

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Major Bence

Hatékony térrekonstrukció mélységkép felületeinek síkszegmensekkel való közelítésével

TDK dolgozat

Témavezető:
Engedy István

BUDAPEST, 2014.

Absztrakt

Mobilis robotok alkalmazásakor gyakran felmerül az igény, hogy a robot képes legyen a körülötte lévő tér érzékelésére. Egy erre alkalmas eszköz például a mélységinformációt nyújtó kamera. Mivel az ilyen eszközök ára az elmúlt néhány évben drasztikusan csökkent, és egyre szélesebb körben hozzáférhetőek, így mobilis robotokat is egyre gyakrabban láthatunk el velük. A mélységképet adó szenzorok térhódításával azonban szükségessé váltak olyan új, a színeképeken nem alkalmazható módszerek, amelyek segítségével gépi látással ruházhatjuk fel a robotokat, vagyis amivel objektumokat érzékelhetnek, vagy felépíthetik környezetük háromdimenziós modelljét.

A mélységkamerák elérhetősége miatt felmerül, hogy kisebb számítási kapacitással rendelkező roboton is térrekonstrukciót használjunk. Az elterjedt megoldások alkalmazhatóságát azonban korlátozza a nagy memória- és/vagy számítási igényük, ami a gyakorlatban azt jelenti, hogy nem működnek valós időben, amennyiben az azokat felhasználó robot nem rendelkezik a megfelelő erőforrásokkal. Szükség van tehát egy olyan algoritmusra, melyet az ilyen robotok is alkalmazhatnak a tér rekonstrukciójára.

Az általam fejlesztett módszer abból a feltételezésből indul ki, hogy az ember alkotta környezetekben található felületek jelentős része jól közelíthető síkokkal. Az algoritmus először a mélységképen szereplő felületelemekre illeszkedő síkok egyenletét határozza meg, majd a kapott síkokon megkeresi azokat a területeket – a síkok egy-egy szegmensét –, amely a valós felületelemet reprezentálhatja.

Mivel az egyik fő szempontunk az volt, hogy olyan eljárást fejlesszünk, melynek futási ideje úgy is kellően rövid, hogy nincsen egy nagy teljesítményű grafikus processzoron párhuzamosítva, ezért újszerű algoritmusokat ötvöztünk klasszikus képfeldolgozási módszerekkel, melyek kihasználják, hogy a mélységkamera által nyújtott mélységkép képpontjai között természetesen értelmezhető a szomszédsági viszony. A dolgozatban részletesen bemutatott, és valós adatokon illusztrált algoritmus alkalmas arra, hogy még egy gyengébb számítási teljesítményű robot látórendszerében felhasználva is felruhazza a robotot a környezet valós idejű, síkszegmensekkel történő rekonstrukciójának képességével.

Abstract

A recurring demand of mobile robot applications is to give robots the ability to sense their surroundings. Depth cameras are one of the examples which are suitable tools for this. As the price of the depth cameras has greatly decreased in the recent years, they became accessible for a much broader audience as before, which lead to them being used more often by mobile robots. However, with the widespread use of these sensors, new algorithms for computer vision are required, which are not applicable on regular color images. These algorithms give robots computer vision, so robots will be able to sense objects and reconstruct the three dimensional model of their surroundings.

The accessibility of depth cameras brings up the possibility of using space reconstruction in robots with smaller processing capabilities. However, the feasibility of the widespread approaches are bounded by their heavy processing and/or memory usage, which, in practice, means that they are not able to be used in real time, if the robot which utilizes them does not have the required resources. Hence, a new algorithm is required for space reconstruction, which can be applied on such robots.

Our developed method is based on the initial assumption that most surfaces in human-made environments can be approximated by plane segments. The first step of the proposed method is defining the equations of the planes which fit on the plane segments of the depth image. Then, the method proceeds to finding the areas on the previously found planes, which represent the planar surface elements of the reconstructed scene.

As one of my primary goals was to create a new algorithm, which does not rely on parallelizing on a graphical processing unit to have a short processing time, I have merged innovative approaches with classic image processing methods. On the depth images, a neighborhood connection can naturally be defined between the pixels, which is utilized by my method. The algorithm described in this paper is illustrated on examples with real life data, and proves to be feasible for using it for providing machine vision to a robot with small processing capabilities.

Tartalomjegyzék

1.	Bevezető	5
1.1.	Kapcsolódó munkák	5
1.2.	A felhasznált mélységkamera	8
2.	Az algoritmus leírása	10
2.1.	A mélységkép előkészítése	10
2.1.1.	A háromdimenziós pontok meghatározása	10
2.1.2.	A felületi normálisok meghatározása	11
2.2.	A síkfelületekre illeszkedő síkok meghatározása	12
2.2.1.	A pontok síkokra illeszkedésének vizsgálata	14
2.2.2.	A síktöredékek összevonása	15
2.3.	A megtalált síkokon fekvő felületelemek határvonalainak meghatározása	17
3.	Az algoritmus pontossága	20
3.1.	Az elkészített szimulációs környezet	20
3.2.	A pontosság mérése	21
3.2.1.	A normálvektorok összehasonlítása	21
3.2.2.	A két meghatározott síkszegmens távolságának meghatározása	22
3.2.3.	Az átlapolódó felület nagyságának meghatározása	22
3.3.	A konstruált tesztek és a kapott eredmények	23
3.3.1.	Távolságfüggés	24
3.3.2.	Szögfüggés	25
3.3.3.	Méretfüggés	27
3.3.4.	A futási idő mérése	28
4.	Összefoglalás	31
5.	Függelék	32
5.1.	Az eljárás pontosságát leíró diagramok	32
5.2.	Valós példák	34
6.	Irodalomjegyzék	39

1. Bevezető

1.1. Kapcsolódó munkák

A háromdimenziós környezetünk, tárgyak, esetleg emberek hatékony, számítógépes rekonstrukciója a mélységszenzorok elterjedésével egy elterjedt kutatási területté vált. Alkalmazása sokrétű: emberi viselkedés figyelésére alkalmas [1], emberi rehabilitációt támogathat [2], egy természetes ember-gép interfészként is funkcionálhat [3], vagy akár háromdimenziós scannerként tárgyak számítógépes modellje alkotható meg vele [4], de ezeken a példákon kívül robotok gépi látásában való felhasználása is kézenfekvő.



1. ábra - 3D rekonstrukció strukturált fényes látórendszerrel [4]

A dolgozatban ismertetett módszer fő célja, hogy egy mobilis robot látórendszerében valósídejű módon felhasználható legyen a környezet háromdimenziós rekonstrukciójára. Mindenképpen célszerű megjegyezni, hogy a valósídejű működéshez nincs szükség minden képkockát feldolgozni, hiszen a felhasznált mélységkamera 30 Hz-es mintavételi frekvenciája feleslegesen sűrű egy mobilis robot sebességéhez képest. Megfelelő feldolgozási sebességnek számít az, ha egy relatív gyorsan, az átlagos emberi sétálási sebességgel (körülbelül 1.5 m/s) közlekedő robot két feldolgozott kép között legfeljebb 10 centimétert halad előre. Ez 15 feldolgozott képet jelent másodpercenként, tehát az algoritmus futási ideje egy képre legfeljebb 70 ms kell legyen.

A térrekonstrukciós eljárások legelterjedtebb csoportjai az alapján, hogy a szenzorból vagy szenzorokból kinyert pontfelhőt miként kezelik:

- Igen részletes rekonstrukcióra képesek azok az eljárások, melyek a regisztrált háromdimenziós pontfelhőt nem alakítják át más formára, hanem a különböző felvételekből származó pontthalmazokat összeillesztik [5]. A megközelítés hátránya, hogy igen számításigényes.

- Egy másik nagy csoport a voxel modell alapú megközelítéseket tartalmazza [6]. Lényege, hogy előre partícionálja a teret egy rácsszerkezet mentén, és amennyiben egy mért pont egy térrészbe esik, az a térrész „foglalt” lesz, ott tömör test található. A rekonstrukció felbontása a rácsszerkezet finomságától függ. Az eljárás hátránya, hogy egyrészt előre behatárolja a lehetséges teret, másrészt pedig igen memóriaigényes.
- Szintén elterjedt csoportot alkotnak a modell alapú megközelítések. Az ilyen módszerek a mért pontokat parametrikus felületekkel kísérlik meg közelíteni, és a teret ilyen elemekből építi fel.

A kifejlesztett algoritmusom modell alapú megközelítés. Mivel elsősorban mobilis robot beltéri navigációjának segítésére készült, azzal a feltételezéssel élek, hogy az ember által készített környezetekben megtalálható felületek nagy része síkszegmensekkel közelíthető. Ezen okból kifolyólag a módszerem a felhasznált mélységképeken olyan síkokat keres, melyek a rekonstruálni kívánt környezet felületeire simulnak. Miután a síkokat meghatározta, egy alkalmas körvonallal síkszegmenseket vág ki belőlük, amik reprezentálni fogják a környezet sík felületeit.

Síkfelületek keresésének feladatára a szakirodalomban szinte kivétel nélkül három különböző módszert használnak. Az egyik ilyen módszer a régiónövelés módszere (Region Growing). Alkalmas kezdőpontokból indulva bővítjük a megtalált síkokat addig, míg már nem tudunk egyik csoporthoz sem hozzávenni új pontot egy megadott hasonlóságmérték átlépése nélkül.

Egy másik, elterjedt szemlélet a Hough transzformáció alkalmazása síkokra, tehát a rekonstruálni kívánt tér minden pontja szavaz arra, hogy milyen síkokra illeszkedik. A szavazás az ún. Hough-térben történik, vagyis esetünkben a síkokat meghatározó 4 dimenziós paraméterterben. A Hough-tér a felbontás függvényében negyedfokú hatvány módjára skálázódik ezért, a céljainkhoz képest túl nagy memóriaigénnyel rendelkezik.

A Hough transzformáció gyorsítására a Randomized Hough Transformationt (RHT) használják, ami nem minden pontot használ fel, hanem véletlenszerűen választ a ponthalmazból (síkok esetében ponthármasokat). Egy-egy ilyen választás egy konkrét síkra szavaz, és ha egy sík elegendő szavazatot kapott, bekerül a megtalált síkok közé, a síkhoz közel lévő pontok pedig törlésre kerülnek a még feldolgozandó pontok közül. A RHT alkalmasnak bizonyul a háromdimenziós rekonstrukcióra való felhasználásra [7].

A harmadik, szélesebb körben elterjedt módszer a RANSAC algoritmus [8] alkalmazása. Ennek lényege, hogy kiválaszt néhány pontot véletlenszerűen (síkok esetében 3-at), majd ezekre illeszt síkot. Minden síkjelölthöz egy pontszámot rendel, mely azt írja le, hogy mennyire jól reprezentálja a modell a pontokat. Végül a legjobb pontszámú síkokból származtatódik az eredmény [9].

Ezen módszereket általában kombinálják, hogy olyan eljárások szülessenek, melyek mindhárom szemléletmód előnyeit fuzionálják. Ilyen például a [10]-ben közölt módszer. Ebben egy olyan algoritmus kerül ismertetésre, melyben a mélységképet csempékre osztják, majd a csempéken belül véletlenszerű pontokat kiválasztva alkalmazzák a Hough módszert. A nyertes sík alapján pedig region growing módszerrel alakítják ki a végleges eredményt. A leírt eljárás hátránya, hogy egy kép feldolgozása közel dupla annyi számítási időt igényel, mint a dolgozatomban közölt módszer.

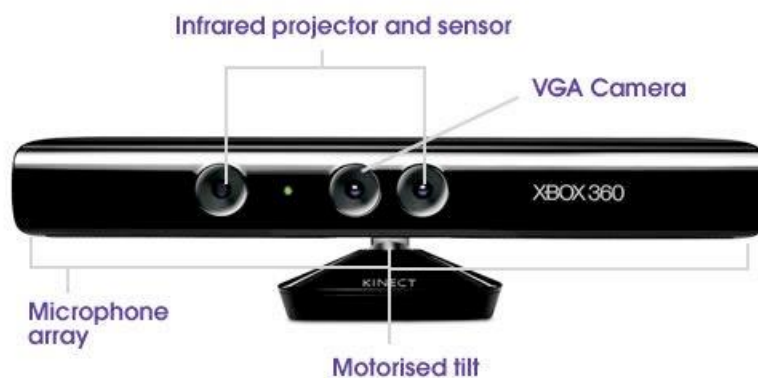
Egy másik megközelítést ír le [11], amiben Hough transzformációt használnak a síkszegmensek szegmentálására. Módszerük futási ideje meghaladja a 150 ms-ot, tehát alkalmazhatósága valósidejű környezetben erősen korlátozott, a bevezetőben kitűzött célt nem teljesíti.

A térrekonstrukcióhoz közel álló terület a mérnöki visszafejtés, vagy „reverse engineering”, melynek lényege, hogy egy legyártott tárgyról (pl.: alkatrész) készült pontfelhőből meghatározható az a legvalószínűbb modell, amelyből származhat a pontfelhő. Ez a feladat igen közel áll a térrekonstrukció feladatához, és sok esetben hasonló eszközkészlettel is oldható meg a két probléma, amiből következik, hogy az egyik területen elért eredmények a másik területen is alkalmazhatóak lehetnek. Erre utal például a [12]-ben leírt algoritmus, amely Hough transzformációt alkalmaz mérnöki visszafejtésre. A pontfelhőkben keresett parametrikus felület a henger, ami egyes emberi környezetekben is előfordulhat (például oszlopok formájában). Egy, a mérnöki visszafejtésben használt koncepció, ami térrekonstrukcióban is felhasználható, a geometriai kényszerek figyelembe vétele. Egy ilyen megoldást közöl [13], melyben a szerző egy olyan módszert ír le, mely a kialakított modelleket úgy alkotja meg, hogy automatikusan feltételez bizonyos szimmetriákat, valamint olyan jellegű heurisztikákkal él, mint például a „közel párhuzamos” felületelemek párhuzamossá tétele. Habár hasonló feltételezésekkel lehet élni ember alkotta környezetekben, nem jellemző, hogy a térrekonstrukciós eljárások használnának ilyen jellegű heurisztikákat.

A szakirodalomban ismertetett térrekonstrukciós eljárások fő problémája, hogy törekednek arra, hogy minél pontosabban határolják körül a síkfelületeket, akár annak rovására is, hogy a kifejlesztett algoritmus végül nem lesz valós időben alkalmazható, vagy csak grafikus számítási egységet (GPU) alkalmazva lehet használni. A mi célunk ezzel szemben egy olyan módszer fejlesztése, ami egy robot szerényebb feldolgozási képességű látórendszerében, működés közben is alkalmazható a környezet felmérésére. Az eljárás tehát tulajdonképpen egy modul lesz, mellyel a robot az őt tartalmazó környezet főbb jellemvonásait megismerheti, és ami bővíthető a pontos alkalmazásból fakadó igények függvényében.

1.2. A felhasznált mélységkamera

Mivel a különböző mélységszenzorok által nyújtott adat nagyban függ attól, hogy milyen konkrét szenzorból származik, ezért fontos, hogy az algoritmus fejlesztése előtt meghatározzuk, hogy pontosan milyen mélységkamerát használunk. A választás a Microsoft Kinect kamerára esett, ugyanis a többi mélységkamerához viszonyítva igen elérhető árú eszköz. Felbontása és mintavételi frekvenciája a térrekonstrukcióra alkalmas, ezen kívül USB porton keresztül csatlakoztatható a robot fedélzeti számítógépéhez.



2. ábra - A Kinect mélységkamera [14]

Az eszköz legfőbb alkotóelemei:

- infravörös projektor
- infravörös kamera
- színes kamera
- motoros beavatkozó, mellyel a Kinect fejét kis mértékben mozgatni lehet

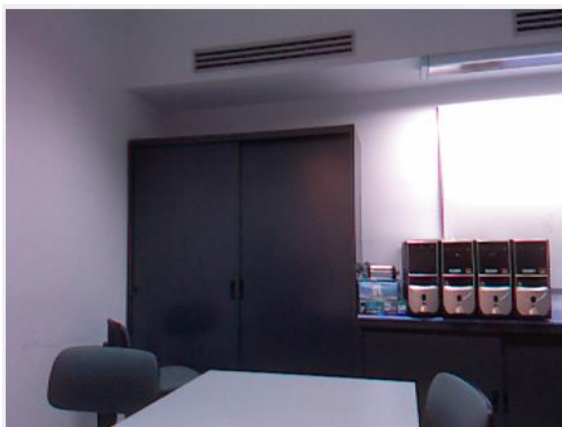
Működését tekintve strukturált fényes látórendszert alkalmaz. A projektor egy megadott mintát vetít ki az infravörös spektrumban, majd ezt az infravörös spektrumot érzékelő kamera érzékeli, és a Kinectben lévő célprocesszor feldolgozza. A kivetített és az érzékelt minta közötti eltérésből, vagy torzulásból számolja ki a Kinect a pontok térbeli elhelyezkedését.



3. ábra - A Kinect által kivetített mintázat [15]

A mélységkamera 30 Hz-es frekvenciával készít felvételeket, 640x480 pixeles felbontásban. A mélységérzékelés finomsága 11 bit. Látószöge vízszintesen 58°, függőlegesen 45°. Dokumentáció szerinti működési tartománya 80 cm és 3.5 m közötti. A műszaki adatok és az elérhető beszerzési ár következtében célunknak teljesen megfelelő.

A Kinect mélységképéről muszáj megemlíteni, hogy a képe az általános mérési zajon kívül azzal is terhelt, hogy a kivetített mintából nem mindenhol képes visszaállítani a mélységinformációt. Ez a mélységképeken 0 értékkel jelzett hiányzó adat formájában manifesztálódik, ami azt jelenti, hogy az adott pontban valamilyen okból kifolyólag nem lehetett meghatározni a távolságot. Olyankor lép fel biztosan, ha egy pont a projektorhoz képest takarásban van, vagy a felület tükröződik, esetleg olyan a felület, hogy elnyeli az infravörös spektrumú fényt, de véletlenszerűen is megjelenhet, a távolság növekedésével egyre valószínűbben. Amennyiben egy objektum túl közel helyezkedik el, az is hiányzó adatként jelenik meg a képen. További hátrány, hogy mivel a természetes napfény is tartalmaz infravörös komponenst, külső helyszínen az eszköz nem használható.



4. ábra – Az algoritmus fázisainak demonstrálásához használt példa

A 4. ábra által bemutatott térről készült mélységképen demonstrálom a fejlesztett algoritmus lépéseit.

2. Az algoritmus leírása

A fejlesztett módszer kétfázisú. Az első fázisban meghatározzuk a felületeket alkotó síkokat, majd a második fázisban megkeressük azokat a síkon fekvő határvonalakat, melyek a mélységképen látható felületek szélei. A teljes folyamat kimenete tehát minden síkszegmensre egy-egy háromdimenziós pontlista, melyek már nem csak a képsíkon, de a térben is körülhatárolják a látható felületelemeket.

2.1. A mélységkép előkészítése

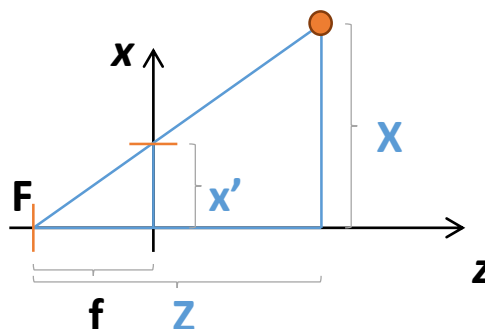
Ebben a speciális esetben, hogy a feldolgozott kép mélységinformációt hordoz, tulajdonképpen egy háromdimenziós pontfelhőt kapunk, melyek pontjai a nekik megfelelő pixelek szomszédosságait öröklik. Ilyen pontfelhőkön klasszikus képfeldolgozási algoritmusok is alkalmazhatóak úgy, hogy azok koncepcióját, alapgondolatát megtartva, szín- helyett mélységértékeket kezelünk.

Ismételve a már lefektetett célt, a fő elgondolás az, hogy létrehozzak egy algoritmust, amit egy gyengébb számítási kapacitással rendelkező robot is felhasználhat a látórendszerében. Hogy párhuzamosítás nélkül is egy megfelelő időkereten belül maradjon a módszerem futási ideje, le kell mondanunk a tökéletes részletességről. A feladat tehát az, hogy a képen lévő nagy kiterjedésű síkfelületeket megtaláljuk.

Ez okból az algoritmus első lépése a mélységkép alulmintavételezése, aminek eredménye, hogy a futási idő drasztikusan lecsökken olyan áron, hogy a kisebb síkszegmensek elveszhetnek. Az alulmintavételezés oly módon történik, hogy az eredeti mélységképet mindkét dimenzió irányában 8-ad részére csökkentem, és a kapott képen lévő pixelek értékeit az eredeti képből, nearest neighbour módszerrel veszem. Az algoritmus leírását taglalva a „kép” kifejezés ezentúl az alulmintavételezett képre utal.

2.1.1. A háromdimenziós pontok meghatározása

Következő lépésként kiszámolom a pixelpontokhoz tartozó háromdimenziós pontot. Ez a pinhole kamera modell szerint történik, a hasonló háromszögek elvét felhasználva.



5. ábra – A pinhole kameramodell szerint történő vetítés 2D-ben

Jelölések: F – fókuszpont; f – fókusz távolság; Z – a pont Z koordinátájának értéke; X – a pont X koordinátájának értéke; x' – a pont vetületének x koordinátája a képen

A kép alapján felírható az arányosság:

$$\frac{X}{Z} = \frac{x'}{f}$$

A pont képen betöltött pozíciója, valamint a fókuszponttól való távolsága ismert (x' , Z). A fókusztávolság (f) pedig a Kinect specifikációjából kiolvasható. Így a kérdéses pont térben betöltött pozíciójának X koordinátája:

$$X = \frac{x'}{f} * Z$$

Ezzel analóg módon számolható az Y komponens is, tehát pontosan meg tudjuk határozni minden pont helyzetét a háromdimenziós térben.

Mivel a kép pontjai kölcsönösen egyértelműen határozzák meg a háromdimenziós pontfelhő pontjait, valamint a használt algoritmusok képfeldolgozási jellege miatt a „pixel”, valamint a „pont” kifejezések ugyanarra utalnak, a dolgozat szóhasználata is ezt tükrözi.

2.1.2. A felületi normálisok meghatározása

A kapott háromdimenziós pontfelhő pontjaira a térben elfoglalt pozíciójukon kívül más információt is ki tudunk számítani. Ahhoz, hogy kialakítsunk pontcsoportokat, melyek egy síkra illeszkednek, hatalmas segítséget nyújt az, ha tudjuk, hogy egy adott pontban milyen a felületi normális. Az egymáshoz közel eső pontok ugyanis nem feltétlenül esnek egy síkra, de a felületi normális ismeretében már egyszerűbben tudjuk őket csoportokra osztani.

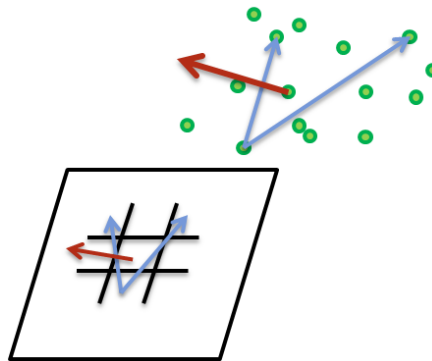
A felületi normális közelítése egy adott pontban úgy történik, hogy síkokat illesztünk a szomszédjainak különböző részhalmazaira, és a kialakított síkok normálvektorának az átlaga alkotja a felületi normális.

A 3D geometriában általában több módon is értelmezhető a szomszédossági fogalom. Használható például a k -Nearest Neighbours, vagy a Fixed Distance Neighbours módszer [16]. Az ily módon definiált szomszédsági halmaz meghatározása a pontfelhő összes pontjára azonban olyan számításokat igényelne, melyek nem használják ki a pontfelhő származtatására használt mélységkép tulajdonságait, vagyis azt, hogy az egyes képpontok között egyértelmű a szomszédosság. A pontfelhő pontjai öröklik a képen lévő vetületük szomszédosságait.

A mélységképen természetesen értelmezhető szomszédossági viszonyt kihasználva meghatározható a felületi normális egy közelítése. Ezt a közelítést csak a teljesen lokális környezetből számítva azonban a Kinect mérési zajával erősen terhelt megoldásra jutnánk. Ezen is segít az alul-mintavételezés.

Minden képpont 8 szomszédja közül kiválasztunk néhányszor 3-3 darabot, előre definiált módon. Minden ilyen hármasra síkot illesztünk, majd az így módon kapott síkok normálvektorainak átlaga lesz a vizsgált középső pontban vett felületi normálvektor. Tehát ha $\text{pos}(x,y)$ jelöli a kép x,y indexpárral meghatározott pontjához tartozó három dimenziós pontot, akkor egy ilyen hármas például a $p1 = \text{pos}(x+1,y-1)$, a $p2 = \text{pos}(x-1,y-1)$, valamint a $p3 = \text{pos}(x,y+1)$. Az erre a három pontra illeszkedő sík normálvektora $(n(x,y))$:

$$n(x,y) = (p1 - p3) \times (p2 - p3)$$



6. ábra - Felületi normálisok számítása

A szomszédhármasok kiválasztásánál csak olyan hármasokat fogadunk el, melyek mélységértékei között nincs egy megadott küszöbnél nagyobb eltérés. Ha ugyanis lenne ilyen eltérés, az arra utalna, hogy a hármas egyik pontja valószínűleg egy távolabbi síkon helyezkedik el, csak szomszédos pixelekre kerültek a kép elkészítésekor fellépő vetítés következtében. Ezt a küszöbértéket úgy állítottam be, hogy a képen csak kis szögben látszódó falakat még egységesnek érzékeljük, így 50 cm lett a végleges, heurisztikus küszöbérték.

Összefoglalva tehát az eddigi, előkészítő lépéseket, minden képponthez tartozik egy háromdimenziós pont, egy felületi normális vektor, valamint értelmezett az ilyen képpontok között a szomszédosság.

2.2. A síkfelületekre illeszkedő síkok meghatározása

A mélységkép előfeldolgozásával megkaptuk az összes képpontra a síkra illeszkedés eldöntéséhez szükséges információt. Egy inkrementális algoritmust alkalmazunk, mely minden képpontra meghatározza, hogy az melyik síkra illeszkedik. Ehhez azzal a feltételezéssel élünk, hogy ha két pont közel helyezkedik el a térben, akkor a képen lévő vetületük is közel van. Ennek gyakorlati következménye, hogy egy adott síkon egymás mellett lévő pontok a képen is egymás mellett lesznek, tehát a síkok kialakításához nem szükséges különböző részhalmazokat kiválogatni a kép pontjaiból és

úgy vizsgálni a közös, legvalószínűbb síkra illeszkedést (Hough transzformáció, RANSAC algoritmus), elegendő csupán a szomszédos pontok közös síkra illeszkedését vizsgálni.

Az algoritmus fontos tulajdonsága, hogy a síkokat, mint *csoportokat* kezeli. Egy ilyen csoporthoz (vagy síkhoz) tartozik tehát azon pixelek halmaza, amiket már megvizsgált az eljárás, és az adott csoportra illeszkednek. Továbbá a csoport fogalom tárolja a sík leírását is a sík normálvektorának, valamint egy síkra illeszkedő pontnak a formájában. Érdekes megjegyezni, hogy ez a síkra illeszkedő pont nem egy, a csoportba tartozó pont, hanem a síkot meghatározó paraméterek egyikét jelöli.

A csoportok kialakításához a Connected Component labeling (esetünkben csoportoknak nevezzük a komponenseket), valamint a Region Growing algoritmusokat ötvöztem és alakítottam át úgy, hogy a céljainknak megfeleljen. A kialakított módszer főbb jellemzői:

- pixelek helyett a 3D pontfelhő pontjain dolgozik, a mélységkép szomszédossági viszonyai alapján
- a komponenseket (csoportokat) modellekkel ruháztam fel, melyek az algoritmus futásának előrehaladásával dinamikusan változnak, hogy a csoport elemeit a legjobban reflektálják
- két szomszédos csoport összehasonlításakor nem csak a szomszédos pontok szerint vizsgálom az egyezést, hanem a pontokat tartalmazó csoportok modellje szerint

A módszer sorfolytonosan minden képpontot feldolgozva megvizsgálja, hogy *illeszkedik-e* arra a síkra, ami tartalmazza a bal oldali szomszédját, majd ugyanezt megteszi a fölötte lévő szomszédjának csoportjával is. Ha mindkét síkra (vagy csoportra) illeszkedik, akkor összevonja a két síkot. Ha csak az egyikre, akkor hozzáadja annak síkjához az adott pontot. Ha viszont egyikre sem, akkor új síkot definiál az adott pontból. Ezen három művelet a következőképpen határoztam meg:

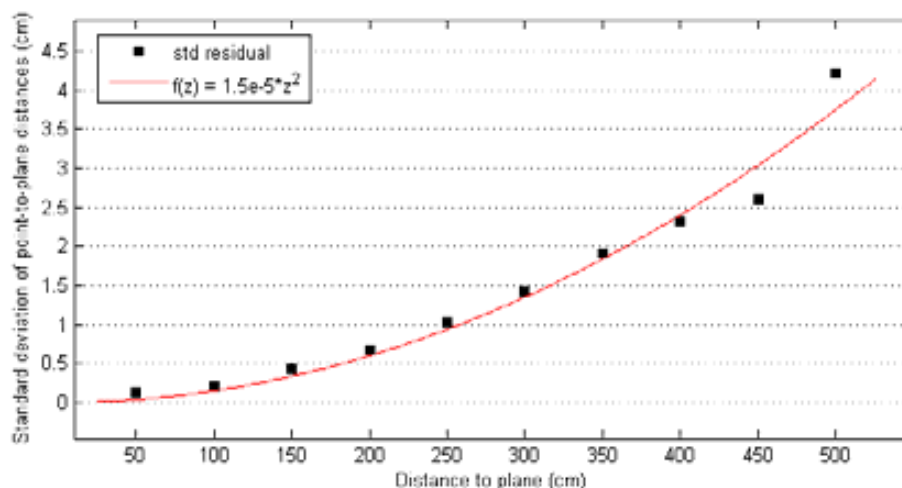
- Két sík összevonásakor létrehozunk egy új síkot, és átállítjuk az összes, két összevonandó síkba tartozó pont referenciáját, hogy erre az új síkra mutasson. Az új sík normálvektora és tárolt, az illeszkedő pontja a két összevont síkból származtatott, a csoportokba tartozó pontok száma szerinti súlyozott átlagképzéssel.
- Pont síkhoz való hozzáadásakor feljegyezzük, hogy a ponthoz melyik sík tartozik, valamint módosítjuk a sík normálisát és a tárolt illeszkedő pontot úgy, hogy az új ponttal vett ponthalmaz normálvektorainak átlaga a sík normálisa, a ponthalmaz súlypontja pedig a síkra illeszkedő pont legyen.
- Új sík definiálásakor a létrehozott új csoportba egyetlen pont fog (egyelőre) tartozni, az új pont. Ez a pont a síkra illeszkedő pont is. A sík normálvektora a pontban számított felületi normális.

Egy sík normálvektora tehát mindig a hozzá tartozó pontok normálvektorainak átlaga, a ráilleszkedő pontja pedig a hozzá tartozó pontok átlaga.

2.2.1. A pontok síkokra illeszkedésének vizsgálata

Egy adott pont csoportba való illeszkedésének eldöntése az algoritmus központi kérdése. Hogy meghatározzuk, hogy a vizsgált pont normálvektorát és pozícióját tekintve illik-e a síkra, ahhoz figyelembe kell venni a Kinect mélységkamera mérési zaját.

A mért mélységértékek természetesen zajjal vannak terhelve, tehát nem hagyatkozhatunk arra, hogy a sík egyenletét a pont kielégíti-e. Definiálunk egy valamilyen széles sávot a sík körül, amibe ha beleesik a pont, akkor pozícióját tekintve a síkhoz tartozik. A sáv szélessége azzal van összefüggésben, hogy milyen mértékű zaj van jelen a mérésekben. Mások már foglalkoztak a Kinect mérési pontosságával, egy ilyen vizsgálat eredményeit [17] használtuk fel az algoritmus küszöbértékeinek beállításához. A küszöbérték a távolság függvénye, pontosabban a felhasznált függvény aktuális távolsághoz tartozó értékének háromszorosa.



7. ábra - A Kinect mérési hibája a távolság függvényében [17]

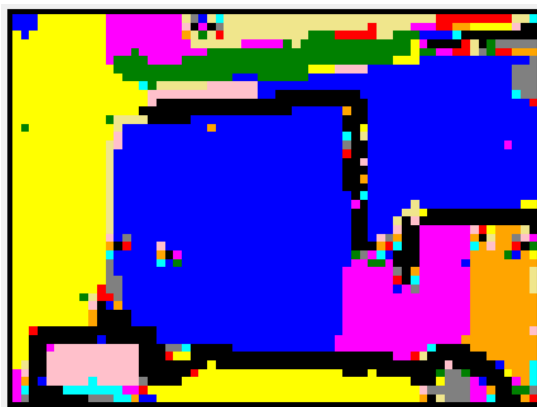
A síkra akkor illeszkedik tehát egy pont a pozícióját tekintve, ha a síktól való távolsága nem lépi át az illesztett függvényből származtatott küszöbértéket.

A pontok között értelmezett szomszédossági viszony olyan plusz információt hordoz, melyet felhasználhatunk az illeszkedés megállapításakor. A pont egy lokális környezete alapján megállapítható, hogy az adott pontban hogyan áll a felületi normális, ami azért fontos, mert a pont egy ehhez hasonló normálvektorú síkra illeszkedik. A felületi normálisokat már kiszámoltuk, és minden pontra tároltuk. Tehát ha egy pont síkhoz való tartozását vizsgáljuk, azt is figyelembe kell venni, hogy a pontban lévő felületi normális és a sík normálisa között ne legyen túl nagy eltérés. Ezt a küszöbértéket

10°-ra állítottam. Azonban ez a küszöb nem veszi figyelembe azt a bizonytalanságot, ami a mérési zajból adódik, ezért egy heurisztikus, távolságfüggő taggal bővítettem.

Egy pont tehát akkor tartozik egy sík felülethez, ha mind a pozícióját tekintve elég közel helyezkedik hozzá, mind pedig a hozzá tartozó felületi normális sem tér el egy megadott küszöbnél jobban a sík normálisától.

A leírt módon feldolgozunk minden pontot, elvégezzük a definiált vizsgálatokat és műveleteket, és így kialakulnak a felületekre illeszkedő síkok. Az eredmény látható a következő ábrán, mely úgy értelmezendő, hogy ha két képpont ugyanolyan színnel jelenik meg, akkor ugyanarra a síkra illeszkednek az algoritmus szerint. Fontos azonban megjegyezni, hogy a megjelenítéshez véges számú színt használok, így ezzel a módszerrel csupán azt láthatjuk, hogy milyen szomszédos régiók tartoznak biztosan különböző síkokhoz. Két, a képen nem szomszédos sík felület az algoritmus működéséből adódóan nem tartozhat egyazon csoportba, így ha a minta képen azonos színnel is jelennek meg, biztosan tudható, hogy az csak a korlátos megjelenített színek miatt van így.



8. ábra - Az algoritmus részeredménye

2.2.2. A síktöredékek összevonása

Az eddig leírtak szerint fennállhat az az eset, hogy egy vizsgálatkor egy pont nem tartozik egy szomszédos csoportba, viszont később olyan síkhoz fog tartozni, ami akár ugyanazt a sík felületet is reprezentálhatja, mint a korábbi vizsgált csoport. Hogy ezt kiküszöböljük, össze kell vonni azokat a síkokat, amelyek valószínűleg ugyanarra a felületre simulnak.

Az összevonások elvégzéséhez először konstruálok egy gráfot a már kinyert síkokból. Minden megtalált csoporthoz felveszek egy gráf csúcsot, és akkor fut két csúcs között él, ha a két, általuk képviselt csoport szomszédos, vagyis létezik olyan képpont, amelyik az egyik csoportba tartozik, egy vele szomszédos pixel viszont már a másikba. Ezen a gráfon egyesítek szomszédos csomópontokat akkor, ha a csomópontokhoz tartozó síkok is hasonlóak.

A feladat azonban komplexebb, mint hogy az összes csomópont-párt (a gráfban élt) egyszer vizsgáljuk meg, hiszen az összevonások következtében a hasonlósági viszonyok megváltozhatnak. A kitalált algoritmus, ami megoldja a problémát, úgy működik, hogy megvizsgál minden csúcsot, hogy van-e olyan szomszédja, amivel összevonható. Ha sikerül összevonnia így két csúcsot, akkor újra meg kell vizsgálni minden szomszédját mind az eredetileg vizsgált csomópontnak, mind az újonnan beolvasztottnak. A feladatot megoldó pszeudokód:

```

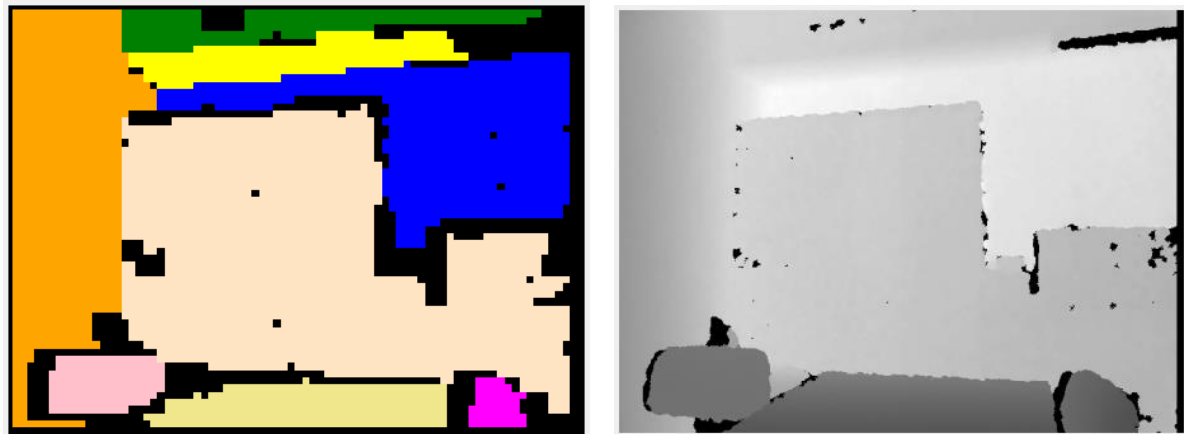
N(v): v csúcs szomszédjainak halmaza
FIFO := {};
Vizsgalando := összes csúcs;
Kimenet := {};
while (Vizsgalando nem üres)
    v = Vizsgalando egy eleme;
    FIFO += N(v);
    foreach v' in FIFO
        if (hasonlo(v,v'))
            v* = merge(v,v');
            FIFO = {};
            FIFO += {N(v) \ v'};
            FIFO += {N(v') \ v};
            v = v*
        else
            FIFO -= v'
    endforeach
    Vizsgalando -= v;
    Kimenet += v;
endwhile

```

Miután a szomszédos összevonható síkokat összevontuk, maradt sok apró síkelem, amiket az algoritmus azért talált, mert mindenre síkot próbál illeszteni. Ezeket a kicsi, néhány pixel nagyságú csoportokat eldobjuk. A kis síkok eldobásával természetesen fennáll annak a veszélye, hogy a robot navigációjához fontos információt veszítünk el. Ez azonban nem okoz nagy problémát, hiszen egyrészt kevés adat alapján számítottak, így a normálvektoruk iránya nem megbízható, másrészt pedig a fel nem dolgozott pontokat megjelölhetjük, és később a robot navigációjának további bemenő paraméterei lehetnek. Továbbá pedig ha a robot a mozgása közben közelebb jut egy ilyen eldobott síkhoz, akkor már nagyobbban fogja érzékelni az algoritmus, és várhatóan az új kép alapján már nem fogjuk eldobni.

A leírt lépések után a síkok száma (természetesen a feldolgozott tér felépítésétől függően) körülbelül 10-es nagyságrendre esik le. Ez már olyan kis számosságú halmaz, hogy páronként ellenőrizhető, hogy van-e két olyan csoport, melyek ugyanazon síkot írják le. Ha igen, akkor a normálvektorukat megváltoztatjuk a két sík, tartalmazott pontszám szerinti súlyozott átlag normálisára, hogy a rekonstrukciónál valóban teljesen párhuzamosak legyenek.

Az síkok kigyűjtésének végeredménye az ábrán látható:



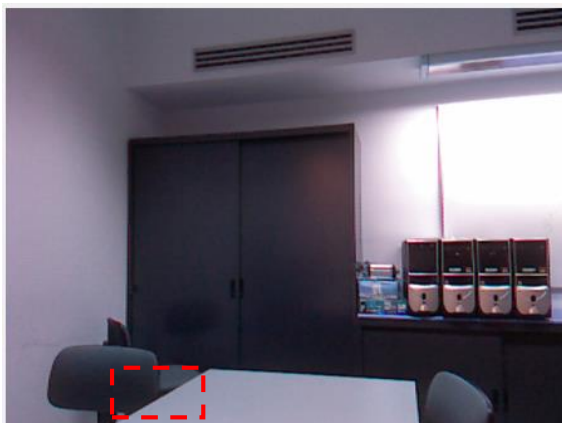
9. ábra - A síkkeresés eredménye

2.3. A megtalált síkokon fekvő felületelemek határvonalainak meghatározása

Mint az a megtalált síkokból leszűrhető, az algoritmus a legnagyobb felületeket gond nélkül megtalálja, az eredeti mélységkép alul-mintavételezésétől függetlenül. A síkok síkszegmensekre való határolásához azonban visszatérhetünk az eredeti képre, hogy nagy felbontású, pontosabb körvonalat kapjunk. A nagyfelbontású kép feldolgozása azonban jelentős hátrányokkal is jár. Ezen hátrányok ismertetéséhez először érdemes átgondolni, hogy hogyan hordoz információt a mélységkép.

A mélységképen az objektumokat több dolog is határolja. Határtípus például az, hogyha diszkontinuitás található a mélységinformációban. Ez azt jelöli, hogy az objektum a képen mellette található objektumtól távolabb helyezkedik el, csupán a képen jelennek meg szomszédosként. Egy másik típus a Kinect hiányzó adatot jelző 0 értéke lehet, a bemutatott mélységképen ezek fekete foltok formájában jelennek meg. Ezek a foltok sokszor nem szabályos formájúak, és a területükhöz képest viszonylag nagy a kerületük. Az utolsó határvonalatípust a megtalált síkok metszésvonalai alkotják.

Ezen felsorolt határtípusokból állnak össze elvileg a felületszegmensek határai. Azonban a gyakorlatban felléphetnek olyan szituációk, amelyek egyik esetben sem tartoznak bele, csupán az emberként fennálló preconcepcióinkból tudjuk, hogy határvonalnak kell lenni bizonyos helyzetekben. Erre látható egy példa az alábbi ábrán:



10. ábra – A széktámla széle, valamint az asztal között nincs diszkontinuitás, sem metszésvonal

Az ilyen szélsőséges esetek kezeléséhez nem elég a felsorolt határtípusokat figyelembe venni, hanem szükség van további számításokra is, amelyekkel már kideríthető, hogy hol van pontosan a határvonal. Az ilyen esetvizsgálatok azonban nagyon megnövelik az algoritmus futási idejét.

Két féle algoritmust implementáltam a nagy felbontású határvonalak meghatározására. Az első úgy járta be a képet, hogy elindult egy ponttól, amiről tudta, hogy biztosan határhoz tartozik, és követte a határvonalat úgy, hogy az mindig a jobb oldalon legyen a haladási irányhoz képest. Addig tette ezt, amíg vissza nem ért a kiindulási pontra. A másik algoritmusom abból a körvonalból indult ki, amelyet a síkkeresés eredményét bemutató 9. ábra szemléltet. Ezt a határvonalat próbálta úgy pontosítani, hogy rávetítette az eredeti mélységképre, majd a határpontokat mozgatta addig, míg megfelelően nem illeszkedtek valós határvonalakra. Ezen módszerek azonban felhasználhatatlannak bizonyultak végül azért, mert a kérdéses esetek vizsgálata túlságosan hosszú futási időhöz vezetett, és heurisztikus szabályok alkalmazásával a helyes körvonalak is túlegyszerűsödhetnek.

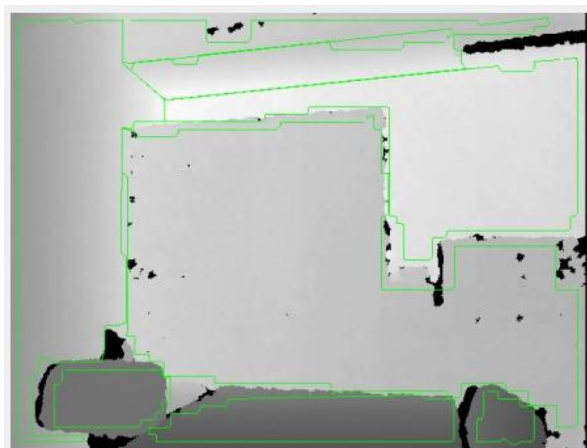
Visszaulva a már említett, a határvonalak pontos meghatározását megnehezítő körülményekre, a fő probléma kétrétű: Egyrészt a pontos körvonal túl részletgazdag a feladathoz, nem szeretnénk ugyanis minden zajszigetet körülhatárolni, ráadásul az ilyen szigetek eltüntetése az eredeti képen olyan képfeldolgozási eljárásokat igényelne, melyek futási ideje meghaladja a kitűzött határt. Másrészt pedig az iteratív kereső algoritmusokat könnyen be tudják csapni a különböző típusú határvonalak együttállásai, melyekre komplex feltételrendszereket kell kitalálni. Az ilyen feltételrendszerek hátulütői, hogy sok esetben különböző szituációk egymásnak ellentmondó szabályokat igényelnének. További hátrányai az iteratív kereső algoritmusoknak, hogy fokozottan nehéz felső korlátot adni a futási idejükre, és ha a keresés zsákutcába kerül, rengeteg számítási idő felhasználásával tudjuk csak korrigálni azt.

Az eredeti mélységképen történő határkeresés tehát olyan problémákhoz vezet, melyek megoldása túl nagy számításigényű a kitűzött célhoz képest. A síkokat kialakító algoritmusnak azonban egy olyan pozitív mellékhatása van, hogy megkapunk egy olyan adatszerkezetet, melyről egyszerűen kiderül, hogy az egyes síkokhoz mely pontok tartoznak. Ezen pontcsoportok körvonalát fogadom el, mint a síkfelületek körvonalai. Ez egy fokozottan hatékony módja a határok kialakításának, hiszen egyrészt adódik az algoritmus korábbi lépéseiből, tehát nem igényel bonyolult számításokat, másrészt pedig nem kapunk túláradó információt, vagyis nincs szükségünk arra, hogy az egyenetlen vonalakat leegyszerűsítsük, így a reprezentáció tömörsége is megoldott.

A határvonalak egyes szakaszai elvileg síkok metszésvonalain fekszenek. Mivel a sík egyenletei az előző lépések után nagyon pontosan rendelkezésünkre állnak (a pontosság mértéke mérésekkel igazolva a 3. fejezetben ismertetett), ezért a metszésvonalak analitikusan, nagy pontossággal és nagyon egyszerűen számíthatóak a síkok egyenletéből. Az eljárás tehát nem csak a szegmensekre simuló síkokat ismeri elég pontosan, hanem a körvonal azon szakaszait is, amelyek két szegmens érintkezésének felelnek meg.

Hogy ezeket a szakaszokat megtaláljuk, a körvonal összes pontjára sorban meghatározzuk, hogy milyen messze van a legközelebbi metszésvonaltól. Ha egymás után több pont is egy megadott küszöbnél közelebb helyezkedik el, akkor ezen pontsorozat első és utolsó pontját levetítjük az adott metszésvonalra, és kicseréljük a pontsorozatot erre a két pontra. A módszer bővíthető azzal, hogy tolerálja, ha néhány ponton keresztül eltávolodik a körvonal a metszésvonaltól.

Az eddig bemutatott példán a körvonalak:



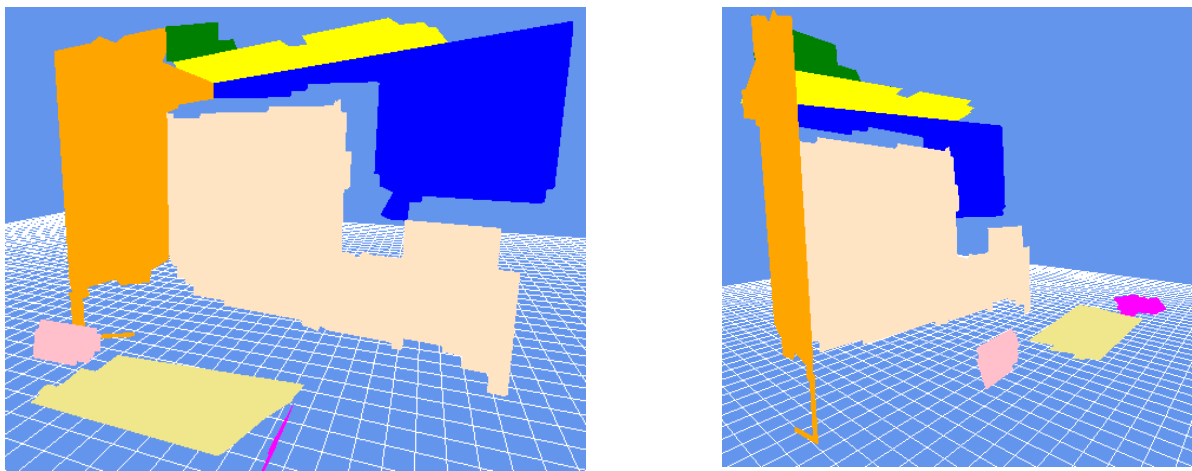
11. ábra – A kapott körvonalak a mélységképen

Azonban ez csak a képsíkon határozza meg a körvonalat, a térrekonstrukcióhoz a pontos, háromdimenziós pontokra van szükségünk. Az alacsony felbontású határvonalat az eredeti, nagyfelbontású mélységképre vetítem, és a határvonalat meghatározó pontokhoz tartozó

háromdimenziós pontok lesznek a háromdimenziós határvonal pontjai. Ha egy körvonalhoz tartozó képpontban 0 érték található (mérési zajból adódóan), akkor egyszerűen kihagyom a körvonalból.

Ezek a háromdimenziós, mélységképről származó pontok mérési zajjal vannak terhelve, így amennyiben ábrázolnánk háromdimenzióban a kapott körvonalat, az nem illeszkedne egy síkra. Hogy ezt orvosoljam, az összes pontot vetítem arra a síkra, amihez tartozik.

A vetítéssel kapott háromdimenziós pontlista a rekonstrukció végeredménye egy adott síkfelületre. Ezen felületek összessége alkotja a síkszegmensekkel rekonstruált teret. Az algoritmus lépéseinek demonstrálására használt kép rekonstrukciója:



12. ábra - A rekonstruált eredmény két különböző szögből

3. Az algoritmus pontossága

Az ismertetett algoritmus tehát alkalmas arra, hogy kialakítsa a kapott mélységkép háromdimenziós modelljét, azonban a rekonstrukció pontosságának és futási idejének meghatározásához különböző méréseket szükséges elvégezni. Ehhez definiálni kell a pontosság mérésének módját, valamint a teszteseteket.

3.1. Az elkészített szimulációs környezet

A hiba méréséhez szükség van arra, hogy a rekonstruálandó felületek olyan tulajdonságait ismerjük pontosan, mint a síkszegmens körvonala, valamint az illeszkedő sík egyenlete. Ha ezeket teljes megbízhatósággal ismerjük, akkor már könnyen meghatározható a hiba is.

Szükség volt tehát egy eszközre, mellyel automatizálva készíthetünk tesztképeket különböző beállításokkal úgy, hogy a tesztképeken szereplő síkszegmenseket meghatározó síkok egyenleteit precízen ismerjük. Ebből a célból elkészítettem egy szimulációs környezetet, mely támogatja, hogy

síkszegmensekből építhessük fel a rekonstruálandó környezetet a térben. Erről a környezetről készítettünk – zajjal terhelt – mélységképet, majd a kapott mélységképen futtatjuk a bemutatott algoritmust. A kapott eredményeket végül összehasonlítjuk az általunk definiált szegmensekkel (a következő alfejezetekben leírt módszereket felhasználva), és így megkapjuk a rekonstrukció hibáját.

A hibamérésben nyújtott segítségén kívül a szimulációs környezet szoftvere arra is használható, hogy vizualizáljuk a feldolgozott képek eredményék 3D-ben. Erre mutat példát a 12. ábra.

A környezetet C# nyelven implementáltam. Különböző feladatokra több külső könyvtárat is felhasználtam, hogy bevett technológiákat használhassak bizonyos célokra. A háromdimenziós teret a Microsoft XNA keretrendszerrel [18] jelenítem meg. A segítségével egyszerűen megoldható a 3D tér felépítése, és a térben való mozgás is.

A síkszegmenseket definiáló poligonok önmagukban nem raszterizálhatóak, hiszen csak pont, szakasz, valamint háromszög primitíveket lehet a XNA rendszeren keresztül a videokártyával raszterizálni. Hogy háromszögekre bontsam a kapott poligonokat, a Poly2tri [19] függvénykönyvtárat használtam. A háromszögek kialakítására a könyvtár kényszereket betartó Delaunay-háromszögelésen [20] alapuló megoldást használ.

Az algoritmus pontosságának meghatározásának egyik fázisában szükség van arra, hogy két kétdimenziós poligon metszését, különbségét képezzük. (Részletesen a 3.2.3 fejezetben ismertetetem a teljes módszert.) A halmazműveletek elvégzésére a Clipper [21] könyvtárat használtam.

3.2. A pontosság mérése

Mivel az algoritmus síkszegmensekkel közelíti a rekonstruálandó tér sík felületeit, ezért a pontosság fő leírója az, hogy a kapott sík egyenlete mennyire áll közel a valós sík egyenletéhez. Egy sík több módon is definiálható, azonban a mérés pontosságának leírásához érdemes azt a formát választani, hogy a síkot egy rá illeszkedő ponttal, valamint a normálvektorával jellemezzük. Ezen forma előnye, hogy minden paraméter kézzel fogható geometriai jelentéssel bír.

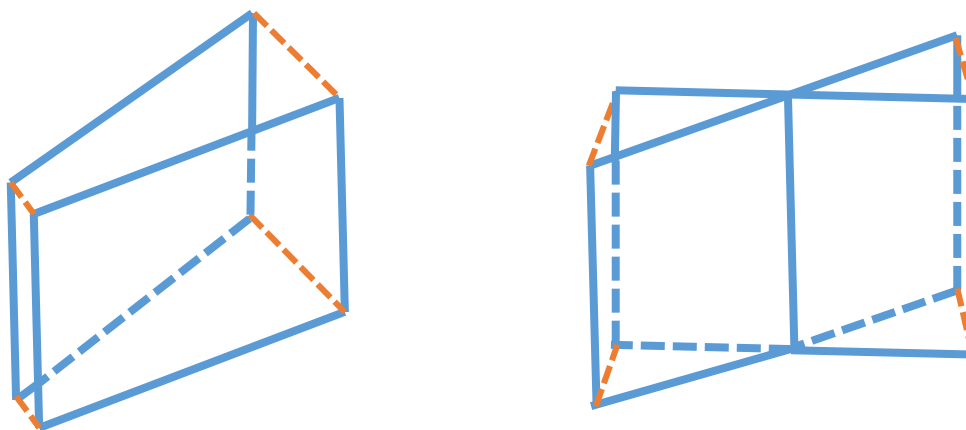
A hibát tehát úgy mérjük, hogy megnézzük a szögeltérést a valós- valamint a rekonstruált normálvektor között, valamint a két síkszegmens távolságát. A síkszegmensek további jellemzője az, hogy a rájuk simuló síkon pontosan hol helyezkednek el. A rekonstrukció akkor pontos, hogyha a simuló síkon a referencia- és a rekonstruált körvonalak által meghatározott területek teljes mértékben egybeesnek.

3.2.1. A normálvektorok összehasonlítása

A normálvektorok összehasonlítása módja természetesen adódik, a két vektor skalár szorzatából származtatható a közbezárt szög.

3.2.2. A két meghatározott síkszegmens távolságának meghatározása

Két síkszegmens távolsága általában nem értelmezhető. Az alábbi ábrán látható két, különböző szituáció:



13. ábra – Példák két nem párhuzamos síkszegmens lehetséges elhelyezkedéseire

Az ábrákon érezhető, hogy a két poligon távolsága csak akkor egyértelmű, ha azok párhuzamosak egymással. Egy pontos értelmezés az lenne, ha kiszámítanánk a két szegmens által közrefogott tér térfogatát, majd elosztanánk az egyik területével. Így valamilyen átlagos távolságot kapnánk. A megközelítés alkalmazhatóságának problémája, hogy ehhez szükség lenne arra, hogy a valós, és a kapott síkszegmens-körvonalak csúcsai egymásnak megfeleltethetők legyenek, ekkor ugyanis a két szegmens által közrefogott test két lapját a két szegmens maga alkotná, a többit pedig a megfeleltetett csúcsok összekötései között kifeszített négyszögek. Ezt mutatja a 13. ábra.

Esetünkben azonban nem megoldható ez az összerendelés, ugyanis a kapott síkszegmens körvonala nem pontosan adja vissza az eredeti körvonalat. Ez okból az előző bekezdésben leírt módszer egy közelítést használom. A közelítés lényege, hogy egy szegmens belsején egyenletesen választunk ki pontokat, majd ezeknek a pontoknak átlagoljuk a másik szegmenstől való távolságukat.

3.2.3. Az átlapolódó felület nagyságának meghatározása

Annak a meghatározásához, hogy a rekonstruálandó felületre illeszkedő síkon a rekonstruált körvonal milyen hibával határozza meg a megfelelő területet, mind a rekonstruált-, mind pedig a referenciaszegmenst a képsíkra vetítem. Ez a hiba mérésének egy fontos lépése, hiszen az algoritmus csak azt képes rekonstruálni, ami a képen szerepel. Az, hogy mennyire pontos a rekonstrukció, annak is a függvénye, hogy mennyire látszódik a felületelem a képen. A vetítés segítségével tehát ahhoz relatívan tudjuk vizsgálni az átlapolódás mértékét, hogy mennyire láthatta az algoritmus.

A hiba méréséhez meghatározzuk azt a területet, amelyet az algoritmus helytelenül nem sorol az eredmény szegmenshez, valamint azon régiók területét is, amiket úgy csoportosított a megoldáshoz, hogy a referenciaszegmensen nem szerepelnek. Ezek alapján két mértékünk van:

- a nem megtalált terület aránya a referencia területhez képest százalékosan
- a hibásan a felületelemhez sorolt terület a referencia területhez képest százalékosan

3.3. A konstruált tesztek és a kapott eredmények

Miután meghatároztuk a hiba mérésének a módját, definiálhatunk teszteseteket, amikkel specifikusan mérhetjük az algoritmus egy-egy tulajdonságát. A mérendő tulajdonságok az algoritmus pontossága

- a szegmens távolságának függvényében,
- a szegmensre látás szögének függvényében,
- a szegmens nagyságának függvényében.

Ezen kívül pedig az algoritmus futási idejének vizsgálata is szükséges, hogy megállapíthassuk, hogy alkalmazható-e valós idejű környezetben.

A mérések első lépése, hogy mi magunk definiáljuk a már ismertetett szimulációs környezetben az aktuális teszteset terét. Erre azért van szükség, mert így teljes pontossággal tudhatók a referencia értékek, amiket az algoritmusnak reprodukálni kell. Ezen kívül a mérések egyszerűbben és gyorsabban elvégezhetők ilyen módon.

A felépített tér alapján létrehozunk egy mélységképet, amit terhelünk általunk generált mérési zajjal. Ezt a mélységképet adjuk az algoritmus bemenetére. A végső lépés az, hogy kiválasztjuk az algoritmus eredményéből a vizsgált szegments, majd összehasonlítjuk a referenciával.

Megjegyzések a teszteseteket és azok hibáit bemutató ábrákkal kapcsolatban:

- Az algoritmusnak nem része, hogy a referencia szegmensek és a rekonstruált szegmensek megjelenített színeit összehangolja, ezért a rekonstruált térben található „falak” teljesen más színnel is megjelenhetnek a mellékelt ábrákon.
- A hibamértékekről mellékelt grafikonok outliereket tartalmaznak olyan esetekben, amikor az algoritmus meg sem találja a referencia síkot. Hogy a grafikonok arányosan ábrázolhatóak legyenek, ezeket az outliereket generáló szintérbeállítások hibamértékeit elhagytam. Az aktuális teszteset hibájának bemutatásakor az elhagyott ilyen beállításokra minden esetben kitérek.

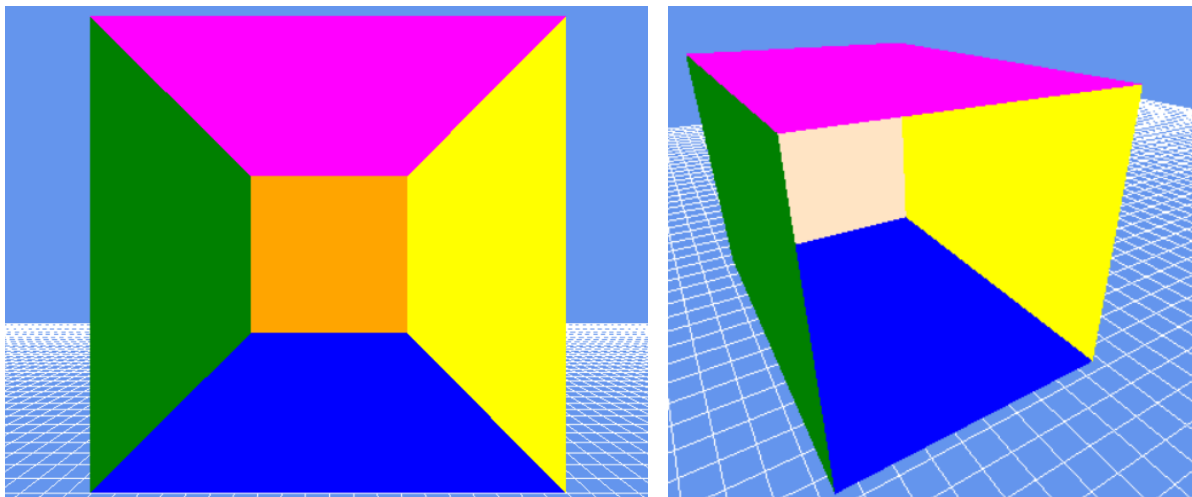
Minden mérést a leírt mérési módoknak és teszteseteknek megfelelően végeztem el, két jelentős eltéréssel. Az egyik, hogy zajjal terheltem a mélységképet annak érdekében, hogy modellezsem a Kinect mélységkamerájának zaját. A zajt távolságtól függő egyenletes eloszlással közelítettem, melynek mértéke egyezik a 7. ábra ismertetett értékekkel.

Ezen felül a mérési eredményeket nem egyetlen mérésből származtattam, hanem az algoritmus több futási eredményéből származó hibamérések kiátlagolásával kaptam. A különböző futásokhoz a felépített tér topológiáját nem változtattam meg, csupán véletlenszerű, kismértékű eltolást alkalmaztam a teljes „szobára”, valamint a képet terhelő zajt is újragenerálom. Ez alól egyetlen kivétel van, a harmadik teszt, vagyis a teljesítmény síkmérettől való függésének vizsgálata. Ebben a tesztben nem csak a teljes „szobát” toltam el a kamerához viszonyítva, hanem a változó méretű szegmenst is a szobán belül.

A hibamérésről készített grafikonok megtalálhatóak az 5.1-es függelékben.

3.3.1. Távolságfüggés

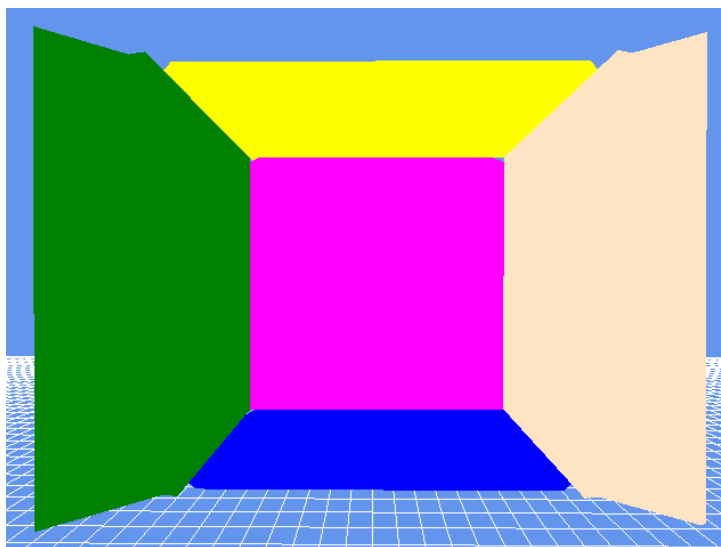
Az algoritmus távolságfüggésének vizsgálatához létrehozott környezet egy téglatest, melynek hiányzik egy lapja. A kamera ezen a lapon néz be, merőlegesen a vizsgált szegmensre. A különböző teszteseteket a vizsgált szegmens távolsága határozza meg.



14. ábra - Az első teszt egy tesztesetének felépítése két különböző szögből

Az előző alfejezetben leírtaknak megfelelően ebből létrehozzuk a mélységképet, és futtatjuk rajta az algoritmust. Az eredmény megjelenítése látható az alábbi ábrán. Az ábrán látható tesztesetekről készített mélységkép a téglatest belsejéből készült, tehát a vizsgált szegmens négy szomszédos

szegmense (a téglatest négy hosszabbik lapja) nem teljes egészében látszódik rajta, a szélek már nem jelennek meg.



15. ábra - Az algoritmus eredménye egy tesztesetre

A vizsgált távolságtartomány a 2.5m és 3.8m közötti intervallum.

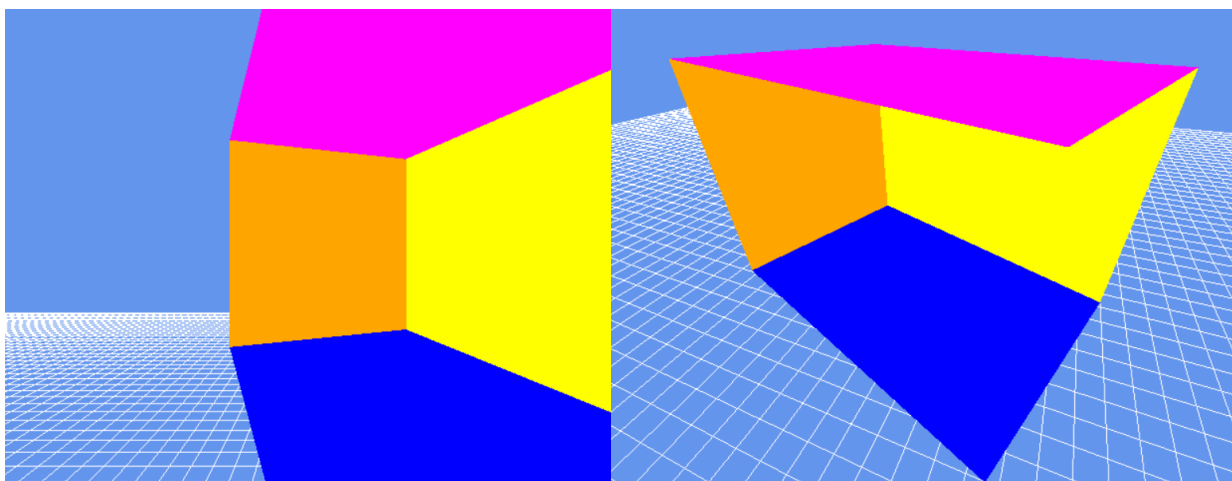
Normálvektor átlagos hibája [°]	0.011
A két sík átlagos távolsága [mm]	0.007
Átlagos helytelenül megtalált terület [%]	0.048
Átlagos a referencia szegmensben nem szereplő terület [%]	6.44

A sík egyenletében fellépő hiba enyhén nő a távolság függvényében.

3.3.2. Szögfüggés

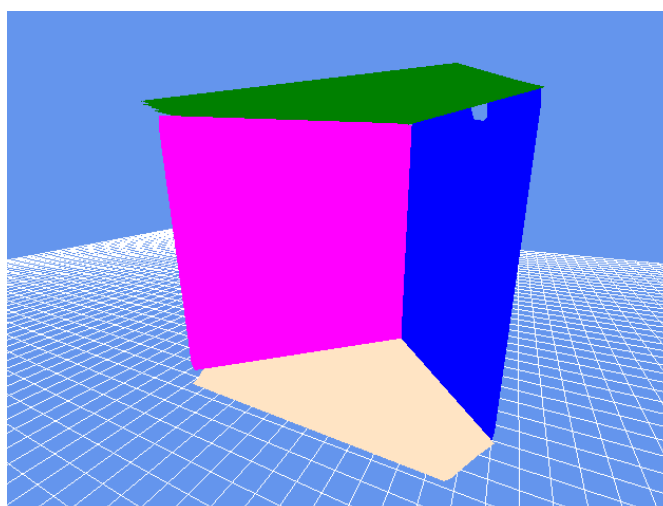
Ehhez a teszthez olyan teszteseteket kellett generálnom, melyekben a rekonstruálandó tér ugyanaz, csak az változik, hogy a kamera milyen szögben lát rá a térre, és a térrel együtt a vizsgált szegmensre.

A használt konstelláció:



16. ábra – A második tesztesethez kialakított virtuális tér

A bal oldali, narancssárga színű szegmens a vizsgált szegmens. A tér egészét forgatjuk.



17. ábra - Az algoritmus eredménye

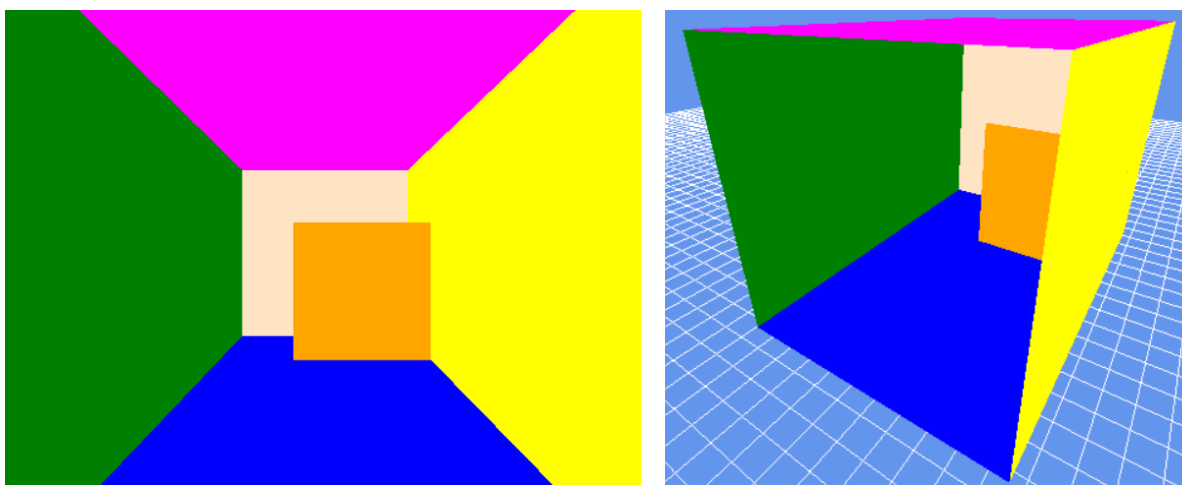
A vizsgált paraméter a referencia sík, valamint a képsík által bezárt szög. Ezt a szöget a 0°-tól 90°-ig terjedő intervallumban vizsgáltam. A 75°-osnál nagyobb beesési szög esetén az algoritmus eredménye hirtelen leromlik, a statisztikák ezért csak az ennél kisebb szögekből származtatott értéket mutatják.

Normálvektor átlagos hibája [°]	0.021
A két sík átlagos távolsága [mm]	0.013
Átlagos helytelenül megtalált terület [%]	1.18
Átlagos a referencia szegmensben nem szereplő terület [%]	3.57

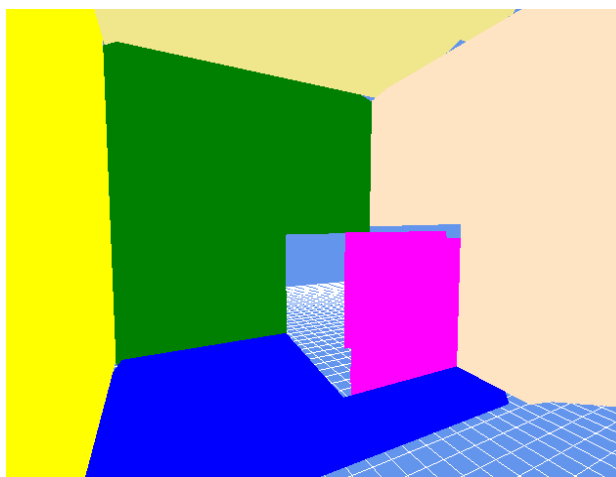
Mind a négy hibamérték nagymértékben megnő, ahogy a vizsgált sík és a képsík közbezárt szöge a derékszöghöz közelít. Ezen a kiugráson kívül a normálvektorban fellépő hiba egy ideig növekvő, majd a körülbelül 45°-os pontot elérve átvált csökkenő tendenciájúra.

3.3.3. Méretfüggés

Az ehhez a méréshez készített tesztkörnyezet nagyon hasonló a távolságfüggés méréséhez készítetthez, csupán az a különbség, hogy tartalmaz még egy szegmenst, ami a hátsó fal és a kamera között helyezkedik el a hátsó fallal párhuzamosan. Ennek a szegmensnek a mérete a konkrét tesztesettől függ.



18. ábra - A harmadik tesztesethez kialakított konstelláció



19. ábra - Az algoritmus eredménye a tesztesetre

A vizsgált paraméter a szegmens mérete állandó távolság mellett. A területe (/felülete) 0.01 m² és 3.51 m² között vett fel értékeket. Az algoritmus által még nem érzékelt szegmensek hibaadatai nem szerepelnek a statisztikában.

A legkisebb érzékelt szegmens területe [m ²]	0.16
Normálvektor átlagos hibája [°]	0.0085
A két sík átlagos távolsága [mm]	0.0029
Átlagos helytelenül megtalált terület [%]	8.13
Átlagos a referencia szegmensben nem szereplő terület [%]	8.38

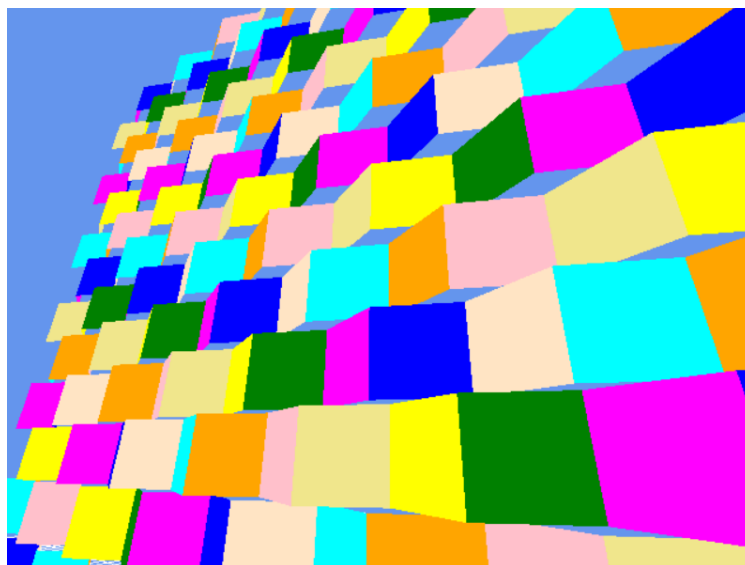
Minden hibamérték csökkenő jellegű a szegmens nagyságának függvényében.

3.3.4. A futási idő mérése

Mivel az algoritmus iteratív elemeket is tartalmaz, ezért futási idejére nem adható egyszerű felső korlát. Ebből kifolyólag olyan környezetre van szükség, amivel felmérhető, hogy a gyakorlatban mi a várható- és a maximális futási idő.

A futási idők vizsgálata egy Intel Core i7-4700HQ 2.40 GHz processzoron történt a C# System.Diagnostics.Stopwatch osztályának felhasználásával, mely a futási időt milliszekundumban, egész számra kerekítve adja meg.

A maximális idő becslésére konstruált környezet sok kicsi, de az algoritmus által már érzékelt, különböző síkokhoz tartozó szegmensekből áll.



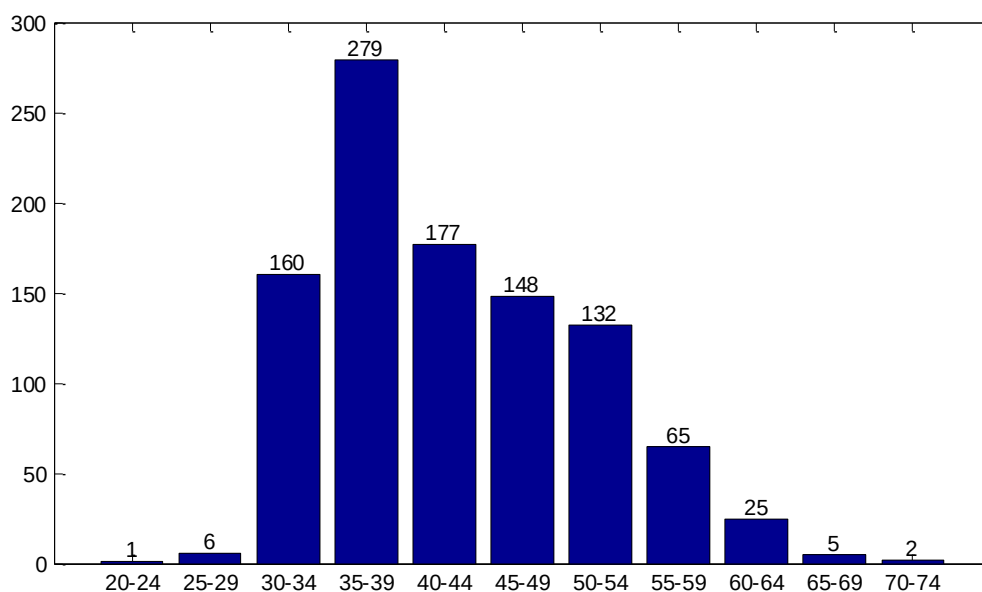
20. ábra - A futási idő mérésére kialakított környezet

Ennek a rácshoz hasonló szerkezetnek változtattam a különböző paramétereit (például a lapok száma, x és y offset) véletlenszerűen, és mértem meg a végrehajtási időket. Összesen 1000 futtatásból

származó eredmények közül kiválasztottam a 10 legnagyobb futási idővel rendelkezőt. Ezekből származó értékek:

Átlagos futási idő [ms]	66.1
Minimális futási idő [ms]	63
Maximális futási idő [ms]	73

Az összes futási időből származó hisztogram:



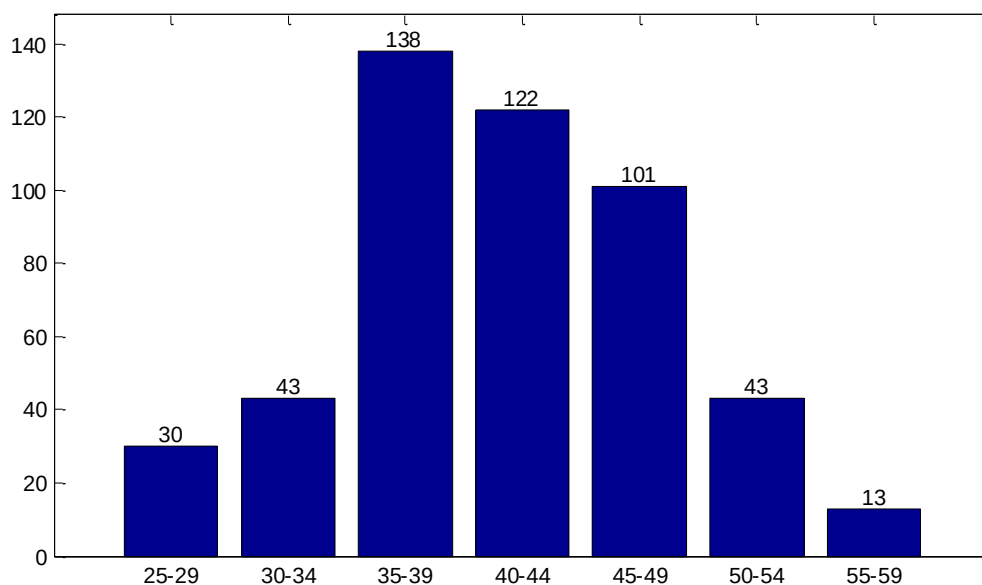
21. ábra – Az egyes tartományokba eső futási idők eloszlása (mesterséges tesztesetek)
Intervallumok [ms] – Darabszám [1]

A maximális futási idő tehát extrém esetben sem lépi át a kitűzött célt. Azonban mindenképp meg kell jegyezni, hogy ilyen magas futási időt csak olyan mesterségesen előállított környezet mélységképe alapján ért el az algoritmus, melyet direkt arra a célra terveztem, hogy a lehető legrosszabb esetet képviselje az algoritmus számára. Hasonló jellegű környezet a valós világban nem fordul elő.

A várható futási idő kiszámítására valós környezetekről készített mélységképeken mértem a sebességet. A felhasznált képek egy része a rekonstruált eredménnyel együtt a függelékben megtekinthető. A futási időkből származtatott minimális, maximális, valamint átlagos futási idő:

Átlagos futási idő [ms]	41.3
Minimális futási idő [ms]	25
Maximális futási idő [ms]	59

A futási idők hisztogramja:



22. ábra - Az egyes tartományokba eső futási idők eloszlása (valós tesztesetek)
Intervallumok [ms] – Darabszám [1]

A futási időre vonatkozó mérések tehát alátámasztották, hogy valós környezetben az ismertetett módszer valós időben is alkalmazható. Érdekes ezen felül megjegyezni, hogy az algoritmust a kutatás szakaszában C# nyelven implementáltam. Amennyiben más, alacsonyabb absztrakciós szintű nyelven implementálnánk, futási idejében jelentős csökkenés lenne várható.

4. Összefoglalás

A dolgozatomban egy olyan módszert mutattam be, amely alkalmas arra, hogy egy kisebb számítási kapacitással rendelkező robot felhasználhassa a gépi látásban, ráadásul nem támaszkodik GPU-val történő párhuzamosításra.

Ismertetésre került a kifejlesztett algoritmus, melynek lényege, hogy a rekonstruálandó teret síkszegmensekkel közelíti. Ez oly módon történik, hogy a mélységképből származtatott pontfelhőt csoportokra osztjuk úgy, hogy homogén csoportokba tartozzanak az egy síkra illeszkedő pontok. Ezekből a csoportokból kialakítjuk a felületekre illeszkedő síkok egyenleteit, majd ezeken a síkokon megkeressük a síkszegmenseket határoló körvonalakat.

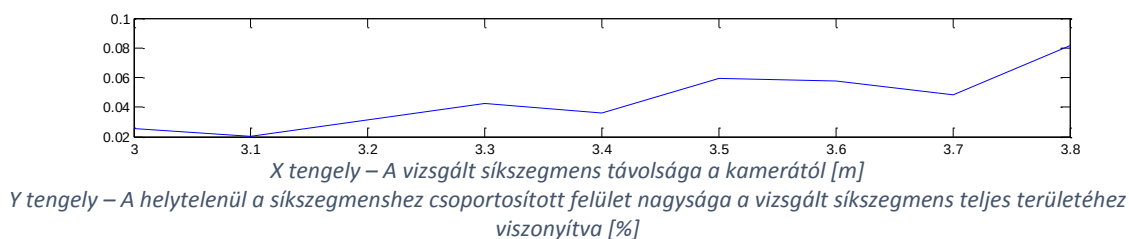
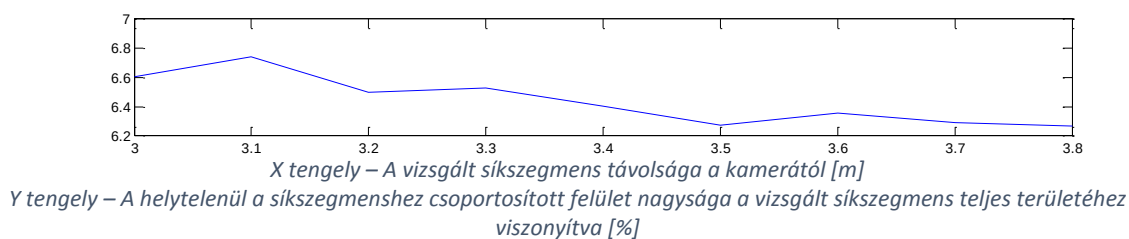
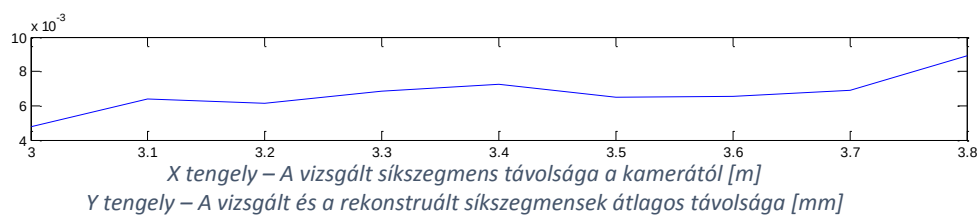
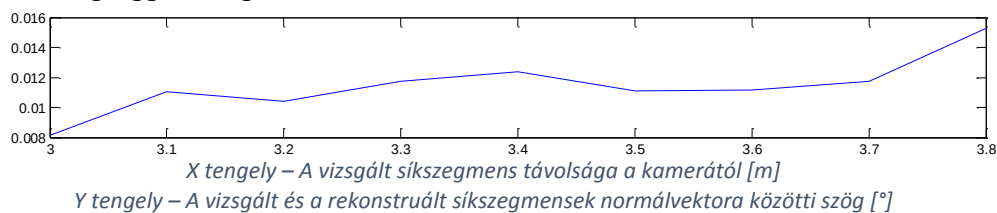
Az algoritmus felhasználhatóságát mérésekkel támasztottam alá. Két szempontból mértem a precizitást: a meghatározott síkegyenletek pontossága szerint, valamint aszerint, hogy a kialakított síkszegmensek által meghatározott felületek mennyire esnek egybe a rekonstruálandó felületekkel. A precizitás meghatározásán felül átlagos, valamint legrosszabb esetben fellépő futási időt is számítottam az algoritmusra szimuláció segítségével. A kapott eredmények alátámasztják, hogy a rekonstrukció valós időben elvégezhető. A mérési eredményekből kiderül, hogy a bemutatott módszer pontossága alkalmas arra, hogy egy mobilis robot látórendszerében felhasználható legyen. Az 1.1-es fejezetben is hivatkozott, mások által fejlesztett megoldásokhoz képest a dolgozatban bemutatott algoritmusról megállapítható, hogy futási idő szempontjából kiemelkedő úgy, hogy a meghatározott síkok egyenlete igen pontosnak nevezhető.

Az algoritmus fejlesztésének következő lépése, hogy két, egymás után következő képből meghatározzuk a kamera elmozdulását, és a síkszegmenseket egy közös térben helyezzük el úgy, hogy figyelembe vesszük a kiszámított elmozdulást. Ehhez szükség van egy módszerre, amivel össze tudjuk rendelni a különböző képekről származó síkszegmenseket. Alkalmas lenne az a megközelítés, hogy konstruálunk egy analitikusan minimalizálható energiafüggvényt, amely az előző, valamint az aktuális képből származtatott síkok egyenletei között fejezi ki az eltérést a kettő közötti transzformáció függvényében. A teljes rendszer alkalmas lesz arra, hogy nem csak egy képből, hanem egy videofelvételből rekonstruálja a felhasználó környezetét

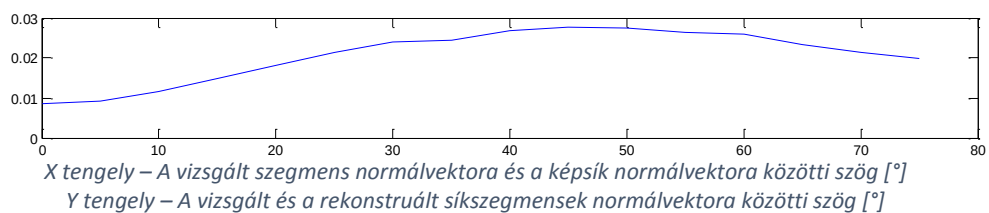
5. Függelék

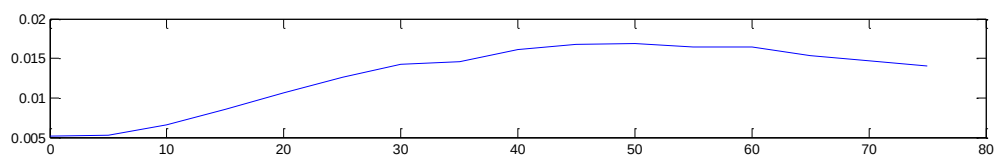
5.1. Az eljárás pontosságát leíró diagramok

5.1.1. A távolságfüggés vizsgálata

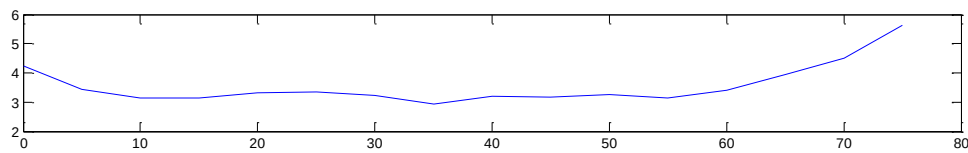


5.1.2. A síkszegmensre látás szögétől való függőség vizsgálata



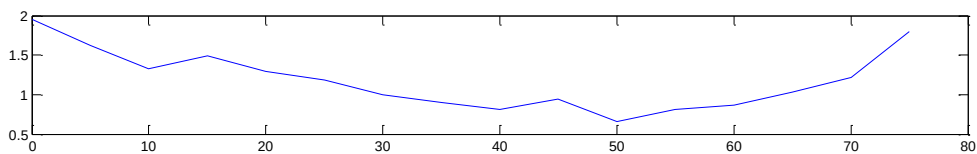


Y tengely – A vizsgált és a rekonstruált síkszegmensek átlagos távolsága [mm]



X tengely – A vizsgált szegmens normálvektora és a képsík normálvektora közötti szög [°]

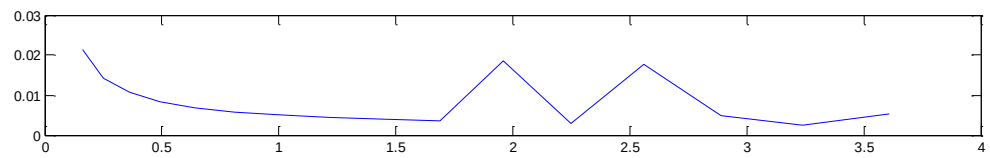
Y tengely – A helytelenül a síkszegmenshez csoportosított felület nagysága a vizsgált síkszegmens teljes területéhez viszonyítva [%]



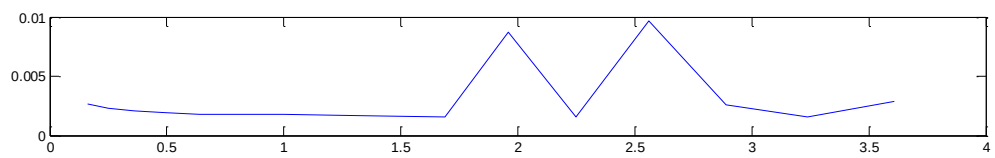
X tengely – A vizsgált szegmens normálvektora és a képsík normálvektora közötti szög [°]

Y tengely – A helytelenül a síkszegmenshez csoportosított felület nagysága a vizsgált síkszegmens teljes területéhez viszonyítva [%]

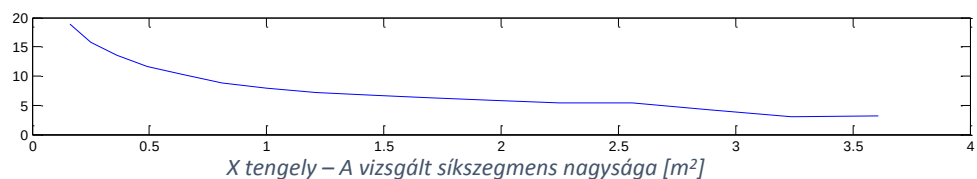
5.1.3. A vizsgált sík nagyságától való függés vizsgálata



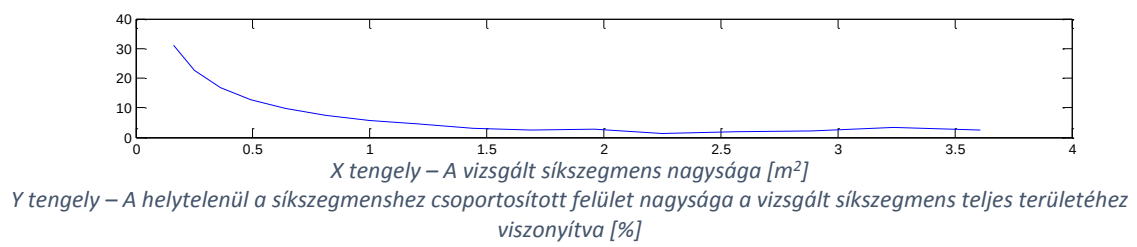
Y tengely – A vizsgált és a rekonstruált síkszegmensek normálvektora közötti szög [°]



X tengely – A vizsgált síkszegmens nagysága [m²]
Y tengely – A vizsgált és a rekonstruált síkszegmensek átlagos távolsága [mm]

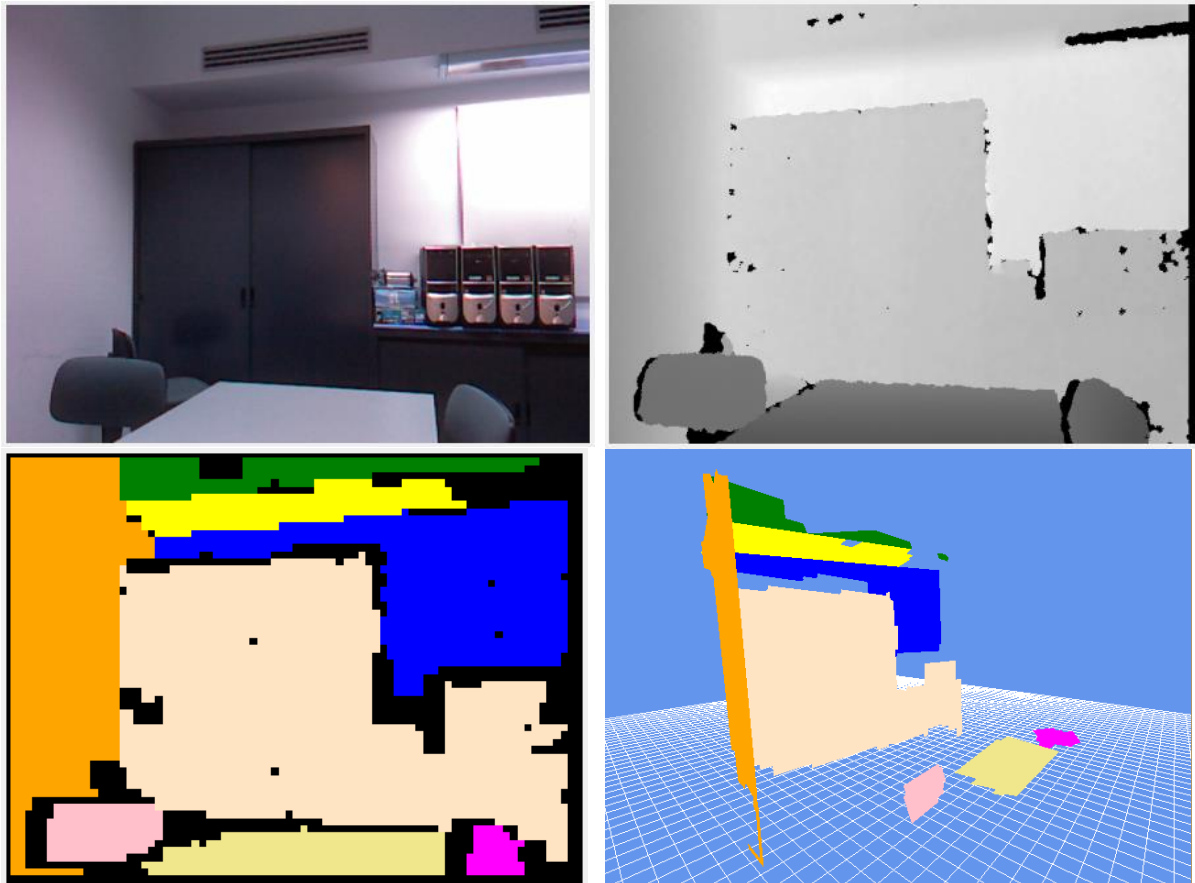


Y tengely – A helytelenül a síkszegmenshez csoportosított felület nagysága a vizsgált síkszegmens teljes területéhez viszonyítva [%]

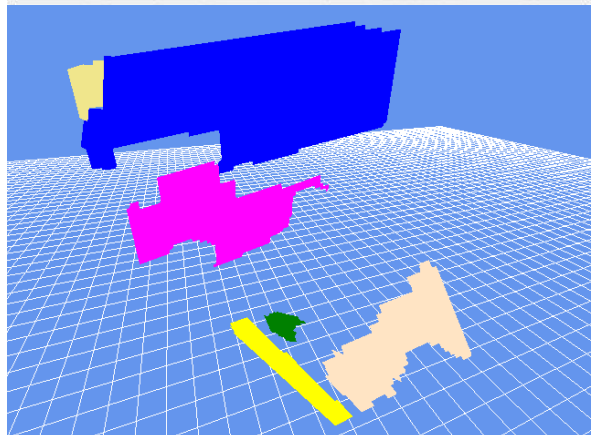
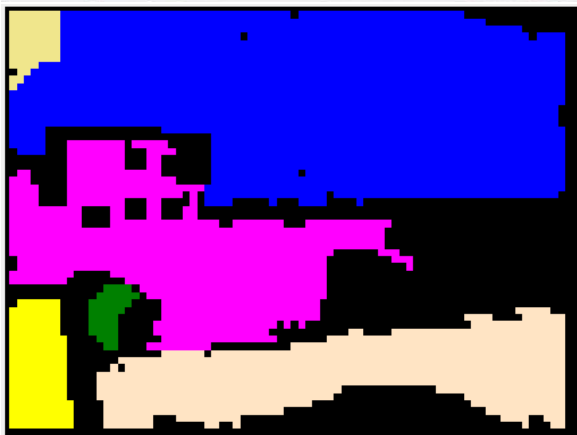
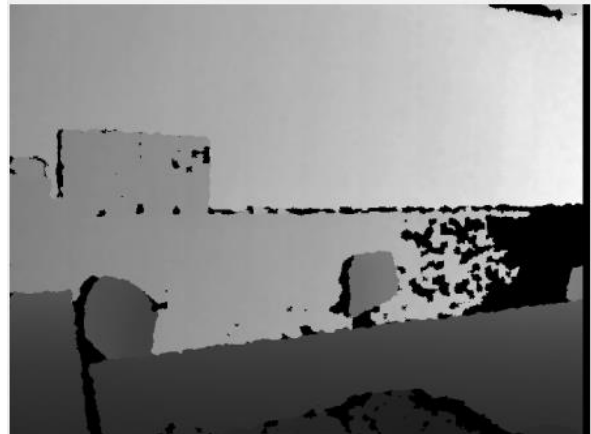
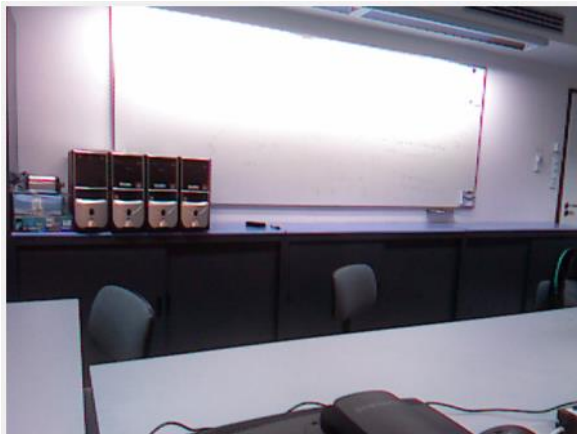


5.2. Valós példák

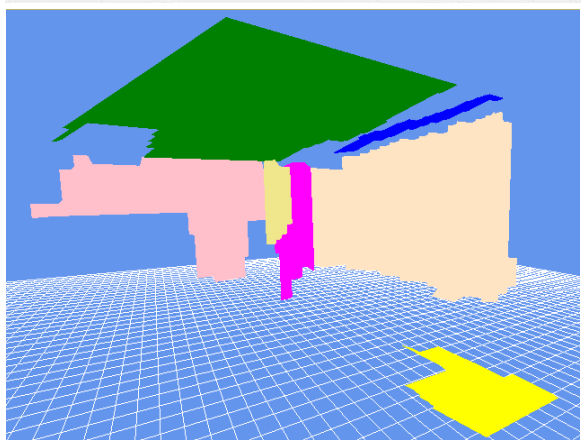
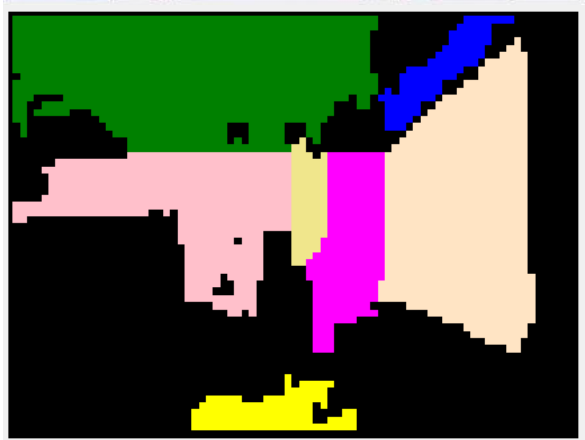
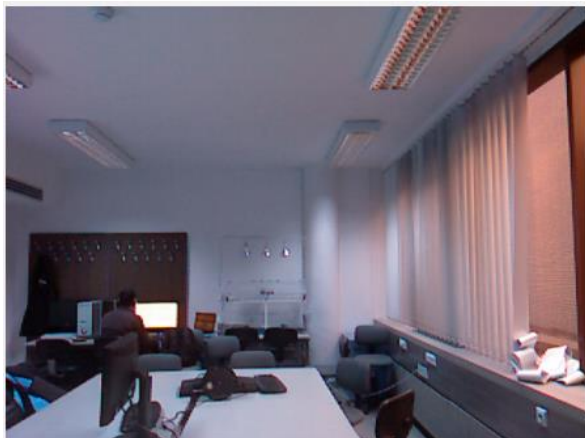
5.2.1. Első példa



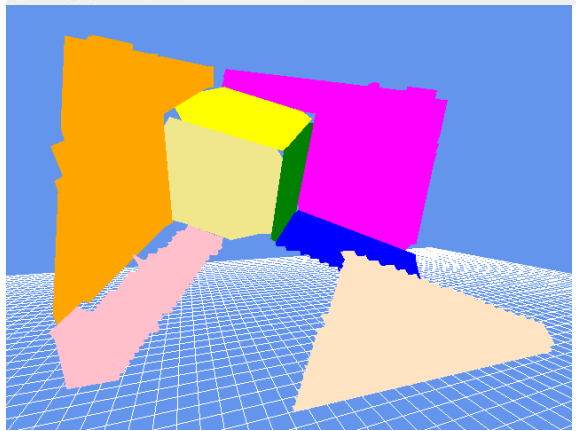
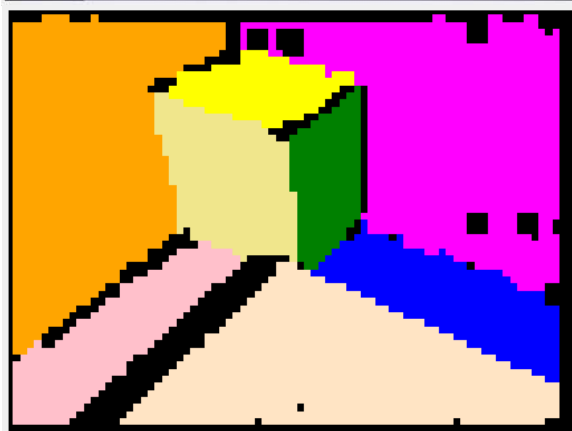
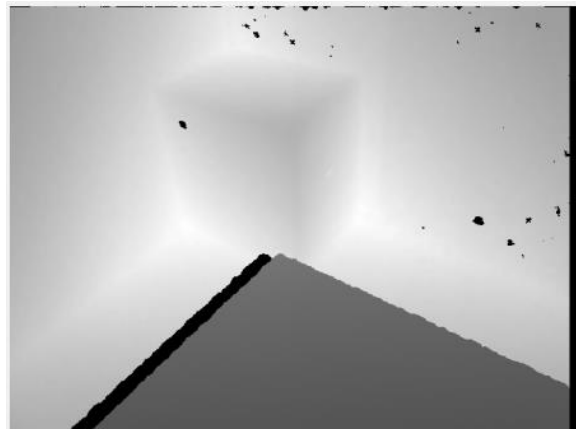
5.2.2. Második példa



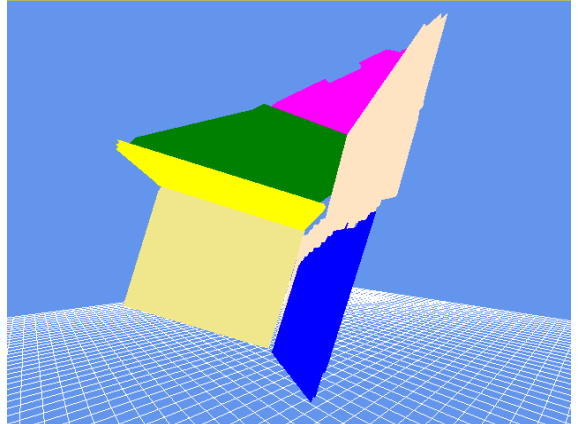
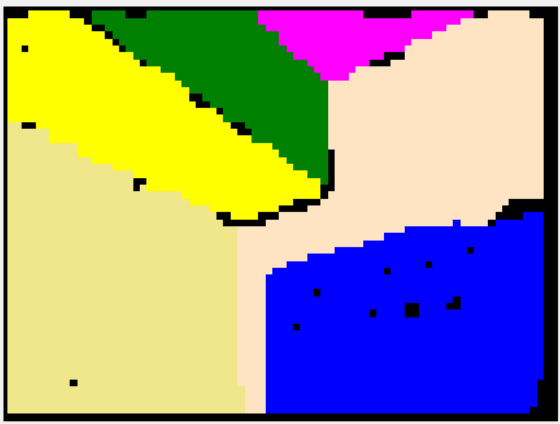
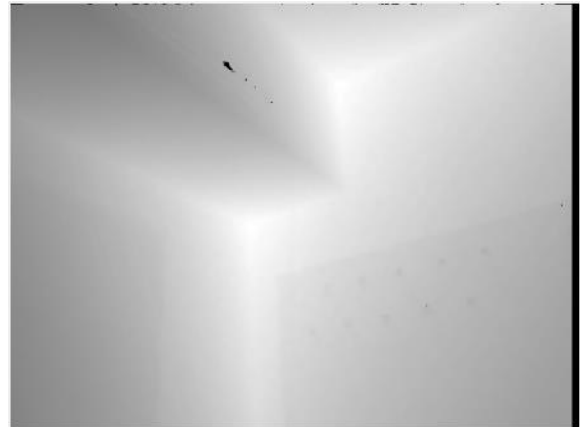
5.2.3. Harmadik példa



5.2.4. Negyedik példa



5.2.5. Ötödik példa



6. Irodalomjegyzék

1. Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., ... & Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1), 116-124.
2. Fern'ndez-Baena, A., Susín, A., & Lligadas, X. (2012, September). Biomechanical validation of upper-body and lower-body joint movements of kinect motion capture data for rehabilitation treatments. In *Intelligent Networking and Collaborative Systems (INCoS), 2012 4th International Conference on* (pp. 656-661). IEEE.
3. Ren, Z., Yuan, J., & Zhang, Z. (2011, November). Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *Proceedings of the 19th ACM international conference on Multimedia* (pp. 1093-1096). ACM.
4. Rocchini, C. M. P. P. C., Cignoni, P., Montani, C., Pingi, P., & Scopigno, R. (2001, September). A low cost 3D scanner based on structured light. In *Computer Graphics Forum* (Vol. 20, No. 3, pp. 299-308). Blackwell Publishers Ltd.
5. Xiao, J., Zhang, J., Adler, B., Zhang, H., & Zhang, J. (2013). Three-dimensional point cloud plane segmentation in both structured and unstructured environments. *Robotics and Autonomous Systems*, 61(12), 1641-1652.
6. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., ... & Fitzgibbon, A. (2011, October). KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 559-568). ACM.
7. Dumitru, R. C., Borrmann, D., & Nüchter, A. (2013). Interior Reconstruction Using the 3d Hough Transform. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(1), 65-72.
8. Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 381-395.
9. Yang, M. Y., & Förstner, W. (2010, January). Plane detection in point cloud data. In *Proceedings of the 2nd int conf on machine control guidance, Bonn* (Vol. 1, pp. 95-104).
10. Hulik, R., Beran, V., Spanel, M., Krsek, P., & Smrz, P. (2012, October). Fast and accurate plane segmentation in depth maps for indoor scenes. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (pp. 1665-1670). IEEE.
11. Okada, K., Kagami, S., Inaba, M., & Inoue, H. (2001). Plane segment finder: algorithm, implementation and applications. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on* (Vol. 2, pp. 2120-2125). IEEE.
12. Rabbani, T., & Van Den Heuvel, F. (2005). Efficient hough transform for automatic detection of cylinders in point clouds. *ISPRS WG III/3, III/4*, 3, 60-65.
13. Kovács, I. (2013.). Mért pontfelhők alapján rekonstruált, 3D-s számítógépes modellek tökéletesítése, BME-VIK TDK dolgozat

14. Cruz, L., Lucio, D., & Velho, L. (2012, August). Kinect and rgb-d images: Challenges and applications. In *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on* (pp. 36-49). IEEE.
15. Fisher, M. <http://graphics.stanford.edu/~mdfisher/Images/KinectIR.png> (2014. október)
16. Rabbani, T., van den Heuvel, F., & Vosselmann, G. (2006). Segmentation of point clouds using smoothness constraint. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5), 248-253.
17. Khoshelham, K. (2011, August). Accuracy analysis of kinect depth data. In *SPRS workshop laser scanning* (Vol. 38, No. 5, p. W12).
18. Microsoft: <http://www.microsoft.com/en-us/download/details.aspx?id=23714> (2014. október)
19. <https://code.google.com/p/poly2tri/> (2014. október)
20. Domiter, V., & Žalik, B. (2008). Sweep-line algorithm for constrained Delaunay triangulation. *International Journal of Geographical Information Science*, 22(4), 449-462.
21. <http://www.angusj.com/delphi/clipper.php> (2014. október)