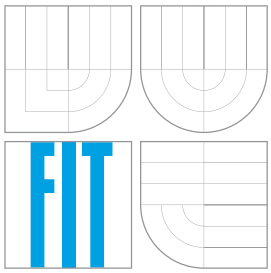
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REKONSTRUKCE 3D SCÉNY Z OBRAZOVÝCH DAT

3D SCENE RECONSTRUCTION FROM IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ FUCHSÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL ŠPANĚL, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato bakalářská práce se zabývá rekonstrukcí scény pomocí RGB–D senzoru Microsoft Kinect. Cílem práce je z nasnímaných dat vytvořit model místnosti ve formě bodů, ze kterého je následně vytvořen půdorys vnitřních prostor, tvořený čárami. Z velké části se práce zabývá registrací jednotlivých snímků (mračen bodů), tedy existujícími metodami a jejich popisem. Následně projekcí bodů a detekcí hran v obraze pomocí Houhgovy transfomace. Dále se práce experimentálně zabývá vlivem nahrávání na výsledky a také zda závisí na nahrávaném prostředí.

Abstract

This bachelor thesis deals with scene reconstruction from images using RGB–D senzo Microsoft Kinect. The aim of the work is to create model of the room in form of points from scanned data, from witch is created plan of the interior. A large part of the work deals with the point cloud registration, thus existing methods and their descriptions. Further, a projection of points on a plane and detection of edges in the obtained image are discussed. The work experimentally examines the influence of recording and the recorded environment on results.

Klíčová slova

Kinect, registrace, mračno bodů, ICP, NDT, LUM, Houghova metoda.

Keywords

Kinect, registration, point cloud, ICP, NDT, LUM, Hough transformation.

Citace

Ondřej Fuchsík: Rekonstrukce 3D scény z obrazových dat, bakalářská práce, Brno, FIT VUT v Brně, 2013

Rekonstrukce 3D scény z obrazových dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Spanela PhD. Uvedl jsem všechny prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Fuchsík
14. května 2013

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Michalu Spanělovi PhD. za cenné a užitečné rady, které vedly k dokončení mé práce. Dále chci poděkovat své rodině a přátelům za podporu.

Obsah

Kapitola 1

Úvod

Rekonstrukce 3D scény je velice rozšířené téma z oblasti počítačové grafiky a počítačového vidění. Rekonstruování scény má široké využití. Můžeme jej využít kupříkladu při mapování neprozkoumaných oblastí využitelné například pro záchranné složky [12], ovládání uživatelského rozhraní jako jsou hry, internetové prohlížeče [10] nebo tvoření modelů budov (věcí) pro archivaci či virtuální prohlídky. Důležitým faktorem jsou zdrojová data a způsob jejich získávání. Data lze získávat pomocí fotoaparátu, klasických kamer nebo pomocí pohybových senzorů.

Cílem mé práce je vytvoření nástroje pro architekty, který automaticky zpracuje data získaná pomocí pohybového senzoru a vytvoří jejich půdorys. Předpokládaná data jsou místnosti nebo objekty v budovách. Zvolil jsem pohybový senzor Kinect od firmy Microsoft, protože je veřejnosti dostupný a jsou k dispozici OpenSource knihovny pro práci s ním. Získaná data z Kinectu jsou tvořena jednotlivými snímky. Každý snímek je reprezentován jako mračno bodů. Aplikce nejdříve spojí jednotlivá mračna do jednoho velkého pomocí metody Iterative Closest Point (ICP). Poté za použití projekce bodů do roviny a Houghovy transformace vytvoří přibližný půdorys místnosti ve formě vektorového popisu (tj. čar).

Práce je rozdělena do osmi kapitol. Kapitoly 2, 3 a 4 jsou teoretické. V kapitolách 2 a 3 jsem představil metody použitelné pro spojení (registraci) mračen bodů a metody pro detekci objektů v obraze. V kapitole 4 se věnuji pohybovému senzoru Microsoft Kinect a jeho parametrům. Jádro práce, tedy návrh a implementace, je soustředěno do kapitol 5 a 6. V kapitole 7 jsou umístěny experimenty pro ověření funkčnosti a výsledky jsou shrnuty v kapitole 8.

Kapitola 2

Metody registrace mračen bodů

Problém srovnání dvou a více mračen bodů, dívajících se na scénu z různých úhlů do jednoho mračna, se nazývá registrace. Cílem metod pro registraci je najít vzájemnou pozici a orientaci dvou mračen, které obsahují překrývající se oblasti. To znamená nalézt transformační matici mezi dvěma mračny.

Pro tuto úlohu lze použít multi–platformní knihovnu Point cloud library[3]. Tato knihovna slouží pro zpracování obrázků a mračen bodů. Skládá se z mnoha algoritmů pro filtrování, detekci klíčových bodů, rekonstrukci povrchu, registraci a dalších. K registraci lze použít jednu z následujících metod.

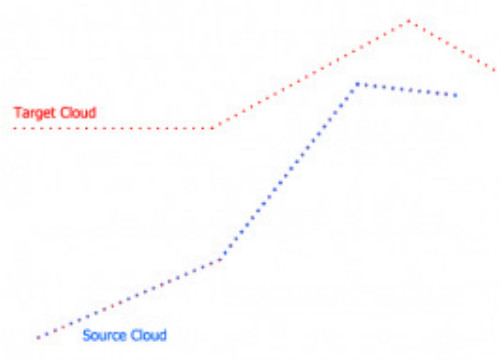
2.1 Iterative Closest Point (ICP)

Algoritmus použitelný pro takové dvě mračna bodů, které obsahují stejnou scénu, zachycenou z různých pozic. Hlavní koncept celého algoritmu lze dle [11] shrnout do tří kroků:

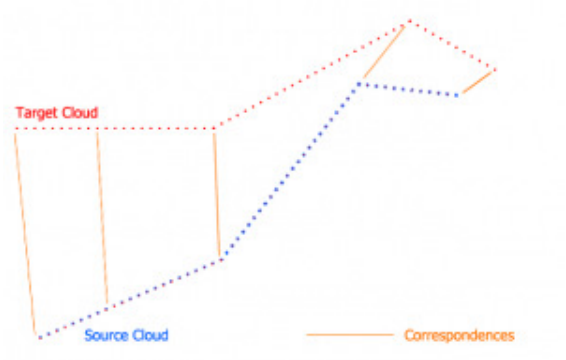
1. Výpočet společných prvků mezi dvěma mračny
2. Výpočet transfomační matice, minimalizující vzdálenost mezi společnými body
3. Úprava mračna

K výpočtu je potřeba ještě definovat maximální vzdálenost H, představuje fakt, že některé body z prvního mračna neobsahují korespondující prvky z druhého mračna. V mnoha implementacích ICP algoritmu si pří výběru hodnoty H musíme vybrat mezi přesností a konvergencí metody. Nízká hodnota znamená, že metoda bude špatně konvergovat. Naopak vysoká hodnota způsobí nesprávné zarovnání mračen.

Prvním krokem algoritmu je najít takové prvky, které se vyskytují v obou mračnech viz obr. 2.1 a obr. 2.2. Pokud bychom znali všechny takové prvky stačilo by zjistit jak jsou tyto prvky pootočeny od originálu. Poté použili transformaci a získali dokonalou shodu viz obr. 2.3.



Obrázek 2.1: Počáteční stav algoritmu ICP. Převzato z [1]



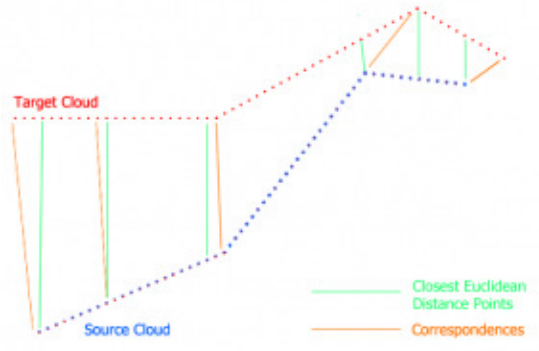
Obrázek 2.2: Ideální první iterace ICP. Převzato z [1]

Obrázek 2.3: Ideální výsledek algoritmu. Převzato z [1]

Algoritmus považuje za odpovídající si ty body, které jsou nejblíže. Za nejbližší bod je považován takový bod z cílového mračna, jenž má nejmenší Euklidovskou vzdálenost k bodu z mračna zdrojového. Mějme dva body p (z cílového mračna) a q (ze zdrojového mračna), Euklidovská vzdálenost je velikost úsečky mezi těmito body v prostoru. Velikost úsečky lze spočítat vztahem 2.1.

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \quad (2.1)$$

Můžeme porovnat jaké body jsou vybrány pomocí Euklidovské vzdálenosti a těmi, které by měli být vybrány v ideálním případě viz obr. 2.4.

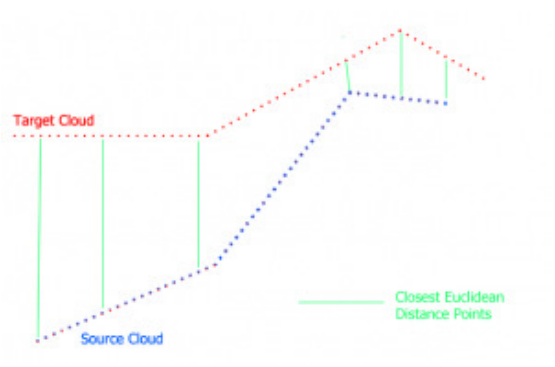


Obrázek 2.4: Porovnání výběru bodů. Převzato z [1]

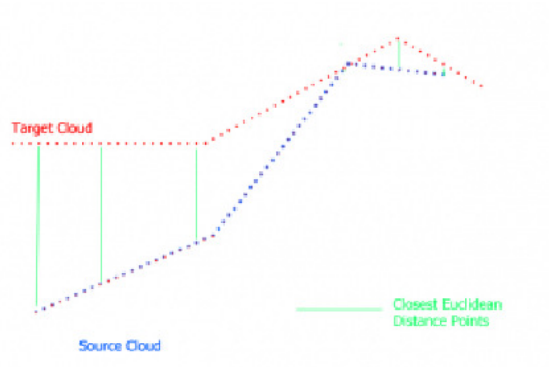
Jak lze vidět, některé z vybraných bodů s nejmenší Euklidovskou vzdáleností mohou být použity, protože rozdíl je minimální. Následně ICP musí vybrat pouze některé vhodné body, tedy provést minimalizaci viz rovnice 2.2.

$$argmin_R, tf(R,t) = \frac{1}{N}\sum_{i=1}^{N} x_i^{target} - (Rx_i^{source} + t) \tag{2.2}$$

Po první a druhé iteraci mohou vypadat mračna viz obr. 2.5 a obr. 2.6.



Obrázek 2.5: První iterace ICP. Převzato z [1]



Obrázek 2.6: Druhá iterace ICP. Převzato z [1]

Po každé iteraci máme k dispozici transformační matici následujícího tvaru. Vnitnřní matice 3x3 je rotační matice a sloupec č.4 je translační vektor.

$$\mathbf{M} = \begin{Vmatrix} R1 & R2 & R3 & x \\ R4 & R5 & R6 & y \\ R7 & R8 & R9 & z \\ 0 & 0 & 0 & 1 \end{Vmatrix} \tag{2.3}$$

Použití algoritmu je široké například tvoření 3D map různých prostorů a oblastí. Algoritmus je implementován v několika open–source knihovnách například v knihovně PCL [3]. Také má dle [11] další variace například IDC a pIC.

2.2 LUM algoritmus

LUM algoritmus je založený na práci Lu a Milios [4], která řeší spojení sítě laserových scanů a jejich korespondujícími rozdíly pro vytvoření lineární soustavy rovnic. Jejich řešení koresponduje s optimálním vyjádřením všech laserových scanů. Tyto laserové scany jsou 2D a LUM algoritmus rozšiřuje tento algoritmus o 3D scany a o volné pohybování objektem ve 3D prostoru, tedy 6 DoF (six degrees of freedom) [2]. Jedná se o GraphSLAM algoritmus.

Kroky algoritmu jsou následující:

1. Výpočet společných prvků
2. Pro každou spojnici mezi dvěma uzly v grafu, vypočítat měrný vektor $D_{i,j}$ a jeho kovarianci $C_{i,j}$
3. Ze všech D a C zkonstruovat lineární systém $GX = B$, s G a B dle 2.4,2.5 a 2.6 a vyřešit pro X
4. Aktualizovat pozice a jejich kovarianty viz [2] kapitola 3, sekce 3.4.

$$G_{i,i} = \sum_{j=0}^{n} C_{i,j}^{-1} \tag{2.4}$$

$$G_{i,j} = C_{i,j}^{-1} \qquad (i \neq j) \tag{2.5}$$

$$B_{i,j} = \sum_{j=0, j\neq i}^{n} C_{i,j}^{-1} \overline{D}_{i,j} \tag{2.6}$$

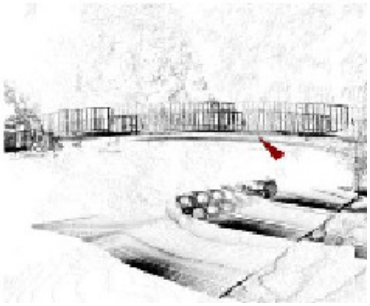
V algoritmu jsou nejnáročnější operace pro vyřešení soustavy rovnic $GX = B$ a výpočet korespondencí. G je matice 6x6 a je řešena Cholského dekompoziční metodou.

Algoritmus je vhodný pro vnitřní skenování místností, jak lze vidět na obrázku 2.7.

Obrázek 2.7: Nezarovnané a zarovnané scany metodou LUM. Převzato z [2]

Dále pro scanování venkovních prostor viz obrázky 2.8, 2.9 a 2.10, kde jsou porovnány výsledky s metodou ICP. Výraznou změnu lze pozorovat při zaregistrování mostu jak je naznačeno šipkou na obr.2.8 a 2.9.

Obrázek 2.8: Algoritmus ICP. Převzato z [2]



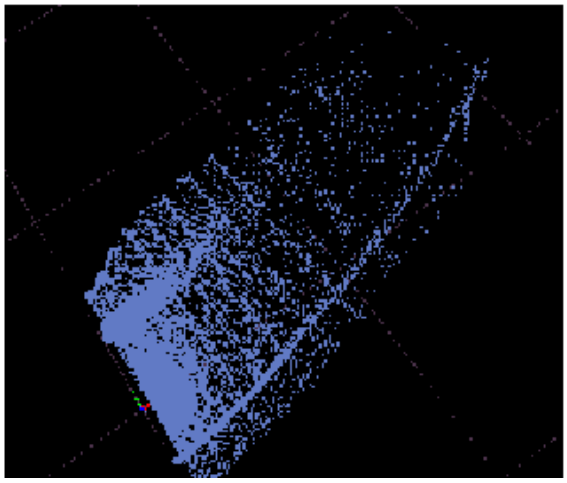
Obrázek 2.9: LUM algoritmus. Převzato z [2]

Obrázek 2.10: Reálné prostředí. Převzato z [2]

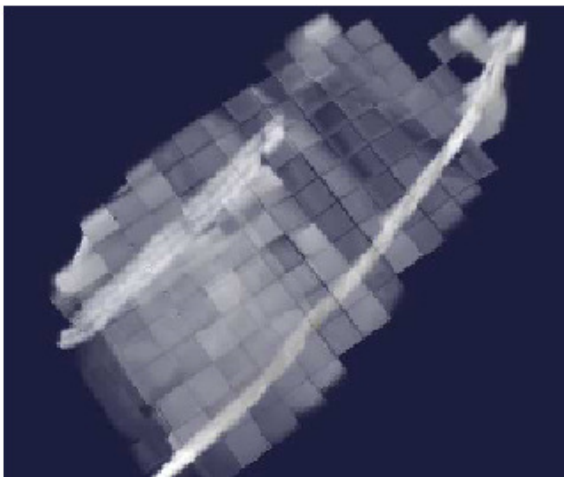
2.3 Normal Distribution transform (NDT)

Metoda NDT [5] byla vytvořena pro důlní stroje. V této metodě je důležitá reprezentace referenčních scanů. Protože používá reprezentaci pomocí průběžných prvních a druhých derivací, lze použít pro registraci standardní numerické optimalizační metody. Například časem ověřená Newtonova metoda. Jelikož body v referenčním scanu nejsou použity přímo pro porovnávání, není potřeba použít výpočetně náročné hledání nejbližších sousedních bodů. NDT mapuje mračna na hladkou povrchovou reprezentaci, která je popsaná jako soubor lokálních funkcí hustoty pravděpodobnosti (HP), z nichž každá popisuje tvar části povrchu.

Prvním krokem algoritmu je rozdělit prostor scanu do mřížky (ve 2D na čtverce, ve 3D na krychle). Hustota pravděpodobnosti je vypočtena pro každou buňku, na základě rozdělení bodů uvnitř buňky. Každá funkce hustoty pravděpodobnosti může být chápána jako aproximace lokálního povrchu, popisující jak jeho orientaci tak hladkost. Normální rozdělení nám dá po částech spojitou reprezentaci mračna bodů, se spojitou derivací. Reprezentaci jako mračno bodů můžeme vidět na obrázku 2.11 a reprezentaci v NDT na obrázku 2.12.



Obrázek 2.11: Reprezentace jako mračno. Převzato z [5]



Obrázek 2.12: NDT reprezentace. Převzato z [5]

Cílem algoritmu je tedy najít pozici aktuálního scanu, s maximální pravděpodobností, že body leží na povrchu referenčního scanu. Aktuální scan je reprezentován jako mračno bodů $X = x_1, \dots x_n$. Rotace a posunutí určitého scanu může být dána jako vektor p. Dále je zde prostorová transformační funkce $T(p, x)$, která pohybuje bodem x v prostoru s pozicí p. Výpočtem funkce hustoty pravděbodobnosti pro nascenované body dle rovnice 2.7 je získána nejlepší pozice p, je to ta hodnota, jenž maximalizuje pravděpodobnostní funkci.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{2.7}$$

Úplný výklad a detailní vysvětlení metody je možné nalézt v disertační práci [5] v sekci 6.

Kapitola 3

Detekce přímek v obraze pomocí Houghovy transformace

V oblasti zpracování obrazu je častým problémem detekovat různé základní tvary jako přímky, kružnice nebo elipsy. Lze použít různé hranové detektory, ale vždy zde budou nedokonalosti v obraze jako chybějící pixely, šum nebo nedokonalý detektor hran. Kvůli tomu není zcela jednoduché získat pomocí hranových detektorů všechny přímky, kružnice nebo elipsy. Proto byla vynalezena Houghova transformační metoda.

Klasická Houghova transformace byla vytvořena pro detekování přímek, ale později se rozšířila o identifikování libovolných tvarů, například kružnice a elipsy. Rozšířená Houghova transformace byla vynalezena Richardem Dudou a Petrem Hartem v roce 1972, kteří ji pojmenovali jako Generalized Hough transform [11].

Přímku lze v obraze vyjádřit pomocí rovnice 3.1 a graficky vykreslit pro každou dvojici bodů (x, y). V Houghově transformaci není přímka chápána jako dvojice bodů $(x_1, y_1), (x_2, y_2)$ atd. ale jejími parametry (m, b). Z výpočetních důvodů je lepší použít místo kartézského systému souřadnic, systém polárních souřadnic. Proto se využívají parametry r a θ. Parametr r vyjadřuje vzdálenost mezi počátkem souřadnic a přímkou. Theta nám udává úhel mezi počátkem a nejbližším bodem [9].

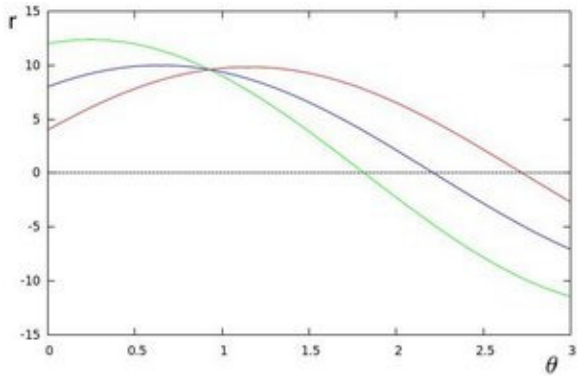
Z rovnice 3.1 můžeme vyjádřit y pro výpočet přímky viz rovnice 3.2. Obecně můžeme pro každý bod (x_0, y_0) definovat soubor přímek, které prochází daným bodem dle rovnice 3.3. To znamená, že každá dvojice (r_θ, θ) vyjadřuje každou přímku procházející bodem (x_0, y_0) [8].

$$r = x\cos\theta + ysin\theta \tag{3.1}$$

$$y = (-\frac{\cos\theta}{\sin\theta})x + (\frac{c}{\sin\theta}) \tag{3.2}$$

$$r_\theta = x_0\cos\theta + y_0\sin\theta \tag{3.3}$$

Když pro každý bod (x_0, y_0) vykreslíme graf přímky, které jím prochází, dostaneme graf sinusoidy. Pro několik dvojic vypadají grafy jako v obrázku 3.1. Pokud se v rovině $\theta - r$ protínají křivky dvou různých bodů, potom obě náleží jedné přímce [9].

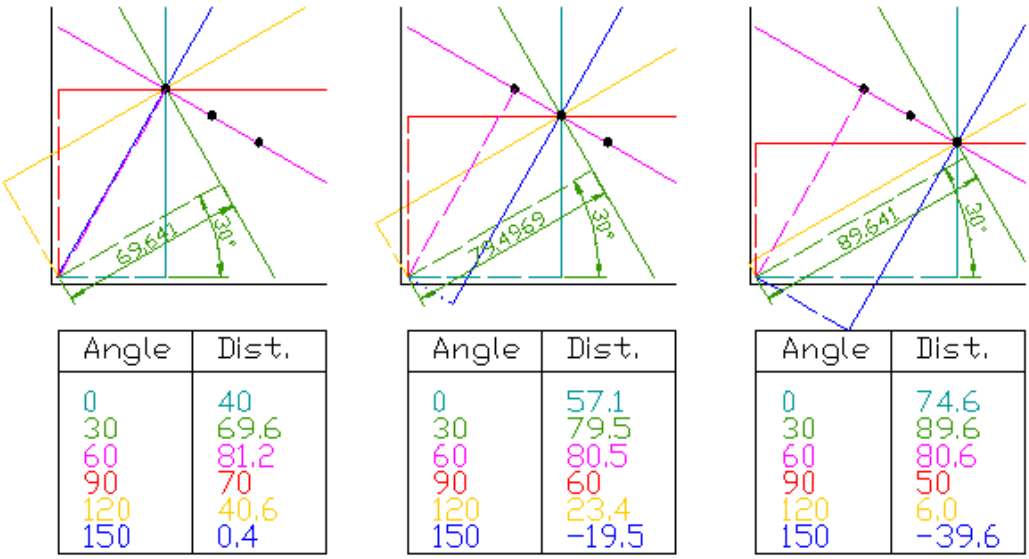


Obrázek 3.1: Ukázka pro tři dvojice. Převzato z [8]

Průsečík křivek z obrázku 3.1 je jeden bod se souřadnicemi $(0.925, 9.6)$, jsou parametry (θ, r) nebo také přímka, na níž leží body $(x_0, y_0)(x_1, y_1)$ a (x_2, y_2). To znamená, že můžeme detekovat přímku pomocí průsečíků křivek. Dále je definován práh, minimálního počtu průsečíků, které jsou nutné pro detekování přímky [9].

Houghova transformace používá k detekování existence přímky pole, nazývané akumulátor. Rozměr akumulátoru je úměrný počtu neznámých parametrů v Houhgově transformaci. Lineární Houghova transformace má neznáme dva parametry a to (m, b) nebo (r, θ). Transformace pro každý pixel a jeho sousední pixely určuje, jestli je zde dost hran v daném pixelu. Vypočtou se parametry dané přímky a poté se zkontroluje zásobník akumulátoru, zda tam vypočtené parametry patří a následně aktualizuje zásobník [9].

Nalezením zásobníků s nejvyššími hodnotami, například vyhledáním lokálních maxim v prostoru akumulátoru, mohou být získány přímky s největší pravděpodobností výskytu. Výsledeky Houghovy transformace jsou uloženy v matici, která se často označuje za akumulátor. Jednou z dimenzí matice jsou úhly θ ostatní dimenze jsou vzdálenost r, a každý prvek obsahuje hodnotu, ta nám říká kolik pixelů leží na přímce s parametry (r, θ). Prvek s nejvyšší hodnotou nám říká, jaká přímka je nejvíce zastoupena ve vstupním obrazu viz [9].



Angle	Dist.
0	40
30	69,6
60	81,2
90	70
120	40,6
150	0,4

Angle	Dist.
0	57,1
30	79,5
60	80,5
90	60
120	23,4
150	-19,5

Angle	Dist.
0	74,6
30	89,6
60	80,6
90	50
120	6,0
150	-39,6

Obrázek 3.2: Ukázka pro tři body a hodnoty přímek v bodech. Převzato z [9]

Na 3.2 můžeme vidět ukázku pro tři body. Pro každý bod jsou vykresleny přímky, které jimi prochází. Dále jsou ke každé přímce vykresleny kolmice, které prochází počátkem. Poté jsou zapsány úhly a vzdálenosti každé z přímek.

Houghova transformace je efektivní, pokud počet hlasování pro určitou přímku spadne do určitého zásobníku, to má za následek ignorování šumu v pozadí a není hlavní zásobník zneviditelněn tím, že by hlasy spadly do sousedních zásobníků. Také pokud je počet parametrů vysoký (například Houghova transformace s více jak třemi parametry), je pravděpodobnost, spadnutí průměrného počtu hlasů do jednoho zásobníku malá. To znamená, že tyto zásobníky nebudou mít o moc větší počet hlasů, než jejich sousedi. Houghova transformace je náchylná na počet parametrů a složitost je dána vztahem $O(A^{m-2})$. Kde A je velikost obrázku a m je počet parametrů [9]. Před použitím Houghovy transformace na zašuměný obrázek je lepší provést vhodný pre–processing obrázku.

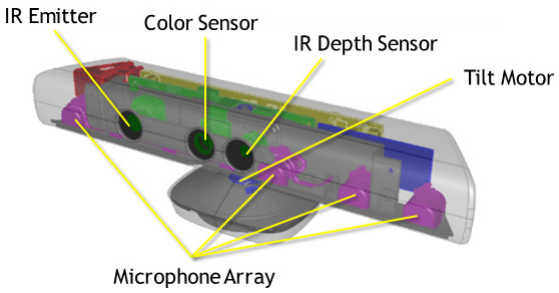
Kapitola 4

Microsoft Kinect

Kinect je zařízení pro snímání pohybu, které vytvořila firma Microsoft pro svou herní konzoli Xbox 360 a pro počítače s operačním systémem Windows. Umožňuje uživateli ovládat svůj Xbox 360, či stolní počítač, aniž by se musel dotknout ovladače. Tyto zařízení se ovládají mluvenými příkazy nebo gesty, např. zvednutí ruky, úkroky apod. V roce 2011 byly vytvořeny vývojové prostředky pro Windows 7, což umožnilo vývojářům začít psát aplikace v jazycích C++/CLI, C# nebo Visual Basic .NET [13].

4.1 Technické parametry

Senzor je usazen na otočném a nastavitelném podstavci. Obal je vyroben z plastu. Kamery a senzory jsou uloženy uvnitř v horizontální poloze, jak lze vidět na obrázku 4.1.



Obrázek 4.1: Konstrukce Kinectu. Převzato z [6]

Uvnitř obalu se tedy nachází RGB kamera, která ukládá tří kanálová data v rozlišení 1280*960 pixelů. Lze tak pořizovat barevné snímky. Dále infračervený emitor a infračervený hloubkový senzor. Emitor vyzařuje světelné paprsky a senzor odražené paprsky snímá. Tyto odražené paprsky jsou zpracovány a z nich je následně vypočtena vzdálenost mezi senzorem a objektem. Lze tedy získat vzdálenost objektu. Také jsou zde uloženy čtyři mikrofony pro snímání zvuku. Jelikož jsou zde mikrofony čtyři, umožňuje nám to zjistit pozici zdroje zvuku a také směr zvukové vlny. V podstavci je zabudován 3–osý akcelerometr, schopný snímat gravitační zrychlení až do dvou G. Ten nám umožní zjistit aktuální orientaci senzoru [6].

Rozlišení hloubkové a RGB kamery lze nastavit pomocí konfiguračního souboru, stejně tak mód hloubkové kamery. Dále se v konfiguračním souboru nastavuje zda chceme sledovat pohyby a gesta osob. Základní rozsah hloubkové kamery je od 800mm do 4000mm. Pokud se jedná o Kinect pro Windows, lze jej přepnout do blízkého režimu, tím se nastaví rozsah kamery od 500mm do 3000mm [6]. Z toho důvodu, že Kinect využívá infračervené paprsky, nedokáže rozpoznat například skleněné dveře a také nebude správně fungovat na přímém slunci, například v exteriérech.

Kinect	Specifikace
Zorný úhel	53° ve svislé poloze, 57° horizontální zorné pole
Rozsah vertikálního náklonu	+27° až -27°
Počet snímků za sekundu	30 snímků za sekundu
Zvukový formát	16–kHz, 24–bit mono PCM
Charakteristika audio vstupu	Pole 4 mikrofonůs 24–bitovým analogodigitálním převodníkem, potalčení šumu, zpracování signálu včetně akustické ozvěny.
Charakteristika akcelerometru	A 2G/4G/8G akcelerometr nastaven na rozsah dvou G, s přeností 1° pro horní hranici

Tabulka 4.1: Technické parametry senzory. Převzato z [6]

4.2 Vývoj a použití

Základní použití Kinectu je to, pro které byl zkonstruován. Tedy pro interakci při hraní her a ovládání aplikací pomocí gest a mluveného projevu, například autentizace uživatelů. Postupem času se vyvíjeli různé open-source ovladače, kupříkladu NITE (PrimeSense) nebo OpenNI software framework, ten umožňuje číst data ze senzorů.

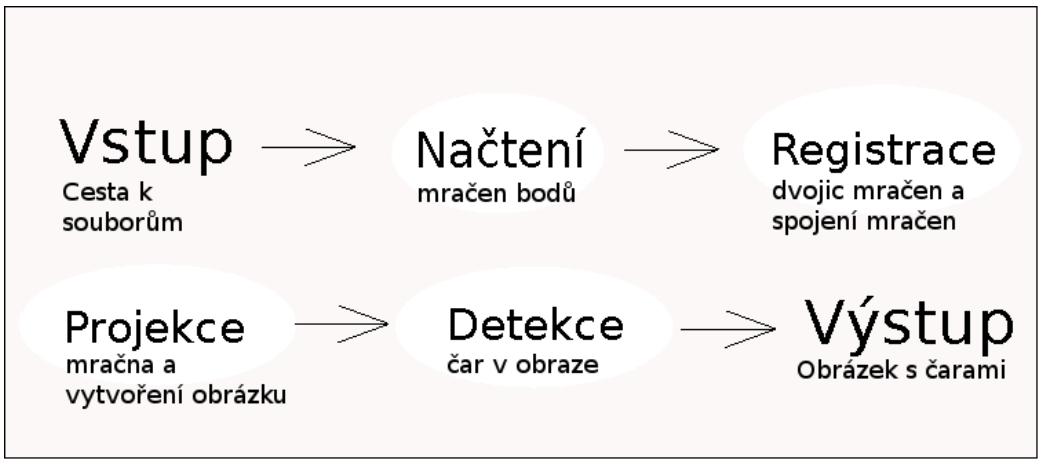
Zařízením se zabývalo mnoho vývojářů, např. Philipp Robbel z MIT zkombinoval Kinect s robotem, aby zmapoval místnost ve 3D a reagoval na gesta lidí, nebo team z MIT vyvíjející JavaScriptové rozšíření pro prohlížeč Google Chrome, které umožňuje jej ovládat pomocí gest. Robot Locomotion Group z MIT použili ovladače pro vývoj pohybem řízeného rozhraní podobnému z filmu Minority Report. Další vývojáři se zabývali 3D scanováním místností a 3D SLAMem. Jedni z vývojářů použili Kinect pro simulování hry na piáno, kdy senzory snímali prsty hudebníka a podle toho reproduktory vydávali zvuk.

Kinect je také využitelný v medicíně. Výzkumníci z univerzity v Minesotě použili Kinect pro měření řady symptomů poruch, jako je autismus, poruchy pozornosti a jíné [13].

Kapitola 5

Návrh aplikace pro tvorbu půdorysu

Celou aplikaci jsem se rozhodl rozdělit na menší části řešící pouze určitou problematiku. Jedním z důvodů byl ten, že jednotlivé části jsou použitelné i samostatně a lze je tak využít i v jiných aplikacích. Jednotlivé části pojí dohromady skript. Rozdělení lze znázornit následujícím diagramem 5.1



Obrázek 5.1: Diagram aplikace

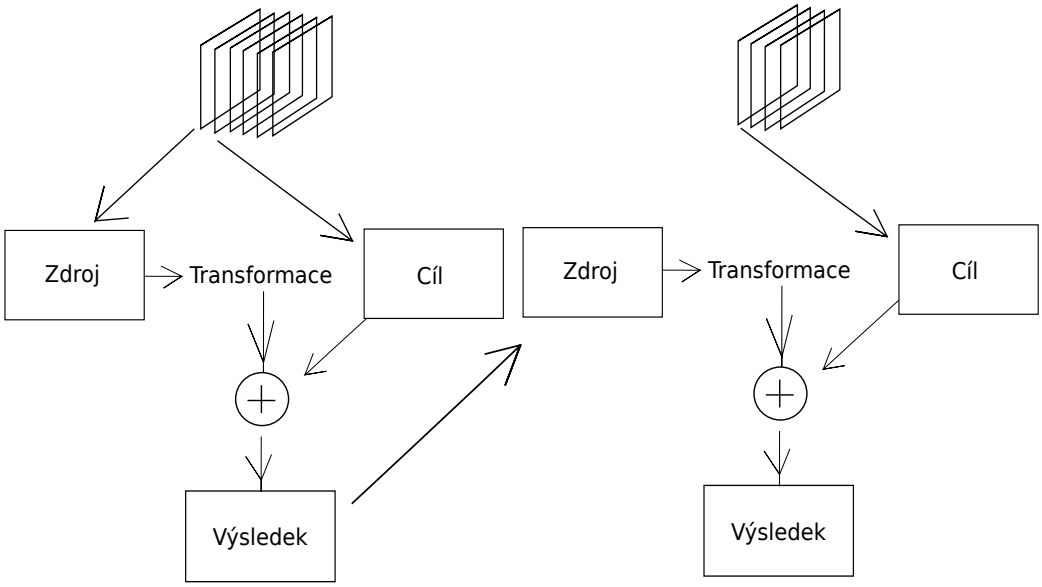
5.1 Nahrávaná scéna

Mezi předpokládané scény zpracovávané aplikací jsem zařadil uzavřené místnosti bez větších skleněných ploch. Důvodem mého rozhodnutí bylo zjištění, že Kinect využívá infračervené paprsky, a proto nedokáže detekovat skleněnou plochu. V exteriérech také nefunguje ideálně. Dále jsem předpokládal, že pro registrační algoritmus bude lepší, když místnost bude obsahovat objekty, pro přesnější korelaci mezi snímky. Například pokud by byla nahrána pouze holá stěna a nebyl by zde žádný objekt. Myslel jsem si, že by algoritmus měl málo význačných bodů a nemusel by zaregistrovat snímky úplně správně. Tyto předpoklady jsem si chtěl ověřit při experimentech.

5.2 Zpracování vstupních dat

Aplikace zajišťující zpracování vstupních dat realizuje registraci a spojení transformovaných snímků. Pro registrování jednotlivých snímků jsem zvolil algoritmus Iterative Closest Point (ICP) 2.1. Je to metoda méně efektivní než jiné metody ale pro můj účel se mi zdála použitelná a dostačující.

V mém návrhu jsem se rozhodl vyzkoušet dva přístupy. První je založen na malém počtu iterací metody ale samotná metoda se počítá mnohokrát po sobě. To mi dalo možnost zobrazit si jednotlivé iterace metody přes visualizér z knihovny PCL. Druhou možností je si nechat vypočíst algoritmus ICP bez krokování a nechat knihovnu PCL aby si řídila počet kroků samostatně. Tento způsob by měl být rychlejší ale méně přesný. Poté co jsou vypočteny jednotlivé transformační matice mezi jednotlivými snímky, nastane proces slučování snímků a jejich transformace. Znázornění na diagramu 5.2.



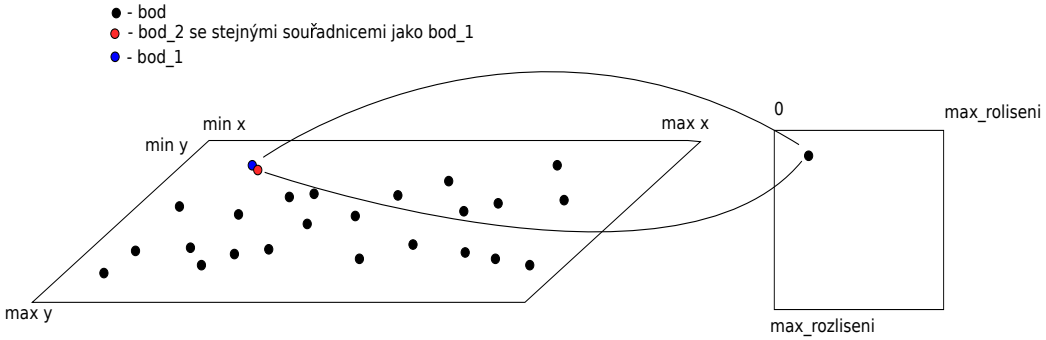
Obrázek 5.2: Proces slučování a transformace snímků

Výstupem aplikace bude jeden snímek, obsahující jedno velké mračno bodů, reprezentující celou natáčenou scénu.

5.3 Vytvoření obrázku mračna

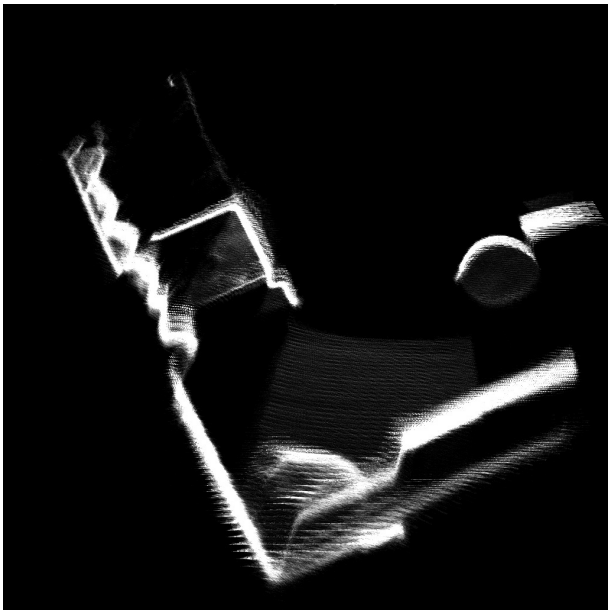
Na vstup této aplikace je přiveden snímek z předchozí části. Při mém návrhu jsem použil následující postup:

1. Načíst obrázek.
2. Zjistit minimální a maximální hodnoty souřadnic x a y .
3. Namapovat rozsah osy x a osy y na rozlišení obrázku.
4. Pomocí mapovací funkce zjistit kde do obrázku padne každý bod.
5. Iterovat o konstantu K dle rozlišení obrázku.
6. Uložit výsledek jako obrázek ve formátu PNG.



Obrázek 5.3: Grafické znázornění postupu č. 2.

Scéna vytvořená tímto postupem je zobrazena na obr. 5.4.



Obrázek 5.4: Roh místnosti s použitím návrhu č. 2. Obrázek je vytvořen v rozlišení 1024 na 1024 bodů.

5.4 Zpracování obrazu a zobrazení úseček

Aby byl výsledek co nejpřesnější, zvažoval jsem volbu post–procesingu obrázku, ikdyž si Houghova transformace dokáže poradit s šumem v obraze. Vybral jsem si mediánový filtr, který daný obrázek vyhladí a odstraní nedokonalosti způsobené projekcí.

Kapitola 6

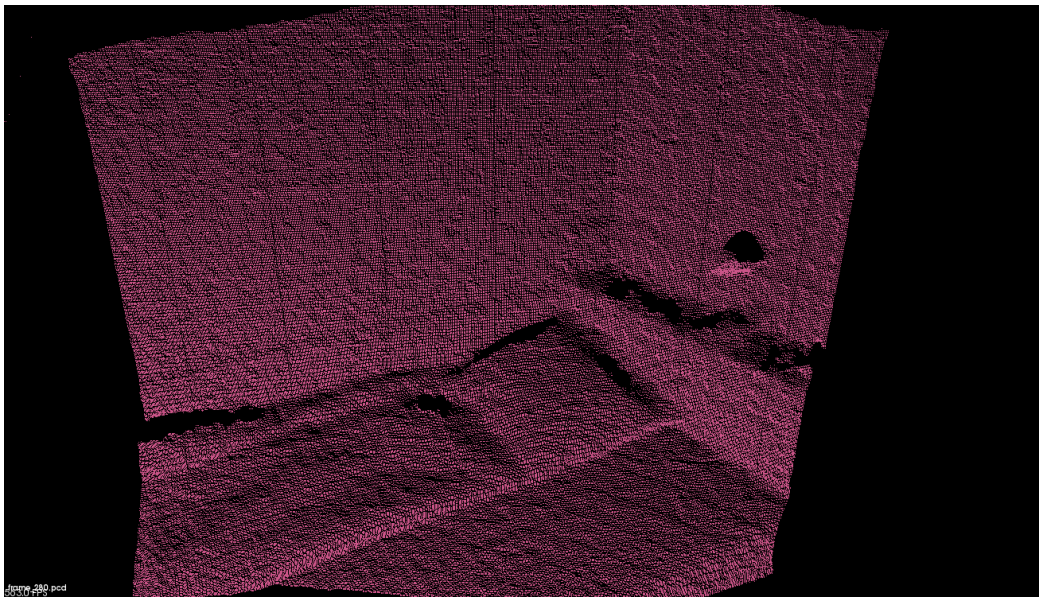
Implementace

Jak již bylo zmíněno v předchozí kapitole, výsledná aplikace je rozdělena na tři části. Každá část má svůj zdrojový kód a také vlastní `Makefile` soubor. Dále je zde skript napsaný v Bash shellu `run.sh` pro spojení jednotlivých částí. V následujících odstavcích bych se chtěl zmínit o vstupních datech, nejdůležitějších funkcích a nastaveních různých metod.

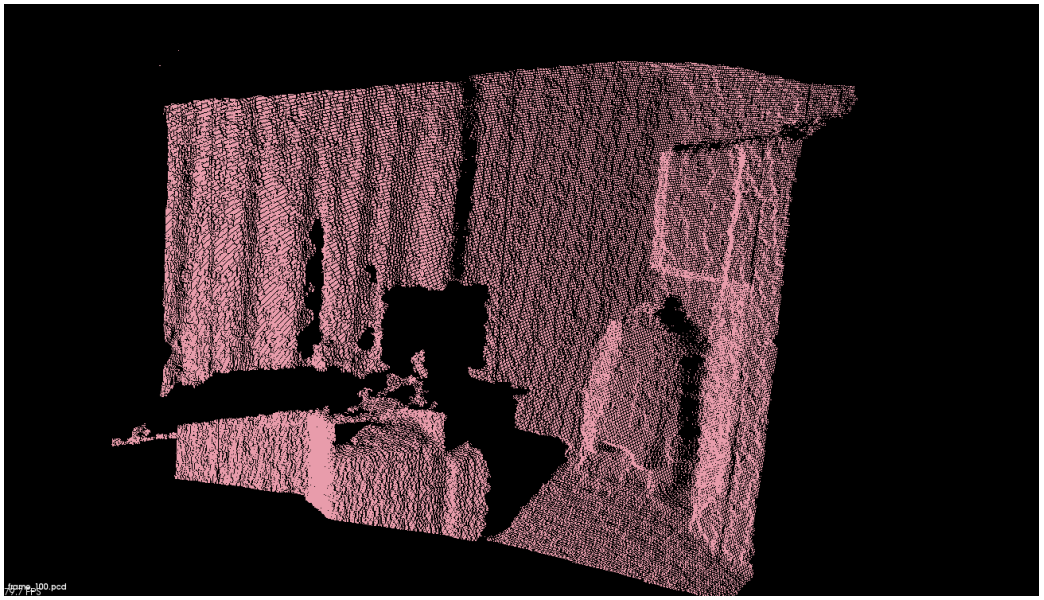
6.1 Vstupní data a jejich formát

Vstupní data jsou nahrána pomocí aplikace z knihovny PCL `pcl_openni_pcd_recoder`. Ta pořizuje snímky pomocí Kinectu a zapisuje je do adresáře zvoleného pomocí přepínače `-dir`. Knihovna PCL používá formát PCD (Point Cloud Data). S vývojem knihovny PCL se vyvíjel i tento formát, a proto je možné se setkat s různými verzemi. Jak data vypadají v prohlížeči PCD souborů nám ukazují obrázky 6.1 a 6.2 Každý soubor obsahuje hlavičku, která udává:

1. Verzi PCD souboru
2. Jakým způsobem jsou reprezentovány body
   (a) Souřadnice $[x, y, z]$
   (b) Souřadnice $[x, y, z]$ a barva bodu
   (c) Souřadnice $[x, y, z]$ a povrchové normály
   (d) A další...
3. Velikost dimenze v bytech
4. Datový typ
5. Šířka mračna
6. Výška mračna
7. Pohled na mračno
8. Počet bodů
9. Způsob uložení PCD (binární nebo ASCII)



Obrázek 6.1: Ukázka PCD dat pořízených Kinectem – postel



Obrázek 6.2: Ukázka PCD dat pořízených Kinectem – roh místnosti

6.2 Aplikace `registrace`

Zdrojový kód se nachází v souboru `registrace.cpp`. Ve zdrojovém kódu se nachází funkce na načtení vstupních dat `loadData`, funkce počítající transformační matice `pairAlign` a funkce `main`. Pomocí parametrů příkazové řádky předáváme programu PCD soubory a parametrem `-s` určujeme rychlost výpočtu algoritmu ICP, který je použit ve funkci `pairAlign`.

Mezi části, které je vhodné zmínit patří funkce pro výpočet transformačních matic. Jako vstup funkce jsou dvě mračna, mezi kterými se má vypočíst matice. Jedno mračno je označeno jako zdrojové a jedno jako cílové. Ve funkci je použit filtr pro zmenšení počtu bodů

což je dobré použít pro data s více jak 100 000 body ve snímku. Jsou zde dvě možnosti registrace, pomalá a rychlá. Pomalá je iterována přes cyklus `for`. Zvolený počet iterací cyklu, byl zvolen po testování na 1000. Při vhodně natočené scéně stačilo iterací méně. Nicméně ve složitějších scénách nebylo párování mračen ideální a výsledné mračno bylo zcela nepoužitelné. V metodě ICP lze nastavit pomocí funkce `setMaximumIterations(number)` maximální počet iterací, po kterých skončí registrace dvou snímků. Dále lze nastavit s užitím metody `setTransformationEpsilon(epsilon)` konvergenci metody. Epsilon nám udává námi zvolenou hranici, kdy je součet rozdílů mezi aktuální a poslední transformací ještě menší. Aby mohla být nalezena finální matice, musí být součet chyb menší než námi zadaná chyba přes metodu `setEuclideanFitnessEpsilon(distance)`.

Nakonec je již předána finální transformace do funkce `main`.

V této funkci se spouští tedy párování dvojic mračen a následně se ukládají transformační matice do vektoru. Poté co se vypočítají transformační matice, dochází k transformaci snímků a jejich spojování dle návrhu 5.2 z předchozí kapitoly. Zde uvedu pouze stručný pseudokód:

```
vytvorit_soubor_pro_vystup()
nacist_prvni_dvojici_mracen()
nacist_jejich_transformacni_matici()
transformuj_prvni_mracno_do_druheho()
spoj_druhe_mracno_s_transformovanym()
pro_zbytek_mracen{
   nacti_dalsi_mracno_v_poradi()
   nacti_nasledujici_matici()
   transformuj_mracno_z_predchoziho_kroku_do_aktualniho()
   spoj_aktualni_mracno_s_mracnem_z_minule_iterace()}
```

Možná stojí za zmínku, že ve funkci `loadData` jsou načítány jednotlivé snímky a jsou zbavovány nedefinovaných bodů (NaN), což snižuje částečně velikost snímku.

Omezením této aplikace je počet zpracovaných snímků. V rámci implementace se mi nepodařilo zajistit správné využití paměti. Program jako takový umí zaregistrovat celou scénu, problém nastává při transformaci a následném spojování jednotlivých snímků. Kdy neustále roste počet využité RAM paměti počítače.

Nutno podotknout, že zdrojový kód je převzat z tutoriálu umístěných na stránkách knihovny PCL [3] a upraven tak, aby reflektoval požadavky mé aplikace.

6.3 Vytvoření obrázku z mračna bodů

Vytvoření obrázku z mračna bodů zajišťuje aplikace `picture`, jejíž zdrojový kód je uložen v souboru `picture.cpp`.

Vstupním parametrem programu je jednak mračno bodů ve formátu PCD a rozlišení výsledného obrázku dané přepínačem `-r`. Jsou povolena pouze tři různá rozlišení a to $1024 * 1024$, $2048 * 2048$ a $4096 * 4096$.

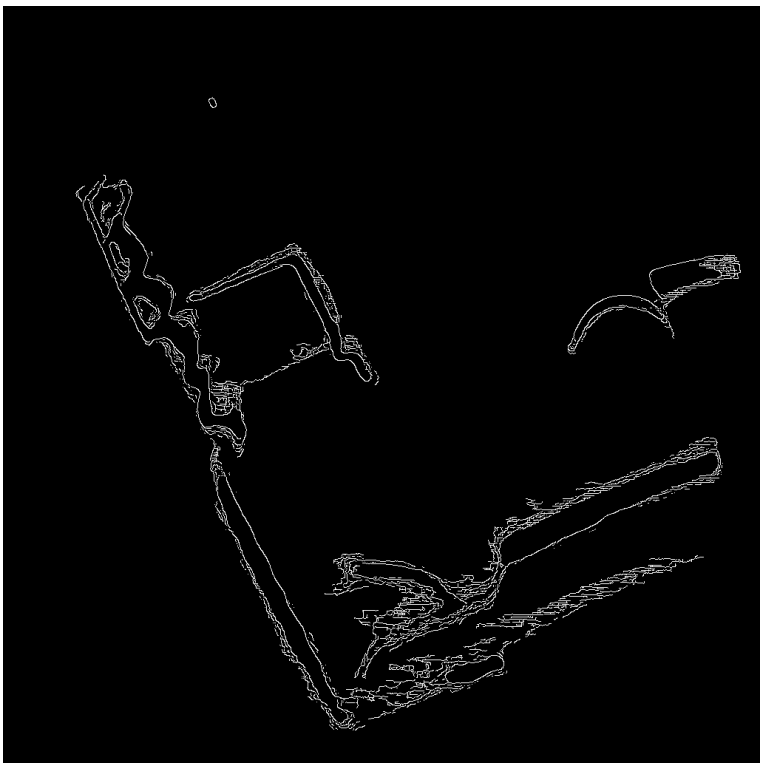
Po zpracování parametrů příkazové řádky a načtení mračna, jsou vypočítány maximální a minimální hodnoty souřadnic x a y. Následně je implementována projekce bodů z mračna do roviny. Používám mapovací funkci `translate()`, která přemapuje bod ze zdrojového rozsahu do rozsahu velikosti obrázku.

V rámci testování aplikace se jevilo jako vhodné pro různá rozlišení zvyšovat hodnotu intezity jednotlivých bodů ve výsledném obrázku různě. U většího rozlišení se zvyšuje o vyšší hodnotu, jelikož body nejsou mapovány tak často na stejnou souřadnici jako u rozlišení nižšího. Mapovací funkce `translate()` realizuje vztah 6.1.

$$t = b_1 + \frac{(s - a_1)(b_2 - b_1)}{(a_2 - a_1)} \tag{6.1}$$

6.4 Zpracování obrazu a vytvoření půdorysu

Poslední částí programu je nalezení hran a jejich vyznačení. Tomu jsem se věnoval v teoretické části 3. Pro tuto aplikaci jsem se tedy rozhodl zvolit Houghovu trasnformaci z knihovny OpenCV [7]. Což je velice rozšířená knihovna pro manipulaci s obrazem. Zaměřená především na počítačové vidění a zpracování obrazu v reálném čase. Obsahuje interface pro jazyky C++, C, Java a Python a je možné jí provozovat na systémech Windows, Linux, Mac OS, iOS a Android. Knihovna je přizpůsobena k vícejádrovému zpracování, což je její nespornou výhodou. Její použití je velice široké například v interaktivním umění nebo pokročilé robotice. Knihovna je vydána pod BSD licencí a její užití je bezplatné. Při prostudování jednotlivých filtrů v knihovně OpenCV a obrázku z 5.3 jsem se rozhodl pro mediánový filtr. Mediánový filtr odstraní z obrázků osamocené body a obrázek se stane celistvým. Mimo zmíněný filtr jsem se rozhodl použít ještě detektor hran, Canny edge detector, také z knihovny OpenCV. Výsledek detektoru si můžeme prohlédnout na obr. 6.3



Obrázek 6.3: Výsledek detekce hran pomocí Cannyho hranového detektoru.

Prostřednictvím funkce `HoughLinesP()` lze nalézt hledané úsečky v obrázku. Zdrojový kód lze nalézt v `hough_transform.cpp`. I u této aplikace je možné zvolit rozlišení v jakém se obrázek nachází. To není implementováno automaticky, jelikož se nikde nealokuje obrázek dle rozlišení. Zvolením stejného rozlišení se zajistí lepší detekce hran a vyznačení správných úseček. Pro každé rozlišení jsem se snažil najít ideální nastavení jednotlivých funkcí, které jsem použil. Jsou to následující funkce s pořadím daném následujícím kódem:

```
// mediánový filtr
medianBlur(src,dst_aux,stBlur);
// Hranový detektor
Canny(dst_aux, dst, thCanny1, thCanny2,apSize,l2grad);
// převedení do stupňů šedi
cvtColor(dst, cdst, CV_GRAY2BGR);
// Houghova transformace
HoughLinesP(dst, lines, 1, CV_PI/180, threshold, minLineLength, maxLineGap);
```

Jednotlivá nastavení jsou měněna pomocí funkce `setParams()`. U mediánového filtru je vstupem obrázek , výstupem nový obrázek a třetím parametrem je stupeň rozostření. Ten musí být kladné, liché číslo větší než 1. V mém případě jsem zvolil pro menší rozlišení menší rozostření a pro vyšší rozlišení vyšší rozostření. K tomu jsem dospěl po testování, kdy u menšího rozlišení nebylo potřeba zahladit tolik chybných míst.

Hranový detektor má na vstupu obrázek, výstupem je nový obrázek a další parametry upravují detekci hran. Jsou velice podobné pro každé rozlišení.

Houghova transformace je ovlivnena zejména parametry `minLineLength`, ten udává jakou musí mít úsečka délku, aby nebyla ignorována. Dále parametrem `maxLineGap`, udává jak daleko mohou maximálně být jednotlivé body, aby byli vyhodnoceny, že leží na jedné úsečce. Posledním parametrem je `treshold`, je to počet hlasů, které musí získat přímka, aby nebyla ignorována.

Kapitola 7

Návrh experimentů

V této části budou popsány experimenty a v následující kapitole poté vyhodnoceny. V následujících odstavcích zmíním jejich průběh a důvod jejich provedení. Experimenty probíhali na následující sestavě s procesorem Intel Core i3 s frenkvencí 2.4GHz, grafickou kartou Ati Radeon Mobility HD5400 512 MB a 4GB RAM s použitím Microsoft Kinect.

7.1 Natáčení scén

První věcí, která stojí za ověření, je zda má způsob natáčení vliv na kvalitu registrace, tedy i na výsledný půdorys. Pro toto ověření jsem pořídil data pro dvě rozdílné místnosti a to různými způsoby. Aby nebyla testovací data extrémně objemná, zvolil jsem pro sběr dat místnosti střední velikosti s rozlohou $22m^2$ a druhou menší s rozlohou $13.6m^2$.

První místnost viz 7.1 jsem natáčel s Kinectem postaveným na provizorním stativu, aby bylo nahrávání plynulé. Snahou prvního nahrávání bylo připravit pro aplikaci nejideálnější podmínky. Například do místnosti nesvítilo žádné sluneční záření, nebyla snímána žádná okna, byli zavřené skříně a některé poličky, aby algoritmus mohl jednodušeji jednotlivé snímky zaregistrovat. Scénu jsem natáčel přibližně 10 sekund a pořídil jsem 324 snímků, s přibližnou velikostí 620 MB. Jeden snímek je tedy cca 1.9 až 2.0 MB velký.

Druhou menší místnost viz 7.2 jsem natáčel s Kinectem v ruce a snažil se jím zachytit veškerý povrch místnosti. Při natáčení byly tedy otevřené poloskleněné dveře a dvouřadé poličky byli otevřené, takže registrační algoritmus nejspíš nebude mít mnoho bodů pro správné spárování. Při druhém nahrávání jsem se snažil tedy natočit data bez ohledu na specifikaci Kinectu, efektivitu a výkonost aplikace. Scénu jsem natáčel přibližně 11 sekund a pořídil jsem 343 snímků, s přibližnou velikostí 690 MB. Jeden snímek je tedy cca 1.9 až 2.0 MB velký.

Obrázek 7.1: První natáčená scéna

Obrázek 7.2: Druhá natáčená scéna

Snahou tedy je si při experimentování s testovacími daty ověřit mé přepoklady a zjištěné informace z uvedených zdrojů. Tedy zda je rozdíl při registraci mezi oběma způsoby natáčení, dále jestli dokáže algoritmus zaregistrovat správně části místnosti s méně objekty (holé zdi) a jaký je rozdíl mezi rychlým a pomalým výpočtem ICP. Vzhledem k omezením, které aplikace má viz 6.2, budou všechny experimenty vyhodnocovány vždy po 25 snímcích.

7.2 Ověření přesnosti aplikace

Výsledkem této práce má být nástroj, který vytváří ze snímků místnosti půdorys. Určitě stojí za ověření jak věrohodně dokáže aplikace danou místnost zpracovat a jak přesný půdorys dokáže vytvořit.

Nejjednodušším způsobem jak zjistit zda půdorys odpovídá realitě je ruční porovnání výsledku aplikace s reálným půdorysem. Reálný půdorys jsem vytvořil ručně po změření rozměrů místnosti a načrtl na papír. Dále jsem uvažoval o utilitě `compare`, která umí najít rozdíly v obrázcích. Protože ani náčrtek ani výsledný půdorys nejsou úplně přesné, tato utilita je pro vyhodnocení nepoužitelná.

Kapitola 8

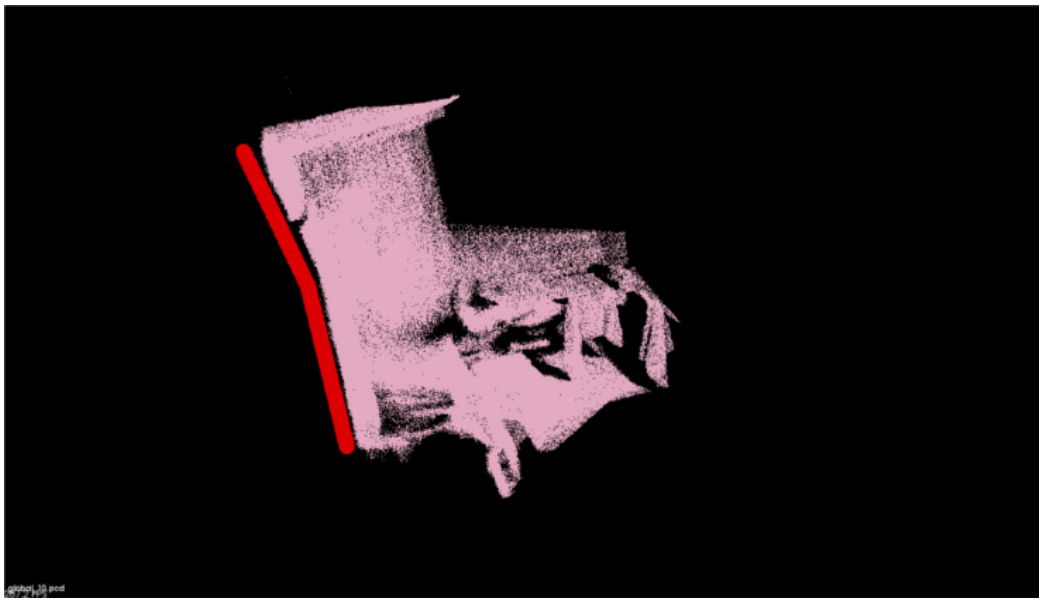
Výsledky

V této kapitole jsou výsledky a zhodnocení jednotlivých experimentů, jimiž byla aplikace podrobena.

Dále v sekci 8.3 se zmiňuji o možných vylepšeních aplikace, možnostech dalšího pokračování a zhodnotil jsem míru funkčnosti mé aplikace.

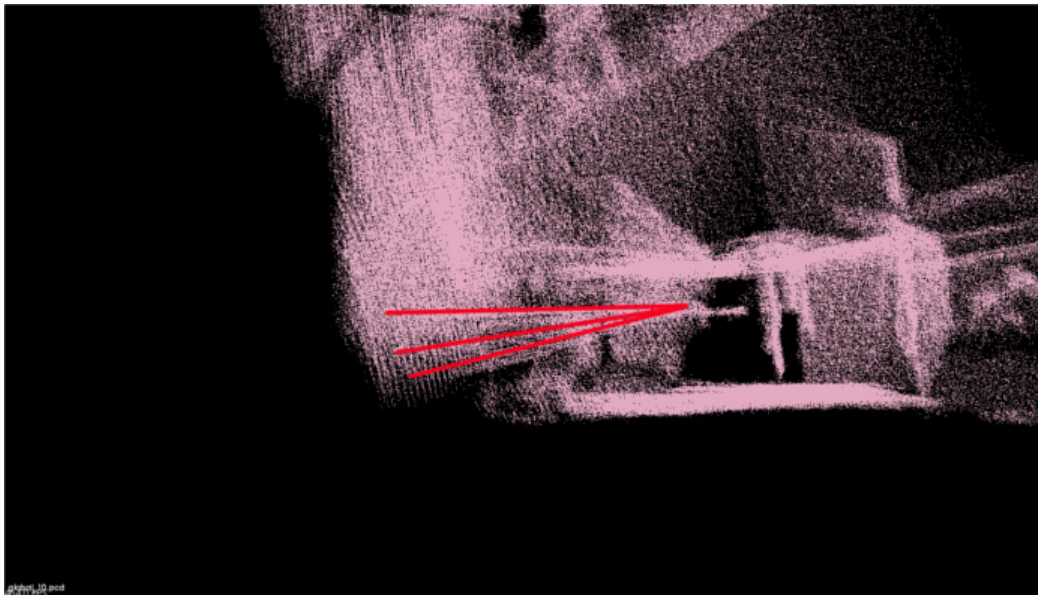
8.1 Registrace snímků

V rámci prvního experimentu jsem se zabýval zda má vliv na registraci způsob nahrávání scény. Jak jsem již zmínil zaznamenal jsem dvě místnosti jejichž specifikaci lze naléz v 7.1. Po zaregistrování jsem si zobrazil jednotlivé úseky z obou scén a zaznamenal jsem drobné odlišnosti v registraci. Při natáčení scény s Kinectem v ruce docházelo k jeho náklonu nejen ve směru horizontálním ale také vertikálním. To mělo za následek lehce nesprávného zarovnání snímků, jak lze pozorovat na obrázcích 8.1 a 8.2.



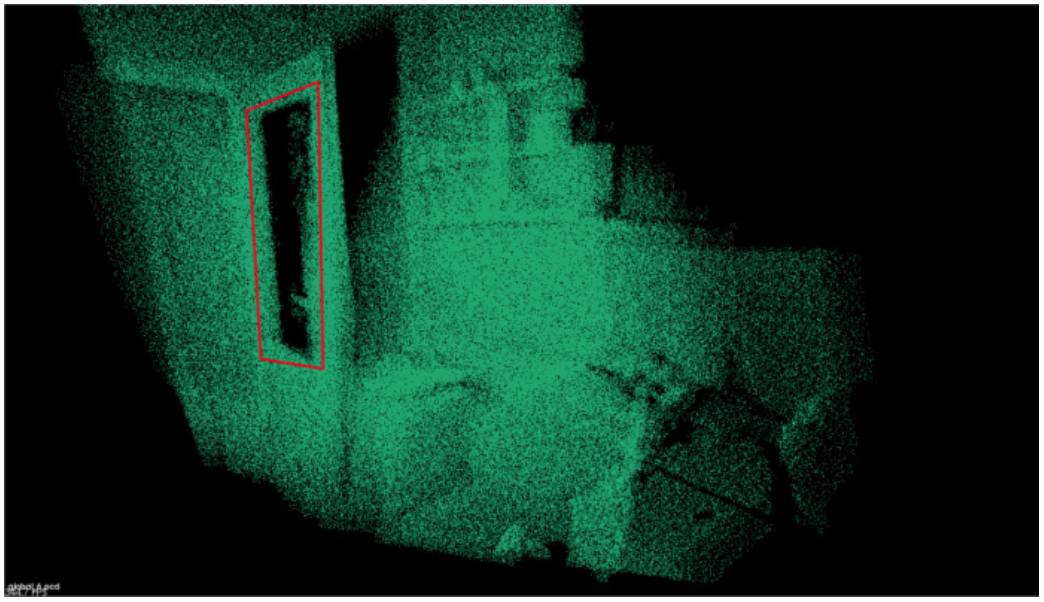
Obrázek 8.1: Ukázka nesprávně zaregistrovaných snímků. Červená čára kopíruje stěnu, která je ve skutečnosti rovná.

K takové chybě při natáčení druhé scény nedošlo. Usoudil jsem, že je tomu tak právě k vůli plynulejšímu nahrávání scény pomocí stativu. Algoritmus tak zarovnává stěnu pouze v jednom směru a zarovná ji přesněji než, když musí danou plochu otáčet ve více směrech najednou. Navíc nemusí řešit posunutí ve svislém směru.



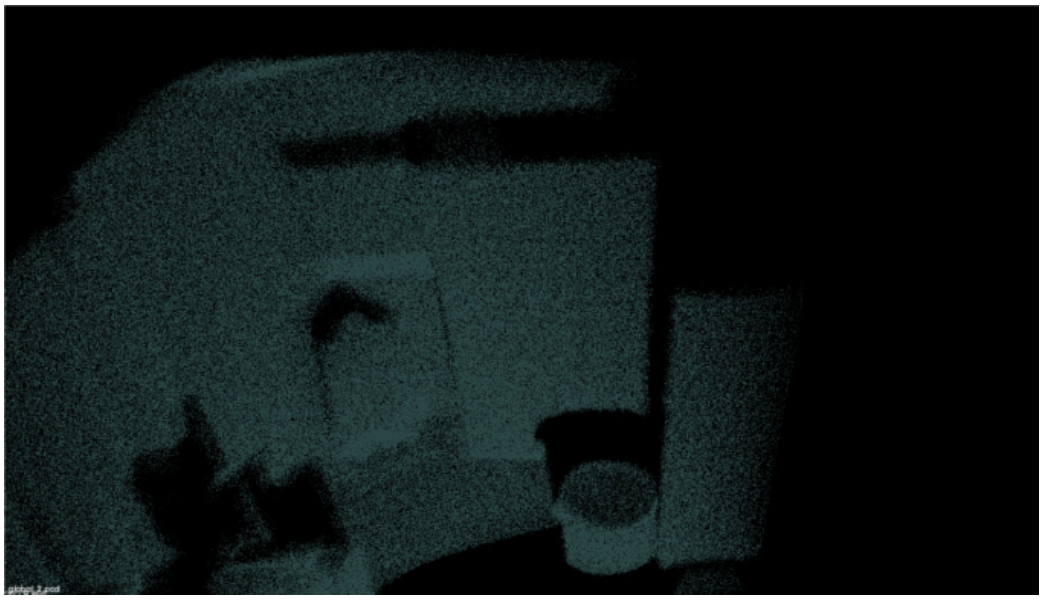
Obrázek 8.2: Ukázka nesprávně zaregistrovaných snímků. Červené úsečky označují rovinu jednotlivých snímků. Při dokonalé registraci by byla vidět pouze jedna z těhto čar.

Následně jsem si chtěl ověřit jak funguje registrace takových místností, které neobsahují vhodný materiál nebo mají málo objektů ve scéně.

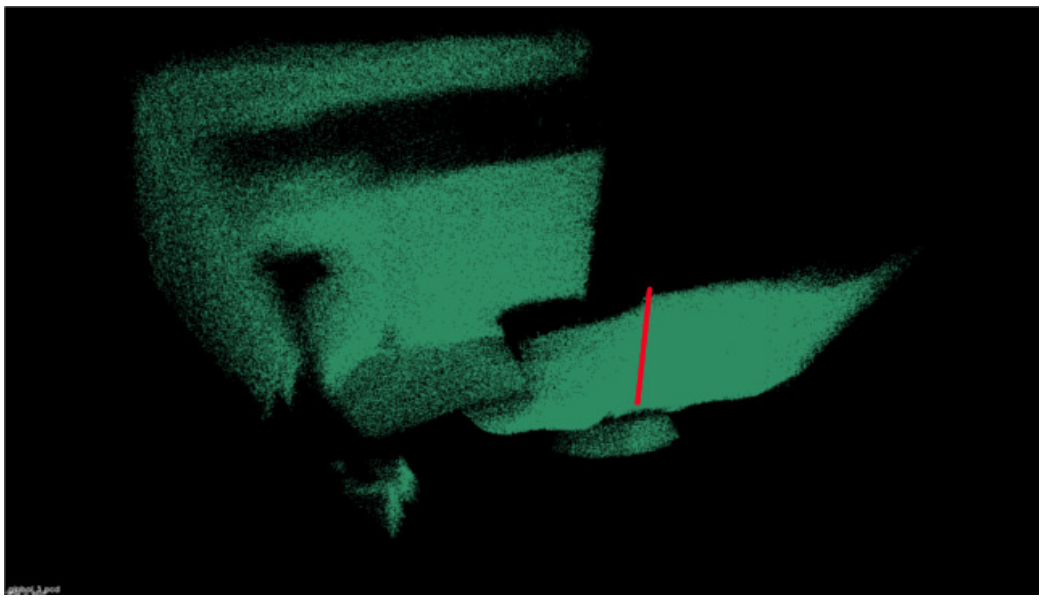


Obrázek 8.3: Jak lze vidět Kinect opravdu nedokáže zaznamenat sklo, je to způsobeno tím, že využívá infračerveného paprsku. Sklo ve dveřích je ohraničeno červenou čarou.

V první menší místnosti byly takovýmto objektem prosklené dveře viz obrázek 8.3. V druhé místnosti byla částí scény pouze holá zeď, jak dopadla registrace lze pozorovat na obrázku 8.5, pro srovnání ukáži jak vypadá scéna o pár snímků zpět viz obrázek 8.4.

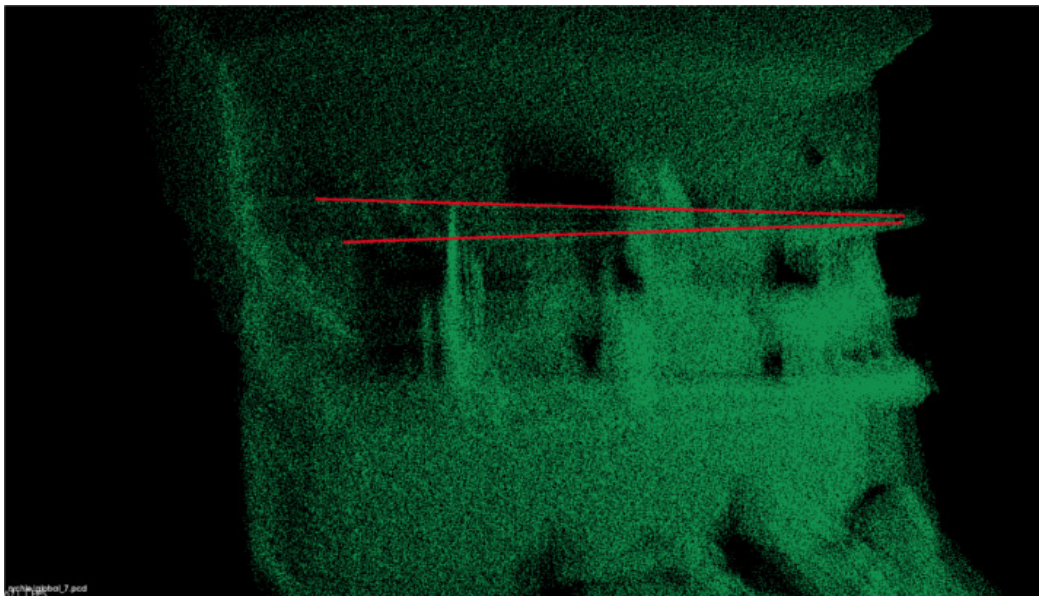


Obrázek 8.4: Takto vypadá scéna před registrováním holé zdi.



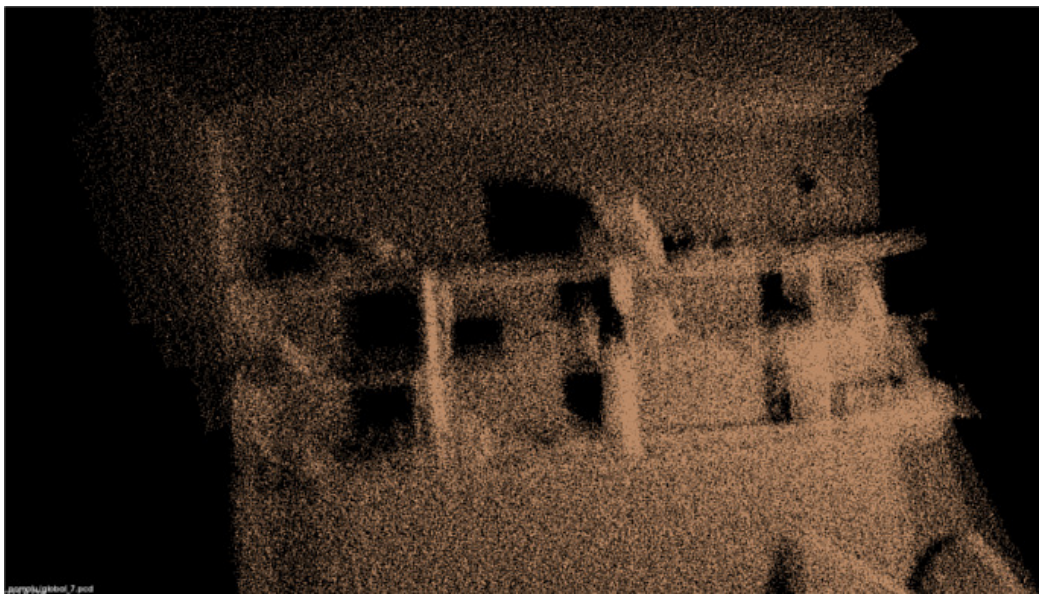
Obrázek 8.5: Jak lze vidět na obrázku holá zeď by měla začínat od červené čáry ale začíná mnohem dříve.

Z toho vyplývá, že alogritmus ICP umí rovné plochy správně natočit, problémem je potom správné posunutí, jelikož nemá dostatek význačných bodů, podle kterých by mohl dané snímky zarovnat. Určitým řešením je do scény přidat objekt, který by přidal význačné body pro ICP algoritmus, nebo takové snímky zahazovat.

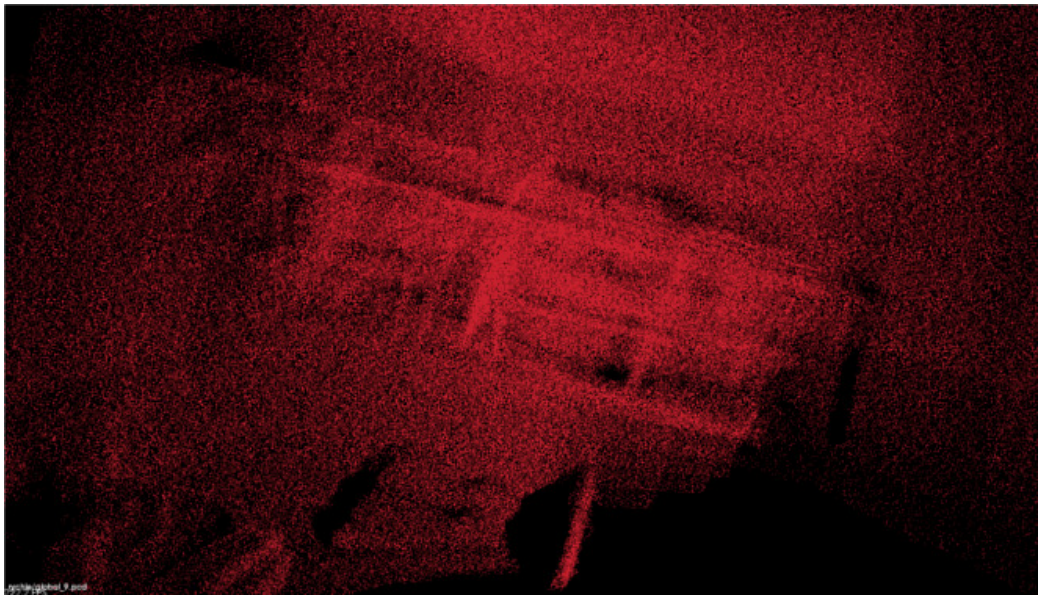


Obrázek 8.6: Červené čáry naznačují vrchní hranu poliček ze dvou snímků, jak lze vidět moc nelícují.

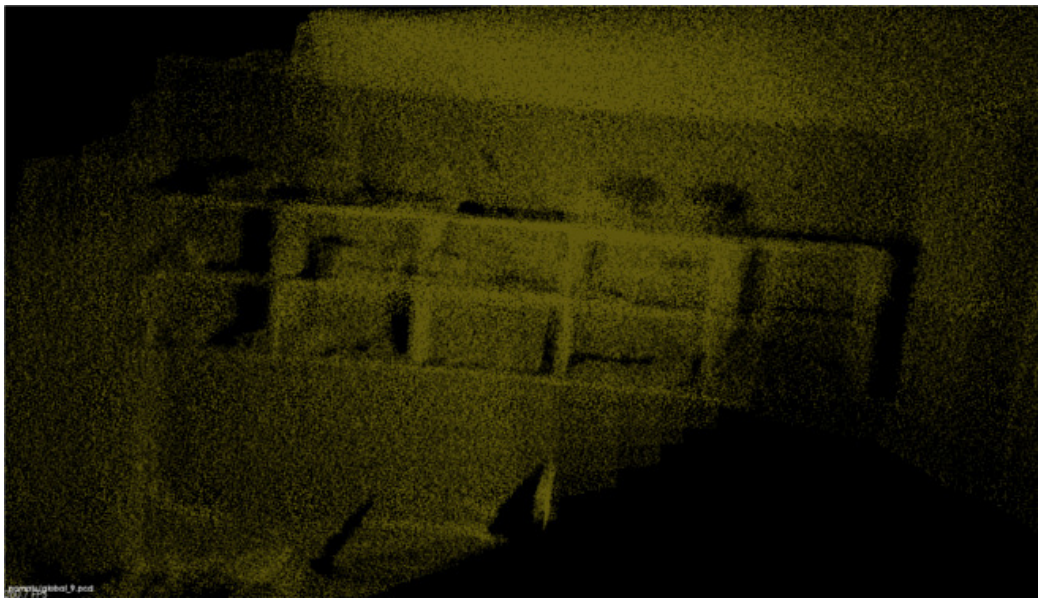
Mimo uvedené výsledky jsem také porovnával rozdíl mezi tím jak se liší rychlejší a pomalejší nastavení algoritmu ICP. Nejdříve jsem zkoumal zaregistrované snímky z druhé větší místnosti. Při tom jsem nenašel žádné výrazné rozdíly. Jinak tomu již bylo u místnosti první, kdy se určité snímky zaregistrovali lépe při pomalejším algoritmu ICP než u rychlejší varianty. Pro příklad zde uvádím obrázky 8.6 a 8.7. Kde lze vidět u horní hrany poliček nepřesné zarovnání, ještě markantnější rozdíly nalezneme při pohledu na obrázky 8.8 a 8.9. Zde se již pomalejší ale přesnější varianta vyplatí.



Obrázek 8.7: Zde vidíme, že vrchní hrana ja zarovnána u všech snímků.



Obrázek 8.8: Ukázka rychlého ICP ve složitější scéně s poličkami, zcela špatně zaregistrováno.



Obrázek 8.9: Pomalejší algoritmus si s poličkami již poradil.

Délka trvání registrace i se spojováním trvala při rychlém nastavení cca 50 sekund, zatímco pomalá varianta 960 sekund. To jsou údaje pro menší místnost, pro tu druhou větší trvala rychlá varianta 65 sekund a pomalá 978 sekund. U té se ale pomalejší varianta nevyplatí vzhledem k výsledkům. Co se týká paměťové náročnosti aplikace alokuje si tolik paměti jak jsou velké data sety. Při spojování dochází k nárustu paměti vždy o další spojovaný snímek, což je vpořádku. Problémem je, že se zvýší při každém spojování ještě o určitý počet MB paměti navíc a při každém dalším spojování je počet MB větší. To má za následek rychlé vyčerpání paměti výpočetního stroje. Při mé konfiguraci počítače vystačí pamět přibližně na 25 snímků. Nepodařilo se mi tuto skutečnost napravit ani s použitím

dealokace paměti, program i poté danou paměť neuvolnil, ikdyž se k ní již nedalo přistoupit. Pokoušel jsem se také zaregistrovat například nejříve snímky po 25 a poté zaregistrovat vzniklé soubory. Při tomto pokusu ale nedopadla registrace úspěšně. Registrovaná mračna jsou od sebe natolik pootočena a obsahují tolik bodů, že algoritmus ICP nedokáže daná mračna úspěšně zarovnat. Při správném fungování by měl program využít přibližně dvojnásobek velikosti data setu, spíše o něco více, jelikož reprezentace mračna bodů v datové struktuře knihovny PCL zabírá o něco více paměťového prostoru, než je velikost načteného souboru.

8.2 Správnost výsledků - provnání s realitou

Přesnost aplikace vzhledem k realitě, to je asi první věc, kterou by člověk chtěl znát o této aplikaci, kdyby ji chtěl použít. Porovnání probíhalo pomocí překryvu reálného náčrtku s výsledkem programu viz obrázek 8.10.

Obrázek 8.10: Ukázka půdorysu místnosti. Jednotlivé výsledky jsou pospojovány ručně. Červeně je vyznačen výsledek aplikace, černě můj náčrtek. Náčrtek není úplně přesný slouží jen pro přibližné porovnání. Jak lze vidět vlevém horním rohu, zde nedopadla nejlépe registrace, a proto také nalezené přímky neodpovídají náčrtku.

Jak lze vidět na obrázcích shoda není dokonalá. Podobný výsledek byl očekáván, jelikož v každé fázi této aplikace dochází ke kumulování různých chyb, aproximacím a k zaokrouhlování. Nicméně částečně to reálnou místnost napodobuje.

V rámci vyhodnocení jsem vyčíslil přibližnou odchylku aplikace od reálného náčrtku cm. Z obrázku 8.10 jsem měřil postupně vzdálenosti mezi chybnými body a body mého náčrtku v pixelech. Následně jsem v měřítku přepočítal tyto hodnoty na cm a vyčíslil průměrnou odchylku $21.269cm$ naměřené hodnoty jsou v tabulce 8.2. Dále jsem vyčíslil chybu v úhlech. Pokud se program odchýlil od určitého úhlu, tak s průměrnou chybou 16° viz tabulka 8.1.

Chyba [°]
20°
20°
18°
33°
10°
15°
4°
10°
Průměrná chyba ve st.
16°

Tabulka 8.1: Naměřené chyby v úhlech a vyčíslení průměrné chyby.

Důležité také je jak se daná scéna natočí. Kinect z jednoho úhlu nedokáže zachytit všechen povrch v místnosti, a proto mohou chybět určitá důležitá data pro zrekonstruování půdorysu. Naopak některá data mohou přebývat. Představíme-li si stůl, který je podpírán čtyřmi nohami, Kinect nasnímá jak nohy tak povrch stolu. Ty budou ve výsledném půdorysu vidět, kdežto povrch a hrany stolu nejspíše ne. Je tomu tak proto, že povrch stolu bude ve svislém směru reprezentovat malé množství bodů. Nohy stolu naopak více a je pravděpodobné, že se vyskytnou ve výsledku.

8.3 Shrnutí

Aplikace funguje docela dobře na datech (přiložena na DVD), která jsou pořízena rozumným způsobem. Tím se myslí například na stativu nebo podobném zařízení. Důležité je, aby nahrávání bylo plynulé a ne chaotické. V opačném případě může dojít k nesprávné registraci a následně k zcela neodpovídajícímu výsledku, jelikož se budou chyby propagovat dále. V rámci implementace lze ještě doladit paměťovou náročnost. Na mém počítači zvládne program zpracovat kompletně cca 25-30 snímků, pokud by měl dostatek paměti tak i více. Tento problém se mi nepodařilo odstranit.

Výsledky aplikace nejsou ideální, proto by bylo vhodné v dalším pokračování se zabývat možností jak jednotlivé čáry sloučit do jedné a následně řešit návaznost jednotlivých čar na sebe. Dalším pokračováním, by mohla být určitá grafická nadstavba s přívětivým uživatelským rozhraním. Program by mohl umět například ze zaregistrované místnosti vytvořit její model. Mohl by obsahovat databázi objektů nasnímaných Kinectem a další zajímavé věci.

Vzdálenost $[px]$	Vzdálenost $[cm]$	Vzdálenost $[px]$	Vzdálenost $[cm]$
12	9.02	21	15.79
12	9.02	20	15.14
12	9.02	32	24.06
12	9.02	29	21.80
12	9.02	18	13.53
12	9.02	32	24.06
65	48.87	26	19.55
45	33.83	31	23.31
78	58.65	15	11.28
10	7.52	57	42.86
11	8.22	41	30.83
9	6.77	46	34.59
5	3.76	14	10.53
43	32.33	9	6.77
43	32.33	19	14.29
48	36.09	19	14.29
13	9.77	13	9.77
18	13.53	19	14.29
9	6.77	17	12.78
10	7.52	11	8.27
81	60.90	22	16.54
69	51.88	40	30.08
58	43.61	43	32.33
36	27.07	50	37.59
56	42.11	43	32.33
36	27.07	16	12.03
23	17.29	27	20.30
39	29.32	61	45.86
71	53.38	75	56.39
4	3.01	4	3.01
5	3.76	6	4.51
9	6.77	8	6.02
9	6.77	8	6.02
Průměrná chyba $[px]$		Průměrná chyba $[cm]$	
28.29		21.27	

Tabulka 8.2: Vzdálenosti v pixelech a centimetrech mezi reálným náčrtkem a chybně zakreslenými čarami.

Kapitola 9

Závěr

Tato bakalářská práce se zabývala rekonstrukcí scény pomocí senzoru Microsoft Kinect a detekcí čar v obraze. Cílem práce bylo vytvořit nástroj pro tvorbu půdorysu vnitřních prostor ve formě vektorového popisu. Byla implementována aplikace využívající metody Iterative Closest Point (ICP) k registraci jednotlivých snímků. Řeší projekci mračna bodů do roviny s následujícím post–procesingem obrázku. S pomocí Cannyho detektoru hran a Houghovy transformace hledá čáry v obraze a ukládá výsledný obrázek.

Implementovaný algoritmus ICP vykazuje obstojné výsledky v rozumně nahrané scéně. Mezi takové patří zejména scény s dostatečným počtem objektů, interiéry a málo prosklené scény, při použití přesnější varianty i v náročnějších scénách.

Použití méně přesné varianty se nedá doporučit v náročnějších scénách, jelikož si algoritmus ICP neporadí s registrací. Poté i výsledný půdorys nedává uspokojivé výsledky. Program byl otestován na dvou datových sadách z nichž jedna je přiložena na DVD v příloze. Porovnání probíhalo ručně přiložením výsledku na reálný náčrtek scény.

Kapitola 10

Seznam příloh

10.1 Obsah CD

10.2 Manuál

10.3 Plakát

Literatura

[1] Adnan: What is ICP: Iterative Closest Point? [online]. http://www.wisegai.com/2012/11/07/what-is-icp-iterative-closest-point/, 2012-11-07 [cit. 2013-04-24].

[2] Borrmann, D.; Elseberg, J.; Lingemann, K.; aj.: The Efficient Extension of Globally Consistent Scan Matching to 6 DoF. Technická zpráva, University of Osnabrück, 1997.

[3] Library, P. C.: PCL - Point Cloud Library (PCL) [online]. http://www.pointclouds.org/, 2013 [cit. 2013-04-24].

[4] Lu, F.; Milios, E.: Globally Consistent Range Scan Alignment for Environment Mapping. Technická zpráva, Department of Computer Science, York University, 1997.

[5] Magnusson, M.: *The Three-Dimensional Normal-Distributions Transform - an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* Dizertační práce, Örebro University, 2009.

[6] Microsoft: Kinect for Windows Sensor [online]. http://msdn.microsoft.com/en-us/library/hh855355.aspx, 2012-11-07 [cit. 2013-04-24].

[7] OpenCV: OpenCV — OpenCV [online]. http://www.opencv.org/, 2013 [cit. 2013-04-24].

[8] OpenNI: Hough Line Transform - OpenCV 2.4.5.0 documentation [online]. http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html#h 2013 [cit. 2013-04-24].

[9] OpenNI: Hough transform - Wikipedia, the free encyclopedia [online]. http://en.wikipedia.org/wiki/Hough_transform, 2013 [cit. 2013-04-24].

[10] Pereira, C.: The Ten Best Unintended Uses for Kinect [online]. http://www.1up.com/news/ten-best-unintended-uses-for-kinect, 2010-12-29 [cit. 2013-04-24].

[11] Segal, A.; Haehnel, D.; Thrun, S.: Generalized-ICP. Technická zpráva, Stanford University, 2009.

[12] Silbert, S.: MIT's real-time indoor mapping system uses Kinect, lasers to aid rescue workers [online]. http://www.engadget.com/2012/09/25/mit-realtime-indoor-mapping-kinect/, 2012-09-25 [cit. 2013-04-24].

[13] Wikipedia: Kinect - Wikipedia, the free encyclopedia [online].
`http://http://en.wikipedia.org/wiki/Kinect`, 2013-03-15 [cit. 2013-04-24].

Příloha A

Obsah CD

- Zdrojové kódy aplikace.
- Demonstrační data
- Elektronická verze této zprávy
- Plakát

Příloha B

Manuál

Pro funkčnost aplikace je nutné, aby byla nainstalována knihovna PCL ve verzi `trunk`. Dále knihovna OpenCV ve verzi 2.4.5 nebo novější. Instalace knihoven je popsána na webu viz literatura.

Pro zkompilování všech zdrojových souborů je připraven skript `install.sh`. Stačí zkopírovat složku `zdroj_kody` do počítače a spustit daný skript.

Poté je možné pomocí skriptu `run.sh` spustit aplikaci. Skript očekává následující povinné parametry:

1. -d (cesta ke vstupním souborům PCD)
2. -r (rozlišení:1024,2048,4096)
3. -o (složka pro výstup)
4. -s (varianta ICP algoritmu:slow,fast)

Příklad použití: ./run -d TESTOVACI_SADA_1 -r 1024 -o VYSLEDKY -s slow

Příloha C

Plakát