunk egy klasztert, ha a klaszterbe tartozó pontok a lekérdezésfelhő egy valós előfordulásának a részei. Egyébként a klaszter hamis pozitív. Ahhoz, hogy elvégezzük a klaszterek ily módon történő felosztását meg kell vizsgálnunk a finomított transzformáció utáni lekérdezésfelhő és a helyszínfelhő átfedésének az arányát. A 5.2 táblázat tartalmazza az átfedések arányát különböző klaszterek esetében. Az átfedést úgy számoljuk, hogy megnézzük a távolságot a lekérdezésfelhő pontjai és azok helyszínfelhőből vett legközelebbi szomszédjai között. Ha ez a távolság kisebb, mint egy határérték, akkor az adott pont a lekérdezésfelhőből átfedő pontnak számít. Egy lekérdezésfelhő és a helyszín közötti átfedés meghatározásához elég megnéznünk, hogy a lekérdezésfelhő pontjainak mekkora része az átfedő pont. Az átfedéshez használt határérték meghatározásához szintén felhasználjuk a korábban kiszámolt $\rho$ értéket. A kiértékelés közben ezt a határértéket $\rho \cdot 2$ értékre állítottuk be. ha egy lekérdezésfelhő által reprezentált objektum teljes egészében megtalálható a helyszínen, és a finomított transzformáció tökéletesen egymásra illeszti a felhőket, akkor az átfedés 100%-ot is elérheti. Vock és társai [108] sok transzformáció hipotézist készítenek, ezért számukra a gyors validáció nélkülözhetetlen. Ők egy gyors, voxel alapú megközelítést alkalmaznak ahhoz, hogy egy transzformáció jóságát meghatározzák. A mi esetünkben a transzformáció hipotézisek száma egyenlő a klaszterek számával, amely pedig közel áll a helyszínen előforduló lekérdezésfelhők számával, így a gyorsabb, de kevésbé pontos validációs módszerekre nincs szükségünk.

Az átfedést megnézzük a kezdeti transzformáció után és a finomított transzformáció

után is. A kezdeti transzformáció után és még a finomított transzformáció előtt különbséget tenni az igaz pozitív és hamis pozitív klaszterek között nagyon nehéz feladat. Ugyanis előfordulhat, hogy a helyszínfelhőn a lekérdezésfelhőhöz nagyon hasonló objektum is szerepel, így az átfedés közöttük magas lesz. Az általunk használt adathalmazok esetében az alacsony átfedés értékekkel ($< 35\%$) rendelkező klasztereket még azelőtt ki tudjuk szűrni, hogy a költséges iteratív módszert alkalmaznánk. Azonban fontos megjegyezni, hogy egy új, korábban nem látott adathalmazra ezt a feltételt nem feltétlenül alkalmazhatjuk.

A tapasztalataink alapján, ha egy klaszter igaz pozitív, akkor a finomított transzformáció után az átfedés 90%-tól nagyobb lesz. A kiértékelés során a T-LESS adathalmaz esetén egy igaz pozitív klaszternél 89,5% volt az átfedés, ezért érdemes lehet a szűrés során is bevezetni egy hibahatárt. Ezen feltételek alapján ki tudjuk szűrni a hamis pozitív klasztereket és az azokhoz tartozó transzformációkat és előfordulásokat az eredmények közül.

### 5.5.2. A $k$-NN összekapcsolási művelet teljesítménye

Az 5.8. ábra megmutatja a $k$-NN összekapcsolás futási idejét különböző $k$ értékekre. Az ábrán látható görbe logaritmikus viselkedést mutat alacsony $k$ értékek esetén a GiST használatának köszönhetően, míg lineáris növekedést nagy $k$ értékeknél. Feltehetőleg, ezt azért látjuk mivel nagyobb $k$ értékek esetén az indexstruktúra gyorsítótárba kerül, így a `get_feat_neighbors` függvényt a `LIMIT` $k$ klóz dominálja, amely lineárisan nő a $k$ értékével. A méréshez olyan véletlen adathalmazt használtunk, amely 100,000 helyszínfelhő kulcspontot és 10,000 lekérdezésfelhő kulcspontot tartalmazott. Fontos megjegyezni, ha az adathalmaz sokkal nagyobb területet fed le (pl. egy teljes épület), akkor a keresés párhuzamosítható.

### 5.5.3. Skálázhatóság vizsgálata

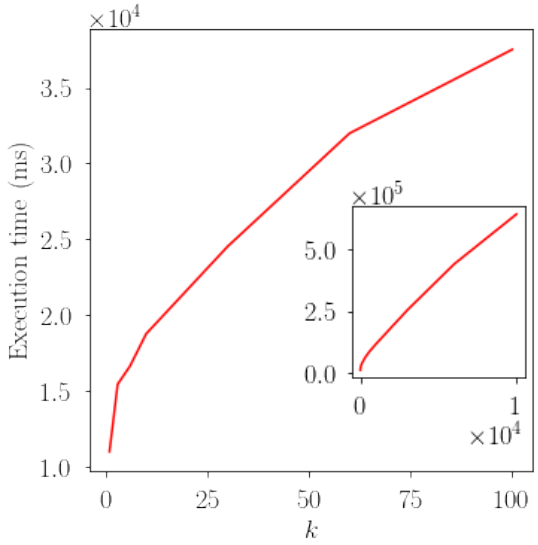
Ebben a fejezetben bemutatjuk, hogyan viselkedik a sablonillesztési folyamatunk a helyszínfelhő méretének növelésével. Ehhez elkészítettük a helyszínfelhők bővített változatait, amelyekben a pontok számát kétszeresére és négyszeresére növeltük az eredeti helyszínhez képest. A célja ezeknek a méréseknek az volt, hogy megvizsgáljuk a folyamat különböző részeinek futási idejét a pontfelhőben lévő pontok számosságának növelése esetén. A megnövelt pontszámú felhők hasonlóak a korábban bemutatott helyszínfelhőhöz (pl. a lekérdezésfelhő ugyanannyiszor fordul elő benne), viszont nagyobb lett a sík, amelyen az objektumok fekszenek és megnőtt a síkon lévő objektumok száma is. A futási idő mérése közben is meg kell különböztetnünk az offline és online fázisokat. Az offline fázis során az

| **A pontfelhő elnevezése** | **Pontok száma a pontfelhőben** | **Pontok száma előfeldolgozás után** | **Kulcspontok száma** | **A lekérdezésfelhő előfordulásainak a száma** |
|---|---|---|---|---|
| Indoor helyszín | 1 108 688 | 537 428 | 10 811 | 3 |
| Indoor objektum | 75 911 | 27 358 | 538 | - |
| TLESS helyszín | 771 781 | 771 781 | 38 904 | 6 |
| TLESS objektum | 57 565 | 57 565 | 2885 | - |

5.1. táblázat: A kiértékeléshez használt adathalmazok leírása. Az *indoor* pontfelhő valós adathalmaz, sok kiugró ponttal és változó sűrűséggel. Így az előfeldolgozás során sok pont eltávolításra kerül. A negyedik oszlop mutatja a kulcspontok számát, az utolsó pedig a lekérdezésfelhő előfordulásainak a számát a hozzá tartozó helyszínfelhőben.

| **Adathalmaz** | **Klaszter név** | **Átfedés (ICP előtt)** | **Átfedés (ICP után)** |
|---|---|---|---|
| Indoor | Klaszter #1 | 84% | 100% |
| | Klaszter #2 | 35% | 100% |
| | Klaszter #3 | 40% | 99% |
| | Hamis pozitív klaszterek | 8% - 25% | 0%-50% |
| T-LESS | Klaszter #1 | 70% | 100% |
| | Klaszter #2 | 83% | 100% |
| | Klaszter #3 | 70% | 100% |
| | Klaszter #4 | 85% | 100% |
| | Klaszter #5 | 53% | 90% |
| | Klaszter #6 | 81% | 100% |
| | Hamis pozitív klaszterek | 18% - 41% | 32% - 50% |

5.2. táblázat: Az átfedések arányai a lekérdezésfelhő és a megfelelő klaszterhez tartozó környezet között az ICP-vel való finomítás előtt és után. A hamis pozitív klasztereket egy sorba aggregáltuk, az igaz pozitív klasztereket (valós előfordulások) külön sorban tüntettük fel. Az adatok azt mutatják, hogy az ICP-vel való finomítás előtt az igaz pozitív klaszterek esetében magasabb az átfedés aránya, mint a hamis pozitívoknál. Ez a különbség jóval nagyobbá válik a finomítás után.




5.8. ábra: A $k$-NN összekapcsolási művelet futási ideje a $k$ függvényében.

elvégzendő algoritmusokat a helyszínfelhőre alkalmazzuk, az online fázis esetében pedig a lekérdezésfelhő előfeldolgozása és maga az előfordulások keresése zajlik. Az offline fázis futási idejének összetevői a következők:

1. Helyszínfelhő betöltése a memóriába a kulcspontok kiválasztásához és a jellegzetességleírók kiszámításához (ezt a komplex műveletet nincs lehetőség adatbázisban elvégezni).

2. A helyszínfelhő kulcspontjainak kiválasztása.

3. A jellegzetességleírók kiszámítása a helyszínfelhő kulcspontjaihoz.

4. A helyszínfelhő pontjainak betöltése az adatbázis megfelelő táblájába és a kulcspontokhoz tartozó jellegzetességleírók betöltése. Indexstruktúrák készítése a feltöltött táblákhoz (GiST indexstruktúra a pontok XYZ koordinátáira és a jellegzetességleírókra)

Az online fázis a következő költségekből áll:

1. A lekérdezésfelhő előfeldolgozása (a lépéseket fentebb összefoglaltuk).

2. $k$-NN összekapcsolás a lekérdezésfelhő jellegzetességleírói és a helyszínfelhő jellegzetességleírói között.

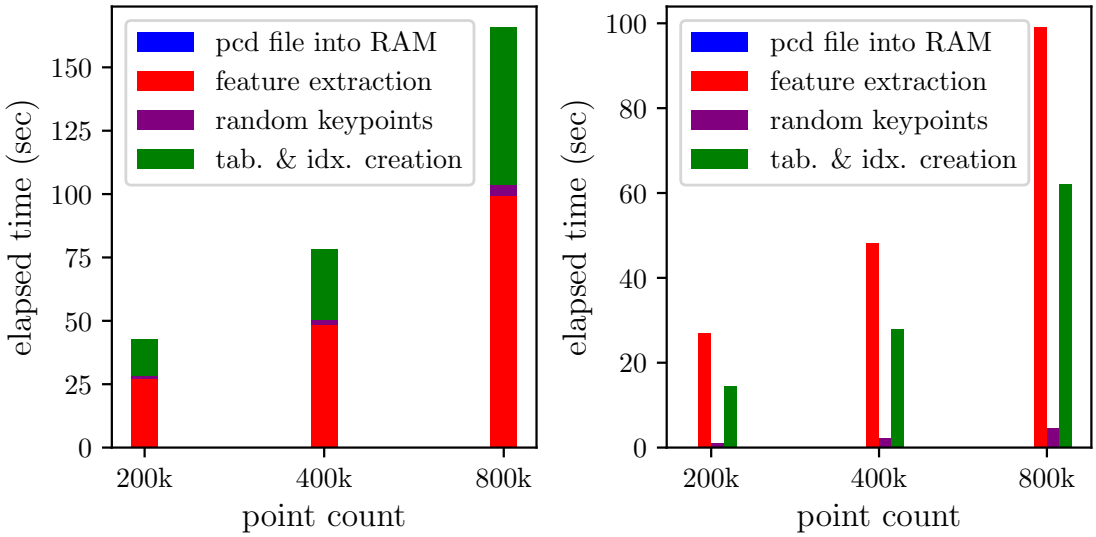
3. DBSCAN klaszterező algoritmus futtatása a $k$-NN összekapcsolás által megadott pontokon (a helyszínfelhő kulcspontjainak a részhalmaza).

4. Minimális súlyú párosítás meghatározása klaszterenként a magyar módszer használatával.

5. Kezdeti transzformációk kiszámítása és végrehajtása a párosítás során kapott megfeleltetések alapján.

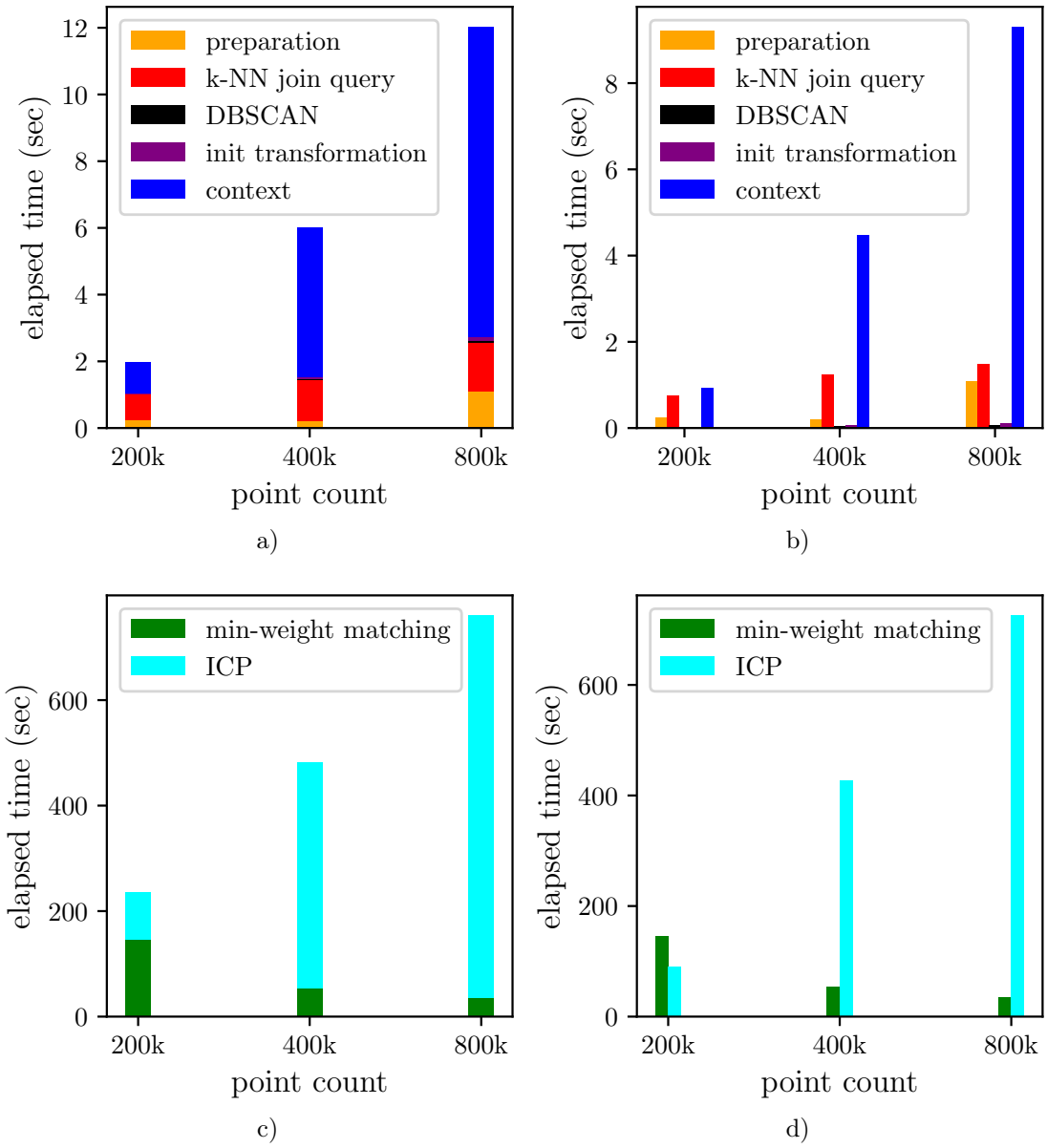
6. A kezdeti transzformációk után kapott transzformált lekérdezésfelhők környezetének meghatározása a helyszínfelhőben (a kiterjesztett befoglaló doboz által meghatározott pontok kiválasztása a helyszínfelhőből)

7. Az ICP algoritmus futtatása a finomított transzformációk kiszámításához.

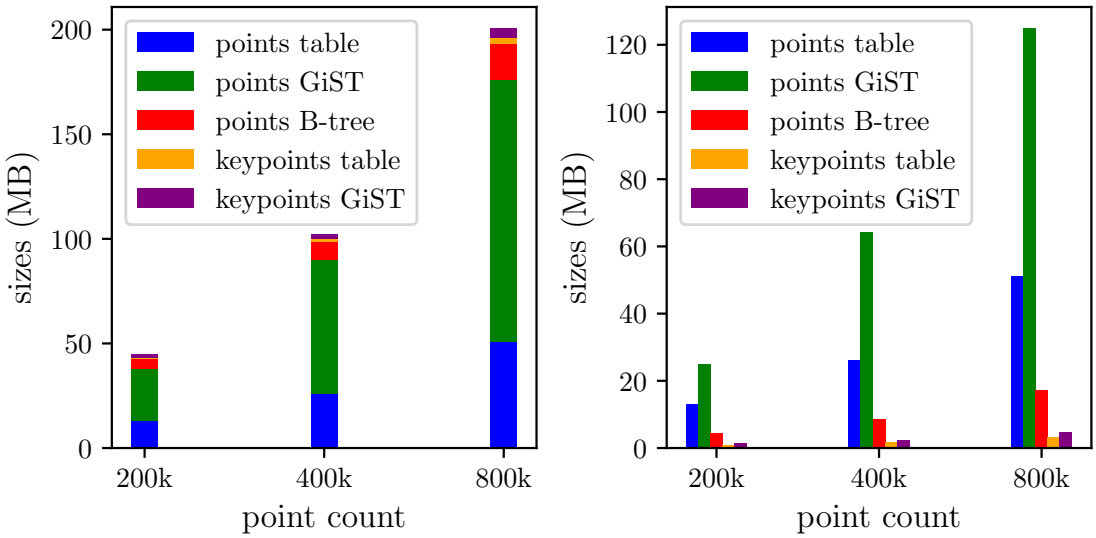



5.9. ábra: Az offline fázis lépéseinek futási idejének skálázódása. A legtöbb időbe a jellegzetességleírók és a táblák feltöltése és az indexstruktúrák készítése kerül.

Az 5.9. ábra az offline fázis futási idejét mutatja különböző méretű helyszínfelhők esetében. Könnyen észrevehető, hogy a legköltségesebb része az offline fázisnak a jellegzetességleírók kiszámolása. Azt is fontos megjegyezni, hogy az ábra alapján a kulcspontok detektálásának a futási ideje elhanyagolható a többi művelethez képest, azonban ez csak azért van így mert véletlenszerűen választjuk azokat. Ha a folyamatunkban kicserélnénk egy valódi kulcspontdetektáló módszerre, akkor annak a futási ideje is nagy részét képezné a teljes időnek. A folyamatunkban jelenleg a jellegzetességleírókat minden pontra kiszámítjuk, mivel az eredeti jellegzetességvektorokat alacsonyabb dimenzióba transzformáljuk a PCA használatával az alacsonyabb tárolási igények és gyorsabb kereshetőség miatt. Ugyanúgy, mint a kulcspontdetektáló módszer esetében, a jelenleg használt jellegzetességleíró és kicserélhető. Ha a jelenlegitől alacsonyabb dimenziójú jellegzetességleíróra




5.10. ábra: Az online fázis lépéseinek futási idejének skálázódása. A felső sor mutatja az alacsonyabb futási időket míg az alsó sor a tőlük két nagyságrenddel nagyobb futási idejű lépéseket (ICP és minimális súlyú párosítás).




5.11. ábra: Az adatbázis elemeinek tárhely igénye. A GiST indexstruktúra igényli a legtöbb tárhelyet a másodlagos tárban.

cseréljük, akkor elegendő csak a kulcspontokra kiszámítani őket. Ezzel felgyorsítható az offline fázis futási ideje, de ez nem szerepelt a céljaink között. Az adatok betöltése az adatbázisba és az indexstruktúrák létrehozása hasonló időt vesz igénybe, mint a jellegzetességleírók kiszámítása a jelenlegi megoldással.

Az online fázisban egyértelműen a megfeleltetések alapján transzformált lekérdezésfelhők környezetének kivágása a helyszínfelhőből a legköltségesebb lépés (5.10 (a) és (b) ábrája). Ez a lépés azonban elkerülhetetlen, mivel az egész felhő használata nem kivitelezhető. A lekérdezésfelhő kiterjesztett befoglaló dobozának megfelelő tartományba eső pontokat a helyszínfelhő pontjait tartalmazó táblából választjuk ki, GiST indexstruktúrát használva. Minél nagyobb a helyszínfelhő, annál költségesebb ez a művelet (a GiST skálázódása a 5.11 ábrán látható). A $k$-NN összekapcsolás költsége nem csak a helyszínfelhő méretétől függ, hanem az adatbázis-kezelő rendszer lapozási módszerétől is.

Az ICP és a minimális súlyú párosítás két nagyságrenddel több időbe kerülnek, mint a korábbi lépések, ezért ezeket a lépéseket külön vizsgáljuk. Az ICP alkalmazásának a futási ideje a kezdeti transzformáció utáni lekérdezésfelhőkkel látható a 5.10 (c) és (d) ábrákon. Az ábrán látható, hogy minél nagyobb a helyszínfelhő, annál több idő lesz az ICP futási ideje. Ennek a magyarázata a következő. Ahogy egyre több pont van a helyszínfelhőben, úgy egyre nagyobb a valószínűsége, hogy olyan pontok is kiválasztásra kerülnek a helyszínfelhőből megfeleltetésként, amelyek nem a lekérdezésfelhő előfordulásában szerepelnek (nem véletlenszerű kulcspontdetektálás esetében a helyzet más lenne). Így jóval több olyan klasztert kapunk a DBSCAN után, amely hamis pozitív klaszter. Ez

azt jelenti, hogy jóval több olyan klaszterünk lesz, amely reménytelen az ICP számára. Az ICP akkor áll le, amikor eléri a maximális iterációszámot vagy egy meghatározott határértéket. Mivel sok klaszter nem valós előfordulás, ezekben az esetekben az ICP eléri a maximális iterációszámot, amely más műveletekhez képest sokkal több ideig tart. Ez megmagyarázza, hogy a helyszínfelhő pontjainak növekedésével miért nő az ICP futási ideje.

További vizsgálataink kimutatták, hogy egy adott klaszterre a Pearson korrelációs együtthatója a lehetséges megfeleltetések számának és a minimális súlyú párosítás futási idejének 0.96196 (p-value: $8.7636 \cdot 10^{-6}$). Más szóval, a megfeleltetések keresésnek az ideje a minimális súlyú párosítás által köbösen skálázódik a pontok számának növekedésével a klaszterben (ha nő a pontszám, nő a párosítás ideje). Egy valós előfordulást magába foglaló klaszter esetében jóval több megfeleltetést várunk, így az ilyen esetekben a párosítás ideje is több lesz, mint nem valós előfordulások esetében. Mivel a felhőket a skálázódási teszthez úgy készítettük el, hogy az objektumok előfordulásainak a sűrűsége ne növekedjen, hanem újabb objektumokat és nagyobb síkot vettünk, ezért a konstans 5%-os kulcspontválasztás miatt, ahogy nő a helyszínfelhő mérete úgy fog egyre kevesebb kulcspont esni egy-egy előfordulásra. Ezért látható a 5.10. ábrán, hogy a pontszám növelésével a párosítás ideje csökken.

5.11. ábra megmutatja a folyamathoz szükséges adatok tárolási igényeit, a pontok számának növekedése mellett. Látható, hogy a GiST indexstruktúra igényli a legtöbb tárhelyet. Egyedül a pontok tárolásához szükséges tárhely közelíti ezt meg. A további tárolandó adatok (pl. kulcspontok táblája, azok indexstruktúrái) méretei elhanyagolhatóak ezekhez képest.

### 5.5.4. Lehetséges továbbfejlesztések

A javasolt folyamatból látható, hogy a jellegzetességleíró alapú sablonillesztési folyamat fontos részei képesek felhasználni az adatbázis-kezelő rendszer előnyeit (pl. $k$-NN összekapcsolás). A DBSCAN klaszterezés szintén elvégezhető lehet adatbázisban, azonban a legjobb tudásunk alapján jelenleg nem létezik implementáció háromdimenziós adatokhoz PostgreSQL-ben. A bemutatott prototípus implementációjában a DBSCAN klaszterezést adatbázison kívül futtattuk.

A kiértékelésünk alapján a javasolt folyamat robusztus a zajra, viszont nem teszteltük a módszerünket olyan esetekre, amikor az objektumok kitakarják egymást vagy nagyon közel állnak egymáshoz.

Az egyik legnagyobb kihívás az algoritmusok paramétereinek megadása: normálvektor

becsléséhez használt sugár, a jellegzetességleírók kiszámításához használt sugár, $k$ paraméter a $k$-NN összekapcsoláshoz, $\varepsilon$ paraméter a DBSCAN klaszterezéshez, $\mu$ a kiterjesztett befoglaló doboz kiszámításához és a határérték az igaz és hamis pozitív klaszterek megkülönböztetéséhez a finomított transzformáció előtt és után. Ezen paraméterek a bemeneti 3D pontfelhők tulajdonságaitól függenek, de nem triviális közöttük az összefüggés. A paraméterek beállításakor érdemes figyelembe venni a pontfelhő sűrűségét, zajosságát, a reprezentált felületek méreteit (pl. egy szoba, egy ház, egy egész utca vagy város) stb. A folyamat elkészítésekor úgy döntöttünk, hogy a paramétereket a pontfelhő sűrűségétől tesszük függővé, annak többszörösére állítjuk be őket, odafigyelve a közöttük lévő megszorításokra (pl. jellegzetességleíró sugarának nagyobbnak kell lennie, mint amit a normálvektorok becsléséhez alkalmazunk). Fontos megjegyezni, hogy az olyan egyszerűen megadhatónak tűnő paraméterek esetében is, mint a jellegzetességleírók sugara nincs széles körben elfogadott megoldás. Egy-egy paraméter esetében megtalálható az optimális megoldás paraméter finomhangolással, azonban ilyen sok paraméter esetében ez nehezen kivitelezhető. Fontos azonban látni, hogy az optimális paraméterek ismerete nélkül is elérhető helyes eredmény, gyakran a futási idő növekedésének az árán.

## 5.6. Összegzés

Ebben a fejezetben bemutattunk egy jellegzetességleíró alapú sablonillesztési folyamatot, amely kihasználja az adatbázis-kezelő rendszerek előnyeit, ezzel lehetővé téve a memóriába nem beférő nagy méretű 3D pontfelhők feldolgozását. Számos lépése a folyamatunknak elvégezhető adatbázisban (pl. $k$-NN összekapcsolás, klaszterezés). A működése a javasolt folyamatunk implementációjának PostgreSQL adatbázis-kezelő rendszert használva kiértékelésre került szimulált és kihívást jelentő valós adatok által is. Ismereteink szerint ez az első olyan folyamat, amely figyelembe veszi az adatbázis-kezelő rendszerek használatát sablonillesztési feladat megoldásához.

A valós idejű alkalmazások néhány másodperces, vagy másodperc alatti időt várnak el. A bemutatott folyamatunk ezt a követelményt nem tudja teljesíteni a jelenlegi formájában. A munkánkban nem törekedtünk a valós idejű alkalmazások kiszolgálására, hanem azt szerettük volna bemutatni, hogyan lehet a sablonillesztési folyamatba az adatbázis-kezelő rendszereket integrálni, hogy nagy pontfelhők esetén is képes legyen jól működni. Meg kell jegyeznünk, hogy a folyamat komponensei nem kifejezetten azzal a céllal voltak kiválasztva, hogy a lekérdezés minél gyorsabban végrehajtódjon. Például, az ICP után kapott finomított transzformáció pontos, azonban maga az ICP sok időt vesz igénybe. Az

ICP maximális iterációszámának csökkentésével gyorsabb futási időket kapunk, cserébe az illesztések pontossága csökkenhet. Vegyük észre, hogy a 5.10. ábra alapján, ha elégséges az alkalmazás szempontjából, hogy az objektum előfordulásait megtaláljuk, de nincs pontos transzformációnk és nem vágjuk ki a helyszínfelhőből a talált előfordulás környezetét, akkor 2 másodperc körüli idővel meg tudjuk válaszolni a lekérdezést.

A folyamat futási ideje tovább csökkenthető, ha az egyes komponenseit párhuzamosan hajtjuk végre. A klaszterezés után kapott klaszterekre párhuzamosan futtatható a minimális súlyú párosítás, és a környezetének kivágása után az ICP-vel való finomított transzformáció becslése is. Jövőbeli terveink között szerepel a folyamat továbbfejlesztése, nagy léptékű pontfelhőkkel való tesztelése.

# 6. fejezet

# Láncolatalapú objektumkeresés

## 6.1. Bevezetés

Az objektumfelismerés célja, hogy egy felvételen képesek legyünk felismerni az ott található objektumokat és meghatározni a helyzetüket [1] (pozíciójukat és orientációjukat). Maga az objektumkeresés, mint feladat értelmezhető kétdimenziós képekre, háromdimenziós pontfelhőkre, de akár kétdimenziós pontfelhőkre is. A feladat számos területen fontos szerepet tölt be. Az önvezető járművek szoftverének elkészítése során fontos, hogy fel tudjuk ismerni a közúti jelzőtáblákat, a közúti forgalom más egyenrangú résztvevőit és a jármű által különösen veszélyeztetett járókelőket. Egy másik tipikus felhasználási eseten az objektumfelismerésnek a robot-objektum interakció [2], melynek során egy robot, vagy egy robotkar és objektumok közötti interakciók (mozgatás, felemelés) megvalósításához meghatározzuk az objektumok helyzetét, így a robot meg tudja találni.

Az objektumfelismerés során előre ismerjük az objektumokat, rendelkezünk valamilyen adattal róluk: CAD modell, 3D pontfelhő, képek stb. A cél az, hogy egy korábban nem látott felvételen megtaláljuk az objektumok előfordulásait. Egy felvételen előfordulhat több különböző objektum és az egyes objektumok is többször szerepelhetnek. Ezen feltételek miatt a sok esetben jól használható iteratív ponthalmaz regisztrációs módszerek önmagukban nem használhatóak.

Az elmúlt évtizedben a legnépszerűbb megoldások lokális jellegzetességleírókat használtak a feladat megoldására (FPFH [42], SI [94], RoPS [57] és mások [66]). A leírók alapján pontok közötti megfeleltetések készíthetőek, amelyekből különböző módszerekkel (pl. RANSAC [15], szavazás-alapú [4, 109]) megkapható az objektumok helyzetei. A lokális jellegzetességleírókat használó jellegzetességleíró alapú regisztrációs folyamatokat

[1] pose
[2] bin picking

az elméleti bevezetőben részletesen bemutattam (2. fejezet). Jegyezzük meg, hogy egy tetszőleges jellegzetességleíró alapú regisztrációs folyamat akkor is alkalmas lehet az objektumfelismerés feladatát megoldani, ha nem speciálisan erre a feladatra készült. Ha egy objektum több előfordulással rendelkezik a felvételen, akkor elég a regisztrációt egymás után többször lefuttatni és minden alkalommal, amikor talált egy előfordulást, az ahhoz tartozó pontokat kivágni a pontfelhőből. Ha valamelyik lépésben nem talál előfordulást, akkor befejezhetjük a folyamatot. Az ilyen jellegű folyamatok végén kapott durva transzformáció általában iteratív regisztrációs módszerek kezdeti transzformációjaként szolgál, amelyek végül egy finomított, pontosabb transzformációt adnak meg kimenetül. Az általam javasolt módszere neve CatMat [3].

A hozzájárulásaim a témához a következők:

- Éldetektáló algoritmus és voxelrács non-maximum suppression (NMS) használatával egy súlyozott gráfot készítek az objektum 3D pontfelhője alapján.
- Az elkészített gráfon olyan útvonalat keresek, amely az objektum fontos részeit érinti és képes a felület jól megkülönböztethető részeit jellemezni.
- Lokális jellegzetességleíró megfeleltetéssel és geometriai megszorításokkal az objektumon létrehozott útvonalhoz hasonló pontláncolatokat keresek a helyszínfelhőkben az objektumok helyzetének a meghatározásához.
- A javasolt módszer egy gyors, voxelrács alapú validációt végez és robusztus a zajra.

## 6.2. Kapcsolódó munkák

A jellegzetességalapú regisztrációs folyamatok sok esetben jól működnek, azonban bizonyos kihívásokkal nehezen küzdenek meg. Ilyen kihívás lehet, ha egy felvételen az objektumok közel vannak egymáshoz (pl. "egymásra hányva" [4]). Ezekben az esetekben a lokális jellegzetességleírók kiszámításakor a sugár paraméter által meghatározott környezetbe belelóghatnak más objektumok részei. Ez olyan jellegzetességleírókat eredményezhet, amelyek teljesen különböznek azoktól, amelyeket az egyes különálló objektumokon határozunk meg.

A fent vázolt problémák miatt a jellegzetességleíró alapú regisztrációs folyamatok mellett megjelentek olyan munkák is, amelyek más megközelítést használtak a feladat megoldására. Drost et al. [45] egy pontpár jellegzetességleíró módszert javasolt. Míg egy

[3] catenarian matching
[4] clutter

hagyományos lokális jellegzetességleíró egy pont környezetébe eső pontok alapján egy pontot jellemez, addig a pontpár jellegzetességleírók a pontfelhő pontjaiból pontpárokat képeznek, és arra számolnak ki különböző jellemzőket. Így ebben az esetben egy jellegzetességleíró nem egy ponthoz tartozik jellemez hanem mindig egy pontpárhoz. Könnyen látható, hogy míg a lokális jellegzetességleírókból egy $N$ pontból álló felhő esetén $N$ darabra van szükség, addig pontpár jellegzetességleíróból $(N * (N-1))/2$ (ha feltesszük, hogy a pontok sorrendje nem számít és minden pontpárra kiszámoljuk). A pontpárokhoz hasonlóan felmerült a ponthármasok használatának lehetősége is. A módszereket összehasonlító munkák szerint azonban ha az adathalmaz minősége megfelelő, a pontpárok jobb teljesítményt tudnak elérni, míg a ponthármasok nagyobb zaj esetén teljesítenek jobban [49]. A pontpárokat használó módszerek jó eredményeket értek el, és sokan továbbfejlesztették [108, 74, 64] az eredeti elképzelést. A továbbfejlesztéseknek köszönhetően a pontpár jellegzetességleírókat használó módszerek jelenleg is felveszik a versenyt az újabb megközelítésekkel.

Az elmúlt években megjelentek, az korábbiaktól merőben eltérő megoldások is [116, 111, 81]. A témakör nagyon szerteágazó és a különböző megközelítések között szoros verseny van. Megjelentek kiváló összefoglaló cikkek, amelyek összegyűjtötték a legismertebb eredményeket [115]. A területen egy fontos mérföldkő volt a BOP Challange [112] (korábban SIXD) megjelenése, ahol a feladat az objektumfelismerés volt, és ahol a legjobb módszerek versenyeztek különböző jellegű adathalmazokon tesztelve. A beküldött módszerek között megtalálhatóak voltak gépi tanulást használó és hagyományos (geometria alapú) módszerek [5] is. Az eredmények között jól látható, hogy a legjobb teljesítményt elérő módszerek RGB adatokat is használnak, nem csak mélység adatot. A csak mélység adatot használó módszerek esetében a pontpár jellegzetességleírót használó módszerek teljesítenek a legjobban. Jegyezzük meg, hogy a legjobb eredmények eléréshez mindig szükséges a kezdeti durva transzformáció finomítása valamilyen iteratív módszerrel (általában az Iterative Closest Point [23] és különböző változatait használják erre a célra).

Az implementációk hiánya miatt, a módszerünk összehasonlítására a Buch és társai [81] munkáját választottuk, akik nyilvánosan elérhetővé tették a kódjukat. A módszerük jó eredményeket ér el pontpár jellegzetességleírót használó és egyéb módszerekkel szemben. Releváns az összehasonlításunk, mivel az ő módszerük esetében is tetszőleges lokális jellegzetességleíró használható, így lehetőségünk volt az eredményeket a leírók teljesítményétől függetlenül vizsgálni. Továbbá ők is használnak ICP módszert a durva transzformáció finomításra. A módszerük lényege, hogy lokális jellegzetességleírók legközelebbi szomszéd

[5] hand-crafted

keresései alapján pont megfeleltetéseket határoznak meg, továbbá számon tartják az objektumok középpontját is. Az egymáshoz tartozó pontpárok adott szögenkénti elforgatása után különböző transzformációkra szavazatokat kapnak. A szavazatok klaszterezése után pedig meghatározható a legjobb transzformáció. A módszer részletes leírása megtalálható a hivatkozott munkában.

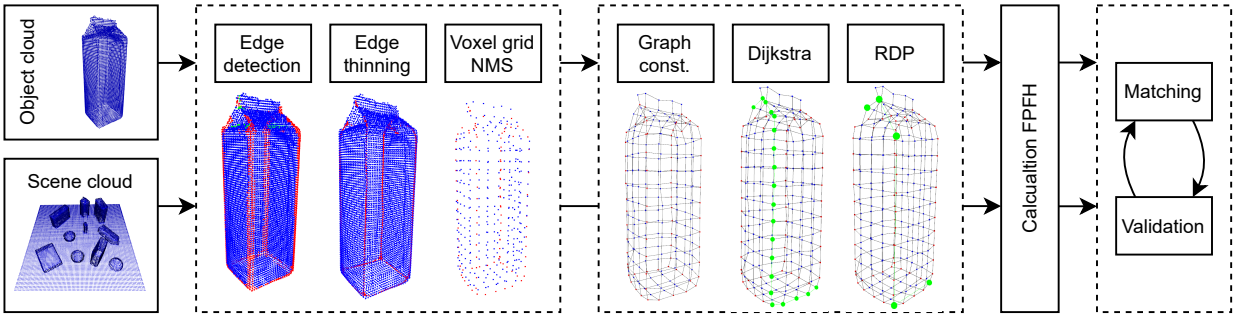
## 6.3. A javasolt módszer leírása

Tegyük fel, hogy két gráfot építünk az objektum pontjai és a helyszínfelhő pontjai felett külön-külön valamilyen módon, ahol az éleket az egymáshoz közeli pontok között húzzuk be. Az éleket súlyokkal is elláthatjuk, például a pontok euklideszi távolsága alapján. Az alábbiakban majd leírjuk, hogy a munkánkban milyen konkrét gráfdefiníciót használunk, most csak annyit jegyezünk meg, hogy több megközelítés létezik az idevonatkozó ún. *proximity graph*-ok meghatározására (ld. ezekről a [25] cikket). Az általánosan megfogalmazott feladat az, hogy az objektumgráf előfordulásait szeretnénk a helyszíngráfban megtalálni. Az egyik fő nehézséget az jelenti, hogy a lézerszkenner mintavételezésének esetlegessége, illetve a zaj miatt sohasem fog tökéletesen ugyanabban az alakban előfordulni az objektum a helyszínfelhőben, mint amilyen pontfelhőt előre ismerünk. Amennyiben a gráfokat ügyesen definiáljuk, pl. voxelrács mintavételezést használva, akkor adott esetben az élsúlyok nem számítanának, így a feladatot vissza lehetne vezetni a *maximum common edge subgraph* feladatra. Ez a probléma azonban APX-nehéz [54]. Összességében elmondható, hogy valamilyen *inexact graph similarity* módszert lehetne használni, amelyről áttekintést nyújt a [46] értekezés.

A fentiek miatt egy olyan megközelítést javaslunk, ahol a pontfelhőknek nem vesszük be az összes pontját a gráfok csúcsainak halmazába, hanem csak láncolatokat definiálunk a pontfelhőkben előforduló élpontok felett. A módszerünk a következő lépésekből áll: éldetektálás az objektum pontfelhőn és a helyszínfelhőn (6.3.1. fejezet), gráf építése és útvonalkeresés az objektum pontfelhőjén (6.3.2. fejezet), hasonló útvonalak keresése a helyszínfelhőn (6.3.3. fejezet).

### 6.3.1. Éldetektálás

Ahhoz, hogy az objektum jól megkülönböztethető részeit megtaláljuk, első lépésben éldetektáló algoritmust használunk pontfelhőkön (az objektum pontfelhőjén és a helyszínfelhőn is). Az éldetektáláshoz a Smooth Shrink Index (SSI) módszert használjuk [76]. A módszer minden ponthoz a felhőben hozzárendel egy SSI értéket, ami a felület meghajlását




6.1. ábra: A javasolt módszer lépései. Az éldetektálás, élvékonyítást és voxelrács NMS-t mindkét felhőn elvégezzük (az élpontok pirosak, a normális pontok kékek). A gráfkonstrukciót, az útvonalkeresést Dijkstra algoritmussal és az RDP módszerrel történő egyszerűsítést csak az objektum pontfelhőjén végezzük el. Végül az FPFH jellegzetességleírókat kiszámoljuk az objektumon a láncolat pontjaira, a helyszínfelhőn pedig minden voxelpontra és elkezdjük a láncolatok keresését.

jellemzi a pont környezetében. Egy adott határérték feletti SSI-vel rendelkező pontokat kiszűrve megkapjuk az élen és az ahhoz hasonló felületeken fekvő pontokat (sarkok, éles felületen lévő pontok stb). A tapasztalataink alapján, az élek megfelelően elkülönülnek a következő határértéket használva: az SSI értékek átlagához hozzáadva az SSI értékek szórását (6.1. ábra, Edge detection).

Az ábrán látható, hogy az SSI szűrés után az élek vastagok. Nagyobb határérték esetén az élek vékonyabbak lennének, azonban nőne a valószínűsége, hogy nem minden élt találunk meg. Azért, hogy kevesebb élpontunk legyen a detektálás után élvékonyítást alkalmazunk. Az él vékonyításhoz az SSI módszer szerzői által említett hagyományos Laplace-simítást használjuk (6.1. ábra). A szerzők kidolgoztak egy módosított változatot is, azonban számunkra a hagyományos módszer jobban működött. Azt is szeretnénk továbbá, hogy élpontok ne legyenek ilyen sűrűek. Ehhez ún. voxelrács non-maximum suppressiont (NMS) használunk, módosítva Hackel és társai ötletét [73]. Egy voxelrácsot alkalmazunk a pontfelhőn, ahol a voxelek oldalhosszát $v$ jelöljük. A $v$ paraméter beállításáról később részletesen írunk. A voxelrácsot az egész felhőre alkalmazzuk és minden voxelben a voxel közepéhez legközelebbi pontot hagyjuk meg. Azokban a voxelekben, ahol a pontok 5%-a élpont, a voxel középpontjához legközelebb lévő élpontot választjuk ki (6.1. ábra, Voxel-grid). A következő lépésben ezeken a csúcsokon építünk gráfot.

A $v$ beállítását úgy szeretnénk kivitelezni, hogy legyen eléggé nagy ahhoz, hogy a zajt elfedje (pl. egy objektumnak ne legyen két éle csak azért mert néhány pont kilóg az élből), de eléggé kicsi ahhoz, hogy az objektum kis részleteit is reprezentálni lehessen vele. A pontfelhők esetén bizonyos szempontból a sűrűségbecsléshez kapcsolódik ez a törekvés. A voxelekbe eső pontok arányával becsülni lehet a pontfelhőhöz kapcsolódó

(valószínűségi) sűrűségfüggvényt, amivel tulajdonképpen egy 3D hisztogramot kapunk. Emiatt érdekes lehet megvizsgálni, hogy a multidimenziós adathalmazokra alkalmazott hisztogramok esetében mi az ideális ládaméret választás. Erre az egyik népszerű és viszonylag friss módszer az ún. Knuth szabály [6] [101], amivel a 3D hisztogramok optimális ládaszámát lehet becsülni. Ezzel a módszerrel a ládák téglatest alakúak lennének, nálunk a voxelek viszont kockák. Így térfogattartó transzformációval kaphatjuk meg a kocka alakú ládák oldalhosszát. Figyelembe véve, hogy a voxelek térfogatának nagysága az algoritmus gyorsaságára is hatással van, a tapasztalatok alapján a Knuth módszerével adódó oldalhossznak a négyszeresét lehet venni a $v$ értékének.

### 6.3.2. Gráfépítés és láncolat meghatározás

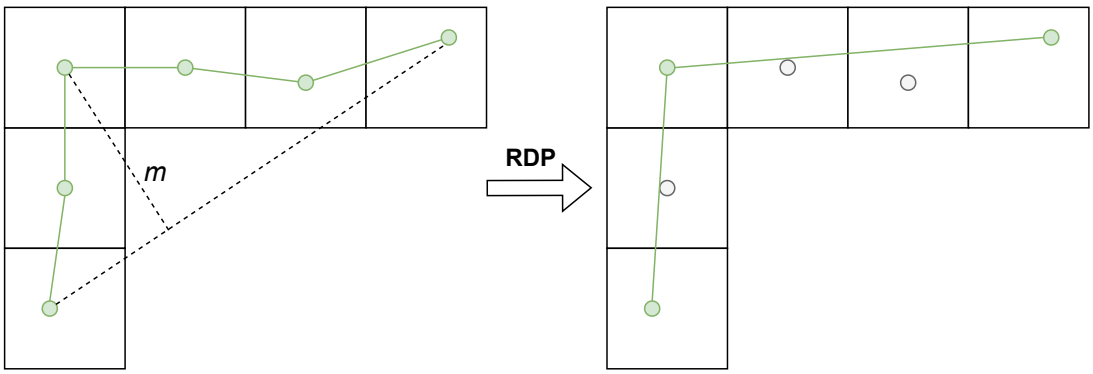
A voxelrács NMS után kapott pontfelhő sokkal ritkább, mint az eredeti. Az előző lépésben leírt módon a voxelrács elkészítése után megkülönböztethetünk élpontokat és normál pontokat. A gráf megépítéséhez akkor és csak akkor kötünk össze két pontot éllel, ha a pontokhoz tartozó voxelek oldalszomszédosak.Egy pontnak tipikusan 6 ilyen szomszédja van, kivéve a voxelrács széleinél. Ha összekötnénk az él és sarok szomszédokat is, sűrűbb gráfot kapnánk, ami lassítaná az útvonal keresést, és tapasztalataink szerint nem járna előnyökkel. Miután összekötöttük az oldalszomszédos voxelbe eső pontokat egy rácsot kapunk (6.1. ábra, Graph building). A gráf éleit súlyozzuk az alapján, hogy az él milyen típusú pontokat köt össze (élpont vagy normál pont). Három eset lehetséges:

- Amikor két élpont között van él, az él súlya megegyezik a pontok közötti euklideszi távolsággal: $w = \|p^1_{edge} - p^2_{edge}\|$
- Egy élpont és egy normál pont közötti él súlya $w = \|p_{edge} - p_{normal}\| + \sqrt{6} \cdot v$
- Két normál pont közötti él súlya pedig: $w = \|p^1_{normal} - p^2_{normal}\| + \sqrt{12} \cdot v$

Ezzel a súlyozással biztosítjuk, hogy a minimális költségű útvonalban mindig az élpontok élvezzenek prioritást. Minden objektumon 8 darab útvonalat keresünk. Meghatározzuk az objektum befoglaló dobozát, és a befoglaló doboz sarkaihoz legközelebb lévő pontokat. Ezek lesznek a sarokpontjaink. Ezután minden sarokpontból elindítunk egy útvonalat az átlósan szemben lévő sarokponthoz. Kezdetben 4 különböző útvonal lesz, mivel a szemközti csúcsok ugyanolyan útvonalat fognak adni. A minimális költségű útvonalat egy kezdő és egy végpont között a Dijkstra algoritmussal határozzuk meg [11] (6.1. ábra, Dijkstra).

[6] Knuth's rule

A hosszú éllel rendelkező objektumok esetén, az útvonalon több egymás melletti pont lesz, amelyek egy egyenesen fekszenek. Könnyű belátni, hogy ezek a pontok nem csökkentik a transzformáció szabadsági fokát, ezért két pont kivételével elhagyhatóak. Ahhoz, hogy ezeket a pontokat kiszűrjük, az útvonalat a Ramer–Douglas–Peucker (RDP) algoritmussal egyszerűsítjük [13] (6.1. ábra, RDP). Az RDP egy térképészetben használt algoritmus, melynek célja, hogy egy töröttvonalat kevesebb pontból álló hasonló töröttvonallal helyettesítsen. Bemeneti paramétere az $\varepsilon$, melyet a felhasználó határoz meg. Az első lépésben az algoritmus a kezdő és a végpontot összeköti egy egyenessel és megnézni, hogy a töröttvonal pontjai közül melyik van a legtávolabb az egyenestől. Ha ez a pont távolabb van mint $\varepsilon$, akkor a pontot megtartjuk és az algoritmus folytatódik: a kiválasztott pontot összekötjük a kezdő és a végponttal és az így kapott két új egyenesen az algoritmus rekurzívan folytatódik. Ha a kiválasztott pont közelebb van, mint $\varepsilon$, akkor az összes pontot töröljük a két végpont között. A célunk az, hogy az egy egyenes lévő pontokat eldobjuk, azonban az útvonal törései minél inkább megmaradjanak. A módszerünkben a következő értéket választottuk: $\varepsilon = (v/\sqrt{2}) - \overline{\Delta}$, ahol $\Delta$ egy voxel középpontjának a távolsága a hozzá legközelebb lévő ponttól ($\overline{\Delta}$ az átlagos távolság). Ezzel a választással az RDP egymás melletti voxelek esetében sem hagyja el a sarokpontot (6.2. ábra). Az RDP alkalmazása után már pontláncolatról beszélünk útvonal helyett, mivel olyan pontok között is lesz él, amelyek között a gráfban nem volt.




6.2. ábra: Az RDP algoritmus működése vázlatosan, egy kétdimenziós példán megmutatva. Az $m$ távolság nagyobb, mint az algoritmus $\varepsilon$ paramétere, így a sarokpont megmarad.

Ahhoz, hogy a láncolat keresését megkönnyítsük a helyszínfelhőben, módosítjuk az RDP algoritmust, hogy a kezdőpont utáni pontot mindig megtartsa, az $\varepsilon$ paramétertől függetlenül (ennek fontosságáról a következő részben fogunk írunk). Ezért előfordulhat, hogy az RDP után egy objektumon 8 különböző láncolatunk lesz. Fontos megjegyezni, hogyha az objektum pontfelhőjén vannak kiugró pontok, a gráf lehet hogy nem összefüggő, így nem biztos, hogy pontosan 8 útvonalunk lesz. Elképzelhető olyan szélsőséges

eset is, amikor egy útvonalat sem tudunk találni - azonban ez valós esetekben nem igazán fordul elő. A láncolat készítés utolsó lépésben a láncolat minden pontjához kiszámoljuk a pont jellegzetességleíróját. A jellegzetességleíró alapján az egyes pontok megkülönböztethetőek és a hasonló felületeken fekvő pontok megtalálhatóak. Az algoritmusunkban a jól ismert és széles körben használt Fast Point Feature Histogram (FPFH) jellegzetességleírót használtuk [42]. Ez a leíró gyorsan kiszámítható és az összehasonlító cikkek szerint is jól teljesít a hasonló módszerekhez képest [66].

### 6.3.3. Hasonló pontláncolatok keresése

Az láncolatok elkészítése után egy objektumhoz legfeljebb 8 láncolat tartozik. A következő lépésben a helyszínfelhőben keresünk láncolatokat, amelyek hasonlítanak az objektum pontfelhőjén talált láncolatokhoz. Az egyszerűség kedvéért tekintsük az objektum pontfelhő egy láncolatát (a többi láncolat keresése párhuzamosan is történhet).

Először megkeressük az objektumon pontfelhőjén lévő láncolat első pontjának legközelebbi szomszédjait a helyszínfelhő pontjai közül a jellegzetességvektorok alapján. A második pont keresésénél nem tudjuk, hogy milyen irányban kell keresnünk, csak azt tudjuk, hogy mekkora távolság van az objektumhoz tartozó láncolaton az első és a második pont között. A második pontot ebben a távolságban keressük egy távolság megszorítás segítségével és annak vizsgálatával, hogy az adott pont élpont vagy normál pont-e. A lehetséges pontok a helyszínfelhőben keresett láncolat $i+1$-edik elemére ($p'$ a képletben) a következőek:

$$D = \{p' \mid \forall k \in [1, i] : |\|p_k - p_{i+1}\| - \|p'_k - p'\|| \leq \delta_{k,i+1}\},$$

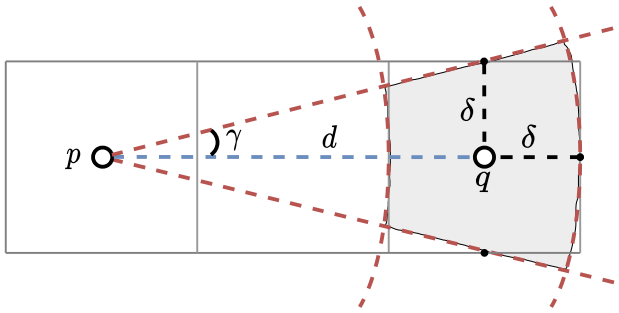
ahol $p_k$ és $p_{k+1}$ az objektumhoz tartozó láncolat egymást követő pontjai, $p'_k$ az $k$. pont a helyszínfelhő pontláncán, és $\delta_{k,i+1}$ az $(i+1)$. pont távolságtoleranciája a $k$. ponthoz viszonyítva.

Mivel a pontok egy voxelen belül bárhol lehetnek, ezért ha biztosítani akarjuk, hogyha létezik a megfelelő pont azt megtaláljuk, akkor az egész voxel területében kell keresnünk, így: $\delta_{1,2} = v/2$. A későbbi pontok esetében a távolság hibahatárt növeljük, a következő módon:

$$\delta_{k,i+1} = v/2 + \frac{\|p_k - p_{i+1}\|}{\rho} \cdot v,$$

ahol $\rho$ az objektum pontfelhőjéhez tartozó befoglaló doboz átmérője. Azaz a láncolat második pontjának keresése alig nagyobb, mint $v/2$ hibahatárral fog történni, egy, a

kezdőponttól távol eső pont hibahatára pedig közel $3v/2$ lesz a kezdőpontra vonatkozóan. Fontos megjegyezni, hogy a láncolat második pontjának a keresésekor azt is megvizsgáljuk, hogy a távolság megszorításnak megfelelő pontok normál vagy élpontok-e és ez alapján tovább szűrjük a találatokat.




6.3. ábra: A távolság és szögmegszorítások illusztrációja: $\delta$ a távolságtolerancia, $\gamma$ a szögtolerancia, $d$ pedig az euklideszi távolság a $p$ és $q$ pontok között.

A láncolat további pontjai keresésekor már nem csak távolság megszorítás, hanem szög megszorítást is használunk:

$$A = \{p' \mid |\angle p_{i-1}p_ip_{i+1} - \angle p'_{i-1}p'_ip'| \leq \gamma_{i+1}\}$$

$$\tan \gamma_{i+1} = \frac{\delta_{i,i+1}}{\|p_i - p_{i+1}\|}$$

ahol $\angle p_{i-1}p_ip_{i+1}$ a láncolat $(i-1)$., $i$., és $(i+1)$. pontjai által bezárt szög, $\gamma_{i+1}$ a szögtolerancia az $(i+1)$. pontra, amely az ugyanerre a pontra vonatkozó távolságtoleranciától függ (6.3. ábra). A távolság és szögmegszorítások meghatároznak egy térrészt, amelyben a láncolat következő pontját keressük $(A \cap D)$. Ha vannak pontok ebben a térrészben, akkor kiszámítjuk a jellegzetességleírójukat és megnézzük a jellegzetességleíróik távolságát az objektumpontfelhő láncolatának megfelelő pontjától. A pontok rendezzük a jellegzetességleírók távolságai alapján. Először a pont a legkisebb jellegzetességleíró távolsággal kerül kiválasztásra, mint következő pont a láncolaton és tovább lépünk. Ha az adott térrész üres vagy nincs több meglátogatatlan pont, akkor visszamegyünk az előző pozícióra a láncolaton és kiválasztjuk a jellegzetességleírók távolság alapján soron következő korábban még nem meglátogatott pontot. A validálásra szoruló láncolatok számának csökkentése érdekében az egy térrészben lévő pontok közül kiválaszthatjuk a jellegzetességleíró távolság alapján $n$ legközelebbit. Mivel egy térrészben sok pont lehet, ezért érdemes ezt a korlátozást beállítani. A kiértékelés során $n = 3$-at használtunk a láncolat elejétől a

láncolat közepéig, utána pedig $n = 2$-re módosítottuk.

### 6.3.4. Validáció

Az algoritmus segítségével összegyűjtjük, az egy kezdőpontból kiinduló összes láncolatot. A következő lépésben a láncolatokat validáljuk két megközelítés alapján: (1) a két láncolat megfelelő pontjai jellegzetességleíróinak a távolsága alapján, (2) a két láncolat alapján becsült transzformációt használva transzformáljuk a voxelpontokat és megnézzük mekkora a távolság az objektum voxelpontjai és a helyszínfelhő pontjai között. Legyen $C = c_1, .., c_n$ az objektumhoz tartozó láncolat és $C' = c'_1, .., c'n$ a helyszínfelhőn talált egyik láncolat (mindkettő $n$ darab csúcsból áll), az $F(\cdot)$ függvény pedig egy pont jellegzetességleíróját adja meg. A két láncolat leírótávolsága (descriptor distance):

$$DD = \sum_{i=1}^{n} \|F(c_i) - F(c'_i)\|$$

A két láncolat által meghatározott $T$ transzformáció alapján számolt pont távolságokat a következő módon kapjuk meg:

$$PD = \sum_{p \in V} \|p' - Tp\|,$$

ahol $V$ az objektumhoz tartozó voxelpontok halmaza, $V'$ pedig a helyszínfelhő voxelpontjainak halmaza ($p' \in V'$). Egy pontból kiinduló minden láncolatra kiszámoljuk a $DD$ és $DE$ értékeket. Ezeket külön-külün rendezzük, azután pedig a rendezések után kapott sorszámok összege alapján kiválasztjuk az $N$ legkisebb értékűt. Minél nagyobb $N$ értéket választunk, annál több egy pontból kiinduló láncolatot kell megvizsgálnunk. Egy kezdőpontból sok láncolat is kiindulhat, amelyek vizsgálata időigényes. A tapasztalataink alapján elég csak az egy kezdőpontból kiinduló 10-20 legjobb láncolatot megvizsgálni, ezután már nem kapunk jó eredményt.

Az $N$ láncolathoz tartozó transzformációkat alkalmazzuk az objektum pontfelhőjére, és ezután megnézzük, hogy az objektum mekkora átfedésben van a helyszínfelhővel. Az átfedés vizsgálatához megnézzük, hogy az objektum pontjai milyen távolságra vannak a helyszínfelhő pontjaitól. Ha egy adott pont távolsága a legközelebbi szomszédjától egy határon belül van, akkor az adott pont illeszkedőnek számít. Ezt a határt érdemes az objektum sűrűsége alapján beállítani, de mindenképpen úgy, hogy megfeleljen az intuíciónak. Az átfedés értékét megkapjuk, ha megnézzük az objektum pontfelhő helyszínpontfelhőre illeszkedő pontjainak az arányát. Ha az átfedés nagyobb, mint egy megadott $\tau$ határérték,

akkor a transzformációt finomítjuk egy iteratív módszerrel (Iterative Closest Point [23]). Mivel hasonló objektumok esetében, a kezdeti transzformáció hamis találatok esetében is magas lehet, akkor fogadunk el egy találatot, ha az ICP-vel finomított transzformáció után az átfedés megfelelően magas lesz.

A módszer képes a sablonillesztés feladat megoldására is, amely azt jelenti, hogy ha egy helyszínfelhőben egy objektumból több előfordulás van, akkor nem csak egyet képes megtalálni. Az ilyen esetekben, ha egy objektumot már megtaláltunk, akkor az objektumhoz tartozó pontokat a helyszínfelhőből eltávolítjuk a további lehetséges láncolat kezdőpontok közül. Ha tudjuk hány előfordulása van a keresett objektumnak a helyszínfelhőben, akkor az adott számú előfordulás megtalálása esetén az algoritmust leállíthatjuk.

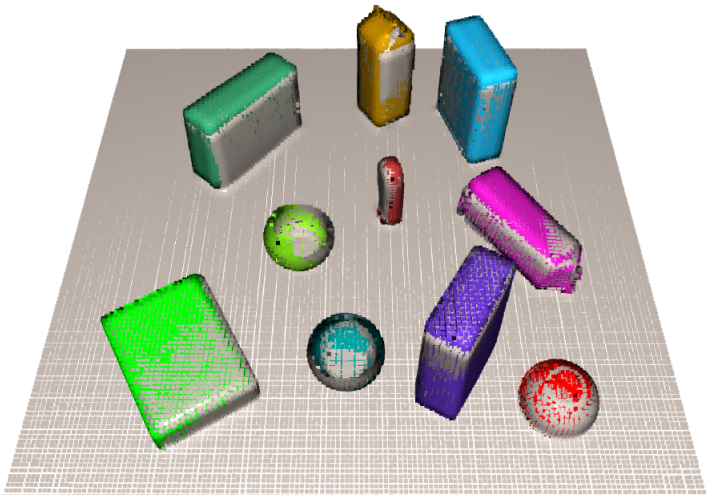
## 6.4. Eredmények

### 6.4.1. Adathalmaz

A módszerünk kiértékeléséhez és összehasonlításához két adathalmazt választottunk a BOP Challange-ben használtak közül [112]: T-LESS [84] és Toyota Light [93] (TYO-L). Mindkét adathalmaz objektumokat tartalmaz: a TYO-L mindennapi háztartási objektumokat, míg a T-LESS ipari vonatkozású textúra nélküli objektumokat. A helyszínfelhők elkészítéséhez egy síkra helyeztük a két adathalmaz különböző objektumait, egy vagy több előfordulással. Az így elkészült színtereket és az objektumokat egy virtuális szkennerrel mintavételeztük, több különböző nézőpontból, így az eredményben a felületek minden oldalról látszanak. A pontfelhők sűrűsége nem egyforma és kitakarások is előfordulnak. Három helyszínfelhőt készítettünk: első helyszín (T-LESS objektumok), második helyszín (TYO-L objektumok) és harmadik helyszín (T-LESS és TYO-L objektumok vegyesen). Az objektumok alakjai változatosak. A TYO-L objektumok többnyire egyszerű háztartási objektumok: a doboz csak élekből és síkokból áll, a tejes doboz pedig nagyon hasonló. Mivel azonban hasonló objektumok, ezért könnyű összetéveszteni őket. A T-LESS objektumok bonyolultabb alakkal rendelkeznek, de hasonló részek vannak rajtuk, amelyek szintén kihívást jelentenek a felismerés során.

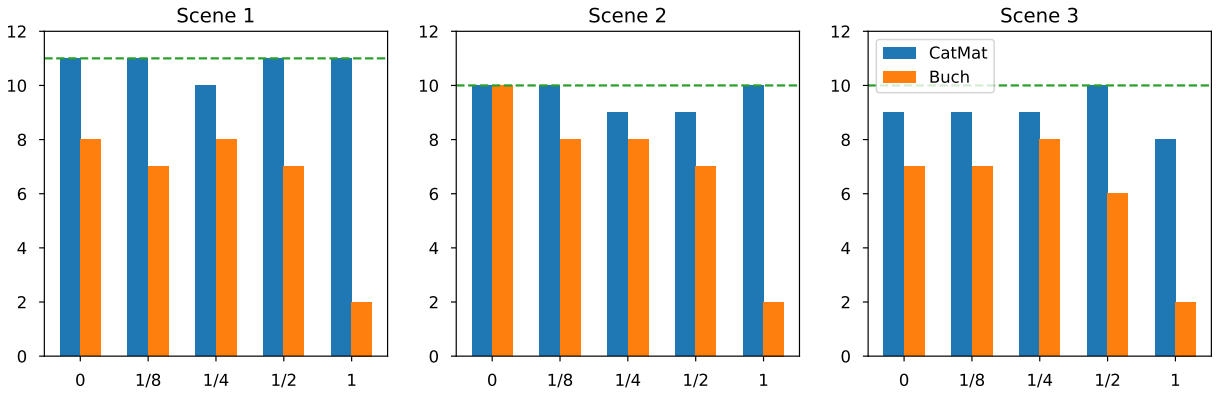
### 6.4.2. Kiértékelés

A munkánkat Buch és társai által készített módszerrel hasonlítottuk össze [81]. Azért, hogy az összehasonlításhoz hasonló feltételeket biztosítsunk, mindkét módszer esetében ugyanazt a jellegzetességleírót (FPFH) használtuk. A robusztusság méréséhez elkészítet-



6.4. ábra: TYO-L objektumokat tartalmazó helyszínfelhő (zaj nélkül) a megtalált és transzformált objektumokkal.

tük a helyszínfelhők zajos változatát: a pontfelhőkhöz Gauss zajt adtunk hozzá, amelynek a szórása a pontfelhőfelbontástól függött. Ahhoz, hogy megkapjuk a zajt a pontfelhőfelbontást megszoroztuk egy konstanssal (6.5. ábra vízszintes tengelye).




6.5. ábra: A helyesen detektált objektumok száma a különböző helyszínfelhőkben, Gauss-zaj hozzáadásával (vízszintes tengely). A zajosságot mutató értékek a pontfelhő felbontásának a szorzói. A zöld vízszintes vonal mutatja az objektumok valós számát az adott helyszínfelhőn.

Az 6.5. ábra megmutatja, hogy a két módszer hány objektumot talált meg az egyes helyszínfelhőkön. A mi módszerünk az összes lehetséges találatból (155 objektum: 31 objektum a három helyszínfelhőn, 5 különböző méretű zajjal) csupán tízet nem talált meg, ezt 4 objektum okozza csupán. A első helyszínen csupán egyetlen egyszer nem találja meg (0.5 zajjal) az elosztót. A második helyszínfelhőn 2 tejes doboz helyezkedett el. Itt 2 zajszinten is az egyik tejes dobozt nem találta meg, és a többi zajszinten is

| Zaj | Első helyszín | | Második helyszín | | Harmadik helyszín | |
|---|---|---|---|---|---|---|
| | Miénk | Buch | Miénk | Buch | Miénk | Buch |
| 0 | 0.951 | **0.971** | **0.971** | 0.968 | **0.971** | 0.968 |
| 1/8 | 0.964 | **0.971** | **0.977** | 0.968 | **0.977** | 0.968 |
| 1/4 | 0.952 | **0.964** | **0.986** | 0.965 | **0.986** | 0.965 |
| 1/2 | 0.956 | **0.973** | **0.977** | 0.959 | **0.977** | 0.959 |
| 1 | **0.950** | 0.938 | 0.971 | 0.946 | **0.971** | 0.946 |

6.1. táblázat: Az átfedések átlagos értékei a finomított transzformációk után.

elmondható, hogy nehezen találta meg őket. Ennek az oka az lehet, hogy a tejes doboz nagyon hasonlít egy másik előforduló objektumhoz, a müzlis dobozhoz. Ezért könnyű a két objektumon található pontokat összekeverni. Tovább nehezíti a helyzetet, hogy a virtuális szkenner által felvett helyszínfelhőn a tejes doboz objektum alja nem minden nézőpontból látszik teljesen, azaz egy kis része ki van takarva.

A harmadik helyszínfelhőn mindkét módszer rosszabbul teljesített, mint a másik két helyszínfelhőn. A mi módszerünk esetében ezt az okozza, hogy a távirányító objektumnak két előfordulása van, ebből jellemzően egyet megtalál, viszont a másik előfordulás esetében mindig hamis találatot kapunk. Ugyanis az egyik elosztó alakja nagyon hasonló és így a pontjai jellegzetességleíróik is. Buch és társai módszere a távirányítókat megtalálja, kivéve nagy zajnál. Az ő módszerük a hasonló elosztókat nem tudta megfelelően azonosítani. Mindkét fajta elosztóból 2-2 szerepel a helyszínen, ezekből jellemzően az egyiket nem tudja megtalálni, magasabb zajnál akár egyiket sem. Az első helyszínen már 3 fajta elosztó szerepel, az előző kettő és egy harmadik fajta, bár ez eltérőbb az előzőektől. A meg nem talált objektumaik körülbelül háromnegyedét az elosztók okozzák.

A 6.1. táblázat azt mutatja, hogy a helyszínfelhőket átlagosan milyen átfedés értékekkel találták meg az objektumokat a módszerek. Az értéket az ICP-vel finomított transzformációk utáni átfedéseket mutatják. Az ICP-nek köszönhetően az értékek hasonlóak, mivel képes nem olyan jó kezdeti transzformációkat is pontosabbá tenni. Ha a kezdeti transzformációkat nézzük, akkor mi módszerünk átlagosan 0.722 fit értékkel rendelkező transzformációkat ad, míg a Buch módszer 0.916-ot. Látható, hogy ezt tekintve a mi módszerünk sokkal rosszabbul teljesít. Ez azért van, mert ha már eléggé jó transzformációt találunk, akkor leállítjuk a keresést és nem nézzük meg az összes láncolatot. Ha az eléggé jó transzformáció határát felemeljük, akkor magasabb értékeket kaphatunk, viszont tovább tart a futtatás. A megközelítésünk lényege az volt, hogy mivel ICP finomítást használunk, ezért elég, ha nagyjából megvan az objektum helye, ezért nem volt elsődleges célunk a minél jobb kezdeti transzformáció megtalálása.

Az implementáció során nem fordítottunk kiemelt figyelmet a kód optimalizálására,

sem a párhuzamosításra. Buch és társai módszere $C++$ programozási nyelv használatával van implementálva, a miénk pedig Pythonban, így természetesen az ő módszerük futási ideje sokkal kedvezőbb. Továbbá esetünkben több paraméter is van, amely a futási időt tudja javítani a robusztusság kárára. Ezek következtében a futási idő összehasonlítások a jövőbeli munkánk részét képezik.

## 6.5. Összefoglalás és lehetséges továbbfejlesztések

Ebben a munkában javasoltunk egy újszerű objektum felismerő módszert, amely a pontfelhők élpontjainak meghatározása után voxelráccsal alulmintavételezi a felhőt és a voxelpontokon gráfot épít. A gráf élsúlyainak beállítása után útvonalat keres az objektumhoz tartozó pontfelhő gráfján egy kijelölt kezdő és végpont között. A megtalált útvonalat egyszerűsíti és az így kapott pontláncolatokhoz hasonló láncolatokat keres a helyszínfelhőn geometriai megszorítások és jellegzetességleíró párosításokkal. A megtalált útvonalak segítségével transzformáció becsülhető, amely képes az objektum pontfelhőjét a helyszínen való előfordulására illeszteni. A kiértékelésünk alapján a módszerünk robusztus a zajra és jó eredményeket ér el hasonló módszerekhez képest.

Egy fontos hiányossága a módszerünknek, hogy feltételezzük, hogy a megkeresni kívánt objektum pontfelhője majdnem teljes egészében jelen van a helyszínfelhőn (legalábbis létezik olyan nézőpont, amelyből vizsgálva nincsenek kitakarva az objektum részei). A valós esetekben azonban gyakran előfordul, hogy egy objektum nagy része ki van takarva vagy túl közel vannak egymáshoz az objektumok. Ezekben az esetekben nem lehetséges az objektumfelhőn detektált láncolathoz hasonlót találni a helyszínfelhőn.

A fenti probléma megoldására a terveink között szerepel a láncolatok kezdő és végpontjainak meghatározásán módosítani és megszorítást alkalmazni a láncolatok hosszára. Ezzel a célunk, hogy sok rövidebb láncolatot találjunk az objektum pontfelhőjén és ezeket a rövidebb láncolatokat keressük a helyszínfelhőn. Ugyanis rövidebb láncolatok esetén csökken annak az esélye, hogy a láncolat egy része kitakarásra került a helyszínfelhőben.

Ehhez kapcsolódik egy másik jövőbeli tervünk is, a kód párhuzamos implementációja. Könnyű látni, hogy a módszer több fontos része is futtatható párhuzamosan. A legfontosabb ezek közül a különböző kezdőpontokból kiinduló láncolatok meghatározása és validálása. A párhuzamosítás legegyszerűbb módja lehet ezen esetek külön kezelése. A párhuzamos implementáció által lehetőségünk nyílna módszerünket futási idő szempontjából is összehasonlítani más módszerekkel.

# 7. fejezet

# Összegzés

A doktori értekezésemben a jellegzetességleíró alapú 3D pontfelhő regisztrációs folyamatokat és azok lépéseit vizsgáltam, különös tekintettel a jellegzetességleíró módszerek működésére. Bemutattam a regisztrációs folyamatok általános lépéseit és a hozzájárulásimat az egyes részekhez. Az bemutatott algoritmusok és módszerek számos alkalmazási területen használhatóak: robotika, autonóm járművek, kiterjesztett valóság alkalmazások.

A 3. és 4. fejezetben olyan megoldásokat vizsgáltam, amelyek hozzájárulnak a jellegzetességleírókon végzett legközelebbi szomszéd keresések felgyorsításához. Az előbbi fejezetben a jellegzetességvektorok dimenziócsökkentésének hatását vizsgáltam a leíróképességre, míg a másikban a valós értékű jellegzetességleírók binarizációjára javasoltam egy újszerű megoldást. A kiértékeléshez összegyűjtöttem és implementáltam az elérhető binarizációs módszereket és az önálló bináris jellegzetességleírókat.

Az 5. és a 6. fejezetekben az objektumfelismerés és sablonillesztés feladatokra javasoltam új módszereket. Az előbbi módszer újdonsága, hogy adatbázis-kezelő rendszert használ a pontfelhőben lévő pontok és azok jellegzetességleíróinak a tárolására és lekérdezésére. A megfeleltetések meghatározását páros gráfon minimális súlyú párosítás meghatározásaként reprezentáltam. Az utóbbi módszerben a javasolt módszer gráfot épít az objektumfelhő érdekes pontjain és a gráfon történő útkereséssel és rekurzív egyszerűsítéssel határozza meg a pontláncolatokat.

A munkám során a jellegzetesség leíró alapú folyamatok több fontos lépését is továbbfejlesztettem és olyan módszereket javasoltam, amelyek képesek megoldani a terület fontos problémáit.

# 8. fejezet

# Summary

In my doctoral thesis, I studied the steps of feature descriptor-based 3D point cloud registration processes, with a special focus on feature descriptor methods. I presented the general steps of the registration process and my contributions to each part. The proposed algorithms and methods can be used in a variety of applications: robotics, autonomous vehicles, and augmented reality applications.

In chapters 3. and 4., I discussed solutions that contribute to speeding up nearest neighbor searches on feature descriptors. In the former chapter, I investigated the impact of dimensionality reduction on the feature descriptors' descriptive performance, while in the latter I proposed a novel solution for binarization of real-valued feature descriptors. For the evaluation, I collected and implemented the available binarization methods and stand-alone binary feature descriptors.

In the chapters 5 and 6, I proposed new methods for object recognition and template matching. The novelty of the former method is that it uses a database management system to store and retrieve points and their feature vectors. I represented the determination of correspondences as the determination of minimum weight matching on a bipartite graph. In the latter method, the proposed method builds a graph on interesting points of the object point cloud and determines point chains by path search on the graph with recursive path simplification. By searching for similar point chains, the occurrences of objects can be found on the scene point clouds.

In my work, I have improved several important steps of the feature description-based processes and proposed methods that can solve important problems in this field.

# A. függelék

# PFH és FPFH jellegzetességleírók

Ez a fejezet Rusu és társai publikációi és [42, 38] továbbá Radu Bogdan Rusu értekezése [47] alapján készült.

A Point Feature Histogram (PFH) és a Fast Point Feature Histogram (FPFH) jellegzetességleírók az értekezésemben több fejezetben is szerepelnek, működésének megismerése fontos az eredmények teljes körű megértéséhez. A PFH és FPFH lokális jellegzetességleírók, amelyeket pont megfeleltetések meghatározásának céljából készítettek, beltéri pontfelhőket használó felhasználási esetekre.

## A.1. Point Feature Histogram

Egy $\mathcal{P}$ pontfelhő $\mathbf{p}_q$ pontjának a PFH jellegzetességvektora a pont $r$ sugarú környezetében lévő szomszédjainak jellemzése. A szomszédos pontok halmazát jelölje $\mathcal{N}_r$, a szomszédok száma pedig legyen $N$, azaz $|\mathcal{N}_r| = N$ A kiszámításához első lépésben $\frac{N(N-1)}{2}$ darab pontpárt alkotunk úgy, hogy minden pont párt alkot minden más ponttal a szomszédságban. Jelölje $\mathbf{p}_i$ és $\mathbf{p}_j$ egy pontpár két pontját, $\mathbf{n}_i$ és $\mathbf{n}_j$ pedig a pontok normálvektorát. Minden pontpárra létrehozunk egy Darboux koordináta-rendszert a következő módon:

$$
\begin{aligned}
u &= \mathbf{n}_j \\
v &= u \times \frac{\mathbf{p}_j - \mathbf{p}_i}{||\mathbf{p}_j - \mathbf{p}_i||_2} \\
w &= u \times v
\end{aligned}
$$

A Darboux koordináta-rendszer segítségével kiszámítható 3 szükséges jellemző:

$$\alpha = v \cdot \mathbf{n}_j$$
$$\phi = u \cdot \frac{\mathbf{p}_j - \mathbf{p}_i}{d}$$
$$\theta = arctan(w \cdot \mathbf{n}_j, u \cdot \mathbf{n}_j)$$

ahol $d = ||\mathbf{p}_j - \mathbf{p}_i||_2$ az Euklideszi távolság a két pont között. A $\mathbf{p}_q$ pont jellegzetességvektorának a létrehozásához a szomszédság alapján létrehozott pontpárok mindegyikére ki kell számítani a jellemzőket, így minden párra megkapjuk a $(\alpha, \phi, \theta, d)$ négyest. Mind a négy jellemzőt kosarazzuk, $b$ darab egyenlő részre, és megszámoljuk az egyes részekbe hány elem esett. A kosarazás után egy $b^4$ elemű hisztogramot kapunk, amely a $\mathbf{p}_q$ pont $r$ sugarú sugárhoz tartozó PFH jellegzetességvektora lesz. A $b$ értéke az alapértelmezett implementációban 5, és a pontok távolsága ($d$) nem része a jellemzőknek, így egy 125 dimenziós jellegzetességvektort kapunk. Egy pontfelhő minden pontjára a PFH kiszámítási költsége $\mathcal{O}(nk^2)$, ahol $n$ a pontfelhő számossága, $k$ pedig a pontfelhő pontjaihoz tartozó szomszédság számossága.

## A.2. Fast Point Feature Histogram

Az FPFH jellegzetességleíró a PFH továbbfejlesztése, egyszerűsítése. A PFH $\mathcal{O}(nk^2)$ számítási költségét $\mathcal{O}(nk)$-ra csökkenti úgy, hogy megőrzi a PFH leíróképességét (ahol $n$ a pontfelhő számossága, $k$ pedig a pontfelhő pontjaihoz tartozó szomszédság számossága).

Az FPFH ugyanazt a három jellemzőt $(\alpha, \phi, \theta)$ számolja ki a pontpárokra, mint a PFH, azonban a pontpárok meghatározása eltérő módon történik. A jellegzetességleíró módszer első lépésben egy ún. Simplified Point Feature Histogram-ot (SPFH) készít úgy, hogy a $\mathbf{p}_q$ lekérdezési pont minden szomszédjával párt alkot $r$ sugarú környezetben és ezekre a pontpárokra kiszámolja a három jellemzőt. A következő lépésben a $\mathbf{p}_q$ minden szomszédjára ugyanúgy elkészíti az SPFH-t, és ezeket a $\mathbf{p}_q$-tól való távolságuk alapján súlyozva megkapjuk a végleges FPFH jellegzetességvektort:

$$\text{FPFH}(\mathbf{p}_q) = \text{SPFH}(\mathbf{p}_q) + \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\omega_i} \cdot \text{SPFH}(\mathbf{p}_i)$$

ahol $N$ a $\mathbf{p}_q$ pont szomszédainak száma, $\mathbf{p}_i (i = 1, .., N)$ a $\mathbf{p}_q$ $i$.-ik szomszédja és $\omega_i$ az $i$.-ik szomszéd távolsága a $\mathbf{p}_q$ ponttól.

Vegyük észre, hogy az FPFH kevesebb információt foglal magába a lekérdezési pont $r$

szomszédságáról, azonban tartalmaz információt a lekérdezési pont szomszédjainak szomszédairól is, így a lekérdezési ponttól legfeljebb $2r$ távolságra lévő pontok is hatással lehetnek a jellegzetességvektorra. A PFH-val ellentétben az FPFH a három jellemzőt $(\alpha, \phi, \theta)$ külön-külön kosara $b$ darab egyenlő részre és az elkészült hisztogramokat összefűzi. Ennek köszönhetően az FPFH jellegzetességvektor kevesebb dimenzióból áll és jóval kevesebb 0 értéket tartalmaz. Az alapértelmezett implementációban az FPFH esetén a $b$ értéke 11, így a végső jellegzetességvektor 33 dimenziós.

# B. függelék

# Iterative Closest Point ponthalmaz regisztráció

Ez a fejezet Paul J. és McKay [23], Chen és Medioni [22], Rusinkiewicz és Levoy [118] szerzők publikációik alapján készült.

Az Iterative Closest Point (továbbiakban ICP) algoritmus többször előkerül az értekezésemben, ugyanis az ICP egy széles körben ismert, jól működő ponthalmaz regisztrációs algoritmus, amelyet gyakran használnak jellegzetességleíró alapú regisztrációs folyamatok utolsó lépéseként, finomított transzformációk meghatározására. Az ICP 1991-es megjelenése óta számos továbbfejlesztés született és a mai napig jelennek meg új munkák ebben a témában [32, 78, 33]. A függelékben az ICP algoritmus egy általános változatát mutatom be.

Legyen $P$ és $Q$ két, háromdimenziós pontokat tartalmazó pontfelhő. Az ICP algoritmus meghatároz egy olyan $T$ térbeli transzformációt, amely $P$ forrás pontfelhőt a $Q$ célfelhőre illeszti. Az ICP algoritmusnak megadható két paraméter, amely ciklus leállását szabályozza. Ha a hibametrika eléri a $\varepsilon_E$ határértéket az algoritmus leáll. Az $N$ pedig a maximális iterációszám. Az általános ICP algoritmus pszeudokódját a 4. algoritmus tartalmazza. Az algoritmus négy fő részre osztható: (1) megfeleltetések meghatározása ($\mathcal{C}$) (2) a hibametrikát minimalizáló transzformáció becslése ($T \in \mathcal{T}$, ahol $\mathcal{T}$ az összes lehetséges transzformáció) (3) transzformáció alkalmazása (4) feltételek ellenőrzése és újrakezdés.

Az ICP különböző változatai általában 6 fő pontban különböznek az algoritmus elsőnek bemutatott változatától:

1. **Pontok kiválasztása** - Azon pontok meghatározása, amelyek részt vesznek a megfeleltetések létrehozásában. Az 4. algoritmus esetében a $P$ pontfelhő minden pontja

**Algoritmus 4** Általános Iterative Closest Point

INPUT: $P, Q$ forrás és cél pontfelhők, $\varepsilon_E$ hibahatár, $N$ maximális iterációszám
OUTPUT: $T$ transzformáció

1: $k = 0$ ▷ iterációszám
2: **repeat**
3: $\mathcal{C} \leftarrow \{\}$ ▷ megfeleltetések halmaza
4: **for all** $\mathbf{p} \in P$ **do**
5: $\mathbf{q} \leftarrow NearestNeighbor(Q, \mathbf{p})$
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mathbf{p}, \mathbf{q})\}$
7: **end for**
8: $T = \arg\min_{T \in \mathcal{T}} E(T) = \arg\min_{T \in \mathcal{T}} \sum_{(\mathbf{p}_i, \mathbf{q}_i) \in \mathcal{C}} ||\mathbf{q}_i - T\mathbf{p}_i||^2$
9: $P = T(P)$ ▷ transzformáció alkalmazása
10: $k = k + 1$
11: **until** $E(T) < \varepsilon_E$ or $k > N$
12: **return** $T$

kiválasztásra kerül.

2. **Párosítás** - A kiválasztott pontok közötti megfeleltetések keresése. A bemutatott algoritmusban a $P$ pontfelhő minden pontja a $Q$ pontfelhőből meghatározott legközelebbi szomszédjával fog megfeleltetést alkotni.

3. **Súlyozás** - A létrehozott megfeleltetések különböző súllyal vehetnek részt a transzformáció meghatározásában. A bemutatott algoritmusban súlyozás nem szerepel.

4. **Elutasítás** - A létrehozott megfeleltetések egy része elutasításra kerülhet, így nem fog részt venni a transzformáció meghatározásában. A bemutatott algoritmusban súlyozás nem szerepel.

5. **Hibametrika** - A különböző ICP változatok eltérő hibametrikával dolgozhatnak. A bemutatott algoritmus a pont-pont távolságot használja hibametrikaként.

6. **Hiba minimalizálása** - A különböző ICP változatok eltérő módszert használhatnak a hiba minimalizálását célzó transzformáció meghatározásakor.

Az értekezésben a Chen és Medioni [22] által javasolt ICP változatot használjuk az Open3D könyvtár implementációja alapján.

Az ICP általunk használt változatában, a 6. lépésben alkalmazott módszer a hiba minimalizálására Kabsch algoritmusa. Az alábbi leírás Kabsch munkái [14, 14] és Arun és társai [19] alapján készült.

A pont megfeleltetések létrehozása után meg kell határozni azt a transzformációt, amely a megfeleltetéseket alkotó pontokat egymásra illesztik. Kabsch algoritmusa alapján

az első lépés a ponthalmazok súlypontjának a kiszámítása és a pontok eltolása úgy, hogy a súlypont az origóba kerüljön. Legyen $C_P$ és $C_Q$ a $P$ forrás pontfelhő és $Q$ cél pontfelhő $\mathcal{C}$ megfeleltetéshalmaz által tartalmazott pontjainak a súlypontjai:

$$C_P = \frac{1}{|\mathcal{C}|} \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{C}} p$$

$$C_Q = \frac{1}{|\mathcal{C}|} \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{C}} q$$

Ekkor, az eltolás után az alábbi ponthalmazokat kapjuk, amelyekben csak a megfeltételhalmaz által tartalmazott pontok szerepelnek:

$$P' = \{\mathbf{p} - C_P\} = \{\mathbf{p}'\}, (\mathbf{p} \in P, (\mathbf{p},\mathbf{q}) \in \mathcal{C})$$

$$Q' = \{\mathbf{q} - C_Q\} = \{\mathbf{q}'\}, (\mathbf{q} \in Q, (\mathbf{p},\mathbf{q}) \in \mathcal{C})$$

A $H$ kovarianciamátrix szinguláris felbontása alapján megkapjuk a forgatáshoz szükséges mátrixokat:

$$H = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{C}} q' p'^T$$

$$H = UDV^T$$

$$s = sign(det(VU^T))$$

Ekkor az optimális forgatás és eltolás:

$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix} U^T$$

$$T = C_Q - RC_P$$

# Szerző publikációi


[1] **Dániel Varga** és Sándor Laki. "Scalable Surface Reconstruction in the Mobile Edge". *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. ACM, 2018. aug.

[2] **Dániel Varga**, Sándor Laki, János Szalai-Gindl, László Dobos, Péter Vaderna és Bence Formanek. "On Fast Point Cloud Matching with Key Points and Parameter Tuning". *Lecture Notes in Computer Science*. Springer International Publishing, 2020, 498–511. old.

[3] János Hegedűs-Kuti, József Szőlősi, **Dániel Varga**, Gábor Farkas, Tamás Ruppert, János Abonyi és Mátyás Andó. "3D Scanning and Model Error Distribution-Based Characterisation of Welding Defects". *Hungarian Journal of Industry and Chemistry* 49.2 (2021), 3–7. old.

[4] **Daniel Varga**, Janos Mark Szalai-Gindl, Bence Formanek, Peter Vaderna, Laszlo Dobos és Sandor Laki. "Template Matching for 3D Objects in Large Point Clouds Using DBMS". *IEEE Access* 9 (2021), 76894–76907. old.

[5] **Dániel Varga**, János Márk Szalai-Gindl és Sándor Laki. "The Descriptiveness of Feature Descriptors with Reduced Dimensionality". *New Trends in Database and Information Systems*. Springer International Publishing, 2021, 317–322. old.

[6] Michelisz Máté, **Daniel Varga** és János Márk Szalai-Gindl. "CatMat: 3D Object Recognition Using Catenarian Matching". *The 2022 IEEE 2nd Conference on Information Technology and Data Science, CITDS, Debrecen, Hungary/Online, 16-18 May 2022*. 2022.

[7] **Dániel Varga**, János Márk Szalai-Gindl, Márton Ambrus-Dobai és Sándor Laki. "QBB: Quantile-Based Binarization of 3D Point Cloud Descriptors". *IEEE Access* 10 (2022), 67839–67850. old.

# További publikációk


[8] Frank Gray. “Pulse code communication”. *United States Patent Number 2632058* (1953).

[9] Raphael M Robinson. “Mersenne and Fermat numbers”. *Proceedings of the American Mathematical Society* 5.5 (1954), 842–846. old.

[10] Harold W Kuhn. “The Hungarian method for the assignment problem”. *Naval research logistics quarterly* 2.1-2 (1955), 83–97. old.

[11] E. W. Dijkstra. “A note on two problems in connexion with graphs”. *Numerische Mathematik* 1.1 (1959. dec.), 269–271. old.

[12] Keinosuke Fukunaga és David R Olsen. “An algorithm for finding intrinsic dimensionality of data”. *IEEE Transactions on Computers* 100.2 (1971), 176–183. old.

[13] David H Douglas és Thomas K Peucker. “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature”. *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973. dec.), 112–122. old.

[14] W. Kabsch. “A discussion of the solution for the best rotation to relate two sets of vectors”. *Acta Crystallographica Section A* 34.5 (1978. szept.), 827–828. old.

[15] Martin A. Fischler és Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. *Commun. ACM* 24.6 (1981), 381–395.

[16] David Freedman és Persi Diaconis. “On the histogram as a density estimator: $L_2$ theory”. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 57.4 (1981), 453–476. old.

[17] Jeffrey Scott Vitter. “Faster methods for random sampling”. *Communications of the ACM* 27.7 (1984. júl.), 703–718. old.

[18] O.D. Faugeras és M. Hebert. “The Representation, Recognition, and Locating of 3-D Objects”. *The International Journal of Robotics Research* 5.3 (1986. szept.), 27–52. old.

[19] K. S. Arun, T. S. Huang és S. D. Blostein. “Least-Squares Fitting of Two 3-D Point Sets”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), 698–700. old.

[20] L. Sirovich és M. Kirby. “Low-dimensional procedure for the characterization of human faces”. *Journal of the Optical Society of America A* 4.3 (1987. márc.), 519. old.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard és L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. *Neural Computation* 1.4 (1989. dec.), 541–551. old.

[22] Y. Chen és G. Medioni. “Object modeling by registration of multiple range images”. *Proceedings. 1991 IEEE International Conference on Robotics and Automation.* 1991, 2724–2729 vol.3.

[23] Paul J. Besl és Neil D. McKay. “A Method for Registration of 3-D Shapes”. *IEEE Trans. Pattern Anal. Mach. Intell.* 14.2 (1992. febr.), 239–256.

[24] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald és Werner Stuetzle. “Surface Reconstruction from Unorganized Points”. *SIGGRAPH Comput. Graph.* 26.2 (1992. júl.), 71–78.

[25] J.W. Jaromczyk és G.T. Toussaint. “Relative neighborhood graphs and their relatives”. *Proceedings of the IEEE* 80.9 (1992), 1502–1517. old.

[26] Roger Weber, Hans-Jörg Schek és Stephen Blott. “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces”. *VLDB.* 98. köt. 1998, 194–205. old.

[27] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan és Uri Shaft. “When is “nearest neighbor” meaningful?”: *International conference on database theory.* Springer. 1999, 217–235. old.

[28] K. Pulli. “Multiview registration for large data sets”. *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062).* 1999, 160–168. old.

[29] A W Fitzgibbon. “Robust Registration of 2D and 3D Point Sets”. *Procedings of the British Machine Vision Conference 2001.* British Machine Vision Association, 2001.

[30] Flip Korn, B-U Pagel és Christos Faloutsos. “On the" dimensionality curse" and the" self-similarity blessing"”. *IEEE Transactions on Knowledge and Data Engineering* 13.1 (2001), 96–111. old.

[31] S. Rusinkiewicz és M. Levoy. “Efficient variants of the ICP algorithm”. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. 2001, 145–152. old.

[32] D. Chetverikov, D. Svirko, D. Stepanov és P. Krsek. “The Trimmed Iterative Closest Point algorithm”. *2002 International Conference on Pattern Recognition*. 3. köt. 2002, 545–548 vol.3.

[33] Sébastien Granger és Xavier Pennec. “Multi-scale EM-ICP: A Fast and Robust Approach for Surface Registration”. *Computer Vision — ECCV 2002*. Szerk. Anders Heyden, Gunnar Sparr, Mads Nielsen és Peter Johansen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, 418–432. old.

[34] Niloy J. Mitra és An Nguyen. “Estimating surface normals in noisy point cloud data”. *Proceedings of the nineteenth conference on Computational geometry - SCG '03*. ACM Press, 2003.

[35] Mark Pauly, Richard Keiser, Leif P. Kobbelt és Markus Gross. “Shape modeling with point-sampled geometry”. *ACM Transactions on Graphics* 22.3 (2003. júl.), 641–650. old.

[36] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. *International Journal of Computer Vision* 60 (2004. nov.), 91–. old.

[37] Hanan Samet. *Foundations of multidimensional and metric data structures*. Academic Press, 2006.

[38] Radu Rusu, Zoltan Marton, Nico Blodow és Michael Beetz. “Persistent Point Feature Histograms for 3D Point Clouds”. *Proc 10th Int Conf Intel Autonomous Syst (IAS-10)* 16 (2008. jan.).

[39] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha és Michael Beetz. “Towards 3D Point cloud based object maps for household environments”. *Robotics and Autonomous Systems* 56.11 (2008. nov.), 927–941. old.

[40] K. Klasing, D. Althoff, D. Wollherr és M. Buss. “Comparison of surface normal estimation methods for range sensing applications”. *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009. máj.

[41] A. Mian, M. Bennamoun és R. Owens. “On the Repeatability and Quality of Keypoints for Local Feature-based 3D Object Retrieval from Cluttered Scenes”. *International Journal of Computer Vision* 89.2-3 (2009. szept.), 348–361. old.

[42] Radu Bogdan Rusu, Nico Blodow és Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. *2009 IEEE international conference on robotics and automation*. IEEE. 2009, 3212–3217. old.

[43] Yu Zhong. “Intrinsic shape signatures: A shape descriptor for 3D object recognition”. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, 2009. szept.

[44] Yu Zhong. “Intrinsic shape signatures: A shape descriptor for 3D object recognition”. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, 2009. szept.

[45] Bertram Drost, Markus Ulrich, Nassir Navab és Slobodan Ilic. “Model globally, match locally: Efficient and robust 3D object recognition”. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010. jún.

[46] Kaspar Riesen, Xiaoyi Jiang és Horst Bunke. “Exact and Inexact Graph Matching: Methodology and Applications”. *Managing and Mining Graph Data*. Springer US, 2010, 217–247. old.

[47] Radu Bogdan Rusu. “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments”. *KI - Künstliche Intelligenz* 24.4 (2010. aug.), 345–348. old.

[48] Federico Tombari, Samuele Salti és Luigi Di Stefano. “Unique shape context for 3d data description”. *Proceedings of the ACM workshop on 3D object retrieval - 3DOR '10*. ACM Press, 2010.

[49] Ulrich Hillenbrand és Alexander Fuchs. “An experimental study of four variants of pose clustering from dense range data”. *Computer Vision and Image Understanding* 115.10 (2011. okt.), 1427–1448. old.

[50] Kevin Lai, Liefeng Bo, Xiaofeng Ren és Dieter Fox. “A large-scale hierarchical multi-view RGB-D object dataset”. *2011 IEEE international conference on robotics and automation*. IEEE. 2011, 1817–1824. old.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot és E. Duchesnay. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), 2825–2830. old.

[52] Radu Bogdan Rusu és Steve Cousins. "3D is here: Point cloud library (PCL)". *2011 IEEE international conference on robotics and automation*. IEEE. 2011, 1–4. old.

[53] Samuele Salti, Federico Tombari és Luigi Di Stefano. "A Performance Evaluation of 3D Keypoint Detectors". *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*. IEEE, 2011. máj.

[54] Laura Bahiense, Gordana Manić, Breno Piva és Cid C. de Souza. "The maximum common edge subgraph problem: A polyhedral investigation". *Discrete Applied Mathematics* 160.18 (2012. dec.), 2523–2541. old.

[55] Marianna Madry, Carl Henrik Ek, Renaud Detry, Kaiyu Hang és Danica Kragic. "Improving generalization for 3D object categorization with Global Structure Histograms". *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012. okt.

[56] Ben Glocker, Shahram Izadi, Jamie Shotton és Antonio Criminisi. "Real-time RGB-D camera relocalization". *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2013, 173–179. old.

[57] Yulan Guo, Ferdous Sohel, Mohammed Bennamoun, Min Lu és Jianwei Wan. "Rotational Projection Statistics for 3D Local Surface Description and Object Recognition". *International Journal of Computer Vision* 105.1 (2013. ápr.), 63–86. old.

[58] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Robot Manipulation*. Springer Berlin Heidelberg, 2013.

[59] Silvio Filipe és Luís A. Alexandre. "A comparative evaluation of 3D keypoint detectors in a RGB-D Object Dataset". *2014 International Conference on Computer Vision Theory and Applications (VISAPP)* 1 (2014), 476–483. old.

[60] R. Hänsch, T. Weber és O. Hellwich. "Comparison of 3D interest point detectors and descriptors for point cloud fusion". *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3 (2014. aug.), 57–64. old.

[61] Kevin Lai, Liefeng Bo és Dieter Fox. "Unsupervised feature learning for 3D scene labeling". *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, 3050–3057. old.

[62] Samuele Salti, Federico Tombari és Luigi Di Stefano. “SHOT: Unique signatures of histograms for surface and texture description”. *Computer Vision and Image Understanding* 125 (2014. aug.), 251–264. old.

[63] Ben Bellekens, Vincent Spruyt, Raf Berkvens, Rudi Penne és Maarten Weyn. “A Benchmark Survey of Rigid 3D Point Cloud Registration Algorithms”. *International Journal On Advances in Intelligent Systems* 1 (2015. jún.).

[64] Tolga Birdal és Slobodan Ilic. “Point Pair Features Based Object Detection and Pose Estimation Revisited”. *2015 International Conference on 3D Vision*. IEEE, 2015. okt.

[65] Sungjoon Choi, Qian-Yi Zhou és Vladlen Koltun. “Robust Reconstruction of Indoor Scenes”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[66] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, Jianwei Wan és Ngai Kwok. “A Comprehensive Performance Evaluation of 3D Local Feature Descriptors”. *International Journal of Computer Vision* 116 (2015. ápr.).

[67] Dirk Holz, Alexandru E. Ichim, Federico Tombari, Radu B. Rusu és Sven Behnke. “Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D”. *IEEE Robotics & Automation Magazine* 22.4 (2015. dec.), 110–124. old.

[68] Dirk Holz, Alexandru E. Ichim, Federico Tombari, Radu B. Rusu és Sven Behnke. “Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D”. *IEEE Robotics Automation Magazine* 22.4 (2015), 110–124. old.

[69] Peter van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde és Romulo Gonçalves. “Massive point cloud data management: Design, implementation and execution of a point cloud benchmark”. *Computers & Graphics* 49 (2015), 92–125. old.

[70] Sai Manoj Prakhya, Bingbing Liu és Weisi Lin. “B-SHOT: A binary feature descriptor for fast and efficient keypoint matching on 3D point clouds”. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), 1929–1934. old.

[71] Samuele Salti, Federico Tombari, Riccardo Spezialetti és Luigi Di Stefano. “Learning a Descriptor-Specific 3D Keypoint Detector”. *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015. dec.

[72] Jens Garstka és Gabriele Peters. “Evaluation of Local 3-D Point Cloud Descriptors in Terms of Suitability for Object Classification”. *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS - Science, and Technology Publications, 2016.

[73] Timo Hackel, Jan D. Wegner és Konrad Schindler. “Contour Detection in Unstructured 3D Point Clouds”. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. jún.

[74] Stefan Hinterstoisser, Vincent Lepetit, Naresh Rajkumar és Kurt Konolige. “Going Further with Point Pair Features”. *Lecture Notes in Computer Science* (2016), 834–848.

[75] S. Hamidreza Kasaei, Luis Seabra Lopes, Ana Maria Tome és Miguel Oliveira. “An orthographic descriptor for 3D object learning and recognition”. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016. okt.

[76] Jianhui Nie. “Extracting feature lines from point clouds based on smooth shrink and iterative thinning”. *Graphical Models* 84 (2016. márc.), 38–49. old.

[77] Siddharth Srivastava és Brejesh Lall. “3D binary signatures”. *ICVGIP ’16*. 2016.

[78] Jiaolong Yang, Hongdong Li, Dylan Campbell és Yunde Jia. “Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (2016), 2241–2254. old.

[79] Qian-Yi Zhou, Jaesik Park és Vladlen Koltun. “Fast Global Registration”. *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, 766–782. old.

[80] Yu Zou, Tao Zhang, Xueqian Wang, Ying He és Jingyan Song. “BRoPH: A compact and efficient binary 3D feature descriptor”. *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (2016), 1093–1098. old.

[81] Anders Glent Buch, Lilita Kiforenko és Dirk Kraft. “Rotational Subgroup Voting and Pose Clustering for Robust 3D Object Recognition”. *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017. okt.

[82] Rémi Cura, Julien Perret és Nicolas Paparoditis. “A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing”. *ISPRS Journal of Photogrammetry and Remote Sensing* 127 (2017), 39–56. old.

[83] Zhen Dong, Zhen Dong, Bisheng Yang, Yuan Liu, Fuxun Liang, Bi jun Li és Yufu Zang. "A novel binary shape context for 3D local surface description". *Isprs Journal of Photogrammetry and Remote Sensing* 130 (2017), 431–452. old.

[84] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis és Xenophon Zabulis. "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects". *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2017).

[85] Sai Prakhya, Bingbing Liu, Weisi Lin, Kun Li és Yong Xiao. "On Creating Low Dimensional 3D Feature Descriptors with PCA". *TENCON 2017 - 2017 IEEE Region 10 Conference*. 2017. nov., 315–320. old.

[86] Sai Manoj Prakhya, Bingbing Liu, Weisi Lin, Vinit Jakhetiya és Sharath Chandra Guntuku. "B-SHOT: A Binary 3D Feature Descriptor for Fast Keypoint Matching on 3D Point Clouds". *Auton. Robots* 41.7 (2017. okt.), 1501–1520.

[87] Winal Prawira, Emir Nasrullah, Sri Ratna Sulistiyanti és FXA Setyawan. "The detection of 3D object using a method of a Harris Corner Detector and Lucas-Kanade Tracker based on stereo image". *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*. IEEE. 2017, 163–166. old.

[88] Alessio Tonioni, Samuele Salti, Federico Tombari, Riccardo Spezialetti és Luigi Di Stefano. "Learning to Detect Good 3D Keypoints". *International Journal of Computer Vision* 126.1 (2017. aug.), 1–20. old.

[89] Fang Wang és Zijian Zhao. "A survey of iterative closest point algorithm". *2017 Chinese Automation Congress (CAC)*. IEEE, 2017. okt.

[90] Jiaqi Yang, Q. Zhang, Ke Xian, Yang Xiao és Zhiguo Cao. "Rotational contour signatures for both real-valued and binary feature representations of 3D local shape". *Comput. Vis. Image Underst.* 160 (2017), 133–147. old.

[91] Liangjie Du, Huasong Min és Yunhan Lin. "CI-SHOT: A Statistical Method for Binary Feature Descriptor on 3D Point Clouds". *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2018, 373–379. old.

[92] Xian-Feng Han, Shi-Jie Sun, Xiang-Yu Song és Guo-Qiang Xiao. *3D Point Cloud Descriptors in Hand-crafted and Deep Learning Age: State-of-the-Art*. 2018.

[93] Tomáš Hodaň, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Fabian Manhardt, Federico Tombari, Tae-Kyun Kim, Jiří Matas és Carsten Rother. "BOP: Benchmark for 6D Object Pose Estimation". *Computer Vision ECCV 2018*. Springer International Publishing, 2018, 19–35. old.

[94] Jihad H'roura, Michaël Roy, Alamin Mansouri, Driss Mammass, Patrick Juillion, Ali Bouzit és Patrice Méniel. "Salient Spin Images: A Descriptor for 3D Object Recognition". *Image and Signal Processing*. Szerk. Alamin Mansouri, Abderrahim El Moataz, Fathallah Nouboud és Driss Mammass. 2018, 233–242. old.

[95] Giseop Kim és Ayoung Kim. "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map". *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018. okt.

[96] Haicheng Liu, P Van Oosterom, Martijn Meijers és Edward Verbree. "Management of large indoor point clouds: an initial exploration." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* (2018).

[97] Yue Pan, Bisheng Yang, Fuxun Liang és Zhen Dong. "Iterative global similarity points: A robust coarse-to-fine integration solution for pairwise 3d point cloud registration". *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, 180–189. old.

[98] Siwen Quan, Jie Ma, Fangyu Hu, Bin Fang és Tao Ma. "Local voxelized structure for 3D binary feature representation and robust registration of point clouds from low-cost sensors". *Inf. Sci.* 444 (2018), 153–171. old.

[99] Zi Jian Yew és Gim Hee Lee. "3DFeat-net: Weakly supervised local 3d features for point cloud registration". *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, 607–623. old.

[100] Qian-Yi Zhou, Jaesik Park és Vladlen Koltun. "Open3D: A modern library for 3D data processing". *arXiv preprint arXiv:1801.09847* (2018).

[101] Kevin H. Knuth. "Optimal data-based binning for histograms and histogram-based probability density models". *Digital Signal Processing* 95 (2019. dec.), 102581. old.

[102] Jiaxin Li és Gim Hee Lee. "USIP: Unsupervised stable interest point detection from 3D point clouds". *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, 361–370. old.

[103] Yunhan Lin, Yalan Sun és Huasong Min. "A General Gray Code Quantized Method of Binary Feature Descriptors for Fast and Efficient Keypoint Matching". *2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS)*. 2019, 1–7. old.

[104] Bin Lu és Yang Wang. "Matching Algorithm of 3D Point Clouds Based on Multiscale Features and Covariance Matrix Descriptors". *IEEE Access* 7 (2019), 137570–137582. old.

[105] Theresa Meyer és Ansgar Brunn. "3D Point Clouds in PostgreSQL/PostGIS for Applications in GIS and Geodesy". *Proceedings of the 5th International Conference on Geographical Information Systems Theory, Applications and Management - Volume 1: GISTAM,* INSTICC. SciTePress, 2019, 154–163. old.

[106] Sebastian Ochmann és Reinhard Klein. "Automatic Normal Orientation in Point Clouds of Building Interiors". *Advances in Computer Graphics*. Springer International Publishing, 2019, 556–563. old.

[107] Riccardo Spezialetti, Samuele Salti és Luigi Di Stefano. "Performance Evaluation of Learned 3D Features". *Image Analysis and Processing – ICIAP 2019*. 2019, 519–531. old.

[108] Richard Vock, Alexander Dieckmann, Sebastian Ochmann és Reinhard Klein. "Fast template matching and pose estimation in 3D point clouds". *Computers & Graphics* 79 (2019), 36–45. old.

[109] Bao Zhao, Xiaobo Chen, Xinyi Le és Juntong Xi. *A Comprehensive Performance Evaluation for 3D Transformation Estimation Techniques*. 2019.

[110] Ruqin Zhou, Xixing Li és Wanshou Jiang. "3D Surface Matching by a Voxel-Based Buffer-Weighted Binary Descriptor". *IEEE Access* 7 (2019), 86635–86650. old.

[111] Tomas Hodan, Daniel Barath és Jiri Matas. "EPOS: Estimating 6D Pose of Objects With Symmetries". *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020. jún.

[112] Tomáš Hodaň, Martin Sundermeyer, Bertram Drost, Yann Labbé, Eric Brachmann, Frank Michel, Carsten Rother és Jiří Matas. "BOP Challenge 2020 on 6D Object Localization". *Computer Vision – ECCV 2020 Workshops*. Springer International Publishing, 2020, 577–594. old.

[113] Ben Nassi, Yisroel Mirsky, Dudi Nassi, Raz Ben-Netanel, Oleg Drokin és Yuval Elovici. "Phantom of the ADAS: Securing Advanced Driver-Assistance Systems from Split-Second Phantom Attacks". *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020. okt.

[114] Andreas Nüchter és Kai Lingemann. "Robotic 3D scan repository". *Universität Osnabrück, Jacobs University Bremen, Julius-Maximilians-Universität Würzburg* (2020). Last accessed: March 03, 2020, `kos.informatik.uni-osnabrueck.de/3Dscans`.

[115] Caner Sahin, Guillermo Garcia-Hernando, Juil Sock és Tae-Kyun Kim. "A review on object pose recovery: From 3D bounding box detectors to full 6D pose estimators". *Image and Vision Computing* 96 (2020. ápr.), 103898. old.

[116] Jun Zhou, Yuanpeng Liu, Jinshan Liu, Qian Xie, Yuqi Zhang, Xusheng Zhu és Xiao Ding. "BOLD3D: A 3D BOLD descriptor for 6DoF pose estimation". *Computers and Graphics* 89 (2020. jún.), 94–104. old.

[117] Bao Zhao, Xiaobo Chen, Xinyi Le, Juntong Xi és Zhaohong Jia. "A Comprehensive Performance Evaluation of 3-D Transformation Estimation Techniques in Point Cloud Registration". *IEEE Transactions on Instrumentation and Measurement* 70 (2021), 1–14. old.

[118] S. Rusinkiewicz és M. Levoy. "Efficient variants of the ICP algorithm". *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. IEEE Comput. Soc.