



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 13.

Tartalomjegyzék

Introduction	3
1. Algoritmusok, paraméterek	4
1.1. Előfeldolgozási lépések	4
1.1.1. Difference of Gaussians	4
1.1.2. Hisztogram kiegyenlítés	5
1.1.3. Egyéb előfeldolgozás	5
1.2. Diszparitás meghatározás	6
1.2.1. Mintaillesztés	7
1.2.2. Paraméterek	7
1.3. Utófeldolgozás	8
1.3.1. Medián szűrő	9
1.3.2. Vizualizáció	9
2. Teszt platform	10
2.1. Programozói interfész	10
2.1.1. Adatstruktúra	11
2.1.2. Feldolgozási lépés	11
2.2. Felhasználói felület	12
2.2.1. Főképernyő	12
2.2.2. Beállítások fül	13
2.2.3. Menüsor	13

3. Lokális struktúra vizsgálata	15
3.1. Modellek	15
3.1.1. Lineáris modell	15
3.1.2. Konstans modell	16
3.2. Illesztési módszerek	16
3.2.1. Kumulált korreláció	16
3.2.2. RANSAC	17
4. Összegzés	18

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 véletlenszerű¹ 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*.

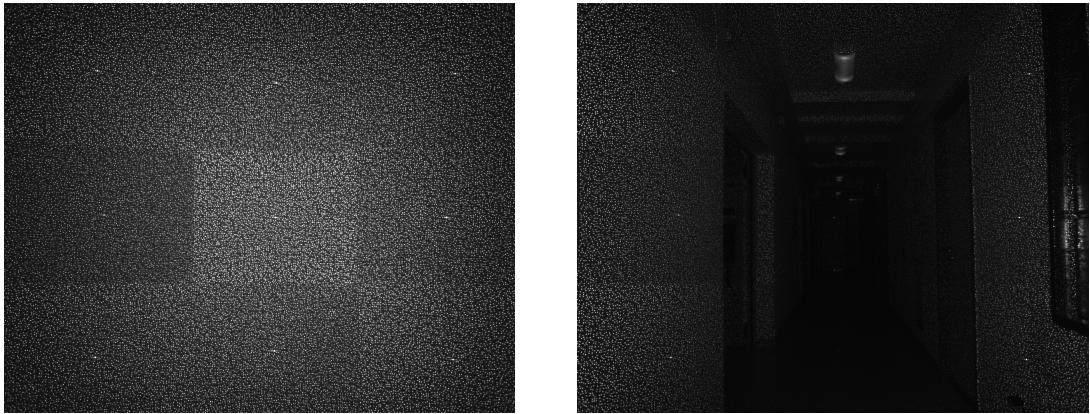
1.1. Előfeldolgozási lépések

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

¹Valójában 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

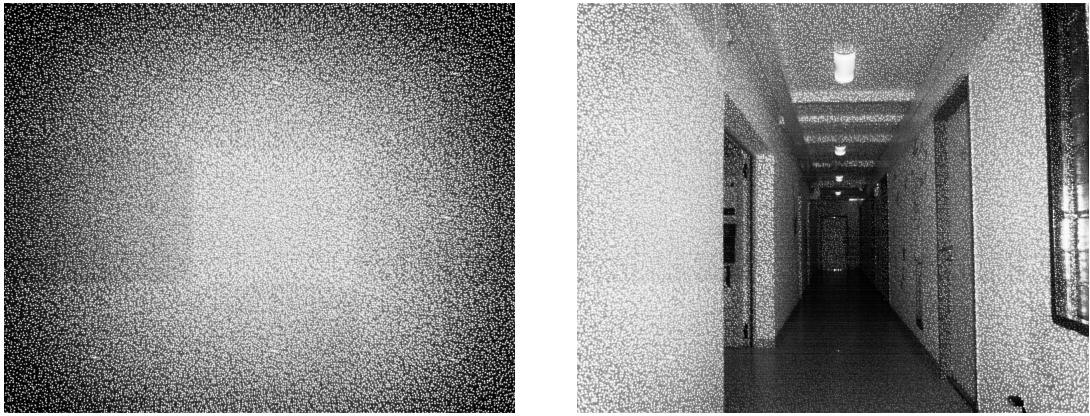
A hisztogram kiegyenlítés ötlete az, hogy a képen belül az összes fényességérték közel azonos mennyiségen legyen reprezentálva. Erre a műveletre az OpenCV szolgáltat implementált megoldást, aminek használata nagyon praktikus.

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 (640×480) 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 3×3 -as vagy 5×5 -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 σ_x és σ_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. SAD²).

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.

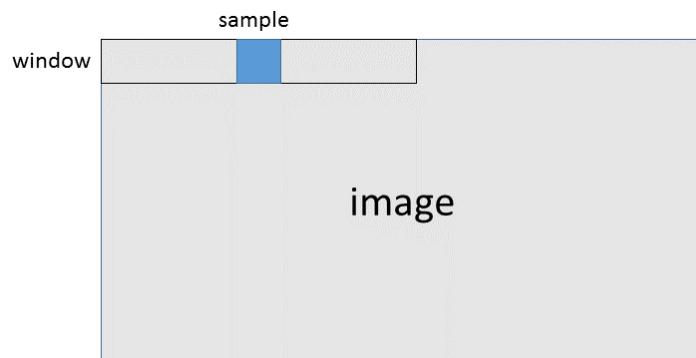
²Sum of Absolute Differences

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.

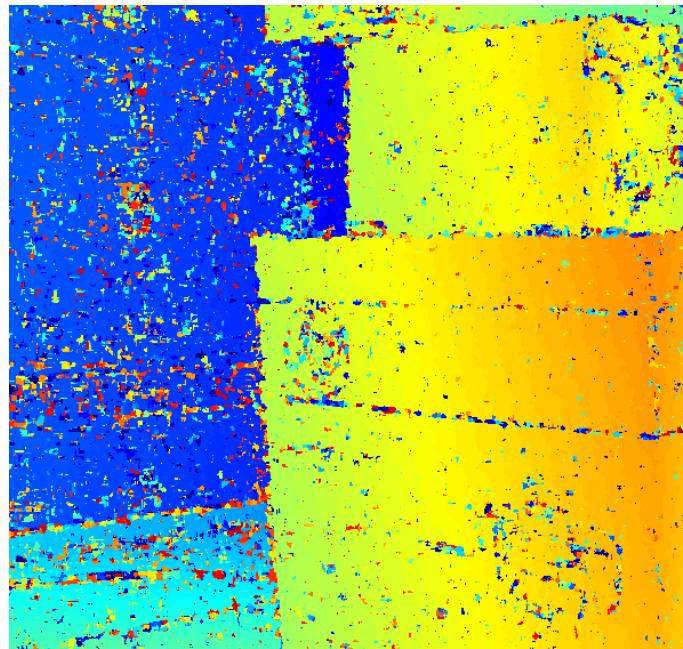
A munka nagy részében kizárolag vizualizáció történt ebben a fázisba. Folytattam kísérleteket medián szűrő alkalmazásával is. Ennek (jelen féléves projekt keretei között) nincs

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 argumentumai 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.

Ahol lehetett és szükséges volt, több szálra futni képes kódot írtam. Ez ugyan extra erőfeszítés, de mivel gyakori volt az új algoritmusok tesztelése, ritka volt a hatékony implementáció. A kézben tartható futásidőkhöz elengedhetetlen volt a párhuzamosítás.

Kihívást jelentett egy használható, intuitív grafikus felület tervezése. Ez részben a rutin hiányának, részben pedig a sok, komplex adat átlátható megjelenítésének igényéből adódott.

2.1. Programozói interfész

Az előző féléves munka során tapasztaltam, mekkora nehézséget tud okozni egy nem jól skálázódó, átgondolatlan program struktúra. Az akkor elkövetett hibákból tanulva igyekeztem most egy egyszerűen bővíthető, jövőbeni ötletekre is felkészített keretet tervezni.

Első és legfontosabb teendőm volt a felhasználói felület és a feldolgozás szeparálása. Ezzel is készülve arra, hogy esetleg a teljes back-end lecserélhető legyen, például egy C++-ban fejlesztett könyvtárra.

2.1.1. Adatstruktúra

A munka jellegéből adódóan szükségét éreztem, hogy a kimenet a feldolgozási lánc bár-mely szakaszában megjeleníthető legyen. Ezért a minimálisan elvárhatótól több képet tárol a program a memóriában. Ennek az emberi vizsgálatokon kívül a feldolgozás során is tapasztalható előnye.

A programban a következő szakaszait tárolja a feldolgozásnak:

- Nyers adat
- Nyers referencia
- Előfeldolgozott adat
- Előfeldolgozott referencia
- Nyers diszparitás
- Utófeldolgozott diszparitás
- Vizualizált diszparitás

Az utófeldolgozott és a vizualizált diszparitás szeparált tárolását az indokolja, hogy a vizualizáció során meg változik az adat jellege: 3 csatornás RGB képet kapunk az egy csatornán tárolt értékek helyett.

Ezek az adatok nem függetlenek egymástól, bizonyos sorrendiségekkel kell állnia köztük. Adódik az igény, hogy a felhasználói felületen bizonyos funkciók csak akkor legyenek elérhetőek, ha a feldolgozás egy része már befejeződött. Ennek támogatása céljából lekérhető a fenti pufferek állapota, feltétel állítható be rájuk.

2.1.2. Feldolgozási lépés

Programozás során minden megvalósított algoritmus egy úgynevezett processingStep absztrakt osztály leszármazottja. Ezeket a lépésekkel hatja végre minden feldolgozási sor.

A mindenki szükséges és vélhetően közös adatokat valamint metódusokat biztosítja az osztály. Ilyen adatok adat- és referenciapuffer, valamint a kimeneti kép. Ugyan ezek nem mindegyike használt az összes lépés által, de az általánosság megtartása miatt bekerültek az ősosztályba.

A feldolgozási sorok minden lépés doYourJob() metódusának hívásával indítják el a lépéseket. Ezt mindenki meg kell valósítani a fejlesztés során.

Ez a struktúra biztosítja a rugalmasságot és egyszerű bővíthetőséget a további munka során.

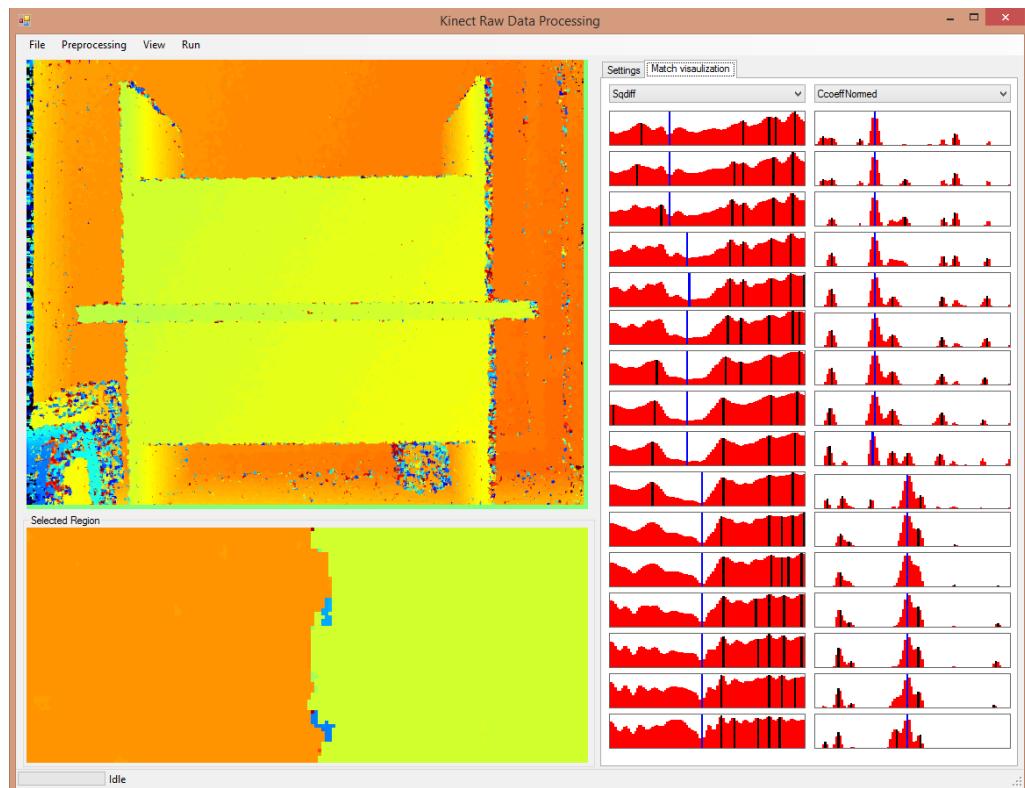
2.2. Felhasználói felület

A felhasználói felület készítésére a Microsoft Visual Studio által nyújtott winforms környezetet használtam. Ugyan léteznek ettől korszerűbb és nagyobb teljesítményű könyvtárak is, egyszerűsége és a fejlesztői környezetbe való nagyfokú integráltsága miatt kézenfekvő választás volt.

A program felülete jelenlegi formájában közel sem optimális vagy végleges. A sok képi információ átlátható megjelenítése problémás, emellett a relatív szerteágazó beállítási lehetőségek ábrázolása is szükséges.

2.2.1. Főképernyő

Az alkalmazás felületét a 2.1. ábra mutatja. A bal felső régió szolgál a kiválasztott puffer (nyers-, előfeldolgozott adat, diszparitáskép...) megjelenítésére. Hasznos funkció lehetne a későbbiekbe, ha a program több puffer egyidejű vizsgálatát is lehetővé tenné, de ez egyelőre a képernyő véges területe miatt nem készült el.



2.1. ábra. Adatok megjelenítése

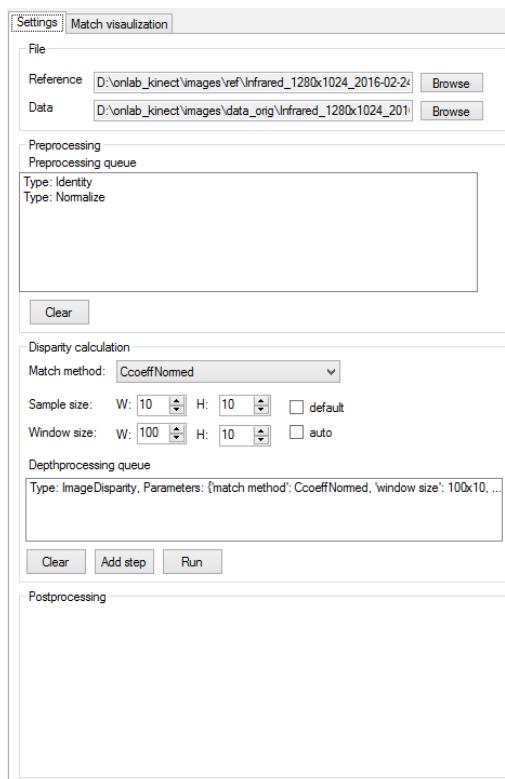
A bal alsó sarokban látható képrégió az aktuális puffer egy részének nagyított képét mutatja. Ezt a funkciót az motiválta, hogy könnyebb legyen az illeszkedés vizsgálata az élek és pontszerű hibák körül.

Az előző féléves munka során, amikor a költségfüggvények hatékonyságát vizsgáltam, elengedhetetlen volt az illeszkedés minőségének valamelyen vizualizációja. Erre szolgál a 2.1.

ábra jobb oldalán látható sáv. Az aktuális megjelenített képen történő kattintással végrehajtódik a mintaillesztés a kattintott hely környezetében, és a jobb oldali sávon látható grafikonokra rajzolódik az illeszkedés jósága. A két oszlopban két külön költséggfüggvény választható az illesztéshez. Az alsó, nagyított régió is a kattintás helyét veszi középpontjául.

2.2.2. Beállítások fül

A beállítások fül alatt konfigurálható az egyes feldolgozási sorok tartalma. Ahogy 2.2. ábrán látható, a három sor külön állítható össze és hajtható végre. Az implementáció jelenlegi formájában nem teljes, a fontosabb részekre fókuszáltam a félév során.



2.2. ábra. Adatok megjelenítése

ban amúgy sem lenne paraméterezhető.

Némi redundanciát adva a rendszerhez, a beállítások fül alatt is betölthető az adat- és referenciakép¹.

2.2.3. Menüsor

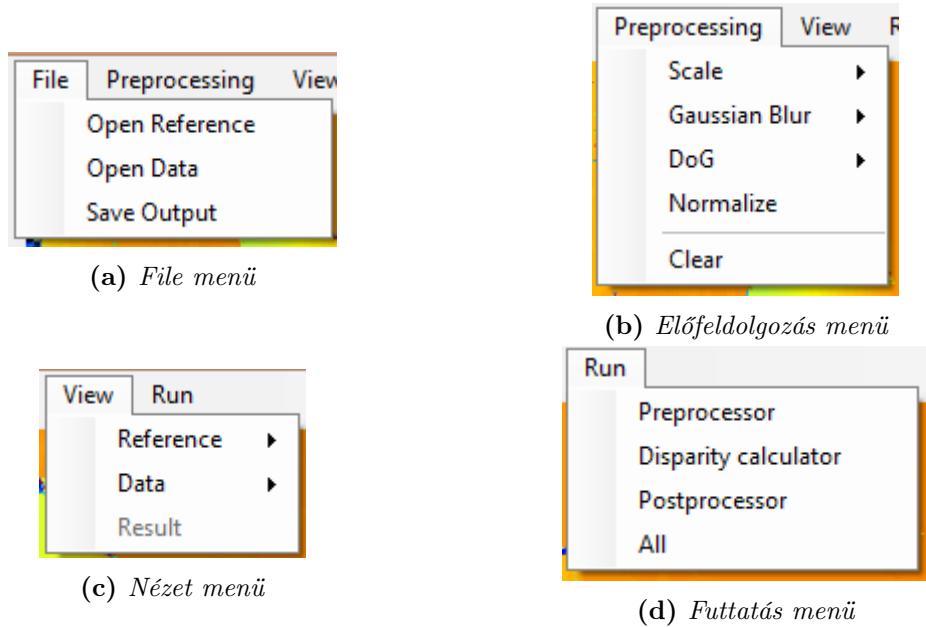
A menüsör elemeit a 2.3. ábra mutatja. A file menü tagjait nem részletezem, céljuk egyértelmű.

¹Még a file menüből érhető el ez a funkció

Az felület jelentős átszervezésen megy keresztül az előző féléves verzióhoz képest. Akkor a beállítások a felső menüsoron keresztül történtek. Ennek nyomai még nem tüntek el teljesen. Az előfeldolgozási lépések hozzáadása még mindig onnan érhető el.

A diszparitás számítás paraméterezése ezen fül alatt található. Megadható az előző fejezetben tárgyalt ablak- és mintaméret, valamint választható az illeszkedés mérőszáma. Jelen állapotában nagy hiányossága ennek a résznek, hogy nem választható meg a feldolgozási stratégia. Elkezdődött ugyan az implementálása a lokális struktúrát figyelembe vevő stratégiáknak, de ezek a felhasználói felületre még nem kerültek ki.

Az utófeldolgozási sor összeállítása sem került implementálásra a felületen. Ennek oka, hogy voltak jóval érdekesebb és magasabb prioritást élvező feladatok. Ebben a sorban jelenleg kizárolag a vizualizáció történik, ami jelen formájá-



2.3. ábra. Menüsáv elemei

Az előfeldolgozás menü alatt az előző fejezetben tárgyalt lépések adhatók hozzá a feldolgozási sorokhoz. Almenüükben a már tárgyalt paramétereik konfigurálhatóak.

A nézet menüben kiválasztható a megjeleníteni kívánt puffer. Az almenük a feldolgozás szakaszait tartalmazzák: referencia- és adatkép esetén az nyers vagy előfeldolgozott, diszparitás esetén pedig a nyers vagy vizualizált állapot kérhető le.

A futtatás menüben indíthatók a feldolgozási sorok. Lehetőség van egyesével indítani őket, vagy futtatható az összes is egyszerre. Amennyiben egyesével indítjuk őket, a program nem engedi futtatni azokat a sorokat, amelyek bemeneti adata még nem áll rendelkezésre.

3. fejezet

Lokális struktúra vizsgálata

Az illeszkedési minőség bizonyos esetekben javítható, ha figyelembe vesszük a kép lokális struktúráját. Ez a megoldás rontja az általanosságot, mert extra megkötéseket követel meg a diszparitásra vonatkozóan.

Az alábbiakban az ilyen feltételeket és a rájuk illesztett modelleket ismertetem. Nem minden modell került megvalósításra, néhol a vizsgálati eredményekből kiderült, nem reális vagy fölösleges részletes.

3.1. Modellek

Ezen fejezet alapfeltevése, hogy a diszparitás kis régiókon belül nem változik tetszőleges. Kis régió fogalom nagyon szubjektív, jelen állás szerint kísérletileg meghatározott fogalom. A továbbiak a kép 16 pixel széles tartományát értem ezalatt.

Már korábban is feltettük¹, hogy nem számítunk tetszőlegesen nagy elmozdulásokra. A modellezés során ennél erősebb feltételek keresése volt a cél.

Két esetet vizsgáltam a féléves munka során: lineáris modell és 2 konstans szakaszból álló modell.

3.1.1. Lineáris modell

A lineáris modell esetében az volt a feltételezés, hogy diszparitás lineárisan változik rövid szakaszon. Ekkor a modell (3.1) alakú.

$$y = a * x + b \quad (3.1)$$

Az illeszkedés hibája nyilvánvalóan igen nagy lesz, ha a diszparitásban ugrás² van a régió belül. Kiegészíthető a modell úgy, hogy 2 egyenes szakaszt illesztünk az adatokra. Így jóval

¹Az ablakméret megválasztásánál

²Ez a helyzet áll fenn például tárgyak pereménél

több paraméter³ meghatározása a feladat ugyan annyi bemenő adatpont használatával. Ha ezt a problémát a régióméret növelésével próbáljuk csökkenteni, akkor pedig a modell alapfeltevése (lineáris változás) sérülhet.

Mélyebb vizsgálatnak nem vetettem alá ezt a modellt. A jelenlegi pontossági igényeken belül még a lineáris változás is elhanyagolható kis szakaszokon. Több mintát megvizsgálva arra jutottam, hogy a konstans modell is elegendő a feladat megoldására.

3.1.2. Konstans modell

Hasonlóan a lineáris modellhez, ebben az esetben is be meg kell engedni egy töréspontot modellben. Így egy három paraméteres modell adódik:

- Diszparitás a régió elején
- Diszparitás a régió végén
- Törés koordinátája

Még így is relatív kevés adat jut egy paramétere (16 pixeles régióméret esetén). A gyakorlati implementációban további optimalizálásként 2 esetet különböztetek meg. Ha a két diszparitás paraméter eltérése igen kicsi (1-2 pixel), akkor 1 paraméteres modellt használunk.

3.2. Illesztési módszerek

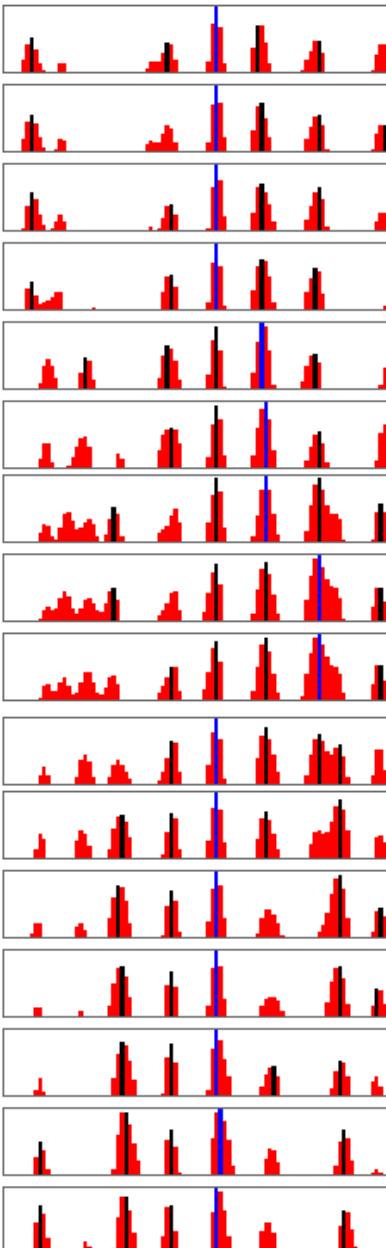
Az előző féléves munkám során a rekonstrukció eredményeként mindenkorábban a legjobb illeszkedési mutatóval rendelkező pont lett elfogadva. A mostani módszerek ettől gyökeresen különböznek. Az első maximumok gyakran hibás egyezés következményei, zajosak lehetnek. Az vizualizált illeszkedési gráfokat vizsgálva szembetűnik, hogy sok esetben az első maximum ugyan hibás, de a helyes eredmény is detektálható lenne az adatokból. Itt jön a képbe ezen fejezet tárgya: a lokális környezet. Ha nem csak az első maximumot tároljuk, hanem az első néhányat, a szomszédos pontok diszparitása alapján kiválaszthatjuk a legalioszínűbb egyezést közülük.

Ezt a választást segítik az feljebb vázolt modellek. Alább pedig két lehetőséget mutatunk be paraméterek meghatározására.

3.2.1. Kumulált korreláció

Az diszparitás meghatározásánál az illeszkedési függvény maximumhelyei közül kell választanunk. Általában az legnagyobb értékű maximumhely adja a helyes eredményt, ám vannak kivételek. A 3.1. ábra egy lokális régió pixeleire kiszámolt illeszkedési függvényeket mutatja. Kék sáv jelöli az első maximumot, fekete sávok a többi lokális maximumot.

³Az egyenesek paraméterein túl a töréspont helyét is meg kell határozni.



3.1. ábra. Illeszkedési függvények

Az ábrázolt régió egy sík felületet ábrázol, tehát konsztans diszparitást kellene látnunk. Az ábrán megfigyelhető, hogy ha nem az első maximumot választottuk volna, a helyes eredmény is megkapható.

A megoldandó probléma: választanunk kell a maximumok között. Erre egy lehetséges megoldás a kumulált korreláció vizsgálata. Lényege, hogy az egyes illeszkedési függvényeket összeadjuk, és az összegen is elvégezzük a maximumkeresést. Ekkor kapunk néhány diszparitás értéket, ahol több adatpont is elfogadható illeszkedést mutat. Ezeket az értékeket felhasználhatjuk, mint klaszterek. A két legmagasabb kumulált korrelációval rendelkező klaszterközéppontot fogadjuk el, mint modell paramétereik.

Hátra van még a töréspont meghatározása. Ehhez minden illeszkedési függvényen ki kell számolnunk a két választott klaszterközépponthoz legjobban illeszkedő maximumok hibáit. Így kapunk 2 hiba függvényt. Az egyik az él előtti szakaszon, a másik az él utáni szakaszon fog kis hibát mutatni. Optimális esetben található egy pont, ahol az egyik függvény egy adott határ alá csökken, míg a másik fölre. Ez lesz a modell 3. paramétere, az él pozíciója.

A választott diszparitás pedig mindenkor minden modell paraméterhez legközelebbi maximum: *nem* helyettesítjük a klaszterközépponttal a mért értéket.

Ennek a módszernek megvannak a korlátai: nem garantált, hogy az összegzett illeszkedés maximumai a helyes eredményt adják. Ekkor egy hamis átlagérték közé csoportosulhatnak az eredmények, aminek következtében akár még romolhat is a kapott diszparitáskép minősége.

3.2.2. RANSAC

A RANSAC algoritmus működésének leírását itt mellőzöm, működése közismert. Munkám során az előző pont végén leírt jelenség elleni védekezésként merült fel a használata. A véletlenszerűen választott paraméterek itt a klaszterközéppontok, amik a talált maximumok közül kerülnek ki. minden iterációban elvégezzük a modell hibájának számítását, majd a legjobban illeszkedőt fogadjuk el megoldásnak.

4. fejezet

Összegzés