



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatika Tanszék

# 3D rekonstrukció Kinect alkalmazásával

## BESZÁMOLÓ

*Hallgató*

Danyi Dávid

*Konzulens*

Kovács Viktor

2016. december 11.

# Tartalomjegyzék

<b>Introduction</b>	<b>2</b>
<b>1. Algoritmusok, paraméterek</b>	<b>3</b>
1.1. Előfeldolgozási lépések . . . . .	3
1.1.1. Difference of Gaussians . . . . .	3
1.1.2. Hisztogram kiegyenlítés . . . . .	4
1.1.3. Egyéb előfeldolgozás . . . . .	4
1.2. Diszparitás meghatározás . . . . .	5
1.2.1. Mintaillesztés . . . . .	6
1.2.2. Paraméterek . . . . .	6
1.3. Utófeldolgozás . . . . .	7
1.3.1. Medián szűrő . . . . .	8
1.3.2. Vizualizáció . . . . .	8
<b>2. Teszt platform</b>	<b>9</b>
2.1. Programozói interfész . . . . .	9
2.2. Felhasználói felület . . . . .	9
<b>3. Lokális struktúra vizsgálata</b>	<b>10</b>
3.1. Modellek . . . . .	10
3.2. Illesztési módszerek . . . . .	10
3.2.1. Kumulált korreláció . . . . .	10
3.2.2. RANSAC . . . . .	10
<b>4. Eredmények</b>	<b>11</b>
<b>5. Összegzés</b>	<b>12</b>

# Bevezetés

Ez a dokumentum a 2016 őszi félévében, Önálló laboratórium 2 tárgy keretei között végzett munkám összefoglalója.

Az itt közölt eredmények építének az előző féléves, azonos témában végzett kutatásomra. Akkor a feladat a strukturált fényt használó 3D rekonstrukciós eljárások vizsgálata volt. Az elvek gyakorlati kipróbálására a Microsoft Kinect adott kiváló platformot. Az előző féléves munka legjavát a technológiával és módszerekkel való ismerkedés adta. A Kinect által szolgáltatott infravörös kép elemzésével próbáltam reprodukálni az eszköz belső működését.

Az előző félév munkája proof-of-concept jellegű volt. A mostani ezen túlmutat. A cél most kettős: egy hosszútávon használható, rugalmas, moduláris keretrendszer fejlesztése a diszparitás meghatározásához, valamint rekonstrukció minőségének javítása a kép lokális struktúrájának figyelembe vételével.

Az első fejezetben röviden összefoglalom a használt algoritmusokat és paramétereiket. Ez részben az előző féléves munka összefoglalása is.

A második, egyben leghosszabb fejezet tartalmazza a fejlesztett keretrendszer leírását. Ismertetésre kerül a program felhasználó felülete, valami a programozási struktúra és a fejlesztői interfész is.

A harmadik fejezet a lokális struktúra figyelembevételével foglalkozik. Az itt tárgyalta algoritmusok kísérleti jellegűek, a későbbiekben behatóbb vizsgálatot és optimalizációt igényelnek.

A negyedik részegység az eredmények rövid összegzését és néhány példát tartalmaz.

## 1. fejezet

# Algoritmusok, paraméterek

A strukturált fényt használó rekonstrukciós eljárások alapja, hogy előre ismert mintát vetítenek a fényképezett objektumra, majd ennek torzulásai alapján következtetnek a mélységinformációra. A Kinect első verziója is így működik. Az eszköz mélységgépet szolgáltató része (kamera és projektor) az infravörös tartományban üzemel. A vetített minta egy lát-szólag véletlenszerű<sup>1</sup> eloszlást követő pontfelhő. A minta formális leírása vagy a generálás algoritmusá nem ismert, ezért a rekonstrukcióhoz elengedhetetlen valamilyen referenciakép készítése. A diszparitás meghatározását ez némileg bonyolítja, extra feldolgozási lépésekkel tesz szükségessé.

Az extra lépések oka, hogy jelentős időbeli különbség van a referencia- és adatkép készítése között. Ez idő alatt szinte garantáltan változnak a fényviszonyok, amit kompenzálni kell. A feladat megoldására 3 lépcsős feldolgozást valósítottam meg, amik a következőkben ismertetésre fognak kerülni.

A rekonstrukció mintaillesztésen alapuló *diszparitás meghatározás*. Az illeszkedés minőségének javítása érdekében szükség van *előfeldolgozási lépésekre*. Az diszparitáskép szűrésére és emberi fogyasztásra alkalmassá tételere pedig szükség van *utófeldolgozásra*.

### 1.1. Előfeldolgozási lépések

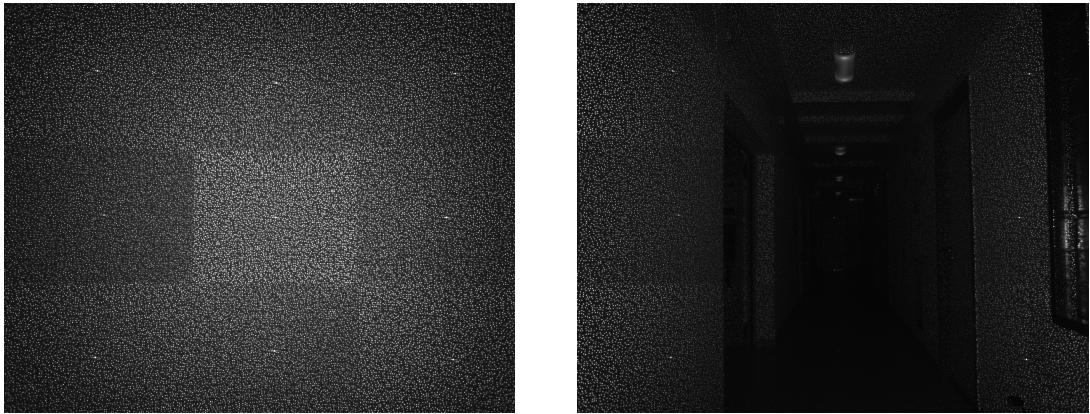
Az előfeldolgozás szükségességét a 1.1. ábra jól mutatja. Ilyen mértékű fényerőkülönbség esetén a legtöbb mintaillesztési eljárás csődöt mondana. A jelenség kiküszöbölésére több algoritmust is próbáltam a félév során, melyek változó mértékben voltak eredményesek. A következőkben ismertetem a kipróbált feldolgozási lépéseket.

#### 1.1.1. Difference of Gaussians

Az első ígéretes irány a felüláteresztő szűrés volt. Ennek egy lehetséges implementáció a DoG algoritmus. A képet két különböző Gauss szűrővel elmosztuk, majd ezeket kivonjuk

---

<sup>1</sup>Valójában úgy terveztek a pontfelhőt, hogy minimális legyen az egy sorban lévő ismétlődő vagy hasonló blokkok száma



(a) Nyers referencia kép

(b) Nyers adat kép

**1.1. ábra.** Fényviszonyok különbsége feldolgozás előtt

egymásból. Gyakran használt szűrő, főleg éldetektálásnál hasznos. Az én választásom is ezért esett rá: az információ ugyan úgy megtalálható a vettített pontok kontúrjaiban, mint magukban a pontokban. Képletszerűen (1.1) írja le a műveletet.

$$dst = gauss(src, \sigma_1) - gauss(src, \sigma_2) \quad (1.1)$$

A szűrőnek 4 lehetséges paramétere van: a két Gauss szűrő kernel mérete valamint szórása.

### 1.1.2. Hisztogram kiegyenlítés

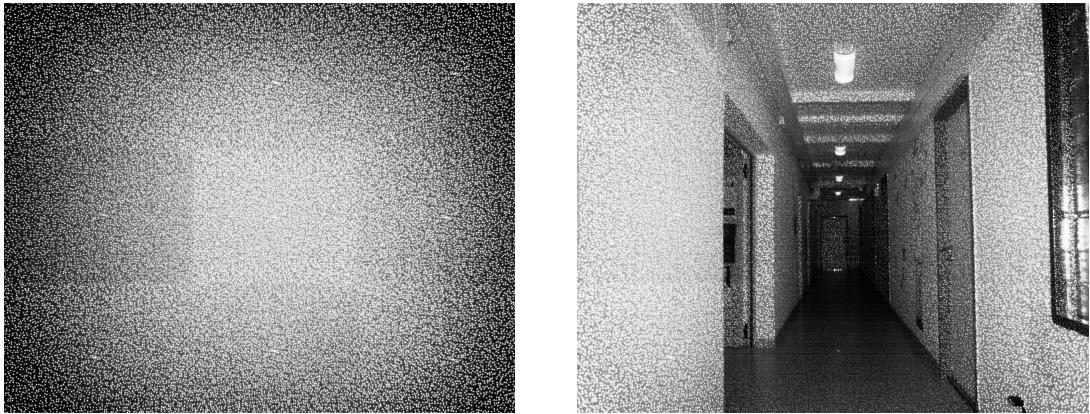
A hisztogram kiegyenlítés ötlete az, hogy a képen belül az összes fényességérték közel azonos mennyiségen legyen reprezentálva. Erre a műveletre az OpenCV szolgáltat implementált megoldást, aminek használata nagyon praktikus.

A 1.2. ábra mutatja 1.1. képeit hisztogram kiegyenlítés után. Meg kell jegyezni, hogy az eljárás nem általános megoldás: nem veszi figyelembe a kép lokális tulajdonságait. A homogén fényerőeloszlás eléréséhez régiónként kellene vizsgálni a képet és egy közös átlagos értékhez igazítani a lokális átlagot.

Egy ilyen, lokális tulajdonságokat figyelembe vevő eljárás implementálása szerepel a jövőbeli célok között. A hisztogram kiegyenlítés használatának egyedüli oka az elfogadható eredmény előállítása és egyszerűsége.

### 1.1.3. Egyéb előfeldolgozás

Az itt tárgyalt eljárások lényegileg különböznek az eddigiek től. A fenti algoritmusok a fényességkiegyenlítést szolgálják, míg az alább következők egyéb célokkal kerültek felhasználásra.



(a) feldolgozott referencia kép

(b) feldolgozott adat kép

**1.2. ábra.** Hisztogram kieggyenlítés

## Uniform skálázás

A fejlesztés során két okból került elő ez az egyszerű feldolgozási lépés. Az első indok a futásidő csökkentése volt. Gyorsabb volt a kisebb méretű képeken kipróbálni az egyes algoritmus változatokat, mint a teljes elérhető felbontáson.

A másik pedig a skálázás diszparitásképre gyakorolt hatásának vizsgálata. Az volt a megfigyelés, hogy a felére csökkentett méretű képen ( $640 \times 480$ ) simább, kevésbé zajos a kimenet. Ennek oka az volt, hogy azonos méretű minta nagyobb relatív méretet fedett le a képből, ezért több információt hordozott. Ehhez adódott még hozzá az a hatás, hogy a felbontás felezése afféle aluláteresztő szűrőként viselkedik.

## Gauss szűrés

A Gauss szűrő aluláteresztő jellegű, simítja a képet. Képfeldolgozási feladatokban gyakran alkalmazzák. A Gauss kernel elemei egy kétdimenziós normális eloszlás mintái. Jellemzően  $3 \times 3$ -as vagy  $5 \times 5$ -ös kernelméret a használatos (ez a kép felbontásától erősen függ).

Az két dimenziós eloszlás (1.2) alapján számítható.

$$G_0(x, y) = Ae^{-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}} \quad (1.2)$$

A Gauss szűrő paraméterei a kernel méret és az eloszlás szórása. Általában  $\sigma_x$  és  $\sigma_y$  meggyezik és négyzetes kernellel dolgozunk, de ettől természetesen el lehet térni.

## 1.2. Diszparitás meghatározás

Az algoritmus futásának ezen szakaszában történik a lényegi 3D információ visszanyerése. A referenciakép és az adatkép mintái relatív helyzetének a meghatározása a cél. Ezen probléma megoldására 2 módszer terjedt el: leírók használata vagy mintaillesztés.

A leírókat használó eljárások elemezik a képeket, majd nagy bizonyossággal meghatározható jellegzetes pontokat keresnek. Ezekből általában kevés van egy képen. Az ilyen algoritmusok úgynevezett ritka pontfelhős szolgáltatnak kimenetként.

A másik elterjedt út a mintaillesztés használata. Itt a képek kis régióiból (néhány pixeles környezet) mintát veszünk, és ezeket keressük a másik képen. Itt az egyezés megbízhatósága jóval kisebb, de cserébe sok adatpontot (sűrű pontfelhő) kapunk. Ezt a megoldást alkalmaztam a fejlesztés során.

A mintaillesztéses eljárás akkor alkalmazható hatékonyan, ha a képek *rektifikáltak*. Ez a feltétel a Kinect esetén az eszköz felépítésénél fogva teljesül.

### 1.2.1. Mintaillesztés

A mintaillesztési algoritmus tárgyalásánál a legfontosabb paraméter az illeszkedés jóságát leíró költségfüggvény. Ezek összehasonlítását és elemzését itt mellőzöm, erről szólt az előző feléves kutatásom. Itt csak a tapasztalatok által legjobbnak ítélt és azóta használt módszert mutatom be röviden.

A legjobb találati arányt a *normált keresztkorrelációs együtthatót* maximalizáló eljárás adta. Ennek a számítása költségesebb, mint az általában használt, hibát minimalizáló eljárások (pl. SAD<sup>2</sup>).

A költségfüggvény matematika leírása (1.3) és (1.4), ahol  $R(x, y)$  jelöli az  $(x, y)$  pontra való illeszkedés jóságát.  $T(x, y)$  a referenciakép,  $I(x, y)$  pedig az adatkép megfelelő képpontját jelöli.

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') * I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} (T'(x', y')^2 * I'(x + x', y + y')^2)}} \quad (1.3)$$

$$\begin{aligned} T'(x', y') &= T(x', y') - \frac{1}{w * h} * \sum_{x'', y''} T(x'', y'') \\ I'(x + x', y + y') &= I(x + x', y + y') - \frac{1}{w * h} * \sum_{x'', y''} I(x + x'', y + y'') \end{aligned} \quad (1.4)$$

### 1.2.2. Paraméterek

A költségfüggvényen kívül számos egyéb paraméter is befolyásolja a diszparitás sikeres meghatározását. Ezek inkább az implementációhoz kötődő, a gyakorlatban jelentős jellemzők.

---

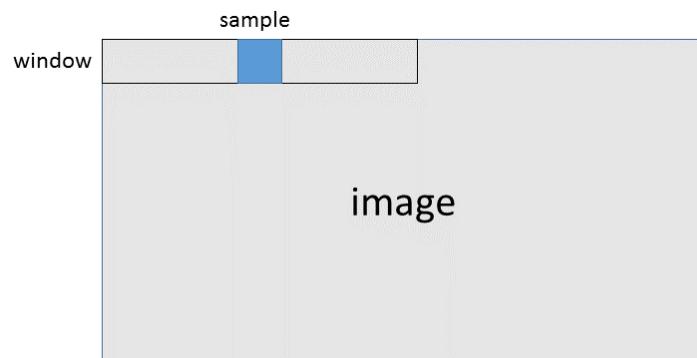
<sup>2</sup>Sum of Absolute Differences

Programozási szemszögből nézve az algoritmus a következőként néz ki. Mintát kell venni az adatképből, és megkeresni a referenciaiképen a legjobban illeszkedő részt. Ezt a kép minden részére meg kell tenni. A két képen észlelt pozíciók különbsége a diszparitás. Annyi könnyebbég adódik, hogy a képek *rektifikáltak*, azaz tudjuk, hogy csak x irányban mozult el a minta. Az illeszkedési minőséget minden eltolásra ki kell számolni, ami nagyon számításigényes, lassú művelet.

Optimalizálási céllal nem a teljes tartományon keressük az illeszkedést, hanem feltesszük, hogy egy adott határon belül marad a diszparitás. Ezt határozza meg az *ablakméret*, amit átlátszó téglalap jelöl a 1.3. ábrán.

A másik fontos, méretkellemetlenül összefüggő állandó a minta mérete. Ezt a régiót csúsztatja végre az algoritmus a képen és keresi a referencia képen. A 1.3. ábrán ez a kék négyzettel jelölt tartomány. Hasonlóan az ablakméréthez, itt is szem előtt kell tartani a teljesítményre gyakorolt hatást. A kis minta méret rontja az egyedi jellegét, hamis találatokat eredményezhet. Azonban nem is növelhető tetszőlegesen, mert akkor már túl nagy lesz az eltérés a referencia és az adat között (itt a kép tartalma okozza a hibát).

Az ablak méretnek legalább el kell érnie a minta méretét.



**1.3. ábra.** Méret paraméterek

Fontos megjegyezni, hogy ezek azok a paraméterek, amik az összes feldolgozási stratégiára jellemzők függetlenül attól, hogy figyelembe vesszük-e a lokális képi struktúrát.

### 1.3. Utófeldolgozás

Az utófeldolgozási fázisban az előállított diszparitáskép manipulációja történik. Alkalmazhatóak zajszűrő eljárások, de akár lehet szó egyszerű vizualizációról is. Érdemi 3D feldolgozás nem történik itt, csak az eredmény kisebb-nagyobb formázása.

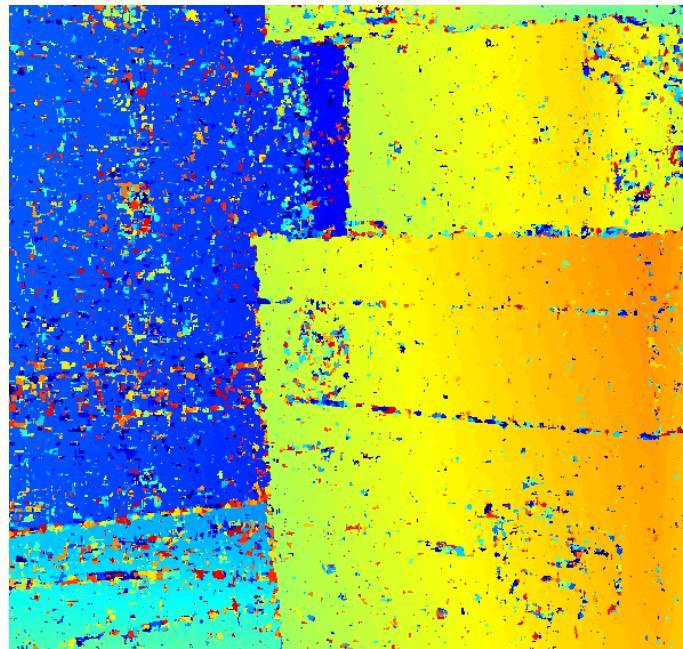
A munka nagy részében kizárolag vizualizáció történt ebben a fázisba. Folytattam kísérleteket medián szűrő alkalmazásával is. Ennek (jelen féléves projekt keretei között) nincs

gyakorlati jelentősége: még a diszparitás számítási fázisban próbáltam a zajszűrést elvégezni.

### 1.3.1. Medián szűrő

A medián szűrő nem lineáris szűrő. Az úgynevezett rang szűrők közé tartozik. Működési elve: a kernelben található képpontokat érték szerint sorba rendezi, majd a kernelközéppont értékét helyettesíti az elemek mediánjával. Kiválóan alkalmas pontszerű hibák szűrésére.

A fent említett tulajdonsága miatt merült fel a használata. A kezdeti próbálkozások kiemelő képei ilyen jellegű hibával terheltnek látszottak. Mint további vizsgálat alapján kiderült, a hibák nem pontszerűek, túl nagy a kiterjedésük ahhoz, hogy ésszerű kernelmérő mediánszűrő kezelni tudja őket. Ilyen jellegű zajjal terhelt képet mutat a 1.4. ábra.



1.4. ábra. Zajjal terhelt kimeneti kép

### 1.3.2. Vizualizáció

Ez a lépés arra szolgál, hogy könnyen átlátható formában jelenítse meg a diszparitást. Nyers formájukban a kapott értékek igen alacsonyak és monokromatikusak. Ez egy alacsony átlagos intenzitású, szürkeárnyalatos képet eredményez. Ez a feldolgozási szakasz interpolálja a kapott számértékeket RGB képre. A képen a közeli pontok vörössel, a távoliak kékkel vannak jelölve, köztük pedig folytonosan változik a szín a diszparitás nagyságával arányosan. Fontos megjegyezni, hogy ez *nem mélységgép*, csak a diszparitás megjelenítése. Ezen eljárás kimenete a 1.4. ábra is.

## 2. fejezet

# Teszt platform

A munka kutatási jellege miatt fontos volt egy rugalmas, moduláris teszt platform létrehozása. A programozáshoz a C# nyelvet választottam a magas szintű funkciói miatt: igen fontos volt a gyors prototípus fejlesztés lehetősége. A képfeldolgozási feladatokhoz az OpenCV 3 könyvtárat használtam, az EmguCV wrapperen keresztül.

Amint már többször hangsúlyozásra került, a modularitás és rugalmasság fontos szempont volt a keretrendszer tervezése során. Ezt a rugalmasságot úgy értem el, hogy feldolgozási sorokba rendeztem az elvégzendő képfeldolgozási feladatokat.

Három sor került definiálásra az előző fejezetben ismertetett struktúrát szem előtt tartva. Egy *előfeldolgozási* sor, egy *diszparitás számító* sor és egy *utófeldolgozási* sor. Köztük éles természetes határvonalként adódik a bemeneti argumentumaik mibenléte. Az előfeldolgozó sor 1 képen értelmezett műveletek hajt végre a referencia- és az adatképen. A diszparitás számító sor 2 képen operál és kimenetként egyet szolgáltat. Az utófeldolgozó pedig ismét egy képen értelmezett műveleteket támogat, amiket a diszparitásképen hajt végre.

### 2.1. Programozói interfész

### 2.2. Felhasználói felület

### **3. fejezet**

## **Lokális struktúra vizsgálata**

**3.1. Modellek**

**3.2. Illesztési módszerek**

**3.2.1. Kumulált korreláció**

**3.2.2. RANSAC**

## 4. fejezet

### Eredmények

## 5. fejezet

### Összegzés