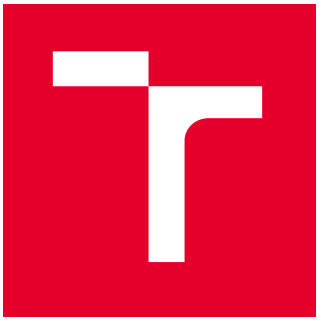


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

FILTRACE DAT Z LIDAR PRO NAVIGACI MOBILNÍHO ROBOTU VE VNĚJŠÍM PROSTŘEDÍ

LIDAR DATA FILTERING FOR MOBILE ROBOT NAVIGATION IN THE OUTDOOR ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tomáš Janech**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Gábrlík, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Tomáš Janech ***ID:*** 230085

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Filtrace dat z LiDAR pro navigaci mobilního robotu ve vnějším prostředí

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je filtrace dat z laserového skeneru (LiDAR) z okolí mobilního robotu pohybujícího se ve vnějším prostředí za účelem navigace robotu po sjízdném terénu.

1. Proveďte průzkum softwarových nástrojů sloužící k práci s daty z LiDAR a jejich filtraci; seznamte se s frameworkem ROS a principem funkce LiDAR.
2. Za pomocí zvoleného nástroje demonstrujte na poskytnutém datasetu základní operace nad laserovými daty, např. ořez a podvzorkování.
3. Implementujte či zprovozněte dostupný algoritmus pro projekci laserových dat do obrazu z kamery.
4. Realizujte filtraci mračna bodů do jednotlivých tříd na základě klasifikovaných obrazových dat.
5. Připravte softwarové řešení tak, aby filtrace probíhala na datech v reálném čase a proveďte testování dle pokynů vedoucího.

DOPORUČENÁ LITERATURA:

MURPHY, Robin R. Introduction to AI Robotics. Massachusetts: MIT Press, 2002. ISBN 9780262133838.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 22.5.2023

Vedoucí práce: Ing. Petr Gábrlík, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Pre využitie mobilných robotov v exteriéri sa vyžaduje, aby bol robot schopný rozpoznať prejazdnosť daného terénu, či sa jedná o členitosť alebo povrch. Táto práca sa zaoberá metódami rozpoznania a filtrácie povrchov na základe obrazovej segmetácie. V práci sú popísané vybrané metódy projekcie 3D bodov a algoritmy ich filtrácie z PointCloudu, ktorým je na záver porovnaná účinnosť.

KĽÚČOVÉ SLOVÁ

priestupnosť terénu, ROS2, PointCloud, segmentácia, mobilná robotika, LiDAR

ABSTRACT

To use robots outdoors it is required for the robot to be aware of the terrain traversability, be it complexity or the surface material itself. This thesis goes in depth about detection and filtration methods for surfaces, based on semantic image segmentation. Thesis also discusses some methods of 3D point projection, PointCloud filtration algorithms and their effectiveness.

KEYWORDS

terrain traversability, ROS2, PointCloud, segmentation, mobile robotics, LiDAR

JANECH, Tomáš. *Filtrace dat z LiDAR pro navigaci mobilního robotu ve vnějším prostředí*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 48 s. Bakalárska práca. Vedúci práce: Ing. Petr Gábrlík, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Tomáš Janech
VUT ID autora:	230085
Typ práce:	Bakalárska práca
Akademický rok:	2022/23
Téma závěrečnej práce:	Filtrace dat z LiDAR pro navigaci mobilního robotu ve vnějším prostředí

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu semestrálnej práce pánovi Ing.Petrovi Gábrlíkovi, Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Filtrácia bodov v blízkom okolí robota je nevyhnutnou súčasťou autonómnej robotiky, existuje niekoľko prístupov ako pri tejto filtrácii postupovať, v tejto práci vybrané prístupy teoreticky preberiem a následne aj implmentujem. Táto práca sa teda predovšetkým venuje spracovaniu dát z LiDAR snímačov pre využitie v autonómnej navigácií.

Cieľom práce je zoznámiť sa z možnosťami filtrácie a klasifikácie Point Cloudových dát, navrhnutie algoritmu na implementáciu elementárnych a pokročilých filtračných operácií na zadaných dátach. Práca sa tak rozvíja na určenie prejazdnosti terénu pre danú robotickú platformu na základe členitosti terénu a druhu povrchu. Druh povrchu bude určený pomocou segmentovaného obrazu z RGB kamery a predaný do algoritmu. Výstupom algoritmu budú filtrované PointCloudové dáta, na základe nastavenia uživateľa.

V práci budú popísané a porovnané vybrané metódy filtrácie dát a projekcie týchto dát na obraz kamery. Z týchto metód bude následne vybraný algoritmus na finálnu klasifikáciu a fitráciu dát. Tieto algorimty budú testované na predom nahraných dátach a výsledný algoritmus aj na robotickej platforme.

1 Teoretický úvod

1.1 Laserové snímače polohy

Laserové snímače polohy fungujú na princípe merania dĺžky doby letu (ToF). Tieto zariadenia sa bežne označujú ako LiDAR. Toto zariadenie sa skladá z monochromatického laseru, ktorého lúč je rozmetaný do priestoru. Tento lúč sa odrazí od meraného objektu a určitá časť sa vracia naspäť do LiDAR-u. Na základe doby od vyslania do prijatia lúča dokážeme vypočítať vzdialenosť meraného objektu od snímača. LiDAR-y môžeme v princípe rozdeliť na:

- Jendoosé (1D)
- Plošné (2D)
- Priestorové (3D)

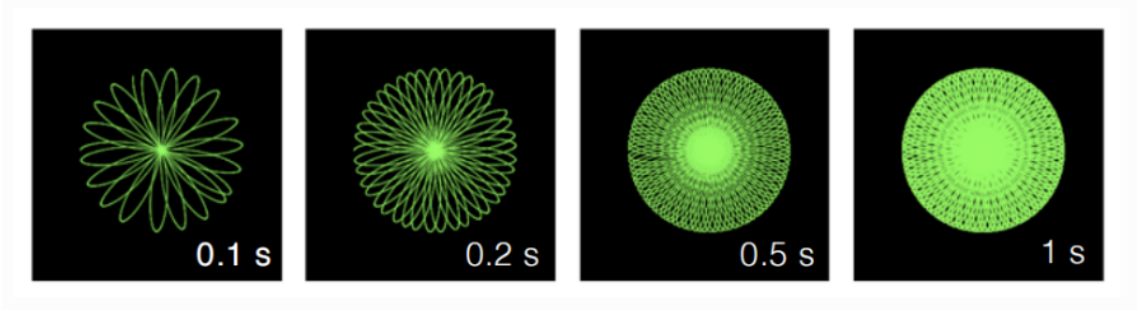
Jednoosé LiDAR snímče ako názov napovedá merajú iba v jednej osi, používajú sa ako detektory prítomnosti alebo merače vzdialenosti.

Dvojosé LiDARy merajú v jednej rovine, táto rovina väčšinou býva vodorovná a používajú sa ako bezpečnostné prvky v priemyslených robotických aplikáciach ako napríklad AGV. Jedna z najznámejších firiem, ktorá vyrába tieto Lidary je SICK.

Priesotrové LiDARy sa používajú pri mapovaní alebo navigácii robotov a môžeme ich opäť rozdeliť na základe ich zorného poľa.

LiDAR s 360° zorným poľom rozmetá laserový lúč v priestore po kružnici, resp. viacerých kružniciach. Toto je spravidla dosiahnuté nakloneným rotujúcim zrkadlom alebo zrkadlami. Výsledný Point Cloud má na rovnej ploche tvar sústredných kružníc so stredom v mieste merania. Tento druh LiDAR-ov sa bežne vyrába vo viac kanálovom prevedení, kedy sa použije niekoľko zdrojov svetla (laserov) a rovnaký počet detektorov. Vzdialenosti sa spravidla merajú na všetkých kanáloch súčasne, z čoho vyplýva, že vždy snímame celý vertikálny rozsah snímača.

LiDAR so zorným poľom menším ako 180°, konkrétne LiDARy LIVOX rozmetajú laserový lúč v tvare číslice 8, táto číslica sa s každou periódou posúva o geometriou definovaný uhol φ, po určitej dobe, ktorú nazývame dobu integrácie T, dokáže tento spôsob úplne pokryť kruhovú plochu. Tento spôsob rozmetávania sa dosahuje použitím Risley-ho hranolu. Tieto hranoly poskytujú rýchle 2D skenovanie v pomerne kompaktnej konštrukcii. Bežne sa skladajú z dvoch valcových rotujúcich klinov, ktoré lámu lúč pri jeho prechode medzi prostrediami [1]. Príklad tohoto rozmetania sa nachádza na Obr. 1.1.



Obr. 1.1: Príklad rozmetania laserového lúča s použitím Risley-ho hranolu pre vybrané doby integrácie T [3]

1.2 Operačný systém robotov ROS

ROS je open-source súbor balíkov často používaný v robotike. Naproti tomu ako názov naznačuje, nejedná sa o operačný systém, pretože nepodporuje podstatné súčasti moderných operačných systémov ako napríklad plánovanie programových požiadaviek. Tento framework avšak umožňuje jednoduché vytvorenie a vzájomnú komunikáciu modulov (node-ov) robotu ako napríklad, riadenie podvozku, zber dát zo snímačov, plánovanie trajektórie a mnohé ďalšie. ROS beží ako služba na už existujúcom operačnom systéme, v robotike často používaný Linux, konkrétne distribúcia Ubuntu. Toto avšak neobmedzuje použitie ROSu iba na Linux, samotný autori poskytujú aj verziu pre Windows alebo MacOs, ktoré sa však neodporúčajú na použitie v robotike.

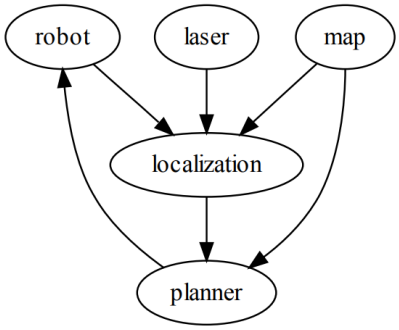
ROS sa v oblasti robotiky uplatnil hlavne vďaka jeho jednoduchosti, otvorenému kódu a podpore viacerých programovacých jazykov [5] čo v praxi znamená, že si programátori robotov môžu vybrať, ktorý im na danú aplikáciu vyhovuje najviac, prípadne v jednom projekte použiť niekoľko jazykov súčasne.

1.2.1 Architektúra ROS

ROS Uzly

Pri použití ROSu môžeme jednotlivé aplikácie rozdeliť do balíkov (*Package*), ktoré už na programovej úrovni obsahujú uzly (*Node*). Tieto uzly vykonávajú určitú funkciu a môžeme ich považovať za čierne skrinky, ktoré spravidla majú vstupy a výstupy, ktoré slúžia na komunikáciu s inými uzlami alebo vizualizáciu.

Na Obr. 1.2 môžeme vidieť jednoduchý príklad použitia ROSu na plánovanie trasy mobilného robotu. V tomto príklade sa pomocou laserového snímača, mapy a samotného robota určí aktuálna poloha v priestore. Za pomoci tejto lokalizácie a mapy sa vypočíta trasa, ktorú by mal robot nasledovať. Výhodou tohto prístupu je



Obr. 1.2: Príklad ROS komunikácie medzi uzlami [5]

abstrakcia jednotlivých úrovní, kedy nemusíme vedieť ako jednotlivé uzly získavajú a spracovávajú dáta. Taktiež sa môže jeden uzol použiť na niekoľko úkonov bez potreby použitia opakujúceho sa kódu.

Komunikácia medzi uzlami

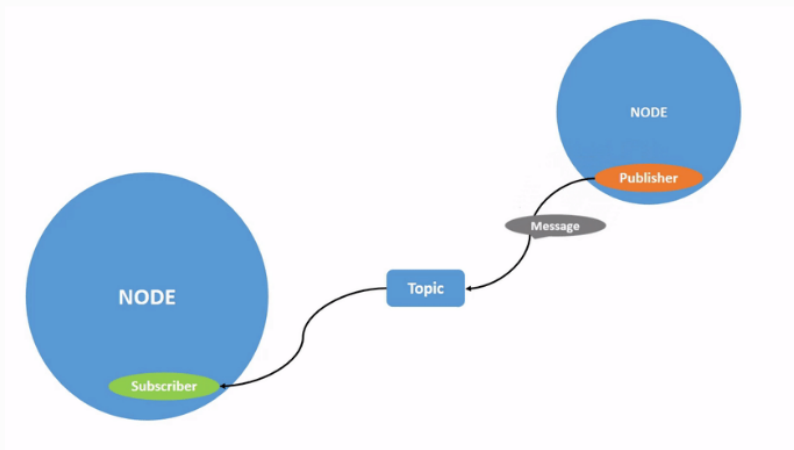
ROS zabezpečuje komunikáciu medzi jednotlivými uzlami prostredníctvom troch typov komunikácie

- Topic
- Služby (services)
- Akcie (actions)

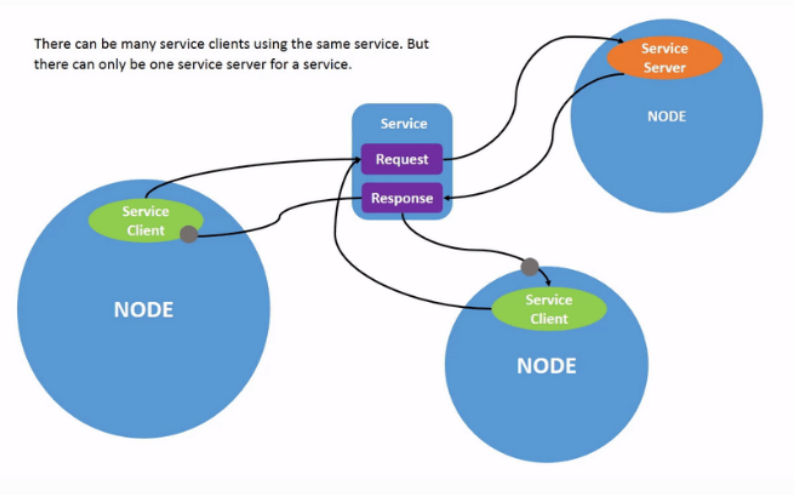
Topic je najjednoduchším spôsobom predávania dát medzi uzlami, ako informácia sa šíria takzvané správy (messages), ktoré majú definovaný typ a obsah. Ich prijatie ani spracovávanie nie je oznamované inými výstupmi. Správy sa väčšinou šíria z jedného zdroja (*Publisher*) do jedného alebo viacerých cieľov (*Subscriber*). ROS avšak dovoľuje aj šírenie z viacerých zdrojov. Táto komunikácia je znázornená na Obr. 1.3.

Služby sú spôsob komunikácie, kde si môže jeden uzol zažiadať o dáta od druhého. Uzly sa delia na servery, ktoré dáta vydávajú a klientov, ktorí o dáta žiadajú. Server nesmie bez žiadosti od klienta poskytnúť žiadne dáta. Na jednom spojení môže existovať niekoľko klientov, ale vždy iba jeden server, v takomto prípade sú dáta zaslané iba tomu klientovi, ktorý o ne zažiadal. Príklad komunikácie je vyobrazený na Obr. 1.4.

Akcie slúžia na komunikáciu pri vykonávaní dlhšie trvajúcich úloh. Skladajú sa z cieľu, spätnej väzby a výsledku. Cieľ a výsledok je komunikácia prostredníctvom služby. Spätná väzba je topic, ktorý sa odosiela po dobu vykonávania operácie. Akcie sa narozdiel od služieb dajú zrušiť, čím prestanú vykonávať svoju činnosť. Po



Obr. 1.3: Komunikácia medzi dvomi uzlami za použitia topic. [11]



Obr. 1.4: Komunikácia medzi viacerými uzlami za použitia služby. [11]

úspešnom dokončení sa pošle správa s výsledkom. Model komunikácie sa skladá z jedného topicu a dvoch služieb - dvoch serverov, dvoch klientov.

1.2.2 Typy správ pre ROS

ROS pri komunikácii medzi dvomi uzlami využíva takzvané správy. Každá správa má presne definovaný typ a dáta ktoré prenáša. ROS poskytuje preddefinované typy správ, ktoré sa bežne používajú pri programovaní robotov. Medzi základné typy správ, ktoré budem v tejto práci používať patria:

- tf
- Image
- CameraInfo
- PointCloud2

tf

tf alebo TransformFunction, je druh správy ktorý umožňuje výpočet bázy pre senzory, aby sa následne dáta z viacerých senzorov zhodovali, aj pokiaľ sa nezhoduje ich báza. Nachádzajú sa v kategórií správ `geometry_msgs`. Tieto transformácie majú rôzne druhy ako napr. translácie alebo rotácie preto nemôžeme presne definovať ich dátovú štruktúru. V praxi sa používajú stacionárne a dynamické transformácie. Dynamické transformácie z pohybov robotu menia svoje parametre a posielajú sa ako správy. Statické dokážeme poslať ako správy alebo parametre, ktoré sa pošlú iba raz pri inicializácii robota.

Image

Druh správy Image sa používa na posielanie obrazových dát napr. z kamery alebo iných zariadení, ktorých výstupom sú obrazové dáta. Typ správy sa nachádza v kategórií `sensor_msgs`. Správa obsahuje hlavičku, ktorá informuje o čase kedy bola vytvorená a iné pomocné informácie. V tejto správe taktiež nájdeme rozmer prenášaného obrázku. Správa taktiež obsahuje položku `encoding`, ktorá udáva ako sú informácie uložené, poradie kanálov, kódovanie jednotlivých pixelov atď.. Nakoniec správa obsahuje dáta, ktoré sú reprezentované ako pole 8-bitových čísiel. Pokiaľ má pixel typ väčší ako 8-bitov na kanál, pixel sa poskladá z viacerých prvkov poľa dát, pri tomto spájaní treba skonrolovať parameter `is_bigendian`, čo nám udáva druh kódovania, big alebo litte endian.

CameraInfo

Správa typu CameraInfo sa používa na posielanie kalibračných údajov kamery. Typ správy je definovaný v kategórií `sensor_msgs`. Správa ako vždy obshauje hlavičku, ktorej časť `frame_id` musí súhlasiť so správou typu Image. V tejto správe ďalej nájdeme matice projekcie kamery ako aj parametre skreslenia kamery. Táto správa sa bežne generuje raz pre plnorozmerový snímač a následne sa využíva pole `roi`, ktoré dokáže rozlíšnie kamery a teda aj projekčné parametre meniť za chodu programu, bez potreby opätovnej kalibrácie.

PointCloud2

Správa typu PointCloud2 sa používa na poslanie Point Cloudov, taktiež ako správu Image ju nájdeme v kategórií `sensor_msgs`. Štruktúra je podobná ako pre správu Image, s rozdielom, že nemáme položku `encoding`, ale položku `fields`, ktorá definuje aké dáta sa nachádzajú v dátovom poli, napr. súradnice x, y, z, intenzita a čas zaznamenania bodu.

1.2.3 Používané balíky ROS

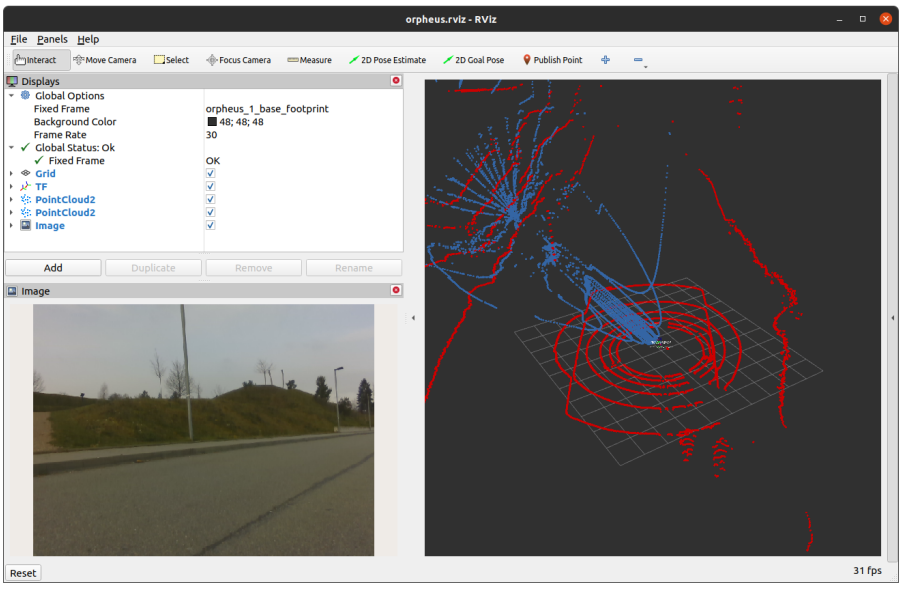
ROS obsahuje množstvo predprogramovaných balíkov, ktoré slúžia k zjednodušeniu práce na robotoch. Niektoré z najpoužívanejších balíkov si rozoberieme v tejto časti. Balík `perception_pcl` bude prebraný v časti 1.3 ako samostatný balík, ktorý nie je závislý na ROS.

ros bag

Balík *ros bag* nám umožňuje nahrať špecifikovanú komunikáciu (topic) do súboru, ktorý môžeme použiť na testovanie algoritmov bez potreby testovania na reálnom hardwari. Po spustení sa tento záznam správa ako skutočná komunikácia v robote s rozdielom neaktuálnosti časových hlavičiek správ, kedže tie sa nahrávajú taktiež. Od ROS verzie Humble sa pri prehrávaní dá použiť pozastavenie, zrýchlenie a spomalenie prehrávania.

RViz

Balík RViz slúži na vizualizáciu. Najčastejšie používané typy správ sú Image, PointCloud2 a tf. RViz podporuje vizualizáciu všetkých typov správ podporovaných ROSom. Po spustení tohto programu sa otvorí okno, ktorého rozloženie sa dá upraviť podľa potrieb programátora. Toto okno je vyobrazené na Obr. 1.5.



Obr. 1.5: Program RViz zobrazujúci 2 Point Cloudy(vpravo) a video(vľavo dole)

RQt

Balík RQt slúži ako grafické rozhranie ROSu, prostredie je nastaviteľné ako v RViz, avšak oproti RVizu dokáže vytvárať grafy alebo zobraziť prepojenie uzlov. Tento balík funguje na princípe pluginov, ktoré sa stále vyvíjajú, prípadne si môže užívateľ napísať vlastné.

1.3 Knižnica na spracovanie Point Cloud dát PCL

Knižnica PCL sa používa pre manipuláciu n-rozmerných Point Cloudov. Táto knižnica bola vyvinutá pre jazyk C++ a poskytuje jednoduchú a dobre optimalizovanú manipuláciu s Point Cloudmi [7]. PCL poskytuje niekoľko častých operácií nad dátami a to je filtrácia, extrakcia tvarov, registrácia Point Cloudov alebo najpoužívanejšie, nájdenie n susedov v okolí bodu. Táto knižnica je súčasťou distribúcie ROS s potrebou dodatočnej inštalácie.

Táto knižnica používa dátový typ PCD (Point Cloud Data), ktorý je považovaný za najefektívnejší spôsob ukladania Point Cloud dát, avšak na prácu v ROS potrebujeme dátový typ PointCloud2. Medzi týmito typmi existuje konverzia, ktorá je súčasťou balíku `pcl_conversions` z tejto knižnice.

1.3.1 Filtre PCL

PCL pozná niekoľko druhov filtrov, v tejto kapitole vyberieme tie najzákladnejšie a popíšeme ich. Medzi základné filtre patrí:

- Pásmová prepusť/zádrž
- Gaussov filter
- 3D Boundary Box
- Lokále extrémy

Pásmová prepusť

Pásmová prepusť je najjednoduchší typ filtra, ktorý ponechá len body ktoré spĺňajú definovanú podmienku, ako napr. hodnotu Z súradnice (výšku). Tento filter sa v knižnici PCL dá invertovať, čím sa z neho stáva filter pásmovej zádrže, ktorý dané body vymaže.

Gassov filter

Gaussov filter slúži na odfiltrovanie šumu v Point Cloudových dátach. Šum v dátach môže vzniknúť po ich manipulácií alebo častejšie zo snímača, ktorý používame.

3D Boudary Box

Boundary Box funguje na rovnakom princípe ako Pásmová prepusť s tým rozdielom, že sa dá použiť pre 3D, čiže odfiltruje všetky body v alebo mimo daného telesa v 3D priestore.

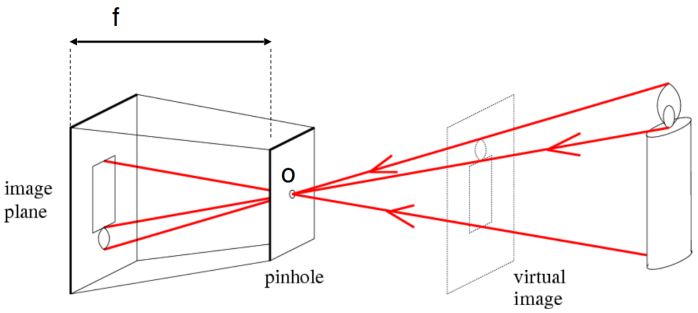
Lokálne extrémy

Filter v definovanom priestore odfiltruje body s najnižšou alebo najvyššou hľadanou hodnotou, môže sa jednať o súradnicu Z alebo napríklad intenzitu. Tento filter sa používa na zníženie počtu bodov v Point Cloude s tým, že zachováva množstvo informácie.

1.4 Geometria kamier

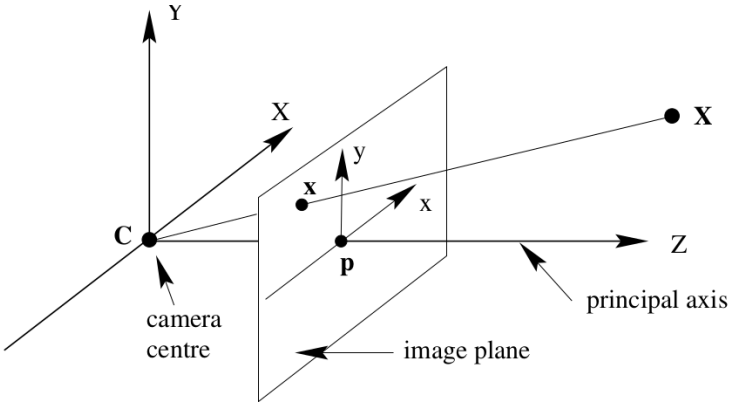
Kamera je vo všeobecnosti zariadenie, ktoré prevádza informácie z 3D priestoru na 2D snímač. Pri tomto prevode dochádza k strate hĺbkovej informácie, túto informáciu dokážeme pri projekcí z 2D do 3D čiastočne navrátiť. V tejto práci sa venujem geometrií jedno okulárovej kamery alebo modelu direkovej komory (pinhole camera). Pri odvodení budem uvažovať ideálnu kameru s nekonečne malým ohniskom a bez skreslenia. Neskôr, pri kalibrácií kamery bude popísané a uvažované aj skreslenie objektívu.

Model bežnej kamery si môžeme predstaviť ako dierkovú komoru, kde sa o-hnisko nachádza pred aktívnou plochou (snímačom) a obraz dopadajúci na snímač je obrátený okolo osi x. Tento model je Obr. 1.6.



Obr. 1.6: Model dierkovej komory [8]

Za účelmi projekcie si môžeme tento model upraviť na jednoduchší, kde sa snímač nachádza ešte pred ohniskom, týmto nedochádza k otáčaniu obrazu a ohinsko môžeme považovať za počiatok kamery. Tento zjednodušený model je Obr. 1.7.



Obr. 1.7: Zjednodušený model kamery [2]

Transformácie

Transformácie ktoré nastávaju pri projekcii sú popísané niekoľkými maticami, najdôležitejšou z nich je intristická alebo vnútorná matica. Ako názov napovedá jedná sa o transformácie ktoré súvisia s vnútornou geometriou kamery. Táto matica sa skladá z troch základných transformácií, 2D translácia, 2D zkosenie a 2D rotácia [9],

$$\mathbf{K} = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & s/f_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.1}$$

Na základe obrázku 1.8 môžeme odvodiť časť tejto matice, konkrétne 2D transláciu. Možeme vidieť že parametre x_0 a y_0 sú posun počiatku snímaču a dopadajúceho lúča. V mojom prípade budú tieto čísla rovné polovici príslušnej velkosti snímača.

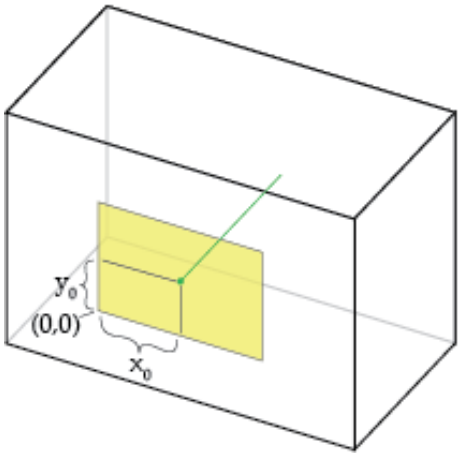
Pokiaľ budeme predpokladať ideálny snímač, skosenie bude $s = 0$ a preto môžeme parameter s v matici zanedbať.

Parametre f_x a f_y predstavujú ohniskovú vzdialensoť pre danú os, tieto parametre sú pre ideálnu kameru rovnaké $f = f_x = f_y$. Preto budeme v matici uvažovať jednu ohniskovú vzdialenosť.

Projekciu budem vykonávať z 3D metrických jednotiek $[m]$ do 2D jednotiek obrazu $[px]$. Na toto môžeme použiť prevodné konštanty alebo celú intristickú maticu K prepočítať do jednotiek pixelov. Ako môžeme z matice vidieť, časť ktorá vyjadruje 2D transláciu už je v jednotkách pixelov, teda prevedieme len časť vyjadrujúcu 2D rotáciu. Na prevod použijeme následovný vzťah,

$$f_{px} = \frac{f_m}{h_{px}} \tag{1.2}$$

kde,
f_{px} - ohnisková vzdialenosť v pixeloch $[px]$,



Obr. 1.8: Model translácie projekcie kamery [9]

f_m - ohnisková vzdialenosť v SI jednotkách $[m]$,
h_{px} - šírka jedného pixelu $[m]$.

Ďalšia transformácia pri projekcii je takzvaná vonkajšia, popisuje ju vonkajšia (*extrinsic*) matica a vyjadruje orientáciu kamery k vonkajšiemu svetu alebo k bodom, ktorých projekciu chceme vykonať. Jedná sa o jednoduchú maticu transformácie, ktorú môžeme popísať ako [9],

$$\mathbf{E} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & w \end{pmatrix} \tag{1.3}$$

kde,
R - matica rotácie, vyjadrená quaternionmi,
t - vektor translácie,
w - mierka.

Maticu celkovej projeckie potom dostaneme ako,

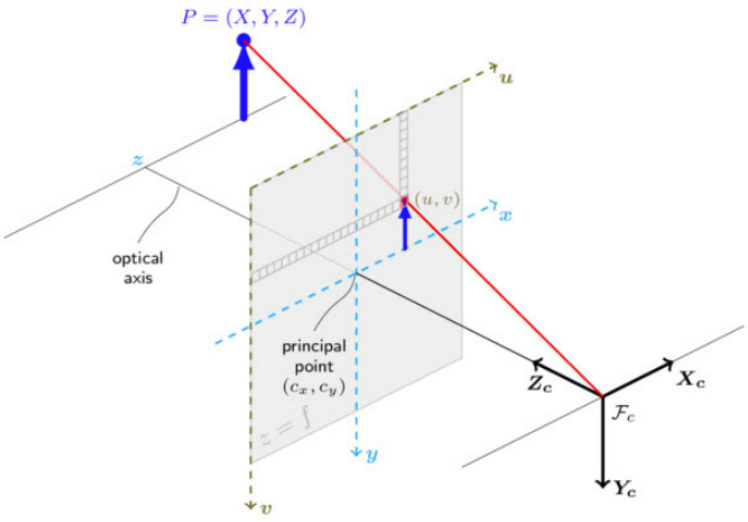
$$\mathbf{P} = \mathbf{K} \times \mathbf{E} \tag{1.4}$$

Je zjavné že matica **P** bude mať rozmer 3x4, preto pre projekciu bodov budeme musieť vektor bodu rozšíriť o jeden skalár. Tento prístup nám rozšíri bod po projekcí o jeden skalár, ktorý bude rovný hodnote Z súradnice daného bodu. Princípy tejto projekcie budú v kapitole 1.4.4.

1.4.1 Projekcia z 2D do 3D

Pri vykonávaní projekcie z 2D plochy do 3D priestoru potrebujeme iterovať cez každý pixel kamery pre ktorý vytvoríme lúč do priestoru, v definovanom okolí tohto

lúču budeme hľadať bod z PointCloudu. Znázornenie tejto metódy je na Obr. 1.9.



Obr. 1.9: Projekcia z 2D plochy do 3D priestoru [4]

Je zjavné, že pre túto prácu táto metóda nie je ideálna, pretože iterujeme cez $n \cdot m$ pixelov, ktorých je násobne viac oproti bodov v PointCloude. Touto metódou by sme taktiež overovali body ktoré pri použití niektorých druhov snímačov nemôžu v 3D existovať.

Túto metódu dokážeme však modifikovať, aby nevyžadovala hladanie bodu v blízkom okolí lúča, ale len zisťovala či sa bod nachádza v priestore ohraničenom uzavretou plochou. Táto modifikácia spočíva vo vytváraní lúča z rohov pixelov, čiže pre každý pixel budú definované 4 lúče, ktoré definujú uzaveretý hexahedron. Pri prepoklade, že dĺžka tohto hexahedronu je nekonečná, alebo aspoň rovná najvzdialenejšiemu bodu, vieme povedať že každý bod, pre ktorý existuje platná projekcia bude uzavretý v jendom z hexahedronov. Potom nám stačí určiť ktoré body sa nachádzajú v ktorom hedrone a vzniká tak projekcia celého PointCloudu. Táto metóda avšak iteruje cez ešte viac pixelov a to konkrétne $(n + 1)(m + 1)$.

Táto metóda projekcie je však využívaná najmä v počítačovej grafike a shaderoch, kedy chceme zistiť aký obraz sa z 3D priestoru prenesie uživateľovi na obrazovku. Alebo pri technológií ray tracing (prípadne path tracing), kde potrebujeme vytvoriť a simulovať lúče svetla.

1.4.2 Projekcia z 3D do 2D

Projekcia z 3D do 2D je v princípe rovnaká ako z 2D do 3D s tým rozdielom, že neiterujeme každý pixel kamery, ale každý bod v priestore. Na tieto body potom aplikujeme zvolenú projekciu a dostávame pixel na senzore kamery.

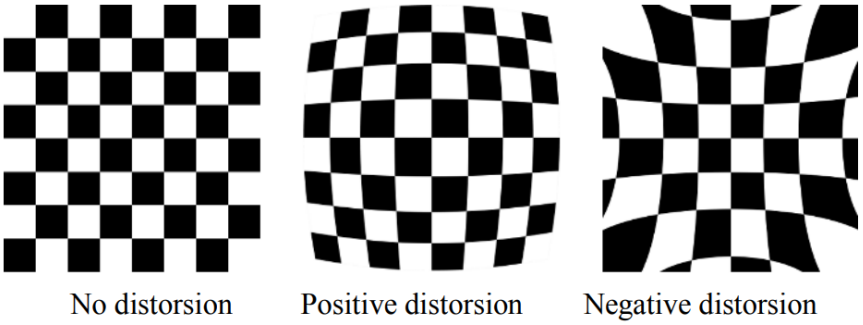
Vzhľadom k tomu, že bodov v priestore je rádovo menej ako pixelov na snímači kamery je tento prístup ideálny na využitie v tejto práci.

Týtmo prístupom môže však dochádzať k "dvojitej"projekcií, kedy sa na snímači kamery objavujú body zo zadnej strany kamery. Kvôli tomuto musíme použiť vhodnú metódu volenia bodov, pre ktoré sa projekcia vykoná.

1.4.3 Kalibrácia kamery

Kalibrácia kamery je dôležitá súčasť aplikácií, ktoré využívajú geometriu kamier. Používa sa na zistenie presných parametrov danej kamery. Tieto paramtere sú teoreticky vypočítateľné, ale nepresnosť výroby spôsobí, že nebudú presné. Kalibrácia dokáže taktiež určiť skreslenie kamery, ktoré môžeme v aplikácií kompenzovať.

Skreslenie obrazu môže byť v podstate ľubovolné, záleží od snímacieho systému kde sa skreslenie objaví. Napr. pri elektrónovej mikroskopií môže byť skreslenie spôsobené prítomnosťou elektromagnetického poľa. Pre optické kamery skreslenie vzniká v geometrií objektívov a tieto skreslenia majú definované tvray. Poznáme kladné a záporné skreslenie, ich tvary sú zobrazené na Obr. 1.10.

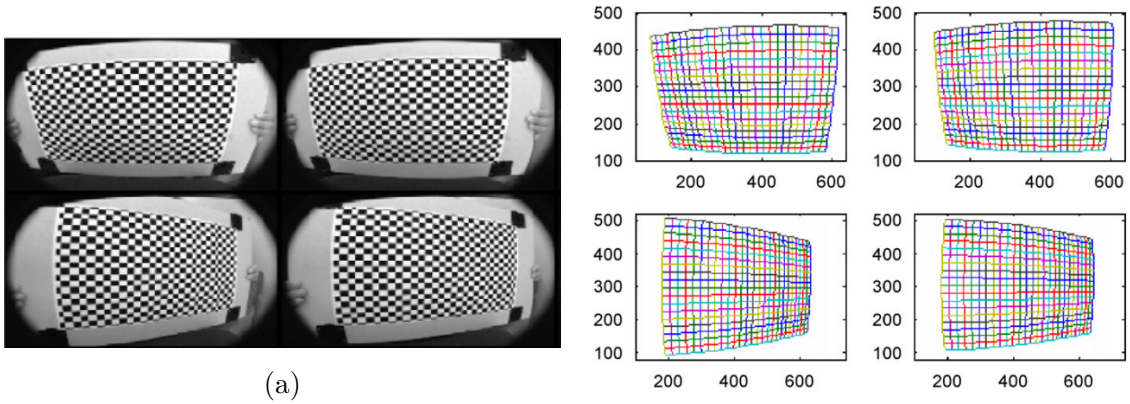


Obr. 1.10: Približné tvary skreslenia optickej kamery [4]

Priebeh kalibrácie:
Kalibrácia prebieha postupným zaznamenávaním kalibračnej štvorcovej mriežky, ktorú môžeme videť na Obr. 1.11a. Z tejto mriežky sa vytvárajú úsečky, ktoré môžeme vidieť na Obr. 1.11b. Následne sa na základe ich zakryvenia a deformácie počítá skreslenie kamery.

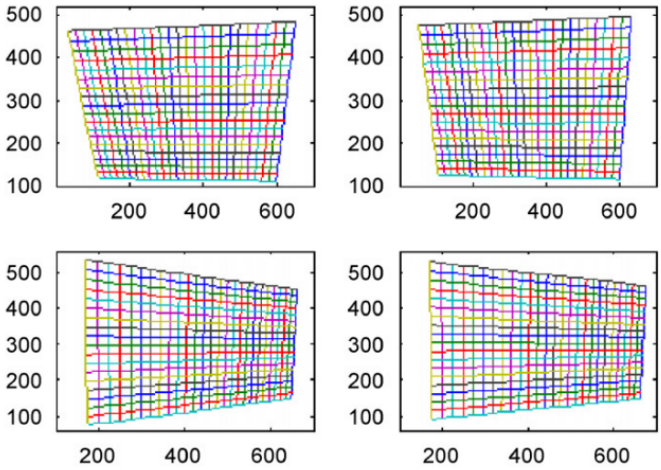
Na správnu kalibráciu, teda takú, ktorá obsahuje aj projekčné matice potrebujeme vedieť rozmer jedného zo štvorcov. Na základe rozmeru dokážeme potom vypočítať intristickú maticu danej kamery.

Po úspešnej kalibrácií a aplikácií korekcie skreslenia by sa mali čiary z Obr. 1.11b vyrovnať a byť na seba kolmé. Pri tejto korekcií môže dôjsť k čiastočnému orezaniu obrazu alebo doplneniu chýbajúcej časti obrazu. Takto správne kalibrovanú mriežku môžeme vidieť na Obr. 1.12.



(a) (b)

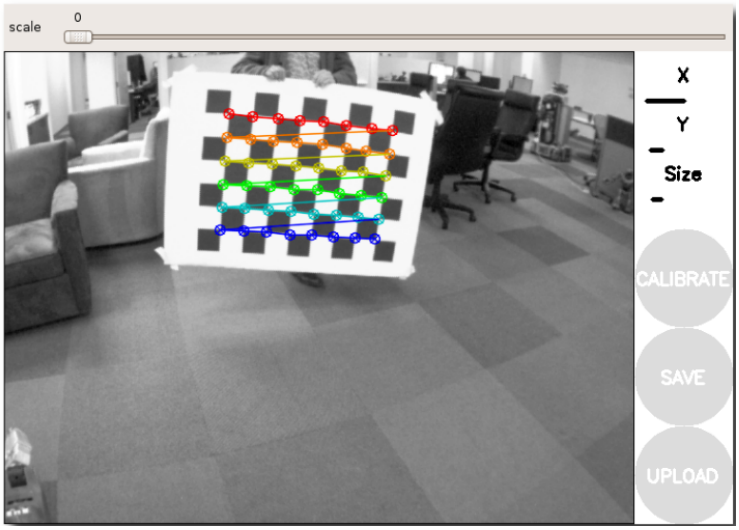
Obr. 1.11: Kalibrácia kamery: a) kalibračná mriežka, b) zobrazenie zaznamenanej mriežky (v ideálnom prípade sú čiary rovnobežné a kolmé) [10]



Obr. 1.12: Zobrazenie kalibračnej mriežky po úspešnej kalibrácií kamery [10]

Kalibrácia kamery v ROSe:

V prostredí ROS nám balík `camera_calibration` umožňuje kalibrovať kamery. Pre jeho správnu funkčnosť je potrebené mať nainšalovaný driver príslušnej kamery a tento balík spustiť so správnimi parametrami. Po spustení príkazu sa otvorí okno prostredia, Obr. 1.13, ktoré nás doprevádza pri kalibrácií. V tomto okne vpravo môžeme vidieť lištu, na ktorej sa postupne zobrazuje priebeh kalibrácie a či má program dostatok údajov pre kalibráciu. Pre získanie údajov potrebujeme kalibračnú mirežku rôzne natáčať, naklánať a približovať. Následne sa rozsieti tlačidlo `CALIBRATE` po ktorého stlačení sa vypočítajú kalibračné údaje príslušnej kamery a tie môžeme následne uložiť do súboru.



Obr. 1.13: Kalibračné prostredie v ROSe [12]

1.4.4 Metódy projekcie

V tejto časti sa budem venovať vybraným metódam projekcie z 3D priestoru na 2D plochu. Tieto metódy projekcie budú v kapitole č.4 implementované a ich výsledky porovnané.

Projekcia delením

Máme zadaný bod v priestore $P = (P_x; P_y; P_z)$. Pokiaľ definujeme počiatok súradnicového systému v strede obrazu potom ľubovolný bod na obraze môžeme definovať ako $M = (M_u; M_v)$, kde $M \in \mathbb{R}^2$.

Pri projekcií bodu P do priestoru M musíme dodržať uhol lúčov, ktoré kamera zachytáva k osi kamery a uhol bodov projekcie. Jedná sa teda o zmenšenie každého z lúčov na definovnú plochu. Pri dodržaní uhlu môžeme ľubovolne meniť ohniskovú vzdialenosť kamery pri zachovaní tvaru projekcie. Pokiaľ nastavíme ohniskovú vzdialenosť na ohniskovú vzdialenosť kamery f dostávame projekciu platnú pre našu kameru. Znázornenie jedno osej varianty tejto projekcie je na Obr. 1.14.
Matemaické vyjadrenie tejto projekcie potom bude,

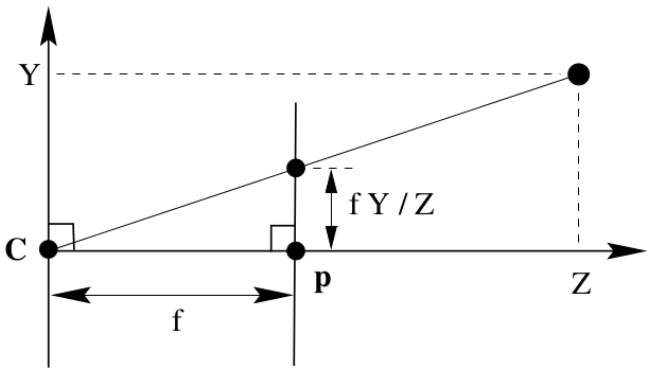
$$tan\left(\frac{P_x}{P_z}\right) = tan\left(\frac{M_u}{f}\right) \tag{1.5}$$

po úprave dostávame,

$$f\frac{P_x}{P_z} = M_u \tag{1.6}$$

pre súradnicu obrazu M_v, potom použijeme vzťah,

$$f\frac{P_y}{P_z} = M_v \tag{1.7}$$



Obr. 1.14: Jedno-osá projekcia delením [2]

Projekcia intristickou maticou

Projekcia spočíva vo využití vypočítanej alebo kalibrovanej intristickej matice, kedy máme zadaný bod v priestor $P = (P_x; P_y; P_z)$, ktorého projekciu vykonávame na plochu $M \in \mathbb{R}^2$. Metóda vychádza z projekcie delením kde sme dostali podiel dvoch súradníc. Pre posun by sme následne iba pripočítali daný offset x_0 a y_0, keďže metóda stále obsahuje delenie súradnicou P_z, môžeme celé riešenie touto súradnicou vynásobiť a dostávame,

$$P_z \cdot M_u = f \cdot P_x + x_0 \cdot P_z \tag{1.8}$$

a,

$$P_z \cdot M_v = f \cdot P_y + y_0 \cdot P_z \tag{1.9}$$

Túto rovnicu dokážeme previesť do maticového zápisu, na to však ako bolo spomentuté vyššie, potrebujeme rozšíriť vektor bodu v priestore o skalár, potom dostávame $P^\star = (P_x; P_y; P_z; 1)$. Projekcia tohto bodu je potom daná ako,

$$P_z \begin{pmatrix} M_v \\ M_u \\ P_z \end{pmatrix} = \mathbf{K}[\mathbf{I}|\mathbf{0}] \times \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \tag{1.10}$$

teda projekcia bude,

$$P_z \begin{pmatrix} M_v \\ M_u \\ P_z \end{pmatrix} = \begin{pmatrix} f & s & x_0 & 0 \\ 0 & f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \tag{1.11}$$

Pre projekciu PointCloudu musíme iterovať pre každy nami vybraný bod. Táto metóda vedie k rovnakému výsledku ako projekcia delením, ale je optimalizovaná,

pretože nevyužíva delenie, ktoré je všeobecne považované za najzložitejšiu matematickú operáciu. Taktiež má táto metóda veľkú výhodu na systémoch, ktoré majú optimalizované alebo priamo integrované maticové násobičky.

Projekcia kalibrovanej kamery

Za projekciu kalibrovanej kamery považujem projekciu projekčnou maticou. Ako už vieme projekčná matica sa skladá z vonkajšej a intrstickej projekčnej matice. K tomuto ešte pridáme kompenzáciu skreslenia. Ako z názvu vyplýva táto metóda vyžaduje kalibrovanú kameru, kvôli zložitosti výpočtov skreslenia. Ak budeme touto metódou vykonávať projekciu celého PointCloudu, môžeme jednotlivé transformácie rozdeliť na transformáciu polohy kamery a následne projekciu.

Transformácia polohy kamery sa dá aplikovať rôznymi spôsobmi, jeden z nich je transformácia celého PointCloudu. Tento prístup má však nevýhodu, že po projekci nám zostáva PoitnCloud transformovaný a na jeho spätnú transformáciu musíme použiť inverznú transformáciu.

Na projekciu priestorových bodov do roviny kamery môžeme taktiež použiť rôzne metódy a v prípade zanedbatelného skreslenia aj projekciu intristickou maticou. Avšak kvôli náročnosti výpočtov skreslenia kamery je výhodné použiť už vytvorené balíky ako napr. v ROSe `image_geometry`.

2 Projekcia 3D bodov na obraz kamery

V tejto kapitole sa venujem implementácií vybraných metód projekcie z kapitoly č.2. Tieto projekcie aplikujem na dáta zaznamenané pri vytváraní datasetu, ich účinnosť a efektivitu porovnám pre výber ideálnej metódy na vytvorenie filtrácie PointCloudov.

2.1 Projekcia delením

Projekciu delením som implememntoval ako program v jazyku C++ ktorý pracuje so samotnými PCD dátami a ako výstup zobrazuje OpenCV maticu (obraz).

Na potreby tejto projekcie si potrebujem definovať pre ktoré body budem projekciu vykonávať. Toto obmedzenie vychádza z parametrov kamery, konkrétne teda zorného poľa, ktoré je pre Intel Realsense D435: $\alpha_h = 69 \pm 1°$ v horizontálnom smere a $\alpha_v = 42 \pm 1°$ vo vertikálnom smere. Pre každý bod budem počítať, či spadá do tohto zorného poľa pomocou porovnania výsledku delenia s tangensom polovičných uhlov zorného poľa.

Ako bolo spomenuté v úvode táto metóda iteruje cez všetky body PointCloudu `cloud`. V mojom prípade je teda táto iterácia zmenšená o body, ktorých projekciu nemá význam počítať. Toto obmedzenie PointCloudu je aplikované v podmienke, ktorá ako už bolo vyššie spomenuté, porovnáva či bod spadá na senzor kamery. Tieto body následne vkladám do samostatného PointCloudu `filtercloud` s fixnou vzdialenosťou osi Z, tento PointCloud je následne normalizovaný na obraz 640x480.

Výpis 2.1: Implementácia projekcie delením

```
1 for (const auto& point: *cloud){
2   if(abs(point.x/point.z) < constx &&
3      abs(point.y/point.z) < consty &&
4      point.z > 0){
5     filtercloud->push_back(pcl::PointXYZ(point.x/point.z,
6                                          point.y/point.z,
7                                          1));
8   }
9 }
```

2.1.1 Výsledky

Ako môžeme vidieť na Obr. 2.1 táto projekcia nie je ideálna. Body lampy (vľavo) a stromov (vpravo) nemajú presnú projekciu na obraz. Toto je však spôsobené posunutím bází LiDARu a kamery, čo sa dá ľahko kompenzovať, transformáciou celého

PointCloudu.

Hlavnou nevýhodou tejto metódy je jej závislosť na dokumentácii ku kamere, ako vieme táto dokumetácia nemusí byť presná a tiež nepočíta so skreslením kamery. Taktiež táto metóda využíva výpočtovo náročné delenie a to pre každý bod PointCloudu, čo pri väčšom množstve dát môže spôsobovať spomalenie algoritmu.

Obr. 2.1: Projekcia bodov PointClodu vykonaná metódou delenia

2.2 Projekcia intristickou maticou

Pre účely tejto projekcie si potrebujem určiť intristickú maticu kamery, túto maticu zatiaľ zistujem výpočtom. Vieme že matica má tvar,

$$\mathbf{K} = \begin{pmatrix} f & s & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

v tejto matici vieme definovať posuny stredu snímača ako polovicu efektívneho rozlíšenia, zkosenie môžeme v našom prípade zanedbať a ohnisková vzdialenosť je daná dokumentáciou. Ohnisková vzdialenosť je v dokumentáci kamery $f = 1,88\ mm$. Aby však bola projekcia vykonávaná priamo do súradnicového systému kamery je potrbené ohniskovú vzdialenosť previesť do jednotiek pixelov, toto vykonám pomocou vzťahu 1.2. Potom výsledná intristická matica bude,

$$\mathbf{K} = \begin{pmatrix} 1342.86 & 0 & 320 \\ 0 & 1342.86 & 240 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

Projekciu intristickou maticou som vykonal v programe MatLab, pretože má optimalizované operácie s maticami. Projekcia bola tiež vykonaná na rovnakých dátach ako projekcia delením aby mohli byť tieto metódy porovnané.

Výpis 2.2: Projekcia intristickou maticou (MatLab)

```
for i=1:PC.Count-16
    x = PC.Location(round(i/16)+1, mod(i,16)+1,1);
    y = PC.Location(round(i/16)+1, mod(i,16)+1,2);
    z = PC.Location(round(i/16)+1, mod(i,16)+1,3);
    if(x > 0)
        Pos = [K [0 0 0]'] * [-y; z; x; 1];
        u = Pos(2)/Pos(3);
        v = Pos(1)/Pos(3);
    end
end
```

2.2.1 Výsledky

Ako možeme vidieť projekcia na Obr. 2.2a nie je správa, jedná sa konkréten o projekciu snímača s väčšou ohniskovou vzdialenosťou, toto je spôsobené tým že na použitej kamere pri zníženom rozlíšení dochádza k 2x2 spájaniu pixelov (*binning*), čo nám spôsobí zväčšenie efektívej velkosti pixelu a neplatí výpočet ohniskovej vzdialenosti v pixeloch. Po úprave intristickej matice na $f_{new} = f/2$ je už projekcia správna. Ako môžeme ale vidieť na Obr. 2.2b stále dochádza k posunu niektorých objektov. Skutočnosť že pri kamere dochádza k spájaniu pixelov bola uvedená až v dokumentácií snímača kamery, kvôli tejto komplikácií nie je tento spôsob vhodný pre všeobecné použitie projekcie.

(a) (b)

Obr. 2.2: Projekcia intristickou maticou. a) vypočítaná matica, b) upravená matica

2.3 Projekcia kalibrovanej kamery

Na aplikovanie tejto projekcie je potrebné kalibrovať kameru, túto kalibráciu som vykonal pomocou ROS balíku `camera_calibration`. Po kalibráci boli hodnoty intristickej matice,

$$\mathbf{K} = \begin{pmatrix} 605.39 & 0 & 328.79 \\ 0 & 603.01 & 248.93 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

Ako bolo spomenuté v úvode kalibrácia dokáže taktiež určiť skreslenie kamery, ktorého parametre sú,

$$k = \begin{pmatrix} 0.105 & -0.156 & 0.006 & 0.004 & 0 \end{pmatrix} \tag{2.4}$$

Na túto projekciu budem používať ROS knižnicu `image_geometry`. Celý kód je dostupný na `https://github.com/tomas-janech/PointCloudFilter`. Túto projekciu budem vykonávať s vonkajšou transformáciou aj kopenzáciou skreslenia.

2.3.1 Vonkajšia transformácia

Paramtere vonkajšej transformácie viem určiť pomocou ROSu, ktorý dokáže vypočítať príslušnú transformáciu medzi dvomi bázami. V ROSe predávame funkcií `lookupTransform` názvy jednotlivých báz a funkcia vracia správu typu `Transform`. Následne dokážeme túto správu pomoocu `eigen` transformovať na tranformačnú maticu typu `eigen::Matrix4`.

Výpis 2.3: Zistenie vonkajšej transformácie

```
1  void Projector::getTransform(const std::string &LidarBase){
2    std::string CameraBase = cameraData.header.frame_id;
3
4    try{
5      transform = buff->lookupTransform(CameraBase,
6                  LidarBase,
7                  tf2::TimePointZero).transform;
8    } catch(const tf2::TransformException & ex){
9      RCLCPP_WARN(<error message>);
10     transform = geometry_msgs::msg::Transform();
11   }
12
13   transformMatrix = tf2::transformToEigen(transform).matrix();
14   inverseTransformMatrix = transformMatrix.inverse();
15
16 }
```

2.3.2 Projekcia PointCloudu

Projekciu PointCloudu som implementoval ako funkciu ktorá prijíma ROS2 správu typu `PointCloud2` a vracia vector bodov po projekcií. Tieto body sú kapsulované v štruktúre, ktorá obsahuje bod po projekcí a pôvodný bod.

Pre využitie vonkajšej transformácie musím aplikovať túto zistenú transformáciu na celý PointCloud, následne body tohto PointCloudu budem predávať funkcií `project3dToPixel` z balíku `image_geometry`. Aby nedochádzalo k dvojitej projekcii, budú tieto body obmedzené podmienkou kladnej osi Z a veľkosťou snímača.

Transformácia PointCloudu

Transformácia PointCloudu spočíva v konvertovaní správy na PCL PointCoud za pomoci pomocnej metódy `ros2pcl` a aplikovaní funkcie z knižnice PCL.

Výpis 2.4: Transformácia PointCloudu

```
1 pcl::PointCloud<pcl::PointXYZ> pointCloud = ros2pcl(pc);
2 pcl::PointCloud<pcl::PointXYZ> computeCloud;
3
4 pcl::transformPointCloud(pointCloud,
5                           computeCloud,
6                           transformMatrix);
```

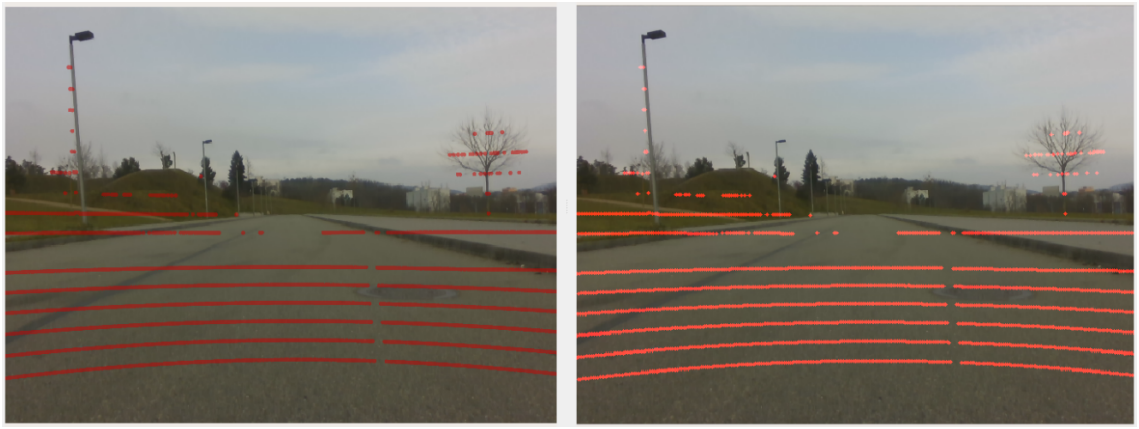
Projekcia bodov

Výpis 2.5: Projekcia bodov na obraz kamery

```
1  for(auto point: computeCloud){
2    if(point.z < 0)
3      continue;
4
5    currentPoint = cameraModel.project3dToPixel({point.x,
6                                                 point.y,
7                                                 point.z});
8    if(<projection condition>)
9      out.emplace_back(currentPoint, point);
10     }
```

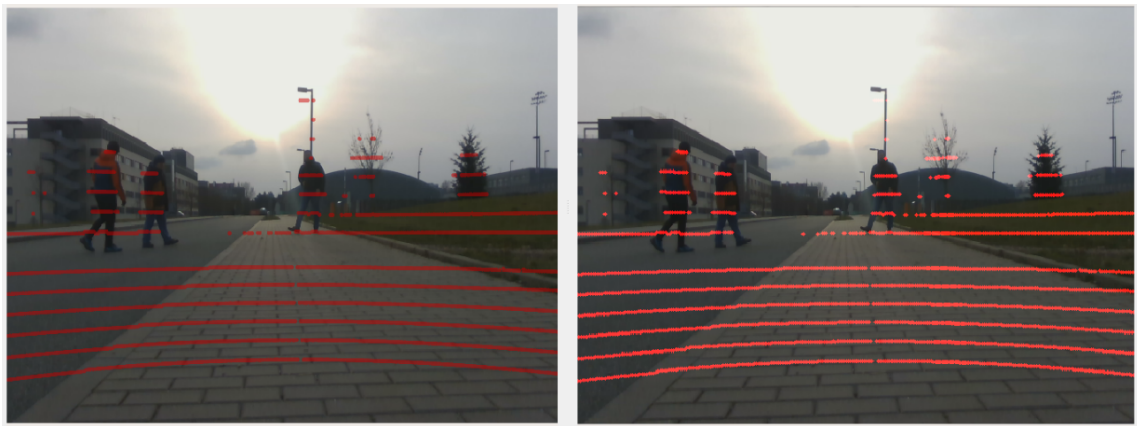
2.3.3 Výsledky

Táto metóda využíva už implementované metódy a kalibračné údaje kamery, preto môžeme povedať, že je najpresnejšia z vybraných metód. Z tohto dôvodu som túto metódu implmentoval ako triedu v jazyku C++ a využijem ju ako základ filtrácie PointCloudu v následujúcej kapitole.



Obr. 2.3: Projekcia kalibrovanej kamery pomocou Rviz (vpravo) a môjho algoritmu (vľavo)

Na Obr. 2.3 môžeme vidieť, že projekcia z programu Rviz a môjho algoritmu je rovnaká (napríklad lampa (vľavo) alebo strom(vpravo)), čo naznačuje správnu implmentáciu algoritmu. Táto skutočnosť sa potvrdzuje aj na Obr. 2.4, kde vidíme, že body, na ktorých sú ludia dopadajú do obrazu presne.



Obr. 2.4: Projekcia kalibrovanej kamery pomocou Rviz (vpravo) a môjho algoritmu (vľavo)

3 Filtrácia PointCloud-u

V tejto kapitole sa venujem implmentácií ROS balíku na filtráciu PointCloudu, najskôr na základne hodnoty daných bodov a neskôr aj na základe segmentovaného obrazu z kamery. Zdrojový kód segmentovanej filtrácie a balík je dostupný na `https://github.com/tomas-janech/PointCloudFilter`. Funkčnosť tohto balíku bude overená na skúšobnom datasete v nasledujúcej kapitole.

3.1 Filtrácia použitím PCL knižnice

Filter navrhnem ako spustiteľný ROS package, ktorý bude v reálnom čase filtrovať hodnoty bodov PointCloudu na úrovni zeme. Na začiatku vytvoríme ROS package, ktorý bude obsahovať jeden zdrojový kód a jeden uzol.

3.1.1 ROS uzol filtrácie

Tento uzol bude komunikovať prostredníctvom topicov, dáta bude prijímať subscriber z LiDAR snímaču a posielať sa budú dáta po filtrácií a dáta vymazané filtráciou, čiže 2 publisher-i. Trieda *Node*, z ktorej som dedil, obsahuje metódy na vytvorenie komunikácie, tieto metódy volám v rámci konštruktora nášho uzlu. Metóda vytvorenia publisheru má ako parameter názov výstupného topicu, ktorý som zvolil `data_out` pre odfiltrovaný a `data_removed` pre vymazaný PointCloud. Metóda vytvorenia subscriberu má parameter názov topicu, z ktorého berie dáta. Taktiež jedným z parametrov je callback funkcia, ktorá sa vykoná vždy keď subscriber príme celé dáta. Ako callback funkciu nastavím moju funkciu na filtráciu dát `filterData()`.

Funkcia filtrácie dát

Funkciu filtrácie vytvorím ako metódu triedy môjho uzlu filtrácie. Táto funkcia odstráni všetky body, ktoré sa nachádzajú na úrovni "zeme", resp. roviny po ktorej sa robot pohybuje. Ako bolo spomenuté v úvode, prijaté dáta, ktoré sú typu PointCloud správy potrebujem konvertovať na PCL PointCloud a následne zvoliť vhodný filter na odstránenie bodov. Tento odfiltrovaný Point Cloud potrebuje následne konvertovať naspäť na PointCloud správu a túto správu poslať do ROS-u.

Konverzia dát

Aby som mohol s dátami pracovať pomocou knižnice PCL, potrebujem zmeniť ich typ na PCL PointCloud. Knižnica PCL už obsahuje túto možnosť konverzie dát. Momentálne potrebujem konvertovať typ správy PointCloud na typ PCL Point Cloud.

Tohto docielim najskôr konverziou na pomocný dátový typ PCLPointCloud2 pomocou funkcie `toPCL()` z knižnice `pcl_conversions`. Tento dátový typ následne konvertujem už na PCL PointCloud pomocou funkcie `fromPCLPointCloud2`.

Výpis 3.1: Konverzia dátového typu zo správy na PCL

```
1  pcl::PCLPointCloud2 pcl_pc2;
2  pcl_conversions::toPCL(message, pcl_pc2);
3  pcl::PointCloud<pcl::PointXYZ>::Ptr
4    InputPointCloud(new pcl::PointCloud<pcl::PointXYZ>);
5  pcl::fromPCLPointCloud2(pcl_pc2, *InputPointCloud);
```

Na záver funkcie filtrácie potrebujem dáta konvertovať naspäť na typ Point Clout správy, aby som ich mohol poslať. Táto konverzia funguje podobne ako tá hore spomenutá, použitím dočasnej premennej typu `PCLPointCloud2` a následnej konverzie do ROS správy.

Výpis 3.2: Konverzia dátového typu z PCL na správu vo funkcí

```
1 void pclPC_to_PCmsg(
2     const pcl::PointCloud<pcl::PointXYZ> &pointCloud,
3     sensor_msgs::msg::PointCloud2 &out_message) const {
4   pcl::PCLPointCloud2 PC_to_PCmsg;
5   pcl::toPCLPointCloud2(pointCloud, PC_to_PCmsg);
6   pcl_conversions::fromPCL(PC_to_PCmsg, out_message);
7   return;
8 };
```

Filtrácia Z súradnice

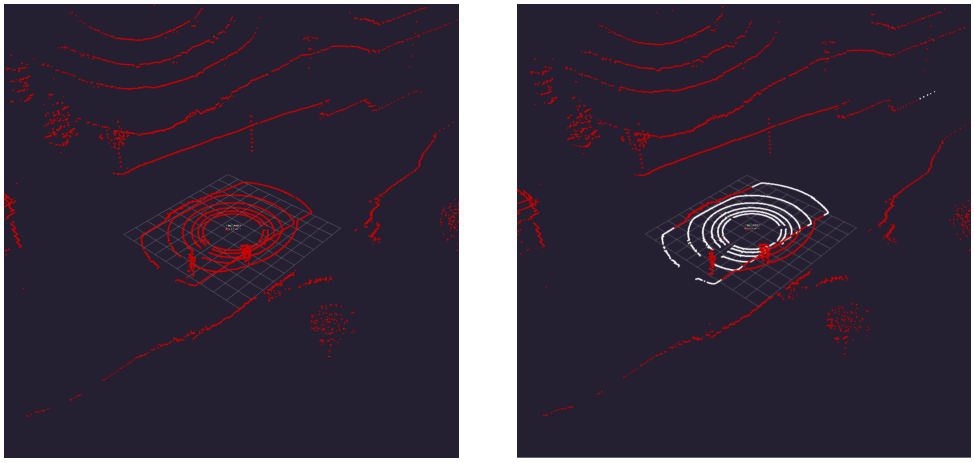
Na filtrovanie Z súradnice použijem filter pásmovej prepusti z knižnice PCL, ktorého limity nastavím na Z-súradnicu <-0.55, -0.46>. Tieto hodnoty boli vybrané ako poloha Velodyne snímača s toleranciou ±5 cm. Filter nastavím na pásmovú prepusť, odfiltrujem dáta a uložím ich do nového Point Cloudu. Následne zmením typ filtra na pásmovú zádrž a dáta filtrujem opätovne. Táto dvojitá filtrácia je použitá kvôli vizualizácií a vo výpise z tohto dôvodu nie je.

Výpis 3.3: PCL filter pásmovej prepusti

```
1 pcl::PassThrough<pcl::PointXYZ> Zfilter;
2 Zfilter.setInputCloud(InputPointCloud);
3 Zfilter.setFilterFieldName("z");
4 Zfilter.setFilterLimits(-0.55,-0.46);
5 Zfilter.setNegative(true);
6 Zfilter.filter(*filtered_cloud);
```

3.1.2 Výsledky filtrácie

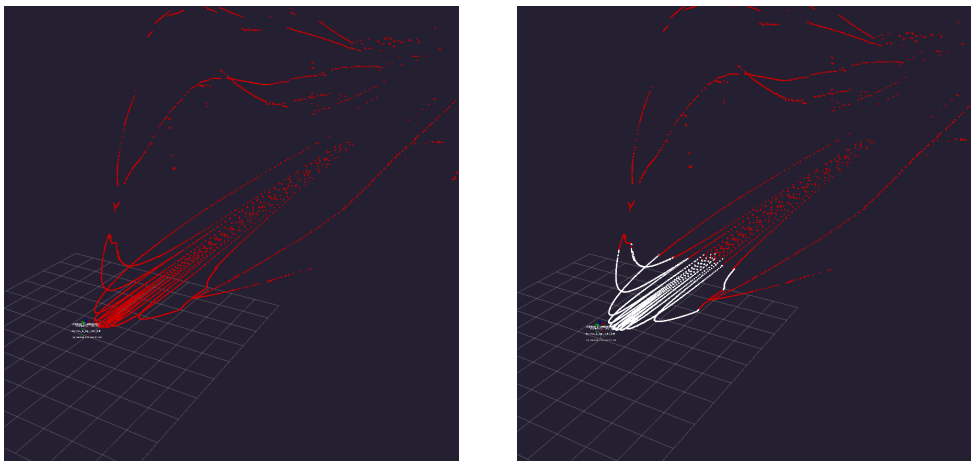
Filtrácia dát zo snímača Velodyne Puck. Na Obr. 3.1 môžeme vidieť, že body po ktorých sa robot pohybuje (cesta) sa úspešne odfiltrovali v celom Point Cloude.



(a) (b)

Obr. 3.1: Filtrácia dát snímača Velodyne Puck, (a) dáta zo snímača (pred filtráciou), (b) dáta(červené) po filtrácií nulových bodov, odfiltrované body (biele)

Filtrácia zo snímača Livox Mid-70. Na dátach, na Obr.3.2, môžeme vidieť že sa úspešne odfiltrovali iba dáta v okolí robota, toto mohlo byť spôsobené nerovnosťou povrchu alebo miernym naklonením snímača.



(a) (b)

Obr. 3.2: Filtrácia dát snímača Livox Mid-70, (a) dáta zo snímača (pred filtráciou), (b) dáta(červené) po filtrácií nulových bodov, odfiltrované body (biele)

3.2 Filtrácia pomocou obrazovej segmentácie

Od tohto filtra sa očakáva, že bude schopný prijať segmentované obrazové dáta, následne vykonať projekciu a na záver tieto dáta rozdeliť do jednotlivých tried. Ako požiadavka bolo tiež stanovené vytvoriť tento filter ako funkčný balík ROS s možnosťou jednoduchej úpravy parametrov a spúšťacieho súboru.

3.2.1 Implementácia filtra

ROS package

Ešte pred začiatkom programovania tohto filtru je potrebné vytvoriť ROS balík, tento balík som nazval `PointCloudFilter`. Po vytvorení prvého zdrojového súboru som vytvoril ROS Node (uzol), ktorému som priradil Subscriber-i a Publisher-i podľa potrieb tohto balíku. Publisher-i teda boli:

- Prijímanie segmentovaných obrazových dát. Táto správa bola typu Image.
- Prijímanie vstupného PointCloudu, typu PointCloud2.
- Prijímanie kamerových dát, predovšetkým kalibrácie. Správa je typu CameraInfo

Ako výstup tohto filtra by stačil iba filtrovaný PointCloud, ale rozhodol som sa dať užívateľovi možnosť posielať na výstup aj dáta, ktoré boli odfiltrované tiež vo forme PointCloud-u. Táto možnosť je definovaná parametrom, ktoré budú vysvetlené neskôr.

Projekcia

Ako už bolo spomentuté v kapitole "Projekcia 3D bodov na obraz kamery", rozhodol som sa použiť projekciu kalibrovanej kamery, pretože využíva ROS balík `image_geometry` a pracuje s najpresnejším modelom konkrétnej kamery. V tejto kapitole bola prebraná aj implmentácia algoritmu filtrácie, ktorý vo filtri využívam.

Výpis 3.4: Projekcia bodov vo filtrácií

```
1  projection.getTransform(msg.header.frame_id);
2  auto points = projection.pc2img(msg);
```

PointCloud segmentácia

Po úspešnej projekcii určím do ktorej triedy patrí daný bod PointCloudu na základe hodnoty daného pixelu na segmentovanom obraze. Následne z filtračnej tabuľky určím či tento bod patrí do triedy bodov, ktoré chceme ponechať alebo ktoré chceme vymazať.

Výpis 3.5: Filtrácia PointCloudu na základe segmentovaných dát

```
1  for(auto point: points){
2    int pointKey = segImage.at<uint8_t>(point.mapped);
3    if(std::find(segClasses.begin(),
4                 segClasses.end(),
5                 pointKey) != segClasses.end())
6      outPc.emplace_back(point.original);
7    else if(<parameter out_removed>)
8      removedOutPc.emplace_back(point.original);}
```

Po segmentácií bodov jednotlivé PointCloudy konvertuje na dátový typ správy PointCloud2 a následne publikujem cez príslušný Publisher.

3.2.2 Optimalizácia algorimtu

Pre tento filtračný algoritmus je možné navrhnúť optimalizáciu najmä prostredníctvom optimalizácie projekcie bodov na obraz kamery. V tomto prípade je možné PointCloud ešte pred projekciou orezať filtrom, ktorého pásmo je tvaru nakloneného ihlanu s vrcholom v počiatku kamery.

Táto optimalizácia by však bola efektívna iba pri LiDAR snímačoch s veľkým zorným poľom, ako napríklad používaný Velodyne. Pre LiDAR LIVOX by táto optimalizácia nebola, efektívna pretože väčšina bodov, ktoré sníma spadá do zorného poľa kamery.

3.2.3 Filtrácia dát snímačov Velodyne a Livox

V tomto balíku využívam parametre a preto je jednoduché zmeniť, ktoré dáta sa budú filtrovať. Jedná sa o zmenu parametru `point_cloud_topic` v konfiguračnom súbore. Taktiež sa dá jednoducho zmeniť zdroj segmentácie (`segmentation_topic`) a jednotlivé triedy segmentácie (`segmentation_classes`).

V nasledujúcej kapitole bude porovnaná efektivita tohto algoritmu pre rôzne vonkajšie prostredia.

3.2.4 Parametre balíku

Ako bolo spomenuté v teórii, ROS používa parameter ako jednorázové nastavenia algoritmov. Tento balík využíva niekoľko parametrov, medzi najpodstatnejšie patria:

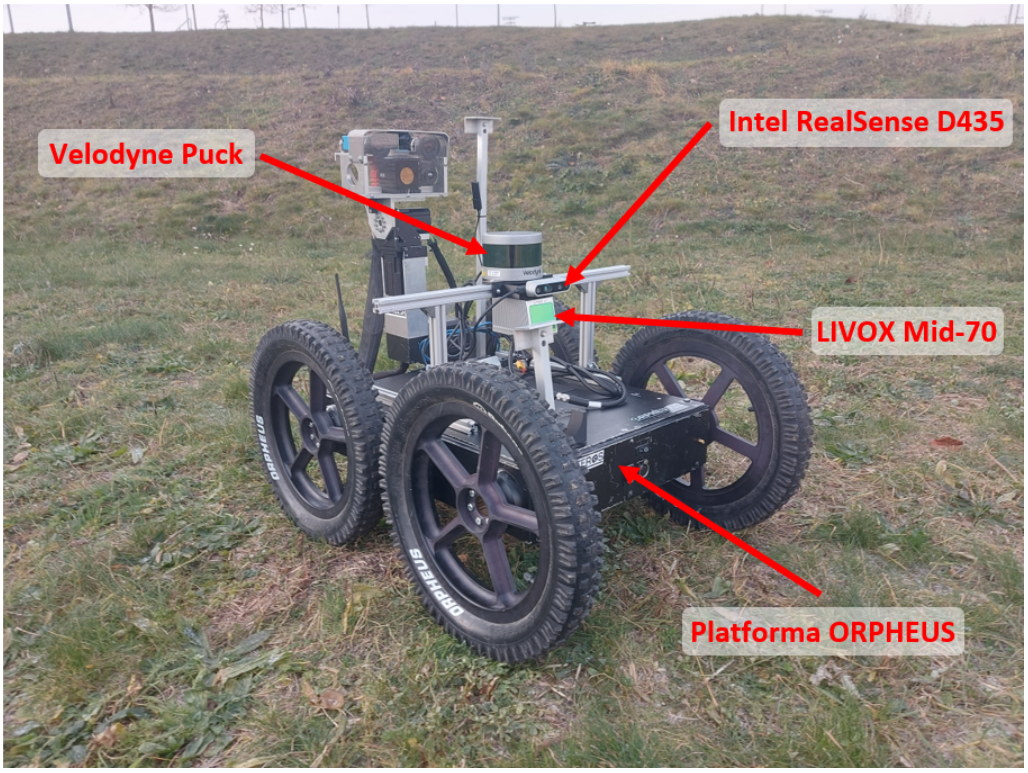
- `point_cloud_topic` - nastavenie vstupného PointCloudu.
- `camera_topic`- topic kamery, ktorý obsahuje kalibračné dáta.
- `segmentation_classes`- triedy segmentácie.

Popis všetkých parametrov a ich dátových typov sa nachádza na GitHube balíku.

4 Testovanie filtrácie

4.1 Zber dát

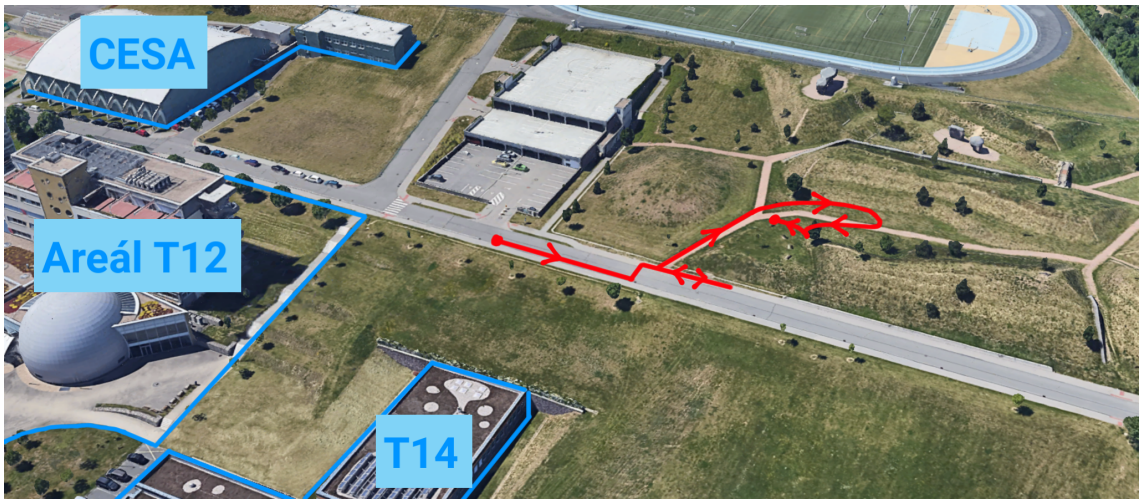
Na zber dát sme použili robotickú platformu Orpheus, ktorá je vyvíjaná na ústave automatizácie. Táto platforma je ovládaná ručne pomocou bezdrôtového ovládača na Xbox. Platforma je vybavená 2 LiDAR snímačmi, snímačom *Livox Mid-70* a snímačom *Velodyne Puck*. Pre účely obrazovej segmentácie je vybavený hĺbkovou kamerou *Intel RealSense D435*, ktorú využívame iba ako RGB kameru. Robot vybavený týmyto snímačmi je na Obr. 4.1



Obr. 4.1: Robotická platforma orpheus vybavená LIDAR snímačmi a kamerou

Dáta boli vytvorené v areále kampusu VUT Brno Sever v Bouldering parku. Táto trasa bola zvolená kvôli vysokému počtu rôznych povrchov (asfalt, betón, štrk, tráva) a členitosti terénu. Dáta zo snímačov boli nahrané pomocou príkazu `ros bag`. Trasa mala dĺžku konania približne 3 minúty, za túto dobu sa zaznamenalo 9.9 GB dát. Presná trasa naznačená v aplikácií Google Earth sa nachádza na Obr. 4.2. Tieto dáta boli zaznamenané v decembri roku 2022.

Na koľko sa táto práca zameriava na spracovanie Point Cloud dát, hĺbkové dáta z kamery budeme zanedbávať. Na dátach z LIDAR snímačov budem testovať navrhnutý algoritmus filtrácie PointCloudu.



Obr. 4.2: Trasa robota (červená) pri vytváraní datasetu

4.2 Výsledky

Výsledky projekcie ktorá bola súčastou tejto práce sú v kapitole "Projekcia 3D bodov na obraz kamery"pri popisoch jednotlivých metód.
Na skúšobných dátach som nastavil filter aby vymazával naśeldovné triedy:

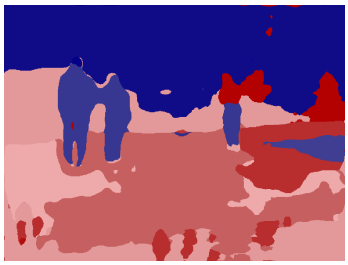
- asfalt (asphalt)
- štrk (gravel)
- tráva (grass)
- hlina (dirt)
- strom (tree)
- budova (building)
- betón (concrete)

Filtrácia sa na celom testovanom datasete vykonávala v reálnom čase, ale presná časová a pamäťová náročnosť algoritmu nebola testovaná.

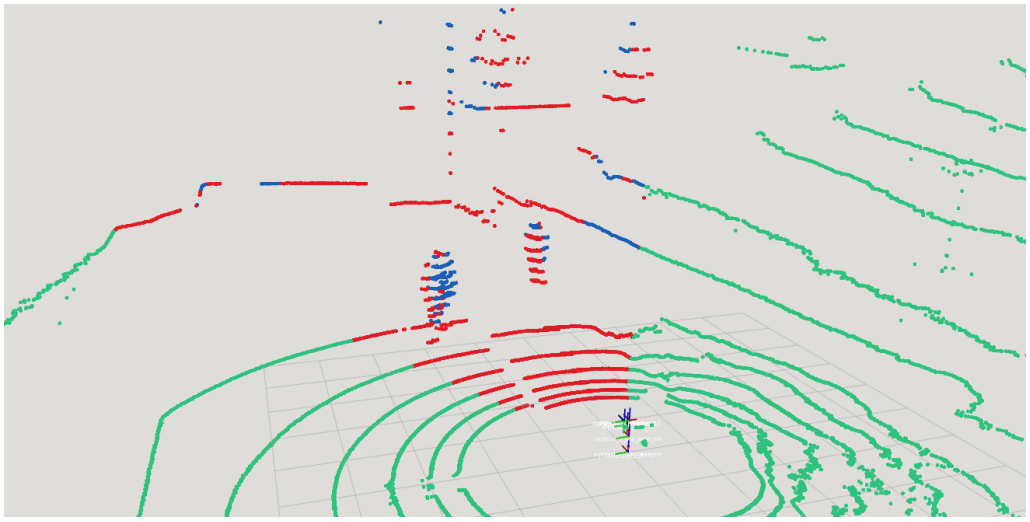
V prvom príklade, kde sa v snímacej oblasti pohybujú osoby môžeme vidieť že segmentácia z obrazových dát 4.3a správe klasifikovala osoby, aj keď na 4.3b môžeme vidieť nepresnoti, hlavne na osobe vpravo, kedy bola vrchá časť tela nesprávne kalsifikovaná. Toto viedlo k nesprávnej klasifikácií na filtrovanej segmentácii, ako môžeme vidieť na Obr. 4.3c. Výsledná filtrácia je zobrazená na Obr. 4.4d a môžeme vidieť že v okolí robota bol povrch úspešne odfiltrovaný.

(a) Obraz kamery

(b) Segmetácia obrazu



(c) Filtrácia segmentácie

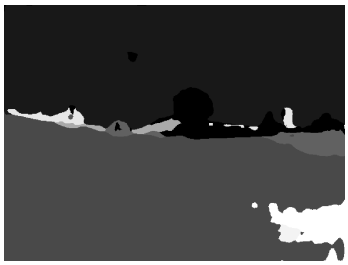


(d) Filtorvaný PointCloud

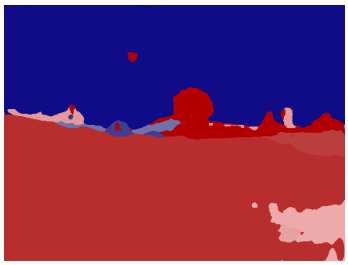
Obr. 4.3: Prvý príklad filtrácie PointCloudu. a) obrazové dáta z kamery, b) ich segmentácia. c) filtrácia segmentovaného obrazu na základe klasifikačných tried a d) výsledok filtrácie PointCloudu (červená odstránené, modrá ponechané)

V druhom príklade na Obr. 4.4a môžeme vidieť obrazové dáta z kamery, vidíme, že ich segmentácia, Obr. 4.4b, prebehla čiastočne úspešne, kedy zlyhala iba pri rozpoznaní štrkovej cesty. PointCloud sa bude teda filtrovať na základe týchto tried ktoré som označil v 4.4c, môžeme vidieť že povrch v blízkosti robota bude odfiltrovaný celý. Toto si môžeme overiť aj na Obr. 4.4d, kde v PointCloude vidíme opäť úspešne odfiltrované (červené) body v tesnej blízkosti robota.

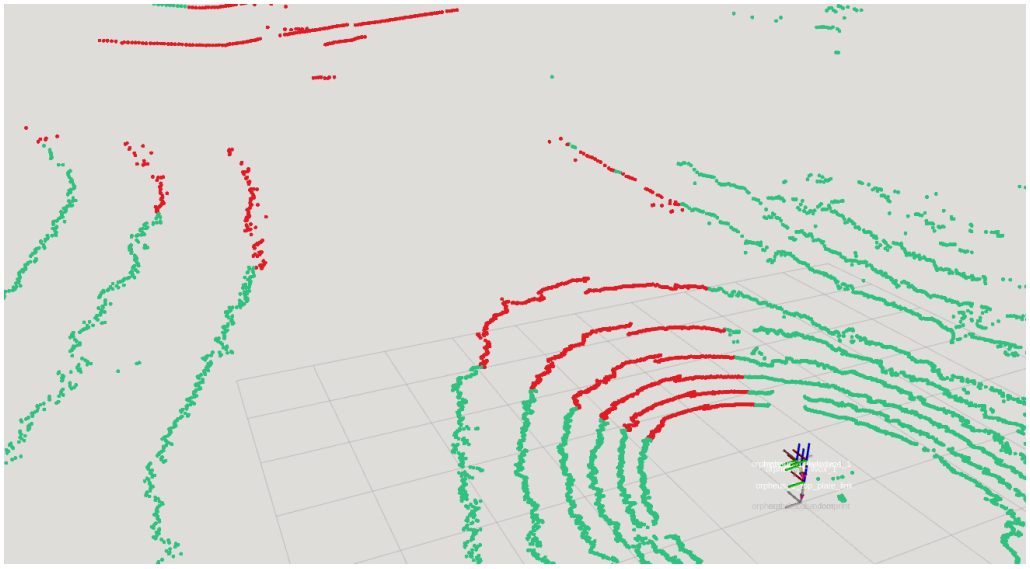
(a) Obraz kamery



(b) Segmetácia obrazu



(c) Filtrácia segmentácie



(d) Filtorvaný PointCloud

Obr. 4.4: Druhý príklad filtrácie PointCloudu. a) obrazové dáta z kamery, b) ich segmentácia. c) filtrácia segmentovaného obrazu na základe klasifikačných tried a d) výsledok filtrácie PointCloudu (červená odstránené, modrá ponechané)

Záver

V práci som sa venoval implementácií filtračných algoritmov PointCoudov, projekcií bodov na obraz a hlavne systému ROS. Na začiatku práce som popísal teoretickú časť systému ROS, rôzne metódy projekcie a manipulácie s PointCloud a obrazovými dátami.

Projekcie 3D bodov na obraz kamery som si popísal a následne aj implemtnoval vo zvolenom jazyku, aby som mohol porovať efektivitu rôznych metód. Metóda projekcie delením sa ukázala ako najjednoduchšia ale aj najmenej presná. Nakoniec som sa rozhodol implementovať metódu projekcie kalibrovanej kamery, pretože má najbližšie reálnej implementácií týchto algoritmov v priemysle, je najpresnejšia a používa model skutočnej kamery.

Na manipuláciu PointCloudov som používal knižnicu PCL a jej nástroje ako filtre, toto sa však ukázalo ako nedostatočné a prešiel som na filtráciu PointCloudu na základe segmetovaných obrazových dát. V tejto práci som teda predpoklaldal že algoritmus segmentácie už je implementovaný, v skutočnosti bol vyvíjaný súčastne s mojím algoritmom, čo spomalovalo môj vývoj filtrácie.

V poslednej kapitole môžeme vidieť príklady použitia tohto filtru, kde vidíme že presnosť a teda aj efektivita navrhnutého algorimtu je závislá od kvality obrazovej segmentácie. V našom prípade nebola segmentácia úplne bezchybná, preto dochádza k malým odchylkám vo filtrovaných dátach.

V práci boli splené všetky body zadania a ich úpravy na základe konzultácií s vedúcim práce. Celkovo túto prácu považujem za prínostnú, pretože mi prehĺbila znalosti v oblasti robotiky, dala praktické skúsenosti pri spracovávaní dát zo snímačov a zlepšila moje programovacie schopnosti.

Literatúra

[1] DUMA, Virgil-Florin a Alexandru SCHITEA. LASER SCANNERS WITH ROTATIONAL RISLEY PRISMS: EXACT SCAN PATTERNS. *PROCEEDINGS OF THE ROMANIAN ACADEMY - Series A*. The Publishing House of the Romanian Academy, 2018, **19**(1), 53-60.

[2] HARTLEY, Richard a Andrew ZISSERMAN. *Multiple View Geometry in Computer Vision*. 2nd Edition. Cambridge, UK: Cambridge university press, 2004, 670 s. ISBN 9780521540513.

[3] LIVOX. Livox Point Cloud and Coordinate System. *Livox Wiki* [online]. 2020 [cit. 2022-12-30]. Dostupné z: `https://livox-wiki-en.readthedocs.io/en/latest/introduction/Point_Cloud_Characteristics_and_Coordinate_System%20.html`

[4] POPESCU, Gabriel, Octavian Laurentiu BALOTA, Daniela IORDAN a Daniel ILLIE. "IN SITU" GEOMETRIC CALIBRATION OF A HYBRID SYSTEM COMPOSED BY UAV/GNSS/IMU/AERIAL CAMERA AND LiDAR. *The 9th International Conference "Agriculture for Life, Life for Agriculture"*. Bucharest, Romania, 2020.

[5] QUIGLEY, Morgan, Brian GERKEY, Ken CONLEY, et al. ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*. 2009, **3**(01).

[6] RAJ, Thinal, Fazida Hanim HASHIM, Aqilah Baseri HUDDIN, Mohd Faisal IBRAHIM a Aini HUSSAIN. A Survey on LiDAR Scanning Mechanisms. *Electronics*. 9. 2020, **9**(5). ISSN 2079-9292. Dostupné z: doi:10.3390/electronics9050741

[7] RUSU, Radu Bogdan a Steve COUSINS. 3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, 2011, 1-4. ISBN 978-1-61284-386-5. Dostupné z: doi:10.1109/ICRA.2011.5980567

[8] SAVARESE, Silvio. *Computational Vision and Geometry Lab: Lecture 2: Camera Models*. Stanford University, 2015. Dostupné také z: `https://cvgl.stanford.edu/teaching/cs231a_winter1415/lecture/lecture2_camera_models_note.pdf`

[9] SIMEK, Kyle. Dissecting the Camera Matrix, Part 1: Extrinsic/Intrinsic Decomposition. *Sightations* [online]. 14.08.2012 [cit. 2023-05-11]. Dostupné z: `https://ksimek.github.io/2012/08/14/decompose/`

[10] WANG, Jianhua, Fanhuai SHI, Jing ZHANG a Yuncai LIU. A new calibration model of camera lens distortion. *Pattern Recognition*. 2008, **41**(2), 607-615. ISSN 00313203. Dostupné z: doi:10.1016/j.patcog.2007.06.012

[11] *ROS 2 documentation: Humble* [online]. Open Robotics, 2022 [cit. 2022-12-30]. Dostupné z: `https://docs.ros.org/en/humble/`

[12] How to Calibrate a Monocular Camera. *ROS org* [online]. [cit. 2023-05-11]. Dostupné z: `http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration`

Zoznam symbolov a skratiek

1D	Jednorozmerné
2D	Dvojrozmerné
3D	Trojrozmerné
AGV	Autonomné pozemné vozidlo – Autonomus groud vehicle
GB	Gigabyte, jednotka objemu dát
LiDAR	Optické meranie vzdialenosti – Light Detection and Ranging
OpenCV	Knižnica na spracovanie obrazových dát – OpenComputerVision
PCD	Dátový typ PointCloud dát
PCL	Knižnica na spracovanie Point Cloudových dát – Point Cloud Library
px	Jednotka obrazového rozmeru
RGB	Trojkanálová farba (červneá-zelená-modrá) – RedGreenBlue
ROS	Operačný systém robotov – Robot Operating system
SI	Medzinárodný systém jednotiek
ToF	Dĺžka doby letu – Time of Flight

Zoznam príloh

A Obsah elektronickej prílohy

Elektornická príloha obsahuje všetky zdrojové súbory implementovaných algoritmov, stromová štruktúra príloh sa nachádza nižšie.

```
/ ..... Koreňový adresár
├── Filter ..... Filter Z súradnice, ROS balík
│   ├── include
│   ├── src
│   │   └── filter.cpp ..... zdrojový súbor filtru, jazyk C++
│   ├── CMakeLists.txt
│   └── package.xml ..... konfiguračný súbor balíku
├── Projekcia-Div ..... Implementácia projekcie delením
│   ├── data ..... testovancie dáta
│   ├── src
│   │   └── projekcia.cpp ..... zdorjový súbor projekcie, jazyk C++
│   └── CMakeLists.txt
├── Projekcia-IM ..... Implementácia projekcie intristickou maticou
│   ├── projekcia.m ..... zdrojový súbor projekcie, Matlab 2021b
│   └── <ostatné> ..... testovancie dáta
└── PointCloudFilter ..... Implementácia filtrácie PointCloudu
    ├── config ..... súbory balíku
    │   └── PC_Filter_params.yaml ..... parametre balíku
    ├── launch ..... spúštacie súbory
    │   └── PC_Filter_launch.py ..... spúštací súbor balíku
    ├── src
    │   ├── PC_Filter.cpp ..... implmentácia PointCloud filtru, jazyk C++
    │   ├── Projector.cpp ..... zdrojový súbor projekcie, jazyk C++
    │   └── Projector.h ..... hlavičkový súbor projekcie, jazyk C++
    ├── CMakeLists.txt
    ├── package.xml ..... konfiguračný súbor balíku
    └── README.md ..... dokumentácia balíku
```

Po dohode s vedúcim elektronická príloha ktorá bude odovzdávaná fyzicky obsahuje taktiež testovancie súbory typu `ros bag`. Tieto súbory nie je možné nahrať do informačného systému VUT z dôvodu ich veľkosti (cca 500 MB) a nie sú nutne potrbné na funkčnosť programu.